CIS 5300: Assignment 4 [75 Points]

Octavio E. Lima Alexander Dong

1 Introduction (10pt)

We implement a Seq2Seq model for Semantic Parsing using a Recurrent Neural Network (RNN) known as "Long Short-Term Memory" (LSTM). We implement two LSTM models; one with attention and one without. More specifically, we convert natural language sentences (such as "*what states border wisconsin?*") into queries that we can execute against a knowledge base. Our training data consists of 480 pairs of the type `(example, logical form)`, whereas the validation and testing sets contain 120 and 280 pairs each.

For illustration, the example above "*what states border wisconsin?*" would have a `logical form` of: `_answer ( NV , ( _population ( NV , V1 ) , _major ( V0 ) , _city ( V0 ) , _loc ( V0 , NV ) , _const ( V0 , _stateid ( wisconsin ) ) ) ) .`

We compare three different approaches to this problem. **(1)** The first is a "Nearest Neighbors" baseline that uses Jaccard Similarity; when passing in a test string, it returns the query (along with its logical form) from the training data that has the highest token overlap with it. **(2)** The second model is a LSTM model without attention. And **(3)** the third is a LSTM that uses attention scores to better process information from long sentences — this is explained below in greater detail.

2 Model Architecture (30pt)

We implement an Encoder-Decoder approach with 1 recurrent layer, "dot"-type attention scores, and a learning rate of 0.001. Considering the complexity of this task, this set of hyperparameters yields impressive results — substantially better that the Nearest Neighbor's baseline. For instance, the Denotation Matches in the Validation Set are roughly 140% greater in the Attention-Based LSTM compared to the baseline.

In the context of a simple Encoder-Decoder (without attention), we encode the input, then decode the encoder output and previous hidden state to get each word of the output. The downside of this approach is that, because the original "hidden state" is meant to capture information from the entire input sentence, the overall performance of the model tends to decrease as sentence lengths increase.

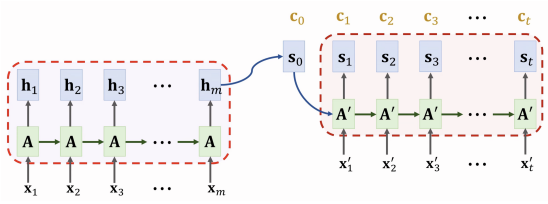

Fig 1: *LSTM Encoder-Decoder Model with Attention*[1].

Through the use of "attention[2]," we enable our decoder to know where to focus on. We first pass the embedded input through the decoder's RNN. We then take this output and pass it through the attention linear layer. We multiply it by the encoding output and take the sum of each encoding to find the sum of each input word's importance; these are the raw scores of each word.

$$RawScore_j = \sum_{i=1}^{len(e)} e_{ji} d_i \tag{1}$$

We take the transformed decoder output (d_i) and pairwise multiply it with the encoding output (e_{ij}). Through a sum along the encoding dimension, we get an array of scores for each word in the input sentence. We apply softmax normalization to find the attention distribution.

$$c_t = \sum_{j=1}^{m} \alpha_{tj} o_j \tag{2}$$

Here, Attention Scores (e_{tj}) are probabilities related to how important each word is. And o_j is the encoder output of each word. These return a weighted sum vector that represents the aggregate knowledge needed for decoding (c_t). We concatenate the output and aggregate vector, then pass them through the output layer.

3 Model Optimization (10pt)

We train our two Attention-based LSTM models using the same hyperparameters.

The hyperparameters that we use in our two LSTM Models (with and without attention) are: **(i)** One recurrent layer, **(ii)** a learning rate of 0.001 **(iii)** 80 epochs —

[1] Wang, Shusen. "Attention for RNN SEQ2SEQ Models." 16 Apr. 2021, `youtube.com/watch?v=B3uws4cLcFw`.

[2] Bahdanau, Dzmitry, et al. "Neural Machine Translation by Jointly Learning to Align and Translate." 19 May 2016, `https://arxiv.org/abs/1409.0473`.

meaning the number of complete passes through the training data, and **(iv)** a batch size of 20 — meaning "the number of training samples to work through before the model's internal parameters are updated,"[3].

We try four different values for the Learning Rate (0.001, 0.010, 0.100, and 1.000). Interestingly, when plotting Token-Level Accuracy by the aforementioned learning rates, we observe a somewhat inverse relationship in performance. In other words, as Learning Rates increase, Token Accuracy decreases for the Testing Set but increases for Validation. That said, we use a learning rate of 0.001 throughout the paper.

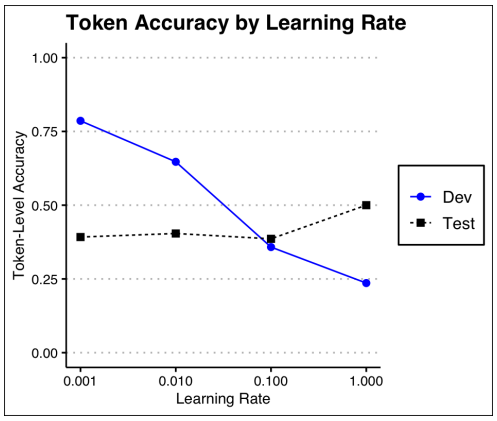

Fig 2: *Token Accuracy by Learning Rate.*

Upon plotting Token-Accuracy by Batch Size, we conclude that our initial value for the Batch Size was the best option when compared to the values of 40, 60, 80, and 100. With a batch size of 20, we reach a T.A. of 0.786 in the Validation Set; and 0.780 in the Testing Set.

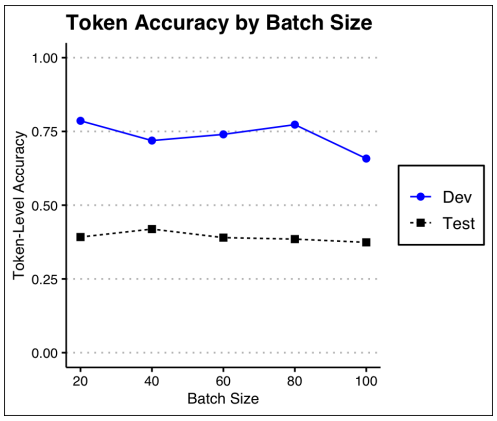

Fig 3: *Token Accuracy by Batch Size.*

4 Output Results (10pt)

As seen in the output table below, our best model is the Attention-Based Long Short-Term Memory Encoder-Decoder. Its performance on Development was significantly better than the other two models across all metrics. What is more, its performance on the Testing Data set was better as well, especially when looking at overall Token Accuracy.

Table 1: Main Results

	Baseline		Simple LSTM		Attention LSTM	
	Dev	Test	Dev	Test	Dev	Test
L.F.M.	0.125	0.007	0.167	0.210	0.408	0.478
T.A.	0.697	0.413	0.739	0.494	0.786	0.780
D.M.	0.200	0.050	0.208	0.225	0.475	0.405

Table 1: *Output Results, where "L.F.M.", "T.A.", and "D.M." respectively mean "Exact Logical-Form Matches," "Token-Level Accuracy," and "Denotation Matches."*

5 Error Analysis (10pt)

Some of the errors from our model may be found below. We observe that three specific types are persistent. These are: **(i)** Wrong States (e.g, "Arizona" rather than "Alabama"), **(ii)** Missing Adjectives, and **(iii)** Unit of Geography. Examples are found on the table below.

Type	Example
Wrong State	y_tok = (NV,_lowest(V0,(_place(V0),_loc(V0,NV)),_const(V0,_**stateid(california)**)))) y_pred = (NV,_lowest(V0,(_place(V0),_loc(V0,NV)),_const(V0,_**stateid(oregon)**)))) y_tok = (NV,(_lake(V0),_loc(V0,NV)),_state(V0),_next_to(V0,NV),_const(V0,_**stateid(texas)**))) y_pred = (NV,(_major(V0),_city(V0),_loc(V0,NV)),_state(V0),_next_to(V0,NV),_const(V0,_**stateid(california)**))))
Missing Adjective	y_tok = (NV,(_**high_point(NV,V1),**_loc(V1,V0),_state(V0),_next_to(V0,NV),_const(V0,_stateid(mississippi)))) y_pred = (NV,(_state(V0),_next_to(V0,NV),_state(V0),_next_to(V0,NV),_const(V0,_stateid(mississippi)))) y_tok = (NV,(_capital(V0),_loc(V0,NV),_state(V0),_next_to(V0,NV),_state(V0),_next_to(V0,NV),_const(V0,_state y_pred = (NV,(_capital(V0),_loc(V0,NV),_**largest(V0,(**_state(V0),_loc(V0,NV),_state(V0),_next_to(V0,NV),_const
Country vs. State vs. City	y_tok = (NV,_highest(V0,(_place(V0),_loc(V0,NV),_const(V0,_**countryid(usa)**)))) y_pred = (NV,_highest(V0,(_place(V0),_loc(V0,NV),_const(V0,_**stateid(florida)**)))) y_tok= (NV,(_river(V0),_traverse(V0,NV),_largest(NV,(_state(V1),_loc(V0,V1),_**city(V0)**,_loc(V0,NV),_const(V0 y_pred = NV,(_river(V0),_loc(V0,NV),_largest(V0,(_state(V0),_loc(V0,NV),_**state(V0),**_loc(V0,NV),_const(V0,_s

Table 2: *Examples of Common Errors.*

6 Conclusion (5pt)

The Attention-Based Encoder-Decoder LSTM does an excellent job — this is especially true if time and memory constraints are not an issue. Nonetheless, such advanced model is **(i)** difficult to implement in light of its complexity, and **(ii)** slow when compared to the Baseline (or compared to itself without attention). We highlight this below. With a training data of merely 480 observations, the Attention-based LSTM took more than twelve minutes to run. To conclude, at a cost, the Attention-Based Encoder-Decoder model yields excellent results when compared to simpler methods. And it is well-suited when the goal is to achieve the best results regardless of time.

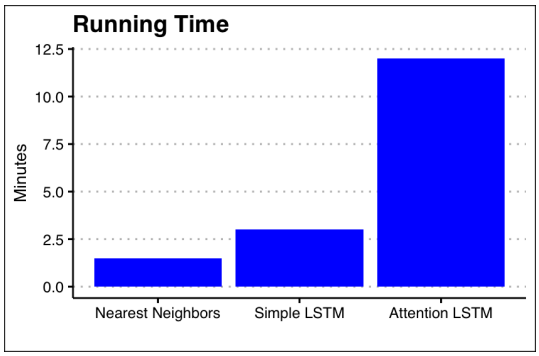

Fig 4: *Running Time in Minutes*

[3]Brownlee, Jason. "Difference between a Batch and an Epoch in a Neural Network." 15 Aug. 2022, `https://machinelearningmastery.com`