1. **Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets. Vary the activation functions used and compare the results.**

# Import necessary libraries

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.datasets import make\_moons

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

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# Generate a synthetic dataset (make\_moons for non-linear classification)

X, y = make\_moons(n\_samples=1000, noise=0.2, random\_state=42)

# Split into training and testing sets

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

# Standardize the dataset

scaler = StandardScaler()

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

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

# Function to build and train an ANN with a specified activation function

def train\_model(activation\_function):

# Define a sequential ANN model

model = Sequential([

Dense(10, activation=activation\_function, input\_shape=(2,)), # Hidden layer with 10 neurons

Dense(5, activation=activation\_function), # Another hidden layer

Dense(1, activation='sigmoid') # Output layer (Binary Classification)

])

# Compile the model with 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=0)

# Evaluate the model

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

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

return history

# Train and compare models with different activation functions

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

histories = {}

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

for activation in activations:

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

histories[activation] = train\_model(activation)

# Plot accuracy over epochs

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

# Graph labels

plt.title('Comparison of Activation Functions')

plt.xlabel('Epochs')

plt.ylabel('Validation Accuracy')

plt.legend()

plt.show()