A Semantic Parser for Geoquery with an Attention Based Sequence to Sequence Model

Abigail Chua

Abstract

One of the greatest challenges of natural language interpretation is it's ambiguity. Semantic parsing attempts to solve this task by translating language into a formal representation that may be treated as source code. In this paper, we present a encoder-decoder model with attention and that leverages a pointer network architecture to implement a copy mechanism. The model is trained and tested on the Geoquery data set.

1 Introduction

Semantic parsing is an important task that translates natural language to formal representations such as Lambda Calculus or Lambda DCS. These forms disambiguate natural language and can be treated like source code that executes on a knowledge base. In this paper, we present a sequence to sequence model that translates questions to prolog form and obtains 77.5% denotation accuracy on a development Geoquery data set (Zelle and Mooney, 1996). Fundamentally, our work follows in the same vein as (Jia and Liang, 2016). However, our model implements several other enhancements, including global attention, a copy mechanism, pre-trained word embeddings and scheduled sampling.

Collaborators include Jason Lin, Ankur Garg, Nidhi Kadkol and Ryan Westerman.

2 Sequence to Sequence Encoder to Decoder Model

Our architecture consists of two major components: an encoder and a decoder (Jia and Liang, 2016). The encoder includes an input embedding layer and a bi-directional LSTM. The decoder is composed of an output embedding layer, a single direction LSTM, a fully connected and a log softmax layer. Our output vector is then a log probability distribution over the vocabulary of ”translated” words. The input and output embedding layers have size 100 and a dropout probability of 0.2. The encoder and decoder LSTMs have a hidden state of size 200.

To train our model, the input sequence is first passed to an embedding layer and encoded by the bi-directional LSTM. The start of sentence token is fed to the decoder LSTM, which is initialized with the encoder's hidden state. The decoder's prediction is then the input for the following time step. This produces a model with 40% token accuracy and 6% denotation accuracy after ten epochs.

The poor performance is most likely because mistakes early in the sequence are fed back to the decoder and are subsequently propagated and amplified. To train the decoder more effectively, we implemented teacher forcing, which feeds the gold token to the decoder at each time step. This substantially improved the model's performance and results in 20.83% denotation accuracy after ten epochs.

3 Global Attention

There are two major flaws, however, with the standard encoder-decoder architecture. Firstly, the encoder may struggle to capture long distance dependencies between source and target words. Secondly, and perhaps more importantly, it is hard to encode entire sentences of varying length in a hidden vector of fixed size. A natural solution, then, is to maintain encodings for each token in the input sequence and a vector that describes how much we should ”focus” on each word. Here, we implement global attention as described in (Luong et al., 2015).

To build attention, we first modify the input to our decoder LSTM. That is, it is now a concatenation of the word embedding and a context vector, c_t. Note that c_t is initialized to the last hidden state of the encoder. To determine c_t for following steps, the bilinear scoring mechanism, given by $a_t = h_t W h_e$, is used to determine the position in the input sequence most like the current hidden state. Here, h_t is the hidden state of the decoder LSTM, W is a learnable weight and h_e is a matrix of hidden states for each time step in the encoder. The softmax of the attention score, $a = softmax(a_t)$, then gives a probability distribution over the input sequence. To obtain the context vector c_t, we multiply the normalized attention scores with the encoder hidden states, so that $c_t = a h_e$. The current hidden state, h_t is then concatenated with c_t, and passed through two fully connected layers and a softmax to produce a distribution over the output vocabulary.

The attention model is trained similarly to the basic encoder-decoder network that we described above. The main difference is that the context vector, c_t, is now an input to the decoder. Implementation of attention significantly improved model performance and results in 63% denotation accuracy.

4 Copy Mechanism

A limitation of the basic attention model is that it only predicts words from the target vocabulary. This frequently means that named entities like locations, organizations and people's names are incorrectly translated. Naturally, a popular modification to the network, is to allow the model to copy tokens from the input sentence. To do this, we implemented a variation of a pointer network that, at a given time step, determines the probability of generating a word or copying a token from the source sequence.

Our implementation of the copy mechanism is based off of the pointer network described in (See et al., 2017). Whereas the basic attention model assumes that the probability of calculating a word w as $P(w) = P_{vocab}$, the copy mechanism now dictates that $P(w) = p_{gen} p_{vocab} + (1 - p_{gen}) \sum_{w_i = w} a_i j$. Here, we approximate p_{gen} as $p_{gen} = \sigma(W_h c_t + W_s h_t)$, where W_h and W_s are learnable weights and σ is the sigmoid function. In their paper, See et. al. define the generation probability as $p_{gen} = \sigma(W_h c_t + W_s h_t + W_x x_t + b_{ptr})$, where W_h, W_s, W_x and b_{ptr} are learnable weights. However, we found that the model's performance substantially decreased when p_{gen} was a function of the input, x_t, and included a bias term. They have therefore been excluded from our network.

To determine $\sum_{w_i = w} a_{ij}$, we first map each word in the input sentence to it's corresponding index in the output vocabulary. This produces a sparse $n \times V$ matrix called T, where n is the length of the input sequence and V is the size of the output vocabulary. T is then multiplied by the vector of attention scores, so that $\sum_{w_i = w} a_i j = a^T T$. Note that during training, there are no missing words in the output vocabulary, so no "expansion" is required. This allows us to maintain the same training regimen as our base attention model.

During inference, we concatenate $p_{gen} p_{vocab}$ with $(1 - p_{gen})a$. The model's prediction then, is the argmax of the previously mentioned tensor. If the returned index, k, is smaller than the output vocabulary, V, then we generate the word k. Otherwise, we copy the word at the $k - V$ index of the input sequence. The inclusion of the copy mechanism, as depicted in table 1, substantially improves the performance of the model and results in an 6% increase in denotation accuracy.

5 Pre-trained Embeddings and Scheduled Sampling

To continue to improve the performance of the model, we used pre-trained 300 dimensional word embeddings to initialize the input and output layers. The word embeddings were allowed to evolve during training and experiments were performed with both the 300 dimensional GloVe and fastText data sets (Pennington et al., 2014) (Bojanowski et al., 2017). Both experiments only showed a small improvement over the baseline copy model, as shown in table 1. Surprisingly, however, the model initialized with GloVe performed about 1% better than those with fastText.

Finally, we implemented scheduled sampling to smooth the transition between training and inference, which requires the model to use it's own predictions instead of gold labels (Bengio et al., 2015). To do this, we pre-trained the model for one epoch with $\epsilon = 1.0$ and then allowed ϵ to decay exponentially with each epoch i. An example used teacher forcing if a randomly generated float, $x \in [0, 1]$ is less than the threshold ϵ. Otherwise, the input to the decoder was it's own prediction. Experiments were conducted with varying values of ϵ. As shown in table 1, the largest improvement

Model	Den. Acc. (%)
Basic Seq2Seq	20.83%
Seq2Seq with Attention	63.30%
Seq2Seq with Copy	69.16%
Seq2Seq with Copy & GloVe embedding	71.66%
Seq2Seq with Copy & fastText embedding	70.83%
Seq2Seq with Copy, GloVe, $\epsilon = 0.99999$	72.50%
Seq2Seq with Copy, GloVe, $\epsilon = 0.9999$	75.83%
Seq2Seq with Copy, GloVe, $\epsilon = 0.999$	71.66%
Seq2Seq with Copy, GloVe, $\epsilon = 0.99$	77.50%
Seq2Seq with Copy, GloVe, $\epsilon = 0.95$	71.66%

Table 1: Model variations and their respective denotation accuracies.

was when $\epsilon = 0.99$, and produced a denotation accuracy of 77.5%. Interestingly, Larger values of ϵ seem to decay too slowly to tangibly improve model performance.

6 Conclusion

With the inclusion of attention and the copy mechanism, initialization of the embedding layers with GloVe and scheduled sampling, our model now produces reasonable predictions. However, given that we are training and testing on such small data sets, it is highly likely that we are over fitted the model. Additionally, there is considerable room for future work such as the tuning of hyper parameters like the learning rate and sizes of hidden states. Furthermore, implementation of different scoring mechanisms as well as local attention, which has been shown to be more performant than it's global counterpart, may improve the model. Finally, future work should include exploring other variations of the architecture, such as reversing input sentences, modifying the number of stacked LSTMs and inclusion of drop out to prevent over fitting.

References

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *TACL* 5:135–146.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. *CoRR* abs/1606.03622.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *EMNLP*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*.