LSTM Model Implementation

Architecture

Deep learning text classification model architectures generally consist of the following components connected in sequence:

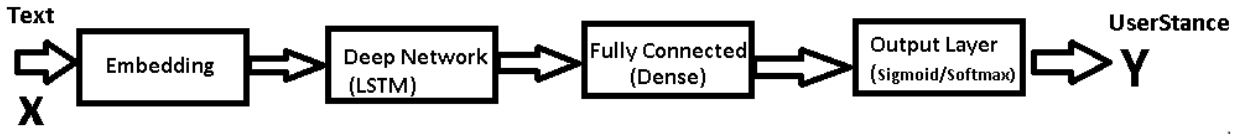


Fig 1 : Deep Learning Architecture

- **Embedding Layer: Word Embedding** is a representation of text where words that have the same meaning have a similar representation. In other words it represents words in a coordinate system where related words, based on a corpus of relationships, are placed closer together. In the deep learning frameworks such as TensorFlow, Keras, this part is usually handled by an **embedding layer** which stores a lookup table to map the words represented by numeric indexes to their dense vector representations.
- **Deep Network :** Deep network takes the sequence of embedding vectors as input and converts them to a compressed representation. The compressed representation effectively captures all the information in the sequence of words in the text. The deep network part is usually an RNN or some forms of it like LSTM/GRU. The dropout is added to overcome the tendency to overfit, a very common problem with RNN based networks.
- **Fully Connected Layer:** The **fully connected layer** takes the deep representation from the RNN/LSTM/GRU and transforms it into the final output

classes or class scores. This component is comprised of fully connected layers along with batch normalization and optionally dropout layers for regularization.

- **Output Layer :** Based on the problem at hand, this layer can have either **Sigmoid** for binary classification or **Softmax** for both binary and multi classification output.

Dataset :

We already have the reddit dataset. I have divided the reddit Brexit full dataset into different periods. For this implementation I have taken all observation which belong to time period (T1) which is in between(16-11-2015 to 24-06-2015) this dataset contains 3367 observations

```
[ ]  df = data[['Author','text','polarization_class','Stance']]
     print(df)
```

```
                 Author  ... Stance
0             rjmlaird  ...      0
1          TotalNewsTV  ...      2
2          TotalNewsTV  ...      2
3          TotalNewsTV  ...      0
4            SeoKungFu  ...      2
...                ...  ...    ...
3362  crappy-throwaway  ...      0
3363           kcergin  ...      0
3364     inside-poland  ...      2
3365     LindaJoyAdams  ...      2
3366             msexm  ...      2

[3367 rows x 4 columns]
```

After the selection of the dataset and required labels, I have balanced the dataset by counting the number of different user belonging to the user stances and data is shuffled.

```
[ ] df.polarization_class.value_counts()
```

```
Neutral    1481
Brexit     1032
Against     854
Name: polarization_class, dtype: int64
```

```
#Balancing classes
num_of_categories = 800 #number of m classes
shuffled = df.reindex(np.random.permutation(df.index))

Neutral = shuffled[shuffled['polarization_class']== 'Neutral'][:num_of_categories]
Brexit  = shuffled[shuffled['polarization_class']== 'Brexit'][:num_of_categories]
Against = shuffled[shuffled['polarization_class']== 'Against'][:num_of_categories]

concated = pd.concat([Neutral,Brexit,Against], ignore_index=True)
#Shuffle the dataset
concated = concated.reindex(np.random.permutation(concated.index))
```

```
[ ] concated.head()
```

	Author	text	polarization_class	Stance
943	GeezMoney	Trade deals and the EU project are very differ...	Brexit	1
2103	epicblob	I don't think "Brexit = Confederate States" is...	Against	0
758	aoide12	Seems like the british people disagree.	Neutral	2
274	insecureguy1786	Is [this](https://www.youtube.com/watch?v=yBkb...	Neutral	2
260	Sunshinelorrypop	They said that there will be no exit polls for...	Neutral	2

- **Learn Word Embedding :** The word embeddings of our dataset can be learned while training a neural network on the classification problem. Before it can be presented to the network, the text data is first encoded so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API provided with Keras. We add padding to make all the vectors of same length. The Embedding layer requires the specification of the vocabulary size(vocab_size), the size of the real-valued vector space Embedding Dimensions = 100, and the maximum length of input documents max_length.

```
Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length)
```

Build Model:

- We are now ready to define our neural network model. The model will use an Embedding layer as the first hidden layer. The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset during training of the model.

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_11 (Embedding)     (None, 1445, 100)         5000000
_________________________________________________________________
lstm_12 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 404
=================================================================
Total params: 5,080,804
Trainable params: 5,080,804
Non-trainable params: 0
_________________________________________________________________
```

- The embedding param count 5000000 = (vocab_size * EMBEDDING_DIM). Maximum input length max_length = 80400. The model during training shall learn the word embeddings from the input text. The total trainable params are 5080804.

Train Model

- Now let us train the model on training set and cross validate on test set. We can see from below training epochs that the model after each epoch is improving the accuracy. After a few epochs we reach

validation accuracy of around 53% which can be improved with large dataset.

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 2068 samples, validate on 1035 samples
Epoch 1/5
2068/2068 [==============================] - 70s 34ms/step - loss: 1.0929 - accuracy: 0.4115 - val_loss: 1.0744 - val_accuracy: 0.4406
Epoch 2/5
2068/2068 [==============================] - 67s 32ms/step - loss: 1.0577 - accuracy: 0.4420 - val_loss: 1.0229 - val_accuracy: 0.4406
Epoch 3/5
2068/2068 [==============================] - 66s 32ms/step - loss: 1.0267 - accuracy: 0.4425 - val_loss: 1.0061 - val_accuracy: 0.4406
Epoch 4/5
2068/2068 [==============================] - 68s 33ms/step - loss: 1.0066 - accuracy: 0.4623 - val_loss: 1.0003 - val_accuracy: 0.5072
Epoch 5/5
2068/2068 [==============================] - 66s 32ms/step - loss: 0.9874 - accuracy: 0.5348 - val_loss: 0.9866 - val_accuracy: 0.5391
```

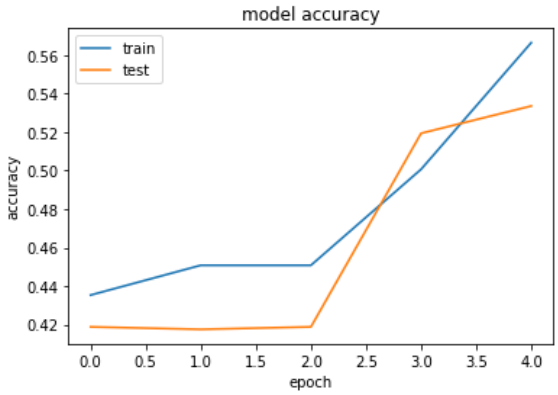
```
print("Testing")
score ,acc= model.evaluate(X[test], Y[test], verbose=0)
print("TrainingLoss:",score)
print("TestAccuracy:",acc)

print("Accuracy: {0:.2%}".format(acc))
```

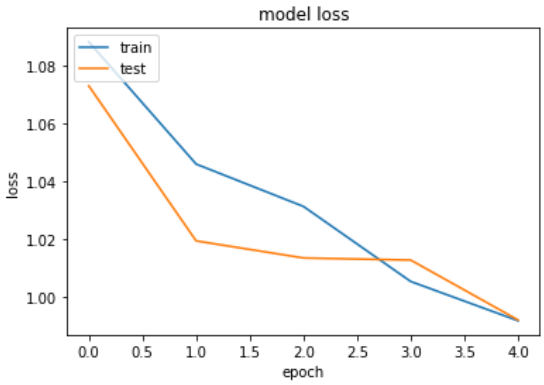
```
Testing
TrainingLoss: 1.1940973690742456
TestAccuracy: 0.5111111402511597
Accuracy: 51.11%
```

Results:

Finally training the classification model on train and validation test set, we get improvement in accuracy with each epoch run. We reach 53.35% accuracy with just around 5 epochs.This can be improved more if try this is on more number of observation in our data

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to a dense Tensor of
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 2327 samples, validate on 776 samples
Epoch 1/5
2327/2327 [==============================] - 71s 30ms/step - loss: 1.0883 - accuracy: 0.4353 - val_loss: 1.0731 - val_accuracy: 0.4188
Epoch 2/5
2327/2327 [==============================] - 70s 30ms/step - loss: 1.0460 - accuracy: 0.4508 - val_loss: 1.0193 - val_accuracy: 0.4175
Epoch 3/5
2327/2327 [==============================] - 70s 30ms/step - loss: 1.0313 - accuracy: 0.4508 - val_loss: 1.0134 - val_accuracy: 0.4188
Epoch 4/5
2327/2327 [==============================] - 69s 30ms/step - loss: 1.0052 - accuracy: 0.5006 - val_loss: 1.0126 - val_accuracy: 0.5193
Epoch 5/5
2327/2327 [==============================] - 71s 31ms/step - loss: 0.9916 - accuracy: 0.5664 - val_loss: 0.9919 - val_accuracy: 0.5335
```



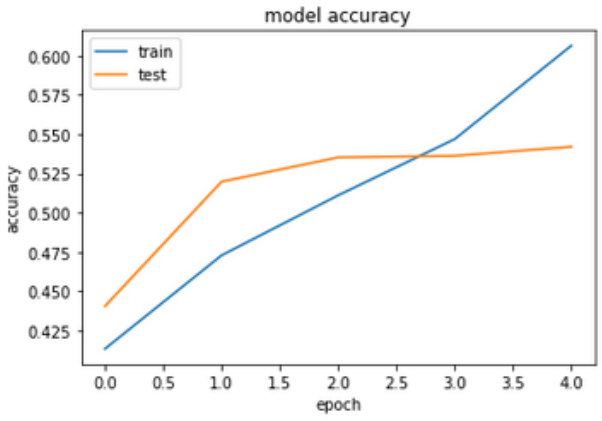


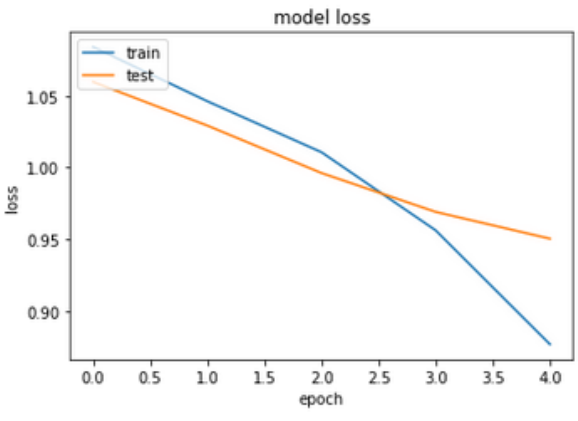
I have tested with different layers and additional parameters with the same dataset and we can see the results are increasing if we add more hyper parameters in our model

Single Convolution Model : We can see that accuracy is increased with the single convolution model which is 54 %

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 1445, 128)         6400000
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1443, 128)         49280
_________________________________________________________________
global_max_pooling1d_2 (Glob (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 387
_________________________________________________________________
activation_4 (Activation)    (None, 3)                 0
=================================================================
Total params: 6,466,179
Trainable params: 6,466,179
Non-trainable params: 0
```

```
Train on 2068 samples, validate on 1035 samples
Epoch 1/5
2068/2068 [==============================] - 24s 12ms/step - loss: 1.0841 - acc: 0.4134 - val_loss: 1.0597 - val_acc: 0.4406
Epoch 2/5
2068/2068 [==============================] - 24s 12ms/step - loss: 1.0462 - acc: 0.4729 - val_loss: 1.0293 - val_acc: 0.5198
Epoch 3/5
2068/2068 [==============================] - 24s 12ms/step - loss: 1.0107 - acc: 0.5111 - val_loss: 0.9962 - val_acc: 0.5353
Epoch 4/5
2068/2068 [==============================] - 24s 12ms/step - loss: 0.9563 - acc: 0.5469 - val_loss: 0.9692 - val_acc: 0.5362
Epoch 5/5
2068/2068 [==============================] - 24s 12ms/step - loss: 0.8770 - acc: 0.6064 - val_loss: 0.9505 - val_acc: 0.5420
```

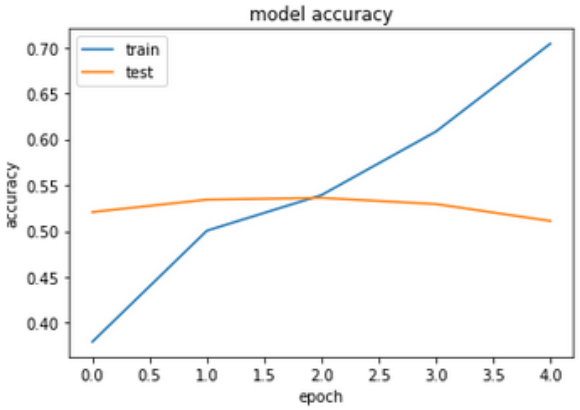


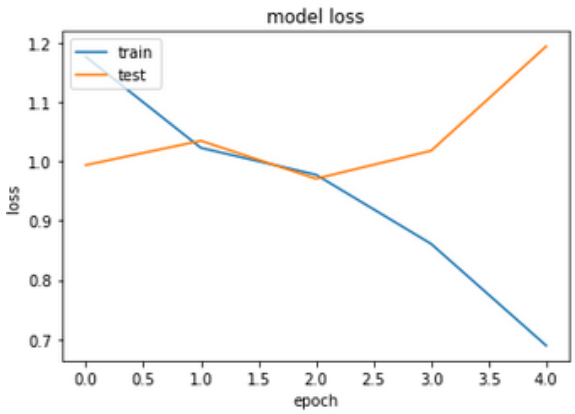


Triple Convolution Model: In this model is not performing well and accuracy is 53%

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 1445, 128)         6400000
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 722, 300)          115500
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 360, 150)          135150
_________________________________________________________________
dropout_5 (Dropout)          (None, 360, 150)          0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 179, 75)           33825
_________________________________________________________________
flatten_1 (Flatten)          (None, 13425)             0
_________________________________________________________________
dropout_6 (Dropout)          (None, 13425)             0
_________________________________________________________________
dense_7 (Dense)              (None, 150)               2013900
_________________________________________________________________
dropout_7 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 453
=================================================================
Total params: 8,698,828
Trainable params: 8,698,828
Non-trainable params: 0
```

```
_________________________________________________________________
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse Inde
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
Train on 2068 samples, validate on 1035 samples
Epoch 1/5
2068/2068 [==============================] - 44s 21ms/step - loss: 1.1765 - acc: 0.3796 - val_loss: 0.9938 - val_acc: 0.5208
Epoch 2/5
2068/2068 [==============================] - 43s 21ms/step - loss: 1.0229 - acc: 0.5005 - val_loss: 1.0348 - val_acc: 0.5343
Epoch 3/5
2068/2068 [==============================] - 43s 21ms/step - loss: 0.9776 - acc: 0.5392 - val_loss: 0.9708 - val_acc: 0.5362
Epoch 4/5
2068/2068 [==============================] - 43s 21ms/step - loss: 0.8610 - acc: 0.6083 - val_loss: 1.0179 - val_acc: 0.5295
Epoch 5/5
2068/2068 [==============================] - 45s 22ms/step - loss: 0.6893 - acc: 0.7041 - val_loss: 1.1941 - val_acc: 0.5111
```

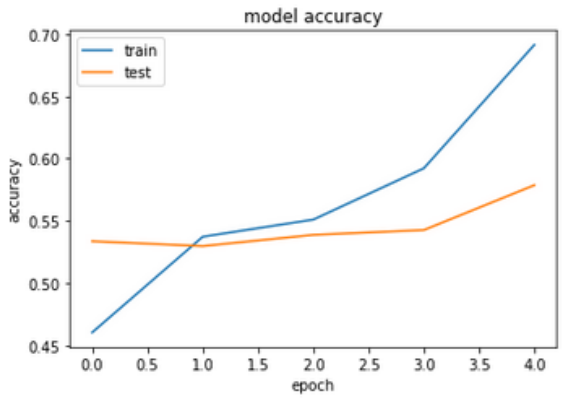


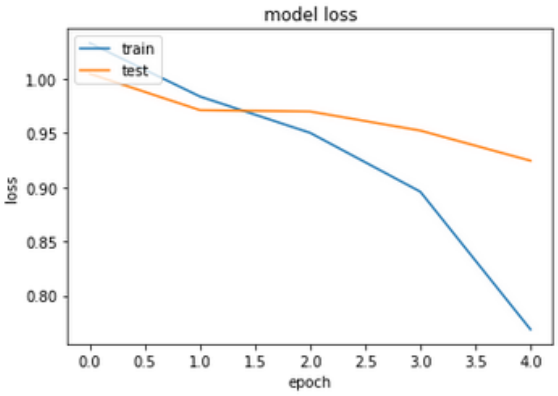


CNN + LSTM : This model perform well than others and its accuracy is increased 57%.

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 1445, 100)         5000000
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 1441, 64)          32064
_________________________________________________________________
dropout_8 (Dropout)          (None, 1441, 64)          0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 360, 64)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               66000
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 303
=================================================================
Total params: 5,098,367
Trainable params: 5,098,367
```

```
Train on 2327 samples, validate on 776 samples
Epoch 1/5
2327/2327 [==============================] - 42s 18ms/step - loss: 1.0330 - accuracy: 0.4602 - val_loss: 1.0047 - val_accuracy: 0.5335
Epoch 2/5
2327/2327 [==============================] - 42s 18ms/step - loss: 0.9837 - accuracy: 0.5372 - val_loss: 0.9711 - val_accuracy: 0.5296
Epoch 3/5
2327/2327 [==============================] - 42s 18ms/step - loss: 0.9502 - accuracy: 0.5509 - val_loss: 0.9699 - val_accuracy: 0.5387
Epoch 4/5
2327/2327 [==============================] - 42s 18ms/step - loss: 0.8959 - accuracy: 0.5922 - val_loss: 0.9523 - val_accuracy: 0.5425
Epoch 5/5
2327/2327 [==============================] - 42s 18ms/step - loss: 0.7689 - accuracy: 0.6914 - val_loss: 0.9245 - val_accuracy: 0.5786
776/776 [==============================] - 3s 4ms/step
```





FUTURE WORK: Try to implement the pretrained model such as word2vec and compare the results after applying word2vec