1. **Build a Deep Feed Forward ANN by implementing the Backpropagation algorithm and test the same using appropriate data sets. Use the number of hidden layers >=4.**

import numpy as np

import pandas as pd

import tensorflow as tf

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

# Set random seed for reproducibility

np.random.seed(42)

# Generate synthetic dataset (customer purchase prediction)

num\_samples = 1000

age = np.random.randint(18, 65, num\_samples)

income = np.random.randint(20000, 120000, num\_samples)

website\_activity = np.random.uniform(0, 1, num\_samples) \* 100

email\_engagement = np.random.uniform(0, 1, num\_samples) \* 100

# Generate target labels (Purchased = 1, Not Purchased = 0)

purchase = (income > 50000) & (website\_activity > 40) & (email\_engagement > 30)

purchase = purchase.astype(int)

# Create DataFrame

df = pd.DataFrame({

'Age': age,

'Income': income,

'Website\_Activity\_Score': website\_activity.round(2),

'Email\_Engagement\_Score': email\_engagement.round(2),

'Purchased': purchase

})

# Save dataset to CSV file

df.to\_csv("customer\_purchase\_data.csv", index=False)

# Load dataset

df = pd.read\_csv("customer\_purchase\_data.csv")

# Separate features (X) and target variable (y)

X = df.drop(columns=['Purchased']).values

y = df['Purchased'].values

# Split into training (80%) and testing (20%) datasets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize feature values

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Function to build and train a Deep ANN model

def build\_and\_train\_ann(activation\_function):

# Define a deep feedforward neural network

model = Sequential([

Dense(64, activation=activation\_function, input\_shape=(X\_train.shape[1],)), # Input layer

Dense(128, activation=activation\_function), # Hidden layer 1

Dense(64, activation=activation\_function), # Hidden layer 2

Dense(32, activation=activation\_function), # Hidden layer 3

Dense(16, activation=activation\_function), # Hidden layer 4

Dense(1, activation='sigmoid') # Output layer (binary classification)

])

# Compile the model using Backpropagation

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=32, validation\_data=(X\_test, y\_test), verbose=1)

# Evaluate model performance

loss, accuracy = model.evaluate(X\_test, y\_test, verbose=0)

print(f"Activation: {activation\_function} | Test Accuracy: {accuracy:.4f}")

return history

# Define activation functions for comparison

activations = ['relu', 'tanh', 'sigmoid']

histories = {}

plt.figure(figsize=(12, 6))

# Train and compare different activation functions

for activation in activations:

print(f"\nTraining model with {activation} activation...")

histories[activation] = build\_and\_train\_ann(activation)

# Plot accuracy over epochs

plt.plot(histories[activation].history['val\_accuracy'], label=f"{activation} Activation")

# Graph labels

plt.title('Deep ANN: Comparison of Activation Functions')

plt.xlabel('Epochs')

plt.ylabel('Validation Accuracy')

plt.legend()

plt.show()