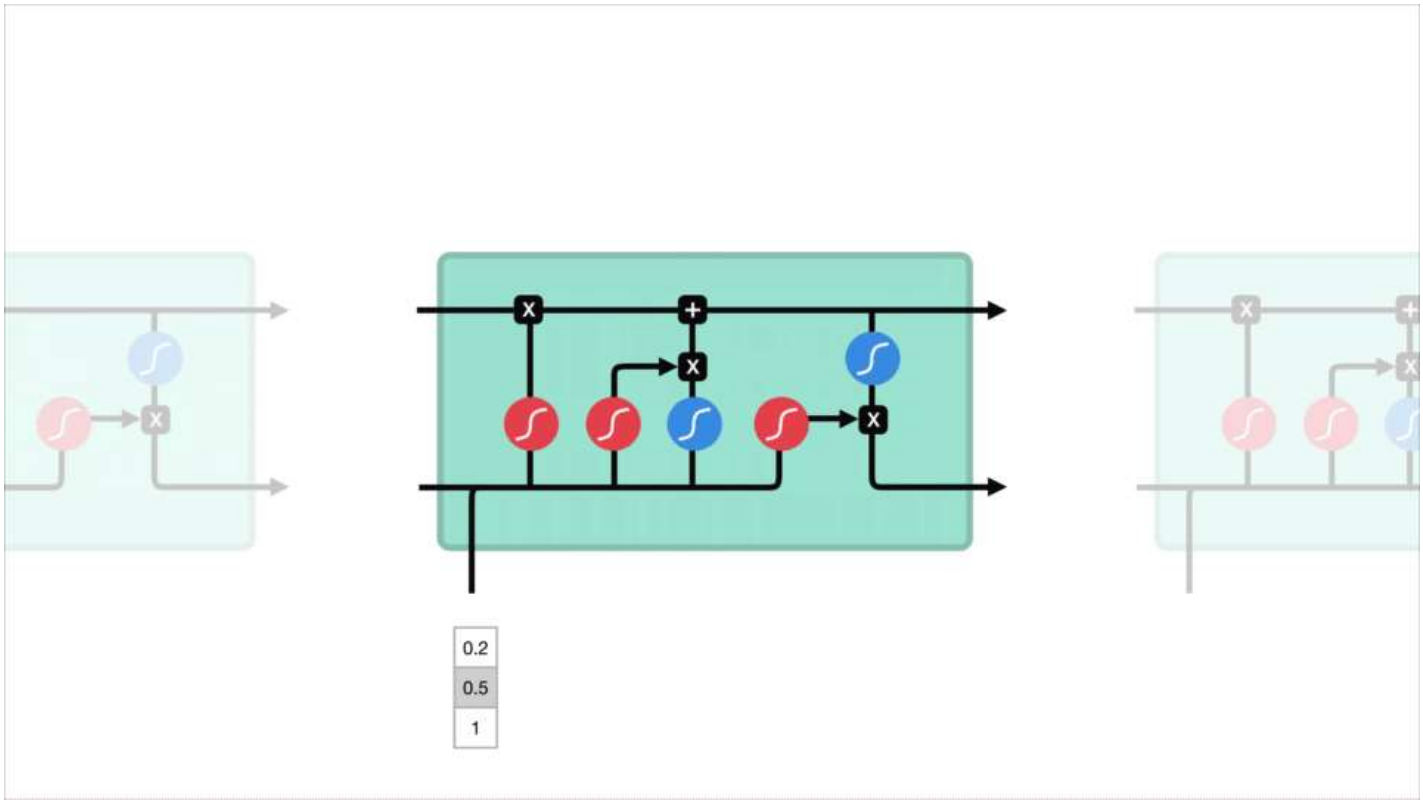
Assignment 7:- Long Short-Term Memory_LSTM

LSTM[Long short term memory] is an improvement over Recurrent Neural Network to address RNN's failure to learn in the presence of past observations greater than 5–10 discrete time steps between relevant input events and target signals (vanishing/exploding gradient issue). LSTM does so by introducing a memory unit called "cell state".



LSTM networks are an extension of recurrent neural networks mainly introduced to handle situations where RNNs fail. Talking about RNN, it is a network that works on the present input by taking into consideration the previous output and storing in its memory for a short period of time. Out of its various applications, the most popular ones are in the fields of speech processing, non-Markovian control, and music composition.

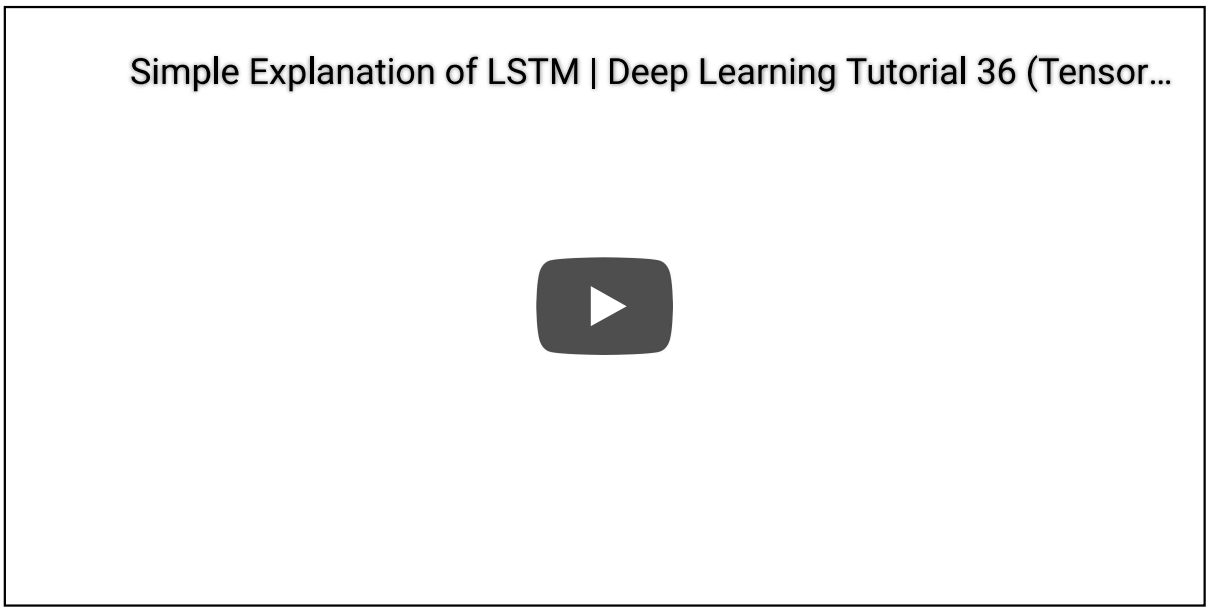
Drawbacks to RNNs.

First, it fails to store information for a longer period of time. At times, a reference to certain information stored quite a long time ago is required to predict the current output. But RNNs are absolutely incapable of handling such “long-term dependencies”. Second, there is no finer control over which part of the context needs to be carried forward and how much of the past needs to be ‘forgotten’. Other issues with RNNs are exploding and vanishing gradients which occur during the training process of a network through backtracking.

Long Short-Term Memory (LSTM)was brought into the picture. It has been so designed that the vanishing gradient problem is almost completely removed, while the training model is left unaltered. Long time lags in certain problems are bridged using LSTMs where they also handle noise, distributed representations, and continuous values. With LSTMs, there is no need to keep a finite number of states from beforehand as required in the hidden Markov model. LSTMs provide us with a large range of parameters such as learning rates, and input and output biases. Hence, no need for

fine adjustments. The complexity to update each weight is reduced to O(1) with LSTMs, similar to that of Back Propagation Through Time (BPTT), which is an advantage.

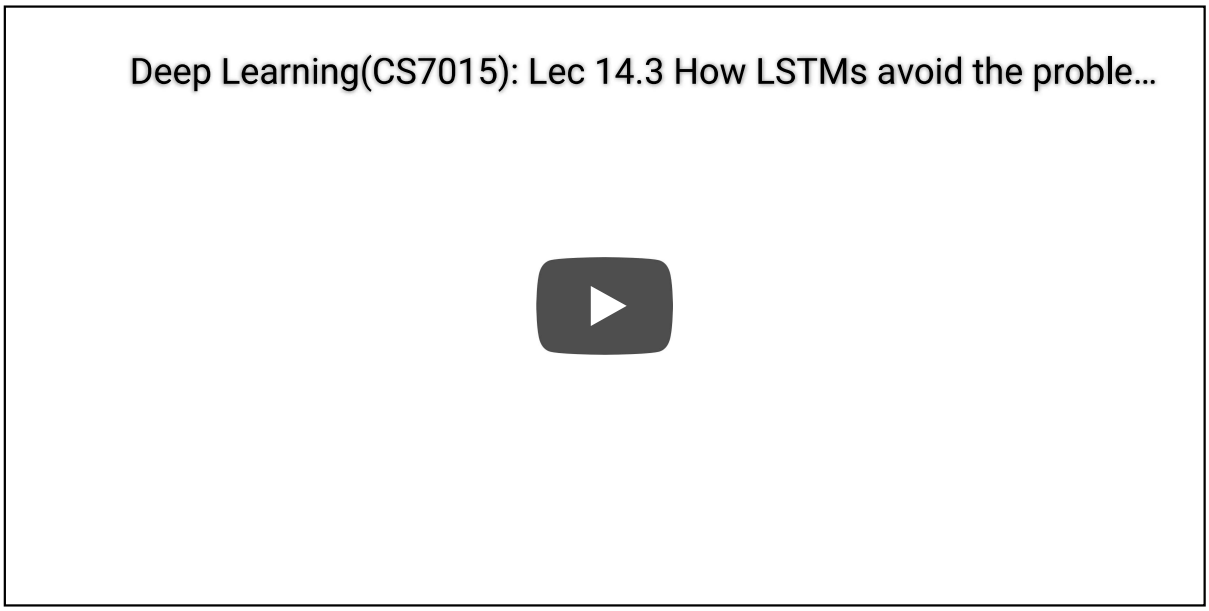
```
###Refer Video
from IPython.display import YouTubeVideo
YouTubeVideo('LfnrRPFhkuY', width=600, height=300)
```



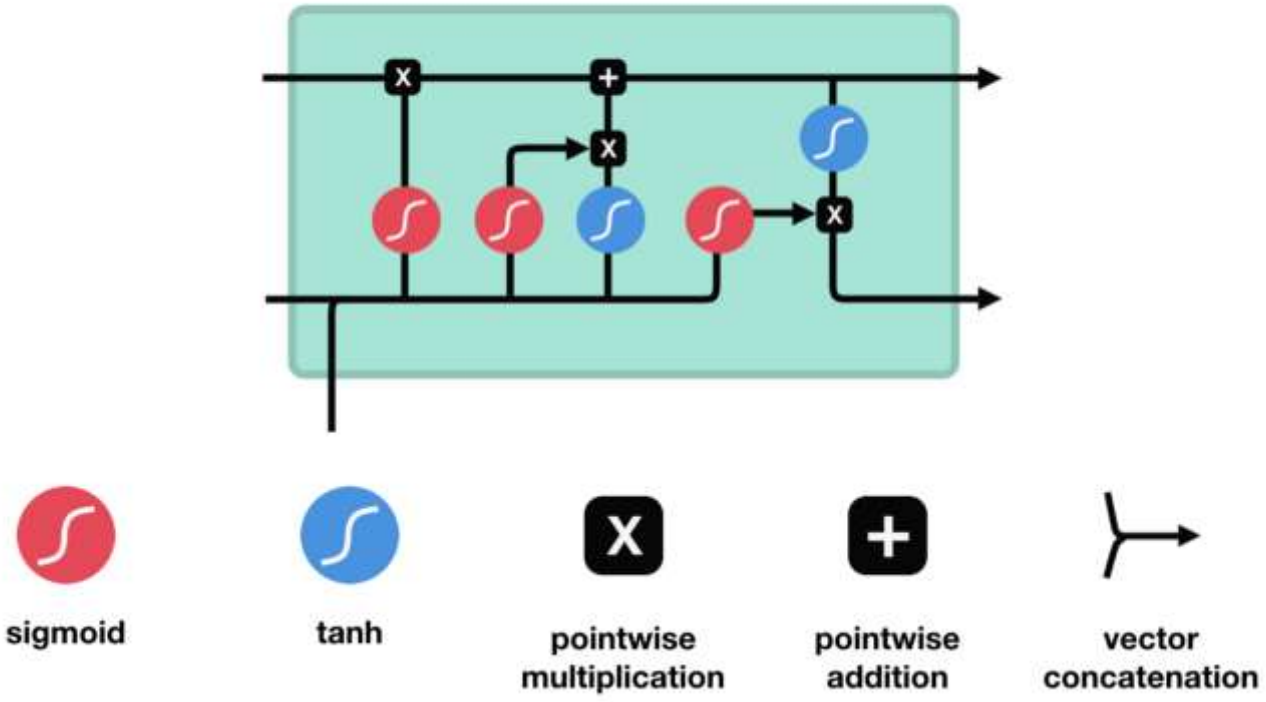
Exploding and Vanishing Gradients problems:

During the training process of a network, the main goal is to minimize loss observed in the output when training data is sent through it. We calculate the gradient, that is, loss with respect to a particular set of weights, adjust the weights accordingly and repeat this process until we get an optimal set of weights for which loss is minimum. This is the concept of backtracking.
Sometimes, it so happens that the gradient is almost negligible. It must be noted that the gradient of a layer depends on certain components in the successive layers. If some of these components are small (less than 1), the result obtained, which is the gradient, will be even smaller. This is known as the scaling effect.
When this gradient is multiplied with the learning rate which is in itself a small value ranging between 0.1-0.001, it results in a smaller value. As a consequence, the alteration in weights is quite small, producing almost the same output as before.
Similarly, if the gradients are quite large in value due to the large values of components, the weights get updated to a value beyond the optimal value. This is known as the problem of exploding gradients. To avoid this scaling effect, the neural network unit was re-built in such a way that the scaling factor was fixed to one. The cell was then enriched by several gating units and was called LSTM.

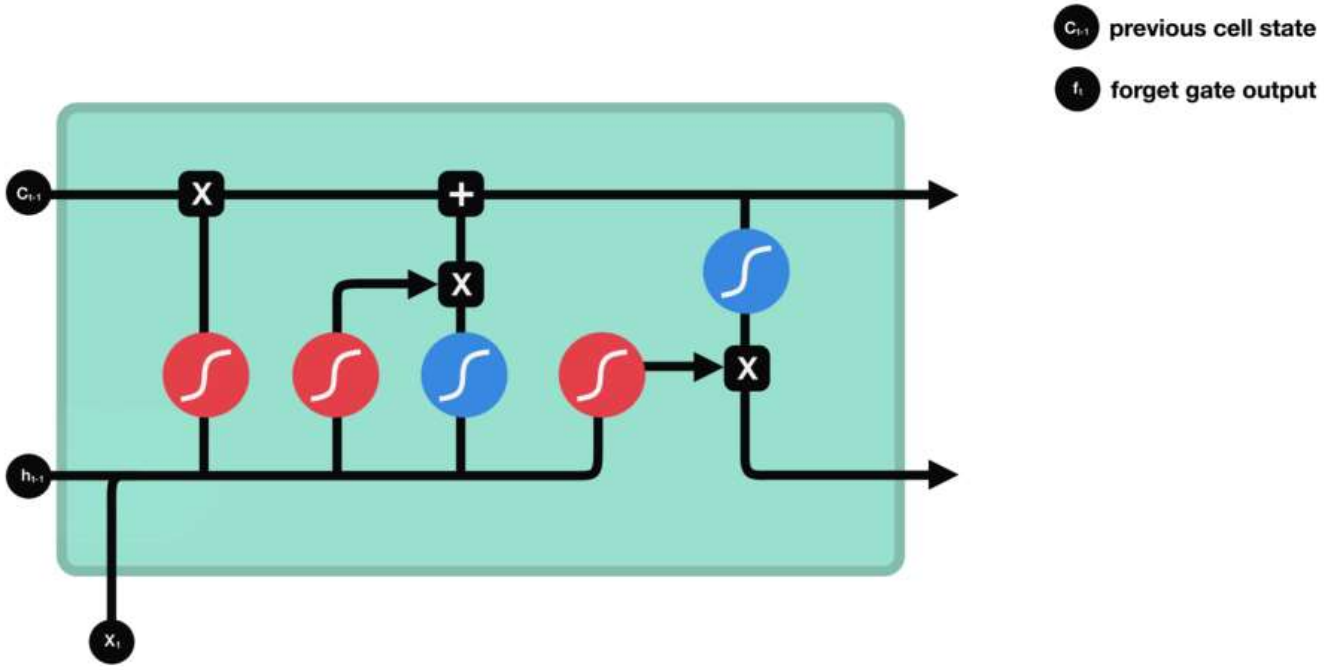
```
###Refer Video
YouTubeVideo('PMv8C-Ws1b8', width=600, height=300)
```



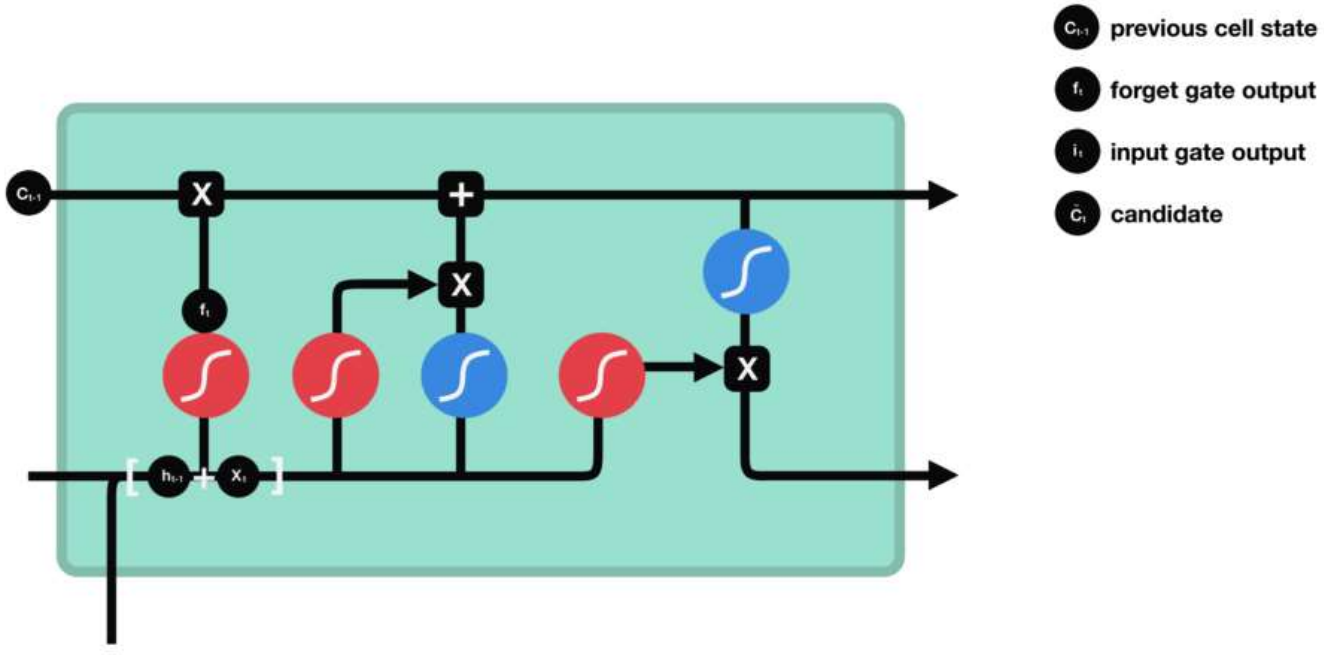
An LSTM has a similar control flow as a recurrent neural network. It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells.



Forget layer: This layer filters or removes info/memory from previous cell state based on current input and previous hidden state. This is done via a sigmoid activation function. This function results only 0 and 1 for inputs. Once it is multiplied to something either it will drop that(multiplication with zero) results in zero or completely pass through(anything multiplied by 1 is same)

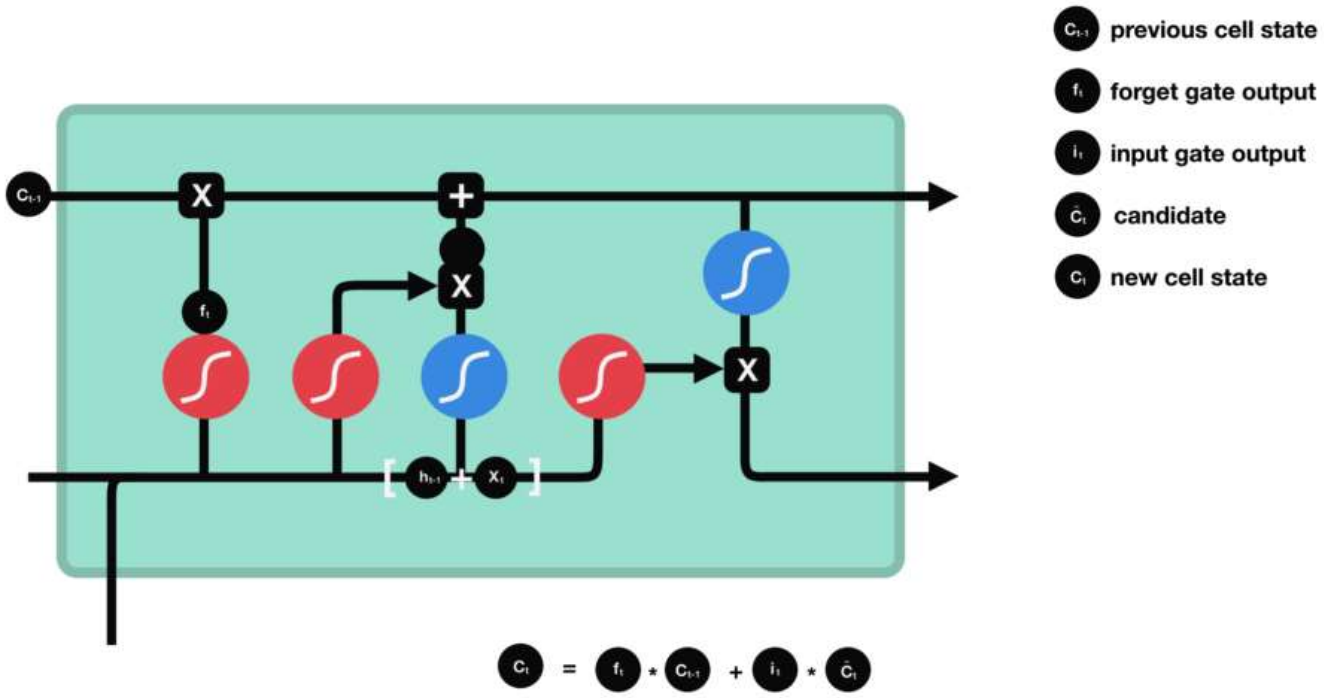


Input Layer: This has again a forget logic, which removes any unwanted information from current input. We also have a modulator which keeps the values in between -1 and 1. This is achieved using a tanh activation function.



Cell State

Now we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant.



Output Layer:

This layer takes current input and current cell state and then outputs the hidden state and cell output. Again we use scaling (tanh) for cell state to keep values in range -1 to 1.

previous cell state

```
###Refer Video
YouTubeVideo('QciIcRxJvsM', width=600, height=300)
```

Let's begin by importing the required libraries.

1. We'll need TensorFlow so we import it as tf.
2. From the TensorFlow Keras Datasets, we import the imdb one.
3. We'll need word embeddings i.e Embedding, Dense and LSTM layers.
4. Our loss function will be binary cross entropy.
5. As we'll stack all layers on top of each other with model.add, we need Sequential for constructing our model.
   For optimization we use an extension of classic gradient descent called Adam.
6. Finally, we need to import pad_sequences. We're going to use the IMDB dataset which has sequences of reviews. While we'll specify a maximum length, this can mean that shorter sequences are present as well; these are not cutoff and therefore have different sizes than our desired one (i.e. the maximum length).
7. We'll have to pad them with zeroes in order to make them of equal length.

```
#importing above mentioned libraries.
```

The next step is specifying the model configuration.
You can easily see how your model is configured, without having to take a look through all the aspects.
We can see that our model will be trained with a batch size of 128, using binary crossentropy loss

and Adam optimization, and only for five epochs (we only have to show you that it works).
20% of our training data will be used for validation purposes, and the output will be verbose, with verbosity mode set to 1 out of 0, 1 and 2. Our learned word embedding will have 15 hidden dimensions and each sequence passed through the model is 300 characters at max. Our vocabulary will contain 5000 words at max.

```
# Model configuration of metrics is accuaracy

#batch size is 128

#embedding hidden dimensions=15[embedding_output_dims]

#loss funciton is BinaryCrossentropy

#max len of sentence is to be 300[max_sequence_length]

#Our vocabulary will contain 5000 words at max.[num_distinct_words]

#nos of epochs=5

#optimizer is adam

#20% is used for validation split

# keep verbosity is 1
```

You might now also want to disable Eager Execution in TensorFlow. While it doesn't work for all, some people report that the training process speeds up after using it. However, it's not necessary to do so – simply test how it behaves on your machine

```
# Disable eager execution
tf.compat.v1.disable_eager_execution()
```

Loading and preparing the data: we can load and prepare the data.
Keras comes with a standard set of datasets, of which the IMDB dataset can be used for sentiment analysis.
we can use imdb.load_data(...)

```
# Load dataset by using (x_train, y_train), (x_test, y_test) for num_distinct_words i.e 5000

#shape of X train and X test
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.n
17465344/17464789 [==============================] - 0s 0us/step
17473536/17464789 [==============================] - 0s 0us/step
(25000,)
(25000,)
```

Once the data has been loaded, we apply pad_sequences. This ensures that sentences shorter than the maximum sentence length are brought to equal length by applying padding with, in this case, zeroes, because that often corresponds with the padding character.
Refer:https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences

```
# Pad all sequences with pad_sequence for x_train and x_test with max sequence length for val
                                                                                # 0.0 because
                                                                                # 0.0 because
```

We can then define the Keras model. we can initialize the model variable with Sequential().
The first layer is an Embedding layer, which learns a word embedding that in our case has a dimensionality of 15.
This is followed by an LSTM layer providing the recurrent segment, and a Dense layer that has one output through Sigmoid a number between 0 and 1.

```
# Define the Keras model
#intialize with sequential()

#intialize first layer for embedding with num_distinct_words, embedding_output_dims and max s

#intialize another layer with LSTM for 10

#adding dense layer with 1 output and having activation function of sigmoid
```

The model can then be compiled. We do so by specifying the optimizer, the loss function, and the metrics that we had specified before.

```
# Compile the model
```

This is also a good place to generate a summary of what the model looks like.

```
# Give a summary
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 300, 15)           75000

 lstm (LSTM)                  (None, 10)                1040

 dense (Dense)                (None, 1)                 11

=================================================================
Total params: 76,051
Trainable params: 76,051
Non-trainable params: 0
_________________________________________________________________
```

Training the Keras model,we can instruct TensorFlow to start the training process.

```
# Train the model for padded inputs, y train, batch size , epochs ,verbose and validation spi
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/5
20000/20000 [==============================] - ETA: 0s - loss: 0.6315 - accuracy: 0.6666
  updates = self.state_updates
20000/20000 [==============================] - 35s 2ms/sample - loss: 0.6315 - accuracy
Epoch 2/5
20000/20000 [==============================] - 24s 1ms/sample - loss: 0.4271 - accuracy
Epoch 3/5
20000/20000 [==============================] - 21s 1ms/sample - loss: 0.3329 - accuracy
Epoch 4/5
20000/20000 [==============================] - 21s 1ms/sample - loss: 0.2819 - accuracy
Epoch 5/5
20000/20000 [==============================] - 21s 1ms/sample - loss: 0.2469 - accuracy
```

The (input, output) pairs passed to the model are the padded inputs and their corresponding class labels. Training happens with the batch size, number of epochs, verbosity mode and validation split that were also defined in the configuration section above.

Evaluating the Keras model We cannot evaluate the model on the same dataset that was used for training it. We fortunately have testing data available through the train/test split performed in the load_data(...) section, and can use built-in evaluation facilities to evaluate the model. We then print the test results on screen. Evaluate the model using the evaluate method by passing the independent test data and dependent test data.

```
# Test the model after training
```

Now print the Test results i.e. the loss and accuracy.

```
# print the loss and accuracy
```

```
Test results - Loss: 0.34331050999641416 - Accuracy: 85.90400218963623%
```

That's cool, 86% is the accuracy of our model. We have come to an end of this project but don't stop here, try as many projects of the similar type to get a better understanding of the use cases. Solve the practice sheet of this project to test yourself.!!

Great job!! You have come to the end of this assignment. Treat yourself for this :))

Do fill this feedback form

You may head on to the next assignment.