Q.1

import numpy as np

from scipy.stats import norm, binom, poisson

# User input for basic probability

p\_event = float(input("Enter the probability of the event occurring (0 to 1): "))

print(f"\nProbability of event occurring: {p\_event}")

# User input for Normal Distribution

mu = float(input("\nEnter the mean (μ) for the normal distribution: "))

sigma = float(input("Enter the standard deviation (σ): "))

x = np.linspace(mu - 3\*sigma, mu + 3\*sigma, 100)

pdf = norm.pdf(x, mu, sigma)

print(f"\nNormal Distribution PDF calculated for μ={mu}, σ={sigma}")

# User input for Binomial Distribution

n = int(input("\nEnter number of trials (n) for binomial distribution: "))

p = float(input("Enter probability of success (p): "))

binomial\_probs = binom.pmf(range(n + 1), n, p)

print(f"\nBinomial Distribution PMF calculated for n={n}, p={p}")

print("Probabilities:", binomial\_probs)

# User input for Poisson Distribution

lambda\_ = float(input("\nEnter the average rate (λ) for Poisson distribution: "))

poisson\_probs = poisson.pmf(range(10), lambda\_)

print(f"\nPoisson Distribution PMF calculated for λ={lambda\_}")

print("Probabilities:", poisson\_probs)

print("\nAll probability concepts calculated successfully!")

Q.2.)

import numpy as np

import matplotlib.pyplot as plt

# Input from user

user\_input = input("Enter activation functions to plot (e.g., relu sigmoid tanh): ").lower().split()

# Prepare x values

x = np.linspace(-10, 10, 100)

# Dictionary of activation functions

activations = {

    "relu": np.maximum(0, x),

    "sigmoid": 1 / (1 + np.exp(-x)),

    "tanh": np.tanh(x),

}

# Plotting

plt.figure(figsize=(8, 6))

for func in user\_input:

    if func in activations:

        plt.plot(x, activations[func], label=func.capitalize())

    else:

        print(f"'{func}' is not a recognized activation function. Skipping.")

plt.legend()

plt.title("Activation Functions")

plt.xlabel("x")

plt.ylabel("f(x)")

plt.grid(True)

plt.show()

Q.3.)

def ANDNOT(a, b):

    return a and not b

# Taking input from the user

a = int(input("Enter first value (0 or 1): "))

b = int(input("Enter second value (0 or 1): "))

# Convert the result to int for consistent output

result = int(ANDNOT(a, b))

print("ANDNOT({}, {}) = {}".format(a, b, result))

Q.4.)

import numpy as np

# Inputs: ASCII values of digits 0-9

X = np.array([[i] for i in range(48, 58)])  # ASCII 0-9

# Labels: Even -> 0, Odd -> 1

y = np.array([(i - 48) % 2 for i in range(48, 58)])

# Initialize weights and bias

w = np.random.rand(1)

b = np.random.rand(1)

lr = 0.1

# Training Perceptron

for \_ in range(100):

    for i in range(len(X)):

        pred = 1 if np.dot(X[i], w) + b > 0 else 0

        w += lr \* (y[i] - pred) \* X[i]

        b += lr \* (y[i] - pred)

print("Perceptron trained to recognize even and odd numbers!")

# User input loop

while True:

    user\_input = input("Enter a digit (0-9) or 'q' to quit: ")

    if user\_input.lower() == 'q':

        break

    if user\_input.isdigit() and 0 <= int(user\_input) <= 9:

        ascii\_val = ord(user\_input)

        pred = 1 if np.dot([ascii\_val], w) + b > 0 else 0

        label = "Odd" if pred == 1 else "Even"

        print(f"The digit {user\_input} is predicted to be: {label}")

    else:

        print("Invalid input. Please enter a single digit from 0 to 9.")

Q.5.) import numpy as np

import matplotlib.pyplot as plt

# Get number of data points from user

num\_points = int(input("Enter the number of data points: "))

# Initialize lists to store input points and labels

X = []

y = []

# Collect input data from the user

for i in range(num\_points):

    x1 = float(input(f"Enter x1 for point {i+1}: "))

    x2 = float(input(f"Enter x2 for point {i+1}: "))

    label = int(input(f"Enter label for point {i+1} (0 or 1): "))

    X.append([x1, x2])

    y.append(label)

# Convert lists to NumPy arrays

X = np.array(X)

y = np.array(y)

# Plot the data points

for i in range(len(X)):

    plt.scatter(X[i, 0], X[i, 1], color="red" if y[i] else "blue", label=f'Class {y[i]}' if i == 0 else "")

plt.title("Perceptron Learning Example (User Input)")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend(["Class 1 (Red)", "Class 0 (Blue)"])

plt.grid(True)

plt.show()

Q.6.)

import numpy as np

def get\_user\_vectors():

    num\_pairs = int(input("Enter the number of vector pairs: "))

    vector\_length = int(input("Enter the length of each vector: "))

    X = []

    for i in range(num\_pairs):

        vector = input(f"Enter vector {i+1} (comma-separated, e.g. 1,-1,1): ")

        vector\_list = list(map(int, vector.strip().split(',')))

        if len(vector\_list) != vector\_length:

            print("Error: Vector length does not match specified length.")

            return None

        X.append(vector\_list)

    return np.array(X)

def main():

    X = get\_user\_vectors()

    if X is None:

        return

    W = X.T @ X  # Hebbian learning rule

    print("\nAssociative Memory Matrix:\n", W)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Q.7.)

import numpy as np

# Sigmoid Activation Function

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

# Derivative of Sigmoid

def sigmoid\_derivative(x):

    return x \* (1 - x)

# User Input

n\_samples = int(input("Enter number of training samples: "))

X = []

y = []

print("\nEnter input values (space-separated, two values per sample):")

for \_ in range(n\_samples):

    x\_input = list(map(float, input("Input: ").split()))

    X.append(x\_input)

print("\nEnter expected output (one value per sample):")

for \_ in range(n\_samples):

    y\_input = [float(input("Output: "))]

    y.append(y\_input)

X = np.array(X)

y = np.array(y)

# Weight Initialization (better distribution)

np.random.seed(1)

W1 = np.random.randn(2, 4)  # 2 inputs -> 4 hidden neurons

W2 = np.random.randn(4, 1)  # 4 hidden neurons -> 1 output

# Training

for epoch in range(50000):

    # Forward Propagation

    l1 = sigmoid(np.dot(X, W1))  # Hidden Layer

    l2 = sigmoid(np.dot(l1, W2))  # Output Layer

    # Error

    l2\_error = y - l2

    # Backpropagation

    l2\_delta = l2\_error \* sigmoid\_derivative(l2)

    l1\_error = l2\_delta.dot(W2.T)

    l1\_delta = l1\_error \* sigmoid\_derivative(l1)

    # Weight Updates

    W2 += l1.T.dot(l2\_delta) \* 0.1

    W1 += X.T.dot(l1\_delta) \* 0.1

print("\n✅ Neural Network trained successfully!")

# Predictions

predictions = sigmoid(np.dot(sigmoid(np.dot(X, W1)), W2))

print("\n📊 Predictions after training:")

print(np.round(predictions, 3))

Q.11.)

import numpy as np

print("Enter 4 binary vectors (1 and -1 only) of same length:")

# Get 4 binary vectors from user

vectors = [

np.array(list(map(int, input(f"Vector {i+1}: ").split())))

for i in range(4)

]

# Initialize weight matrix

size = len(vectors[0])

W = np.zeros((size, size))

# Training using Hebbian learning rule

for v in vectors:

W += np.outer(v, v)

# No self-connections (diagonal should be 0)

np.fill\_diagonal(W, 0)

# Get noisy test input from user

test = np.array(list(map(int, input("Enter noisy pattern: ").split())))

print("Running network...")

# Run the network for 10 iterations

for \_ in range(10):

test = np.sign(W @ test)

print("Recovered pattern:")

print(test)

Q.12.)

import tensorflow as tf

import numpy as np

print("Enter the number of data points :")

n=int(input())

x=[]

y=[]

for i in range(n):

    features=list(map(float,input(f"Enter the features for the data points {i+1} :").split()))

    label=int(input("Enter label :"))

    x.append(features)

    y.append([label])

x=np.array(x)

y=np.array(y)

model = tf.keras.models.Sequential([

    tf.keras.Input(shape=(len(x[0]),)),

    tf.keras.layers.Dense(1, activation='sigmoid')

])

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

model.fit(x,y,epochs=100,verbose=0)

print("Training Complete,Accuracy :")

loss,acc=model.evaluate(x,y,verbose=0)

print(f"Accuracy :{acc\*100:.2f}%")

Q.13.)

import tensorflow as tf

from tensorflow.keras import layers, models

import numpy as np

# Load MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

# Normalize the pixel values to be between 0 and 1

x\_train = x\_train[..., np.newaxis] / 255.0

x\_test = x\_test[..., np.newaxis] / 255.0

# Define the CNN model

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10, activation='softmax')  # 10 output classes for digits 0-9

])

# Compile the model

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=5, validation\_data=(x\_test, y\_test))