**Assignment No.: ML 3**

**Title:** 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.

**Objective:**

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 (5 points).

**Contents for Theory: D:/Sem-I 2024-25/LP-III/BankCustomer.csv**

**Dataset Description**

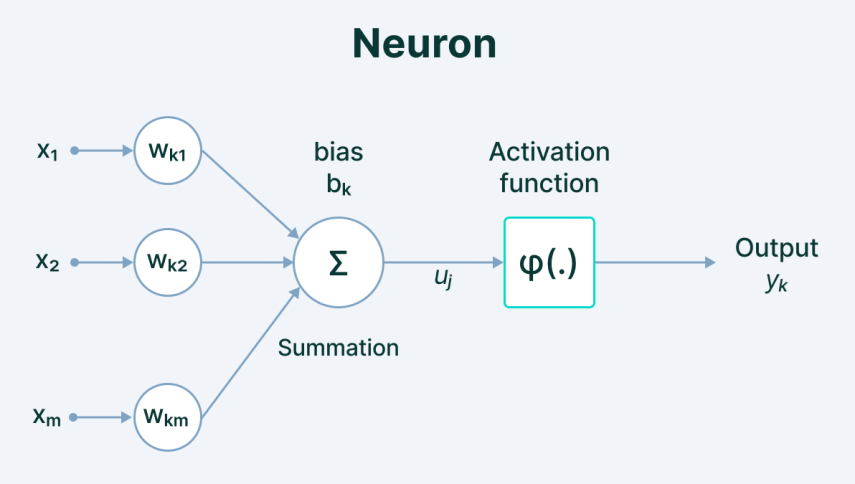
The dataset contains 10,000 customer records with 14 features:

* **CustomerId**: Unique ID for each customer.
* **CreditScore**: Customer’s credit score.
* **Geography**: Customer’s location.
* **Gender**: Male or Female.
* **Age**: Customer’s age.
* **Tenure**: Number of years the customer has been with the bank.
* **Balance**: Account balance.
* **NumOfProducts**: Number of bank products the customer is using.
* **HasCrCard**: Does the customer have a credit card? (1 for Yes, 0 for No)
* **IsActiveMember**: Is the customer an active member? (1 for Yes, 0 for No)
* **EstimatedSalary**: Customer’s estimated salary.
* **Exited**: Target variable (1 if the customer left, 0 otherwise).

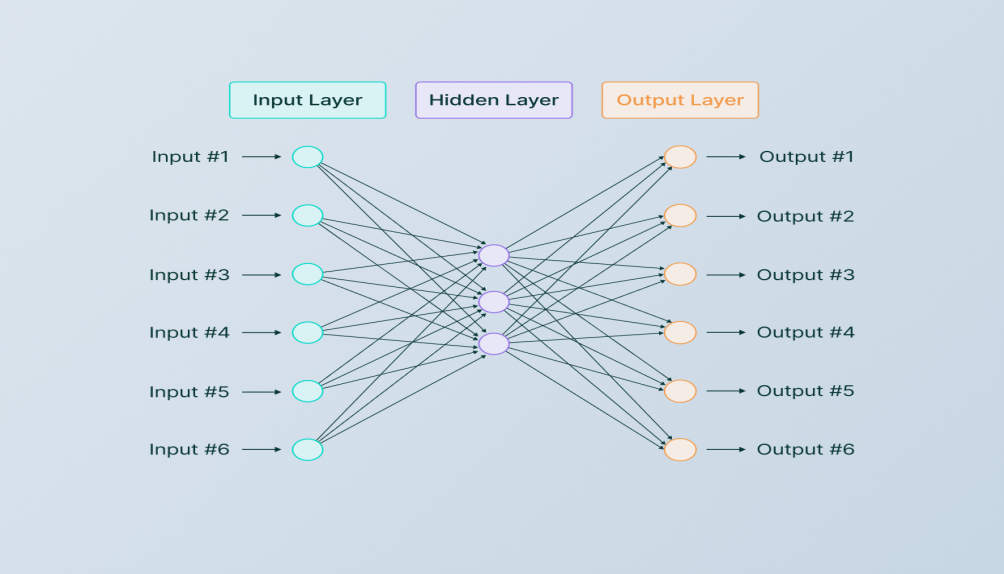
### Neural Networks: Neural networks are a subset of machine learning models inspired by the structure and function of the human brain. They are particularly powerful for tasks where traditional algorithms struggle, such as image recognition, natural language processing, and complex pattern recognition.

#### Structure of Neural Networks:

1. **Neurons (Nodes)**:
   * **Definition**: The basic unit of a neural network, analogous to a biological neuron. Each neuron receives input, processes it, and passes an output to the next layer of neurons.
   * **Function**: Each neuron performs a weighted sum of its inputs, adds a bias term, and applies an activation function to produce its output.



1. **Layers**:
   * **Input Layer**: The first layer of the network, which receives the input data.
   * **Hidden Layers**: Layers between the input and output layers. They process inputs through weighted connections and pass the result to the next layer. Networks with multiple hidden layers are called deep neural networks (DNNs).
   * **Output Layer**: The final layer that produces the network's output, which could be a classification label, a regression value, or another prediction.



1. **Weights and Biases**:
   * **Weights**: Parameters that determine the strength of the connection between neurons in different layers. They are adjusted during training to minimize the error of the model.
   * **Biases**: Additional parameters that are added to the weighted sum before applying the activation function, allowing the model to shift the activation function and fit the data better.
2. **Activation Functions**:
   * **Definition**: Functions applied to the output of each neuron to introduce non-linearity into the network, enabling it to learn complex patterns.
   * **Common Activation Functions**:
   * **ReLU (Rectified Linear Unit)**: *f(x)=max(0,x)* Introduces non-linearity while being computationally efficient.
   * **Sigmoid**: *f(x)=1/(1+e−x)​* Maps input to a value between 0 and 1.
   * **Tanh**: *f(x)=tanh(x)* Maps input to a value between -1 and 1.

#### Training Neural Networks:

1. **Forward Propagation**:
   * The process of passing the input data through the network, layer by layer, to compute the output predictions.
2. **Loss Function**:
   * **Definition**: A function that measures the difference between the predicted output and the actual target. It guides the network on how well it is performing.
   * **Examples**:
     + **Mean Squared Error (MSE)**: Used for regression tasks.
     + **Binary Cross-Entropy**: Used for binary classification tasks.
3. **Backpropagation**:
   * **Definition**: The process of calculating the gradient of the loss function with respect to each weight in the network and updating the weights using gradient descent to minimize the loss.
   * **Gradient Descent**: An optimization algorithm used to adjust the weights in the direction that reduces the loss function. Variants include Stochastic Gradient Descent (SGD), Adam, and RMSprop.
4. **Epochs and Batches**:
   * **Epoch**: One complete pass through the entire training dataset.
   * **Batch**: A subset of the training data used to update the model weights. Mini-batch training is a common practice where the dataset is divided into small batches.

**Steps of the Code:**

Step 1: Read the Dataset

Step 2: Data Preprocessing

Step 3: Splitting the Data

Step 4: Normalizing the Data

Step 5: Building the Neural Network

Step 6: Training the Model

Step 7: Evaluating the Model

|  |
| --- |
| # Step 1: Import necessary libraries  import pandas as pd  import numpy as np  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import accuracy\_score, confusion\_matrix  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Dropout  # Step 2: Load the dataset  data = "D:/Sem-I 2024-25/LP-III/BankCustomer.csv"  df = pd.read\_csv(data)  # Optional: Drop unnecessary columns (CustomerId, Surname, etc.)  df.drop(['CustomerId', 'Surname'], axis=1, inplace=True)  # Step 3: Feature set and target set  X = df[['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']]  y = df['Exited']  # Target variable  # Convert categorical data (Geography, Gender) to numeric using one-hot encoding  X = pd.get\_dummies(X, columns=['Geography', 'Gender'], drop\_first=True)  # Step 4: Split the dataset into train and test sets (80% train, 20% test)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Step 5: Normalize the train and test data using StandardScaler  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.transform(X\_test)  # Step 6: Initialize and build the neural network model  model = Sequential()  # Input layer  model.add(Dense(units=64, activation='relu', input\_shape=(X\_train.shape[1],)))  # Hidden layers  model.add(Dense(units=32, activation='relu'))  model.add(Dropout(0.5))  # Dropout for regularization  model.add(Dense(units=16, activation='relu'))  # Output layer  model.add(Dense(units=1, activation='sigmoid'))  # Binary classification  # Compile the model  model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  # Step 7: Train the model  history = model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.2)  # Step 8: Evaluate the model  y\_pred = (model.predict(X\_test) > 0.5).astype("int32")  # Step 9: Calculate accuracy score  accuracy = accuracy\_score(y\_test, y\_pred)  print(f'Accuracy: {accuracy \* 100:.2f}%')  # Step 10: Print confusion matrix  cm = confusion\_matrix(y\_test, y\_pred)  print("Confusion Matrix:\n", cm)  # Step 11: Visualize the confusion matrix (optional)  import seaborn as sns  import matplotlib.pyplot as plt  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  plt.xlabel('Predicted')  plt.ylabel('True')  plt.title('Confusion Matrix')  plt.show()  # Step 12: Count customers predicted to leave  count\_leave = np.sum(y\_pred)  print(f"Count of customers predicted to leave the bank: {count\_leave}")  # Step 13: Possible points of improvement  # - Tune the model (layers, neurons, dropout, epochs, etc.)  # - Use more advanced models (like LSTM or GRU if you have sequential data).  # - Try cross-validation for better accuracy measurement.  # - Use techniques like SMOTE to handle class imbalance if necessary. |