Homework 4

Kelsey Hawkins

Introduction

Books have been around from as long as history is recorded, as it was created to record history and generations of stories, poems, and knowledge. Certain books, are so good that we need to create more books that are a continuation of those books, regardless of the author writing it or not. In this problem, a recurrent neural network will be built, trained on a book's text, and utilized to generate new text. This continuation of the book can be utilized in many scenarios and models can be trained for creative outlet or replacing authors in the future.

Analysis

To begin solving the problem, a book must be chosen. A personal favorite is romeo and juliet, as it was studied heavily in my high school classes. I was forced to write monalogues and poems modeled after the play, so it is held solid in my memory.

To prepare the book, it must be split into word vectors and tokenized. This was done simply as the book was pre-loaded into a text file so the setup and tokenizing was easy, as well as the standardization was completed with a simple function.

The book has a total of 28,864 words, and 6334 unique words represented in the vocab set. This shows that the book re-uses a huge number of words, and its known that Shakespere liked utilizing lengthy sentences and romantic language.

Methods

The first model built is a simple LSTM model. It consists of an embedding vector size of 50, and sequence length of 100. Within the sequential model, there is an embedding layer with a dimension of 20, one LSTM layer with 100 node output, one dropout layer with 20% probability, and one dense layer with a softmax activation function.

The second model built is a deep LSTM model, building off of the first model. The change made was more LSTM layers. Within the sequential model, there is an embedding layer, three LSTM layers all separated with dropout layers, each including regularization for the the three kernel, recurrent, and bias values, and two dense layers with the first having a relu activation function and second having a softmax actication function.

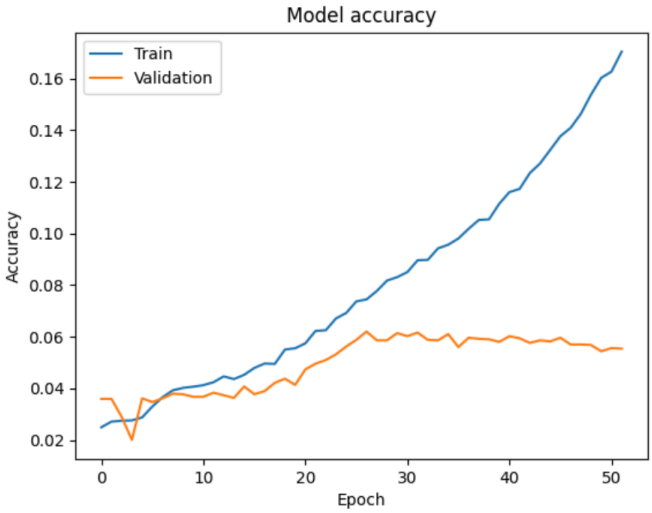
Both models utilized categorical crossentropy loss, adam optimizer, and accuracy as the validation metric. They were trained for 500 epochs, or until the early stopping ended the training early with a 50 epoch patience. Checkpoints including early stopping and model checkpoint were utilized and tested to ensure they were aiding the model's training and success. The model was then evaluated with analyzing the accuracy and loss against the validation accuracy and loss, paired with a visual plot.

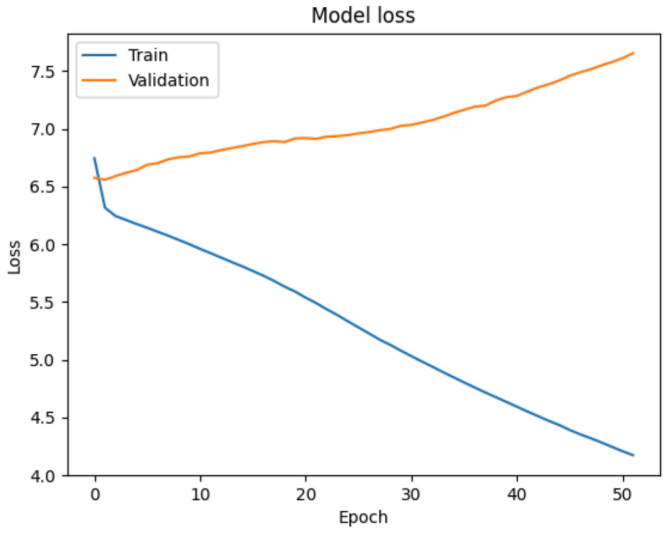
With the building of the first model, a basic approach was taken with utilizing classwork code and minimal changes to it. The model was validated on loss, so each change was compared to this value. I tested out learning rate schedulers but ended up not utilizing them due to their detrimental effects on the model overfitting. The second model also was experimented with starting basic and building more complex layers, adding regularization, dropout, and dense layers. I began by adding one LSTM layer at a time, then adding dropout layers, and finally testing out regularizers for the three values (kernel, recurrent, bias). My model was still overfitting so I played with output sizes as well as dropout values- both aided in reducing overfitting, but not enough to make the model perfect.

Results

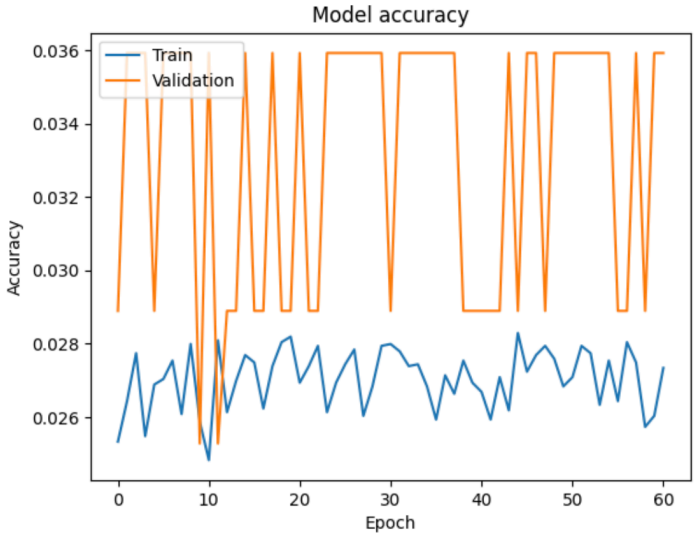
After training both models, it is clear the deeper LSTM model did better, but still was overfit and has lower performance metrics than desired.

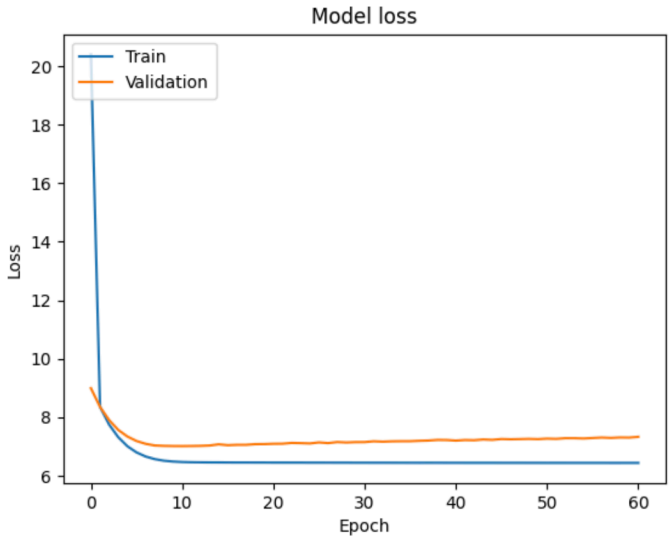
The basic LSTM was extremely overfit, with accuracy scores of 2.5%, loss of 6.74, and validation accuracy of 3% and validation loss of 6.57. In the plot below, it is clear that the model suffered greatly from overfitting as the accuracy curves were inverse relationships which is very bad. The loss curves were also inverse from eachother and shows the overfitting was terrible further into training. The model failed to generalize to the sequence data, and the output sequences were random and did not form complete sentences.





The deeper LSTM was still extremely overfit, but a bit better than the basic LSTM. The model metrics were as low as the basic model, but slightly better. Accuracy score was 2.5%, loss was 20.4, and validation accuracy was 3% and validation loss was 9. This shows that the model is horriby overfit and there needs to be some major changes to the model in order to improve it. The loss being 20 was very concerning. The basic LSTM model is not capturing the patterns in the data well enough to cause the output sequences to be random and overfit to the training data, however the deeper LSTM is suffering from the same issue. The plot below proves this further as the curves for both accuracy and loss spiral inversely to the validation curves and prove the overfitting is detrimental to the model's generative capabilities. The accuracy also jumped around a lot, which is not good for a model that is supposed to be predicting sequences and iteratively improving on accuracy, not jumping around which means the learning rate may need further tuning.





Both models have generated sequences that make no sense, and do not form complete sentences. Both plots show the model suffered greatly with overfitting and not generalizing to the data well. If I had to choose one model, it would be the shallow/basic LSTM model as it performed slightly better than the basic one, but in production I would not choose either, and fire whoever thought it was a good idea to use it. I would want to utilize transfer learning or tune the model much more than done here.

Model	Output
Basic LSTM - Seed	thou alack my child is dead and with my child my joys are buried friar lawrence peace ho for shame cure lives not in these confusions heaven and yourself had part in this fair maid now heaven hath all and all the better is it for the maid your part in her you could not keep from death but heaven keeps his part in eternal life the most you sought was her promotion for your heaven she should be and weep ye now seeing she is above the clouds as high as heaven itself o in this love you love your

Model	Output
Basic LSTM - Generated	mountain didst famine daughter buried talked swan ask rise native statue noses churchyard heralds vows tutor churchyard aching adjacent odd paper flies shrieks may scarf concludes each bade believe churchyard dumps battlements dog journey augmenting briefly bleeding bleeds side thence spoken scars grace torn remedy packthread steel between tears private
Deep LSTM - Seed	city is much bound to him juliet nurse will you go with me into my closet to help me sort such needful ornaments as you think fit to furnish me tomorrow lady capulet no not till thursday there is time enough capulet go nurse go with her to church tomorrow exeunt juliet and nurse lady capulet we shall be short in our provision now near night capulet tush i will stir about and all things shall be well i warrant thee wife go thou to juliet help to deck up her not to bed tonight let me alone play the housewife
Deep LSTM - Generated	carelessly pupil lazypuffing stabs a shows guests injuries unsavoury carrion bier gall lure treason kisses sups quench going befits behoveful forfeit smell talk report bower honestly calm adventure knows post coil learn knocks digging discover deliciousness souls madness singer food punto than catch weapons love miss beseeming hatred lark maid

Table 2: Model Generated Sequences

Model	Accuracy	Validation Accuracy	Loss	Validation Loss
Basic LSTM	2.49%	3.59%	6.74	6.57
Deep LSTM	2.53%	2.89%	20.4	8.99

: Model Metrics Output

Reflection

This project was very interesting in being able to see what the model was doing as we made it deeper, and in this case a deep learning model was needed. In the debugging process, I explored many different ways to create a deep learing LSTM, and in the next project or if I have time I would incorporate transfer learning. As transfer learning has huge benefits with CNN's and neural networks, I know it could help here.

For fun, I had chatGPT write me some code to utilize the Keras Tuner to tune almost every aspect of the LSTM model (layers, learning rate, embedding dimensions) to see how it performed to see if transfer learning is needed - and it still did not do as well as transfer learning would!