Recurrent Neural Networks w/ SimpleRNN and LSTM

CSE547-51 Homework 3

Phoenix Daak

03/04/2025

Presentation Link: 2025-03-06 14-44-45.mov

Problem 1

Outline:

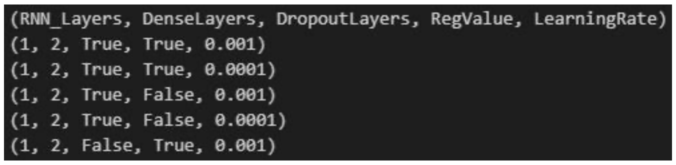
During this experiment a baseline recurrent neural network (RNN) was built to analyze the how travelers expressed their feelings on Twitter. After this model was trained, it should be able to predict the sentiment of a new tweet. For this experiment, SimpleRNN was used as the RNN layers along with densely connected layers. Because RNN's take in sequential data, the tweets within the Twitter dataset were encoded so that the dimensions fit RNN layers. Once the data formatting was complete, the data was fed through and networks. Different experiments with the hyperparameters were performed to evaluate what combination of RNN layers, dense layers, dropout, regularization, and learning rate resulted in the highest validation accuracy.

Data Formatting:

The original data provided were tweets in the English language that were posted on Twitter during the year of 2015. To train the model on this data, it needed to be formatted properly so that the RNN layers could accept it. For this, the entire dataset was tokenized where every unique word was tracked within a dictionary. From there, sequences were formed form the tokenized data. One-hot encoding was then performed to translate the words into an array where each column represented each unique word. This was performed on both the training data and the training labels. The data was then split 80/20 for the training and validation data generators. These generators were fed into the RNN where the experimentation began.

Generating Hyperparameter Permutations:

Different combination of hyperparameters were used to experiment to find the best combination. For this, lists that contained different numbers of RNN layers, Dense layers, dropout, regularization values, and learning rates were initialized. The product of each list was performed and combined where every combination of the initialized hyperparameters were considered. See the figure below for the formatting of the parameters:



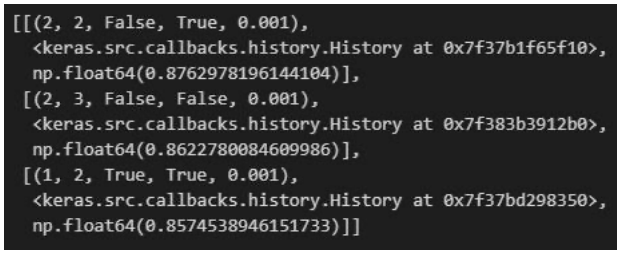
```
(RNN_Layers, DenseLayers, DropoutLayers, RegValue, LearningRate)
(1, 2, True, True, 0.001)
(1, 2, True, True, 0.0001)
(1, 2, True, False, 0.001)
(1, 2, True, False, 0.0001)
(1, 2, False, True, 0.001)
```

Figure 1. First 5 of 32 combinations of hyperparameters tested

Each of the combinations were fed into a model initialization function where the model was created using these parameters. The defaulted architecture within the initialization process included at least 1 RNN layer and 1 Dense layer for the 'softmax' categorical classification function. The top three combinations that resulted in the best validation accuracy were plotted to analyze the fitting of the graphs.

Results:

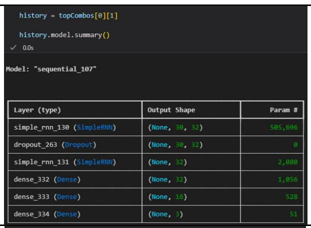
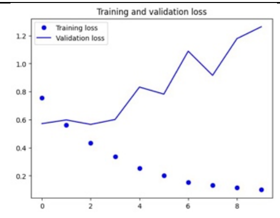
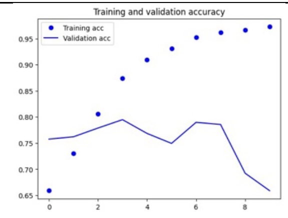
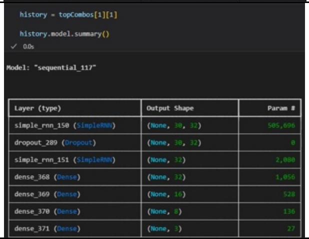
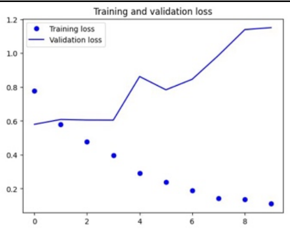
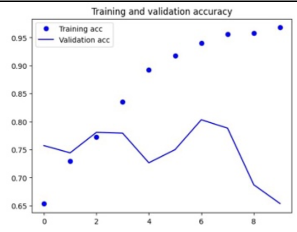
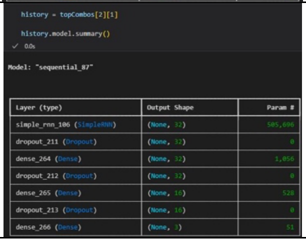
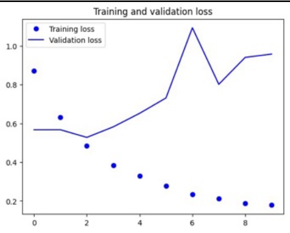
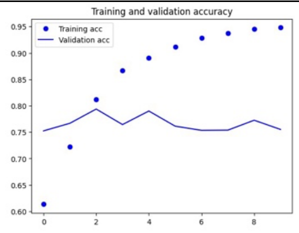
While training models with the different combinations, the results were stored within a list that contained every result from each iteration. It was then sorted in descending based on the average validation accuracy so that the top combinations were first in the list. The sorted list can be seen below:



```
[[(2, 2, False, True, 0.001),
  <keras.src.callbacks.history.History at 0x7f37b1f65f10>,
  np.float64(0.8762978196144104)],
 [(2, 3, False, False, 0.001),
  <keras.src.callbacks.history.History at 0x7f383b3912b0>,
  np.float64(0.8622780084609986)],
 [(1, 2, True, True, 0.001),
  <keras.src.callbacks.history.History at 0x7f37bd298350>,
  np.float64(0.8574538946151733)]]
```

Figure 2. Sorted list of hyperparameters based on validation accuracy (RNN_Layers, Dense Layers, Dropout, Regularization, Learning Rate)

The top three combinations were plotted to see how the training lined up with the validation during the training process. Below are the results of the top three models that yielded the highest validation accuracy:

Parameters	Validation Accuracy	Model Architecture	Loss	Accuracy
(2, 2, False, True, 0.001)	0.876			
(2, 3, False, False, 0.001)	0.862			
(1, 2, True, True, 0.001)	0.857			

```
history = topCombos[0][1]

history.model.summary()
✓ 0.0s

Model: "sequential_107"

┃ Layer (type)                 ┃ Output Shape           ┃       Param # ┃
│ simple_rnn_130 (SimpleRNN)   │ (None, 30, 32)         │       505,696 │
│ dropout_263 (Dropout)        │ (None, 30, 32)         │             0 │
│ simple_rnn_131 (SimpleRNN)   │ (None, 32)             │         2,080 │
│ dense_332 (Dense)            │ (None, 32)             │         1,056 │
│ dense_333 (Dense)            │ (None, 16)             │           528 │
│ dense_334 (Dense)            │ (None, 3)              │            51 │
```

```
history = topCombos[1][1]

history.model.summary()
✓ 0.0s

Model: "sequential_117"

┃ Layer (type)                 ┃ Output Shape           ┃       Param # ┃
│ simple_rnn_150 (SimpleRNN)   │ (None, 30, 32)         │       505,696 │
│ dropout_289 (Dropout)        │ (None, 30, 32)         │             0 │
│ simple_rnn_151 (SimpleRNN)   │ (None, 32)             │         2,080 │
│ dense_368 (Dense)            │ (None, 32)             │         1,056 │
│ dense_369 (Dense)            │ (None, 16)             │           528 │
│ dense_370 (Dense)            │ (None, 8)              │           136 │
│ dense_371 (Dense)            │ (None, 3)              │            27 │
```

```
history = topCombos[2][1]

history.model.summary()
✓ 0.0s

Model: "sequential_87"

┃ Layer (type)                 ┃ Output Shape           ┃       Param # ┃
│ simple_rnn_106 (SimpleRNN)   │ (None, 32)             │       505,696 │
│ dropout_211 (Dropout)        │ (None, 32)             │             0 │
│ dense_264 (Dense)            │ (None, 32)             │         1,056 │
│ dropout_212 (Dropout)        │ (None, 32)             │             0 │
│ dense_265 (Dense)            │ (None, 16)             │           528 │
│ dropout_213 (Dropout)        │ (None, 16)             │             0 │
│ dense_266 (Dense)            │ (None, 3)              │            51 │
```

Figure 3. Results of top three hyperparameter combinations

The hyperparameters that resulted in the highest validation accuracy was taken into consideration when plotting. Since all permutations of the predefined parameters were generated, new models were trained for each combination. The validation accuracy during the training process was tracked and returned. The training history of these models were then plotted as seen before. Since the experiment relied only on the validation accuracy, overfitting was common within these samples.

The training and validation curves deviated from each other rather quickly (about 2 epoch). However, the results still presented high accuracy within the beginning stages of the training process.

Problem 2

Outline:

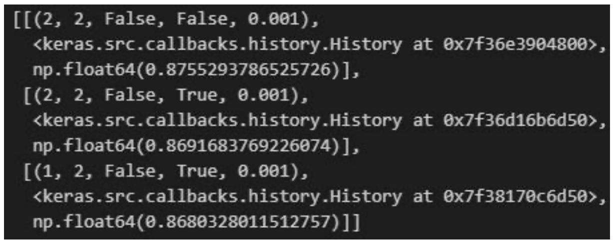
For this experiment, the type of recurrent layers was switched. Instead of utilizing the SimpleRNN for the RNN layers, long short-term memory (LSTM) layers were used. This resulted in a different type of fitting with similar validation accuracy as the previous experiment.

Data formatting:

The data formatting for LSTM was the same used in the previous experiment. This allowed for the one-hot encoded data to be used again. The data was still split 80% training data and 20% validation data. After this was configured, permutations like before were generated and tested based off their validation accuracy.

Generating Hyperparameter Permutations:

After the data was formatted, permutations were then made again. These included different number of RNN layers, Dense layers, whether to include dropout layers and regularization, and finally the learning rate. A sample of the results show how the parameters were formatted along with which combinations resulted in the best validation accuracy:



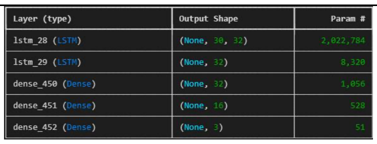
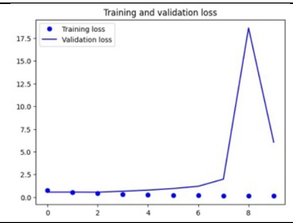
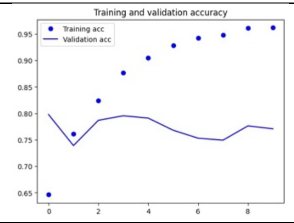
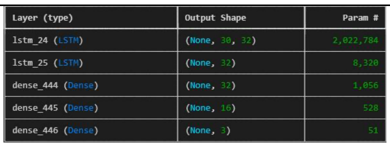
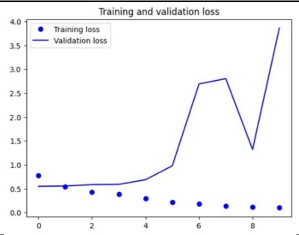
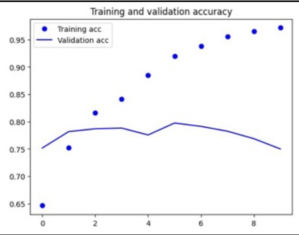
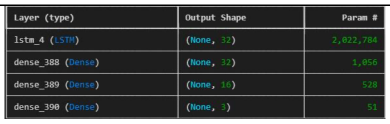
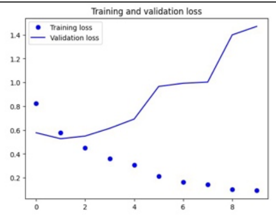
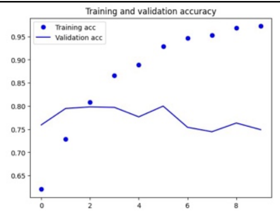
```
[[(2, 2, False, False, 0.001),
  <keras.src.callbacks.history.History at 0x7f36e3904800>,
  np.float64(0.8755293786525726)],
 [(2, 2, False, True, 0.001),
  <keras.src.callbacks.history.History at 0x7f36d16b6d50>,
  np.float64(0.8691683769226074)],
 [(1, 2, False, True, 0.001),
  <keras.src.callbacks.history.History at 0x7f38170c6d50>,
  np.float64(0.8680328011512757)]]
```

Figure 4. Format of hyperparameters (RNN Layers, Dense Layers, Dropout, Regularization, Learning Rate)

These combinations of parameters were tested using a model initialize function that build a network based on the list of parameters to use.

Results:

After running the test, the combinations along with their validation accuracy acquired during the testing were tracked in list that was shown before. This list was then sorted based on their accuracy and the top three accuracies were taken and plotted. Below show a table of the graphs for the top three combinations of hyperparameters during the test.

Parameters	Validation Accuracy	Architecture	Loss	Accuracy
(2, 2, False, False, 0.001)	0.876			
(2, 2, False, True, 0.001)	0.869			
(1, 2, False, True, 0.001)	0.868			

```
Layer (type)          Output Shape         Param #
lstm_28 (LSTM)        (None, 30, 32)       2,022,784
lstm_29 (LSTM)        (None, 32)           8,320
dense_450 (Dense)     (None, 32)           1,056
dense_451 (Dense)     (None, 16)           528
dense_452 (Dense)     (None, 3)            51
```

```
Layer (type)          Output Shape         Param #
lstm_24 (LSTM)        (None, 30, 32)       2,022,784
lstm_25 (LSTM)        (None, 32)           8,320
dense_444 (Dense)     (None, 32)           1,056
dense_445 (Dense)     (None, 16)           528
dense_446 (Dense)     (None, 3)            51
```

```
Layer (type)          Output Shape         Param #
lstm_4 (LSTM)         (None, 32)           2,022,784
dense_388 (Dense)     (None, 32)           1,056
dense_389 (Dense)     (None, 16)           528
dense_390 (Dense)     (None, 3)            51
```

Figure 5. Results and architecture for testing permutations of hyperparameters

The top combinations yielded different parameters when using the LSTM layers. However, the accuracy was still relevant to that of SimpleRNN. The fitting on the training and validation loss presented less over fitting during the earlier epochs. In 2/3 tests, the training and validation graphs start to deviate at around 4 epoch compared to 2 epoch in SimpleRNN.