

## MICP\_3

**1. In the code provided there are three mistakes which stop the code from running successfully; find those mistakes and explain why they need to be corrected to be able to get the code run.**

Errors that I have faced are

1. Tensorflow version error
2. Input\_dimension in our case it is 2000
3. Neuron in the last layer is 3 because we have 3 for prediction (pos, neg, unsp)
4. activation function I have used softmax as it is giving better accuracy

As far I have learned softmax gives good accuracy then other activation functions

```
[73] model = Sequential()
model.add(layers.Dense(512, activation='relu', input_dim=input_dim))
model.add(layers.Dense(3, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['acc'])

[77] hist=model.fit(X_train,y_train, epochs=5, verbose=1, batch_size=256, validation_data=(X_test,y_test))
```

Train on 75000 samples, validate on 25000 samples

Epoch	1/5	2/5	3/5	4/5	5/5
75000/75000	[.....] - 5s 70us/sample - loss: 0.0944 - acc: 0.9899 - val_loss: 1.2552 - val_acc: 0.5822	75000/75000 [.....] - 5s 69us/sample - loss: 0.0714 - acc: 0.9921 - val_loss: 1.3020 - val_acc: 0.5090	75000/75000 [.....] - 5s 67us/sample - loss: 0.0643 - acc: 0.9923 - val_loss: 1.3504 - val_acc: 0.5126	75000/75000 [.....] - 5s 67us/sample - loss: 0.0598 - acc: 0.9925 - val_loss: 1.3936 - val_acc: 0.5108	75000/75000 [.....] - 5s 67us/sample - loss: 0.0566 - acc: 0.9925 - val_loss: 1.4170 - val_acc: 0.5038

**2. Add embedding layer to the model, did you experience any improvement?**

**ANS:**

The accuracy got decreased but in the previous model it got overfit but when I have added embedded layer the accuracy and validation accuracy are at same point which means it got underfit.

colab.research.google.com/drive/1ULVgNyg66ehpN-nWEANSMT8oOn3Tje7authuser=1fscrollTo=elq4JfGgyv6

MICP3.ipynb

2nd Question adding embedded layer

```
[78] from tensorflow.keras.preprocessing.sequence import pad_sequences
max_review_len = max([len(s.split()) for s in sentences])
vocab_size = len(tokenizer.word_index)+1
sentences_pre = tokenizer.texts_to_sequences(sentences)
padded_docs = pad_sequences(sentences_pre,maxlen=max_review_len)
```

```
[79] #from keras.layers import Embedding
model.add(Embedding(vocab_size, 50, input_length=max_review_len))
X_train, X_test, y_train, y_test = train_test_split(padded_docs, y, test_size=0.25, random_state=1000)
temp_X=X_test
temp_y=y_test
```

```
[17] len(padded_docs)
#max_review_len
#padded_docs.shape
#X_train.shape
#vocab_size
```

```
100000
```

```
[84] #from tensorflow.keras.layers import Embedding
m = Sequential()
m.add(Embedding(vocab_size, 50, input_length=max_review_len))
m.add(Flatten())
m.add(layers.Dense(300,input_dim=max_review_len, activation='relu'))
m.add(layers.Dense(1, activation='softmax'))
#m = Sequential()
#m.add(layers.Dense(512, activation='relu',input_shape=(max_review_len,)))
#m.add(layers.Dense(512, activation='relu'))
#m.add(layers.Dense(3, activation='softmax'))
```

```
[17] #max_review_len
#padded_docs.shape
#X_train.shape
#vocab_size
```

```
100000
```

```
[84] #from tensorflow.keras.layers import Embedding
m = Sequential()
m.add(Embedding(vocab_size, 50, input_length=max_review_len))
m.add(Flatten())
m.add(layers.Dense(300,input_dim=max_review_len, activation='relu'))
m.add(layers.Dense(1, activation='softmax'))
#m = Sequential()
#m.add(layers.Dense(512, activation='relu',input_shape=(max_review_len,)))
#m.add(layers.Dense(512, activation='relu'))
#m.add(layers.Dense(3, activation='softmax'))
m.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['acc'])
#m.fit(X_train,y_train, epochs=5, verbose=1, batch_size=256, validation_data=(X_test,y_test))
```

```
[108] m.fit(X_train,y_train,batch_size=256,epochs=5,verbose=1,validation_data=(X_test,y_test))
```

```
Train on 75000 samples, validate on 25000 samples
Epoch 1/5
75000/75000 [=====] - 233s 3ms/sample - loss: 0.8886 - acc: 0.5116 - val_loss: 0.6730 - val_acc: 0.6254
Epoch 2/5
75000/75000 [=====] - 228s 3ms/sample - loss: 0.7859 - acc: 0.5497 - val_loss: 0.5682 - val_acc: 0.7039
Epoch 3/5
75000/75000 [=====] - 228s 3ms/sample - loss: 0.7294 - acc: 0.5977 - val_loss: 0.5154 - val_acc: 0.7777
Epoch 4/5
75000/75000 [=====] - 229s 3ms/sample - loss: 0.6495 - acc: 0.6623 - val_loss: 0.4957 - val_acc: 0.7855
Epoch 5/5
75000/75000 [=====] - 221s 3ms/sample - loss: 0.5317 - acc: 0.7464 - val_loss: 0.4182 - val_acc: 0.8173
<tensorflow.python.keras.callbacks.History at 0x7f31dcf52988>
```

### 3. Apply the code on 20\_newsgroup data set we worked in the previous classes

From sklearn. datasets import fetch\_20newsgroups

newsgroups\_train = fetch\_20newsgroups (subset='train', shuffle=True, categories = categories,)

The screenshot shows a Google Colab notebook titled "MICP3.ipynb". The left sidebar displays a file explorer with various files and folders. The main code editor contains the following Python code:

```
3rd Question training model on 20newsgroups dataset

[38] from sklearn.datasets import fetch_20newsgroups
newsgroups_train = fetch_20newsgroups(subset='train', shuffle=True)

[44] sentences=newsgroups_train.data
y=newsgroups_train.target

[53] max_review_len= max([len(s.split()) for s in sentences])
vocab_size= len(tokenizer.word_index)+1
sentences_pre = tokenizer.texts_to_sequences(sentences)
padded_docs= pad_sequences(sentences_pre,maxlen=max_review_len)

[57] max_review_len
y
X_train, X_test, y_train, y_test = train_test_split(padded_docs, y, test_size=0.25, random_state=1000)
max_review_len

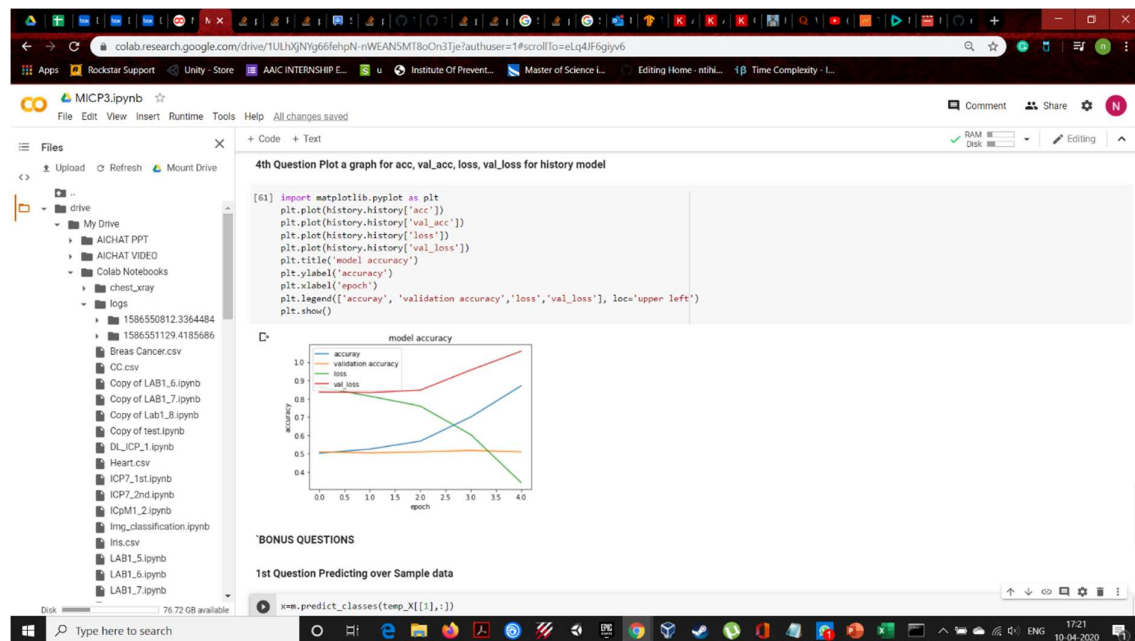
[58] p = Sequential()
p.add(Embedding(vocab_size, 50, input_length=max_review_len))
p.add(Flatten())
p.add(layers.Dense(300,input_dim=max_review_len, activation='relu'))
p.add(layers.Dense(20, activation='softmax'))
p.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['acc'])

[60] h1p.fit(X_train,y_train, epochs=5, verbose=1, batch_size=256, validation_data=(X_test,y_test))

Train on 8485 samples, validate on 2829 samples
Epoch 1/5
8485/8485 [=====] - 132s 16ms/sample - loss: 12.3796 - acc: 0.0529 - val_loss: 3.0152 - val_acc: 0.0375
Epoch 2/5
8485/8485 [=====] - 127s 15ms/sample - loss: 3.0063 - acc: 0.0500 - val_loss: 2.9930 - val_acc: 0.0559
Epoch 3/5
8485/8485 [=====] - 127s 15ms/sample - loss: 2.9784 - acc: 0.0574 - val_loss: 3.0011 - val_acc: 0.0604
Epoch 4/5
8485/8485 [=====] - 127s 15ms/sample - loss: 2.9676 - acc: 0.0673 - val_loss: 2.9778 - val_acc: 0.0633
Epoch 5/5
8485/8485 [=====] - 132s 16ms/sample - loss: 2.9420 - acc: 0.0704 - val_loss: 2.9549 - val_acc: 0.0675
```

The output console shows the training progress over 5 epochs, including loss and accuracy metrics for both training and validation sets.

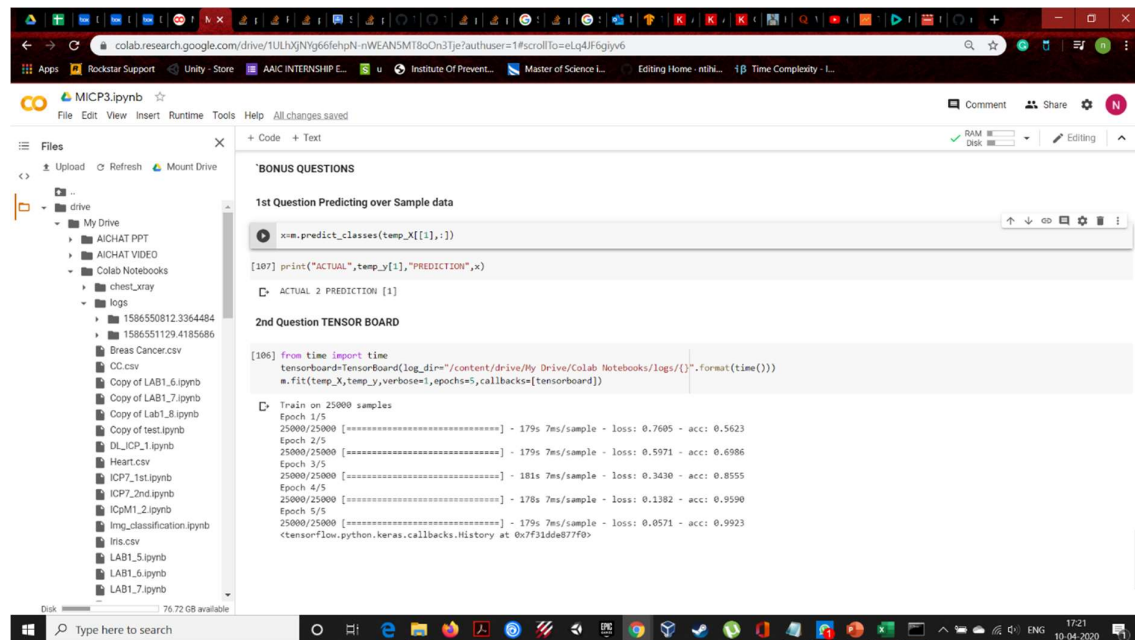
## 4. Plot the loss and accuracy using history object



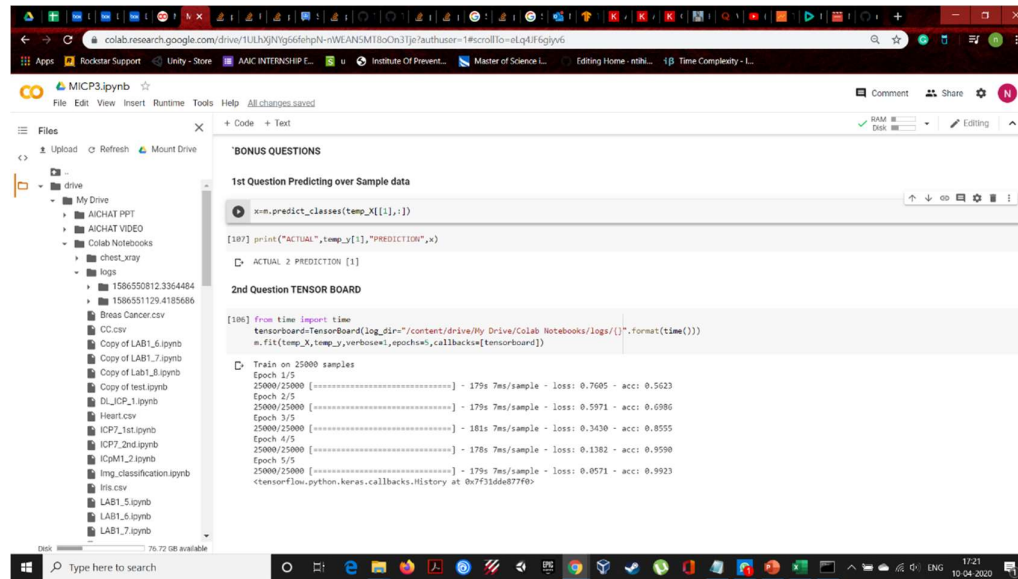
## \*Bonus

### Question

## 1. Predict over one sample of data and check what will be the prediction for that.



## 2. Plot loss and accuracy in Tensorboard.



The screenshot shows a Jupyter Notebook environment. The left sidebar displays a file explorer with a directory structure including 'drive' and 'logs'. The main code area contains the following Python code:

```
1 x = m.predict_classes(temp_X[1:],1)

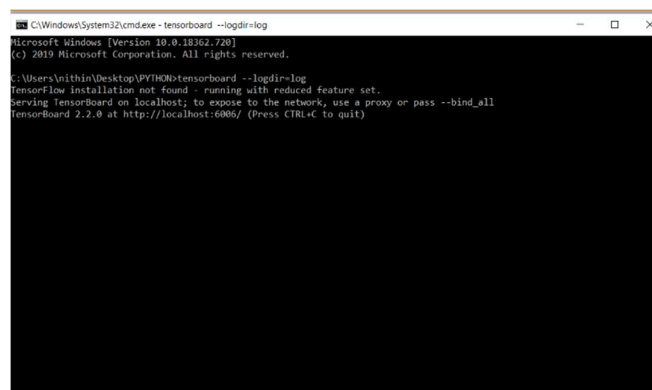
1897 print("ACTUAL",temp_y[1],"PREDICTION",x)

ACTUAL 2 PREDICTION [1]

2nd Question TENSOR BOARD

186 from time import time
   tensorboard=TensorBoard(log_dir="content/drive/My Drive/Colab Notebooks/logs/"),format(time()))
   n.fit(temp_X,temp_y,verbose=1,epochs=5,callbacks=[tensorboard])

Train on 25000 samples
Epoch 1/5
25000/25000 [=====] - 179s 7ms/sample - loss: 0.7605 - acc: 0.5623
Epoch 2/5
25000/25000 [=====] - 179s 7ms/sample - loss: 0.5971 - acc: 0.6986
Epoch 3/5
25000/25000 [=====] - 181s 7ms/sample - loss: 0.3430 - acc: 0.8555
Epoch 4/5
25000/25000 [=====] - 178s 7ms/sample - loss: 0.1382 - acc: 0.9598
Epoch 5/5
25000/25000 [=====] - 179s 7ms/sample - loss: 0.0571 - acc: 0.9923
<tensorflow.python.keras.callbacks.History at 0x7f31d6e877f0>
```

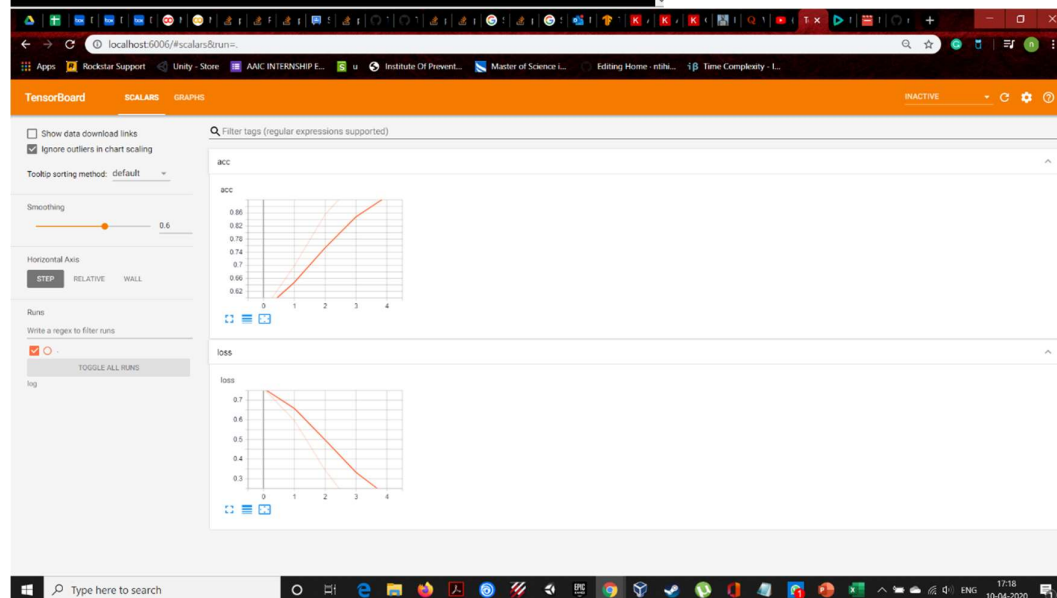


The screenshot shows a Windows command prompt window with the following text:

```
C:\Windows\System32\cmd.exe - tensorboard --logdir=log

Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\lithin\Desktop\PYTHON>tensorboard --logdir=log
TensorFlow installation not found - running with reduced feature set.
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.2.0 at http://localhost:6006/ (Press CTRL+C to quit)
```



The screenshot shows the TensorBoard web interface. The left sidebar contains controls for data download links, chart scaling, and smoothing. The main area displays two plots: 'acc' (accuracy) and 'loss' (loss). The 'acc' plot shows a red line representing accuracy over 4 epochs, starting at approximately 0.56 and rising to nearly 1.0. The 'loss' plot shows a red line representing loss over 4 epochs, starting at approximately 0.76 and falling to near 0.0. The interface also includes a search bar for filter tags and a 'Runs' section for managing data.