Practical - 9

AIM : Study of Artificial Neural Network (ANN) using the Churn_Modelling.csv dataset. The dataset includes customer attributes such as RowNumber, CustomerId, Surname, CreditScore, Geography, Gender, Age, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary, and Exited. Implement and analyze various activation functions within the ANN to predict customer churn. Compare the performance of different activation functions and evaluate their impact on the model's accuracy in predicting customer attrition based on the given attributes.

Imported Libraries :

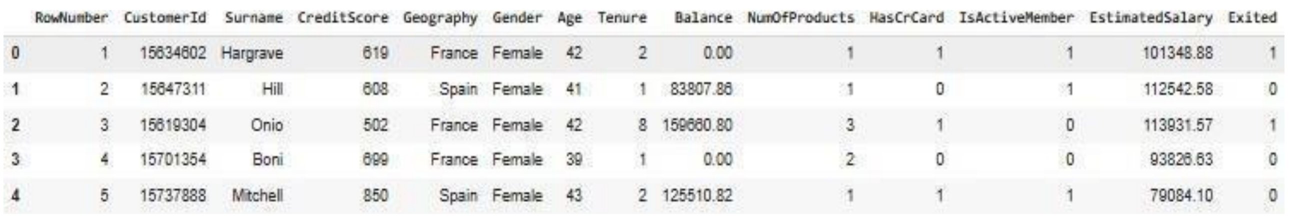
Code : import pandas as pd from
sklearn.preprocessing import OneHotEncoder from
sklearn.model_selection import train_test_split from
sklearn.preprocessing import
OneHotEncoder from keras.models import Sequential from
keras.layers import Dense from sklearn.metrics import
confusion_matrix, accuracy_score

Step 1: Load the Dataset

1. Import the Churn_Modelling.csv dataset, which contains customer details.
2. This dataset includes attributes like RowNumber, CustomerId, Surname, CreditScore, Geography, Gender, Age, Balance, NumOfProducts, and Exited (indicating if a customer has churned).

Code : df = pd.read_csv("/content/drive/MyDrive/DATASET/Churn_Modelling.csv")
df.head()

Output :



```
   RowNumber  CustomerId  Surname  CreditScore  Geography  Gender  Age  Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary  Exited
0          1    15634602  Hargrave         619     France  Female   42       2       0.00              1          1               1        101348.88       1
1          2    15647311      Hill         608      Spain  Female   41       1   83807.86              1          0               1        112542.58       0
2          3    15619304      Onio         502     France  Female   42       8  159660.80              3          1               0        113931.57       1
3          4    15701354      Boni         699     France  Female   39       1       0.00              2          0               0         93826.63       0
4          5    15737888  Mitchell         850      Spain  Female   43       2  125510.82              1          1               1         79084.10       0
```

Step 2: Prepare Feature and Target Data

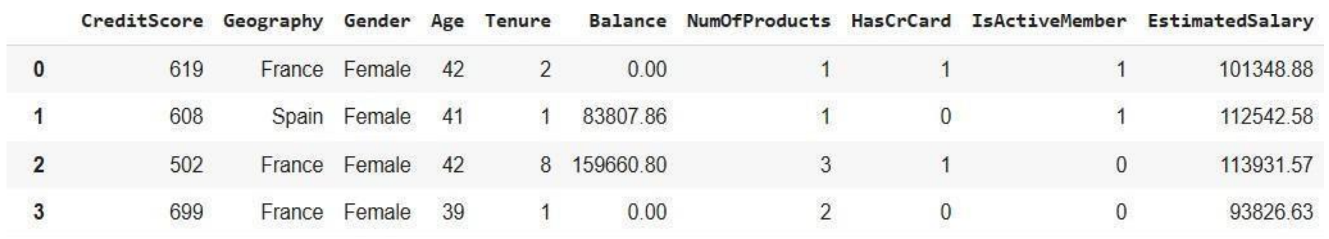
1. **Feature Data (X)** : Select all columns from CreditScore to EstimatedSalary.
2. **Target Data (Y)** : Select the Exited column (indicating churn).

Code :

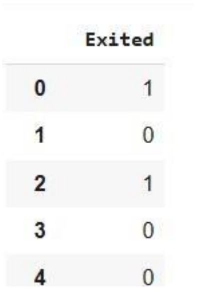
X = df.loc[:,'CreditScore':'EstimatedSalary']

```
Y = df['Exited']
```

Output :



	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	France	Female	42	2	0.00	1	1	1	101348.88
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58
2	502	France	Female	42	8	159660.80	3	1	0	113931.57
3	699	France	Female	39	1	0.00	2	0	0	93826.63



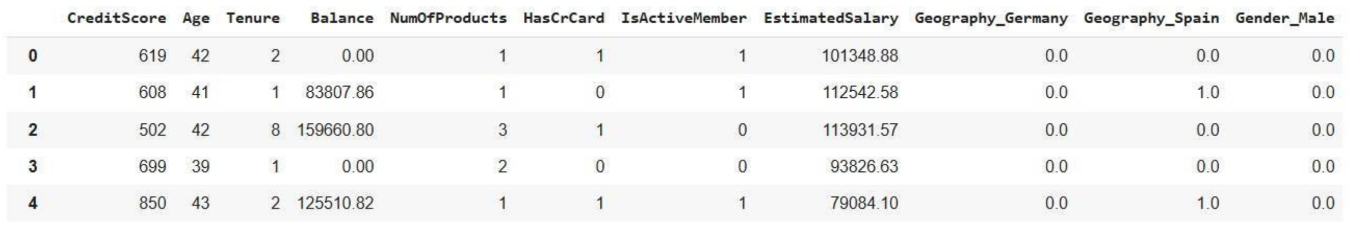
	Exited
0	1
1	0
2	1
3	0
4	0

Step 3: Process Categorical Variables

1. Identify categorical columns in X (e.g., Geography, Gender).
2. Convert these columns into numeric form using one-hot encoding, dropping the first column of each categorical variable to avoid redundancy.

Code :
```
categorical_cols = ['Geography', 'Gender'] enc
= OneHotEncoder(handle_unknown='ignore',
drop='first') encoded_features =
enc.fit_transform(X[categorical_cols]).toarray()
encoded_df    =         pd.DataFrame(encoded_features,
columns=enc.get_feature_names_out(categorical_cols))
X = X.drop(categorical_cols, axis=1) X =
pd.concat([X, encoded_df], axis=1)
```

Output :



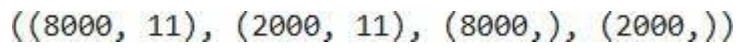
	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_Germany	Geography_Spain	Gender_Male
0	619	42	2	0.00	1	1	1	101348.88	0.0	0.0	0.0
1	608	41	1	83807.86	1	0	1	112542.58	0.0	1.0	0.0
2	502	42	8	159660.80	3	1	0	113931.57	0.0	0.0	0.0
3	699	39	1	0.00	2	0	0	93826.63	0.0	0.0	0.0
4	850	43	2	125510.82	1	1	1	79084.10	0.0	1.0	0.0

Step 4: Split Data into Training and Testing Sets

1. Divide the X dataset into training and testing subsets (e.g., 80% for training, 20% for testing).

   Code :

   X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
   X_train.shape, X_test.shape, y_train.shape, y_test.shape **Output :**



```
((8000, 11), (2000, 11), (8000,), (2000,))
```

Step 5: Create the ANN Model

1. Using the **Keras** library, create a **Sequential** ANN model.
2. Add input layers and two hidden layers: ○ Initialize weights using a uniform distribution. ○ Use **ReLU** (Rectified Linear Unit) activation for hidden layers.

   ○ Use **Sigmoid** activation for the output layer to predict churn probability. **Code :**

   classifier = Sequential() classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11)) classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu')) classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid')) **Output :**

Step 6: Compile the Model

1. Compile the model with:

   ○ **Adam** optimizer (for efficient gradient descent), ○ **Binary Crossentropy** loss function (suitable for binary classification), ○ **Accuracy** as the evaluation metric. **Code :**

   classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

Step 7: Model Summary

1. Print a summary of the neural network architecture, which includes the number of layers, and parameters. **Code :** classifier.summary() **Output :**

Step 8: Train the Model

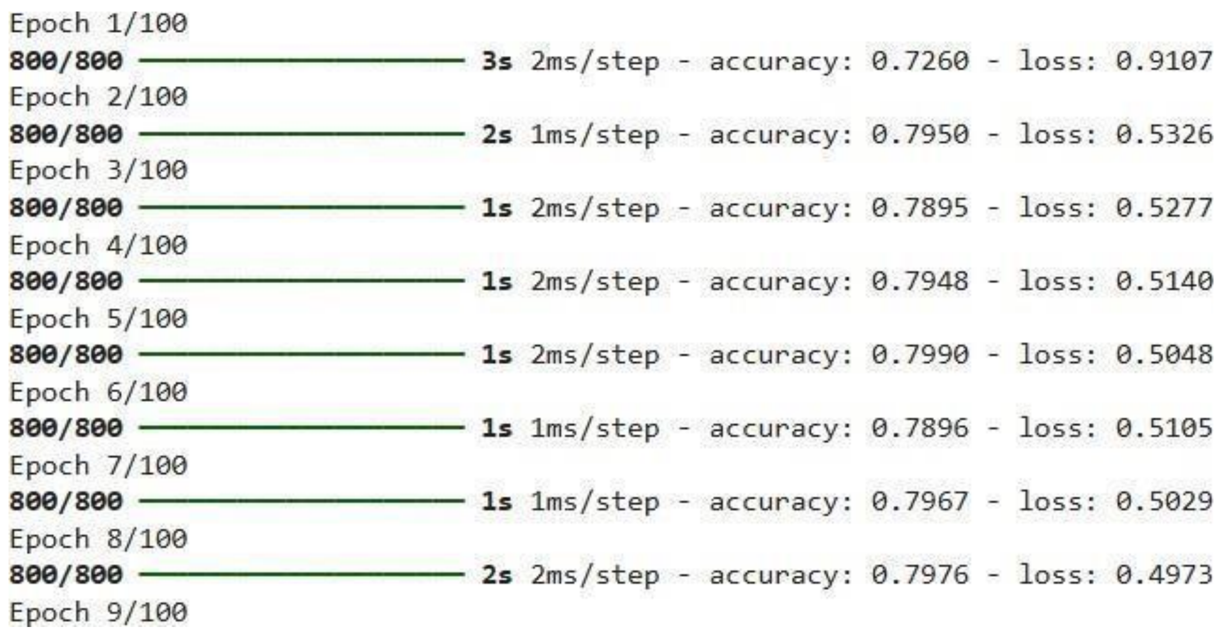
1. Train the ANN model using the training dataset with :

- Batch size of 10 (number of samples per gradient update),

100 epochs (iterations over the entire dataset).

Code : classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)

Output :



```
Epoch 1/100
800/800 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7260 - loss: 0.9107
Epoch 2/100
800/800 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.7950 - loss: 0.5326
Epoch 3/100
800/800 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7895 - loss: 0.5277
Epoch 4/100
800/800 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7948 - loss: 0.5140
Epoch 5/100
800/800 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.7990 - loss: 0.5048
Epoch 6/100
800/800 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.7896 - loss: 0.5105
Epoch 7/100
800/800 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.7967 - loss: 0.5029
Epoch 8/100
800/800 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7976 - loss: 0.4973
Epoch 9/100
```

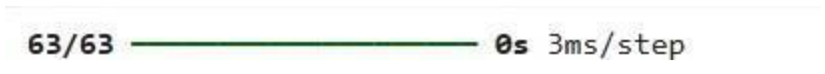
Step 9: Make Predictions

1. Use the test dataset to generate predictions.
2. Set a threshold of 0.5 for classifying outputs:
   - If prediction ≥ 0.5, classify as churned (1),
   - If prediction < 0.5, classify as not churned (0).

Code :

```
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

Output :

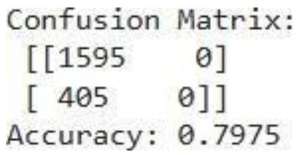


```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

Step 10: Evaluate the Model

1. Print a **Confusion Matrix** to evaluate performance.
2. Calculate and display the **accuracy** of the model in predicting customer churn.

Code : cm = confusion_matrix(y_test, y_pred) print("Confusion Matrix:\n", cm) accuracy = accuracy_score(y_test, y_pred) print("Accuracy:", accuracy) **Output :**



```
Confusion Matrix:
 [[1595    0]
 [ 405    0]]
Accuracy: 0.7975
```