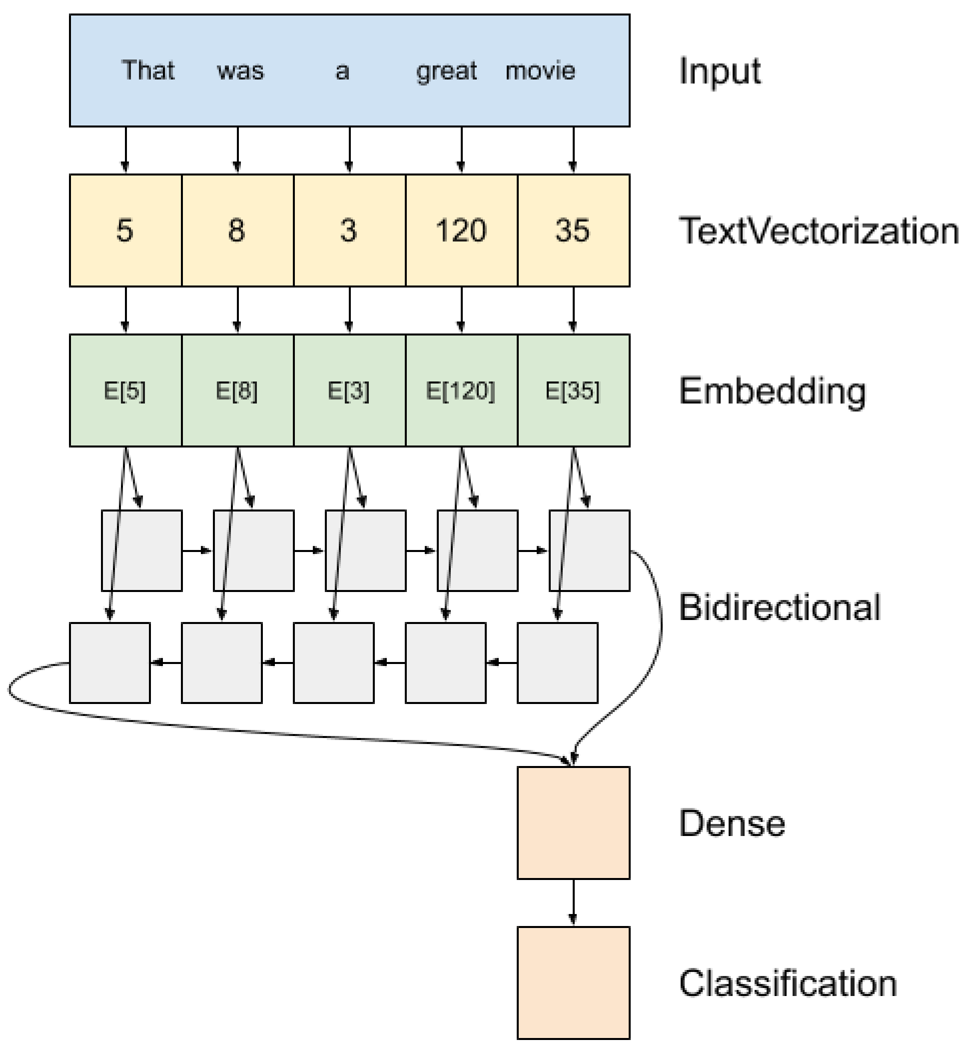
Jaynil Gaglani 2017140013
Yash Gandhi 2017140014
Shubham Gogate 2017140015

Aim:
To choose a suitable deep learning technique for a given scenario

Problem Statement:
Text classification using recurrent neural networks on the IMDB large movie review dataset for sentiment analysis.

Tool/Language:
Programming language: Python

Theory:



1. The model is built as a tf.keras.Sequential.
2. The first layer is the encoder, which converts the text to a sequence of token indices.
3. After the encoder is an embedding layer. An embedding layer stores one vector per word. When called, it converts the sequences of word indices to sequences of vectors. These vectors are trainable. After training (on enough data), words with similar meanings often have similar vectors.
   This index-lookup is much more efficient than the equivalent operation of passing a one-hot encoded vector through a tf.keras.layers.Dense layer.
4. A recurrent neural network (RNN) processes sequence input by iterating through the elements. RNNs pass the outputs from one timestep to their input on the next timestep.
   The tf.keras.layers.The bidirectional wrapper can also be used with an RNN layer. This propagates the input forward and backwards through the RNN layer and then concatenates the final output.
   - The main advantage to a bidirectional RNN is that the signal from the beginning of the input doesn't need to be processed all the way through every timestep to affect the output.
   - The main disadvantage of a bidirectional RNN is that you can't efficiently stream predictions as words are being added to the end.
5. After the RNN has converted the sequence to a single vector the two layers. Dense do some final processing and convert from this vector representation to a single logit as the classification output.

Code:

Text classification with an RNN

```
[1] !pip install -q tensorflow_datasets
```

```
[2] import numpy as np

    import tensorflow_datasets as tfds
    import tensorflow as tf

    tfds.disable_progress_bar()
```

Import `matplotlib` and create a helper function to plot graphs:

```
[3] import matplotlib.pyplot as plt

    def plot_graphs(history, metric):
      plt.plot(history.history[metric])
      plt.plot(history.history['val_'+metric], '')
      plt.xlabel("Epochs")
      plt.ylabel(metric)
      plt.legend([metric, 'val_'+metric])
```

Experiment 8: Choose a deep learning technique

The IMDB large movie review dataset is a *binary classification* dataset—all the reviews have either a *positive* or *negative* sentiment.

```
[5]  dataset, info = tfds.load('imdb_reviews', with_info=True,
                               as_supervised=True)
     train_dataset, test_dataset = dataset['train'], dataset['test']

     train_dataset.element_spec
```

```
(TensorSpec(shape=(), dtype=tf.string, name=None),
 TensorSpec(shape=(), dtype=tf.int64, name=None))
```

Initially this returns a dataset of (text, label pairs):

```
[6]  for example, label in train_dataset.take(1):
       print('text: ', example.numpy())
       print('label: ', label.numpy())
```

```
text:  b"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
label:  0
```

Next shuffle the data for training and create batches of these `(text, label)` pairs:

```
[7]  BUFFER_SIZE = 10000
     BATCH_SIZE = 64
```

```
[8]  train_dataset = train_dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
     test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(tf.data.experimental.AUTOTUNE)
```

```
[10] for example, label in train_dataset.take(1):
       print('texts: ', example.numpy()[:3])
       print()
       print('labels: ', label.numpy()[:3])
```

```
texts:  [b'The fight scenes play like slow-motion Jackie Chan and the attempts at wit are pathetic (worst pun by far: "Guess what? This time I heard you coming").
 b"Serum is about a crazy doctor that finds a serum that is supposed to cure all diseases through the power of the mind. Instead it creates some kind of monster t
 b'The buzz for this film has always been about the fabulous graphics that make Kevin Bacon disappear. Sadly, they stopped there. They should have continued to ma

labels:  [0 0 0]
```

Create the text encoder

The raw text loaded by `tfds` needs to be processed before it can be used in a model. The way to process text for training is using the `experimental.preprocessing.TextVectorization` layer.

Create the layer, and pass the dataset's text to the layer's `.adapt` method:

```
VOCAB_SIZE=1000
encoder = tf.keras.layers.experimental.preprocessing.TextVectorization(
    max_tokens=VOCAB_SIZE)
encoder.adapt(train_dataset.map(lambda text, label: text))
```

The `.adapt` method sets the layer's vocabulary. Here are the first 20 tokens. After the padding and unknown tokens they're sorted by frequency:

```
[12] vocab = np.array(encoder.get_vocabulary())
     vocab[:20]
```

```
array(['', '[UNK]', 'the', 'and', 'a', 'of', 'to', 'is', 'in', 'it', 'i',
       'this', 'that', 'br', 'was', 'as', 'for', 'with', 'movie', 'but'],
      dtype='<U14')
```

Once the vocabulary is set, the layer can encode text into indices. The tensors of indices are 0-padded to the longest sequence in the batch

```
[13] encoded_example = encoder(example)[:3].numpy()
     encoded_example
```

```
array([[  2, 537, 137, ...,   0,   0,   0],
       [  1,   7,  43, ...,   0,   0,   0],
       [  2,   1,  16, ...,   0,   0,   0]])
```

```
for n in range(3):
  print("Original: ", example[n].numpy())
  print("Round-trip: ", " ".join(vocab[encoded_example[n]]))
  print()
```

```
Original:  b'The fight scenes play like slow-motion Jackie Chan and the attempts at wit are pathetic (worst pun by far: "Guess what? This time I heard you coming'
Round-trip:  the fight scenes play like [UNK] [UNK] [UNK] and the attempts at [UNK] are [UNK] worst [UNK] by far guess what this time i heard you coming the stars

Original:  b"Serum is about a crazy doctor that finds a serum that is supposed to cure all diseases through the power of the mind. Instead it creates some kind of
Round-trip:  [UNK] is about a crazy [UNK] that finds a [UNK] that is supposed to [UNK] all [UNK] through the power of the mind instead it [UNK] some kind of monst

Original:  b'The buzz for this film has always been about the fabulous graphics that make Kevin Bacon disappear. Sadly, they stopped there. They should have conti
Round-trip:  the [UNK] for this film has always been about the [UNK] [UNK] that make [UNK] [UNK] [UNK] [UNK] they [UNK] there they should have [UNK] to make the s
```

Model:

```
[15] model = tf.keras.Sequential([
         encoder,
         tf.keras.layers.Embedding(
             input_dim=len(encoder.get_vocabulary()),
             output_dim=64,
             # Use masking to handle the variable sequence lengths
             mask_zero=True),
         tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
         tf.keras.layers.Dense(64, activation='relu'),
         tf.keras.layers.Dense(1)
     ])
```

The embedding layer uses masking to handle the varying sequence-lengths. All the layers after the `Embedding` support masking:

```
[16] print([layer.supports_masking for layer in model.layers])

     [False, True, True, True, True]
```

```
[17] # predict on a sample text without padding.

     sample_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
     predictions = model.predict(np.array([sample_text]))
     print(predictions[0])

     [-0.00081357]
```

Now, evaluate it again in a batch with a longer sentence. The result should be identical:

```
[18] # predict on a sample text with padding

     padding = "the " * 2000
     predictions = model.predict(np.array([sample_text, padding]))
     print(predictions[0])

     [-0.00081357]
```

Compile the Keras model to configure the training process:

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

Train the model

```
history = model.fit(train_dataset, epochs=10,
                    validation_data=test_dataset,
                    validation_steps=30)
```

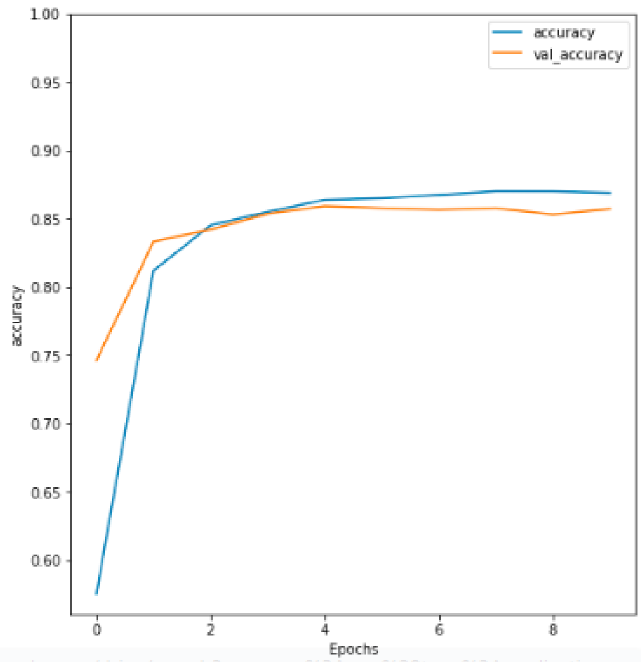
```
Epoch 1/10
391/391 [==============================] - 41s 104ms/step - loss: 0.6343 - accuracy: 0.5754 - val_loss: 0.4760 - val_accuracy: 0.7464
Epoch 2/10
391/391 [==============================] - 40s 102ms/step - loss: 0.4198 - accuracy: 0.8121 - val_loss: 0.3766 - val_accuracy: 0.8333
Epoch 3/10
391/391 [==============================] - 40s 103ms/step - loss: 0.3490 - accuracy: 0.8453 - val_loss: 0.3456 - val_accuracy: 0.8422
Epoch 4/10
391/391 [==============================] - 40s 102ms/step - loss: 0.3314 - accuracy: 0.8551 - val_loss: 0.3317 - val_accuracy: 0.8536
Epoch 5/10
391/391 [==============================] - 40s 103ms/step - loss: 0.3136 - accuracy: 0.8639 - val_loss: 0.3255 - val_accuracy: 0.8594
Epoch 6/10
391/391 [==============================] - 40s 101ms/step - loss: 0.3081 - accuracy: 0.8652 - val_loss: 0.3243 - val_accuracy: 0.8578
Epoch 7/10
391/391 [==============================] - 39s 101ms/step - loss: 0.3053 - accuracy: 0.8674 - val_loss: 0.3234 - val_accuracy: 0.8568
Epoch 8/10
391/391 [==============================] - 39s 101ms/step - loss: 0.3028 - accuracy: 0.8702 - val_loss: 0.3235 - val_accuracy: 0.8578
Epoch 9/10
391/391 [==============================] - 40s 101ms/step - loss: 0.3007 - accuracy: 0.8701 - val_loss: 0.3206 - val_accuracy: 0.8531
Epoch 10/10
391/391 [==============================] - 39s 100ms/step - loss: 0.2996 - accuracy: 0.8690 - val_loss: 0.3183 - val_accuracy: 0.8573
```

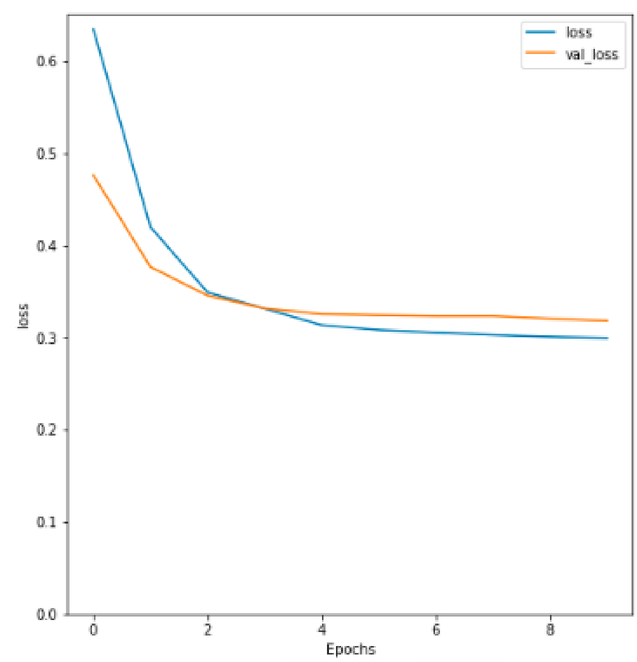
Graphs and Results:

```
[22] test_loss, test_acc = model.evaluate(test_dataset)

     print('Test Loss: {}'.format(test_loss))
     print('Test Accuracy: {}'.format(test_acc))
```

```
391/391 [==============================] - 17s 43ms/step - loss: 0.3150 - accuracy: 0.8582
Test Loss: 0.31495431065559387
Test Accuracy: 0.8582000136375427
```

```
[23] plt.figure(figsize=(16,8))
     plt.subplot(1,2,1)
     plot_graphs(history, 'accuracy')
     plt.ylim(None,1)
     plt.subplot(1,2,2)
     plot_graphs(history, 'loss')
     plt.ylim(0,None)

     (0.0, 0.6510409474372864)
```





If the prediction is >= 0.0, it is positive else it is negative.

```
sample_text = ('The movie was cool. The animation and the graphics '
               'were out of this world. I would recommend this movie.')
predictions = model.predict(np.array([sample_text]))
print(predictions)

[[0.7695097]]
```

Stack two or more LSTM layers

```
[26] model = tf.keras.Sequential([
         encoder,
         tf.keras.layers.Embedding(len(encoder.get_vocabulary()), 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)
     ])
```

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

```
[28] history = model.fit(train_dataset, epochs=10,
                         validation_data=test_dataset,
                         validation_steps=30)
```

```
Epoch 1/10
391/391 [==============================] - 75s 192ms/step - loss: 0.6229 - accuracy: 0.5958 - val_loss: 0.4537 - val_accuracy: 0.7870
Epoch 2/10
391/391 [==============================] - 73s 185ms/step - loss: 0.4000 - accuracy: 0.8246 - val_loss: 0.3550 - val_accuracy: 0.8396
Epoch 3/10
391/391 [==============================] - 72s 184ms/step - loss: 0.3462 - accuracy: 0.8510 - val_loss: 0.3353 - val_accuracy: 0.8432
Epoch 4/10
391/391 [==============================] - 72s 184ms/step - loss: 0.3278 - accuracy: 0.8603 - val_loss: 0.3220 - val_accuracy: 0.8547
Epoch 5/10
391/391 [==============================] - 71s 183ms/step - loss: 0.3160 - accuracy: 0.8659 - val_loss: 0.3345 - val_accuracy: 0.8547
Epoch 6/10
391/391 [==============================] - 72s 185ms/step - loss: 0.3061 - accuracy: 0.8689 - val_loss: 0.3236 - val_accuracy: 0.8484
Epoch 7/10
391/391 [==============================] - 72s 184ms/step - loss: 0.3059 - accuracy: 0.8694 - val_loss: 0.3181 - val_accuracy: 0.8609
Epoch 8/10
391/391 [==============================] - 72s 184ms/step - loss: 0.3056 - accuracy: 0.8704 - val_loss: 0.3231 - val_accuracy: 0.8474
Epoch 9/10
391/391 [==============================] - 72s 183ms/step - loss: 0.3012 - accuracy: 0.8732 - val_loss: 0.3162 - val_accuracy: 0.8547
Epoch 10/10
391/391 [==============================] - 72s 185ms/step - loss: 0.2965 - accuracy: 0.8745 - val_loss: 0.3261 - val_accuracy: 0.8599
```

Graphs and Results:

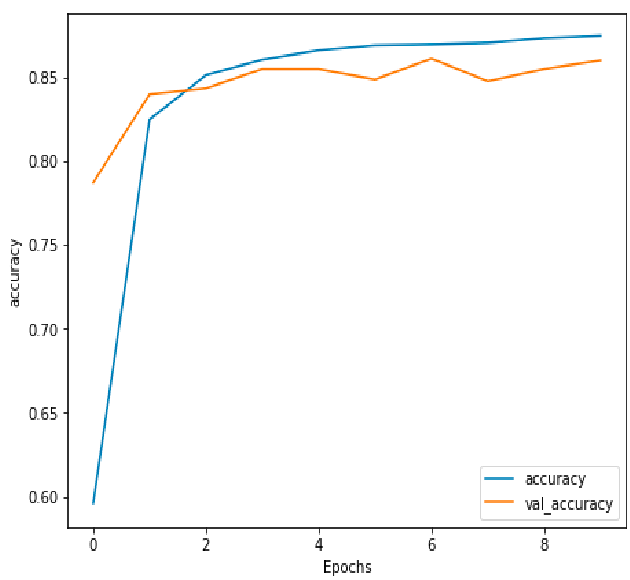
```
[29] test_loss, test_acc = model.evaluate(test_dataset)

     print('Test Loss: {}'.format(test_loss))
     print('Test Accuracy: {}'.format(test_acc))
```

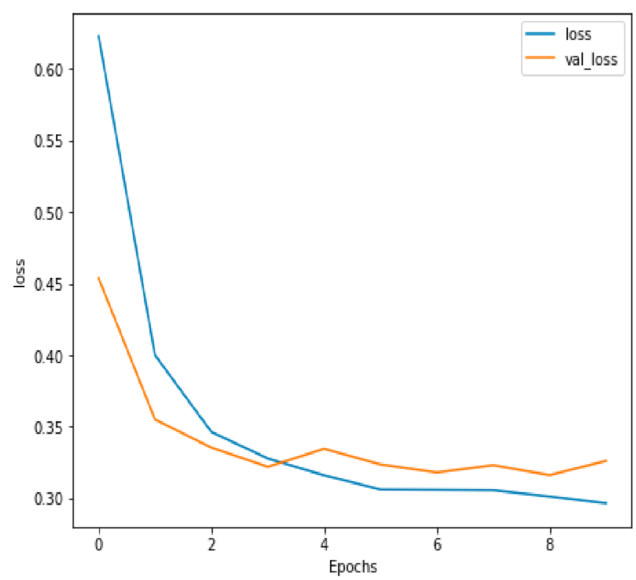
```
391/391 [==============================] - 31s 80ms/step - loss: 0.3298 - accuracy: 0.8634
Test Loss: 0.32984232902526855
Test Accuracy: 0.8634399771690369
```

```
[30] # predict on a sample text without padding.

     sample_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
     predictions = model.predict(np.array([sample_text]))
     print(predictions)
```

```
[[-1.7364554]]
```

```
[31] plt.figure(figsize=(16,6))
     plt.subplot(1,2,1)
     plot_graphs(history, 'accuracy')
     plt.subplot(1,2,2)
     plot_graphs(history, 'loss')
```





Conclusion:

	Simple RNN Model	Stacked with two or more LSTM
Test Accuracy	0.85	0.8634

Text classification is performed on IMDB dataset using RNN. First, we created the text encoder using Keras library to convert the text into mathematical vectors. In the next step, a simple bidirectional RNN model is trained on the dataset for classification and accuracy of the model is measured. Similarly, we stacked two or more LSTM models for the same task and an improvement in the classification accuracy can be seen from the above table.