Artificial Intelligence Assignment # 6

Name: Ahmed Kasteer

Roll Number: 20F-0336

Section: 6C

Task 1

1)

import numpy as np  
  
# sigmoid activation function and its derivative  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
def sigmoid\_derivative(x):  
 return sigmoid(x) \* (1 - sigmoid(x))  
  
# input data  
X = np.array([[0, 0], [1, 0], [0, 1], [1, 1]])  
  
# target output data  
Y = np.array([[1], [0], [0], [1]])  
  
# initialize weights  
w1 = np.array([[0.1, 0.1], [0.1, 0.1]])  
b1 = np.array([[0.1, 0.1]])  
w2 = np.array([[0.1], [0.1]])  
b2 = np.array([[0.1]])  
  
# hyperparameters  
learning\_rate = 0.1  
epochs = 2  
  
for i in range(epochs):  
 # forward propagation  
 Z1 = np.dot(X, w1) + b1  
 H = sigmoid(Z1)  
 Z2 = np.dot(H, w2) + b2  
 Y\_pred = sigmoid(Z2)  
  
 # compute loss  
 error = Y\_pred - Y  
 loss = np.mean(error \*\* 2)  
  
 # backward propagation  
 delta2 = error \* sigmoid\_derivative(Y\_pred)  
 delta1 = np.dot(delta2, w2.T) \* sigmoid\_derivative(H)  
  
 # update weights and biases  
 w2 -= learning\_rate \* np.dot(H.T, delta2)  
 b2 -= learning\_rate \* np.sum(delta2, axis=0, keepdims=True)  
 w1 -= learning\_rate \* np.dot(X.T, delta1)  
 b1 -= learning\_rate \* np.sum(delta1, axis=0)  
  
 # print loss  
 print(f"epoch: {i+1}, loss: {loss:.4f}")  
  
# print final weights  
print(f"w1: {w1}")  
print(f"b1: {b1}")  
print(f"w2: {w2}")  
print(f"b2: {b2}")



2)

import numpy as np  
  
# tanh activation function and its derivative  
def tanh(x):  
 return np.tanh(x)  
  
def tanh\_derivative(x):  
 return 1 - np.tanh(x) \*\* 2  
  
# input data  
X = np.array([[0, 0], [1, 0], [0, 1], [1, 1]])  
  
# target output data  
Y = np.array([[1], [0], [0], [1]])  
  
# initialize weights  
w1 = np.array([[0.1, 0.1], [0.1, 0.1]])  
b1 = np.array([[0.1, 0.1]])  
w2 = np.array([[0.1], [0.1]])  
b2 = np.array([[0.1]])  
  
# hyperparameters  
learning\_rate = 0.1  
epochs = 2  
  
# training loop  
for i in range(epochs):  
 # forward propagation  
 Z1 = np.dot(X, w1) + b1  
 H = tanh(Z1)  
 Z2 = np.dot(H, w2) + b2  
 Y\_pred = tanh(Z2)  
  
 # compute loss  
 error = Y\_pred - Y  
 loss = np.mean(error \*\* 2)  
  
 # backward propagation  
 delta2 = error \* tanh\_derivative(Y\_pred)  
 delta1 = np.dot(delta2, w2.T) \* tanh\_derivative(H)  
  
 # update weights and biases  
 w2 -= learning\_rate \* np.dot(H.T, delta2)  
 b2 -= learning\_rate \* np.sum(delta2, axis=0, keepdims=True)  
 w1 -= learning\_rate \* np.dot(X.T, delta1)  
 b1 -= learning\_rate \* np.sum(delta1, axis=0)  
  
 # print loss  
 print(f"epoch: {i+1}, loss: {loss:.4f}")  
  
# print final weights  
print(f"w1: {w1}")  
print(f"b1: {b1}")  
print(f"w2: {w2}")  
print(f"b2: {b2}")



Task 2

import numpy as np  
import matplotlib.pyplot as plt  
  
# Define the XAND function dataset  
X = np.array([[0, 0], [1, 0], [0, 1], [1, 1]])  
Y = np.array([[1], [0], [0], [1]])  
  
# Set the random seed for reproducibility  
np.random.seed(42)  
  
# Initialize the weights randomly  
w1 = np.random.randn(2, 2)  
b1 = np.random.randn(1, 2)  
w2 = np.random.randn(2, 1)  
b2 = np.random.randn(1, 1)  
  
# Hyperparameters  
learning\_rate = 0.1  
epochs = 1000  
  
# Lists to store the error values for plotting  
errors = []  
  
# Sigmoid activation function  
def sigmoid(x):  
 return 1 / (1 + np.exp(-x))  
  
# Forward propagation function  
def forward\_propagation(X, w1, b1, w2, b2):  
 Z1 = np.dot(X, w1) + b1  
 H = sigmoid(Z1)  
 Z2 = np.dot(H, w2) + b2  
 Y\_pred = sigmoid(Z2)  
 return Y\_pred, H  
  
# Backpropagation function  
def back\_propagation(X, Y, Y\_pred, H, w2):  
 delta2 = (Y\_pred - Y) \* Y\_pred \* (1 - Y\_pred)  
 delta1 = np.dot(delta2, w2.T) \* H \* (1 - H)  
 dw2 = np.dot(H.T, delta2)  
 db2 = np.sum(delta2, axis=0, keepdims=True)  
 dw1 = np.dot(X.T, delta1)  
 db1 = np.sum(delta1, axis=0, keepdims=True)  
 return dw1, db1, dw2, db2  
  
# Training loop  
for epoch in range(epochs):  
 # Forward propagation  
 Y\_pred, H = forward\_propagation(X, w1, b1, w2, b2)  
  
 # Compute the error (mean squared error)  
 error = np.mean((Y\_pred - Y) \*\* 2)  
 errors.append(error)  
  
 # Backpropagation  
 dw1, db1, dw2, db2 = back\_propagation(X, Y, Y\_pred, H, w2)  
  
 # Update weights and biases  
 w1 -= learning\_rate \* dw1  
 b1 -= learning\_rate \* db1  
 w2 -= learning\_rate \* dw2  
 b2 -= learning\_rate \* db2  
  
# Plot the errors  
plt.plot(errors)  
plt.xlabel('Epochs')  
plt.ylabel('Error')  
plt.title('Error vs. Epochs')  
plt.show()  
  
# Print the final weights and biases  
print('Final Weights:')  
print('w1:', w1)  
print('b1:', b1)  
print('w2:', w2)  
print('b2:', b2)

