CPSC 393 Assignment #4 Report Nikki Poentis

Abstract

Long Short Term Memory models or LSTMs are neural networks that are capable of working with sequential data. As an improvement of Recurrent Neural Networks (RNNs), LSTMs uses its recurrent nature to reference relevant information at previous time steps when generating output. For this assignment, we were assigned to build 3 LSTMs: a multi-class classification model for BBC articles into its given category, a time series prediction model on a dataset of airlines passengers over a sequence of dates, and finally a text generation model for writing nursery rhymes.

Methods

Multi-class Classification Tutorial:

For the classification model, the data set we were given was a bunch of articles and its category. After removing the stopwords and converting each article into a sequence of tokens, I used these sequences to train a model with Embedding, Dropout, Bidirectional LSTM, and Dense layers.

Time Series Prediction Tutorial:

Similarly to the classification model, I followed along with this tutorial to generate a LSTM model that uses a LSTM layer and Dense layer. Since it predicts a single quantitative variable, this model can be essentially seen as a regression model. The only difference with an actual linear regression model is that the LSTM takes the input data in a specific [samples, time steps, features] format; so before training the model, we had to first configure the data to be structured as so. After building the basic model, the tutorial also goes through 4 different adjustments to hopefully improve the model: using the window method, using time steps, having memory between batches, and using stacked LSTMs with memory between batches.

LSTM Nursery Rhymes:

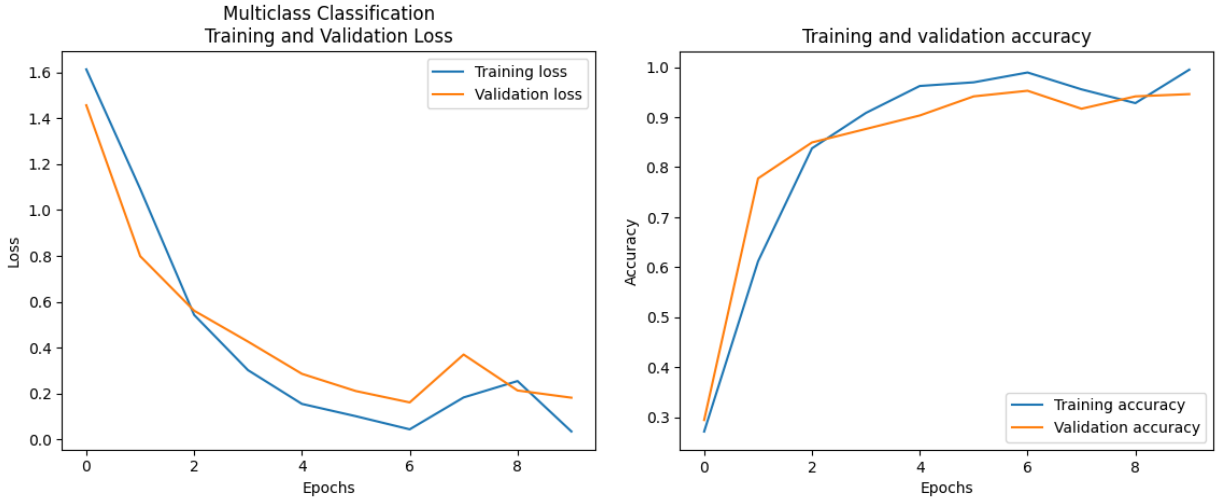
Neural networks typically take a long time to run, so when I started writing my code, I used an example file that held only two nursery rhymes. Since the "dataset" we were given was just a text file with a bunch of nursery rhymes, it was very important to first process the data before I could even think about building the model. I first quickly scrolled through some of the text file so I knew what I was working with. I noticed that the title for each of the nursery rhymes were in all caps. So when I imported in the data, I read the lines into two separate lists, one for titles and one for text, with each nursery rhyme as a single string item in the 'text' list. Afterwards, the rest of my pre-processing included converting each nursery rhyme into n-gram sequences of tokens, padding each sequence so they are all the same length, and generating a vocabulary list for each of the words.

The LSTM model that I trained used four layers: an embedding layer, the LSTM layer, a Dropout layer, and finally the Dense layer with the softmax activation function. In order for the model to actually generate a nursery rhyme–which is essentially predicting the next word in a sequence–I needed to choose a word to start the new rhyme. Instead of choosing a word to start the new rhyme, I wanted the model to be able to write the nursery rhyme all on its own, so I randomly selected a word from the vocab list and made that word be the start of the rhyme. In a nested for loop that iterated through the required 30 lines and generated 20 words for each. For each word, it converts the current rhyme (randomly generates the first word during the first iteration) into a list of tokens, use the model to generate a token for the next word, convert this token into its word state, and concat it into the nursery rhyme state.

Of course since the training data only consisted of two rhymes, the model didn't have much to learn from and it didn't perform well at all–after one or two lines, the rest of the output consisted of the word "spit" over and over again. However, since the first couple of lines were all different words, even though it didn't make any sense, I took it as sign that the model actual worked and took the leap to run all my code again using the full nursery_rhyme.txt file.

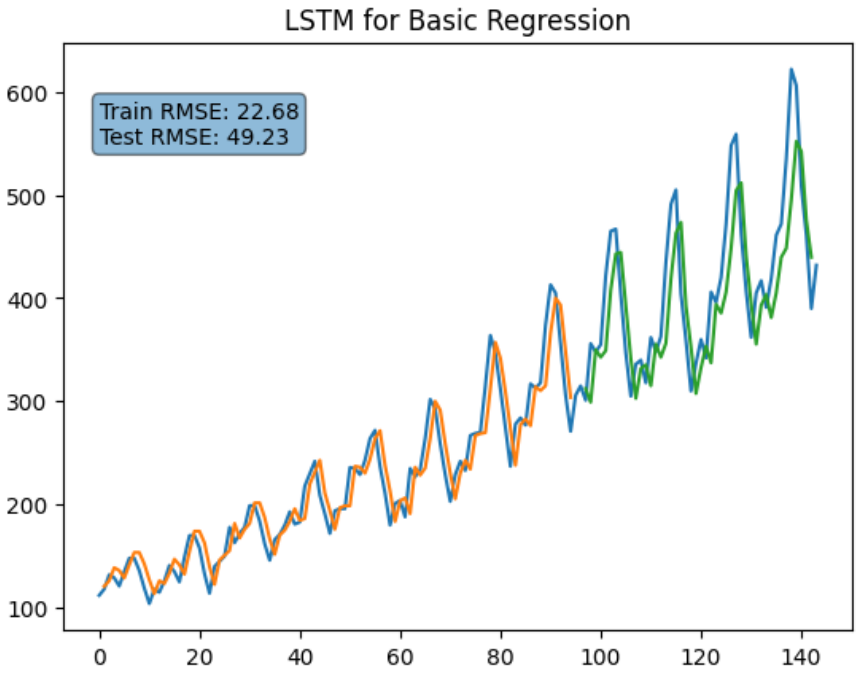
Results & Discussion

Multi-class Classification Tutorial:

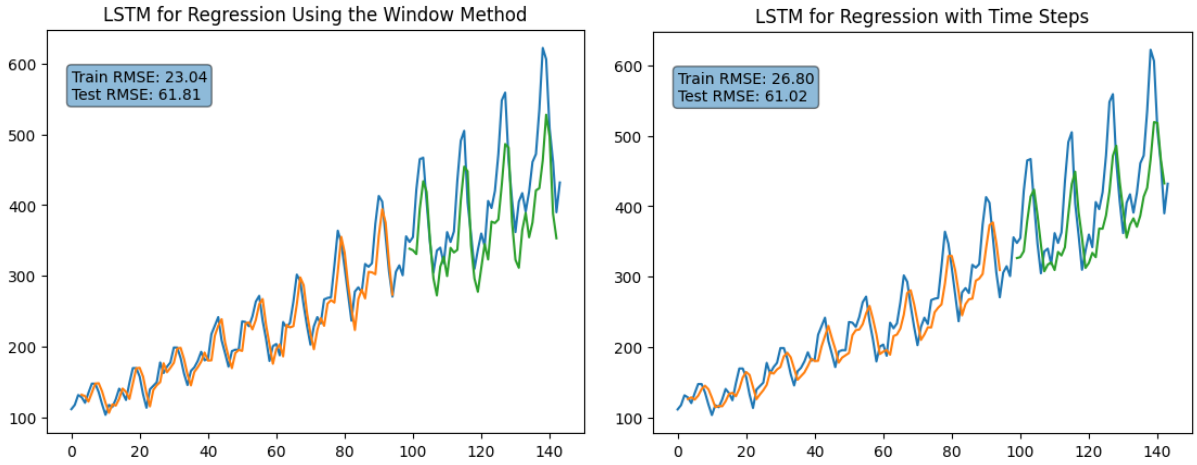


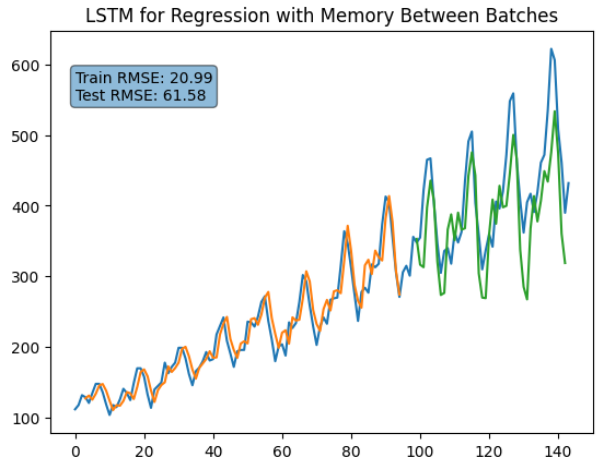
Since the training and validation loss/accuracy curves for the multi-class classification model are almost the same, there curves alone don't show signs of overfitting. Also, the curves show that at 8 epochs loss approaches 0 and accuracy is approaching, which indicates good performance. However, when making predictions, the model was able to accurately classify one line of text but not the second. Of course this LSTM model is still at its most basic level, so we can't expect it to be correct 100% of the time. I think that training the model with 1780 articles is not enough given that each text is so long.

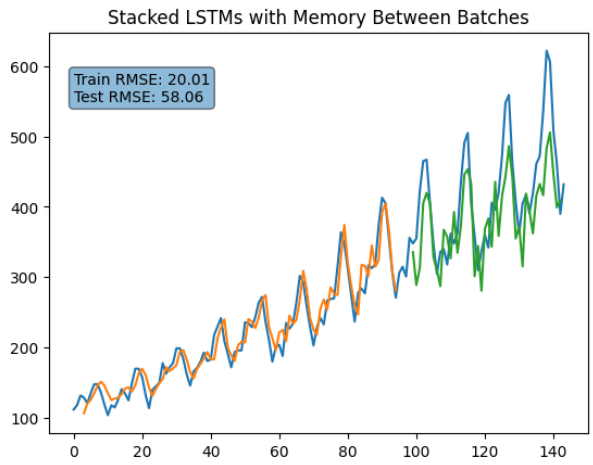
Time Series Prediction Tutorial:



Based on the training and testing RMSE scores for the time series prediction model and the graph above, the model did a pretty good job at predicting the amount of passengers. Relative to the rest of the data, an average error of 23 passengers for training and 49 passengers for testing is low. However, since the testing error is much higher than the training error, its obvious that the model is overfit.







For each further adjustment on the model, the testing error actually increases and the model becomes overfit. Since none of the "improvements" on the model actually improved performance, I think that the overfitting is due to the small amount of data to train the model on; the dataset only represents only 144 time periods in total–96 for training and 48 for testing.

LSTM Nursery Rhymes:

Since there is no metric for measuring the performance of the LSTM in text generation, there is really no way to say for certain whether or not the model did a good job or not. If you read the new nursery rhyme, the model's output, it makes zero sense. However, I don't think that the model did a bad job. Even though its not obvious, the generated words aren't just randomly spit out, if you read closely you can see that each word is somewhat related to the words around it. For example, the second word "I" is followed by the word "was", which is then followed with "a little boy". The model was able to learn and appropriately generate short sequences of words that logically follow each other. Comparing to writing a nursery rhyme based solely on randomly selecting the next word, the model performed okay.

Conclusion

Overall, I like my decision to use the entire rhyme as a sequence–rather than splitting up the rhyme by line like the examples I found online.When I initially wrote the code for my model, using the example txt file with the two nursery rhymes, I ran the code twice using the two types of sequences. Comparing the output from these two models, I thought that using the rhyme as a sequence produced a better output. Even though both versions had the same problem with the word "spit", the model using each line as its own sequence generated approximately three lines before diverging into the "spit" lines. While it technically wrote one more line than the rhyme version, the output had multiple repeated words like "tarts tarts", "drink drink", and "haw hum hum hum hum hum" throughout all three lines. So while the model I ended up with may not have generated a "good" nursery rhyme that made complete sense, I think the way I formatted the input data was a good call.

Of course there are many adjustments to the model that I could have made in order to improve its performance. My next steps if I were to continue with “perfecting” this model would be to include the attention mechanism or use a transformer network instead. I think that this would help the model perform better not only because attention allows for better relationships between the encoder and decoder, but also because it can handle longer, more complex sequences. Since the output sequence is so long, you can tell that the quality of the generated text gets worse over time.