Introduction

In this tutorial, we will be discussing what deep learning is and some of the limitations of a fully connected layer after which we will dive fully into the LSTM model and see how we can build one. LSTM model can be applied to majorly text-based problems is known to perform well since it was introduced.

At the end of this tutorial, you should be familiar with:

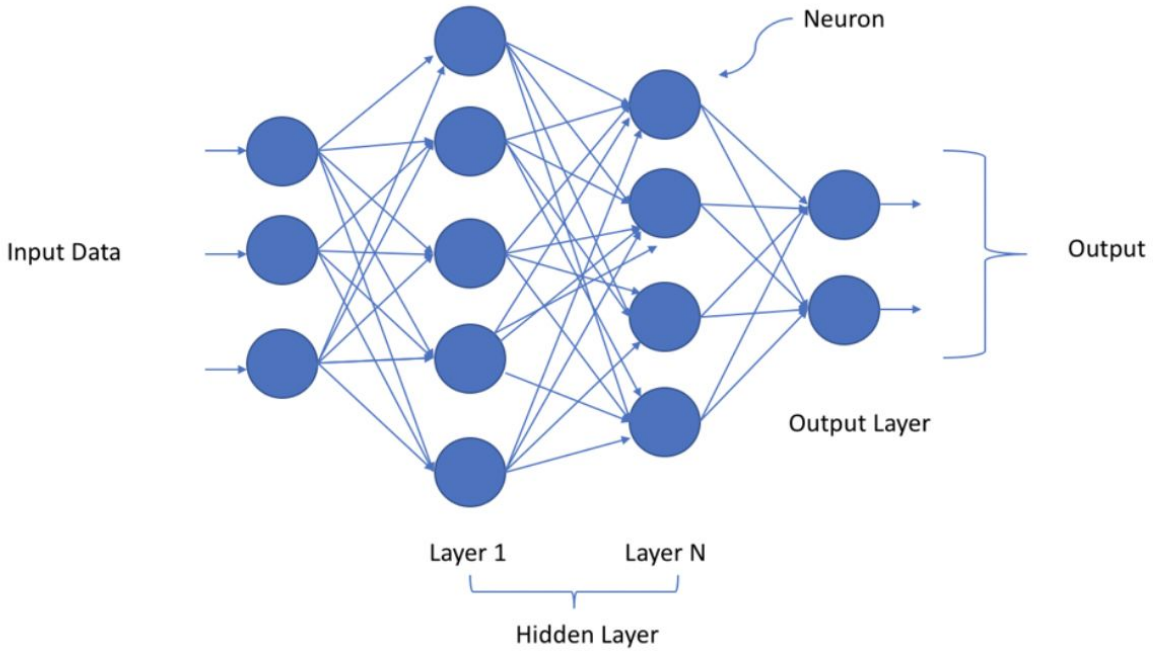
- Deep Learning.
- Some imitations of a fully connected neural network.
- Have an understanding of how deep neural network works.
- Recurrent Neural Networks
- Saw the applications of RNNs
- Dived into LSTMs and some of its variant
- Practised building an LSTM model.

Introduction to Deep Neural Network

Deep neural network also known as deep learning can simply be defined by a neural network that contains more than one single hidden layers. It is a type of machine learning system that uses many layers of nodes to obtain high-level functions from the input data. Deep learning is known for transforming input data into creative and abstract components. Deep learning is an advance neural network algorithm that was built by the inspiration gotten from the biological structure and functioning of the human brain in order to build intelligent systems. Deep learning came about when normal machine learning systems and neural network algorithm failed to carry out complex tasks or failed to work on an unstructured dataset.

Understanding how deep neural network works

A deep neural network is represented as an ordered layered organization of neurons with connection to other neurons, the neurons are similar to the neurons found in the human brain. Information passes from one neuron to the other based on the input data while forming a complex network that learns with some feedback mechanism.



From the above image, we can see that the input data is passed to the first layer, then some mathematical computation is performed at that layer and the result of that computation is passed as inputs to the next layer which then repeats the same process and provides the output. A layer can have many neurons and computes a small function.

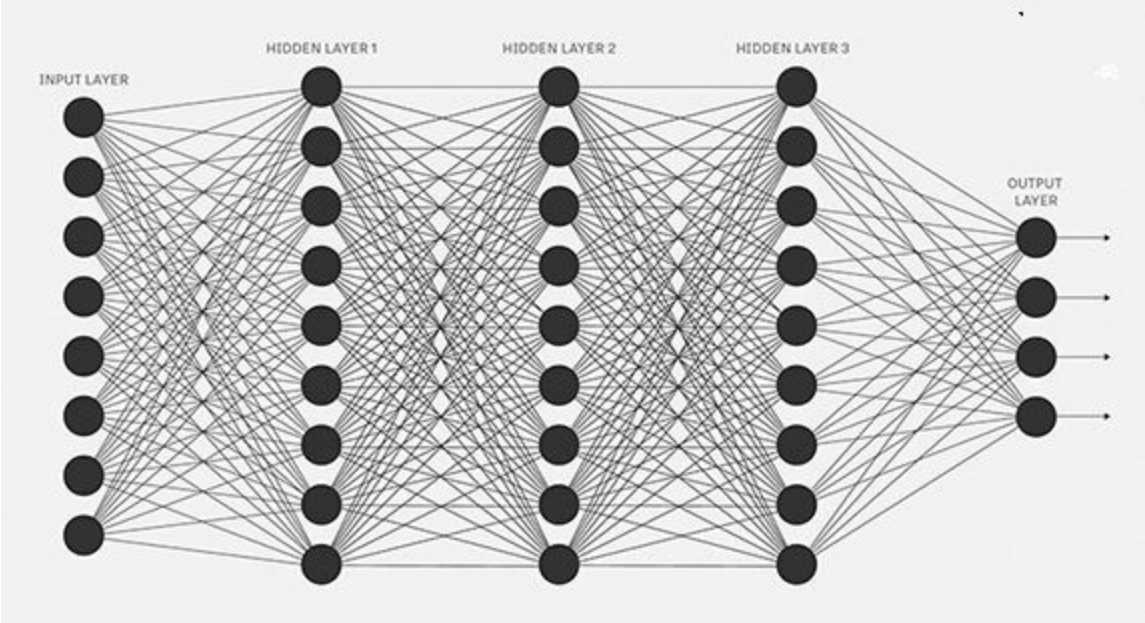
Deep learning structure

The deep learning structure is similar to a neural network structure in that they contain an input layer, the hidden layer and the final output layer.

Input layer: This layer contains the input data that we are trying to build the predictive model or deep learning model for. For a deep learning model, this is mostly a preprocessed unstructured dataset, for example, an image file, a text file, a voice file or a video content.

Hidden layer: This layer determines the complexity of the deep learning model. The hidden layer can be set to be as many as the person building the model wants, but also note that the higher the number of the hidden layers, the longer it takes for the deep learning model to trains.

Output layer: The output layer contains the output of the deep learning model. For example, if we are trying to predict if an image is a cat or a dog, the output of the deep learning model will be a prediction indicating the correct prediction, which is either a cat or dog.



When a deep learning model is being trained, some underlying mathematical computation is performed on each neuron in each layer. FIrstly, each connection between the neurons in different layers contains a weight, the input layer can contain a node known as the bias, this is usually assigned the value of 1, after the initial computation, the output is passed through an activation function that determines the final output of that neuron and input of the next neurone in the next layer, the computation performed on each neuron in each layer can be calculated as

$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

Activation function: The activation function is responsible for determining the output of a deep learning model, the model accuracy and also the model computational efficiency of a training model. Some type of activation functions are linear function, sigmoid function, tanh, relu, leaky relu etc. This activation can make or break your deep learning model. This means that they help in making your model in performing better or are responsible for producing bad models, it alls boils down in you choosing the right activation function for that task.
Weights: Weights can be found on the edge connecting two neurons together, it defines the influence of the input to the output for the next neuron. The weights assigned are usually random and are iteratively updated to get the weights that produce the best model, this happens during the model training.
Bias: This is the additional parameter added to the neural network which adjusts the output along with the weighted sum of the inputs to the neuron. Bias is a constant that helps the model in a way that fits best for any input data.

Types of Deep Neural Network Models

Some types of deep learning models are

1. Unsupervised Pre-trained Networks
2. Convolution Neural Networks
3. Recurrent Neural Networks
4. Recursive Neural Networks

Limitations of a Fully Connected Network

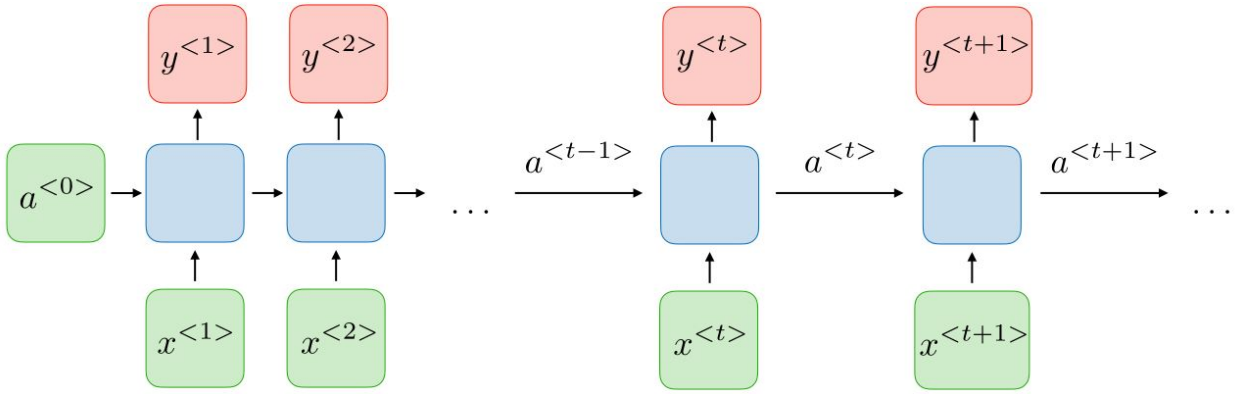
A fully connected network consists of a series of fully connected layers, they connect every neuron in one layer to every neuron in another layer. Neurons in fully connected layers have connections to all activations in the previous layer.

Limitations

1. Fully connected layers are very computationally expensive, this is because of the full connections in each layer which results in the model taking longer to train. This is the main disadvantage of the fully connected network.
2. It takes a longer time to build. To build a fully connected network is tedious and requires a high level of attentiveness so that nothing is missed.
3. Fully connected networks are not good feature extractors for an image problem, compared to CNN.
4. Longer training time.

Introduction to Recurrent Neural Networks

A recurrent neural is a class of artificial neural networks where the connections between nodes form a directed graph along a temporal sequence, allowing it to exhibit a temporal dynamic behaviour, they are also known as RNNs. They are derived from a feedforward neural network and can use their internal state to process variable-length sequences of inputs. They are applied to various tasks such as handwriting recognition, speech recognition etc.



They allow previous outputs to be used as the new inputs while still having the hidden states. They remember the past and are influenced by the decisions made in the past and not only during training. Recurrent neural networks take one or more input vectors and produce one or more output vectors and these output vectors are not only influenced by the weights applied on inputs but also by a hidden state vector representing the context based on prior input and output influencing the same input to produce the different outputs depending on different inputs in the series.

According to Wikipedia
Recurrent neural networks were based on David Rumelhart's work in 1986.[6] Hopfield networks - a special kind of RNN - were discovered by John Hopfield in 1982. In 1993, a neural history compressor system solved a "Very Deep Learning" task that required more than 1000 subsequent layers in an RNN unfolded in time.

The advantages and disadvantages of recurrent neural network architecture are:

Advantages

1. Can process inputs of any length
2. Model size does not increase with the size of the input.
3. COmputation always takes into account historical information.
4. Weights are shared across time

Disadvantages

1. Slow computation
2. Difficulty in accessing information stored a long time ago.
3. Cannot consider any future input for the current state.

Applications of Recurrent Neural Networks

Recurrent neural networks are mostly applied to the natural language processing and speech recognition fields. They include:

1. Music generation
2. Sentiment classification
3. Name entity recognition
4. Machine Translation

What are LSTMs

LSTM is known as Long short term memory networks were invented by Hochreiter and Schmidhuber in 1997, since their creation, LSTM have set accuracy records in multiple applications domains and have proven to be one of the best recurrent neural network models in recent times.

LSTM started to revolutionize speech recognition, therefore outperforming traditional models in certain speech applications. LSTM has also improved large vocabulary speech recognition and text to speech synthesis. LSTM is an artificial neural network architecture used in deep learning and contains a feedback connection, it can process an entire sequence of data such as text and video as well as single data points such as images. It is composed of a cell, an input gate, an output gate and a forget gate and it has the ability to forget to remember values over arbitrary time intervals and the gates mentioned above regulates the flow of information into and out of the cell.

This LSTM network works well on time series data, they are suitable for classifying, processing and making predictions on time series data since there can be no lags of unknown duration between important events in a time series. They were developed to deal with the exploding and vanishing gradient problems that are encountered when we train a traditional recurrent neural network.

LSTM Architecture

The common architecture is composed of a cell and three regulators are known as the gates that are responsible for the flow of information in the LSTM model. These gates are known as the input gate, an output gate and a forget gate. The cell is responsible for keeping track of the dependencies between elements in the input sequence, the input gate controls how new values flow into the cell, the forget gate controls how a value remains in a cell, the output gate controls the extent to which the value in the cell is used to compute the output.

LSTM Cells: The cell state gets the pointwise multiplied by the forget vector, it has the possibility of dropping values in the cell state if it gets multiplied by values near 0, after that we take the output from the input gate and carry out a pointwise addition which updates the cell state to new values that the neural network finds relevant, this gives us the new cell state.

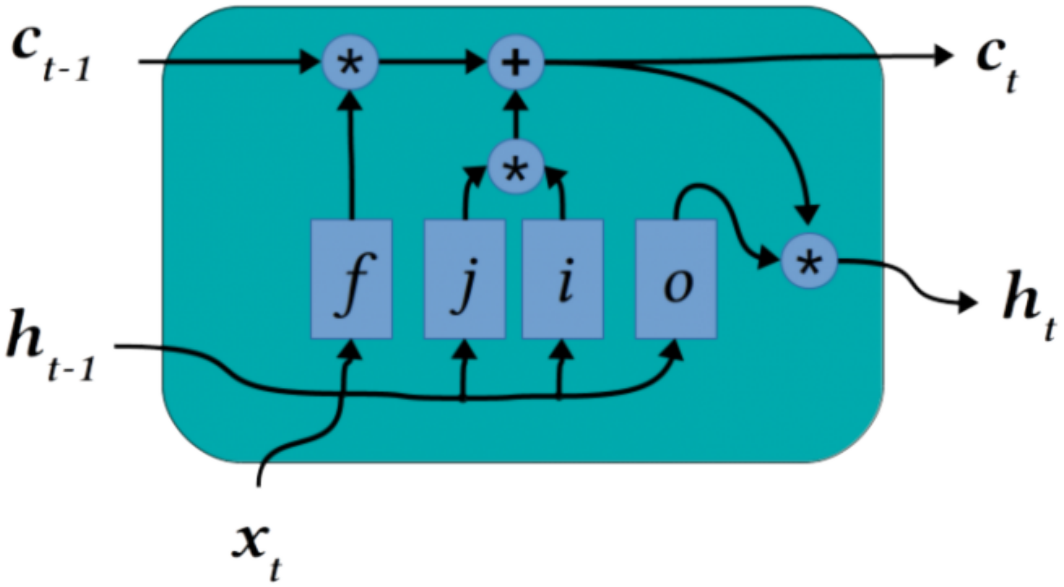
Forget gate: This gate is responsible for deciding which gate is kept or thrown away, both information from the previous hidden state and information from the current input is passed through a sigmoid function. The sigmoid function outputs values between 0 and 1, values close to 0 means to forget and values close to 1 means to keep.

Input gate: The input gate updates the cell state, we first pass the previous hidden state and current input into a sigmoid function, this decides which values will be updated by transforming the values between 0 and 1. 0 for non-important state and 1 for important states. We also pass the hidden state and current input into the tanh function to suppress the values between -1 and 1 to regulate the network, then we multiply the tanh output with the sigmoid output, the sigmoid output determines which information is important to retain from the tanh output.

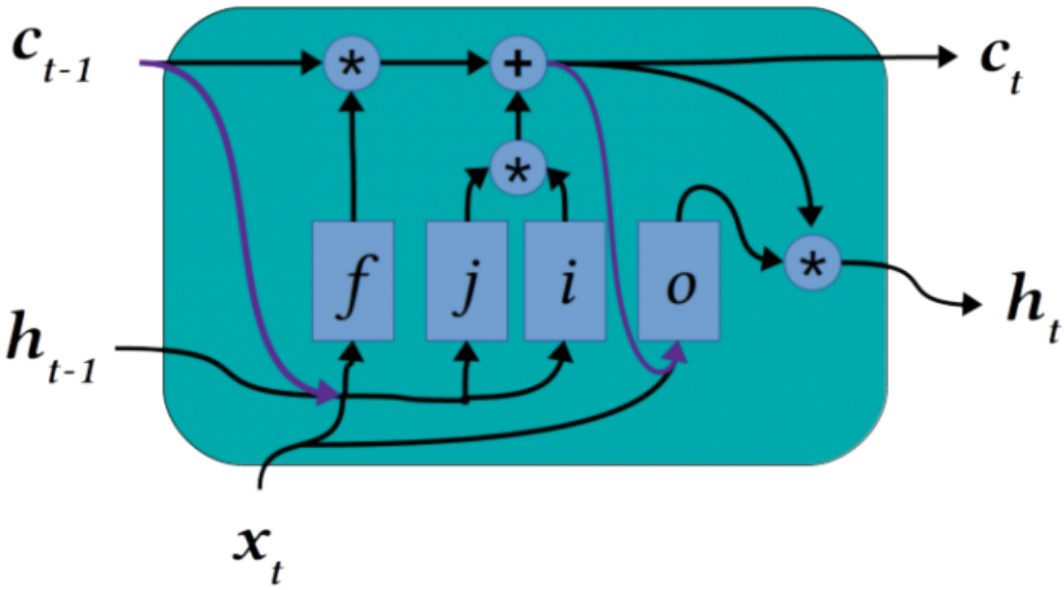
Output gate: This gate decides what the next hidden state would be, we pass the previous hidden state and the current input into a sigmoid function, we then pass the newly modified cell state to the tanh function, then we multiply the tanh output with the sigmoid output to decide what information the hidden state should contain. The output is the hidden state, the new cell state and the new hidden state is then moved over to the next step.

Some LSTM Variants

1. LSTM Classic: The classic LSTM architecture is characterized by a persistent linear cell state surrounded by non-linear layers feeding input and returns an output from it. The LSTM contains 4 gating layers namely the forget gate, two input gate and an output gates. The forget gate determines which value of the old cell state needs to be removed based on the current input data. The two input gates work together to decides what value to add to the cell state depending on the input, the two input gates have different activation functions. The output gate decides what parts of the cell state should be passed to the output. The classic LSTM overcomes the gradients vanishing problem in a recurrent neural network unrolled in time by connecting all-time points via a persistent cell state.

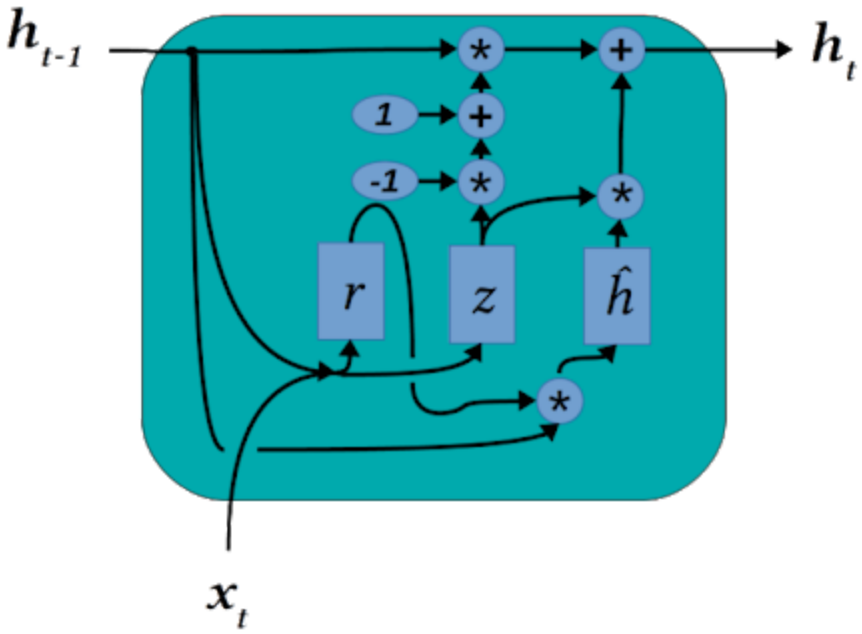


2. **Peephole Connections:** The peephole connection agents always want to know the memories it already has in place before replacing them with a new one.



The peephole connections simply concatenate the cell state contents to the gating layer inputs, this configuration offers an improved ability to count.

3. **Gated Recurrent Unit:** This trains faster than the classical LSTM because of its simplicity, The GRUs combine the gating functions of the input gate and forget gate into a single gate. This makes the cell state positions meant for forgetting will be matched by the entry of new data. The GRUs combines the cell state and hidden output into a single hidden state layer while also having an intermediate internal hidden state.



Hands-on Practice

Now we are going to use LSTM and see how it works in practice. For this hands-on practice, we are going to do a text classification task with LSTM, we will be using the Tensorflow library for this task. We will be using the IMDB large movie review dataset to perform sentiment analysis.

The Data: IMDB large movie review dataset

The Large Movie review dataset is a large dataset for binary sentiment classification containing substantially more data that previous benchmark datasets. It contains a set 25,000 highly polar movie reviews for training and 25,000 for testing. The data can be gotten from here.

First, we import the packages

```
!pip install tf-nightly
import tensorflow_datasets as tfds
import tensorflow as tf
```

```
import matplotlib.pyplot as plt
```

The above code first installs the TensorFlow nightly version then imports the tfds library from TensorFlow dataset and lastly we import the TensorFlow library, we also import the matplotlib library for plotting graphs.
Graph helper function

```
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])
 plt.show()
```

The above code helps with plotting graphs, we will see it being used later.

Download Dataset

```
dataset, info = tfds.load('imdb_reviews/subwords8k', with_info=True,
                          as_supervised=True)
train_examples, test_examples = dataset['train'], dataset['test']
```

Above, we download the dataset from the tfds library we loaded earlier and then split it into train and test set.

```
encoder = info.features['text'].encoder
print('Vocabulary size: {}'.format(encoder.vocab_size))

Vocabulary size: 8185
```

Here we encode the text and print the vocabulary size, we have a vocabulary size of 8185.

Prepare the dataset

```
BUFFER_SIZE = 10000
BATCH_SIZE = 64

train_dataset = (train_examples
                 .shuffle(BUFFER_SIZE)
                 .padded_batch(BATCH_SIZE))
```

```
test_dataset = (test_examples
                .padded_batch(BATCH_SIZE))
```

What the above code does is that it creates a batch of this encoded strings using the padded batch to zero pad the sequences to the length of the longest string in the batch.

Create the model

```
model = tf.keras.Sequential([
   tf.keras.layers.Embedding(encoder.vocab_size, 64),
   tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
   tf.keras.layers.Dense(64, activation='relu'),
   tf.keras.layers.Dense(1)
])

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

Build a tf.keras.Sequential model and start with an embedding layer, this embedding layer converts the sequence of word indices to sequence of vectors. This makes word with similar meanings often have similar vectors.
The bi-directional wrapper is used with the LSTM layer, this propagates the input forward and backwards through the RNN layer and then concatenates the output. This helps the LSTM model to learn long-range dependencies.
Finally, we compile the model to configure the training process.

Train the model

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

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

391/391 [==============================] - 20s 50ms/step - loss: 0.4226 -
accuracy: 0.8499
Test Loss: 0.4225866496562958
Test Accuracy: 0.8499199748039246
```

We train the model with the above code with 10 iterations and validation step=30, we then calculate the test accuracy and test score. The model achieved an accuracy of 85% and loss of 42%.

Make predictions

```
def pad_to_size(vec, size):
 zeros = [0] * (size - len(vec))
 vec.extend(zeros)
 return vec
def sample_predict(sample_pred_text, pad):
 encoded_sample_pred_text = encoder.encode(sample_pred_text)

 if pad:
   encoded_sample_pred_text = pad_to_size(encoded_sample_pred_text, 64)
 encoded_sample_pred_text = tf.cast(encoded_sample_pred_text, tf.float32)
 predictions = model.predict(tf.expand_dims(encoded_sample_pred_text, 0))

 return (predictions)
```

The above functions help with padding the input text so they can be of the same length and creates a sample prediction. If the prediction is >=0.5 it is positive else it is negative.

```
sample_pred_text = ('The movie was cool. The animation and the graphics '
                    'were out of this world. I would recommend this movie.')
predictions = sample_predict(sample_pred_text, pad=False)
print(predictions)
[[-0.5369778]]
```

From the above code, we see that the model returns a negative prediction and clearly, from the text we see that it is positive, this can be improved by adding more layers to the model, I will leave that up to you to do.
Now, we have completely created an LSTM model for sentiment analysis. Link to the notebook can be found here

Further Readings

If you will like to know more about LSTM models and how they can be built, I will share a lost of resources you can use, these are resources I used myself and I am sure they can help you.

- **Deep neural networks**
- **Fundamental of deep learning**
- **A layman's guide to deep neural networks**
- **LSTM**
- **Recurrent neural networks**
- **Recurrent neural network**
- **HowLSTMs work**
- **Types of LSTMs**

Summary

Now we have come to the end of this tutorial, In this tutorial, we discussed what deep learning networks are and built an LSTM model. Particularly we:

- Introduced Deep Learning
- Saw the limitations of a fully connected neural network
- Introduced RNNs
- Saw the applications of RNNs
- Dived into LSTMs and some of its variant
- Practised building an LSTM model.

If you have any questions leave it in the comments. It will be answered shortly.