Comparative Analysis of RNN Architectures for Sentiment Classification

Varsha Sabhnani

Code Repository: https://github.com/vsabhnan/RNN

Dataset Summary

The raw dataset was formatted as a table with the first column representing reviews and the second column representing the sentiment (positive or negative). The raw review text was converted to lower case and stripped of punctuation and special characters. The text was then tokenized using nltk.tokenize. The top 10,000 most frequent words were retained, after which the reviews were converted into a sequence of token IDs. Finally, the sequences were padded to lengths of 25, 50, and 100 respectively.

The dataset consisted of 50,000 reviews. These were split equally into training and testing using the first 25,000 as the training set and the remaining as the testing set. Below is a table summarizing the distribution of the review length (based on number of words):

Table 1: Distribution of Review Length

Statistic	Number of Words
Mean	230.3
Std. Dev	170.6
Minimum	4
1st Quartile	126
Median	172
3rd Quartile	280
Maximum	2469

The average length of review was 230.3 words while the median was 172. The first was 126 while the third quartile was 280, indicating that a considerable amount of reviews were of a good length. However, there were definitely outliers with very short and very long reviews, which is somewhat expected in a dataset like this.

There were 122,664 unique words seen in the training set, of which the most frequent 10,000 were kept for tokenization.

Model Configuration

In each of the models tested, the following were constant:

- Embedding layer of size 100
- 2 hidden layers of size 64
- Dropout of 0.4 to reduce overfitting
- Batch size of 32
- Fully connected output layer with sigmoid activation for binary classification
- 5 epochs for training

Loss was calculated as binary cross entropy loss. Note that experiments were run on a CPU.

The following parameters were varied:

- Architecture: RNN, LSTM, Bidirectional LSTM
- Activation function: Sigmoid, ReLU, Tanh
- Optimizer: Adam, Stochastic Gradient Descent (SGD), RMSProp
- Sequence Length: 25, 50, 100
- Stability strategy: No gradient clipping, Gradient clipping

Results

I started with a baseline model with the following parameters:

- Architecture: LSTM
- Activation functions: Tanh
- Optimizer: Adam
- Input sequence length: 50
- No gradient clipping

I then each of the parameters one by one. The table below summarizes the results of these experiments:

Table 2: Preliminary Results from Experiments

Architecture	Activation Function	Optimizer	Sequence Length	Gradient Clipping	Accuracy	F1 Macro	Time Per Epoch
LSTM	tanh	adam	50	FALSE	0.724	0.722	30.932
RNN	tanh	adam	50	FALSE	0.729	0.729	16.150
BiLSTM	**tanh**	**adam**	**50**	**FALSE**	**0.734**	**0.731**	**45.272**

LSTM	**sigmoid**	**adam**	**50**	**FALSE**	**0.753**	**0.753**	**30.629**
LSTM	relu	adam	50	FALSE	0.696	0.694	33.339
LSTM	tanh	sgd	50	FALSE	0.499	0.333	29.958
LSTM	tanh	rmsprop	50	FALSE	0.712	0.705	32.124
LSTM	tanh	adam	25	FALSE	0.683	0.683	19.625
LSTM	**tanh**	**adam**	**100**	**FALSE**	**0.805**	**0.805**	**55.930**
LSTM	**tanh**	**adam**	**50**	**TRUE**	**0.750**	**0.750**	**31.147**

The first row represents the baseline model. Remaining results are color-coded based on experiment. The best result of each experiment (in terms of accuracy and F1 score) is bolded if it is better than the baseline model.

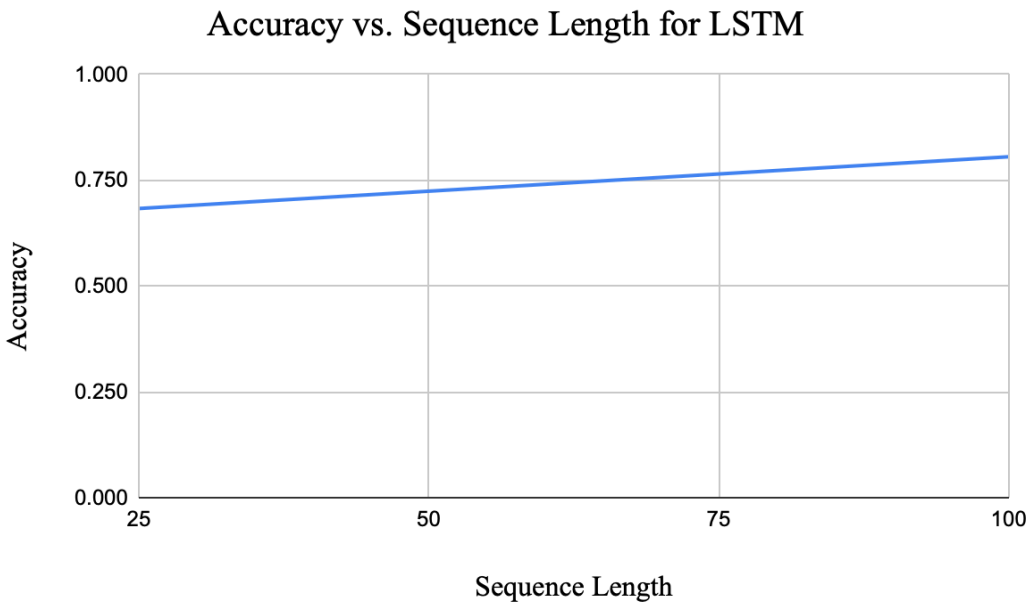
The results achieved by the baseline model were relatively good: accuracy of 0.724, F1-score of 0.722, and a running time of 31 seconds per epoch.

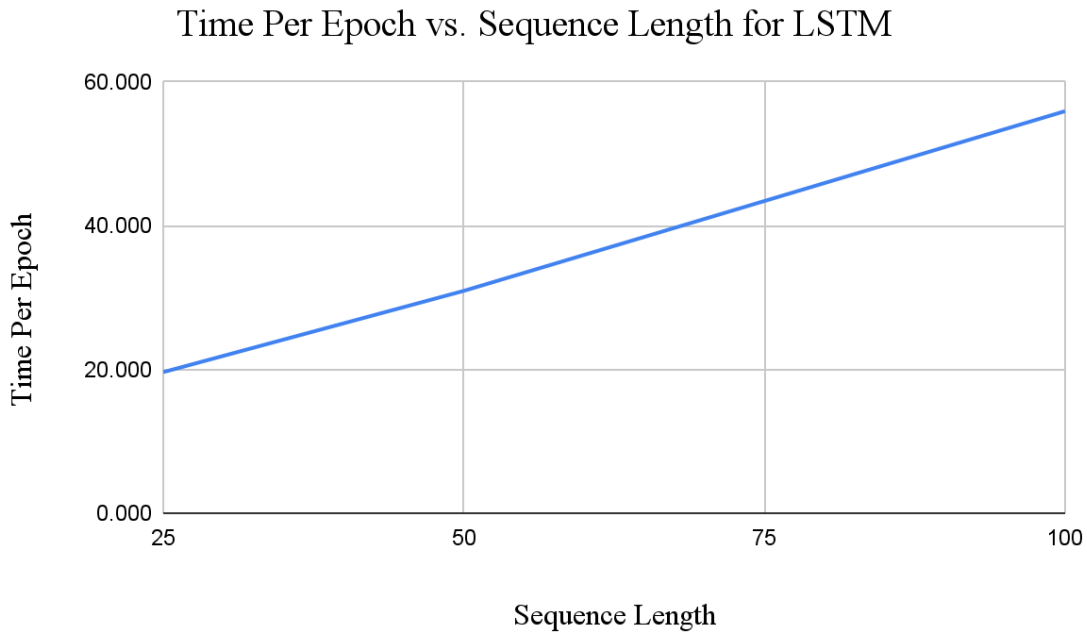
In the first set of experiments (blue), all models performed fairly well with an accuracy between 0.724 and 0.734, and F1 macro between 0.722 and 0.731. LSTM gave the worst results, while Bidirectional LSTM gave the best results. In terms of time taken, RNN was the quickest at 16 seconds per epoch, while Bidirectional LSTM was the slowest at 45 seconds per epoch. LSTM was in the middle at around 31 seconds per epoch.

In the next set of experiments (green), the activation functions were tested. The sigmoid activation function provided the best results with an accuracy of 0.753 and F1 score of 0.753. ReLU performed the worst with an accuracy of 0.696 and F1-score of 0.694. All three activation functions performed similarly in terms of time per epoch, ranging from 30.6 seconds per epoch to 33.3 seconds per epoch.

The next set of experiments tested the effect of the optimizers. Both stochastic gradient descent and RMSProp performed worse than Adam in about the same time per epoch. In particular, stochastic gradient descent resulted in an accuracy of around 0.5, essentially resulting in performance similar to random chance.

The next set of experiments tested the effect of sequence length. As would be expected, longer sequence length resulted in better outcomes. A sequence length of 100 resulted in an accuracy of 0.805 and an F1 score of 0.805. In comparison, a sequence length of 25 resulted in a low accuracy and F1-score of 0.683. However, this came at the expense of time. A sequence length of 100 took almost twice as long per epoch than a sequence length of 50. Below are two plots showing the effect of changing sequence length on accuracy and time taken.





Both plots show a linear relationship, but the relationship between sequence length and time is much steeper than the relationship between sequence length and accuracy. This means that increasing the sequence length will give small increases in accuracy but add a comparatively larger burden on run time.

Finally, using gradient clipping seemed to give better results than no gradient clipping, increasing the accuracy and F1-score to 0.750. It took about the same amount of training time as running it without gradient clipping.

The biggest effect on accuracy and F1 score seems to be achieved by increasing sequence length. The next two improvements were changing the activation function to sigmoid and implementing gradient clipping. Finally, changing the architecture could also increase the accuracy and F1 score, but those improvements seem more minimal compared to the first few.

I performed the first three changes incrementally to measure the effect on the accuracy and F1 score. As the first set of experiments only involved changing the parameters one-by-one, it is not guaranteed that combining all the changes together would result in better results. The results are summarized in the table below:

Table 3: Comparison of Baseline Model with Optimal Changes from Table 2

Architecture	Activation Function	Optimizer	Sequence Length	Gradient Clipping	Accuracy	F1 Macro	Time Per Epoch
LSTM	tanh	adam	50	FALSE	0.724	0.722	30.932
LSTM	tanh	adam	100	FALSE	0.805	0.805	55.930
LSTM	**sigmoid**	**adam**	**100**	**FALSE**	**0.820**	**0.820**	**52.706**
LSTM	sigmoid	adam	100	TRUE	0.819	0.819	51.294

Updating the baseline model to have an activation function of sigmoid and a sequence length of 100 produced the best results with an accuracy and F1-score of 0.82. However, the time taken per epoch did increase significantly from roughly 31 seconds to 53 seconds. Implementing gradient clipping with this specification did not produce significantly different results, and the run time was about the same as well.

I then used the parameters from the best model (sigmoid activation function, adam optimizer, sequence length of 100 and no gradient clipping) with RNN and bidirectional LSTMs to compare their performance with LSTMs. The results are presented below:

Table 4: Comparison of LSTM, RNN and Bidirectional LSTM with Optimal Parameters

Architecture	Activation Function	Optimizer	Sequence Length	Gradient Clipping	Accuracy	F1 Macro	Time Per Epoch
LSTM	**sigmoid**	**adam**	**100**	**FALSE**	**0.820**	**0.820**	**52.706**
RNN	sigmoid	adam	100	FALSE	0.644	0.644	30.194
BiLSTM	sigmoid	adam	100	FALSE	0.809	0.808	73.050

The LSTM model with optimal parameters outperforms both RNN and bidirectional LSTM. Impressively, the LSTM model runs significantly quicker than bidirectional LSTM. It is also worth noting that the performance of the RNN model seems to suffer from these parameter changes, either due to the change in activation function or due to a change in sequence length. Since these parameters were optimised with the LSTM baseline, it is possible that different parameters would optimize the other models. Therefore, I perform a final set of experiments to check whether the performance of the RNN and bidirectional LSTM models can be improved further.

Table 5: Comparison of Best LSTM model with RNN and Bidirectional LSTM Variations

Architecture	Activation Function	Optimizer	Sequence Length	Gradient Clipping	Accuracy	F1 Macro	Time Per Epoch
LSTM	**sigmoid**	**adam**	**100**	**FALSE**	**0.820**	**0.820**	**52.706**
RNN	tanh	adam	25	FALSE	0.676	0.675	10.361
RNN	relu	adam	25	FALSE	0.701	0.700	11.567
RNN	sigmoid	adam	25	FALSE	0.679	0.679	11.199
RNN	tanh	adam	50	FALSE	0.729	0.729	15.300
RNN	relu	adam	50	FALSE	0.750	0.750	15.665
RNN	sigmoid	adam	50	FALSE	0.669	0.668	15.848
RNN	tanh	adam	100	FALSE	0.497	0.384	29.878
RNN	**relu**	**adam**	**100**	**FALSE**	**0.786**	**0.785**	**32.345**
RNN	sigmoid	adam	100	FALSE	0.644	0.644	29.482
BiLSTM	tanh	adam	100	TRUE	0.801	0.801	77.572
BiLSTM	tanh	adam	100	FALSE	0.803	0.803	78.111
BiLSTM	**relu**	**adam**	**100**	**TRUE**	**0.819**	**0.819**	**77.179**
BiLSTM	relu	adam	100	FALSE	0.673	0.671	77.097

The best LSTM specification outperforms both the RNN and Bidirectional LSTM models, although the performance of the bidirectional LSTM is very close. However, the LSTM model runs significantly faster which is in its favor. The RNN model with ReLU activation function also achieves a relatively high performance (accuracy of 0.786 and F1 score of 0.785), however, it is clear from the remaining results that it is quite sensitive to the parameters used. BiLSTM, on the other hand, is generally more stable, although performs poorly when ReLU is used without gradient clipping.

Conclusion

Based on the experiments, the best results were achieved using an LSTM with sequence length 100, adam optimizer, no gradient clipping, and sigmoid activation function. While bidirectional LSTM also offered good results, the lower run time in the case of LSTMs would make it preferable when using a CPU, especially when hyperparameter tuning is required. RNNs could be a decent option if computing power is limited, however, their performance is relatively unstable and might require more hyperparameter tuning. Additionally they do not seem to achieve as good results as either LSTMs or bidirectional LSTMs. Therefore, when considering stability, speed and performance, LSTMs would be the best choice.