Report

CSI 5138 Homework Exercise 3

Pavan Balaji Kumar

Student number: 300169572

In this assignment, Vanilla RNN and Long short-term memory was used to solve the IMDB sentiment classification problem. Both the models were trained with its hidden state set to one of 20, 50, 100, 200, 500 values. The results obtained and observation made are discussed in this report.

Some of the terminologies used are:

- LSTM_model_x – denotes a Long short-term memory model where x denotes the id of the model.
- RNN_model_x - denotes a Vanilla RNN model where x denotes the id of the model.
- X_train, Y_train – The training dataset and training target variable
- X_test, Y_test – The testing dataset and testing target variable
- word_embeddings – Glove pretrained word embeddings with 300 dimensions
- word_index – The word and its sequence number given by the tokenizer for all the words in the dataset
- embedding_weights – the word_embeddings for the words in word_index
- get_embedding_matrix – Function that returns embedding_weights
- get_word_embeddings – Function that returns the glove word embeddings
- get_data – Function to load data required from a directory
- text_to_sequences – function to convert the text input to sequence input
- create_LSTM – Function to create a LSTM model
- create_RNN – Function to create a RNN model

The IMDB dataset consists of 12500 positive reviews and 12500 negative reviews in the training dataset and testing dataset for various movies in IMDB's repository. The task is to classify each review as positive or negative based on the input sequence. LSTM and Vanilla RNN was used to solve this task. Let ua look at the LSTM model's performance first. Only the first 200 words are used from the review to perform the task.

Obtaining the embedding_weights required for the embedding layer from the glove pre-trained embeddings takes a lot of time and resources. Hence, I have already loaded the word embeddings corresponding to the index's words and stored it as a NumPy file to retrieve it for later use. I have attached this file in the zip file..

1. Performance of LSTM on the IMDB dataset

The LSTM model, with its state dimension set to 20, performs reasonably well in the task of classifying positive and negative reviews. The model's architecture starts with an

embedding layer, whose initial weights are set to glove pre-trained embedding given by embedding weights. It is followed by a LSTM layer with it state dimension set to 20 and dropout set to 0.2 and l2 regularization with regularization parameter set to 0.001 to avoid overfitting. The output of the LSTM is set into a fully connected neural network which performs the operation of logistic regression to determine the probabilities of the output.

```
Model: "LSTM_with_state_dimension_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 200, 300)          26574600
_________________________________________________________________
lstm (LSTM)                  (None, 20)                25680
_________________________________________________________________
flatten (Flatten)            (None, 20)                0
_________________________________________________________________
dense (Dense)                (None, 128)               2688
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 130
=================================================================
Total params: 26,611,354
Trainable params: 36,754
Non-trainable params: 26,574,600
```

Figure 1 LSTM with 20 state dimensions

The LSTM with state dimension 20 got training set accuracy of 84.70 % and a test set accuracy of 82.45 % with a loss of 0.3780 and 0.4178 respectively after 10 epochs. There is no major difference between the two losses which indicates that the model has not overfitted.

```
Epoch 10/10
391/391 [==============================] - 5s 13ms/step - loss: 0.3780 - acc: 0.8470
```

Figure 2 LSTM with 20 state dimensions Training Accuracy

```
782/782 [==============================] - 5s 6ms/step - loss: 0.4178 - acc: 0.8245
```

Figure 3 LSTM with 20 state dimensions Testing Accuracy

The state dimension of the LSTM was changed to 50 from 20 while retained all the other layers and setting from the LSTM model with 20 state dimensions. Doing so resulted in some results. Bothe the training and testing accuracy increased couple of percentages to that of the previous model. The models architecture can be seen in Figure 4 given below.

```
Model: "LSTM_with_state_dimension_50"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_15 (Embedding)     (None, 200, 300)          26574600
_________________________________________________________________
lstm_21 (LSTM)               (None, 50)                70200
_________________________________________________________________
flatten_11 (Flatten)         (None, 50)                0
_________________________________________________________________
dense_39 (Dense)             (None, 128)               6528
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_40 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_41 (Dense)             (None, 2)                 130
=================================================================
Total params: 26,659,714
Trainable params: 85,114
Non-trainable params: 26,574,600
```

Figure 4 LSTM with 50 state dimensions

The above model achieved a training accuracy of 84.44% and testing accuracy of 84.43% which increased from 82.45 %. There was no significant performance increase observed while increasing the state dimension. There was also no difference between the training and testing loss this indicating the absence of overfitting.

```
Epoch 10/10
391/391 [==============================] - 6s 14ms/step - loss: 0.3831 - acc: 0.8444
```

Figure 5 LSTM with 50 state dimensions Training Accuracy

```
782/782 [==============================] - 6s 7ms/step - loss: 0.3836 - acc: 0.8443
```

Figure 6 LSTM with 50 state dimensions Testing Accuracy

Changing the hidden dimension of the LSTM layer from 50 to 100 and keeping all the other hyperparameters such as dropout ratio, the depth of the layers and the l2 regularisation parameters same resulted in an increase in performance. The testing accuracy was at 84.72% a slight increase compared to the previous on and significant increase compared to the model with 20 state dimensions. The model's architecture is given in figure 8.

```
Epoch 10/10
391/391 [==============================] - 6s 16ms/step - loss: 0.3823 - acc: 0.8424
```

Figure 7 LSTM with 100 state dimensions Training Accuracy

```
782/782 [==============================] - 5s 6ms/step - loss: 0.3761 - acc: 0.8472
```

Figure 8 LSTM with 100 state dimensions Testing Accuracy

```
Model: "LSTM_with_state_dimension_100"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 200, 300)          26574600
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               160400
_________________________________________________________________
flatten_2 (Flatten)          (None, 100)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               12928
_________________________________________________________________
dense_7 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 130
=================================================================
Total params: 26,756,314
Trainable params: 181,714
Non-trainable params: 26,574,600
```

Figure 9 LSTM with 100 state dimensions

Further increasing just, the number of state dimension to 200 in LSTM layer and keeping all other hyperparameters resulted in a tiny increase in performance. The training and testing performance 85.18% and 84.81%. The difference between training and testing loss was also minimum indicating a perfect fit. But this was not the case when the number of hidden states was increased to 500. The testing accuracy did not increase anymore. The test accuracy actually reduced slightly to 84.27%. The architecture of both the models is given in figure 10.

```
Model: "LSTM_with_state_dimension_200"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 200, 300)          26574600
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               400800
_________________________________________________________________
flatten_3 (Flatten)          (None, 200)               0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               25728
_________________________________________________________________
dense_10 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_11 (Dense)             (None, 2)                 130
=================================================================
Total params: 27,009,514
Trainable params: 434,914
Non-trainable params: 26,574,600
```

```
Model: "LSTM_with_state_dimension_500"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_16 (Embedding)     (None, 200, 300)          26574600
_________________________________________________________________
lstm_24 (LSTM)               (None, 500)               1602000
_________________________________________________________________
flatten_16 (Flatten)         (None, 500)               0
_________________________________________________________________
dense_48 (Dense)             (None, 128)               64128
_________________________________________________________________
dense_49 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_50 (Dense)             (None, 2)                 130
=================================================================
Total params: 28,249,114
Trainable params: 1,674,514
Non-trainable params: 26,574,600
```

Figure 10 LSTM with 200 state dimensions(R)
and LSTM with 500 state dimensions(L)

Another LSTM layer was added to the model and both the LSTM layer's state dimension was set to 200 hundred. The sequence of the first LSTM layer is fed into the second one. The dropout ratio of both layers was set to 0.2 and l2 regularization was added to prevent overfitting. Building the neural network in the above mentioned way also did not result in a improvement in performance. The testing accuracy was similar to other models seen before. The architecture of the models is given in Figure 13.

```
Epoch 10/10
391/391 [==============================] - 16s 40ms/step - loss: 0.3903 - acc: 0.8436
```

Figure 11 Two LSTM layers with 200 state dimensions Training Accuracy

```
782/782 [==============================] - 9s 12ms/step - loss: 0.3953 - acc: 0.8442
```

Figure 12 Two LSTM layers with 200 state dimensions Testing Accuracy

```
Model: "LSTM_with_state_dimension_200"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_12 (Embedding)     (None, 200, 300)          26574600
_________________________________________________________________
lstm_19 (LSTM)               (None, 200, 200)          400800
_________________________________________________________________
lstm_20 (LSTM)               (None, 200)               320800
_________________________________________________________________
flatten_12 (Flatten)         (None, 200)               0
_________________________________________________________________
dense_36 (Dense)             (None, 128)               25728
_________________________________________________________________
dense_37 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_38 (Dense)             (None, 2)                 130
=================================================================
Total params: 27,330,314
Trainable params: 755,714
Non-trainable params: 26,574,600
```

Figure 13 Model with stacked LSTM with state dimension set to 200

From the results it can be seen that Long short-term memory works well in this task of classifying the positive and negative reviews from IMDB database. It was observed that after a certain number of state dimension there was no increase in the accuracy of the model when regularization was applied. Removing the regularization resulted in better training accuracy which increased with increase in the number of state dimensions but the testing accuracy did not increase more than 85% resulting in overfitting of the training data. Stacking two or more layers of LSTM also did not have any significant impact as the results were similar to that of models with single layer of LSTM.

Now let us see the performance of Vanilla RNN on the task of classification of positive and negative reviews from the IMDB Dataset.

2. Performance of Vanilla RNN on the IMDB dataset

The RNN model, with its state dimension set to 20, does poorly on the classification task. The first layer of the model was the embedding layer, similar to that of the LSTM model. The embeddings are then passed into the RNN layer. The Dropout with a ratio of 0.2 and l2 regularization with a regularization parameter of 0.001 is applied to this layer to prevent the model from overfitting. The output from the RNN is then passed into a feed-forward neural network that performs logistic regression to get the probabilities of the input being a positive or negative review. The architecture of the model can be seen in figure 14.

The model did not perform well and got only a training and testing accuracy of 65.33 % and 66.87%, respectively. Comparing to an LSTM with the same number of dimensions, the testing accuracy of RNN is approximately 20 percent less.

```
Model: "RNN_model_with_statedimension_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_18 (Embedding)     (None, 200, 300)          26574600
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 20)                6420
_________________________________________________________________
flatten_18 (Flatten)         (None, 20)                0
_________________________________________________________________
dense_54 (Dense)             (None, 128)               2688
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_55 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_56 (Dense)             (None, 2)                 130
=================================================================
Total params: 26,592,094
Trainable params: 17,494
Non-trainable params: 26,574,600
```

Figure 14 RNN with 20 state dimensions

```
Epoch 10/10
391/391 [==============================] - 5s 13ms/step - loss: 0.3780 - acc: 0.8470
```

Figure 15 RNN with 20 state dimensions Training accuracy

```
782/782 [==============================] - 15s 19ms/step - loss: 0.6122 - acc: 0.6687
```

Figure 16 RNN with 20 state dimensions Testing accuracy

Increasing the state dimension from 20 to 50 and retaining all the other settings like l2 regularization and dropout percentages, not much performance increase was observed. The performance of the RNN decreased. Testing accuracy of the model went from 66.87% to 64.28%. The model's architecture is given in figure 19.

```
Epoch 10/10
782/782 [==============================] - 131s 168ms/step - loss: 0.6317 - acc: 0.6592
```

Figure 17 RNN with 50 state dimensions Training accuracy

```
782/782 [==============================] - 14s 18ms/step - loss: 0.6428 - acc: 0.6428
```

Figure 18 RNN with 50 state dimensions Testing accuracy

```
Model: "RNN_with_state_dimension_50"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 200, 300)          26574600
_________________________________________________________________
simple_rnn_6 (SimpleRNN)     (None, 50)                17550
_________________________________________________________________
flatten_5 (Flatten)          (None, 50)                0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               6528
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_16 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_17 (Dense)             (None, 2)                 130
=================================================================
Total params: 26,607,064
Trainable params: 32,464
Non-trainable params: 26,574,600
```

Figure 19 RNN with 50 state dimensions

Further increasing the number of state dimensions did not result in any improvement in the results. Instead, as the number of state dimensions was increased to 100 from 50, the model could not learn. The training accuracy of the model did not increase with each iteration. Even if it increased for the initial few epochs, the loss starts to go up again, indicating the model's inability to learn. This is case with the models with the number of states was increased to 200 and 500. The RNN model cannot minimize the loss when the number of states is more than a certain number.

Different methods like adding additional RNN layers, trying to classify based on the sequences returned by the RNN layer and adding an average pooling layer was tried in the model shown in figure 20 but did not help the model to learn any better from the input data.

```
Model: "RNN_with_state_dimension_100"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 200, 300)          26574600
_________________________________________________________________
simple_rnn_7 (SimpleRNN)     (None, 200, 100)          40100
_________________________________________________________________
average_pooling1d_1 (Average (None, 100, 100)          0
_________________________________________________________________
simple_rnn_8 (SimpleRNN)     (None, 100)               20100
_________________________________________________________________
flatten_6 (Flatten)          (None, 100)               0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               12928
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_20 (Dense)             (None, 2)                 130
=================================================================
Total params: 26,656,114
Trainable params: 81,514
Non-trainable params: 26,574,600
```

Figure 20 RNN with 100 state dimensions

The results obtained from all the models are summarised in the table below.

Method	State dimension	No of Layers	Training Accuracy %	Testing Accuracy %
LSTM	20	1	84.70	82.45
LSTM	50	1	84.44	84.43
LSTM	100	1	84.24	84.72
LSTM	200	1	85.18	84.81
LSTM	500	1	85.49	84.27
LSTM	200	2	84.36	84.42
RNN	20	1	64.71	65.26
RNN	50	1	65.92	64.28
RNN	100	2	50.85	50.00
RNN	200	3	50.41	49.94
RNN	500	3	50.18	50.00

Table 1. Results

3. Conclusion:

Observing the models, the Long short-term memory networks, a modified version of a vanilla RNN network work outperform the RNN on the IMDB dataset classification task by a large margin. The LSTM achieved maximum accuracy of 84.81 without overfitting, but an RNN network could only get an accuracy of 65.26. This big difference in performance is because the vanilla version of the RNN suffers from gradient disappearing and gradient exploding, which makes it nearly impossible for RNNs to traverse loss when the state dimension is huge. After a certain number of states, The RNN is not able to perform well. To conclude, LSTM performs better than Vanilla RNN in tasks where the sequence length in consideration is a large number, which the RNN fails to solve correctly.