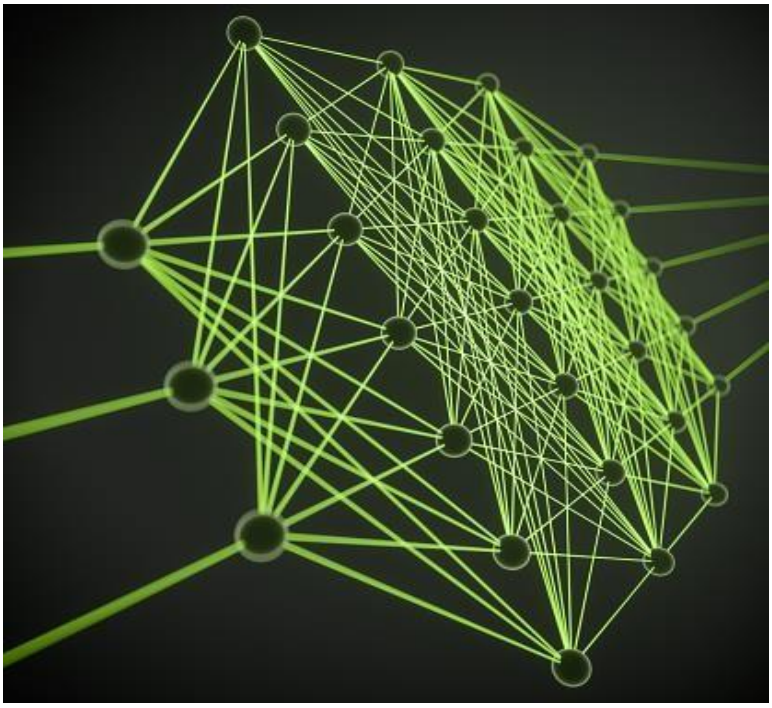
Twitter Sentimental Analysis using RNN and LSTM



Step 1: Downloading and Uploading dataset

First, we will download the twitter sentiment dataset from Kaggle the link is given below:

Dataset Link: Sentiment Analysis Dataset | Kaggle

After downloading this dataset from the website, we will replace the word in labels as follows:

Negative Sentiment: 0

Positive Sentiment: 1

After this we will upload the data on our drive that we will mount on the Google Colab. **Step 2: Changing runtime and loading the data in Colab**

- Create a new notebook in Colab
- Go on the Runtime tab and change the Runtime type to GPU and save it.
- Mount the Drive in which you uploaded the Dataset you want to train the model.
- After mounting the data read the dataset using the code given below.

```
import pandas as pd
 train_df =
pd.read_csv('/content/drive/MyDrive/train.csv',encoding='latin-
1')
train_df
 train_df['label'].value_counts()
```

Output:

```
1    8582
0    7781
Name: label, dtype: int64
```

Step 3: Downloading and Loading pre-embedded data from the web

- Run the following command in the notebook to download the embedded dataset

```
!wget http://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
```

- Now run `!unzip glove.6B.zip` to unzip the dataset archive in the machine **Output:**

```
Archive:  glove.6B.zip
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

Step 4: Vectorizing the embedded dataset

- Now we will run the following code to vectorize the code.
- And load it into the dictionary function so it can be called directly.
- We can see that there are four files, and each are the same embedded dataset files but with different dimensions.
- We will use the 50d the fifty-dimensional ones.
- This file contains 400000.

```
import numpy as np


words = dict()
 def add_to_dict(d, filename):
with open(filename, 'r') as f:
for line in f.readlines():
line = line.split(' ')
 try:
d[line[0]] = np.array(line[1:], dtype=float)
except: continue
 add_to_dict(words,
'glove.6B.50d.txt') words
```

Output:

```
{'the': array([ 4.1800e-01,  2.4968e-01, -4.1242e-01,  1.2170e-01,  3.4527e-01,
        -4.4457e-02, -4.9688e-01, -1.7862e-01, -6.6023e-04, -6.5660e-01,
         2.7843e-01, -1.4767e-01, -5.5677e-01,  1.4658e-01, -9.5095e-03,
         1.1658e-02,  1.0204e-01, -1.2792e-01, -8.4430e-01, -1.2181e-01,
        -1.6801e-02, -3.3279e-01, -1.5520e-01, -2.3131e-01, -1.9181e-01,
        -1.8823e+00, -7.6746e-01,  9.9051e-02, -4.2125e-01, -1.9526e-01,
         4.0071e+00, -1.8594e-01, -5.2287e-01, -3.1681e-01,  5.9213e-04,
         7.4449e-03,  1.7778e-01, -1.5897e-01,  1.2041e-02, -5.4223e-02,
        -2.9871e-01, -1.5749e-01, -3.4758e-01, -4.5637e-02, -4.4251e-01,
         1.8785e-01,  2.7849e-03, -1.8411e-01, -1.1514e-01, -7.8581e-01]),
         400000
```

Step 5: Downloading required NLP libraries and Tokenizing the data

- Download wordnet and omw file to deal with string data from NLP library
- After this we will tokenize the data. Which divides the data into small parts, so the sentences become easier to use or translate into other language or format.
- After this we will Lemmatizer our data. Lemmatizer will convert the words to the most common form of that word or the most used similar word so it is easier to detect.

```
import nltk
 nltk.download('wordnet
')
 nltk.download('omw-1.4')
nltk.download('wordnet')



tokenizer = nltk.RegexpTokenizer(r"\w+")
 tokenizer.tokenize('@user when a father is dysfunctional and
is')
 from nltk.stem import
WordNetLemmatizer
 lemmatizer =
WordNetLemmatizer()

#lemmatizer.lemmatize('feet')
 def message_to_token_list(s):   tokens = tokenizer.tokenize(s)
lowercased_tokens = [t.lower() for t in tokens]   lemmatized_tokens =
[lemmatizer.lemmatize(t) for t in lowercased_tokens]   useful_tokens = [t
for t in lemmatized_tokens if t in words]



  return useful_tokens

message_to_token_list('@user feet a fathers is dysfunctional and is')
```

Output:

Before tokenizing:

```
@user when a father is dysfunctional and is'
```

After tokenizing:

```
['user', 'when', 'a', 'father', 'is', 'dysfunctional', 'and', 'is']
```

Before Lemmatizer:

```
@user feet a fathers is dysfunctional and is
```

After Lemmatizer:

```
['user', 'foot', 'a', 'father', 'is', 'dysfunctional', 'and', 'is']
```

Step 6: Vectorizing and Dividing Test and Training data

- Now we will create a function that will loop and vectorize the data and return as a float/decimal valued data.
- Now we will divide the data into training and testing with 70% for training and 30% for testing which is considered a standard.

```
def message_to_word_vectors(message, word_dict=words):
  processed_list_of_tokens = message_to_token_list(message)
   vectors =
[]
   for token in
processed_list_of_tokens:     if token
not in word_dict:
      continue


    token_vector = word_dict[token]
vectors.append(token_vector)


  return np.array(vectors, dtype=float)


message_to_word_vectors('@user when a father is dysfunctional and is').sha
pe
```

```
train_df = train_df.sample(frac=1, random_state=1)
train_df.reset_index(drop=True, inplace=True)
 split_index_1 = int(len(train_df) * 0.7)
split_index_2 = int(len(train_df) * 0.85)

train_df, val_df, test_df = train_df[:split_index_1], train_df[split_index
_1:split_index_2], train_df[split_index_2:]
len(train_df), len(val_df), len(test_df) test_df
```

<u>Step 7</u>: Finding out the overall size of data and padding

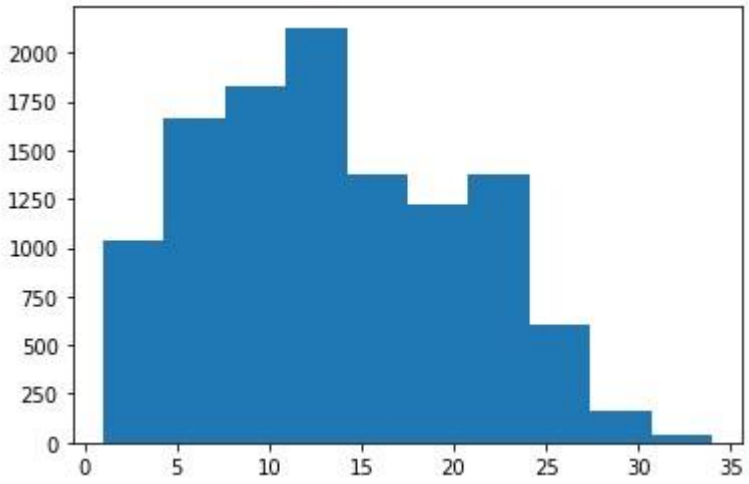
- We will define `df_to_X_y` so we can count the sentences size.
- Then will plot to see the max and min num of tokens our dataset has in a sentence.
- After this we will pad our dataset to the max size of the token, so we don't get bad results. So, our training is accurate, and our shape of data is relevant.

Function for counting the token size for our dataset

```
def df_to_X_y(dff):
  y = dff['label'].to_numpy().astype(int)

  all_word_vector_sequences = []
   for message in
dff['tweet']:
    message_as_vector_seq = message_to_word_vectors(message)
         if message_as_vector_seq.shape[0]
== 0:
      message_as_vector_seq = np.zeros(shape=(1, 50))

all_word_vector_sequences.append(message_as_vector_seq)
     return
all_word_vector_sequences, y
```

Now we print and get an output graph for the token size

```
X_train, y_train = df_to_X_y(train_df)

print(len(X_train), len(X_train[0]))


print(len(X_train), len(X_train[2]))


sequence_lengths = []

for i in range(len(X_train)):
  sequence_lengths.append(len(X_train[i]))

import matplotlib.pyplot as plt
 plt.hist(sequence_lengths)



pd.Series(sequence_lengths).describe()
```




```
count    11454.000000
mean        13.446831
std          6.945165
min          1.000000
25%          8.000000
50%         13.000000
75%         19.000000
max         34.000000
dtype: float64
```

This function is used for padding undersized data.

```
from copy import deepcopy
 def pad_X(X,
desired_sequence_length=57):
  X_copy = deepcopy(X)


  for i, x in enumerate(X):
x_seq_len = x.shape[0]
    sequence_length_difference = desired_sequence_length - x_seq_len
         pad = np.zeros(shape=(sequence_length_difference,
50))


    X_copy[i] = np.concatenate([x, pad])
     return
np.array(X_copy).astype(float)

X_train = pad_X(X_train)

X_train.shape
 y_train.shape
```

```
X_val, y_val = df_to_X_y(val_df)
X_val = pad_X(X_val)

X_val.shape, y_val.shape

X_test, y_test = df_to_X_y(test_df)
X_test = pad_X(X_test)

X_test.shape, y_test.shape
```

Step 8: Modeling RNN

- We will create our model by designing Lstm, Dropout, Dense and flatten layer.
- We will get lstm layers to assign waits
- Dropout to remove overfitting
- Dense to get the most values
- Then flatten to get the output layer

Code:

```
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential model
= Sequential([])
 model.add(layers.Input(shape=(57, 50)))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2))
model.add(layers.LSTM(64, return_sequences=True))
model.add(layers.Dropout(0.2)) model.add(layers.Flatten())



model.add(layers.Dense(1, activation='sigmoid'))
model.summary()
```

- We are using shape 57 as the max token size is 50
- Using three LSTM layers and 64 sized filter
- Dropout of 0.2 so our model doesn't over fit
- Flatten over layer in the end
- And dense to get the values and sigmoid activation function because it's a binary class.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 57, 64)            29440

 dropout (Dropout)           (None, 57, 64)            0

 lstm_1 (LSTM)               (None, 57, 64)            33024

 dropout_1 (Dropout)         (None, 57, 64)            0

 lstm_2 (LSTM)               (None, 57, 64)            33024

 dropout_2 (Dropout)         (None, 57, 64)            0

 flatten (Flatten)           (None, 3648)              0

 dense (Dense)               (None, 1)                 3649

=================================================================
Total params: 99,137
Trainable params: 99,137
Non-trainable params: 0
```

Step 9: Compiling model

- First, we are defining the location where we want to save our trained weights.
- Optimizer we used is Adam.
- Loss is Binary Crossentropy.

```
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.optimizers import Adam from
tensorflow.keras.metrics import AUC from
tensorflow.keras.callbacks import ModelCheckpoint


cp = ModelCheckpoint('/content/drive/MyDrive/models/tw.h', save_best_only=
True)
model.compile(optimizer=Adam(learning_rate=0.0001),
loss=BinaryCrossentropy(),
metrics=['accuracy', AUC(name='auc')]) frequencies =
pd.value_counts(train_df['label']) frequencies
```

```
weights = {0: frequencies.sum() / frequencies[0], 1: frequencies.sum() / f
requencies[1]}
weights
```

Step 10: Fitting the model

- Now we will fit our model using the fit command.
- We are giving 20 epochs to train our data.

```
model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=2
0, callbacks=[cp], class_weight=weights)
```

<u>Step 11</u>: Loading our model and printing the accuracy of our model

- Now we will load our trained model and the weights.
- The give it a test data to get the accuracy of our model given below.

```
from tensorflow.keras.models import load_model
 best_model =
load_model('/content/drive/MyDrive/models/tw.h')
 test_predictions = (best_model.predict(X_test) >
0.5).astype(int)
 from sklearn.metrics import
classification_report
 print(classification_report(y_test,
test_predictions))
```

Output:

```
77/77 [==============================] - 1s 6ms/step
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      1147
           1       0.88      0.81      0.85      1308

    accuracy                           0.84      2455
   macro avg       0.84      0.84      0.84      2455
weighted avg       0.84      0.84      0.84      2455
```

We have accuracy of positive sentiment: 88 %

We have accuracy of negative sentiment: 81 %

Step 12: Now we will give it a tweet to check if it gives good results

```
a='he is very good person '
b=message_to_token_list(a) print(b)
c=message_to_word_vectors(a, word_dict=words)
print(c) l=1
#y = d.to_numpy().astype(int)
 all_word_vector_sequences =
[]



if c.shape[0] == 0:
  c = np.zeros(shape=(1, 50))
   all_word_vector_sequences.append(c)
 d=pad_X(all_word_vector_sequences,
desired_sequence_length=57)
 print(best_model.predict(d))
 t_predictions = (best_model.predict(d) >
0.5).astype(int)
 print('The Sentiment of this tweet is:
',t_predictions)
```

Output:

```
['he', 'is', 'very', 'good', 'person']
1/1 [==============================] - 0s 24ms/step
[[0.9154879]]
1/1 [==============================] - 0s 25ms/step
The Sentiment of this tweet is:  [[1]]
```

Actual: Positive Sentiment

Predicted: Positive Sentiment

```
['he', 'is', 'very', 'bad', 'person']

 1/1 [==============================] - 0s 31ms/step
 [[0.04338082]]
 1/1 [==============================] - 0s 30ms/step
 The Sentiment of this tweet is:  [[0]]
```

Actual: Negative Sentiment

Predicted: Negative Sentiment