INTERNSHIP: PROJECT REPORT

Name Of The Student	Likhitha M
Internship Project Topic	RIO Internship Project Variant(RIO 45) Automate detection of different emotions from textual comments and feedback.
Name of the Organization	TCS iON
Name of the Industry Mentor	Mr. Debashis Roy
Name of the Institute	Prepinsta Technologies

Start Date	End Date	Total Effort(hrs)	Project Environment	Tools used
20-10-2021	30-10-2021	13	Google Collab Chrome, Windows 10	Python 3 (Tensorflow)

Project Synopsis :

Since the birth of Artificial Intelligence in 1950 and its rebirth in the 20th century, it has contributed significantly toproviding effective solutions to major human and societal problems under various fields including natural language processing(NLP), which employs computational and linguistics techniques to aid computers understand and sometimesgenerate human languages in the form of texts and speech/voice.The state of being emotional is often aligned with making conscious arousal of feelings subjectively or with influencefrom the environment, thus emotions such as happiness, sadness, fear, anger, surprise, and so on are derived from thepersonal (subjective) experiences of individuals and as well as their interactions with their surroundings (audio/visualsignals) Emotions play vital roles in the existence or the complete make-up of individuals. They provide observers with
information regarding our current state and well-being.

Solution Approach :

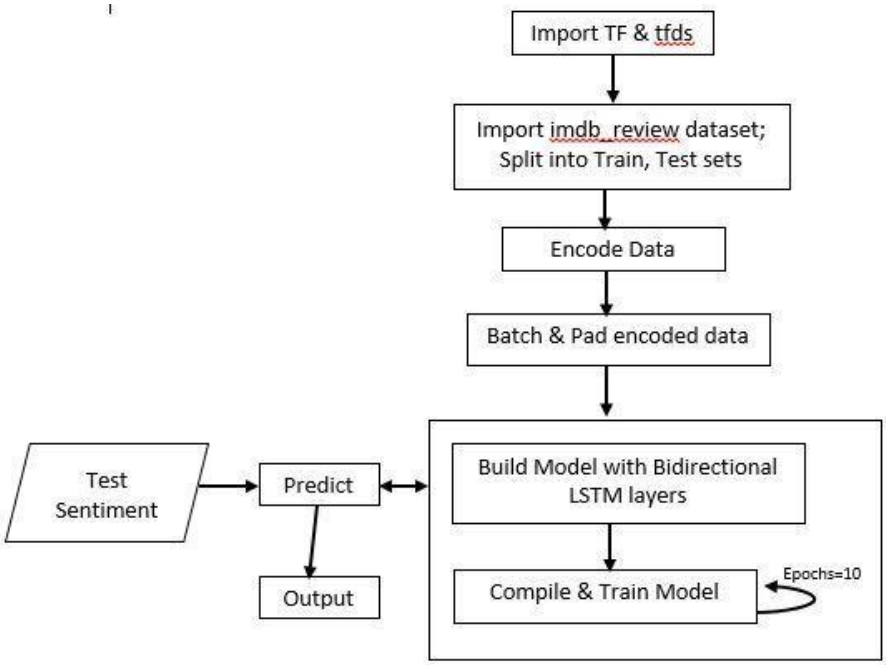
Deep Learning Algorithms such as Recurrent Neural Network using the Long Short TermMemory (LSTM)Can be used for a task. This model is used to perform sentiment analysis on movie reviews from the Large Movie Review Dataset, sometimes known as the IMDB dataset. The text encoder can be used to break down the sentiments into numerical values. On supplying test sentiments to the model, on the outcome it will try to predict whether it is a positive, negative or neutral feedback!

Assumption:

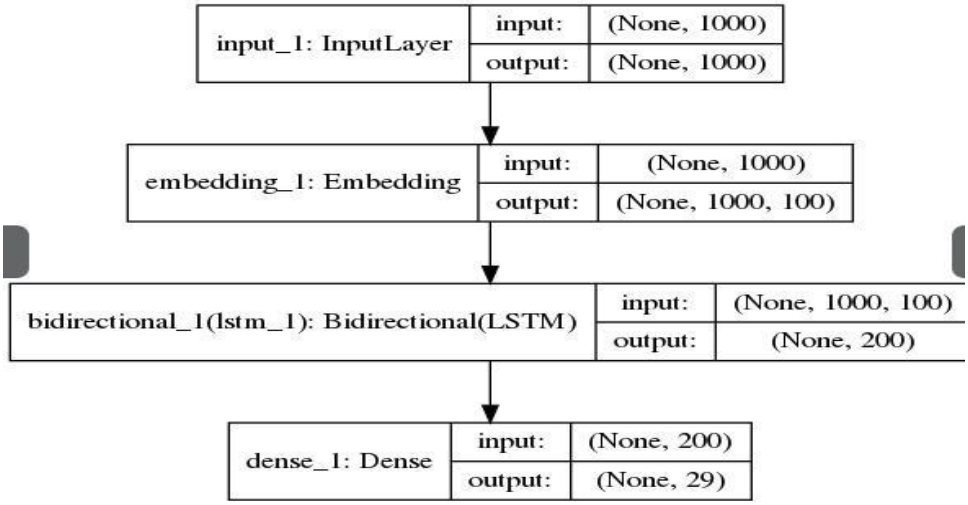
IMDB dataset that contains the text of 50,000 movie reviews from the Internet Movie Database. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are balanced, meaning they contain an equal number of positive and negative reviews So, its a binary or two-class-classification without 'neutral' label. However, This model does not mask the padding applied to the sequences. The Model will predict the sentiments giving a prediction score. If the prediction is >= 0.5, it is positive. On testing the model, the prediction value of negative sentiments can go up to -4. Therefore, it is assumed that for neutral sentiments (positive-negative combination), prediction values in the range -1 to +0.5 can be assumed (this also depends on accuracyand length of the test sentiment). Also, the dataset imported is already pre-processed, and is directly splitted in training & test sets by tensorflow_datasets library. This library also provides an encoder which breaks down a given string into a list of numerical values.

Project Diagrams:

Project Work Flow:



LSTM Model Flow-Chart Example:



Algorithm :

Recurrent Neural Network (RNN) using multi-layer Bidirectional Long Short Term Memory (LSTM) is the algorithm used.

Outcome:

For the Following Statement:

Season 4 - Stuck in a Rut Script
westsideschl1 January 2021
1. Overacted like reciting highly scripted very fake dialogue,
and all in sequence as if after every short line the director pauses to allow the actors to memorize the ne xt line.
And, of course, practice a devilish smile.
2. Over reliance on romance issues with the usual eye candy actors.
Lucifer, of course, has short beard growth (very manly fake) as if the devil really needs that along with m uscles (shown frequently).
Why for someone with God like super powers. To showcase the body we have a very fake (and overused) holding up a car as it's wheels spin in trying to get away.
3. The usual 10 seconds of staged dead body scene (very cheaply popular in crime series); the usual inter departmental detective conflicts & issues. 4. Special effects consist mostly of two cheap (thankfully alwa ys brief) film items: Colored eyes (seen in all superpower movies),
and wings that are so cheap they would fall apart if moved. 5. Hell looked like a studio CGI quickie: A ba ckground of trees; above are clouds;
the throne is a jagged rock nestled amongst other fake jagged rocks. Huh! Not the Hell I know.
Anyway, gotta have a King to sit there to stop (contain) the demons of Hell from coming to Earth.
Please, send the demons, they would be a lot more interesting than this script.
The output is : NEGATIVE

Whereas for:
Lucifer is a great show, a lot of it is the charming and amusing devil we all love to see.
The strength of this show is the balance it strikes between the detective work and the biblical/divine ele ments.
The greatest punch is delivered when one drives the other.
Lucifer made a strong start and I think the first seasons were it's best.
Low points include too much of the detective work and less biblical elements, it then resembles CSI mor e and more.
You don't want another detective show. Less time should also be devoted to side characters like Maze.
They are often used as space fillers for no good reason, characters should appear when needed. Overall not too bad. I hope there are more episodes.
The output is :NEGATIVE

Here,
sorryyyyyy tmr onwards the show's gonna end

The output is : Neutral

Detailed Presentation with Proof of Reasonable Accuracy:

We need to create batches of training data for your model. The reviews are all different lengths, so use padded_batch to zero pad the sequences while batching. Each batch will have a shape of (batch_size, sequence_length) because the padding is dynamic each batch will have a different length. Therefore we initialize BUFFER_SIZE = 10000 & BATCH_SIZE = 64. The training set is shuffled on every code execution, delivering variation in final accuracy.

MODEL: *tf.keras.Sequential*model (6 Layers used)

We selected Keras sequential model here since all the layers in the model only have single input and produce single output.

1. The first layer 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. The resulting dimensions are: (batch, sequence,embedding).

2. The *tf.keras.layers.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 output. This helps the RNN to learn long range dependencies. Two Bidirectional Layers are used in thismodel.

3. 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. A recurrent neural network (RNN) processes sequence input by iterating through the elements. RNNs pass the outputs from one timestep to their input—and then to thenext.

4. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/(1 - rate) such that the sum over all inputs isunchanged.

5. This fixed-length output vector is piped through a fully-connected (Dense) layer with 64 hidden units. The last layer is densely connected with a single output node. This uses the default linear activation function that outputs logits for numerical stability. Another option is to use the sigmoid activation function that returns a float value between 0 and 1, representing a probability, or confidencelevel.

The number of layers used in the model along with the hidden units count in maintaining good accuracy.

COMPILING THE MODEL:

The model needs a loss function and an optimizer for training. Since this is a binary classification problem and the model outputs logits (a single-unit layer with a linear activation), we'll use the *binary_crossentropy*loss function, which is better for dealing with probabilities- it measures the "distance" between probability distributions, or in our case, between the ground-truth distribution and the predictions.

Considering all the above parameters, the final step- when training the model, the number of iterations over entire training set (epochs) with respect to loss function impacts the final accuracy! *model.fit*result with *epochs = 10* gives the following Accuracy:

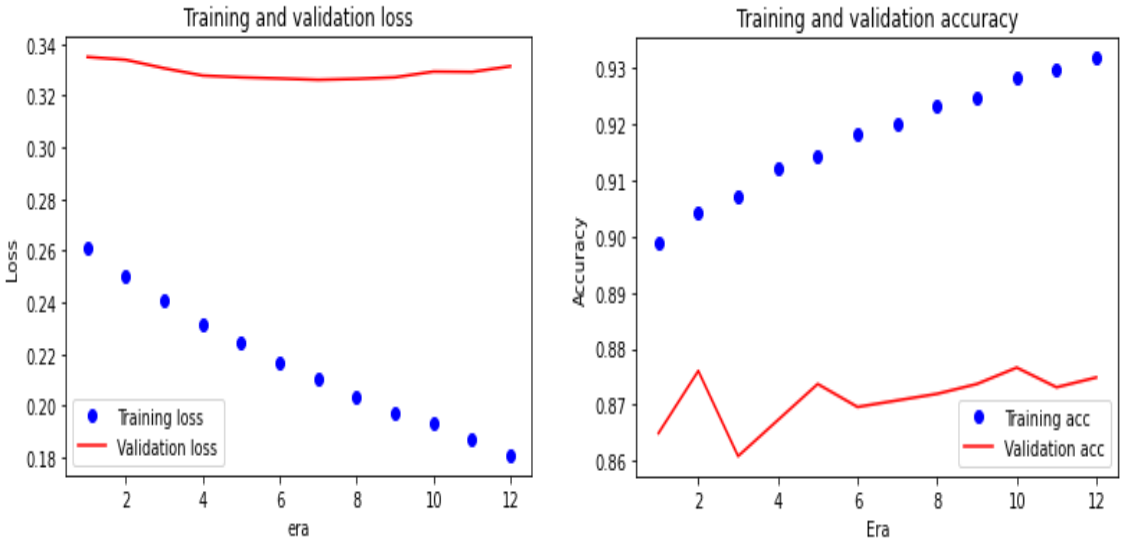
```
Epoch 1/12
658/658 [==============================] - 7s 10ms/step - loss: 0.2611 - accuracy: 0.8988 -
val_loss: 0.3350 - val_accuracy: 0.8649
Epoch 2/12
658/658 [==============================] - 7s 10ms/step - loss: 0.2502 - accuracy: 0.9042 -
val_loss: 0.3338 - val_accuracy: 0.8760
Epoch 3/12
658/658 [==============================] - 7s 10ms/step - loss: 0.2411 - accuracy: 0.9072 -
val_loss: 0.3306 - val_accuracy: 0.8608
Epoch 4/12
658/658 [==============================] - 7s 10ms/step - loss: 0.2314 - accuracy: 0.9123 -
val_loss: 0.3277 - val_accuracy: 0.8673
Epoch 5/12
658/658 [==============================] - 7s 11ms/step - loss: 0.2244 - accuracy: 0.9144 -
val_loss: 0.3271 - val_accuracy: 0.8737
Epoch 6/12
658/658 [==============================] - 7s 11ms/step - loss: 0.2168 - accuracy: 0.9183 -
val_loss: 0.3266 - val_accuracy: 0.8696
Epoch 7/12
658/658 [==============================] - 7s 11ms/step - loss: 0.2102 - accuracy: 0.9199 -
val_loss: 0.3261 - val_accuracy: 0.8708
Epoch 8/12
658/658 [==============================] - 7s 11ms/step - loss: 0.2036 - accuracy: 0.9234 -
val_loss: 0.3265 - val_accuracy: 0.8719
Epoch 9/12
658/658 [==============================] - 7s 10ms/step - loss: 0.1970 - accuracy: 0.9247 -
val_loss: 0.3271 - val_accuracy: 0.8737
Epoch 10/12
658/658 [==============================] - 7s 10ms/step - loss: 0.1933 - accuracy: 0.9282 -
val_loss: 0.3293 - val_accuracy: 0.8766
Epoch 11/12
658/658 [==============================] - 7s 11ms/step - loss: 0.1871 - accuracy: 0.9297 -
val_loss: 0.3292 - val_accuracy: 0.8731
Epoch 12/12
658/658 [==============================] - 7s 11ms/step - loss: 0.1808 - accuracy: 0.9318 -
val_loss: 0.3313 - val_accuracy: 0.8749
```

FINAL ACCURACY & LOSS FUNCTION VALUES:

```
658/658 [==============================] - 3s 5ms/step
- loss: 0.3113 - accuracy: 0.8800

Loss:       0.31128421425819397
Accuracy:   0.8799600005149841
```

Graphs:



Exceptions considered:

The Prediction value changes if the padding is True or False; for short sentences, *padding = False* gives better prediction value compared to *pad = True*. However, for large paragraphs, the prediction values remains approximately the same.

Accuracy & Loss Function values vary slightly on every compilation of the model. This may give a slight difference in the prediction value which may impact the neutral sentiments.

Enhancement Scope:

Increasing the accuracy is the biggest enhancement scope. This can be achieved by adjusting the number of layers in the model- it is believed that neither too many nor too less layers can give good accuracy. Other factors include adjusting the epochs values with respect to the BATCH_SIZE.

Link to Code and executable file:

1. https://colab.research.google.com/drive/190qSrqHWuefOyyD7j-eaDgJRgYBhlc-u#scrollTo=TxAHVGlY4aDI
2. https://colab.research.google.com/drive/1LTVgl0jDu0MDn5QH47B_qqFa1sEXPVKp#scrollTo=FewEBEILzjwr
3. https://github.com/Likhitha0510/TCS-iON-RIO-45