Machine Learning Project
Phase 2 Deep Learning
Fake News Detection

Submitted to: Sir Usman Shahid

Member 1

Name: Aneeq Asghar
Roll number: 17L-4525
Section: EE-7A

Member 2

Name: Salman Shah
Roll number: 17L-4449
Section: EE-7A

Introduction:

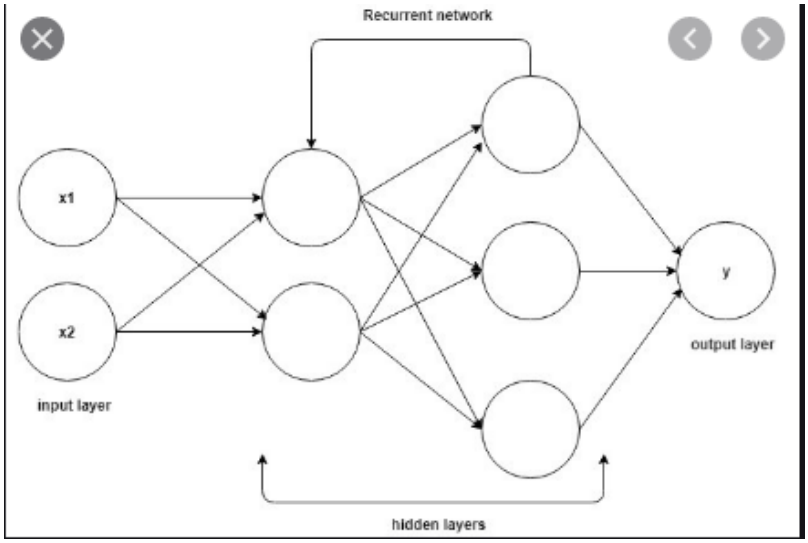
The machine learning project we selected is on Fake News detection. In phase 1 of the project, we predicted the Fake News using a simple machine learning model. The machine learning model we built in phase one of the project is on Logistic regression. Now, in the second phase of the project, we used deep learning to predict the news.

Deep Learning Algorithms:

There are two deep learning algorithms to predict news as we have to analyze the words to predict the news, by converting the words into tokens :

- Recurrent Neural Networks

  It uses the multilayer neural network but the issue over here is that it has a vanishing gradient problem as we will be using backpropagation by derivating the term many times. By having a large number of derivatives in the process we are not able to reach the local minima and a vanishing gradient issue occurs.



- Long Short Term Memory

  The issue of vanishing gradient is resolved by using Long Short-Term Memory which remembers the previous output of the nodes and sends it as an input to the next. It also stores the output of this function as an input to the node as a feedback to improve the performance and accuracy of the model. Such as, if the word is sent from a sentence to the hidden layer of the neuron, then the weight is assigned to it and the word is preprocessed and the previous output is also given to the layer of neural network and the sequence is maintained.

```
import nltk
nltk.download('punkt')
```

The command shown above is used to convert the dataset to a list.

```
import tensorflow as tf
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Tensorflow is used for deep learning and neural networks.
2. Pandas library is imported to import the dataset.
3. Numpy library is used for mathematical library operations
4. Seaborn and matplotlib library is used for data visualization in plotting images and graphs.

The basic procedure for applying deep learning in our model is to first clean the dataset and convert the words to tokens for statistical computation and prediction. The dataset is cleaned using the

Cleaning the dataset:

It is done using by firstly removing all the Not a Number columns and rows in the dataset by using the dropna command.

Secondly, we remove the URL column from the dataset as we do not need it to predict the label of the news:

```
df.drop(columns=['URLs'],inplace=True)
```

Thirdly, we combined both the Headline and the Body of the news through which we will do the computations as shown below:

```
df['orignal'] = df['Headline']+ ' ' + df['Body']
```

Fourthly, we removed the English stopwords from the dataset which aren't of importance while predicting the label of the news. We are removing these stopwords to reduce the size of the data and to reduce the load on the PC. This is done by using the following commands:

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/aneeq/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
from nltk.corpus import stopwords
```

```
stop_words = stopwords.words('english')
```

```
import gensim
from gensim.utils import simple_preprocess
from gensim.parsing.preprocessing import STOPWORDS
```

```
def preprocess(text):
    result =[]
    for token in gensim.utils.simple_preprocess(text):
        if token not in gensim.parsing.preprocessing.STOPWORDS and len(token)>3 and token not in stop_words:
            result.append(token)
    return result
```

```
df['clean'] =df['orignal'].astype(str).apply(preprocess)
```

Now, the data frame clean is changed to a tuple by using a for loop and appending in the list of words:

```
list_of_words =[]
for i in df.clean:
    for j in i:
        list_of_words.append(j)
```

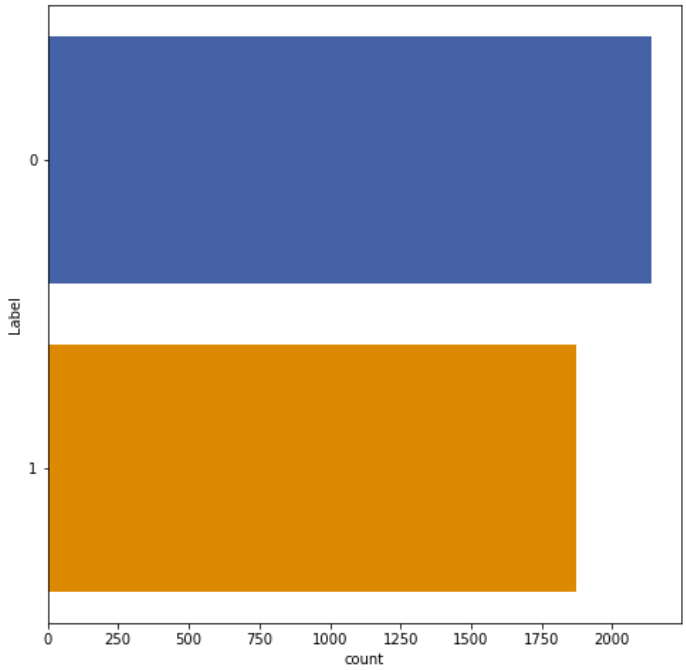
These bag of words are converted to a continuous list by appending a space after each line as shown below:

```
df['clean_joined']= df['clean'].apply(lambda x: " ".join(x))
```

Data Visualization:

Data is classified as fake or true based on the label on it. It is visualized by using the following commands:

```
plt.figure(figsize = (8,8))
sns.countplot(y= 'Label' , data=df)
```



Convert words to tokens and padding:

Words are converted to tokens using the tokenizer and the maximum length of an individual line or row is computed. The maximum length is used to pad the lines by appending zeros after the length of the individual line. Data is separated into training and test set to check the accuracy of the model where 20% of the data is in the test set whereas, 80% of the data is in the training set. The tokenizer is used to convert the data into tokens or numbers replaced for each word in the data. Tokenizer is used to substitute a randomly generated identifier in order to prevent unauthorized access.

```
from nltk.tokenize import word_tokenize
maxlen = -1
for doc in df.clean_joined:
    tokens = nltk.word_tokenize(doc)
    if(maxlen<len(tokens)):
        maxlen= len(tokens)
```

```
print("the maximum number of words in any document is =  ", maxlen)
```

```
the maximum number of words in any document is =   2683
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(df.clean_joined, df.Label, test_size=0.2)
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
tokenizer = Tokenizer(num_words= total_words)
tokenizer.fit_on_texts(x_test)
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
padded_train = pad_sequences(train_sequences , maxlen =2683, padding ='post', truncating='post')
padded_test = pad_sequences(test_sequences , maxlen =2683, padding ='post', truncating='post')
```

The last thing we will do is to implement the Deep learning algorithm LSTM in it.
We will do the word embedding. It is done to boost the generalization and performance of LSTM. Here, we will form a sequential model and use a bidirectional LSTM layer with 128 units as we have 128 output dimensions of the word embedding. The summary of the model is shown below:

Iteration 1 of model 1:

```
model= Sequential()
model.add (Embedding (total_words, output_dim =128 ))
model.add(Bidirectional(LSTM(128)))
model.add(Dense(1, activation = 'relu'))
model.compile(optimizer= 'adam' , loss = 'binary_crossentropy'  , metrics = ['acc'] )
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 128)         5372032
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               263168
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 5,635,457
Trainable params: 5,635,457
Non-trainable params: 0
```

```
y_train = np.asarray(y_train)
model.fit(padded_train,y_train, batch_size =64 , validation_split =0.1 , epochs =2)
```

```
Epoch 1/2
46/46 [==============================] - 695s 15s/step - loss: 0.6849 - acc: 0.7179 - val_loss: 0.2081 - val_acc: 0.8972
Epoch 2/2
46/46 [==============================] - 721s 16s/step - loss: 0.0641 - acc: 0.9757 - val_loss: 0.1441 - val_acc: 0.9533
```

```
pred = model.predict(padded_test)
```

```
prediction= []
for i in range (len(pred)):
    if pred[i].item() >0.5:
        prediction.append(1)
    else:
        prediction.append(0)
```

```
from sklearn.metrics import accuracy_score
```

```
accuracy = accuracy_score(list(y_test), prediction)
print("Model accuracy with Relu activation function: ", accuracy)
from sklearn.metrics import confusion_matrix
ca = confusion_matrix(list(y_test), prediction)
print("Confusion matrix with Relu activation function: \n",ca)
plt.figure(figsize = (10,10))
sns.heatmap(ca, annot=True)
```

```
Model accuracy with Relu activation function:  0.5511221945137157
Confusion matrix with Relu activation function:
 [[184 247]
 [113 258]]
```

```
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           0       0.62      0.43      0.51       431
           1       0.51      0.70      0.59       371

    accuracy                           0.55       802
   macro avg       0.57      0.56      0.55       802
weighted avg       0.57      0.55      0.54       802
```

ITERATION 2 of MODEL 1:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 100)         4196900
_________________________________________________________________
bidirectional_2 (Bidirection (None, 200)               160800
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 201
=================================================================
Total params: 4,357,901
Trainable params: 4,357,901
Non-trainable params: 0
_________________________________________________________________
LSTM=100
 None
Model accuracy with relu activation function:  0.5374064837905237
Confusion matrix:
 [[431   0]
 [371   0]]
Classification Report of sigmoid activation function
              precision    recall  f1-score   support

           0       0.54      1.00      0.70       431
           1       0.00      0.00      0.00       371

    accuracy                           0.54       802
   macro avg       0.27      0.50      0.35       802
weighted avg       0.29      0.54      0.38       802
```

ITERATION 1 of MODEL 2:

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 128)         5372032
_________________________________________________________________
bidirectional_3 (Bidirection (None, 256)               263168
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 5,635,457
Trainable params: 5,635,457
Non-trainable params: 0
_________________________________________________________________
LSTM=128  Activation=sigmoid
 None
Model accuracy with sigmoid activation function:  0.4613466334164589
Confusion matrix:
 [[  0 431]
 [  1 370]]
Classification Report of sigmoid activation function
               precision    recall  f1-score   support

           0       0.00      0.00      0.00       431
           1       0.46      1.00      0.63       371

    accuracy                           0.46       802
   macro avg       0.23      0.50      0.32       802
weighted avg       0.21      0.46      0.29       802
```

ITERATION 2 OF MODEL 2:

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 100)         4196900
_________________________________________________________________
bidirectional_4 (Bidirection (None, 200)               160800
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 201
=================================================================
Total params: 4,357,901
Trainable params: 4,357,901
Non-trainable params: 0
_________________________________________________________________
LSTM=100  Activation=sigmoid
 None
Model accuracy with sigmoid activation function:  0.5361596009975063
Confusion matrix:
 [[399  32]
 [340  31]]
Classification Report of sigmoid activation function
              precision    recall  f1-score   support

           0       0.54      0.93      0.68       431
           1       0.49      0.08      0.14       371

    accuracy                           0.54       802
   macro avg       0.52      0.50      0.41       802
weighted avg       0.52      0.54      0.43       802
```

ITERATION 1 OF MODEL 3:

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, None, 100)         4196900
_________________________________________________________________
bidirectional_4 (Bidirection (None, 200)               160800
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 201
=================================================================
Total params: 4,357,901
Trainable params: 4,357,901
Non-trainable params: 0
_________________________________________________________________
LSTM=100  Activation=sigmoid
 None
Model accuracy with sigmoid activation function:  0.5361596009975063
Confusion matrix:
 [[399  32]
 [340  31]]
Classification Report of sigmoid activation function
              precision    recall  f1-score   support

           0       0.54      0.93      0.68       431
           1       0.49      0.08      0.14       371

    accuracy                           0.54       802
   macro avg       0.52      0.50      0.41       802
weighted avg       0.52      0.54      0.43       802
```

ITERATION 2 OF MODEL 3:

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, None, 100)         4196900
_________________________________________________________________
bidirectional_6 (Bidirection (None, 200)               160800
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 201
=================================================================
Total params: 4,357,901
Trainable params: 4,357,901
Non-trainable params: 0
_________________________________________________________________
LSTM=100  Activation=Tanh
 None
Model accuracy with tanh activation function:  0.5374064837905237
Confusion matrix:
 [[431   0]
 [371   0]]
Classification Report of tanh activation function
              precision    recall  f1-score   support

           0       0.54      1.00      0.70       431
           1       0.00      0.00      0.00       371

    accuracy                           0.54       802
   macro avg       0.27      0.50      0.35       802
weighted avg       0.29      0.54      0.38       802
```

TABLE SHOWING ACCURACIES:

MODEL	**Iteration**	**Accuracy**
1 (Relu Activation Function)	1 (128 LSTM units)	55% ~97.57% with the model
1 (Relu Activation Function)	2 (100 LSTM units)	54%
2 (Sigmoid Activation Function)	1 (128 LSTM units)	46%
2 (Sigmoid Activation Function)	2 (100 LSTM units)	54%

3 (Hyperbolic Tangent Function)	1 (128 LSTM units)	54%
3 (Hyperbolic Tangent Function)	2 (100 LSTM units)	54%

Model 1 with iteration 1 selected due to the highest accuracy which is 98% when the model is fitted to the dataset. Hence, a Rectified Linear Unit with an LSTM of 128 units is selected.

Issues:

Fitting the model onto the dataset took a lot of time and my PC got stuck several times hence, I could not do the number of iterations required to select a model to predict the result.

Conclusion:

Computing for 128 LSTM units give a better accuracy rather than 100 LSTM iterations and we learnt Recurrent neural networks and Long short term memory through this project.