NLP Assignment_03

>Basha 2211ai03

>Ashok 2211ai22

>Sanatan 2211ai24

> Importing Libraries

In [1]:
```
import spacy
import re
import sys
import os
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from keras.layers import LSTM,Dense,Flatten
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow import keras
from progressbar import progressbar
import pandas as pd
import numpy as np
pb = progressbar(2)
```

> Extracting Zip files to Locations
>
> postive data to POS/pos Folder
>
> Negetive Data to NEG/neg Folder

In [2]:
```
import zipfile

pos_zip="/content/pos.zip"
```

```
neg_zip="/content/neg.zip"


with zipfile.ZipFile(pos_zip, 'r') as zip_ref:

    zip_ref.extractall("/content/POS")

with zipfile.ZipFile(neg_zip, 'r') as zip_ref:

    zip_ref.extractall("/content/NEG")
```

read_text function takes input folder locations of data

```
def read_text(folder_locations,max_files_toread):
    returns list[list[str],list[int]
```

In [3]:

```
def read_text(folder_locations,max_files_toread):
    text_data=[]

    if type(folder_locations)!=str:

        text_file_Locations=[]

        for index,location in enumerate(folder_locations):

            text_file_Locations.append( [ os.path.join(location,file_name)\
                                        for file_name in
os.listdir(location)][:max_files_toread]  )

    else:

        print(""" Folder locations in list or tuple format
        \n\t ex:[Folder loc1,Folder loc2,Folder loc3]""")

        return

    class_names = [0 for _ in folder_locations]
```

```
    present_length = 0
    for location_ind,text_file_Location in enumerate(text_file_Locations):
        for index,text_file in enumerate(text_file_Location):
            pb.print(index,len(text_file_Location))
            with open(text_file,'r',encoding='utf-8') as f:
                try:
                    text_data.append(f.readlines())
                except Exception as e:
                    print(e,text_file)
        class_names[location_ind]=len(text_data)-present_length
        present_length = len(text_data)
    class_labels=[]
    for index,no_of_files_inclass in enumerate(class_names):
        class_labels+=[index for number in range(no_of_files_inclass)]
    return text_data,class_labels
text_data,class_labels = read_text(["/content/NEG/neg",
                                    "/content/POS/pos"],5000)
```

```
[>.....................................................................................
```

In [4]:

```
text_data[1],class_labels[1]
```

Out[4]:
```
(['OK, here it is: "Nazi mountaineer befriends the Dalai Lama." What we do is, first we get a major star with no idea whatsoever how to do a Germanic accent, and we let him flounder around between French, German, American, and British for over 2 hours. Then we concoct a series of wildly improbable events and space them apart very widely, so that the plot inches along almost imperceptibly. But just to make sure the viewer doesn´t fall asleep, we throw in details which are shockingly absurd, such as our hero smoking a cigarette at an altitude of 22,000 feet. Naturally, we must also remember that our target audience does not want to read too many subtitles, so we have every character, even the lowliest peasant in the forbidden closed-off city of Lhasa in 1943, speak perfect English, also with dubious accents. Of course, the trickiest part is how to handle the spiritual and political aspects of the story, so what we do is this: we have the Dalai Lama befriend the now-reformed Nazi because the latter is so good at fiddling with film projectors, radios, antique cars, and any other devices with represent the freedom of the capitalist west. In return, our hero learns from his young protegé a kind of vague, undefined Buddhism which is never really brought out or treated in a serious fashion. We also have lots of scenes with the hero flaunting all the marks of respects and protocol which the rest of the Tibetan society accords the Dalai Lama, even as we pretend that the hero has deep and profound reverence for these people and their spiritual leader. In other words, we just expect the audience to believe that this guy is now a Buddhist, sort of, in his own way, even though we ourselves don´t seem to know what his transformation entails or how far we want it to go. And last but not least, we hang a statistic onto the end of the film about how appallingly the Chinese have treated the Tibetans (which is certainly true), thus opening ourselves up to charges that we have made a "political" movie, even though it is nothing of the sort. So, zat ist my idea. Vat do you zink? Can ve make zis movie?'],
 0)
```

> Removing HTML tags from text data

In [5]:
```
regex = re.compile(r'<[^>]+>')

def remove_html(string):

    return regex.sub(' ', string)

text_data=[remove_html(text[0]) for text in text_data]
```

In [6]:
```
text_data[0]
```

Out[6]:
```
"The direction struck me as poor man's Ingemar Bergman. The inaudible dialogue was annoying. The somber stoicism that all characters except Banderas' showed made me think they were drugged. I think the director ruined it for me."
```

Data tokenizer

In [7]:
```
def tokenize(texts):

    for text_ind,text in enumerate(texts):

        texts[text_ind]=text.lower()

    nlp = spacy.load('en_core_web_sm')

    tokenized_texts = []

    for ind,text in enumerate(texts):

        pb.print(ind,len(texts))

        doc = nlp(text)

        tokens = [token.text for token in doc]

        tokenized_texts.append(tokens)

    return tokenized_texts
```

In [8]:
```
tokenized_texts = tokenize( text_data )
```

```
[------------------------------------------------------------------------------
-------------->]
```

Calculating frequent_tokens

In [9]:
```
def freqent_tokens( text_data, frequency=5 ):

    counter={}

    for text in text_data:

        for token in text:

            if token in counter:
```

```
                counter[token]+=1

            else:

                counter[token]=1

    freq_tokens=[]

    for i in counter.items():

        if i[1]>frequency:

            freq_tokens.append(i[0])


    return freq_tokens

freq_tokens = freqent_tokens(tokenized_texts,frequency=300)

f" Most frequent Tokens length  {len(freq_tokens)}"
```

Out[9]:
```
' Most frequent Tokens length  830'
```

In [10]:
```
tokenized_texts[1][:10]
```

Out[10]:
```
['ok', ',', 'here', 'it', 'is', ':', '"', 'nazi', 'mountaineer', 'befriends']
```

Padding sequences

In [11]:
```
def Pad_sequences(sequences):
    avg_len=0

    for text in sequences:

        avg_len+=len(text)

    avg_len=int(avg_len/len(tokenized_texts))
```

```
    for text_ind,text in enumerate(sequences):

        if len(text)>=avg_len:

            sequences[text_ind]=text[:avg_len]

        else:

            for i in range(avg_len-len(text)):
                sequences[text_ind].append('PAD')

Pad_sequences(tokenized_texts)
```

converting words to one hot coding vectors

In [12]:

```
def one_hot_coding(tokenized_texts):

    token_to_number={x:to_categorical(ind,
            num_classes=len(freq_tokens)+1,dtype='uint8') for ind,
                    x in enumerate(freq_tokens)}

    token_to_number['unk']=np.array([0 for x in range(len(freq_tokens))]+
[1])

    for text in tokenized_texts:

        for token_ind,token in enumerate(text):

            if token in token_to_number:

                text[token_ind]=token_to_number[token]

        else:

                text[token_ind]=token_to_number['unk']

one_hot_coding(tokenized_texts)
```

In [13]:
```
X = np.array(tokenized_texts,dtype='uint8')
```

In [14]:
```
Data_set_1=X
```

Reading tweets Dataset

In [15]:
```
data2=pd.read_csv("/content/2013semeval_train.csv")
```

In [16]:
```
data2.tweet
```

Out[16]:
```
0       Gas by my house hit $3.39!!!! I\u2019m going t...
1       Theo Walcott is still shit\u002c watch Rafa an...
2       its not that I\u2019m a GSP fan\u002c i just h...
3       Iranian general says Israel\u2019s Iron Dome c...
4       Tehran\u002c Mon Amour: Obama Tried to Establi...
                              ...
9679    RT @MNFootNg It's monday and Monday Night Foot...
9680    All I know is the road for that Lomardi start ...
9681    "All Blue and White fam, we r meeting at Golde...
9682    @DariusButler28  Have a great game agaist Tam...
9683    "I'm pisseeedddd that I missed Kid Cudi's show...
Name: tweet, Length: 9684, dtype: object
```

Cleaning unicodes out of text

In [17]:
```
unicode_escape_regex = re.compile(r'\\u([0-9a-fA-F]{4})')


def convert_escape_sequence(match):
    return chr(int(match.group(1), 16))



result = unicode_escape_regex.sub(convert_escape_sequence, data2.tweet[1])
def change_string(string):
    return unicode_escape_regex.sub(convert_escape_sequence,string)
print(result)
```

```
Theo Walcott is still shit, watch Rafa and Johnny deal with him on Saturday.
```

In [18]:
```
data2.tweet=data2.tweet.apply(change_string)
```

Tokenizing

In [19]:
```
tokenized_texts = tokenize( data2.tweet )
```

```
[----------------------------------------------------------------------------------
------------>.]
```

Calculating freq tokens

In [20]:
```
freq_tokens = freqent_tokens(tokenized_texts,frequency=200)


f" Most frequent Tokens length  {len(freq_tokens)}"
```

Out[20]:
```
' Most frequent Tokens length  138'
```

Padding and converting to one_hot_coding

In [21]:
```
Pad_sequences(tokenized_texts)
one_hot_coding(tokenized_texts)
```

In [22]:
```
X = np.array(tokenized_texts,dtype='uint8')
```

In [23]:
```
Data_set_2=X
```

In [24]:
```
Data_set_1.shape,Data_set_2.shape
```

Out[24]:
```
((10000, 273, 831), (9684, 23, 139))
```

Convertin dataset_1 labels to numpy array

In [25]:
```
Data_set_1_labels=np.array(class_labels)
```

Converting dataset_2 labels to numpy array

In [26]:
```
unique_labels={x:ind for ind,x in enumerate(data2.label.unique())}
```

In [27]:
```
data2.label = data2.label.apply(lambda x:unique_labels[x])
```

In [28]:
```
Data_set_2_labels = data2.label.values
```

Final Data

In [29]:
```
Data_set_1.shape,Data_set_1_labels.shape,Data_set_2.shape,Data_set_2_labels.
```

Out[29]:
```
((10000, 273, 831), (10000,), (9684, 23, 139), (9684,))
```

Recurrent Neural Network Models

In [30]:
```
model_1_RNN=keras.Sequential([
    LSTM(256,input_shape=Data_set_1.shape[1:]),
    Dense(2,activation='sigmoid')
])
model_1_RNN.compile(loss='sparse_categorical_crossentropy',optimizer='adam',
['accuracy'])

model_2_RNN=keras.Sequential([
    LSTM(256,input_shape=Data_set_2.shape[1:]),
    Dense(3,activation='sigmoid')
])
model_2_RNN.compile(loss='sparse_categorical_crossentropy',optimizer='adam',
['accuracy'])
```

In [31]:
```
from sklearn.model_selection import train_test_split

x_train_1, x_test_1, y_train_1, y_test_1 = \
    train_test_split(Data_set_1, Data_set_1_labels, test_size=0.25,
random_state=42)
x_train_2, x_test_2, y_train_2, y_test_2 = \
    train_test_split(Data_set_2, Data_set_2_labels, test_size=0.25,
random_state=42)
```

In [32]:
```
history_1 = model_1_RNN.fit(x_train_1,y_train_1,epochs=10)
```

```
Epoch 1/10
235/235 [==============================] - 10s 28ms/step - loss: 0.6947 - accuracy: 0.4
992
Epoch 2/10
235/235 [==============================] - 6s 26ms/step - loss: 0.6929 - accuracy: 0.54
67
Epoch 3/10
235/235 [==============================] - 6s 27ms/step - loss: 0.6721 - accuracy: 0.56
83
Epoch 4/10
235/235 [==============================] - 6s 26ms/step - loss: 0.6744 - accuracy: 0.56
56
Epoch 5/10
235/235 [==============================] - 6s 28ms/step - loss: 0.6343 - accuracy: 0.58
41
Epoch 6/10
235/235 [==============================] - 6s 27ms/step - loss: 0.6043 - accuracy: 0.61
25
Epoch 7/10
235/235 [==============================] - 7s 29ms/step - loss: 0.5777 - accuracy: 0.62
63
Epoch 8/10
235/235 [==============================] - 6s 27ms/step - loss: 0.5462 - accuracy: 0.64
73
Epoch 9/10
235/235 [==============================] - 6s 27ms/step - loss: 0.5347 - accuracy: 0.65
27
Epoch 10/10
235/235 [==============================] - 7s 28ms/step - loss: 0.5060 - accuracy: 0.65
56
```

In [33]:

```
history_2 = model_2_RNN.fit(x_train_2,y_train_2,epochs=10)
```

```
Epoch 1/10
227/227 [==============================] - 3s 6ms/step - loss: 0.9625 - accuracy: 0.527
5
Epoch 2/10
227/227 [==============================] - 2s 7ms/step - loss: 0.8952 - accuracy: 0.588
2
Epoch 3/10
227/227 [==============================] - 1s 6ms/step - loss: 0.8716 - accuracy: 0.602
5
Epoch 4/10
227/227 [==============================] - 1s 5ms/step - loss: 0.8621 - accuracy: 0.608
2
Epoch 5/10
227/227 [==============================] - 1s 5ms/step - loss: 0.8493 - accuracy: 0.611
5
Epoch 6/10
227/227 [==============================] - 1s 5ms/step - loss: 0.8339 - accuracy: 0.621
1
Epoch 7/10
227/227 [==============================] - 1s 5ms/step - loss: 0.8242 - accuracy: 0.619
3
Epoch 8/10
227/227 [==============================] - 1s 5ms/step - loss: 0.8220 - accuracy: 0.626
3
Epoch 9/10
227/227 [==============================] - 1s 5ms/step - loss: 0.8047 - accuracy: 0.640
4
Epoch 10/10
227/227 [==============================] - 1s 5ms/step - loss: 0.7830 - accuracy: 0.647
5
```

In [34]:

```
from sklearn.metrics import precision_recall_fscore_support
y_pred_1=model_1_RNN.predict(x_test_1)
scores=precision_recall_fscore_support(np.argmax(y_pred_1,axis=1),y_test_1,a
print(f"""\n\nRNN Model_1 For DataSet1
    Precision = {scores[0]}
    Recall    = {scores[1]}
    f1_score  = {scores[2]}
""")
```

```
79/79 [==============================] - 1s 13ms/step


RNN Model_1 For DataSet1
    Precision = 0.513189365537869
    Recall    = 0.5278461473999495
    f1_score  = 0.4361392228507667
```

In [36]:
```
from sklearn.metrics import precision_recall_fscore_support
y_pred_2=model_2_RNN.predict(x_test_2)
scores=precision_recall_fscore_support(np.argmax(y_pred_2,axis=1),y_test_2,a
print(f"""\n\nRNN Model_2 For DataSet_2
    Precision = {scores[0]}
    Recall    = {scores[1]}
    f1_score  = {scores[2]}
""")
```

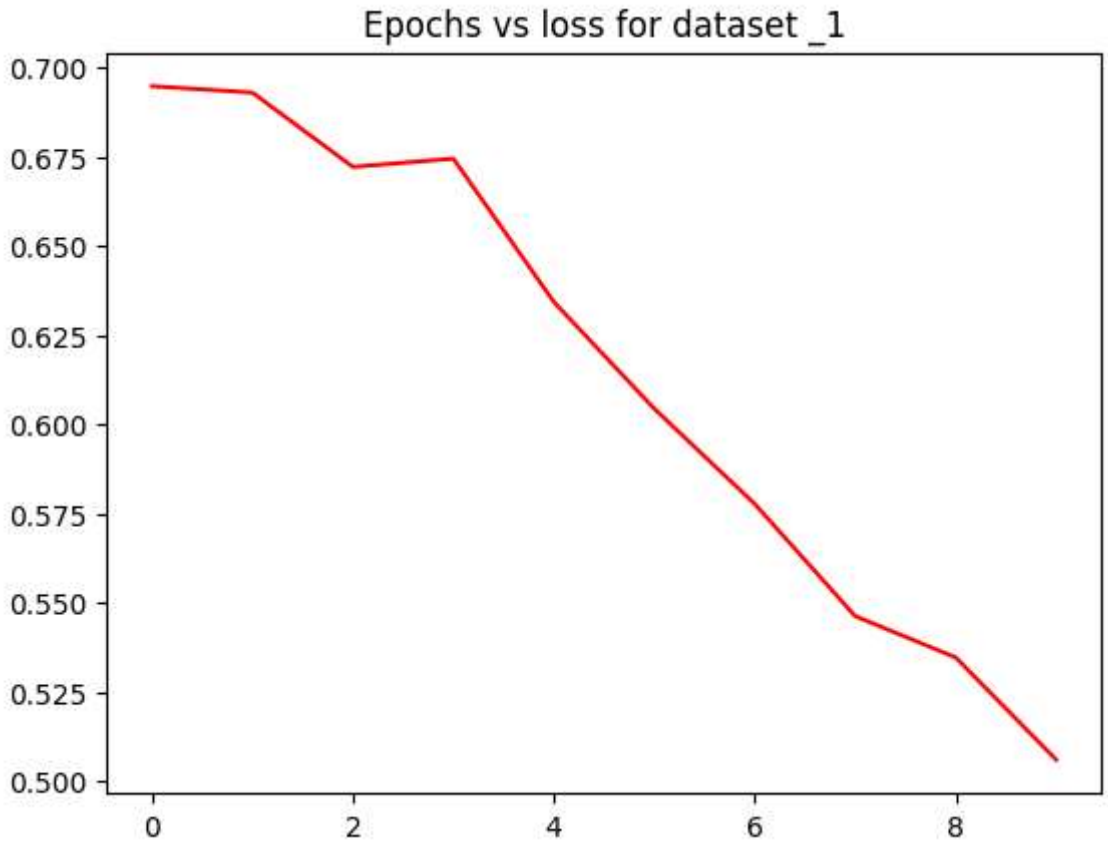
```
76/76 [==============================] - 2s 6ms/step


RNN Model_2 For DataSet_2
    Precision = 0.4880038046095922
    Recall    = 0.5205781582898968
    f1_score  = 0.49196638884233307
```

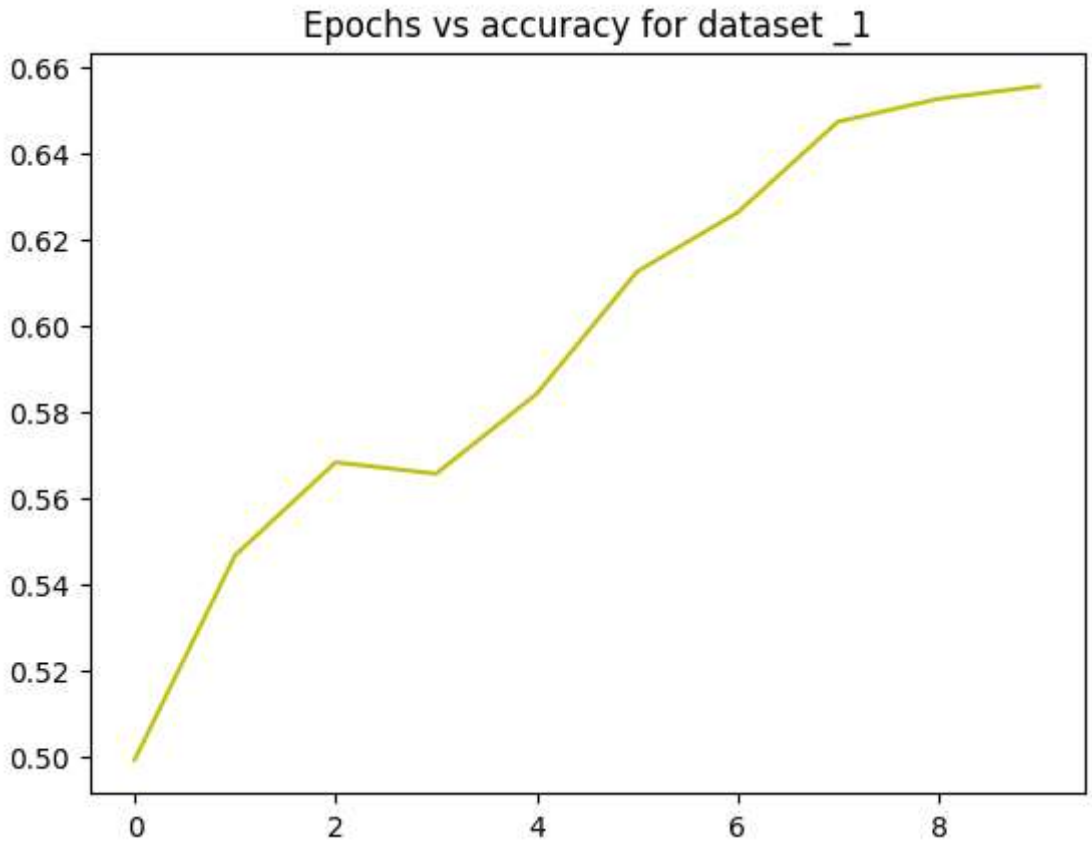
In [37]:
```
history_1.history
```

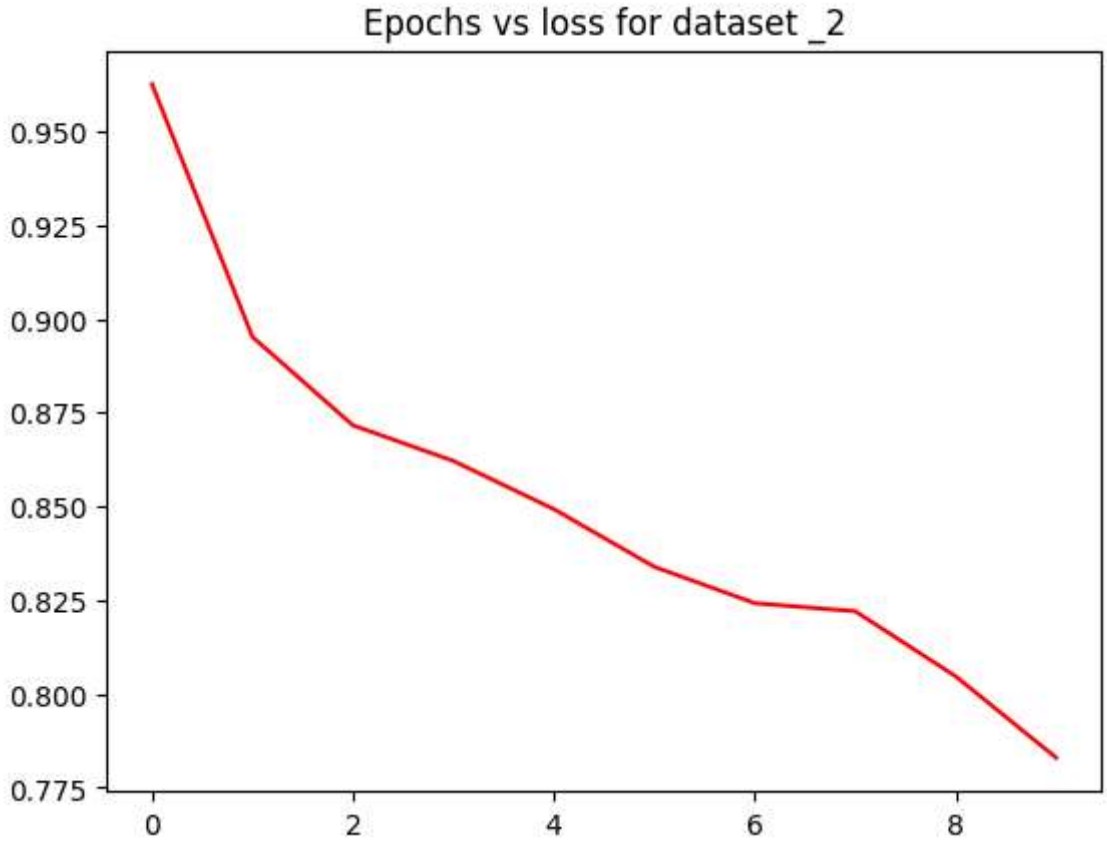
Out[37]:
```
{'loss': [0.6946829557418823,
  0.6928868889808655,
  0.6720706224441528,
  0.6743826270103455,
  0.6342559456825256,
  0.6043278574943542,
  0.5776535272598267,
  0.5462218523025513,
  0.5346569418907166,
  0.5060117840766907],
 'accuracy': [0.4991999864578247,
  0.54666668176651,
  0.5682666897773743,
  0.5655999779701233,
  0.5841333270072937,
  0.6125333309173584,
  0.6262666583061218,
  0.6473333239555359,
  0.6526666879653931,
  0.6556000113487244]}
```

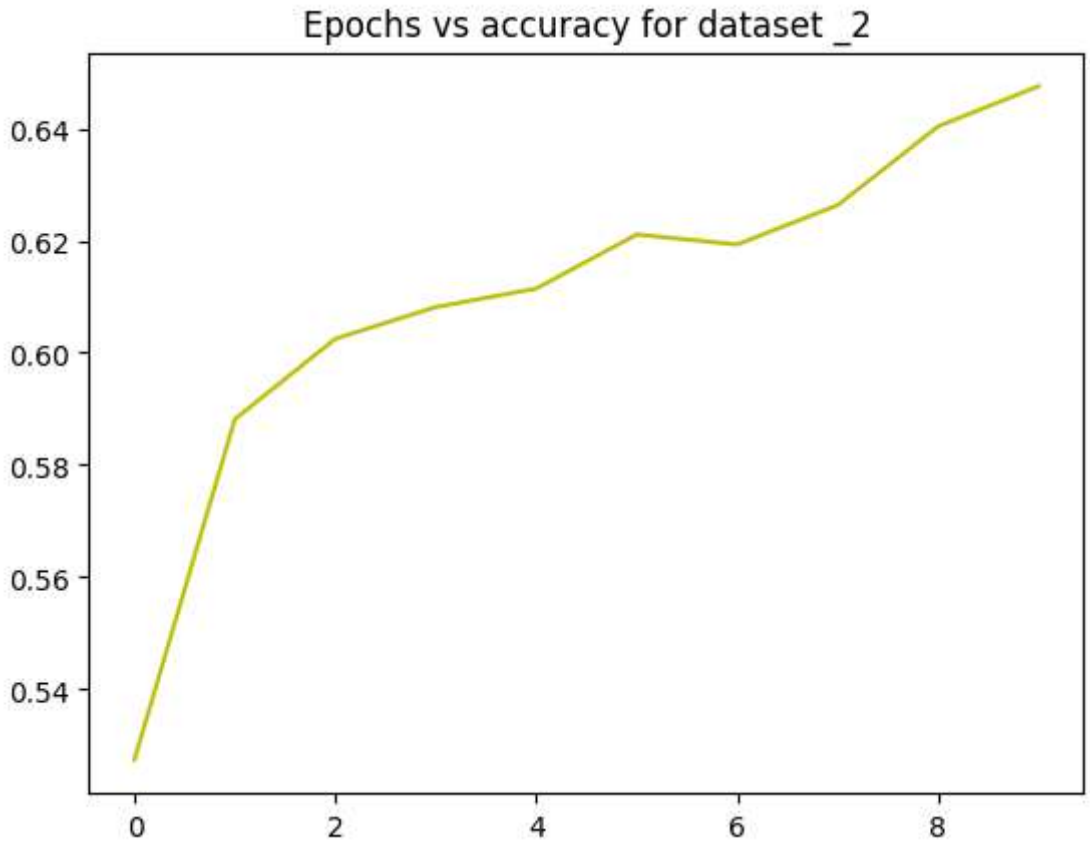
In [38]:
```
plt.plot([x for x in range(10)],history_1.history['loss'],c='r')
plt.title("Epochs vs loss for dataset _1")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
plt.plot([x for x in range(10)],history_1.history['accuracy'],c='y')
```

```
plt.title("Epochs vs accuracy for dataset _1")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
plt.plot([x for x in range(10)],history_2.history['loss'],c='r')
plt.title("Epochs vs loss for dataset _2")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
plt.plot([x for x in range(10)],history_2.history['accuracy'],c='y')
plt.title("Epochs vs accuracy for dataset _2")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
```




Epochs vs accuracy for dataset _1
0.66
0.64
0.62
0.60
0.58
0.56
0.54
0.52
0.50
0
2
4
6
8


Epochs vs loss for dataset _2
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0
2
4
6
8



Feed forward Neural Networks

The feed-forward neural network (FFNN) architecture I am proposing has two hidden layers with non-linear activation functions between them. The input layer is connected to the first hidden layer, which has 256 neurons. The first hidden layer is connected to the second hidden layer, which has 128 neurons. Finally, the second hidden layer is connected to the output layer, whose size depends on the number of classes in the problem.

Here is a visual representation of the architecture:

```
Input Layer (Size depends on input data)
            |
            v
Hidden Layer 1 (Size: 256) with Non-linearity
            |
            v
Hidden Layer 2 (Size: 128) with Non-linearity
            |
            v
Output Layer (Size depends on the number of classes)
```

The choice of non-linearity for this architecture is left open, with some popular choices being the Rectified Linear Unit (ReLU), hyperbolic tangent (tanh), or Gaussian Error Linear Unit (GELU).

For binary classification problems, the binary cross-entropy loss function is often used. For multi-class classification problems, the categorical cross-entropy loss function is commonly used.

Feed-Forward Neural Networks are useful for several reasons:

They are capable of modeling complex non-linear relationships between inputs and outputs.
They are relatively simple to understand and implement.
They can be trained efficiently on large datasets using techniques like stochastic gradient descent and backpropagation.
They can be applied to a wide range of problems, including classification, regression, and prediction.

However, FFNNs are limited in their ability to handle sequential or time-series data, as they do not take into account the temporal relationships between inputs. They are also prone to overfitting if the model is too complex or if the dataset is too small.

In [38]:
```
with open("logs.txt" , 'w') as f:
    for ind,history in enumerate([history_1,history_2]):
        for data in ['loss','accuracy']:
            for epoch,value in enumerate(history.history[data]):
                f.writelines(f"""For dataset_{ind+1}
                            \t\t\t For epoch_{epoch} \t{data}=
{round(value,2)}\n""")
```