**Sample Code:**

import numpy as np def sigmoid(x):

"""Sigmoid activation function.""" return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

"""Derivative of the sigmoid activation function.""" return x \* (1 - x)

def initialize\_weights(input\_size, hidden\_size, output\_size): """Initialize weights randomly."""

np.random.seed(42)

weights\_input\_hidden = np.random.rand(input\_size, hidden\_size) weights\_hidden\_output = np.random.rand(hidden\_size, output\_size) return weights\_input\_hidden, weights\_hidden\_output

def train\_neural\_network(X, y, epochs, learning\_rate, hidden\_size): input\_size = X.shape[1]

output\_size = y.shape[1]

weights\_input\_hidden, weights\_hidden\_output = initialize\_weights(input\_size, hidden\_size, output\_size)

for epoch in range(epochs): # Forward pass

hidden\_layer\_input = np.dot(X, weights\_input\_hidden) hidden\_layer\_output = sigmoid(hidden\_layer\_input)

output\_layer\_input = np.dot(hidden\_layer\_output, weights\_hidden\_output) predicted\_output = sigmoid(output\_layer\_input)

# Backward pass

error = y - predicted\_output

output\_delta = error \* sigmoid\_derivative(predicted\_output) hidden\_layer\_error = output\_delta.dot(weights\_hidden\_output.T)

hidden\_layer\_delta = hidden\_layer\_error \* sigmoid\_derivative(hidden\_layer\_output) # Update weights

weights\_hidden\_output += hidden\_layer\_output.T.dot(output\_delta) \* learning\_rate weights\_input\_hidden += X.T.dot(hidden\_layer\_delta) \* learning\_rate

return weights\_input\_hidden, weights\_hidden\_output

# Example usage

# Assuming X\_train and y\_train are your training data and labels # Convert labels to one-hot encoding

num\_classes = len(np.unique(y\_train)) y\_train\_onehot = np.eye(num\_classes)[y\_train] # Train the neural network

epochs = 10000

learning\_rate = 0.01

hidden\_size = 4

weights\_input\_hidden, weights\_hidden\_output = train\_neural\_network(X\_train, y\_train\_onehot, epochs, learning\_rate, hidden\_size)

# Make predictions

hidden\_layer\_output = sigmoid(np.dot(X\_test, weights\_input\_hidden)) predicted\_output = sigmoid(np.dot(hidden\_layer\_output, weights\_hidden\_output)) # Round predictions to the nearest class

predicted\_labels = np.argmax(predicted\_output, axis=1) # Print predictions or evaluate other metrics as needed print(predicted\_labels)