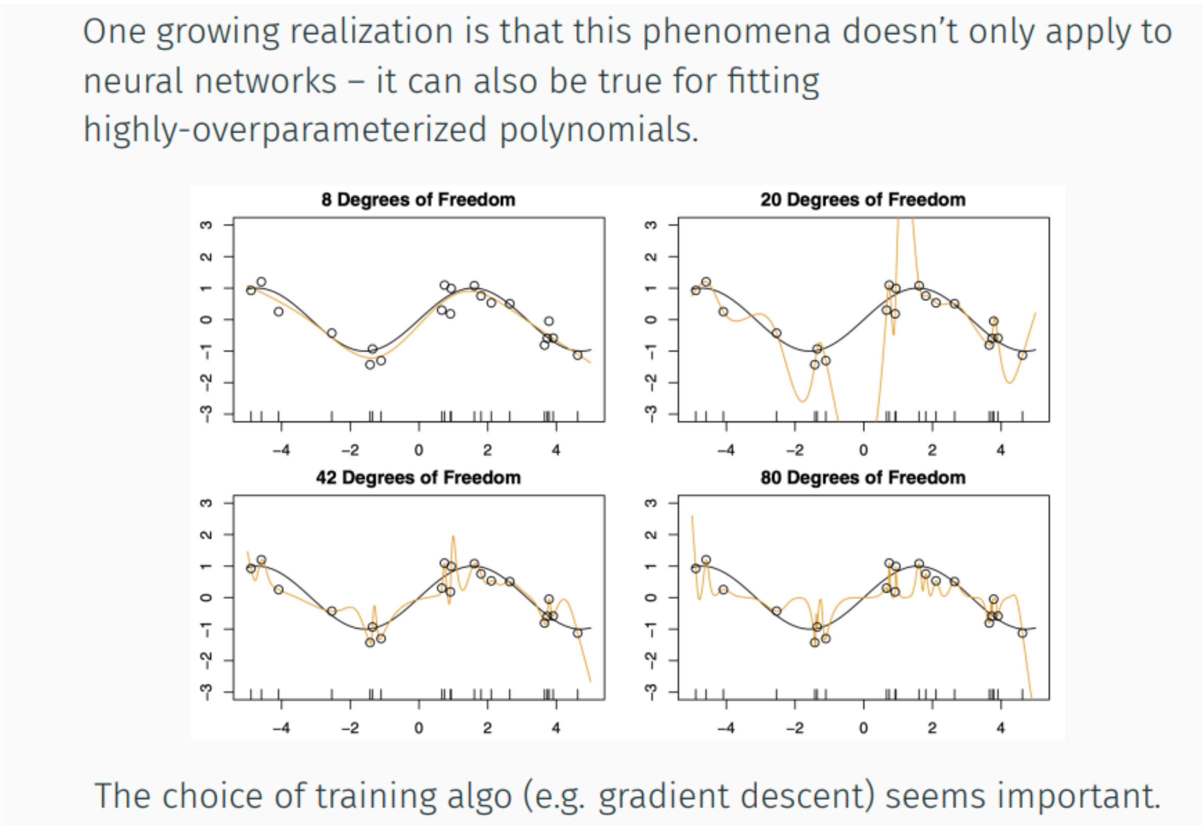
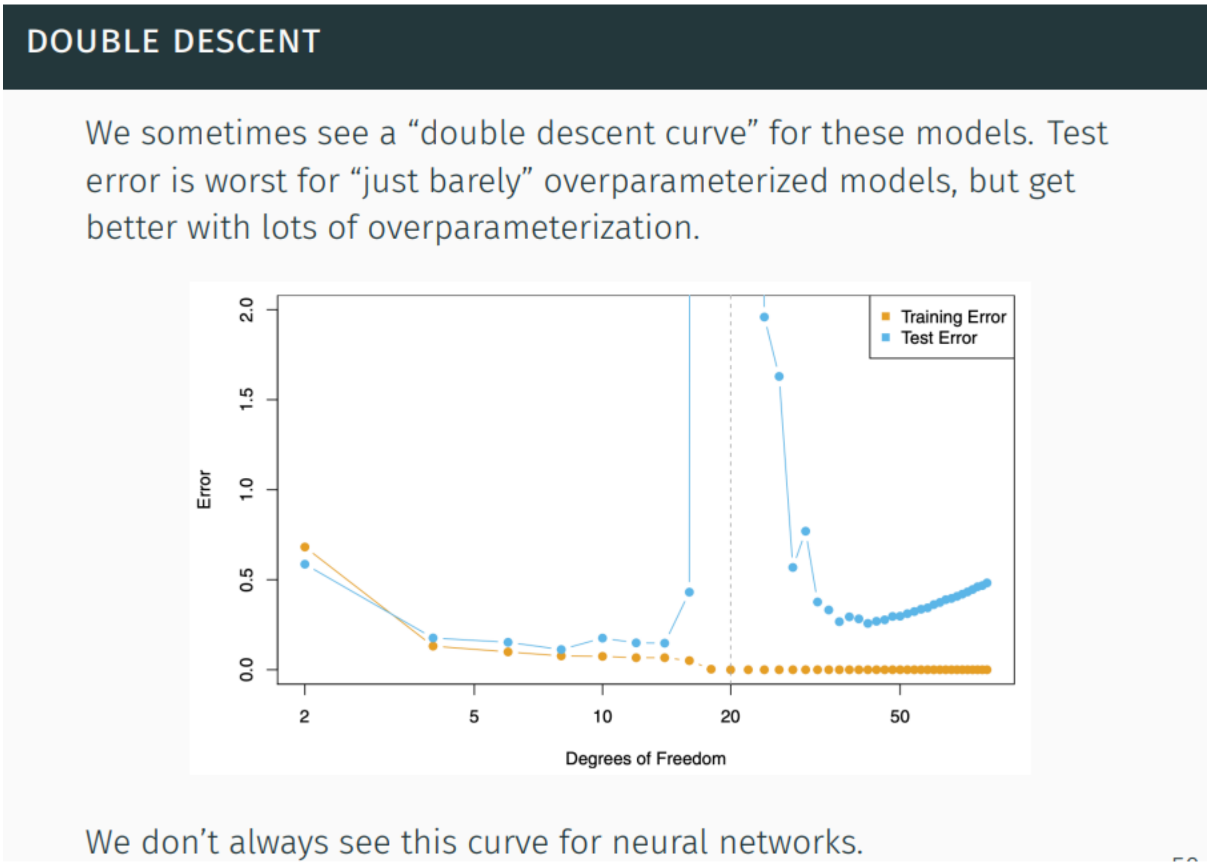
Members: Abhishek Kalra (ak7468), Sri Harsha Tavidisetty Rajendra(st4403), Varshitha Chennamsetti (vc2209)

Exploring the Double Descent phenomenon in Deep Learning Methods using variable architectures, data and model parameters

Introduction:

The observation of deep double descent was stumbled upon when the machine learning community experimented fitting data with more than required parametrized models. Like explained in the lecture the observation was for overparameterized polynomials overfitting was a drawback but that was not really the case for highly-overparameterized polynomials because the generated graph (for example graph 4: 80 degrees of freedom) does not have much loss when compared to graph 2 (20 degrees of freedom).





But this double descent phenomena is not really usual in neural networks, in this project we are trying to experiment with different model architectures and increasing complexities to see if we can observe a double descent. Latest research suggests that apart from increasing complexity double descent can also be tried and observed by trying various other methods like:-

1) Increasing the number of training epochs
2) Increasing the amount of data used for training(By data augmentation)
3) Inducing intentional label noise in the data

In this project we have worked on trying to see the double descent by employing the below approaches:-

1. Increasing Model Complexity by:
    a. Using Different Architectures: Both RNN and CNN models have been employed.
    b. Number of hidden layers
    c. Number of Neurons in the hidden layers
2. Increasing the number of training epochs
3. Increasing the amount of data used for training(Here we did not need data augmentation because we already were working with a huge dataset)

Data and Methods

Dataset Loading and Preparation

Only the tweet text and the polarity label are taken for the training dataset. The tweet text is then cleaned and denoised. This is done to remove any unnecessary symbols and numbers from the text. The polarity label in the features has two classes and they are labeled as 0 for negative polarity and 4 for positive polarity. In order to encode the label, wherever there is 4, the label is changed to 1. The dataset is then split into train and test datasets.

Tokenizing the data

Since the model will only work with numbers, the tweets text needs to be converted into a sequence of numbers. A tokenizer is used on the training dataset which creates a dictionary of words. These indices in the dictionary are used to convert each sentence into a sequence of indices. Furthermore, these sequences are padded with zeros to a maximum length because the neural network is only able to allow a constant shaped input. For the test dataset, the tokenizer that was used to fit the train dataset is used to convert the sentences.

Deep Learning Models

RNN

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. In the current exercise we have implemented LSTM (long short-term memory) RNN model with variable number of neurons and hidden layers

CNN

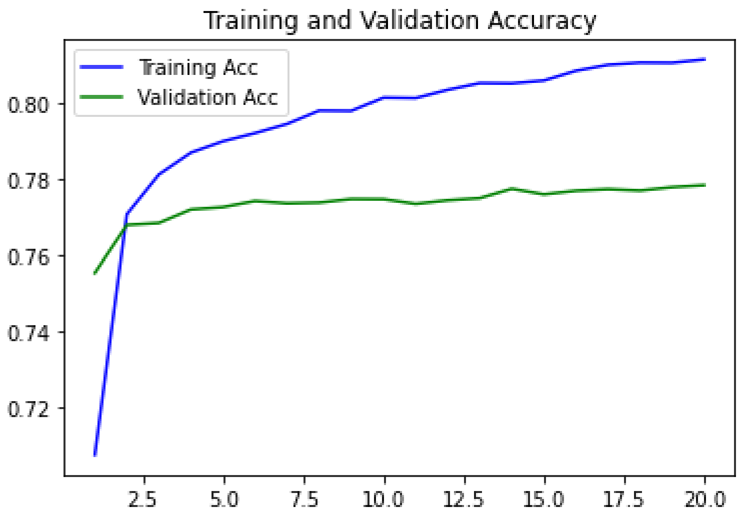
A 1d convolution can be applied to these sequences. The model architecture consists of only one convolution layer, one max pooling layer, a flattening layer and two fully connected layers. The activation function for the final layer is given as sigmoid and binary cross entropy loss is used for minimization.

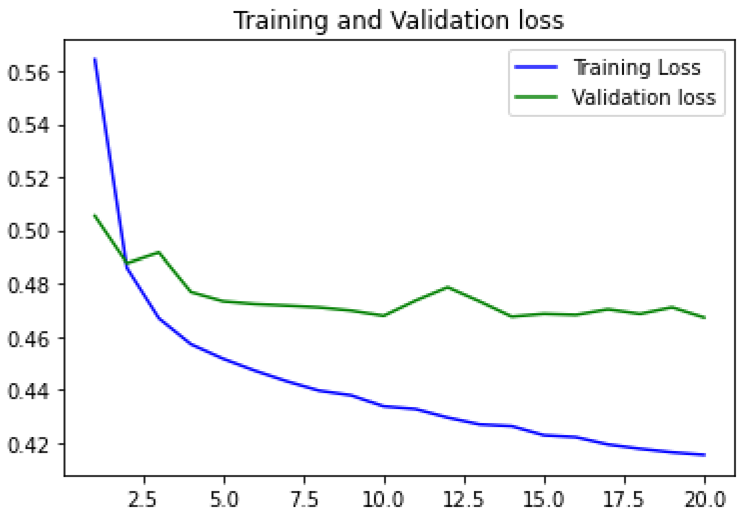
Results and Discussion

The number of data samples (42,880 records) for training dataset are constant across the RNN and CNN models and account for 4% of the total data taken into account for training and testing. The dataset is huge so due to computational limitations and excessive runtime we capped it to 4% which helps us view the results in a reasonable time.

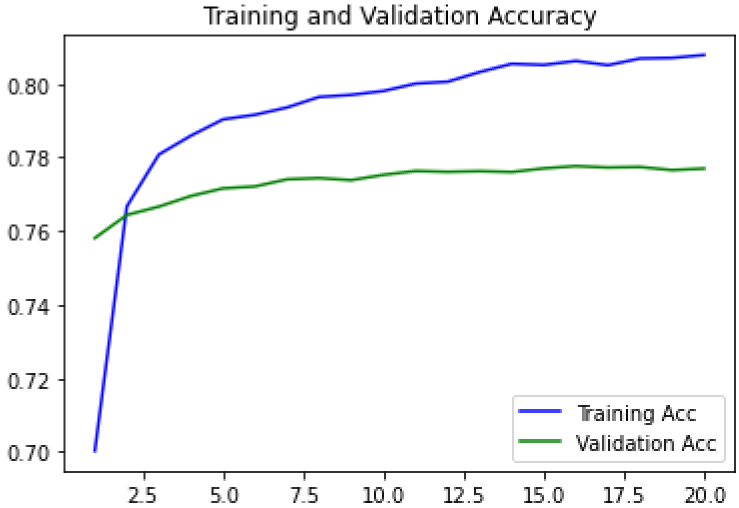
RNN Models

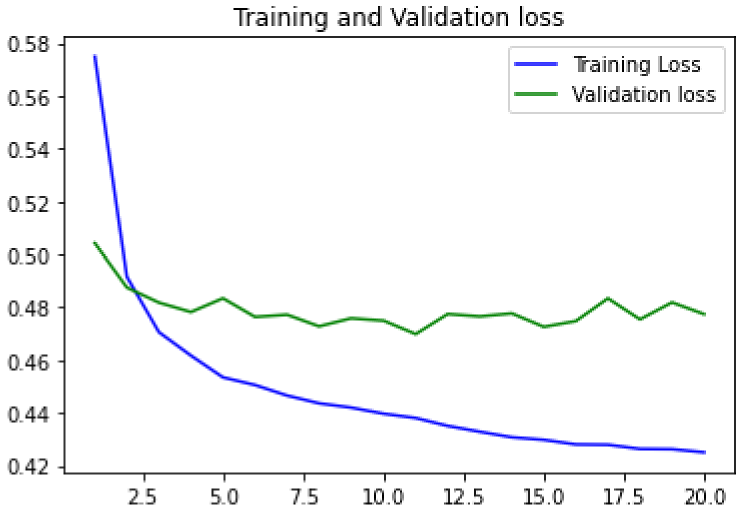
Model 1 Single Layer LSTM
Dataset size: 42880 samples used for training(4% of the original dataset)
Number of epochs: 20



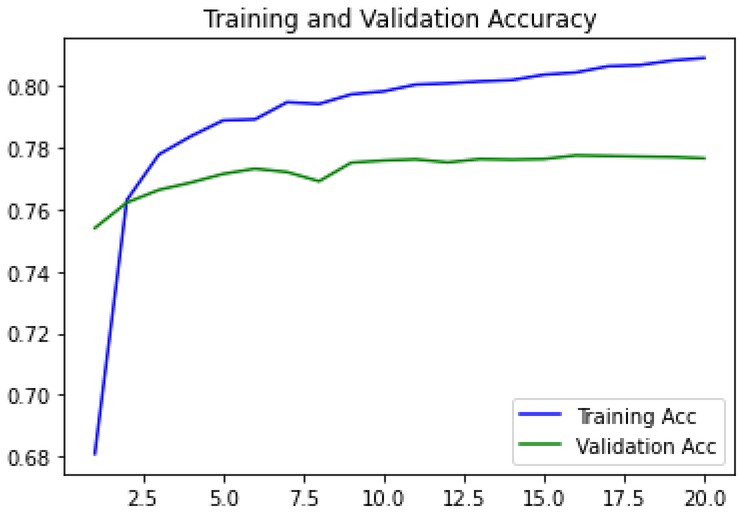


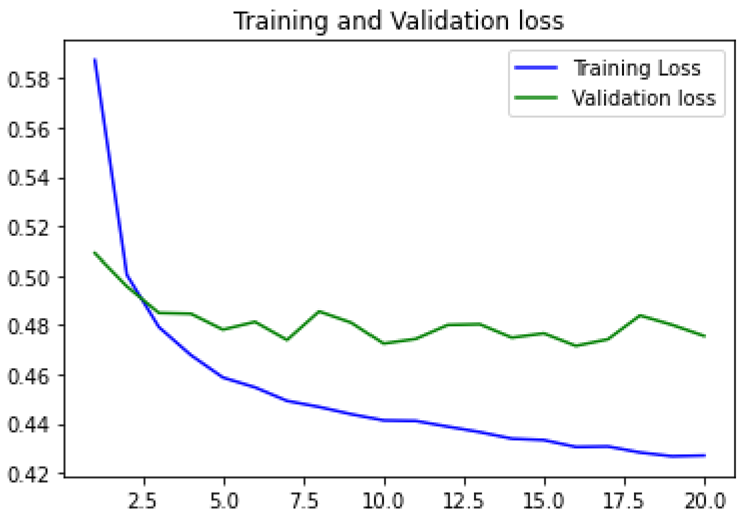
Model 2 Double Layer LSTM
Dataset size: 42880 samples used for training(4% of the original dataset)
Number of epochs: 20





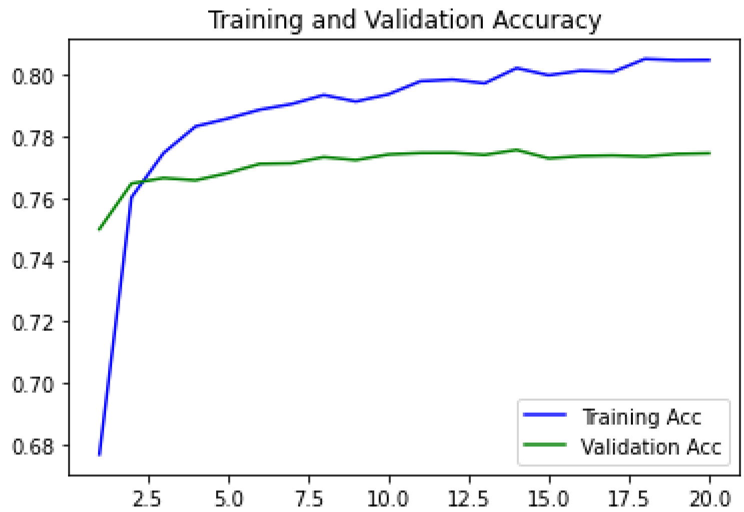
Model 3 Triple Layer LSTM
Dataset size: 42880 samples used for training(4% of the original dataset)
Number of epochs: 20

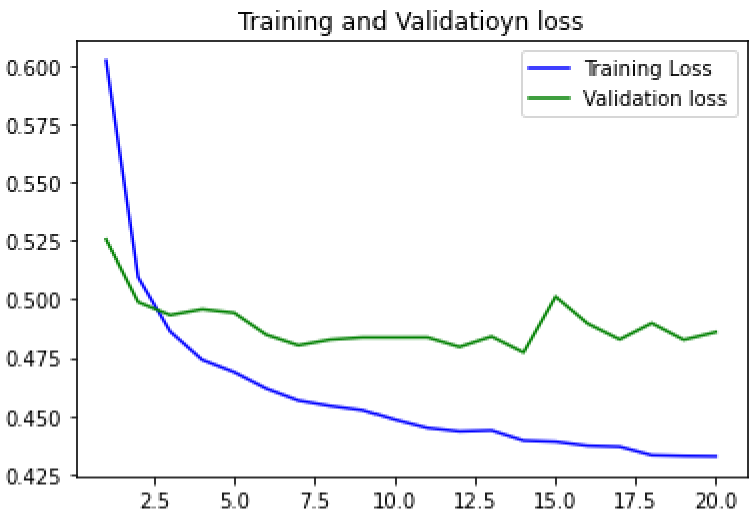

Training and Validation Accuracy
0.80
0.78
0.76
0.74
0.72
0.70
0.68
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training Acc
Validation Acc


Training and Validation loss
0.58
0.56
0.54
0.52
0.50
0.48
0.46
0.44
0.42
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Training Loss
Validation loss

Model 4 Triple Layer LSTM with varying neurons(5,15,25)

5 neurons
Dataset size: 42880 samples used for training(4% of the original dataset)
Number of epochs: 20

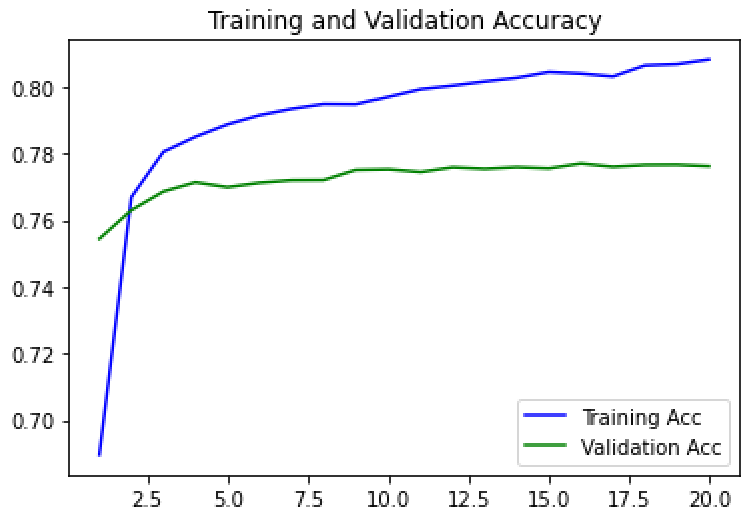


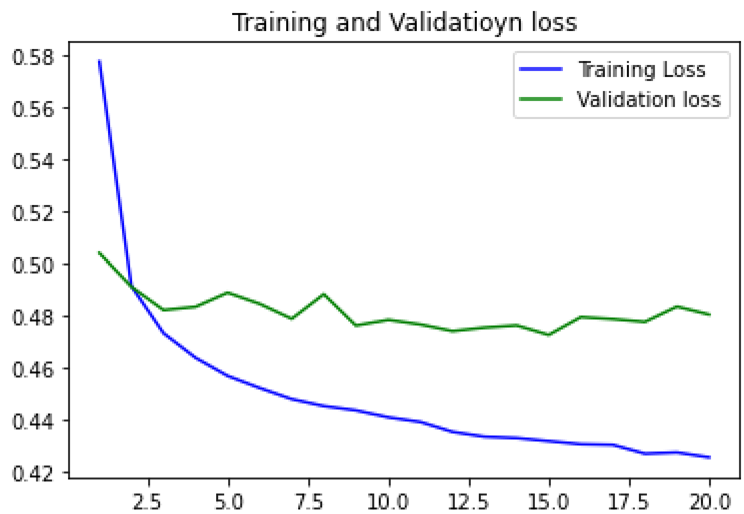


15 neurons

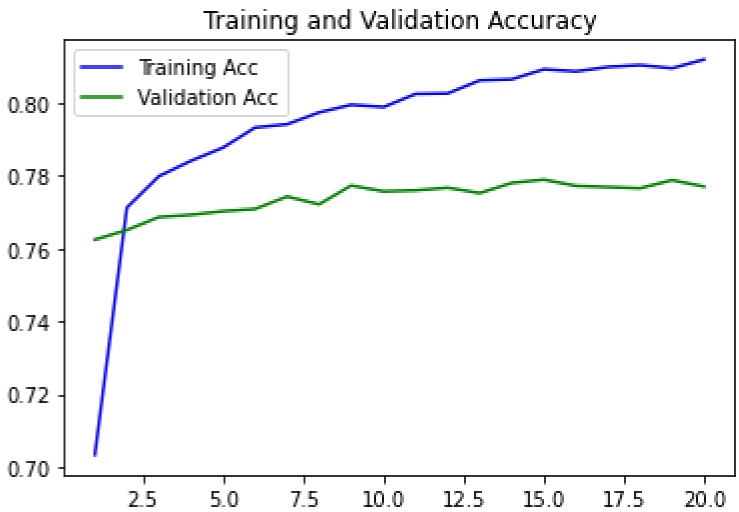
Dataset size: 42880 samples used for training(4% of the original dataset)

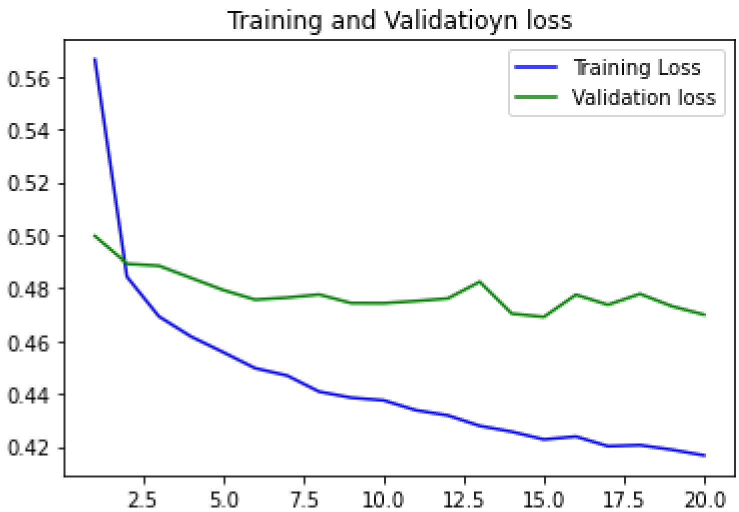
Number of epochs: 20





25 neurons
Dataset size: 42880 samples used for training(4% of the original dataset)
Number of epochs: 20



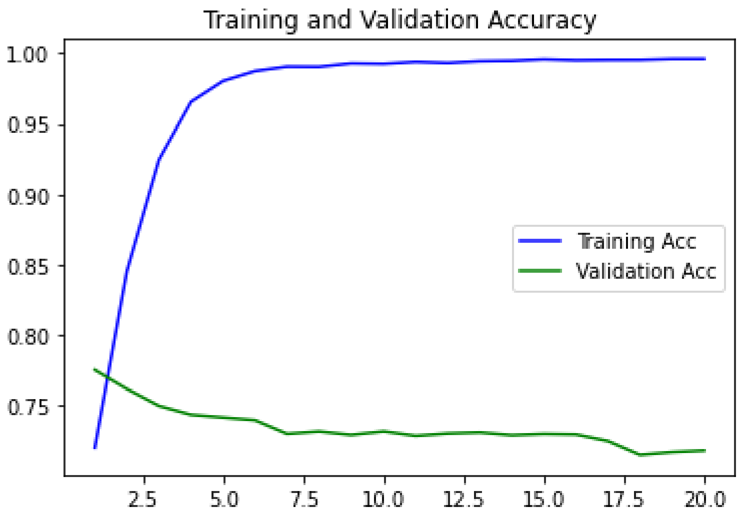


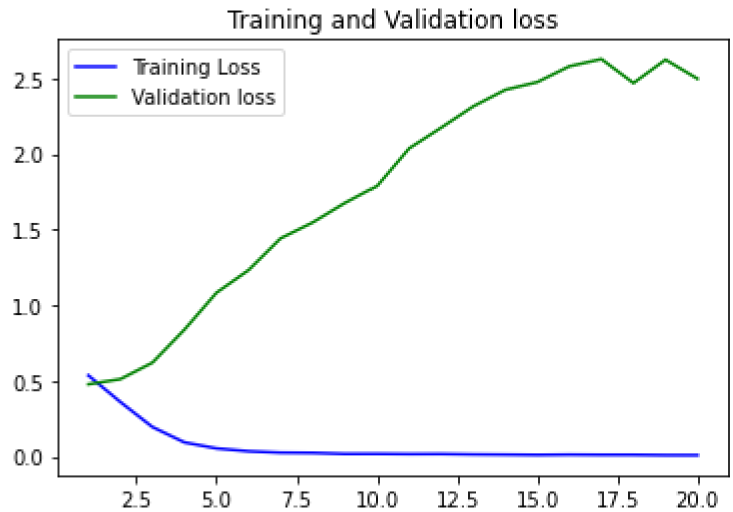
CNN Models

Model 5

Dataset size: 42880 samples used for training(4% of the original dataset)

Number of epochs: 20

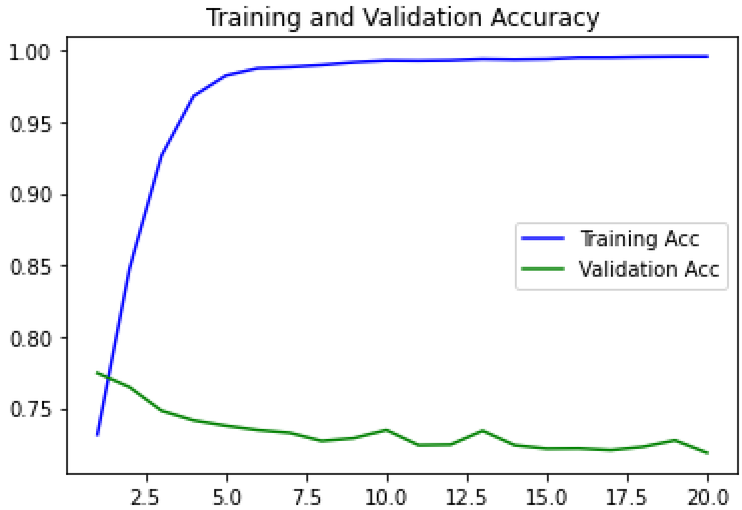


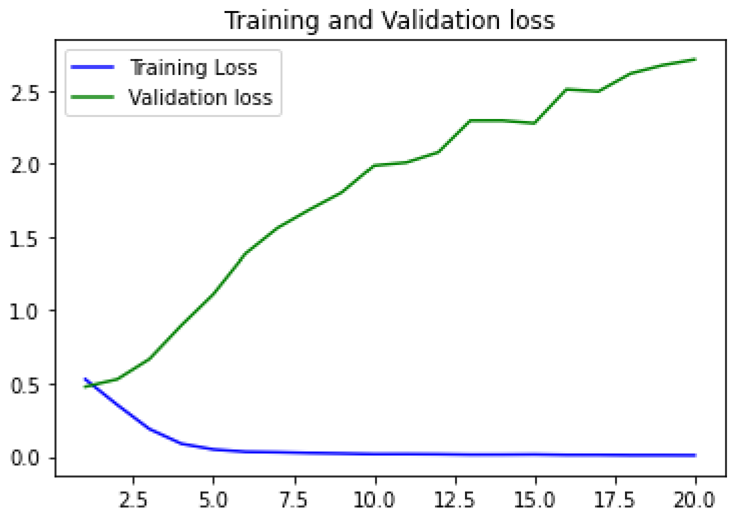


Model 6

Dataset size: 42880 samples used for training(4% of the original dataset)

Number of epochs: 20



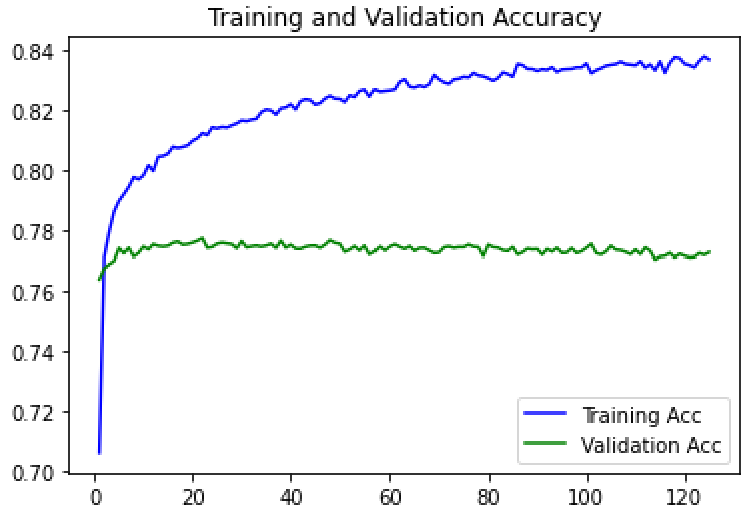


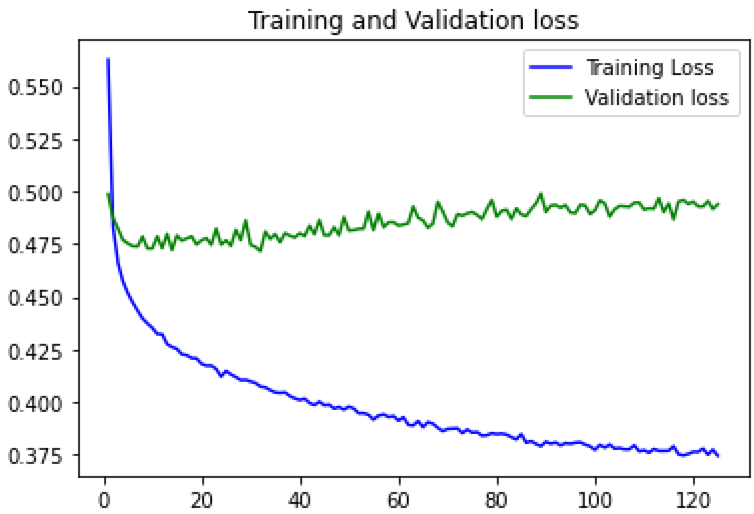
Trying to Observe double descent by training the model for more number of epochs

Model 7 Single Layer LSTM

Dataset size: 42880 samples used for training(4% of the original dataset)

Number of epochs: 125



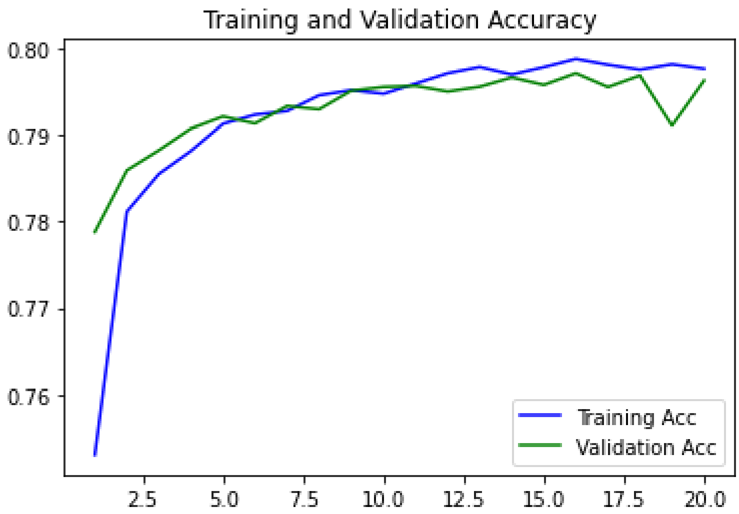


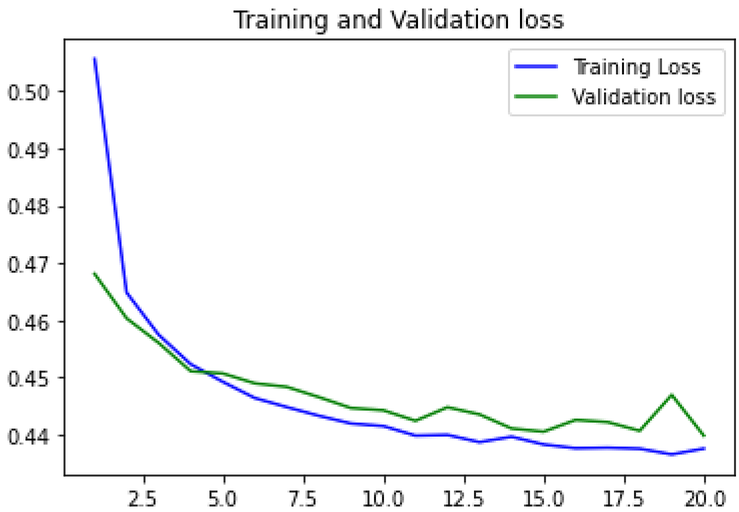
Trying to observe double descent by using more data to train, 4 times the data than what was used previously to train Model 1 LSTM

Model 7 Single Layer LSTM

Dataset size: 171520 samples used for training(16% of the original dataset)

Number of epochs: 20





Conclusion

For most of the models 1- 6, we weren't really able to observe the double descent phenomenon which might be owing to the limited training dataset and epochs. However, for model 7 where we trained it using more data we did observe a slight spike and fall in the test loss which is not really huge though and may warrant further experimentation. In an indeal world with an unlimited amount of compute at our disposal, we would have liked to probe this observation further. However, owing to computational limitations we ended up limiting the number of epochs to 20.

References:

Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2021). Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, *2021*(12), 124003.

https://towardsdatascience.com/an-easy-tutorial-about-sentiment-analysis-with-deep-learning-and-keras-2bf52b9cba91 (Accessed on 5th May 2022)

https://www.tensorflow.org/guide/keras/train_and_evaluate (Accessed on 4th May 2022)

https://www.youtube.com/watch?v=R29awq6jvUw

Import statements

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
import tensorflow_datasets as tfds

import re
from nltk.tokenize.treebank import TreebankWordDetokenizer
from gensim.utils import simple_preprocess

import numpy as np
import matplotlib.pyplot as plt
import pickle
import os
from glob import glob
from tqdm import tqdm
```

Enabling GPU

Because we might be working with huge amounts of data, GPU is used with tensorflow to accelrate the processing. The only drawback with using the GPU on colab is that we can only use it 12 hours at time. We will only be able to use it after another 12 hours after last use.

```
# check if GPU is accessible with tf
tf.config.list_physical_devices()
```

```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
 PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
# command to check GPU status
!nvidia-smi
```

```
Sat May  7 23:27:19 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   37C    P0    26W / 250W |      2MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

```
+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Dataset Loading and Preparation

The dataset we are about to use is the 'Sentiment140' dataset which has information regarding a tweet. And we are going to find the polarity of that tweet using sentiment analysis through neural networks!

```
# Seperation of data into train and test sets
ds_train, ds_train_info = tfds.load('sentiment140', split='train[:4%]', shuffle_files=True ,w
#ds_test, ds_test_info = tfds.load('sentiment140', split='test[:5%]', shuffle_files=True , wi
```

```
Downloading and preparing dataset sentiment140/1.0.0 (download: 77.59 MiB, generated: 3(
```

Dl Completed...: 0/0 [00:00<?, ? url/s]

Dl Size...: 0/0 [00:00<?, ? MiB/s]

Extraction completed...: 0/0 [00:00<?, ? file/s]

1599959/0 [09:32<00:00, 2837.33 examples/s]

```
Shuffling and writing examples to /root/tensorflow_datasets/sentiment140/1.0.0.incomple
```

100% 1599999/1600000 [00:05<00:00, 476577.76 examples/s]

424/0 [00:00<00:00, 2207.69 examples/s]

```
Shuffling and writing examples to /root/tensorflow_datasets/sentiment140/1.0.0.incomple
```

100% 497/498 [00:00<00:00, 17496.66 examples/s]

```
Dataset sentiment140 downloaded and prepared to /root/tensorflow_datasets/sentiment140/1
```

```
# Features of the dataset
for i in ds_train.take(1):
  print(list(i.keys()))
```

```
['date', 'polarity', 'query', 'text', 'user']
```

```
# Taking the first four rows of the dataset and seeing as a dataframe
tfds.as_dataframe(ds_train.take(4))
```

	date	polarity	query	text	user
0	b'Mon Jun 01 18:08:26 PDT 2009'	4	b'NO_QUERY'	b"i'm 10x cooler than all of you! "	b'katie4593'
1	b'Mon Jun 01 23:55:43 PDT 2009'	0	b'NO_QUERY'	b'O.kk? Thats weird I cant stop following people on twitter... I have tons of people to unfollow '	b'migaruler'
2	b'Mon May 04 06:08:51 PDT 2009'	4	b'NO_QUERY'	b'what a beautiful day not to got to my first class '	b'ocean_waves301'

```
# More info regarding the dataset
print(ds_train_info.features)
```

```
FeaturesDict({
    'date': Text(shape=(), dtype=tf.string),
    'polarity': tf.int32,
    'query': Text(shape=(), dtype=tf.string),
    'text': Text(shape=(), dtype=tf.string),
    'user': Text(shape=(), dtype=tf.string),
})
```

For data preparation, only the polarity as the label and the tweet text as the feature is taken for training. The tweet text is cleaned by removing unnecessary symbols. The labels are encoded in order to pass it to the convolution neural network model.

```
# Using it as a numpy array
train_text, train_polarity = tfds.as_numpy(tfds.load('sentiment140', split='train[:4%]', shuf
    as_supervised=True ))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow_datasets/core/
Instructions for updating:
Use `tf.data.Dataset.get_single_element()`.
WARNING:tensorflow:From /usr/local/lib/python3.7/dist-packages/tensorflow_datasets/core/
Instructions for updating:
Use `tf.data.Dataset.get_single_element()`.
```

```
print(type(train_text), train_text.shape)
print(type(train_polarity), train_polarity.shape)
```

```
<class 'numpy.ndarray'> (64000,)
<class 'numpy.ndarray'> (64000,)
```

```
train_text[:10] # First ten tweet records
```

```
array([b"i'm 10x cooler than all of you! ",
```

```
       b'O.kk? Thats weird I cant stop following people on twitter... I have tons of peo
       b'what a beautiful day not to got to my first class ',
       b".@HildyGottlieb &amp; I was just saying to Maha'al yesterday, everything we eve
       b'kinda sad and confused  why do guys do this?',
       b'@Real_DavidCook YES &amp; YES ',
       b"@GDGOfficial But it's another beautiful day here in europe, you have to make th
       b'Working through hundreds of assignments ',
       b'driving with the moonroof and windows open is THE BEST THING IN THE WORLD. Sitt
       b"@scott_mills Gutted! I worked for the fringe last year, won't be back this year
      dtype=object)
```

Data Cleaning and Formatting

Pre-Processing the data to remove non-useful information A) E-mails B) URLs C) Special Characters

```
def clean_data(data):
    data = str(data)
    #Removing URLs with a regular expression
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    data = url_pattern.sub(r'', data)

    # Remove Emails
    data = re.sub('\S*@\S*\s?', '', data)

    # Remove new line characters
    data = re.sub('\s+', ' ', data)

    # Removing the b" "
    data = re.sub("b\'", "", data)
    data = re.sub("b\"", "", data)

    # Remove distracting single quotes and double quotes
    data = re.sub("\'", "", data)
    data = re.sub("\"", "", data)

    return data


# Applying function to numpy
clean_lamb =  lambda data: clean_data(data)
clean_v_func = np.vectorize(clean_lamb)
train_text = clean_v_func(train_text)
```

```
train_text[:10]
```

```
     array(['im 10x cooler than all of you! ',
            'O.kk? Thats weird I cant stop following people on twitter... I have tons of peop
            'what a beautiful day not to got to my first class ',
            '&amp; I was just saying to Mahaal yesterday, everything we ever needed to know w
```

```
       'kinda sad and confused why do guys do this?', 'YES &amp; YES ',
       'But its another beautiful day here in europe, you have to make the most of it Ro
       'Working through hundreds of assignments ',
       'driving with the moonroof and windows open is THE BEST THING IN THE WORLD. Sitti
       'Gutted! I worked for the fringe last year, wont be back this year '],
      dtype='<U1653')
```

Here we are writing a function for converting sentences to words and then back to sentences after removing noise then tokenizing the sentences into words and setting the deacc parameter to True removes punctuations

```
def convert_to_words(sentences):
  list_of_words = simple_preprocess(str(sentences), deacc=True)
  return TreebankWordDetokenizer().detokenize(list_of_words) # Makes words go back to sentenc
```

```
convert_lamb =  lambda data: convert_to_words(data)
convert_v_func = np.vectorize(convert_lamb)
train_text = convert_v_func(train_text)
```

```
train_text[:10]
```

```
array(['im cooler than all of you',
       'kk thats weird cant stop following people on twitter have tons of people to unfo
       'what beautiful day not to got to my first class',
       'amp was just saying to mahaal yesterday everything we ever needed to know was in
       'kinda sad and confused why do guys do this', 'yes amp yes',
       'but its another beautiful day here in europe you have to make the most of it roo
       'working through hundreds of assignments',
       'driving with the moonroof and windows open is the best thing in the world sittin
       'gutted worked for the fringe last year wont be back this year'],
      dtype='<U1141')
```

```
print(np.unique(train_polarity)) # 0 - negative , 4 - positive
Y_Tr = train_polarity
print(Y_Tr[0])
```

```
[0 4]
4
```

```
# Encoding the labels
train_polarity[train_polarity == 4] = 1 # Replacing the values to 1
```

```
train_polarity[90]
```

```
1
```

Split the same dataset into train and validation datasets.

```
# Train - validation spilt
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(train_text, train_polarity, test_size=0.3
```

Model construction and training

The words are tokenized using a keras preprocessing layer. For the model 1D convolution model (is used. It is extremely fast on small datastets and it converges faster.

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

import keras
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense, Conv1D, MaxPooling1D, GlobalMaxPooling1D
from keras import Model, layers
from keras import Input

from keras.callbacks import EarlyStopping
```

Assign numbers to words to use convolutional network on it.

```
max_len=max([len(row.split()) for row in train_text])
print("Maximum length:",max_len)
```

```
Maximum length: 375
```

```
tokenizer = Tokenizer() # Defining a tokenizer
tokenizer.fit_on_texts(X_train) # Applying it to the numpy array of texts
sequences = tokenizer.texts_to_sequences(X_train) # Converting words to sequence of numbers
train_text = pad_sequences(sequences, maxlen=max_len , padding="post") # Padding them for con
```

```
tokenizer.word_index['hit']
```

```
540
```

```
train_text
```

```
array([[ 15,  34, 476, ...,   0,   0,   0],
       [  2, 106,  56, ...,   0,   0,   0],
```

```
       [ 23, 445,  21, ...,   0,   0,   0],
       ...,
       [441, 573, 188, ...,   0,   0,   0],
       [344, 123,  20, ...,   0,   0,   0],
       [460,  30, 356, ...,   0,   0,   0]], dtype=int32)
```

```
test_sequences = tokenizer.texts_to_sequences(X_test) # Converting words to sequence of numbe
test_text = pad_sequences(test_sequences, maxlen=max_len, padding="post") # Padding them for
```

```
test_text.shape
```

```
(21120, 375)
```

```
vocab_size=len(tokenizer.word_index) + 1
print(vocab_size)
```

```
31617
```

```
model= Sequential()
model.add(layers.Embedding(vocab_size, 100, input_length=max_len))
model.add(layers.Conv1D(32, 8, activation="relu"))
model.add(layers.MaxPooling1D(2))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation="relu"))
model.add(layers.Dense(1,  activation="sigmoid"))
```

```
model.summary()
```

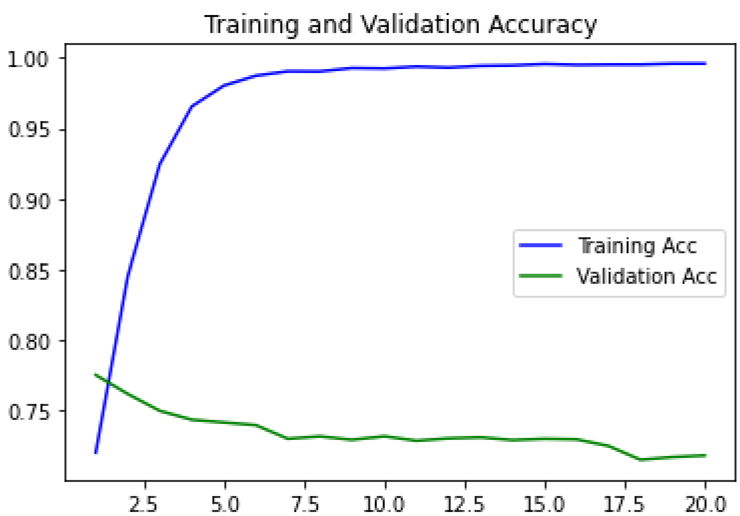
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, 375, 100)          3161700   
                                                                 
 conv1d (Conv1D)             (None, 368, 32)           25632     
                                                                 
 max_pooling1d (MaxPooling1D  (None, 184, 32)          0         
 )                                                               
                                                                 
 flatten (Flatten)           (None, 5888)              0         
                                                                 
 dense (Dense)               (None, 10)                58890     
                                                                 
 dense_1 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 3,246,233
Trainable params: 3,246,233
Non-trainable params: 0
_________________________________________________________________
```

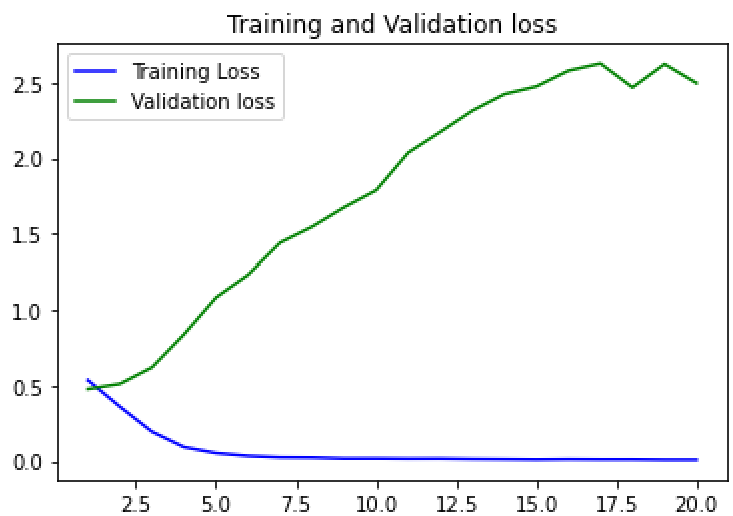
```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
history = model.fit(train_text,y_train, validation_data= ( test_text , y_test),epochs=20)
```

```
Epoch 1/20
1340/1340 [==============================] - 19s 6ms/step - loss: 0.5371 - accuracy: 0.7
Epoch 2/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.3614 - accuracy: 0.84
Epoch 3/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.1962 - accuracy: 0.92
Epoch 4/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0944 - accuracy: 0.96
Epoch 5/20
1340/1340 [==============================] - 7s 6ms/step - loss: 0.0545 - accuracy: 0.98
Epoch 6/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0357 - accuracy: 0.98
Epoch 7/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0265 - accuracy: 0.99
Epoch 8/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0250 - accuracy: 0.99
Epoch 9/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0187 - accuracy: 0.99
Epoch 10/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0188 - accuracy: 0.99
Epoch 11/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0173 - accuracy: 0.99
Epoch 12/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0177 - accuracy: 0.99
Epoch 13/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0149 - accuracy: 0.99
Epoch 14/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0135 - accuracy: 0.99
Epoch 15/20
1340/1340 [==============================] - 7s 6ms/step - loss: 0.0116 - accuracy: 0.99
Epoch 16/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0134 - accuracy: 0.99
Epoch 17/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0122 - accuracy: 0.99
Epoch 18/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0121 - accuracy: 0.99
Epoch 19/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0098 - accuracy: 0.99
Epoch 20/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0096 - accuracy: 0.99
```

```
# plotting the results

acc = history.history.get('accuracy')
val_acc = history.history.get('val_accuracy')
loss = history.history.get('loss')
val_loss = history.history.get('val_loss')

epochs = range(1, 21)
```

```
plt.plot(epochs, acc, 'b', label='Training Acc')
plt.plot(epochs, val_acc, 'g', label='Validation Acc')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label="Training Loss")
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```





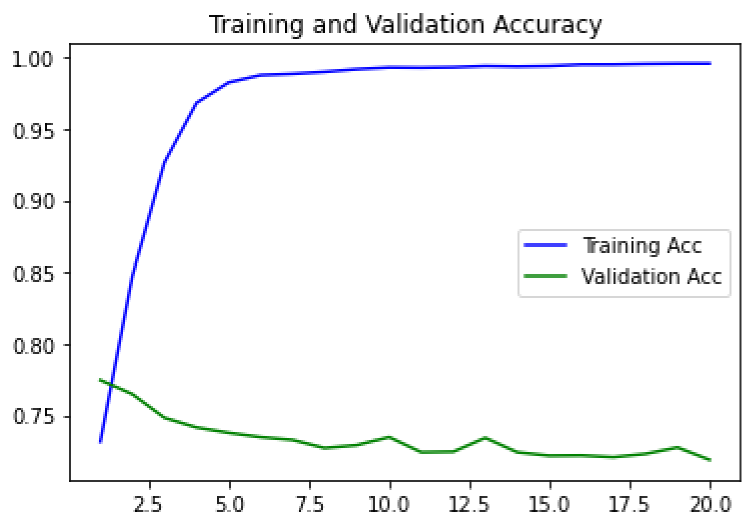

```
from keras import regularizers
from keras.callbacks import ModelCheckpoint
model= Sequential()
model.add(layers.Embedding(vocab_size, 100, input_length=max_len))
model.add(layers.Conv1D(32, 8, activation="relu"))
model.add(layers.MaxPooling1D(2))
model.add(layers.Flatten())
model.add(layers.Dense(10, activation="relu"))
model.add(layers.Dense(1,  activation="sigmoid"))
```

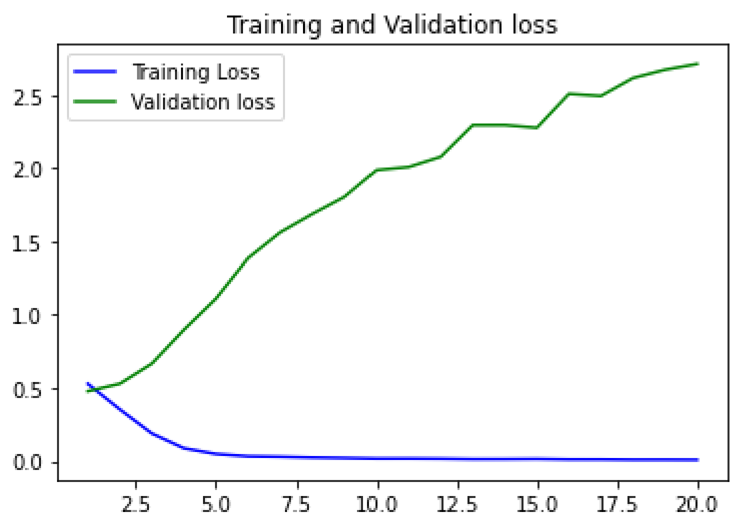
```
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
#modelHistory = model.fit(train_text,y_train, validation_data= ( test_text , y_test),epochs=1
history = model.fit(train_text, y_train, epochs=20,validation_data=(test_text, y_test))
```

```
Epoch 1/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.5290 - accuracy: 0.73
Epoch 2/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.3540 - accuracy: 0.84
Epoch 3/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.1893 - accuracy: 0.92
Epoch 4/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0884 - accuracy: 0.96
Epoch 5/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0493 - accuracy: 0.98
Epoch 6/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0335 - accuracy: 0.98
Epoch 7/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0303 - accuracy: 0.98
Epoch 8/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0251 - accuracy: 0.99
Epoch 9/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0217 - accuracy: 0.99
Epoch 10/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0178 - accuracy: 0.99
Epoch 11/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0179 - accuracy: 0.99
Epoch 12/20
1340/1340 [==============================] - 7s 6ms/step - loss: 0.0168 - accuracy: 0.99
Epoch 13/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0140 - accuracy: 0.99
Epoch 14/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0142 - accuracy: 0.99
Epoch 15/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0153 - accuracy: 0.99
Epoch 16/20
1340/1340 [==============================] - 8s 6ms/step - loss: 0.0122 - accuracy: 0.99
Epoch 17/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0121 - accuracy: 0.99
Epoch 18/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0101 - accuracy: 0.99
Epoch 19/20
1340/1340 [==============================] - 7s 6ms/step - loss: 0.0101 - accuracy: 0.99
Epoch 20/20
1340/1340 [==============================] - 7s 5ms/step - loss: 0.0092 - accuracy: 0.99
```

```
# plotting the results

acc = history.history.get('accuracy')
val_acc = history.history.get('val_accuracy')
loss = history.history.get('loss')
val_loss = history.history.get('val_loss')

epochs = range(1, len(acc)+1)
```

```
plt.plot(epochs, acc, 'b', label='Training Acc')
plt.plot(epochs, val_acc, 'g', label='Validation Acc')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label="Training Loss")
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```





Setting up the RNN Model

RNN models have been implemented using sequential models from the Keras API. Differential Model complexity is achieved by varying the following: A) Number of hidden layers B) Number of Neuron units in each layer Essentially, I'll start

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

```
from keras.preprocessing.sequence import pad_sequences
from keras import regularizers
train_text, train_polarity = tfds.as_numpy(tfds.load('sentiment140', split='train[:4%]', shuf
    as_supervised=True ))
train_text = clean_v_func(train_text)
train_text = convert_v_func(train_text)
max_words = 5000
#max_len = 200
max_len=max([len(row.split()) for row in train_text])

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_text)
sequences = tokenizer.texts_to_sequences(train_text)
tweets = pad_sequences(sequences, maxlen=max_len)
```

```
from keras.layers import Embedding
embedding_layer = Embedding(1000, 64)
```

```
labels = train_polarity

y = []
for i in range(len(labels)):
    if labels[i] == 0:
        y.append(0)
    if labels[i] == 4:
        y.append(1)

y = np.array(y)

labels = tf.keras.utils.to_categorical(y, dtype="int32")
del y
print (labels[:6])
```

```
[[0 1]
 [1 0]
 [0 1]
 [0 1]
 [1 0]
 [0 1]]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tweets,labels, test_size=0.33, random_sta
print ((X_train.shape),(X_test.shape),(y_train.shape),(y_test.shape))
```

```
(42880, 375) (21120, 375) (42880, 2) (21120, 2)
```

```
from keras.models import Sequential
from keras import layers
```

```
from keras import regularizers
from keras import backend as K
from keras.callbacks import ModelCheckpoint
```

Model1:

A single Layer LSTM

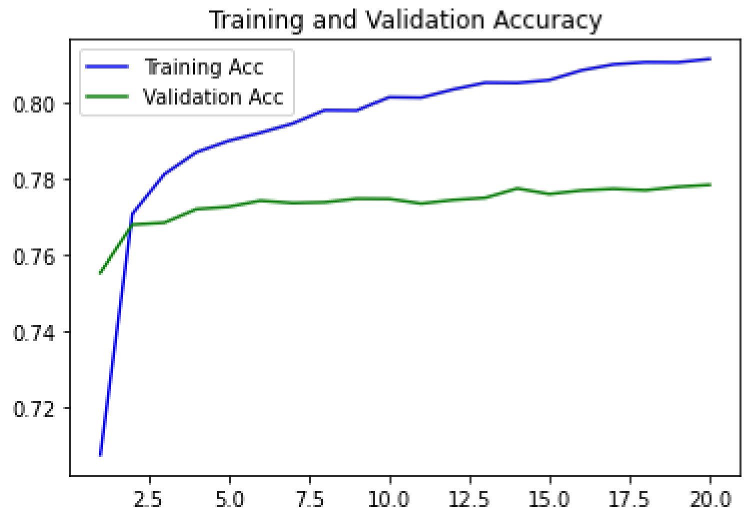
```
model1 = Sequential()
model1.add(layers.Embedding(max_words, 20,input_length=max_len))
model1.add(layers.LSTM(15,dropout=0.5))
model1.add(layers.Dense(2,activation='softmax'))

model1.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['accuracy'])
#Implementing model checkpoins to save the best metric and do not lose it on training.
checkpoint1 = ModelCheckpoint("best_model1.hdf5", monitor='val_accuracy', verbose=1,save_best
history = model1.fit(X_train, y_train, epochs=20,validation_data=(X_test, y_test),callbacks=[
```

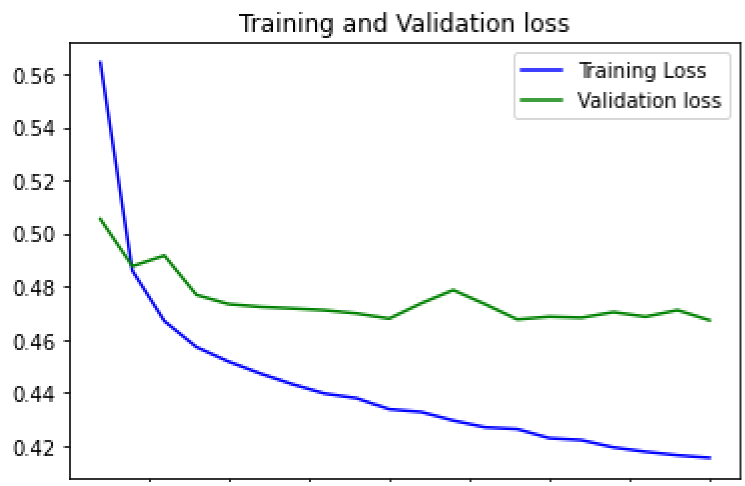
```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify
Epoch 1/20
1338/1340 [============================>.] - ETA: 0s - loss: 0.5645 - accuracy: 0.70
Epoch 1: val_accuracy improved from -inf to 0.75516, saving model to best_model1.hdf
1340/1340 [==============================] - 27s 18ms/step - loss: 0.5643 - accuracy
Epoch 2/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4859 - accuracy: 0.77
Epoch 2: val_accuracy improved from 0.75516 to 0.76771, saving model to best_model1.
1340/1340 [==============================] - 25s 19ms/step - loss: 0.4859 - accuracy
Epoch 3/20
1337/1340 [============================>.] - ETA: 0s - loss: 0.4669 - accuracy: 0.78
Epoch 3: val_accuracy improved from 0.76771 to 0.76823, saving model to best_model1.
1340/1340 [==============================] - 25s 19ms/step - loss: 0.4669 - accuracy
Epoch 4/20
1337/1340 [============================>.] - ETA: 0s - loss: 0.4571 - accuracy: 0.78
Epoch 4: val_accuracy improved from 0.76823 to 0.77178, saving model to best_model1.
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4571 - accuracy
Epoch 5/20
1340/1340 [==============================] - ETA: 0s - loss: 0.4517 - accuracy: 0.78
Epoch 5: val_accuracy improved from 0.77178 to 0.77240, saving model to best_model1.
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4517 - accuracy
Epoch 6/20
1337/1340 [============================>.] - ETA: 0s - loss: 0.4470 - accuracy: 0.79
Epoch 6: val_accuracy improved from 0.77240 to 0.77401, saving model to best_model1.
1340/1340 [==============================] - 25s 19ms/step - loss: 0.4472 - accuracy
Epoch 7/20
1338/1340 [============================>.] - ETA: 0s - loss: 0.4432 - accuracy: 0.79
Epoch 7: val_accuracy did not improve from 0.77401
1340/1340 [==============================] - 25s 18ms/step - loss: 0.4431 - accuracy
Epoch 8/20
1337/1340 [============================>.] - ETA: 0s - loss: 0.4395 - accuracy: 0.79
Epoch 8: val_accuracy did not improve from 0.77401
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4396 - accuracy
```

```
Epoch 9/20
1340/1340 [==============================] - ETA: 0s - loss: 0.4379 - accuracy: 0.79
Epoch 9: val_accuracy improved from 0.77401 to 0.77453, saving model to best_model1.
1340/1340 [==============================] - 25s 19ms/step - loss: 0.4379 - accuracy
Epoch 10/20
1337/1340 [============================>.] - ETA: 0s - loss: 0.4336 - accuracy: 0.80
Epoch 10: val_accuracy did not improve from 0.77453
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4337 - accuracy
Epoch 11/20
1340/1340 [==============================] - ETA: 0s - loss: 0.4327 - accuracy: 0.80
Epoch 11: val_accuracy did not improve from 0.77453
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4327 - accuracy
Epoch 12/20
1338/1340 [============================>.] - ETA: 0s - loss: 0.4296 - accuracy: 0.80
Epoch 12: val_accuracy did not improve from 0.77453
1340/1340 [==============================] - 26s 20ms/step - loss: 0.4295 - accuracy
Epoch 13/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4269 - accuracy: 0.80
Epoch 13: val_accuracy improved from 0.77453 to 0.77476, saving model to best_model1
1340/1340 [==============================] - 26s 20ms/step - loss: 0.4269 - accuracy
Epoch 14/20
1337/1340 [============================>.] - ETA: 0s - loss: 0.4262 - accuracy: 0.80
```

```
# plotting the results

acc = history.history.get('accuracy')
val_acc = history.history.get('val_accuracy')
loss = history.history.get('loss')
val_loss = history.history.get('val_loss')
print(acc)
print(loss)
print(val_loss)
epochs = range(1, 21)
print (epochs)

plt.plot(epochs, acc, 'b', label='Training Acc')
plt.plot(epochs, val_acc, 'g', label='Validation Acc')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label="Training Loss")
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

```
[0.7072761058807373, 0.77052241563797, 0.7809934616088867, 0.7867537140846252, 0.7897388
[0.5643373131752014, 0.485925555229187, 0.4669242203235626, 0.457147002201538, 0.451696
[0.5053774118423462, 0.48760607838630676, 0.4916101098060608, 0.47681108117103577, 0.473
range(1, 21)
```





Model 2:

A double Layered LSTM

```
model2 = Sequential()
model2.add(layers.Embedding(max_words, 20,input_length=max_len))
model2.add(layers.LSTM(15,dropout=0.5,return_sequences = True))
model2.add(layers.LSTM(units = 15,dropout=0.5, return_sequences = False))
model2.add(layers.Dense(2,activation='softmax'))

model2.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['accuracy'])
#Implementing model checkpoins to save the best metric and do not lose it on training.
checkpoint1 = ModelCheckpoint("best_model2.hdf5", monitor='val_accuracy', verbose=1,save_best
history = model2.fit(X_train, y_train, epochs=20,validation_data=(X_test, y_test),callbacks=[
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specif
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specif
Epoch 1/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.5748 - accuracy: 0.70
```

```
Epoch 1: val_accuracy improved from -inf to 0.75805, saving model to best_model2.hdf
1340/1340 [==============================] - 53s 38ms/step - loss: 0.5749 - accuracy
Epoch 2/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4915 - accuracy: 0.76
Epoch 2: val_accuracy improved from 0.75805 to 0.76435, saving model to best_model2.
1340/1340 [==============================] - 48s 36ms/step - loss: 0.4916 - accuracy
Epoch 3/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4706 - accuracy: 0.78
Epoch 3: val_accuracy improved from 0.76435 to 0.76662, saving model to best_model2.
1340/1340 [==============================] - 49s 37ms/step - loss: 0.4706 - accuracy
Epoch 4/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4618 - accuracy: 0.78
Epoch 4: val_accuracy improved from 0.76662 to 0.76951, saving model to best_model2.
1340/1340 [==============================] - 51s 38ms/step - loss: 0.4618 - accuracy
Epoch 5/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4536 - accuracy: 0.79
Epoch 5: val_accuracy improved from 0.76951 to 0.77159, saving model to best_model2.
1340/1340 [==============================] - 52s 39ms/step - loss: 0.4535 - accuracy
Epoch 6/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4506 - accuracy: 0.79
Epoch 6: val_accuracy improved from 0.77159 to 0.77211, saving model to best_model2.
1340/1340 [==============================] - 51s 38ms/step - loss: 0.4506 - accuracy
Epoch 7/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4466 - accuracy: 0.79
Epoch 7: val_accuracy improved from 0.77211 to 0.77405, saving model to best_model2.
1340/1340 [==============================] - 52s 39ms/step - loss: 0.4466 - accuracy
Epoch 8/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4437 - accuracy: 0.79
Epoch 8: val_accuracy improved from 0.77405 to 0.77438, saving model to best_model2.
1340/1340 [==============================] - 52s 39ms/step - loss: 0.4437 - accuracy
Epoch 9/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4420 - accuracy: 0.79
Epoch 9: val_accuracy did not improve from 0.77438
1340/1340 [==============================] - 52s 39ms/step - loss: 0.4421 - accuracy
Epoch 10/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4396 - accuracy: 0.79
Epoch 10: val_accuracy improved from 0.77438 to 0.77528, saving model to best_model2
1340/1340 [==============================] - 52s 39ms/step - loss: 0.4397 - accuracy
Epoch 11/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4381 - accuracy: 0.80
Epoch 11: val_accuracy improved from 0.77528 to 0.77633, saving model to best_model2
1340/1340 [==============================] - 53s 39ms/step - loss: 0.4381 - accuracy
Epoch 12/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4352 - accuracy: 0.80
Epoch 12: val_accuracy did not improve from 0.77633
1340/1340 [==============================] - 53s 39ms/step - loss: 0.4351 - accuracy
Epoch 13/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4330 - accuracy: 0.80
Epoch 13: val_accuracy did not improve from 0.77633
1340/1340 [==============================] - 53s 39ms/step - loss: 0.4329 - accuracy
Epoch 14/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4308 - accuracy: 0.80
```

```
# plotting the results
```

```
acc = history.history.get('accuracy')
val_acc = history.history.get('val_accuracy')
loss = history.history.get('loss')
val_loss = history.history.get('val_loss')
print(acc)
print(loss)
print(val_loss)
epochs = range(1, 21)
print (epochs)

plt.plot(epochs, acc, 'b', label='Training Acc')
plt.plot(epochs, val_acc, 'g', label='Validation Acc')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label="Training Loss")
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

```
[0.7002565264701843, 0.7666744589805603, 0.7808068990707397, 0.785937488079071, 0.790368
[0.5748934745788574, 0.4916088581085205, 0.47063905000686646, 0.4617633521556854, 0.4535
[0.5042362809181213, 0.48754119873046875, 0.4818371534347534, 0.4782625138759613, 0.4834
range(1, 21)
```

Model 3:

A triple Layered LSTM

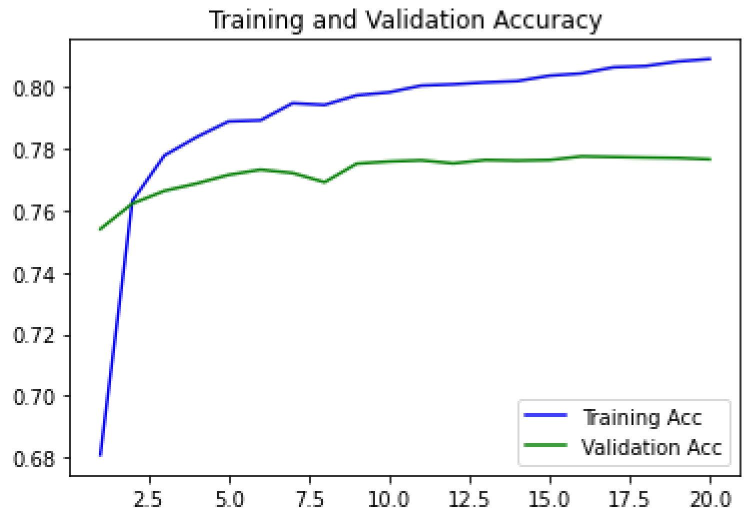
```
model3 = Sequential()
model3.add(layers.Embedding(max_words, 20,input_length=max_len))
model3.add(layers.LSTM(15,dropout=0.5,return_sequences = True))
model3.add(layers.LSTM(units = 15,dropout=0.5, return_sequences = True))
model3.add(layers.LSTM(units = 15,dropout=0.5, return_sequences = False))
model3.add(layers.Dense(2,activation='softmax'))

model3.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['accuracy'])
#Implementing model checkpoins to save the best metric and do not lose it on training.
checkpoint1 = ModelCheckpoint("best_model3.hdf5", monitor='val_accuracy', verbose=1,save_best
history = model3.fit(X_train, y_train, epochs=20,validation_data=(X_test, y_test),callbacks=[
```

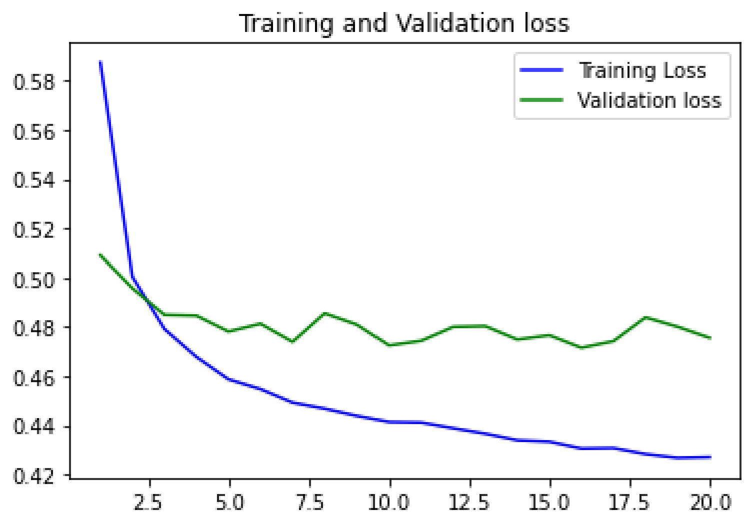
```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify
Epoch 1/20
1340/1340 [==============================] - ETA: 0s - loss: 0.5875 - accuracy: 0.68
Epoch 1: val_accuracy improved from -inf to 0.75388, saving model to best_model3.hdf
1340/1340 [==============================] - 77s 54ms/step - loss: 0.5875 - accuracy
Epoch 2/20
1340/1340 [==============================] - ETA: 0s - loss: 0.5004 - accuracy: 0.76
Epoch 2: val_accuracy improved from 0.75388 to 0.76212, saving model to best_model3.
1340/1340 [==============================] - 72s 54ms/step - loss: 0.5004 - accuracy
Epoch 3/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4792 - accuracy: 0.77
Epoch 3: val_accuracy improved from 0.76212 to 0.76624, saving model to best_model3.
1340/1340 [==============================] - 72s 54ms/step - loss: 0.4792 - accuracy
Epoch 4/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4678 - accuracy: 0.78
Epoch 4: val_accuracy improved from 0.76624 to 0.76861, saving model to best_model3.
1340/1340 [==============================] - 72s 54ms/step - loss: 0.4678 - accuracy
Epoch 5/20
1340/1340 [==============================] - ETA: 0s - loss: 0.4587 - accuracy: 0.78
Epoch 5: val_accuracy improved from 0.76861 to 0.77140, saving model to best_model3.
1340/1340 [==============================] - 72s 54ms/step - loss: 0.4587 - accuracy
Epoch 6/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4548 - accuracy: 0.78
Epoch 6: val_accuracy improved from 0.77140 to 0.77311, saving model to best_model3.
1340/1340 [==============================] - 80s 60ms/step - loss: 0.4547 - accuracy
Epoch 7/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4491 - accuracy: 0.79
Epoch 7: val_accuracy did not improve from 0.77311
1340/1340 [==============================] - 72s 54ms/step - loss: 0.4492 - accuracy
```

```
Epoch 8/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4468 - accuracy: 0.79
Epoch 8: val_accuracy did not improve from 0.77311
1340/1340 [==============================] - 81s 60ms/step - loss: 0.4467 - accuracy
Epoch 9/20
1340/1340 [==============================] - ETA: 0s - loss: 0.4438 - accuracy: 0.79
Epoch 9: val_accuracy improved from 0.77311 to 0.77505, saving model to best_model3.
1340/1340 [==============================] - 73s 55ms/step - loss: 0.4438 - accuracy
Epoch 10/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4412 - accuracy: 0.79
Epoch 10: val_accuracy improved from 0.77505 to 0.77576, saving model to best_model3
1340/1340 [==============================] - 81s 60ms/step - loss: 0.4413 - accuracy
Epoch 11/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4410 - accuracy: 0.80
Epoch 11: val_accuracy improved from 0.77576 to 0.77614, saving model to best_model3
1340/1340 [==============================] - 81s 60ms/step - loss: 0.4411 - accuracy
Epoch 12/20
1340/1340 [==============================] - ETA: 0s - loss: 0.4387 - accuracy: 0.80
Epoch 12: val_accuracy did not improve from 0.77614
1340/1340 [==============================] - 73s 54ms/step - loss: 0.4387 - accuracy
Epoch 13/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4366 - accuracy: 0.80
Epoch 13: val_accuracy improved from 0.77614 to 0.77623, saving model to best_model3
1340/1340 [==============================] - 73s 54ms/step - loss: 0.4365 - accuracy
Epoch 14/20
1339/1340 [============================>.] - ETA: 0s - loss: 0.4339 - accuracy: 0.80
Epoch 14: val_accuracy did not improve from 0.77623
```

```
# plotting the results

acc = history.history.get('accuracy')
val_acc = history.history.get('val_accuracy')
loss = history.history.get('loss')
val_loss = history.history.get('val_loss')
print(acc)
print(loss)
print(val_loss)
epochs = range(1, 21)
print (epochs)

plt.plot(epochs, acc, 'b', label='Training Acc')
plt.plot(epochs, val_acc, 'g', label='Validation Acc')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label="Training Loss")
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

```
[0.6809002161026001, 0.7629663944244385, 0.7777052521705627, 0.7836287021636963, 0.7887:
[0.5875165462493896, 0.5004152655601501, 0.4791983664035797, 0.4678001403808594, 0.4587(
[0.509067714214325, 0.4957297444343567, 0.4849430322647095, 0.4846324622631073, 0.47815(
range(1, 21)
```





Model 4:

Varying the units in hidden layers of a triple layered LSTM

```
history_data = []

neurons = [5,15,25]
for i in neurons:
  model4 = Sequential()
  model4.add(layers.Embedding(max_words, 20,input_length=max_len))
  model4.add(layers.LSTM(i,dropout=0.5,return_sequences = True))
  model4.add(layers.LSTM(units = i,dropout=0.5, return_sequences = False))
  model4.add(layers.Dense(2,activation='softmax'))
  model4.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['accuracy'])
  #Implementing model checkpoins to save the best metric and do not lose it on training.
  #checkpoint1 = ModelCheckpoint("best_model4.hdf5", monitor='val_accuracy', verbose=1,save_b
```

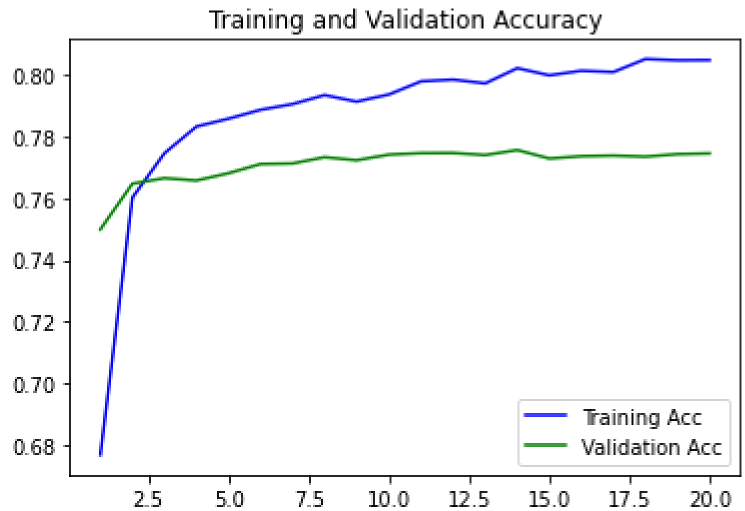
```
history = model4.fit(X_train, y_train, epochs=20,validation_data=(X_test, y_test))
history_data.append(history)
```

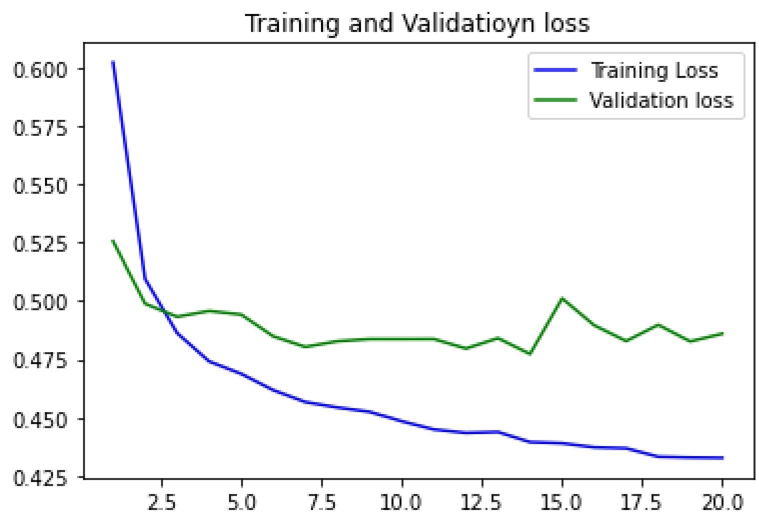
```
Epoch 1/20
1340/1340 [==============================] - 49s 35ms/step - loss: 0.6023 - accuracy
Epoch 2/20
1340/1340 [==============================] - 46s 35ms/step - loss: 0.5095 - accuracy
Epoch 3/20
1340/1340 [==============================] - 44s 32ms/step - loss: 0.4863 - accuracy
Epoch 4/20
1340/1340 [==============================] - 46s 35ms/step - loss: 0.4741 - accuracy
Epoch 5/20
1340/1340 [==============================] - 44s 33ms/step - loss: 0.4688 - accuracy
Epoch 6/20
1340/1340 [==============================] - 44s 33ms/step - loss: 0.4618 - accuracy
Epoch 7/20
1340/1340 [==============================] - 44s 33ms/step - loss: 0.4567 - accuracy
Epoch 8/20
1340/1340 [==============================] - 47s 35ms/step - loss: 0.4544 - accuracy
Epoch 9/20
1340/1340 [==============================] - 44s 33ms/step - loss: 0.4525 - accuracy
Epoch 10/20
1340/1340 [==============================] - 46s 35ms/step - loss: 0.4484 - accuracy
Epoch 11/20
1340/1340 [==============================] - 43s 32ms/step - loss: 0.4449 - accuracy
Epoch 12/20
1340/1340 [==============================] - 44s 33ms/step - loss: 0.4434 - accuracy
Epoch 13/20
1340/1340 [==============================] - 44s 32ms/step - loss: 0.4438 - accuracy
Epoch 14/20
1340/1340 [==============================] - 43s 32ms/step - loss: 0.4395 - accuracy
Epoch 15/20
1340/1340 [==============================] - 46s 35ms/step - loss: 0.4390 - accuracy
Epoch 16/20
1340/1340 [==============================] - 44s 33ms/step - loss: 0.4373 - accuracy
Epoch 17/20
1340/1340 [==============================] - 43s 32ms/step - loss: 0.4368 - accuracy
Epoch 18/20
1340/1340 [==============================] - 46s 35ms/step - loss: 0.4332 - accuracy
Epoch 19/20
1340/1340 [==============================] - 43s 32ms/step - loss: 0.4329 - accuracy
Epoch 20/20
1340/1340 [==============================] - 46s 35ms/step - loss: 0.4327 - accuracy
Epoch 1/20
1340/1340 [==============================] - 51s 37ms/step - loss: 0.5779 - accuracy
Epoch 2/20
1340/1340 [==============================] - 48s 36ms/step - loss: 0.4914 - accuracy
Epoch 3/20
1340/1340 [==============================] - 51s 38ms/step - loss: 0.4734 - accuracy
Epoch 4/20
1340/1340 [==============================] - 49s 36ms/step - loss: 0.4639 - accuracy
Epoch 5/20
1340/1340 [==============================] - 51s 38ms/step - loss: 0.4569 - accuracy
Epoch 6/20
1340/1340 [==============================] - 51s 38ms/step - loss: 0.4522 - accuracy
Epoch 7/20
```

```
1340/1340 [==============================] - 49s 37ms/step - loss: 0.4479 - accuracy
Epoch 8/20
1340/1340 [==============================] - 51s 38ms/step - loss: 0.4453 - accuracy
Epoch 9/20
```

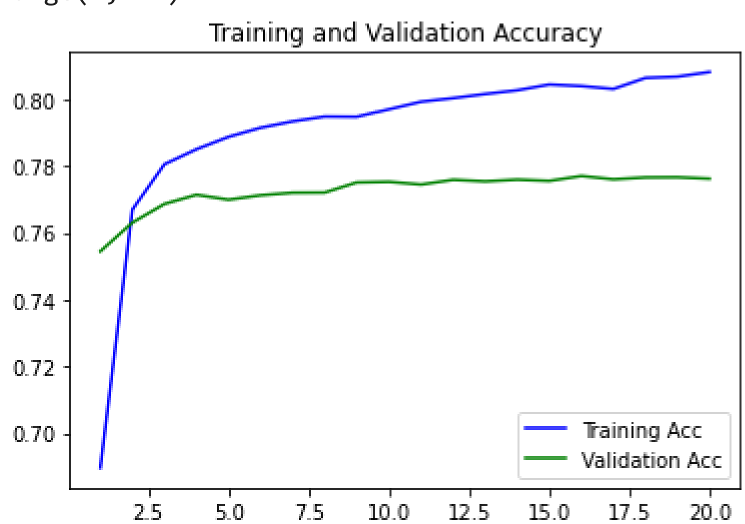
```
# plotting the results
for history in history_data:
  acc = history.history.get('accuracy')
  val_acc = history.history.get('val_accuracy')
  loss = history.history.get('loss')
  val_loss = history.history.get('val_loss')
  print(acc)
  print(loss)
  print(val_loss)
  epochs = range(1, 21)
  print (epochs)

  plt.plot(epochs, acc, 'b', label='Training Acc')
  plt.plot(epochs, val_acc, 'g', label='Validation Acc')
  plt.title('Training and Validation Accuracy')
  plt.legend()

  plt.figure()

  plt.plot(epochs, loss, 'b', label="Training Loss")
  plt.plot(epochs, val_loss, 'g', label='Validation loss')
  plt.title('Training and Validatioyn loss')
  plt.legend()
  plt.show()
```

```
[0.6766790747642517, 0.7601912021636963, 0.7746268510818481, 0.7833255529403687, 0.7
[0.6022629737854004, 0.5095149278640747, 0.48628726601600647, 0.47411832213401794, 0
[0.5255181193351746, 0.4989033639431, 0.4932955205440521, 0.4957827925682068, 0.4942
range(1, 21)
```






```
[0.6896222233772278, 0.7668610215187073, 0.780457079410553, 0.784958004951477, 0.788
[0.5778828859329224, 0.49142885208129883, 0.4733632206916809, 0.4638881981372833, 0.4
[0.5041282773017883, 0.4912038743495941, 0.48227062821388245, 0.48352929949760437, 0
range(1, 21)
```



Testing if double descent occurs by increasing in number of epochs

We have tested by training models with more complexity by increasing number of layers to try and see double descent now let us try out and see if we can observe a double descent with the increase in number of epochs. A single Layer LSTM (Testing with 200 epoches to try and observe if there is double descent when number of epoches are increased)

```
model1 = Sequential()
model1.add(layers.Embedding(max_words, 20,input_length=max_len))
model1.add(layers.LSTM(15,dropout=0.5))
model1.add(layers.Dense(2,activation='softmax'))

model1.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['accuracy'])
#Implementing model checkpoins to save the best metric and do not lose it on training.
checkpoint1 = ModelCheckpoint("best_model1.hdf5", monitor='val_accuracy', verbose=1,save_best
history = model1.fit(X_train, y_train, epochs=125,validation_data=(X_test, y_test),callbacks=
```

```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify
Epoch 1/125
1338/1340 [============================>.] - ETA: 0s - loss: 0.5626 - accuracy: 0.70
Epoch 1: val_accuracy improved from -inf to 0.76373, saving model to best_model1.hdf
1340/1340 [==============================] - 26s 18ms/step - loss: 0.5627 - accuracy
Epoch 2/125
1338/1340 [============================>.] - ETA: 0s - loss: 0.4831 - accuracy: 0.77
```

```
Epoch 2: val_accuracy improved from 0.76373 to 0.76738, saving model to best_model1.
1340/1340 [==============================] - 25s 19ms/step - loss: 0.4832 - accuracy
Epoch 3/125
1339/1340 [============================>.] - ETA: 0s - loss: 0.4663 - accuracy: 0.77
Epoch 3: val_accuracy improved from 0.76738 to 0.76875, saving model to best_model1.
1340/1340 [==============================] - 25s 18ms/step - loss: 0.4663 - accuracy
Epoch 4/125
1337/1340 [============================>.] - ETA: 0s - loss: 0.4575 - accuracy: 0.78
Epoch 4: val_accuracy improved from 0.76875 to 0.76984, saving model to best_model1.
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4575 - accuracy
Epoch 5/125
1337/1340 [============================>.] - ETA: 0s - loss: 0.4519 - accuracy: 0.78
Epoch 5: val_accuracy improved from 0.76984 to 0.77410, saving model to best_model1.
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4519 - accuracy
Epoch 6/125
1338/1340 [============================>.] - ETA: 0s - loss: 0.4476 - accuracy: 0.79
Epoch 6: val_accuracy did not improve from 0.77410
1340/1340 [==============================] - 25s 19ms/step - loss: 0.4474 - accuracy
Epoch 7/125
1339/1340 [============================>.] - ETA: 0s - loss: 0.4436 - accuracy: 0.79
Epoch 7: val_accuracy improved from 0.77410 to 0.77420, saving model to best_model1.
1340/1340 [==============================] - 25s 19ms/step - loss: 0.4436 - accuracy
Epoch 8/125
1340/1340 [==============================] - ETA: 0s - loss: 0.4398 - accuracy: 0.79
Epoch 8: val_accuracy did not improve from 0.77420
1340/1340 [==============================] - 25s 19ms/step - loss: 0.4398 - accuracy
Epoch 9/125
1337/1340 [============================>.] - ETA: 0s - loss: 0.4374 - accuracy: 0.79
Epoch 9: val_accuracy did not improve from 0.77420
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4375 - accuracy
Epoch 10/125
1337/1340 [============================>.] - ETA: 0s - loss: 0.4353 - accuracy: 0.79
Epoch 10: val_accuracy improved from 0.77420 to 0.77457, saving model to best_model1
1340/1340 [==============================] - 26s 20ms/step - loss: 0.4354 - accuracy
Epoch 11/125
1338/1340 [============================>.] - ETA: 0s - loss: 0.4325 - accuracy: 0.80
Epoch 11: val_accuracy did not improve from 0.77457
1340/1340 [==============================] - 26s 19ms/step - loss: 0.4325 - accuracy
Epoch 12/125
1338/1340 [============================>.] - ETA: 0s - loss: 0.4323 - accuracy: 0.79
Epoch 12: val_accuracy improved from 0.77457 to 0.77533, saving model to best_model1
1340/1340 [==============================] - 27s 20ms/step - loss: 0.4323 - accuracy
Epoch 13/125
1340/1340 [==============================] - ETA: 0s - loss: 0.4277 - accuracy: 0.80
Epoch 13: val_accuracy did not improve from 0.77533
1340/1340 [==============================] - 26s 20ms/step - loss: 0.4277 - accuracy
Epoch 14/125
1337/1340 [============================>.] - ETA: 0s - loss: 0.4262 - accuracy: 0.80
Epoch 14: val_accuracy did not improve from 0.77533
```

```
# plotting the results

acc = history.history.get('accuracy')
val_acc = history.history.get('val_accuracy')
```

```
loss = history.history.get('loss')
val_loss = history.history.get('val_loss')
print(acc)
print(loss)
print(val_loss)
epochs = range(1, 126)
print (epochs)

plt.plot(epochs, acc, 'b', label='Training Acc')
plt.plot(epochs, val_acc, 'g', label='Validation Acc')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label="Training Loss")
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

```
[0.7060167789459229, 0.7710821032524109, 0.7794309854507446, 0.7861940264701843, 0.7897
[0.5626755356788635, 0.483186811208725, 0.46633559465408325, 0.4575105905532837, 0.4519
[0.4986889362335205, 0.48743948340415955, 0.48237720131874084, 0.47692030668258667, 0.47
```

Testing if double descent occurs by increasing the amount of data we use for training

Model 7: We are going to use 4 times the data we used previously with the same Model 1 architecture(Single layer LSTM) to try and see if we can observe the double descent phenomenon.

```
ds_train, ds_train_info = tfds.load('sentiment140', split='train[:16%]', shuffle_files=True ,
# Using it as a numpy array
train_text, train_polarity = tfds.as_numpy(tfds.load('sentiment140', split='train[:16%]', shu
# Applying function to numpy
train_text = clean_v_func(train_text)
train_text = convert_v_func(train_text)
Y_Tr = train_polarity
```

```
max_words = 5000
#max_len = 200
max_len=max([len(row.split()) for row in train_text])

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_text)
sequences = tokenizer.texts_to_sequences(train_text)
tweets = pad_sequences(sequences, maxlen=max_len)
labels = train_polarity

y = []
for i in range(len(labels)):
    if labels[i] == 0:
        y.append(0)
    if labels[i] == 4:
        y.append(1)

y = np.array(y)
labels = tf.keras.utils.to_categorical(y, dtype="int32")
del y
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tweets,labels, test_size=0.33, random_sta
print ((X_train.shape),(X_test.shape),(y_train.shape),(y_test.shape))
model1 = Sequential()
model1.add(layers.Embedding(max_words, 20,input_length=max_len))
model1.add(layers.LSTM(15,dropout=0.5))
model1.add(layers.Dense(2,activation='softmax'))

model1.compile(optimizer='rmsprop',loss='binary_crossentropy', metrics=['accuracy'])
#Implementing model checkpoins to save the best metric and do not lose it on training.
checkpoint1 = ModelCheckpoint("best_model1.hdf5", monitor='val_accuracy', verbose=1,save_best
```

```
checkpoint1 = ModelCheckpoint('best_model1.hdf5', monitor='val_accuracy', verbose=1,save_best
history = model1.fit(X_train, y_train, epochs=20,validation_data=(X_test, y_test),callbacks=[
```

```
(171520, 569) (84480, 569) (171520, 2) (84480, 2)
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify
Epoch 1/20
5359/5360 [============================>.] - ETA: 0s - loss: 0.5057 - accuracy: 0.75
Epoch 1: val_accuracy improved from -inf to 0.77880, saving model to best_model1.hdf
5360/5360 [==============================] - 165s 30ms/step - loss: 0.5057 - accuracy
Epoch 2/20
5358/5360 [============================>.] - ETA: 0s - loss: 0.4648 - accuracy: 0.78
Epoch 2: val_accuracy improved from 0.77880 to 0.78593, saving model to best_model1.
5360/5360 [==============================] - 165s 31ms/step - loss: 0.4648 - accuracy
Epoch 3/20
5359/5360 [============================>.] - ETA: 0s - loss: 0.4574 - accuracy: 0.78
Epoch 3: val_accuracy improved from 0.78593 to 0.78823, saving model to best_model1.
5360/5360 [==============================] - 168s 31ms/step - loss: 0.4574 - accuracy
Epoch 4/20
5358/5360 [============================>.] - ETA: 0s - loss: 0.4523 - accuracy: 0.78
Epoch 4: val_accuracy improved from 0.78823 to 0.79074, saving model to best_model1.
5360/5360 [==============================] - 154s 29ms/step - loss: 0.4523 - accuracy
Epoch 5/20
5360/5360 [==============================] - ETA: 0s - loss: 0.4492 - accuracy: 0.79
Epoch 5: val_accuracy improved from 0.79074 to 0.79214, saving model to best_model1.
5360/5360 [==============================] - 168s 31ms/step - loss: 0.4492 - accuracy
Epoch 6/20
5358/5360 [============================>.] - ETA: 0s - loss: 0.4463 - accuracy: 0.79
Epoch 6: val_accuracy did not improve from 0.79214
5360/5360 [==============================] - 170s 32ms/step - loss: 0.4464 - accuracy
Epoch 7/20
5358/5360 [============================>.] - ETA: 0s - loss: 0.4448 - accuracy: 0.79
Epoch 7: val_accuracy improved from 0.79214 to 0.79334, saving model to best_model1.
5360/5360 [==============================] - 155s 29ms/step - loss: 0.4448 - accuracy
Epoch 8/20
5360/5360 [==============================] - ETA: 0s - loss: 0.4432 - accuracy: 0.79
Epoch 8: val_accuracy did not improve from 0.79334
5360/5360 [==============================] - 169s 31ms/step - loss: 0.4432 - accuracy
Epoch 9/20
5358/5360 [============================>.] - ETA: 0s - loss: 0.4419 - accuracy: 0.79
Epoch 9: val_accuracy improved from 0.79334 to 0.79510, saving model to best_model1.
5360/5360 [==============================] - 170s 32ms/step - loss: 0.4419 - accuracy
Epoch 10/20
5360/5360 [==============================] - ETA: 0s - loss: 0.4414 - accuracy: 0.79
Epoch 10: val_accuracy improved from 0.79510 to 0.79554, saving model to best_model1
5360/5360 [==============================] - 171s 32ms/step - loss: 0.4414 - accuracy
Epoch 11/20
5359/5360 [============================>.] - ETA: 0s - loss: 0.4398 - accuracy: 0.79
Epoch 11: val_accuracy improved from 0.79554 to 0.79566, saving model to best_model1
5360/5360 [==============================] - 169s 31ms/step - loss: 0.4398 - accuracy
Epoch 12/20
5360/5360 [==============================] - ETA: 0s - loss: 0.4399 - accuracy: 0.79
Epoch 12: val_accuracy did not improve from 0.79566
5360/5360 [==============================] - 156s 29ms/step - loss: 0.4399 - accuracy
Epoch 13/20
5360/5360 [==============================] - ETA: 0s - loss: 0.4386 - accuracy: 0.79
```

```
Epoch 13: val_accuracy did not improve from 0.79566
5360/5360 [==============================] - 170s 32ms/step - loss: 0.4386 - accurac
Epoch 14/20
```

```
# plotting the results

acc = history.history.get('accuracy')
val_acc = history.history.get('val_accuracy')
loss = history.history.get('loss')
val_loss = history.history.get('val_loss')
print(acc)
print(loss)
print(val_loss)
epochs = range(1, 21)
print (epochs)

plt.plot(epochs, acc, 'b', label='Training Acc')
plt.plot(epochs, val_acc, 'g', label='Validation Acc')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label="Training Loss")
plt.plot(epochs, val_loss, 'g', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```