CS388: Natural Language Processing Project 2

Nidhi Kadkol
University of Texas at Austin UTEID: nk9368
nidhikadkol@gmail.com

Abstract

This project implements semantic parsing with encoder-decoder models to translate a sentence from natural language to a formal representation for answering knowledge-based questions. We implement an encoder-decoder model and then improve the performance using attention and scheduled sampling. Collaborators: Shivangi Mahto, Abigail Chua, Hailey Hultquist.

1 Introduction

In this project we implement a basic encoder-decoder model using LSTMs. We also use Attention [4] to improve the performance of the model, and extend the model by implementing 3 kinds of scheduled sampling [3]. To evaluate our model, we use denotation matches (the answer that the logical form gives when executed against the knowledge base), token-level accuracy (checking if characters match position by position) and exact logical form matches (the predicted sequence exactly matches the gold standard).

2 Basic Model

2.1 Implementation

We pass in a list of input embeddings to an encoder LSTM, which returns a vector representation of the sentence. For the decoder model, we use a single-celled LSTM which takes in one output token embedding at a time and predicts the next token [1], [2]. We use an input and output embedding layer respectively to get the embeddings of input (natural language) and output (formal representation) tokens. After calculating the loss for a training example, we back propagate through the decoder, the output embedding layer, the encoder, and the input embedding layer - thus training the parameters of all these networks.

2.2 Hyper-parameters

We used the Adam optimizer with a learning rate of 0.001 and a dropout of 0.2 for both the input embedding model and the output embedding model. The dimension of the input and output embeddings were 100, and the hidden size for both the encoder and decoder models were 200. The encoder was a bidirectional LSTM with one layer. We used Cross Entropy Loss for optimizing. After running the models with these specifications for 10 epochs, we get around **68%** token accuracy and **10.8%** denotation accuracy.

3 Attention

Attention allows us to look at the vector representations for each word in the input sentence instead of attempting to pack the meaning of the entire sentence into a single vector representation. This allows us to give due importance to particular words of the input sentence at each decoding step, thus giving our performance a great boost.

3.1 Implementation

We use the bilinear attention model [4] and compute it after the LSTM cell using the same hyperparameters as before. We calculate e_{ij} from the decoder hidden state $\bar{h}_i$ and each of the token encoder hidden states h_j.

Since the encoder is bidirectional, $\bar{h_i}$ is of size 200 and h_j is of size 400. So we pass $\bar{h_i}$ through a linear layer whose output size is 400 before we multiply with h_j. We then softmax e_{ij} over j to get the attention weights a_{ij}. We do a matrix multiplication of the a_{ij} values with h_j vectors which gives us a context vector c_i. We concatenate this vector with $\bar{h_i}$ and pass it through a $tanh$ non-linearity, and finally pass this through a fully connected layer with output size equal to the length of the output vocabulary.

3.2 Results

For the basic model we were iterating over the length of the input sentence while training. At inference time we appended the predicted token to a list and stopped predicting tokens for an input once the EOS token was predicted. However at times the EOS token would not get predicted and the decoding would get stuck in an infinite loop. To overcome this, we had to cut off the prediction after it reached a certain length. Deciding this length was tricky, as it affected the final denotation and token accuracy. The variance of the accuracy over different runs was also a challenge when deciding the length. Based on experiments as shown in Table 1 we noticed that in general, the longer we let the maximum length be, the better the accuracy was. Keeping evaluation time and accuracy as factors, we decided to stop predictions if the length was the maximum decoder length limit in Geoquery (65) times 3. Running this model for 22 epochs we achieved **55%** exact logical form matches, **81.5%** token level accuracy, and **61.7%** denotation accuracy.

Epochs	max_len	max_len * 2	max_len * 3	max_len * 4
10	44.2	36.7	41.7	47.5
20	49.2	54.2	51.7	58.3
30	51.7	55.8	60	62.3

Table 1: Denotation matches (%) on the development set at inference time for different maximum length decode outputs (max_len = 65)

4 Scheduled Sampling

During training time, we always give in the gold token as input to the decoder at every time step. At decode time however we give in the previously predicted token from the decoder. If there is a wrong prediction at any time step, the decoder does not handle it, and treats it like a correct input. We want to teach the decoder to recover from its own mistakes at training time, so at iteration i we input to the decoder the gold token with probability ϵ_i and the predicted output from the decoder in the previous iteration with probability $1 - \epsilon_i$. We start with a high value of ϵ_i and decay it as learning progresses, so that initially the decoder learns to predict the correct output given the correct input, and later it learns to deal with its mistakes if the input is wrong. This is scheduled sampling, and we experiment with 3 types as explained by Bengio [3]:

1. Linear decay: $\epsilon_i = max(\epsilon_{i-1}, k - ci)$
2. Exponential decay: $\epsilon_i = k^i$ where $k < 1$
3. Inverse sigmoid decay: $\epsilon_i = k/(k + exp(i/k))$ where $k \geq 1$

k and c are hyperparameters that we have to set.

4.1 Experiments

4.1.1 Linear Decay

We try $c = 1 * 10^{-5}$ and $c = 4 * 10^{-5}$ and run experiments with $k = 1$. Figure 1, Figure 2, and Figure 3 show the change in exact logical form matches, denotation accuracy, and token level accuracy respectively over the epochs. The lines in green represent the accuracies when there is no scheduled sampling, that is the basic model with attention. Blue is when $c = 10^{-5}$, and dark pink is when $c = 4 * 10^{-5}$. The model with $c = 4 * 10^{-5}$ performs better than the one with $c = 10^{-5}$. We notice that the model without linear scheduled sampling performs better than with, and suspect it is because decaying starts too early. We then ran an experiment where we decayed (started scheduled sampling) after 25 epochs, shown in red with $c = 4 * 10^{-5}$. The accuracies tend to vary widely over different runs of the experiment. So it is not possible to say with certainty after which epoch we should start scheduled sampling. Thus, we decide to keep track of the denotation accuracy on the

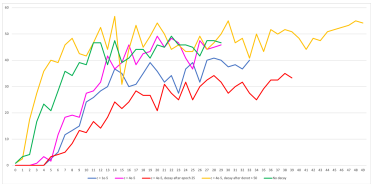

Figure 1: Linear decay - Exact

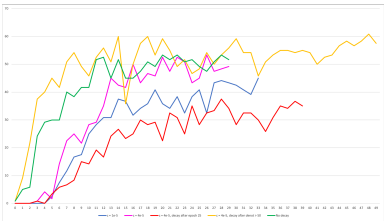

Figure 2: Linear decay - Denotation

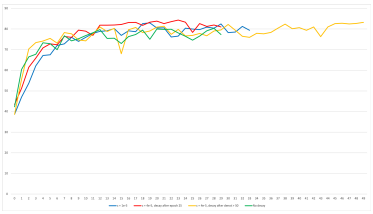

Figure 3: Linear decay - Token-level

development set, and once it hits 50%, we start scheduled sampling where ϵ_i is calculated for the i^{th} iteration after we begin scheduled sampling. The line in yellow shows this graph. For this run denotation accuracy jumped from **41.6%** in epoch 6 to **50.83%** in epoch 7, so we start scheduled sampling from there. We see that this performs much better than all the other variations across all 3 evaluation metrics.

4.1.2 Exponential Decay

We ran similar experiments for exponential decay. The graph lines in black represent the experiment where no decay is implemented. The pink and blue lines represent exponential decay with $k = 0.999$ and $k = 0.9999$ respectively. $k = 0.9999$ shows a clear improvement over the original model. The red line shows the experiment where $k = 0.9999$ but we perform scheduled sampling only after epoch 25. The yellow line is when $k = 0.9999$ and we begin decay after the denotation accuracy reaches 50%. Green line is for $k = 0.999$ and we begin decay after the denotation accuracy reaches 50%. In all the 3 delayed decay graphs, we see a sharp drop in accuracies in the epoch the decay starts. This is possibly because it is the first time the model is experiencing (possibly wrong) predicted outputs as input to the decoder and hence predicting incorrect tokens based on that. We see that the accuracy quickly picks up and performs better than the model with no scheduled sampling.

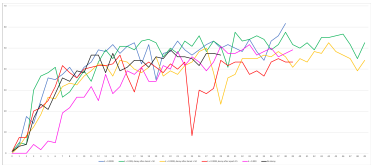

Figure 4: Exponential decay - Exact

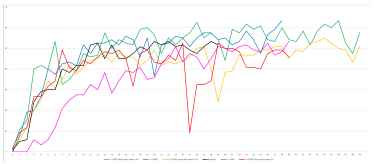

Figure 5: Exponential decay - Denotation

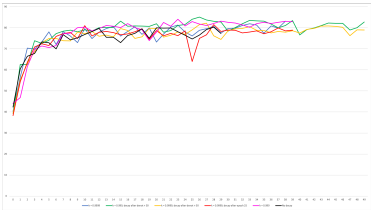

Figure 6: Exponential decay - Token-level

4.1.3 Inverse Sigmoid Decay

We again run similar experiments for this type of sampling. The green line represents no scheduled sampling, so we compare the other lines to this. Blue is when $k = 1500$. Red is when $k = 500$ and we start scheduled sampling only after the denotation accuracy hits 50%. Yellow is when $k = 1500$ and we start scheduled sampling only after the denotation accuracy hits 50%. From the figures, clearly $k = 1500$ is not a well-performing model. $k = 500$ does well but as the number of iterations increases the value of ϵ_i quickly becomes too small to continue decaying with this value of k.

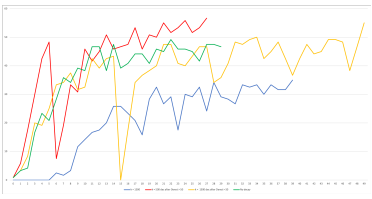

Figure 7: Inverse Sigmoid decay - Exact

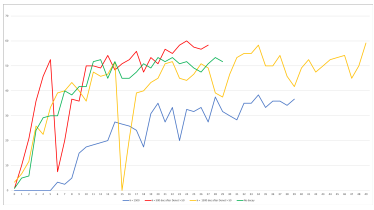

Figure 8: Inverse Sigmoid decay - Denotation

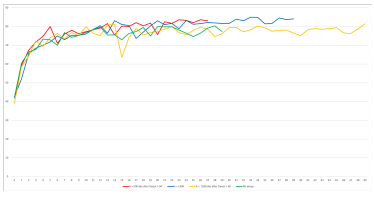

Figure 9: Inverse Sigmoid decay - Token-level

References

[1] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *arXiv preprint arXiv:1409.3215*, 2014.

[2] Robin Jia and Percy Liang. Data Recombination for Neural Semantic Parsing In *ACL*, 2016.

[3] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *arXiv preprint arXiv:1506.03099*, 2015.

[4] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. In *arXiv preprint arXiv:1508.04025*, 2015.