MSDS 422 Assignment 8
Husein Adenwala

Introduction

The purpose of this assignment to use Recurrent Neural Network (RNN) architecture was to perform sentiment classification of the IMDB review. The classification will be binary, i.e. positive (1) or negative (2).

Data preparation, exploration, visualization

The IMDB dataset was imported using the dataset module in keras. For this assignment, I used only the label dataset. The dataset contained 50,000 label data/reviews with the labels being positive(1) or negative(0). The label dataset was evenly divided between train and test respectively (25,000 each).

```
total_num_examples=100000,
splits={
    'test': 25000,
    'train': 25000,
    'unsupervised': 50000,
```

Data review

The data had already been preprocessed so that the reviews (sequences of words) have been converted to sequences of integers, where each integer represents a specific word in a dictionary.

The dataset info includes a text encoder tfds.features.text.SubwordTextEncoder and the encoder vocabulary size is:

```
Vocabulary size: 8185
```

This encoder can encode/decode any string.

Original Encoded data

```
(<tf.Tensor: shape=(163,), dtype=int64, numpy=
 array([  62,   18,   41,  604,  927,   65,    3,  644, 7968,   21,   35,
        5096,   36,   11,   43, 2948, 5240,  102,   50,  681, 7862, 1244,
           3, 3266,   29,  122,  640,    2,   26,   14,  279,  438,   35,
          79,  349,  384,   11, 1991,    3,  492,   79,  122,  188,  117,
          33, 4047, 4531,   14,   65, 7968,    8, 1819, 3947,    3,   62,
          27,    9,   41,  577, 5044, 2629, 2552, 7193, 7961, 3642,    3,
          19,  107, 3903,  225,   85,  198,   72,    1, 1512,  738, 2347,
         102, 6245,    8,   85,  308,   79, 6936, 7961,   23, 4981, 8044,
           3, 6429, 7961, 1141, 1335, 1848, 4848,   55, 3601, 4217, 8050,
           2,    5,   59, 3831, 1484, 8040, 7974,  174, 5773,   22, 5240,
         102,   18,  247,   26,    4, 3903, 1612, 3902,  291,   11,    4,
          27,   13,   18, 4092, 4008, 7961,    6,  119,  213, 2774,    3,
          12,  258, 2306,   13,   91,   29,  171,   52,  229,    2, 1245,
        5790,  995, 7968,    8,   52, 2948, 5240, 8039, 7968,    8,   74,
        1249,    3,   12,  117, 2438, 1369,  192,   39, 7975])>,
 <tf.Tensor: shape=(), dtype=int64, numpy=0>)
```

Decoded data

```
'This was an absolutely terrible movie. Don't be lured in by Christopher Walken or Michael Ironside. Both are great actors, but this must simply be their worst role in history. Even their great acting could not redeem this movie's ridiculous storyline. This movie is an early n
ineties US propaganda piece. The most pathetic scenes were those when the Columbian rebels were making their cases for revolutions. Maria Conchita Alonso appeared phony, and her pseudo-love affair with Walken was nothing but a pathetic emotional plug in a movie that was devoid
of any real meaning. I am disappointed that there are movies like this, ruining actor's like Christopher Walken's good name. I could barely sit through it.'
```

Encoded text and label in a data frame.

	label	text
0	1	[173, 29, 185, 13, 115, 1956, 8044, 3, 398, 12...
1	1	[133, 2237, 64, 1229, 1795, 6, 4, 615, 7974, 1...
2	0	[3567, 47, 3634, 7978, 7974, 7981, 2, 7998, 58...
3	0	[7128, 127, 2196, 2185, 1098, 1040, 687, 1145,...
4	1	[249, 4, 2256, 3293, 453, 3483, 7961, 5014, 12...

Data Transformation: Padding

The data was padded with trailing zeros padded to the maximum size of that dimension in each batch and the batch size was set to 64.

This is shown below:

```
array([[  12,  359,    4, ...,    0,    0,    0],
       [  12,   31,  165, ...,    0,    0,    0],
       [6177, 7961,    9, ...,    0,    0,    0],
       ...,
       [  62,  180,   35, ...,    0,    0,    0],
       [5243, 8002,  903, ...,    0,    0,    0],
       [ 173,    9, 3136, ...,    0,    0,    0]]), array([1, 1, 1, 0, 1, 1, 1, 1, 1, 0
       1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1]))
```

Review research design and modeling methods

For this analysis, I used the Recurring Neural Network (RNN) and LSTM RNN model to classify the sentiment of IMDB reviews. RNNs are ideal for solving problems where the sequence is more important than the individual items themselves. LSTM stands for Long short-term memory and is a special kind of Neural Network called Recurrent Neural Network. LSTM's relative insensitivity to gap length is an advantage over RNNs. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs.

I started with the base model of RNN without regularization (Model 1) and compared it with LSTM (model 2) with regularization.

I used regularization techniques of dropout and gradient clipping (clipnorm) to reduce overfitting in model 2.

Review results, evaluate models

All models included the following parameters:

Optimizer: Adam

Epoch: 10

Activation function: Relu (hidden layer), Sigmoid (output layer),

Regularization: dropout, and clipnorm (model 2 only).

Model 1: RNN- No Regularization

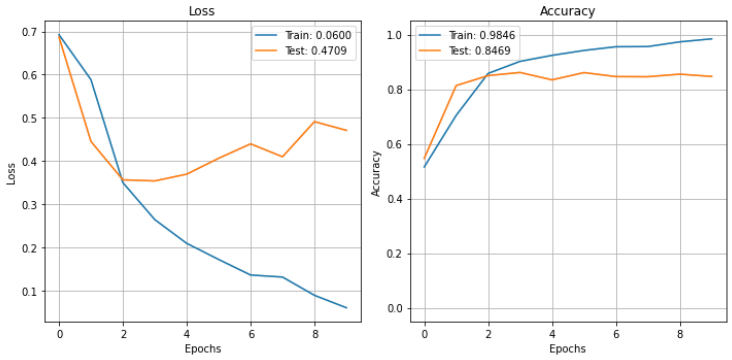
I used this as the base model. First I created an embedding layer of 64 for the text corpus of 8150 which was then connected to layer of 64 bi direction gates (for a total of 128) which was connected to a dense layer of 64 neurons with a Relu activation function which connected to the output layers with one neuron which applied the sigmoid activation function.

Time to process: 1h 16min 8s

Train: `loss: 0.0489 - accuracy: 0.9916`

Test: `loss: 0.4870 - accuracy: 0.8363`

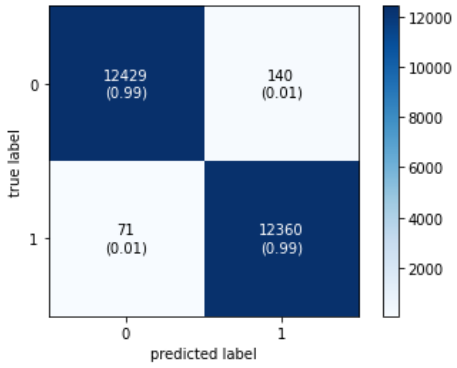
The graph clearly shows overfitting.




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 64)          523840
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               16512
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 548,673
Trainable params: 548,673
Non-trainable params: 0
_________________________________________________________________
```

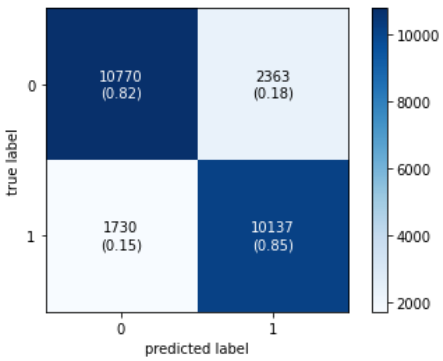
The confusion matrix and the performance metrics shows that the model has very high accuracy on training data but not on the test data which suggests overfitting, but accuracy of true positive and true negative is balanced.

Train



	precision	recall	f1-score	support
0	0.99	0.99	0.99	12569
1	0.99	0.99	0.99	12431
accuracy			0.99	25000
macro avg	0.99	0.99	0.99	25000
weighted avg	0.99	0.99	0.99	25000

Test



	precision	recall	f1-score	support
0	0.86	0.82	0.84	13133
1	0.81	0.85	0.83	11867
accuracy			0.84	25000
macro avg	0.84	0.84	0.84	25000
weighted avg	0.84	0.84	0.84	25000

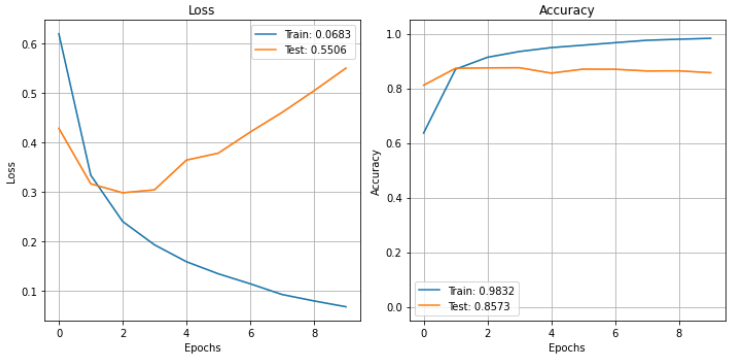
Model 2: LSTM-Regularization

Similar to the base model where I first created an embedding layer of 64 dimensions for the text corpus of 8150, which was then connected to layers of 64 bi direction gates (128 total) and then to a layer of 32 bidirectional layer (64), which was connected to a dense layer of 64 neurons with a dropout of 0.5 with a Relu activation function, which then connected to the output layers with one neuron which applied the sigmoid activation function.

Train: `loss: 0.0486 - accuracy: 0.9889`

Test: `loss: 0.5509 - accuracy: 0.8572`

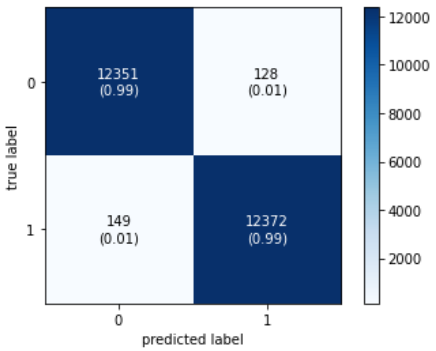
The graph clearly shows overfitting but the difference in accuracy between test and train is slightly less than model 1




```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 64)          523840
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 128)         66048
_________________________________________________________________
bidirectional_2 (Bidirection (None, 64)                41216
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 635,329
Trainable params: 635,329
Non-trainable params: 0
_________________________________________________________________
```

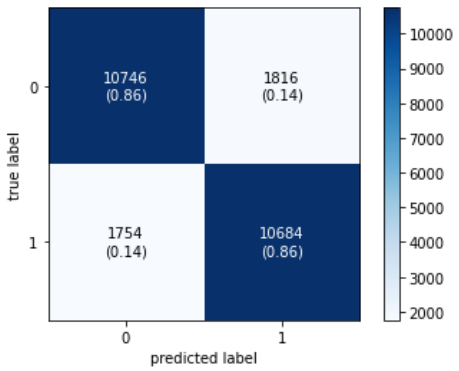
Like the model above, The confusion matrix and the performance metrics shows that the model has very high accuracy on training data but not on the test data which suggests overfitting , thought the difference between test and train is less compared to model 1.

Train



	precision	recall	f1-score	support
0	0.99	0.99	0.99	12479
1	0.99	0.99	0.99	12521
accuracy			0.99	25000
macro avg	0.99	0.99	0.99	25000
weighted avg	0.99	0.99	0.99	25000

Test



	precision	recall	f1-score	support
0	0.86	0.86	0.86	12562
1	0.85	0.86	0.86	12438
accuracy			0.86	25000
macro avg	0.86	0.86	0.86	25000
weighted avg	0.86	0.86	0.86	25000

Summary of the 2 models

Overall, Model 2 with LSTM performed better based on the test accuracy, but not significantly better than expected. The Regularization technique of dropout and gradient clipping created marginal improvement in the model.

	Bidirectional layer	Number of dense layer	Training Accuracy	Test Accuracy
Model 1 RNN	1	1	99.16	83.63
Model 2 LSTM	2	1	98.89	85.72

Implementation and programming

This assignment was done using google colab utilizing Tensoflow Keras packages, NumPy, scikit-learn and pandas.

I loaded the IMDB preprocessed data (encoded) from keras dataset. I created a test and train dataset of 25000 reviews in each.

I used nmpy and python dictionary function and keras encoded to review the data.

Using the model method from Kera's, I created all 2 model objects, which included the model architecture. In that architecture, I used dropout (0.2) regularization method on the hidden layer/layers from Keras package.

Then I used the compile method to select the algorithm and optimizer, gradient clipping and metric for the model.

I then fitted the model using 10 epochs using the training data and test data for validation. The model fit was saved in object (history).

I used the model evaluate method to get the accuracy and loss on training and test data.

I used the Keras plot history function on the history of the model to plot the loss and accuracy plots for each epoch on the train and test/validation data.

I used the confusion matrix function from scikit-learn and created a confusion matrix for test and train data for both models.

I used the scikit-learn classification report to get the accuracy, recall and precision statistics for test and train data for both models.

Exposition, problem description, and management recommendations

Based on my analysis above, Management should utilize the RNN/LSTM architecture for the sentiment classification problem--I would recommend Model 2- LSTM architecture. This model performed best based on test accuracy and has a smaller difference between test and train accuracy. However, both models were overfitting and regularization did not make any significant difference in reducing the overfitting problem.

Looking at the accuracy and loss graph we do not see any effect of vanishing or exploding gradient.

The LSTM model adds complexity to the model and requires more processing time; however, it generally performs better in NLP or Time series problem. Model 2 has the potential to improve performance by testing different padding strategies and adjusting the network architecture.

Appendix

Code for model 1 and 2

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64,mask_zero=True),
    tf.keras.layers.Bidirectional(tf.keras.layers.SimpleRNN(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation=tf.nn.sigmoid)
])
```

```
model2 = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64, mask_zero= True),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64,  return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
model2.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(1e-4, clipnorm=1.),
              metrics=['accuracy'])
```

Code to create dataframe of the train data set

```
df_2train = tfds.as_dataframe(dataset['train'], info)
```