Double-click (or enter) to edit

Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix.

```
import pandas as pd
import numpy as np
import  seaborn as sb
```

```
#Read the dataset
df=pd.read_csv("Churn_Modelling.csv")
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure
0	1	15634602	Hargrave	619	France	Female	42	2
1	2	15647311	Hill	608	Spain	Female	41	1
2	3	15619304	Onio	502	France	Female	42	8
3	4	15701354	Boni	699	France	Female	39	1
4	5	15737888	Mitchell	850	Spain	Female	43	2

```
df.isnull().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
```

```
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

Distinguish the feature and target set and divide the data set into training and test sets.

```
X=df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember',
states=pd.get_dummies(df['Geography'],drop_first = True)
gender=pd.get_dummies(df['Gender'],drop_first = True)
```

```
df=pd.concat([X,gender,states],axis=1)
```

```
df.head(5)
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	619	42	2	0.00	1	1	1
1	608	41	1	83807.86	1	0	1
2	502	42	8	159660.80	3	1	0
3	699	39	1	0.00	2	0	0
4	850	43	2	125510.82	1	1	1

```
X=df.drop(['Exited'],axis=1)
Y=df['Exited']
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=2)
```

Normalize the train and test data

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
```

```
X_train=sc.fit_transform(X_train)
```

```
X_test=sc.fit_transform(X_test)
```

```
X_train
```

```
    array([[ 0.40603944, -1.2255974 , -0.69902887, ..., -1.08690612,
            -0.58768841, -0.56877202],
           [-1.05419348,  0.37842889, -0.69902887, ..., -1.08690612,
             1.70158197, -0.56877202],
           [-2.13124471, -0.84817945, -0.69902887, ..., -1.08690612,
            -0.58768841, -0.56877202],
           ...,
           [ 2.06304133, -0.28205252, -0.69902887, ...,  0.92004266,
            -0.58768841, -0.56877202],
           [-1.03348096,  1.41632826, -0.00897076, ...,  0.92004266,
             1.70158197, -0.56877202],
           [ 0.03321401, -1.03688842,  0.68108736, ..., -1.08690612,
            -0.58768841, -0.56877202]])
```

```
X_test
```

```
    array([[ 0.83723003, -0.07477521,  1.74412033, ..., -1.11753101,
            -0.557843  , -0.58556189],
           [-0.05072348,  0.21871667, -1.0364621 , ..., -1.11753101,
             1.79261907, -0.58556189],
           [-0.32949959,  1.68617608,  0.70140192, ...,  0.89482975,
            -0.557843  , -0.58556189],
           ...,
           [ 0.29000287, -0.27043646, -0.6888893 , ...,  0.89482975,
             1.79261907, -0.58556189],
           [ 0.76495475, -0.17260584,  0.35382912, ...,  0.89482975,
             1.79261907, -0.58556189],
           [ 1.13665622,  0.02305542,  0.35382912, ...,  0.89482975,
            -0.557843  , -0.58556189]])
```

Initialize and build the model. Identify the points of improvement and implement the same.

```
import keras
```

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

```
Classifier = Sequential()
Classifier.add(Dense(150,input_dim=11,activation = 'tanh'))
Classifier.add(Dense(150,activation = 'tanh'))
Classifier.add(Dense(150,activation = 'tanh'))
Classifier.add(Dense(1, activation='sigmoid'))
Classifier.compile(loss='binary_crossentropy', optimizer='adam', metrics = ['accuracy'])
Classifier.fit(X_train,Y_train,epochs=100,batch_size=15)
```

```
Epoch 1/100
467/467 [==============================] - 2s 3ms/step - loss: 0.4106 - accuracy:
Epoch 2/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3591 - accuracy:
Epoch 3/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3525 - accuracy:
Epoch 4/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3462 - accuracy:
Epoch 5/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3410 - accuracy:
Epoch 6/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3413 - accuracy:
Epoch 7/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3360 - accuracy:
Epoch 8/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3369 - accuracy:
Epoch 9/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3291 - accuracy:
Epoch 10/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3263 - accuracy:
Epoch 11/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3220 - accuracy:
Epoch 12/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3172 - accuracy:
Epoch 13/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3158 - accuracy:
Epoch 14/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3100 - accuracy:
Epoch 15/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3060 - accuracy:
Epoch 16/100
467/467 [==============================] - 2s 3ms/step - loss: 0.3001 - accuracy:
Epoch 17/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2933 - accuracy:
Epoch 18/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2877 - accuracy:
Epoch 19/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2822 - accuracy:
Epoch 20/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2763 - accuracy:
Epoch 21/100
467/467 [==============================] - 2s 5ms/step - loss: 0.2686 - accuracy:
Epoch 22/100
467/467 [==============================] - 3s 6ms/step - loss: 0.2611 - accuracy:
Epoch 23/100
467/467 [==============================] - 2s 4ms/step - loss: 0.2512 - accuracy:
Epoch 24/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2446 - accuracy:
Epoch 25/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2372 - accuracy:
Epoch 26/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2267 - accuracy:
Epoch 27/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2163 - accuracy:
Epoch 28/100
467/467 [==============================] - 2s 3ms/step - loss: 0.2090 - accuracy:
Epoch 29/100
```

```
Classifier.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_4 (Dense)             (None, 150)               1800      
                                                                 
 dense_5 (Dense)             (None, 150)               22650     
                                                                 
 dense_6 (Dense)             (None, 150)               22650     
                                                                 
 dense_7 (Dense)             (None, 1)                 151       
                                                                 
=================================================================
Total params: 47,251
Trainable params: 47,251
Non-trainable params: 0
_________________________________________________________________
```

```
y_pred =Classifier.predict(X_test)
y_pred[y_pred>0.5]=1
y_pred[y_pred<0.5]=0
```

```
94/94 [==============================] - 0s 2ms/step
```

```
y_pred = y_pred.astype(int)
y_pred
```

```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [0],
       [1]])
```

```
count1=0
count0=0
for i in y_pred:
  if i==1:
    count1+=1
  else:
    count0+=1
print(count1)
print(count0)
```

```
510
2490
```

```
from sklearn.metrics import classification_report, confusion_matrix
cm=confusion_matrix(Y_test,y_pred)
print(cm)
```

```
[[2204  211]
```

```
 [ 286  299]]
```

```
print(classification_report(Y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.91      0.90      2415
           1       0.59      0.51      0.55       585

    accuracy                           0.83      3000
   macro avg       0.74      0.71      0.72      3000
weighted avg       0.83      0.83      0.83      3000
```