Nathan Scollar

DATA 641: Homework 3 Report

Link to Github Repository:

https://github.com/nathanscollar/Natural-Language-Processing-Homework-3

Dataset Summary:

The dataset utilized in this assignment is the IMDB review dataset, accessed from https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews. It contains 50,000 movie reviews that can be used for natural language processing tasks. After the initial tokenization step, I found that there were an average of 230.26 tokens per review, and the full vocabulary size was 122,664 unique words. For my preprocessing, the initial step was to load in the dataframe and split it in half into train and test data. I simply used the first 25,000 reviews as training and the last 25,000 as testing. For each review, the preprocess_text function was used to convert all text to lower case and all special characters and punctuation are removed. From there, I used the Keras Tokenizer with the num_words parameter set to 10,000 to tokenize the sentences and only keep the 10,000 most frequent words, with everything else being replaced with an <OOV> token. Then, each review is converted to a sequence of token IDs. The last step of preprocessing is padding/truncating reviews. For that, I made it so that for each length of sequence, it is the first x tokens that remain. For example, with sequence length 25 and a 100 token review, the first 25 tokens from that review would be kept. Lastly, I had to encode the labels as 0's and 1's to allow for binary classification.

Model Configuration:

I utilized a model that met all of the parameters outlined in the assignment. The first layer in the model is an embedding layer with size 100, using Embedding from tensorflow.keras.layers. After that, a hidden layer with size 64 is included, and followed by a dropout layer with a dropout value of 0.3. That is then followed by another hidden layer with size 64 and another dropout layer with a dropout value of 0.3. The final layer is a fully connected output layer, using Dense from tensorflow.keras.layers, which uses a sigmoid activation function. The hidden layers utilize different layers from tensorflow.keras.layers depending on the model, with SimpleRNN for the RNN model, LSTM for the LSTM model, and Bidirectional(LSTM) for the Bidirectional LSTM model. The dropout layers were created using the Dropout layer from tensorflow.keras.layers. For gradient clipping, I performed my testing with a clipnorm value equal to 5, which is set within each of the optimizers. This seemed like a logical choice based on my research of strong clipnorm values, although it is possible that a slightly smaller choice would have been better given the small size of this model. I kept other optimizer parameters at the default, such as the learning rate.

Comparative Analysis:

My full table of results is available in the linked Github repository under metrics_DATA643_HW3.csv. In this section however, I have included some additional analysis that is not performed in the discussion section to get a better idea of different trends in accuracy, F1 score, and training times.

Model	Sequence Length	Training Time per Epoch (s)
RNN	25	8.36
RNN	50	11.76
RNN	100	20.54
LSTM	25	11.41
LSTM	50	18.88
LSTM	100	32.23
Bidirectional LSTM	25	15.1
Bidirectional LSTM	50	24.22
Bidirectional LSTM	100	43.54
Overall	25	11.62
Overall	50	18.35
Overall	100	32.11

Figure 1: Table comparing training times per epoch by sequence length and model.

The table above looks at training time by epoch over different models and sequence lengths, along with looking at the overall values over all models for each sequence length. There was a clear trend both with models and sequence length and training time per epoch. I noticed that as sequence length increases, the training time increases on all three models. Along with that, as models got more complex, the training time increased, with RNNs being the quickest and Bidirectional LSTMs taking the longest.

Model	Parameters	Accuracy	F1 Score	Training Time
LSTM	ReLU, RMSProp, 100, Gradient Clipping	0.819	0.8142	32 sec/epoch
LSTM	Tanh, RMSProp, 100, Gradient Clipping	0.8167	0.8169	32 sec/epoch
RNN	ReLU, RMSProp, 100, No Gradient Clipping	0.81	0.8115	23 sec/epoch
LSTM	Tanh, RMSProp, 100, No Gradient Clipping	0.8082	0.7952	31.8 sec/epoch
Bidirectional LSTM	Sigmoid, Adam, 100, Gradient Clipping	0.8006	0.82	45.4 sec/epoch

Figure 2: This figure shows my five best configurations by test accuracy.

The above figure shows my five best resulting configurations, and will be addressed further in the discussion section.

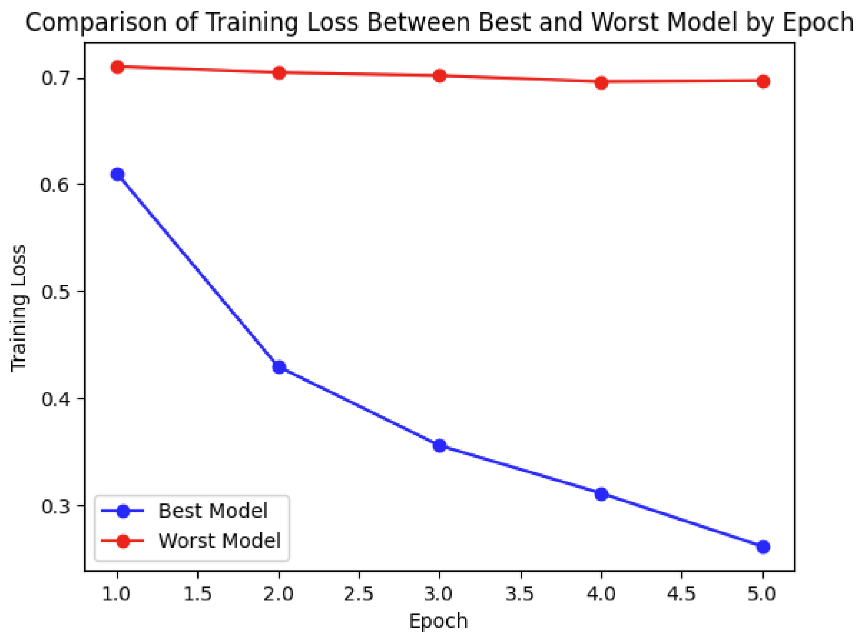

Figure 3: This figure shows training loss between a very strong model and a very weak model

The above plot in figure 3 was one of the two plots that was required for submission and is also included in my Github repository, but I wanted to address it further here, because the other one of the two plots is about comparing different sequence lengths in terms of F1 score and Accuracy and it is already addressed in the discussion section. The code above actually does not take my best model, but it utilizes the second highest accuracy model shown above. My best model actually had some exploding gradients in the later epochs that made the plot incredibly distorted and unable to show the difference between the two models. Therefore, I used my second best model as the best model above, which is the LSTM with Tanh activation, RMSProp optimizer, 100 sequence length, and Gradient Clipping. The worst model above is one of the many that got 0.501 accuracy by predicting all 1's, which in this case was an RNN with Sigmoid activation, SGD optimizer, 50 sequence length, and gradient clipping. We can see how the loss of the best model significantly decreases over time. Many of the models did overfit to some capacity, but it still resulted in a strong test accuracy (0.8167 in this case). The worst model above didn't really learn anything, which is why it predicted all 1's for the outputs.

Discussion:

As seen in Figure 2 above, my highest performing configuration out of all 162 combinations tested was the LSTM with the ReLU activation function and the RMSProp optimizer, with a sequence length of 100 and gradient clipping enabled. This had an accuracy value of 0.819, which was the highest of any of the 162 combinations tested. It also had a strong F1 score of 0.8142.

Model	Sequence Length	Accuracy	F1 Score
RNN	25	0.6073	0.6307
RNN	50	0.6085	0.6460
RNN	100	0.6077	0.6252
LSTM	25	0.6146	0.5932
LSTM	50	0.6406	0.6232
LSTM	100	0.6551	0.6316
Bidirectional LSTM	25	0.6118	0.6084
Bidirectional LSTM	50	0.6365	0.6350
Bidirectional LSTM	100	0.6128	0.6081

Overall	25	0.6112	0.6108
Overall	50	0.6285	0.6347
Overall	100	0.6252	0.6216

Figure 4: This figure compares the Accuracy and F1 score of different sequence lengths and models

Sequence length certainly impacted performance, with differing effects depending on the model being used. The above table takes a look at the averaged accuracy and F1 values for the three model types broken down by sequence length. For each model/sequence length combination, there were 18 accuracy and F1 values averaged together, since there is one with each combination of the three activation functions, three optimizers, and two levels of gradient clipping. The overall level includes all three model types as well, meaning there were 54 accuracy and F1 values averaged together for each sequence length.

Looking at the data in the table, the overall data showed the strongest performance with a sequence length of 50, with the highest accuracy and F1 score. This was followed by a sequence length of 100, which had the second highest accuracy and second highest F1 score, whereas a sequence length of 25 had the worst accuracy and worst F1 score. This can be broken down by model to see further trends. For the RNN model, sequence length seemed to have the smallest impact of the three models. All three sequence lengths had very similar accuracy values, with slightly more variance in F1 scores. The sequence length of 50 had the highest F1 score, with a sequence length of 100 having the worst F1 score. This does make sense as RNNs are known for struggling with long term dependencies, so it may not have been able to handle the large sequence lengths as well. For the LSTM model, there was a clear trend that increasing sequence length resulted in increasing accuracy and F1 score. There was a major jump between sequence lengths of 25 and 50, with average accuracy increasing by over 2.5% and average F1 score increasing by exactly 0.03. These values increased again with a sequence length of 100, although the increase was not as large. It is noticeable that the LSTM model was able to handle long term dependencies very well with its strong performance on longer sequence lengths. Lastly, the Bidirectional LSTM model had easily the best performance with a sequence length of 50, with an accuracy over 2% higher than the other two sequence lengths and an F1 score over 0.03 larger than the other two sequence lengths.

To conclude, looking at sequence length, RNN performed best with a sequence length of 50, LSTM performed best with a sequence length of 100, and Bidirectional LSTM performed best with a sequence length of 50. Overall, the highest accuracy and F1 score was found with a sequence length of 50, and there was a noticeable increase in accuracy of 50 and 100 sequence lengths over the 25 sequence length.

Model	Optimizer	Accuracy	F1 Score
RNN	Adam	0.6564	0.6836
RNN	SGD	0.5327	0.5237
RNN	RMSProp	0.6344	0.6946
LSTM	Adam	0.7322	0.7515
LSTM	SGD	0.5084	0.3760
LSTM	RMSProp	0.6696	0.7204
Bidirectional LSTM	Adam	0.7143	0.7031
Bidirectional LSTM	SGD	0.5110	0.4401
Bidirectional LSTM	RMSProp	0.6358	0.7083
Overall	Adam	0.7010	0.7127
Overall	SGD	0.5173	0.4466
Overall	RMSProp	0.6466	0.7078

Figure 5: This figure shows Accuracy and F1 scores by optimizer and model

The choice of optimizer certainly impacted performance as well. The above table shows the same breakdown that I performed with sequence length, but this time with different optimizers. I broke it down by model, along with including three overall rows that average over all models. Each model and optimizer pair contains accuracy and F1 values averaged from 18 configurations, with those being combinations of the three different activation functions, the three different sequence lengths, and whether or not there is gradient clipping. As a whole, the Adam optimizer performed the best over the entirety of the data, with the highest accuracy and F1 score. The RMSProp optimizer was easily the second best, with a solid accuracy and a strong F1 score. However, the SGD optimizer was not useful at all. It resulted in many instances where it simply predicted all zeros or all ones depending on the activation function, and it very rarely saw high accuracy scores. These trends remain pretty much the same over the individual models.

The results of gradient clipping over the entirety of the dataset were rather interesting, as there were many configurations where gradient clipping had a very minor impact or no impact at all. There were other configurations where the impact was massive. For example, in the RNN model, with ReLU activation and RMSProp optimizer and 100 sequence length, the accuracy was 0.81 without gradient clipping, but dropped to a value of 0.5738 with gradient clipping.

However, with the same parameters except in the LSTM model, gradient clipping significantly helped the accuracy, increasing it from 0.7449 without gradient clipping to 0.819 with gradient clipping. As a whole, the average accuracy with or without gradient clipping throughout the dataset was very similar (0.6207 with and 0.6220 without), although gradient clipping helped in many situations while also hurting in others. I've learned that gradient clipping is certainly useful for improving the stability of a model by handling exploding gradients, but I have also learned that different models and activation functions end up with gradients in different ranges, so there definitely isn't one threshold for gradient clipping that works for every model. It is a useful tool for handling exploding gradients and through additional testing can be used effectively.

Conclusion:

The optimal configuration with the specified model under CPU constraints was an LSTM with the ReLU activation function and the RMSProp optimizer, with a sequence length of 100 and gradient clipping enabled. This had a resulting accuracy of 0.819, which was the highest of the 162 configurations I tested. Throughout my testing, I did notice some potential issues with F1 score, and while F1 score is still a valid metric, I believe accuracy was the best measure of success in this project. Many of the tested configurations ended up having an accuracy value of 0.501 and an F1 score of 0.6676, mostly with the SGD optimizer. There were also other configurations that finished with an accuracy value of 0.499 and an F1 score of 0. This confused me at first, but after some further digging, I noticed that this was due to certain configurations always choosing either 0 or 1. The first set of values (0.6676 F1 score) was due to a model always selecting 1, whereas the second set of values (0 F1 score) was due to a model always selecting 0. This showed me that F1 score was slightly biased towards models that more frequently picked 1 rather than 0. Because the dataset is not at all imbalanced (approximately 50.01% positive and 49.99% negative), accuracy is a solid metric for comparison, and this is why I believe I made the right choice. My selected model had an average training time per epoch of 32 seconds with CPU constraints, which meant over the 5 epochs it took about 4 and a half minutes to train, which is a reasonable amount of time to train a single model.

Works Cited:

ChatGPT, "chatgpt.com". Used strictly for assistance in coding.