Assignment_4

October 9, 2024

1 BUILD A NEURAL NETWORK-BASED CLASSIFIER FOR CUSTOMER CHURN.

1.1 Importing Libraries and Dataset

[30]:
```
import numpy as np
import pandas as pd
import os
```

[31]:
```
df = pd.read_csv('Churn_Modelling.csv')
```

[32]:
```
df.shape
```

[32]:
```
(10000, 14)
```

[33]:
```
df.head()
```

[33]:
```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

[6]:
```
df.duplicated().sum()
```

[6]:
```
0
```

[7]:
```
df['Exited'].value_counts()
```

[7]:
```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

[8]:
```
df['Gender'].value_counts()
```

[8]:
```
Gender
Male      5457
Female    4543
Name: count, dtype: int64
```

[9]:
```
df.drop(columns = ['RowNumber','CustomerId','Surname'],inplace = True)
```

[10]:
```
df.head()
```

[10]:
```
   CreditScore Geography  Gender  Age  Tenure    Balance  NumOfProducts  \
0          619    France  Female   42       2       0.00              1
```

```
1          608     Spain  Female   41        1   83807.86              1
2          502    France  Female   42        8  159660.80              3
3          699    France  Female   39        1       0.00              2
4          850     Spain  Female   43        2  125510.82              1

   HasCrCard  IsActiveMember  EstimatedSalary  Exited
0          1               1        101348.88       1
1          0               1        112542.58       0
2          1               0        113931.57       1
3          0               0         93826.63       0
4          1               1         79084.10       0
```

1.2 Converting categorical values to numerical values

[11]:
```python
df = pd.get_dummies(df, columns = ['Geography', 'Gender'], drop_first = True)
```

[12]:
```python
df
```

[12]:
```
      CreditScore  Age  Tenure    Balance  NumOfProducts  HasCrCard  \
0             619   42       2       0.00              1          1
1             608   41       1   83807.86              1          0
2             502   42       8  159660.80              3          1
3             699   39       1       0.00              2          0
4             850   43       2  125510.82              1          1
…             …    …     …        …            …        …
9995          771   39       5       0.00              2          1
9996          516   35      10   57369.61              1          1
9997          709   36       7       0.00              1          0
9998          772   42       3   75075.31              2          1
9999          792   28       4  130142.79              1          1

      IsActiveMember  EstimatedSalary  Exited  Geography_Germany  \
0                  1        101348.88       1              False
1                  1        112542.58       0              False
2                  0        113931.57       1              False
3                  0         93826.63       0              False
4                  1         79084.10       0              False
…                …              …     …                …
9995               0         96270.64       0              False
9996               1        101699.77       0              False
9997               1         42085.58       1              False
9998               0         92888.52       1               True
9999               0         38190.78       0              False

      Geography_Spain  Gender_Male
0               False        False
1                True        False
```

```
2               False        False
3               False        False
4                True        False
…               …          …
9995            False         True
9996            False         True
9997            False        False
9998            False         True
9999            False        False

[10000 rows x 12 columns]
```

[13]:
```
X = df.drop(columns = ['Exited'])
y = df['Exited']
```

[14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2,␣
↪random_state=1)
```

1.3 Scaling the values

[15]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

[16]:
```
X_train_scaled
```

[16]:
```
array([[-0.23082038, -0.94449979, -0.70174202, …,  1.71490137,
        -0.57273139,  0.91509065],
       [-0.25150912, -0.94449979, -0.35520275, …, -0.58312392,
        -0.57273139, -1.09278791],
       [-0.3963303 ,  0.77498705,  0.33787579, …,  1.71490137,
        -0.57273139, -1.09278791],
       …,
       [ 0.22433188,  0.58393295,  1.3774936 , …, -0.58312392,
        -0.57273139, -1.09278791],
       [ 0.13123255,  0.01077067,  1.03095433, …, -0.58312392,
        -0.57273139, -1.09278791],
       [ 1.1656695 ,  0.29735181,  0.33787579, …,  1.71490137,
        -0.57273139,  0.91509065]])
```

1.4 Building Neural Network

[17]:
```
import tensorflow
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
```

[18]:
```
model  = Sequential()

model.add(Dense(11,activation = 'relu', input_dim = 11))
model.add(Dense(11,activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))
```

```
D:\anaconda3\envs\dp_env\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[19]:
```
model.summary()
```

```
Model: "sequential"


 Layer (type)                          Output Shape                          ␣
↪Param #

 dense (Dense)                         (None, 11)                              ␣
↪132

 dense_1 (Dense)                       (None, 11)                              ␣
↪132

 dense_2 (Dense)                       (None, 1)                               ␣
↪ 12


 Total params: 276 (1.08 KB)

 Trainable params: 276 (1.08 KB)

 Non-trainable params: 0 (0.00 B)
```

1.5 Training Model

[20]:
```
model.compile(loss = 'binary_crossentropy', optimizer = 'Adam',␣
↪metrics=['accuracy'])
```

[23]:
```
history = model.fit(X_train_scaled, y_train, epochs = 100, validation_split=0.2)
```

```
Epoch 1/100
200/200              0s 1ms/step -
accuracy: 0.8589 - loss: 0.3405 - val_accuracy: 0.8506 - val_loss: 0.3501
Epoch 2/100
200/200              0s 963us/step -
accuracy: 0.8629 - loss: 0.3311 - val_accuracy: 0.8475 - val_loss: 0.3491
Epoch 3/100
200/200              0s 1ms/step -
accuracy: 0.8612 - loss: 0.3450 - val_accuracy: 0.8481 - val_loss: 0.3490
Epoch 4/100
200/200              0s 998us/step -
accuracy: 0.8655 - loss: 0.3288 - val_accuracy: 0.8487 - val_loss: 0.3481
Epoch 5/100
200/200              0s 1ms/step -
accuracy: 0.8632 - loss: 0.3336 - val_accuracy: 0.8550 - val_loss: 0.3489
Epoch 6/100
200/200              0s 931us/step -
accuracy: 0.8653 - loss: 0.3365 - val_accuracy: 0.8575 - val_loss: 0.3482
Epoch 7/100
200/200              0s 913us/step -
accuracy: 0.8671 - loss: 0.3314 - val_accuracy: 0.8506 - val_loss: 0.3479
Epoch 8/100
200/200              0s 945us/step -
accuracy: 0.8698 - loss: 0.3225 - val_accuracy: 0.8525 - val_loss: 0.3486
Epoch 9/100
200/200              0s 985us/step -
accuracy: 0.8637 - loss: 0.3315 - val_accuracy: 0.8544 - val_loss: 0.3488
Epoch 10/100
200/200              0s 864us/step -
accuracy: 0.8663 - loss: 0.3324 - val_accuracy: 0.8531 - val_loss: 0.3482
Epoch 11/100
200/200              0s 914us/step -
accuracy: 0.8668 - loss: 0.3286 - val_accuracy: 0.8494 - val_loss: 0.3485
Epoch 12/100
200/200              0s 891us/step -
accuracy: 0.8591 - loss: 0.3410 - val_accuracy: 0.8506 - val_loss: 0.3480
Epoch 13/100
200/200              0s 869us/step -
accuracy: 0.8646 - loss: 0.3341 - val_accuracy: 0.8506 - val_loss: 0.3480
Epoch 14/100
200/200              0s 890us/step -
```

```
accuracy: 0.8685 - loss: 0.3325 - val_accuracy: 0.8512 - val_loss: 0.3478
Epoch 15/100
200/200            0s 859us/step -
accuracy: 0.8582 - loss: 0.3431 - val_accuracy: 0.8525 - val_loss: 0.3479
Epoch 16/100
200/200            0s 899us/step -
accuracy: 0.8701 - loss: 0.3198 - val_accuracy: 0.8544 - val_loss: 0.3481
Epoch 17/100
200/200            0s 962us/step -
accuracy: 0.8635 - loss: 0.3355 - val_accuracy: 0.8506 - val_loss: 0.3486
Epoch 18/100
200/200            0s 927us/step -
accuracy: 0.8653 - loss: 0.3250 - val_accuracy: 0.8537 - val_loss: 0.3490
Epoch 19/100
200/200            0s 889us/step -
accuracy: 0.8572 - loss: 0.3374 - val_accuracy: 0.8500 - val_loss: 0.3485
Epoch 20/100
200/200            0s 852us/step -
accuracy: 0.8621 - loss: 0.3286 - val_accuracy: 0.8550 - val_loss: 0.3471
Epoch 21/100
200/200            0s 877us/step -
accuracy: 0.8517 - loss: 0.3506 - val_accuracy: 0.8500 - val_loss: 0.3485
Epoch 22/100
200/200            0s 881us/step -
accuracy: 0.8641 - loss: 0.3319 - val_accuracy: 0.8519 - val_loss: 0.3481
Epoch 23/100
200/200            0s 894us/step -
accuracy: 0.8643 - loss: 0.3293 - val_accuracy: 0.8519 - val_loss: 0.3489
Epoch 24/100
200/200            0s 984us/step -
accuracy: 0.8637 - loss: 0.3317 - val_accuracy: 0.8519 - val_loss: 0.3480
Epoch 25/100
200/200            0s 893us/step -
accuracy: 0.8661 - loss: 0.3317 - val_accuracy: 0.8487 - val_loss: 0.3511
Epoch 26/100
200/200            0s 882us/step -
accuracy: 0.8710 - loss: 0.3168 - val_accuracy: 0.8531 - val_loss: 0.3485
Epoch 27/100
200/200            0s 931us/step -
accuracy: 0.8649 - loss: 0.3302 - val_accuracy: 0.8537 - val_loss: 0.3478
Epoch 28/100
200/200            0s 1ms/step -
accuracy: 0.8603 - loss: 0.3354 - val_accuracy: 0.8531 - val_loss: 0.3487
Epoch 29/100
200/200            0s 975us/step -
accuracy: 0.8701 - loss: 0.3166 - val_accuracy: 0.8487 - val_loss: 0.3498
Epoch 30/100
200/200            0s 936us/step -
```

```
accuracy: 0.8615 - loss: 0.3389 - val_accuracy: 0.8494 - val_loss: 0.3488
Epoch 31/100
200/200            0s 938us/step -
accuracy: 0.8630 - loss: 0.3314 - val_accuracy: 0.8556 - val_loss: 0.3496
Epoch 32/100
200/200            0s 890us/step -
accuracy: 0.8540 - loss: 0.3410 - val_accuracy: 0.8512 - val_loss: 0.3488
Epoch 33/100
200/200            0s 888us/step -
accuracy: 0.8622 - loss: 0.3304 - val_accuracy: 0.8519 - val_loss: 0.3487
Epoch 34/100
200/200            0s 887us/step -
accuracy: 0.8646 - loss: 0.3310 - val_accuracy: 0.8519 - val_loss: 0.3488
Epoch 35/100
200/200            0s 867us/step -
accuracy: 0.8689 - loss: 0.3254 - val_accuracy: 0.8512 - val_loss: 0.3516
Epoch 36/100
200/200            0s 909us/step -
accuracy: 0.8614 - loss: 0.3295 - val_accuracy: 0.8506 - val_loss: 0.3485
Epoch 37/100
200/200            0s 935us/step -
accuracy: 0.8738 - loss: 0.3091 - val_accuracy: 0.8525 - val_loss: 0.3503
Epoch 38/100
200/200            0s 900us/step -
accuracy: 0.8653 - loss: 0.3292 - val_accuracy: 0.8487 - val_loss: 0.3491
Epoch 39/100
200/200            0s 886us/step -
accuracy: 0.8574 - loss: 0.3453 - val_accuracy: 0.8506 - val_loss: 0.3495
Epoch 40/100
200/200            0s 869us/step -
accuracy: 0.8662 - loss: 0.3322 - val_accuracy: 0.8519 - val_loss: 0.3493
Epoch 41/100
200/200            0s 915us/step -
accuracy: 0.8674 - loss: 0.3182 - val_accuracy: 0.8506 - val_loss: 0.3496
Epoch 42/100
200/200            0s 908us/step -
accuracy: 0.8633 - loss: 0.3266 - val_accuracy: 0.8494 - val_loss: 0.3494
Epoch 43/100
200/200            0s 893us/step -
accuracy: 0.8684 - loss: 0.3219 - val_accuracy: 0.8506 - val_loss: 0.3482
Epoch 44/100
200/200            0s 906us/step -
accuracy: 0.8698 - loss: 0.3246 - val_accuracy: 0.8506 - val_loss: 0.3487
Epoch 45/100
200/200            0s 939us/step -
accuracy: 0.8672 - loss: 0.3203 - val_accuracy: 0.8506 - val_loss: 0.3488
Epoch 46/100
200/200            0s 895us/step -
```

```
accuracy: 0.8630 - loss: 0.3313 - val_accuracy: 0.8519 - val_loss: 0.3491
Epoch 47/100
200/200             0s 872us/step -
accuracy: 0.8592 - loss: 0.3351 - val_accuracy: 0.8487 - val_loss: 0.3512
Epoch 48/100
200/200             0s 1ms/step -
accuracy: 0.8597 - loss: 0.3372 - val_accuracy: 0.8481 - val_loss: 0.3503
Epoch 49/100
200/200             0s 949us/step -
accuracy: 0.8627 - loss: 0.3330 - val_accuracy: 0.8481 - val_loss: 0.3518
Epoch 50/100
200/200             0s 1ms/step -
accuracy: 0.8620 - loss: 0.3330 - val_accuracy: 0.8487 - val_loss: 0.3512
Epoch 51/100
200/200             0s 953us/step -
accuracy: 0.8626 - loss: 0.3380 - val_accuracy: 0.8506 - val_loss: 0.3505
Epoch 52/100
200/200             0s 1ms/step -
accuracy: 0.8626 - loss: 0.3330 - val_accuracy: 0.8506 - val_loss: 0.3501
Epoch 53/100
200/200             0s 905us/step -
accuracy: 0.8674 - loss: 0.3210 - val_accuracy: 0.8500 - val_loss: 0.3492
Epoch 54/100
200/200             0s 892us/step -
accuracy: 0.8716 - loss: 0.3264 - val_accuracy: 0.8487 - val_loss: 0.3501
Epoch 55/100
200/200             0s 908us/step -
accuracy: 0.8578 - loss: 0.3330 - val_accuracy: 0.8500 - val_loss: 0.3502
Epoch 56/100
200/200             0s 879us/step -
accuracy: 0.8718 - loss: 0.3143 - val_accuracy: 0.8519 - val_loss: 0.3493
Epoch 57/100
200/200             0s 1ms/step -
accuracy: 0.8700 - loss: 0.3164 - val_accuracy: 0.8487 - val_loss: 0.3496
Epoch 58/100
200/200             0s 1ms/step -
accuracy: 0.8683 - loss: 0.3156 - val_accuracy: 0.8512 - val_loss: 0.3503
Epoch 59/100
200/200             0s 897us/step -
accuracy: 0.8620 - loss: 0.3336 - val_accuracy: 0.8506 - val_loss: 0.3493
Epoch 60/100
200/200             0s 953us/step -
accuracy: 0.8683 - loss: 0.3153 - val_accuracy: 0.8487 - val_loss: 0.3500
Epoch 61/100
200/200             0s 976us/step -
accuracy: 0.8667 - loss: 0.3236 - val_accuracy: 0.8487 - val_loss: 0.3503
Epoch 62/100
200/200             0s 989us/step -
```

```
accuracy: 0.8654 - loss: 0.3220 - val_accuracy: 0.8531 - val_loss: 0.3493
Epoch 63/100
200/200            0s 1ms/step -
accuracy: 0.8686 - loss: 0.3171 - val_accuracy: 0.8512 - val_loss: 0.3504
Epoch 64/100
200/200            0s 1ms/step -
accuracy: 0.8691 - loss: 0.3240 - val_accuracy: 0.8494 - val_loss: 0.3506
Epoch 65/100
200/200            0s 946us/step -
accuracy: 0.8700 - loss: 0.3243 - val_accuracy: 0.8512 - val_loss: 0.3511
Epoch 66/100
200/200            0s 985us/step -
accuracy: 0.8669 - loss: 0.3217 - val_accuracy: 0.8500 - val_loss: 0.3515
Epoch 67/100
200/200            0s 925us/step -
accuracy: 0.8635 - loss: 0.3289 - val_accuracy: 0.8506 - val_loss: 0.3521
Epoch 68/100
200/200            0s 937us/step -
accuracy: 0.8663 - loss: 0.3148 - val_accuracy: 0.8512 - val_loss: 0.3523
Epoch 69/100
200/200            0s 947us/step -
accuracy: 0.8618 - loss: 0.3391 - val_accuracy: 0.8537 - val_loss: 0.3509
Epoch 70/100
200/200            0s 890us/step -
accuracy: 0.8637 - loss: 0.3324 - val_accuracy: 0.8519 - val_loss: 0.3516
Epoch 71/100
200/200            0s 983us/step -
accuracy: 0.8731 - loss: 0.3118 - val_accuracy: 0.8519 - val_loss: 0.3515
Epoch 72/100
200/200            0s 960us/step -
accuracy: 0.8679 - loss: 0.3173 - val_accuracy: 0.8519 - val_loss: 0.3531
Epoch 73/100
200/200            0s 897us/step -
accuracy: 0.8664 - loss: 0.3172 - val_accuracy: 0.8531 - val_loss: 0.3520
Epoch 74/100
200/200            0s 957us/step -
accuracy: 0.8615 - loss: 0.3299 - val_accuracy: 0.8550 - val_loss: 0.3521
Epoch 75/100
200/200            0s 906us/step -
accuracy: 0.8660 - loss: 0.3269 - val_accuracy: 0.8512 - val_loss: 0.3523
Epoch 76/100
200/200            0s 875us/step -
accuracy: 0.8670 - loss: 0.3244 - val_accuracy: 0.8494 - val_loss: 0.3507
Epoch 77/100
200/200            0s 978us/step -
accuracy: 0.8653 - loss: 0.3264 - val_accuracy: 0.8512 - val_loss: 0.3512
Epoch 78/100
200/200            0s 967us/step -
```

```
accuracy: 0.8748 - loss: 0.3060 - val_accuracy: 0.8525 - val_loss: 0.3519
Epoch 79/100
200/200            0s 1ms/step -
accuracy: 0.8619 - loss: 0.3265 - val_accuracy: 0.8506 - val_loss: 0.3517
Epoch 80/100
200/200            0s 1ms/step -
accuracy: 0.8605 - loss: 0.3334 - val_accuracy: 0.8519 - val_loss: 0.3515
Epoch 81/100
200/200            0s 909us/step -
accuracy: 0.8679 - loss: 0.3117 - val_accuracy: 0.8519 - val_loss: 0.3515
Epoch 82/100
200/200            0s 888us/step -
accuracy: 0.8575 - loss: 0.3369 - val_accuracy: 0.8500 - val_loss: 0.3527
Epoch 83/100
200/200            0s 1ms/step -
accuracy: 0.8723 - loss: 0.3160 - val_accuracy: 0.8519 - val_loss: 0.3528
Epoch 84/100
200/200            0s 934us/step -
accuracy: 0.8757 - loss: 0.3085 - val_accuracy: 0.8500 - val_loss: 0.3515
Epoch 85/100
200/200            0s 906us/step -
accuracy: 0.8719 - loss: 0.3147 - val_accuracy: 0.8512 - val_loss: 0.3528
Epoch 86/100
200/200            0s 948us/step -
accuracy: 0.8748 - loss: 0.3132 - val_accuracy: 0.8500 - val_loss: 0.3529
Epoch 87/100
200/200            0s 939us/step -
accuracy: 0.8716 - loss: 0.3158 - val_accuracy: 0.8462 - val_loss: 0.3529
Epoch 88/100
200/200            0s 916us/step -
accuracy: 0.8700 - loss: 0.3221 - val_accuracy: 0.8500 - val_loss: 0.3515
Epoch 89/100
200/200            0s 904us/step -
accuracy: 0.8550 - loss: 0.3405 - val_accuracy: 0.8500 - val_loss: 0.3524
Epoch 90/100
200/200            0s 915us/step -
accuracy: 0.8684 - loss: 0.3267 - val_accuracy: 0.8506 - val_loss: 0.3524
Epoch 91/100
200/200            0s 962us/step -
accuracy: 0.8651 - loss: 0.3158 - val_accuracy: 0.8500 - val_loss: 0.3518
Epoch 92/100
200/200            0s 867us/step -
accuracy: 0.8763 - loss: 0.3082 - val_accuracy: 0.8487 - val_loss: 0.3528
Epoch 93/100
200/200            0s 895us/step -
accuracy: 0.8731 - loss: 0.3100 - val_accuracy: 0.8494 - val_loss: 0.3534
Epoch 94/100
200/200            0s 959us/step -
```

```
accuracy: 0.8691 - loss: 0.3215 - val_accuracy: 0.8512 - val_loss: 0.3525
Epoch 95/100
200/200              0s 950us/step -
accuracy: 0.8738 - loss: 0.3153 - val_accuracy: 0.8487 - val_loss: 0.3529
Epoch 96/100
200/200              0s 883us/step -
accuracy: 0.8727 - loss: 0.3225 - val_accuracy: 0.8512 - val_loss: 0.3534
Epoch 97/100
200/200              0s 982us/step -
accuracy: 0.8696 - loss: 0.3132 - val_accuracy: 0.8487 - val_loss: 0.3532
Epoch 98/100
200/200              0s 972us/step -
accuracy: 0.8746 - loss: 0.3075 - val_accuracy: 0.8531 - val_loss: 0.3534
Epoch 99/100
200/200              0s 956us/step -
accuracy: 0.8662 - loss: 0.3235 - val_accuracy: 0.8494 - val_loss: 0.3543
Epoch 100/100
200/200              0s 890us/step -
accuracy: 0.8733 - loss: 0.3141 - val_accuracy: 0.8481 - val_loss: 0.3524
```

[24]:
```
y_log = model.predict(X_test_scaled)
```

```
63/63              0s 1ms/step
```

[25]:
```
y_pred = np.where(y_log > 0.5, 1, 0)
```

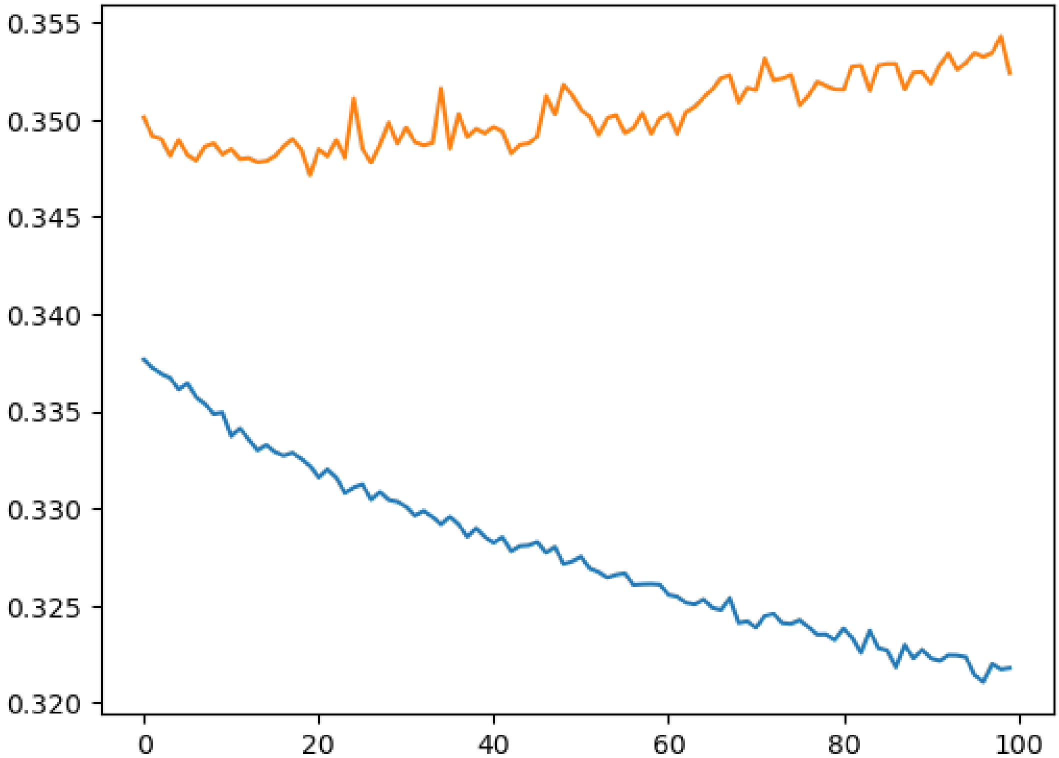
[26]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

[26]: 0.861

[27]:
```
import matplotlib.pyplot as plt
```

[28]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

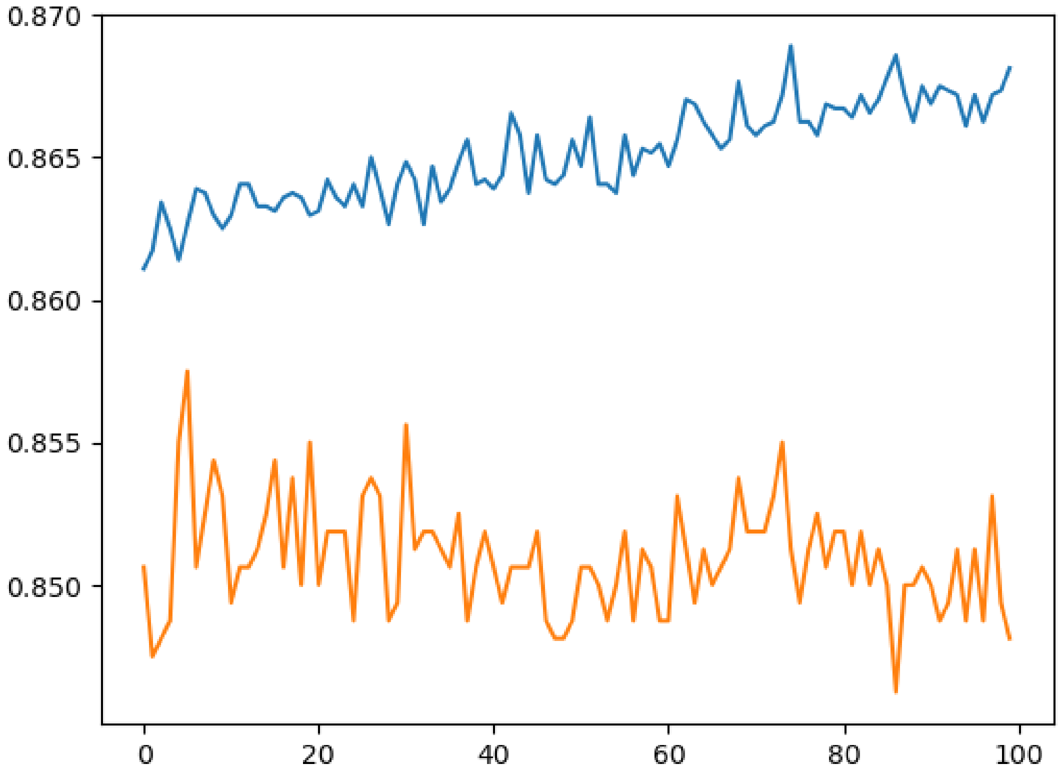
[28]: [<matplotlib.lines.Line2D at 0x1e077a871a0>]



[29]:

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

[29]: [<matplotlib.lines.Line2D at 0x1e079c918e0>]


0.870
0.865
0.860
0.855
0.850
0
20
40
60
80
100

[ ]:

[ ]: