NLP HW 2

Parker Sell

4/9/23

Variables	CNN Base Model	LSTM Model	Batch LSTM Model	Batch LSTM3 Model
Training Time (h:min)	~1:30	~1:40	~0:12	~1:15
Epoch Time (min)	3	22	2.5	2.5
Num Epochs	30	5	5	30
Sample/s (1/s)	270	15	5*32 = 160	160
Max Num Param per Layer	25k	~750k	~750k	~750k
Total Num Param	~13.4M	~14.7 M	~14.7M	~14.7M
Total Num Param without Embeddings	55k	1.3M	1.3M	1.3M
Testing Accuracy	.70	.33	.34	.33

Data Preprocessing

To keep the comparisons the same, I followed the exact same data preprocessing steps as shown in the blog with the word2vec model and the paddings. The only thing that changed is that for the Batch LSTM, I loaded the data into a custom PyTorch Dataset object called MyDataset to assist in the batching process. However, the data outputted didn't change per row. This means that I created an even dataset with 21k training size and 9k testing size with even distribution of the three classes in the sentiment analysis rating.

Training Comparisons

I used the tqdm python package to get a better understanding of how long each epoch's iteration took. Without batching, CNN can go through over 270 samples per second one at a time. Since CNN was done in the blog without batching, I tried LSTM without batching as well. This resulted in a very slow iteration time with about 15 samples per second. Additionally, each epoch took about 20 minutes. For this reason, I reduced the number of epochs for LSTM to 5 from 30 for CNN. I also tried a batched version of

LSTM as many example articles utilize batching. I tried a batch size of 32 and this sped up training significantly.

Parameter Comparisons

The CNN is much more efficient in terms of parameters per layer than the LSTM model. The CNN model had 5000 parameters for the smallest filter and 25000 for the largest layer. In comparison, as you can see in the table above, the LSTM model had two layers each with around 500k to 750k parameters. So, it is evident that the LSTM is more computationally expensive. Additionally, there is a column in the table that describes the total number of parameters. However, this number is significantly increased because we are doing backpropagation on the word2vec embeddings. If these are removed, you can see the difference in the parameters is much more different between the CNN and LSTM.

Test Results Comparisons

Since I followed the CNN blog very closely, and only changing paths and minor versioning differences, I received a very similar loss distribution. The testing accuracy was reported to be an average of 70% over the three classes. However, my implementation for LSTM was naïve, and for both implementations always predicted a single value. Typically, I would think that this meant that the dataset was uneven, however, since the subset of the data that I used was so small it must mean that my LSTM is not learning properly. I tried many different architectures by modifying the forward function in the PyTorch model. However, nothing showed improvement. And since the base LSTM model without batching took a long time, I didn't have time to try many things. Additionally, I trained the batch LSTM for the last time with 30 epochs, but the model still didn't improve. If I had more time, I would look at the data more closely, and adjust the architecture. I have had similar issues before, and it usually results in not understanding the architecture well enough. Additionally, it was with time data and that required that we adjust the input of the data to be in a windowed format.

What I learned

I learned that CNN is way more efficient than LSTM, however my hypothesis is that LSTM is also more effective when working properly. I also found that the relearning and finetuning of the embeddings requires a significant number of parameters. I would test if this finetuning increased performance for CNN. I would also look at different forms of embedding to see it was more effective for the different models. I learned a lot more about how to implement the two models in pytorch and especially the importance of batching. I saw a significant speed up in time, and so I would make sure to preprocess the data using the PyTorch Dataset and Dataloader objects for easy batching.