Assignment No 01

Name- Thorve Avishkar Shrikrushna

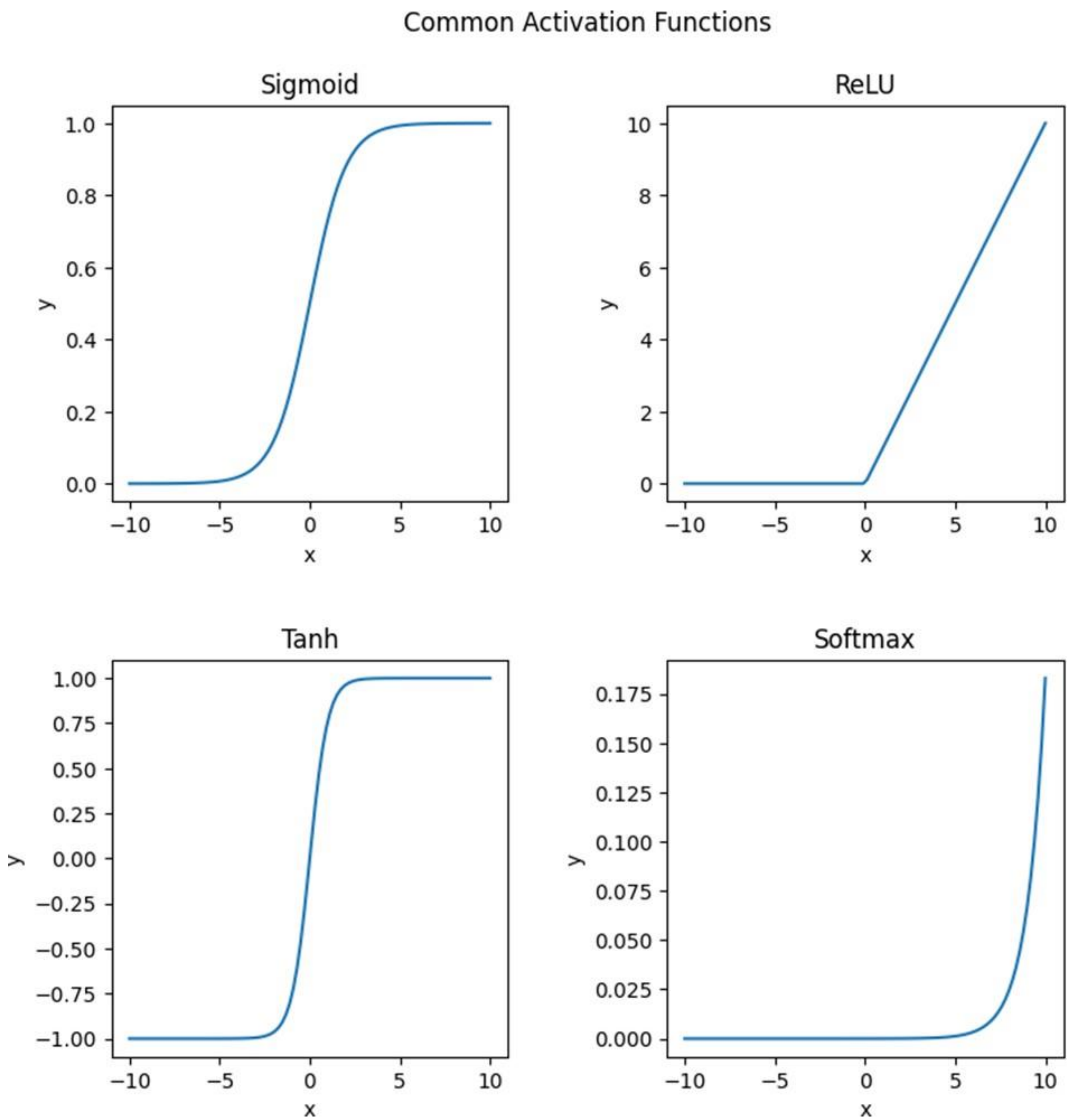
Roll No- 63

Title- Activation functions that are being used in neural networks.

Program:

```
import numpy as np import

matplotlib.pyplot as plt def

sigmoid(x): return 1 / (1 +

np.exp(-x)) def relu(x): return

np.maximum(0, x) def

tanh(x): return np.tanh(x) def

softmax(x):

    return np.exp(x) / np.sum(np.exp(x)) x =

np.linspace(-10, 10, 100)  axs[0, 0].plot(x,

sigmoid(x)) axs[0, 0].set_title('Sigmoid')

axs[0, 1].plot(x, relu(x)) axs[0,

1].set_title('ReLU') axs[1, 0].plot(x,

tanh(x)) axs[1, 0].set_title('Tanh') axs[1,

1].plot(x, softmax(x)) axs[1,

1].set_title('Softmax') fig.suptitle('Common

Activation Functions') for ax in axs.flat:

    ax.set(xlabel='x', ylabel='y')

plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.4, hspace=0.4) plt.show()
```

Output:


Common Activation Functions
Sigmoid
ReLU
Tanh
Softmax
1.0
0.8
0.6
0.4
0.2
0.0
10
8
6
4
2
0
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
−10
−5
0
5
10
x
y

Assignment No 02

Name- Thorve Avishkar Shrikrushna

Roll No- 63

Title- AND NOT function using Mc Culloch-Pitts Neural Net.

Program:

```
import numpy as np def

linear_threshold_gate(dot, T):

    '''Returns the binary threshold output''' if
    dot >= T: return 1 else:

        return 0 input_table

= np.array([

    [0,0],

    [0,1],
    [1,0],
    [1,1]

])

print(f'input          table:\n{input_table}')

weights = np.array([1,-1]) dot_products =

input_table @ weights T = 1 for i in

range(0,4):

        activation = linear_threshold_gate(dot_products[i], T)

        print(f'Activation: {activation}')
```

Output:

input
table: [[0
0]

[0 1]

[1 0]

[1 1]]

Activation: 0

Activation: 0

Activation: 1

Activation: 0

Assignment No 03

Name- Thorve Avishkar Shrikrushna

Roll No- 63

Title- Perceptron Neural Network to recognize even and odd numbers. Given numbers are in ASCII from 0 to 9.

Program:

```
import numpy as np class

Perceptron:

    def_init_(self, input_size, lr=0.1): self.W = np.zeros(input_size
        + 1) self.lr = lr

    def activation_fn(self, x):
        return 1 if x >= 0 else 0

    def predict(self, x):

        x = np.insert(x, 0, 1) z =
        self.W.T.dot(x)

        a = self.activation_fn(z)
        return a

    def train(self, X, Y, epochs): for _
        in range(epochs):

            for i in range(Y.shape[0]): x =
                X[i]

                y = self.predict(x) e = Y[i] - y self.W = self.W

                + self.lr * e * np.insert(x, 0, 1)

X = np.array([

    [0,0,0,0,0,0,1,0,0,0], # 0

    [0,0,0,0,0,0,0,1,0,0], # 1

    [0,0,0,0,0,0,0,0,1,0], # 2

    [0,0,0,0,0,0,0,0,0,1], # 3

    [0,0,0,0,0,0,1,1,0,0], # 4
```

```
    [0,0,0,0,0,0,1,0,1,0], # 5

    [0,0,0,0,0,0,1,1,1,0], # 6

    [0,0,0,0,0,0,1,1,1,1], # 7

    [0,0,0,0,0,0,1,0,1,1], # 8

    [0,0,0,0,0,0,0,1,1,1], # 9


])


Y = np.array([0, 1, 0, 1, 0, 1, 0, 1, 0, 1])


# Create the perceptron and train it perceptron
= Perceptron(input_size=10) perceptron.train(X,
Y, epochs=100)


# Test the perceptron on some input data test_X
= np.array([

    [0,0,0,0,0,0,1,0,0,0], # 0

    [0,0,0,0,0,0,0,1,0,0], # 1

    [0,0,0,0,0,0,0,0,1,0], # 2

    [0,0,0,0,0,0,0,0,0,1], # 3

    [0,0,0,0,0,0,1,1,0,0], # 4

    [0,0,0,0,0,0,1,0,1,0], # 5

    [0,0,0,0,0,0,1,1,1,0], # 6

    [0,0,0,0,0,0,1,1,1,1], # 7

    [0,0,0,0,0,0,1,0,1,1], # 8

    [0,0,0,0,0,0,0,1,1,1], # 9

])
```

for i in range(test_X.shape[0]): x = test_X[i]

y = perceptron.predict(x) print(f'{x} is

{"even" if y == 0 else "odd"}')

Output:

[0 0 0 0 0 0 1 0 0 0] is even [0

0 0 0 0 0 0 1 0 0] is odd

[0 0 0 0 0 0 0 0 1 0] is even [0

0 0 0 0 0 0 0 0 1] is odd

[0 0 0 0 0 0 1 1 0 0] is even

[0 0 0 0 0 0 1 0 1 0] is even

[0 0 0 0 0 0 1 1 1 0] is even

[0 0 0 0 0 0 1 1 1 1] is even

[0 0 0 0 0 0 1 0 1 1] is even

[0 0 0 0 0 0 0 1 1 1] is odd

Assignment No 04

Name- Thorve Avishkar Shrikrushna

Roll No- 63

Title- Demonstrate the Perceptron learning law with its decision regions.

Program:

```
import numpy as np import

matplotlib.pyplot as plt from

sklearn.datasets import load_iris iris

= load_iris()

X = iris.data[:, [0, 2]] y =

iris.target  y = np.where(y == 0,

0, 1)  w = np.zeros(2) b = 0 lr =

0.1 epochs = 50  def

perceptron(x, w, b): z =

np.dot(x, w) + b return

np.where(z >= 0, 1, 0)

for epoch in range(epochs): for i
    in range(len(X)):

        x = X[i] target
        = y[i]

        output = perceptron(x, w, b) error
        = target - output w

        += lr * error * x
```

b += lr * error x_min, x_max = X[:, 0].min() - 0.5, X[:,

0].max() + 0.5

y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5 xx,

yy = np.meshgrid(np.arange(x_min, x_max, 0.02),

np.arange(y_min, y_max, 0.02))

Z = perceptron(np.c_[xx.ravel(), yy.ravel()], w, b) Z

= Z.reshape(xx.shape) plt.contourf(xx, yy, Z,

cmap=plt.cm.Paired) plt.scatter(X[:, 0], X[:, 1], c=y,

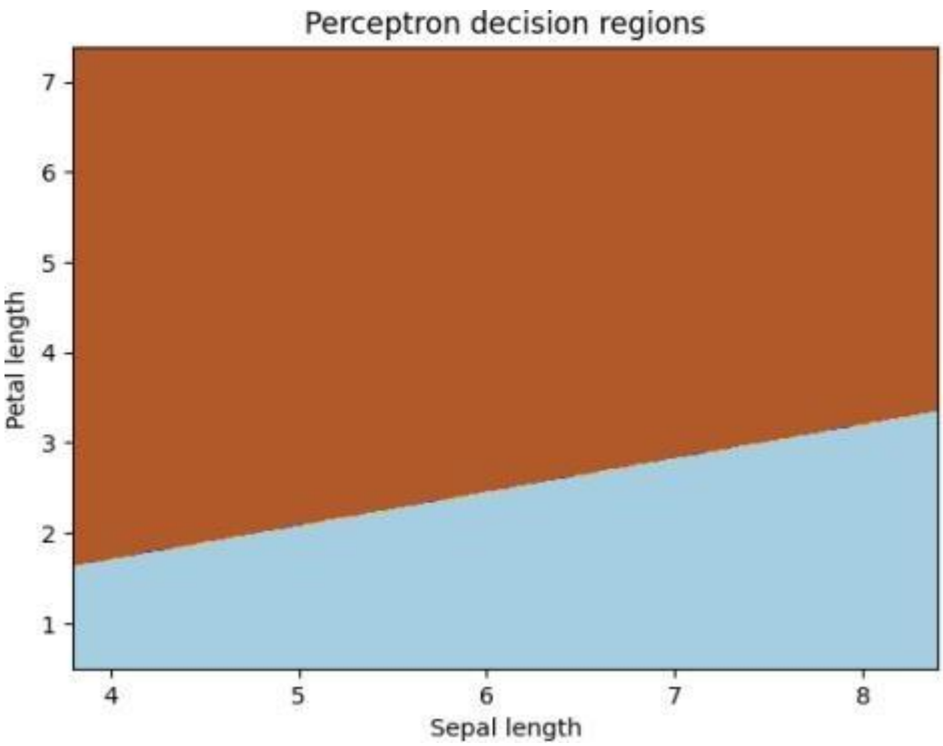
cmap=plt.cm.Paired) plt.xlabel('Sepal length')

plt.ylabel('Petal length')

plt.title('Perceptron decision regions')

plt.show()

Output:



Assignment No 05

Name- Thorve Avishkar Shrikrushna

Roll No-63

Title-Bidirectional Associative Memory with two pairs of vectors.

Program:

```
import numpy as np x1=np.array([1,1,1,-
1]) y1=np.array([1,-1]) x2=np.array([-1,-
1,1,1]) y2=np.array([-1,1])
W=np.outer(y1,x1)+np.outer(y2,x2)
def bam(x):     y=np.dot(W,x)
y=np.where(y>=0,1,-1)     return y
x_test=np.array([1,-1,-1,-1])
y_test=bam(x_test) print("Input
x:",x_test)
print("Output:",y_test)
```

Output:

```
Input x: [ 1 -1 -1 -1]
Output: [ 1 -1]
```

Assignment No 06

Name- Thorve Avishkar Shrikrushna

Roll No-63

Title- Artificial Neural Network trining process of Forward Propagation, Back Propagation.

Program:

```
import numpy as np class NeuralNetwork:     def
_init_(self, input_size, hidden_size, output_size):
self.W1 = np.random.randn(input_size, hidden_size)
self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size)
        self.b2 = np.zeros((1, output_size))     def
sigmoid(self, x):         return 1 / (1 + np.exp(-
x))     def sigmoid_derivative(self, x):
return x * (1 - x)     def
forward_propagation(self, X):         self.z1 =
np.dot(X, self.W1) + self.b1         self.a1 =
self.sigmoid(self.z1)         self.z2 =
np.dot(self.a1, self.W2) + self.b2         y_hat =
self.sigmoid(self.z2)
        return y_hat     def
backward_propagation(self, X, y, y_hat):
        self.error = y - y_hat
        self.delta2 = self.error * self.sigmoid_derivative(y_hat)
self.a1_error = self.delta2.dot(self.W2.T)
        self.delta1 = self.a1_error * self.sigmoid_derivative(self.a1 self.W2 += self.a1.T.dot(self.delta2)
self.b2 += np.sum(self.delta2, axis=0, keepdims=True)         self.W1 += X.T.dot(self.delta1)
self.b1 += np.sum(self.delta1, axis=0) def train(self, X, y, epochs, learning_rate=0.1)   for i in
range(epochs):
            y_hat   =   self.forward_propagation(X)
self.backward_propagation(X, y, y_hat)          if
i % 1000 == 0:
                print("Error at epoch", i, ":", np.mean(np.abs(self.error)))
def predict(self, X):
        return self.forward_propagation(X) X
= np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) y
= np.array([[0], [1], [1], [0]])
nn = NeuralNetwork(input_size=2, hidden_size=4, output_size=1)
nn.train(X, y, epochs=10000) predictions = nn.predict(X)
print("\nPredictions:\n", predictions)
```

Output:

```
[[5.55111512e-16]
[6.66666667e-63]
[6.66666667e-63]
[6.66666667e-63]]
```

Assignment No 07

Name- Thorve Avishkar Shrikrushna

Roll No- 63

Title: Python program to show back propagationnetwork for XOR function with Binary Input and Output.

Program:

```
import numpy as np class
XORNetwork:     def __init__(self):
self.W1 = np.random.randn(2, 2)
self.b1 = np.random.randn(2)
self.W2 = np.random.randn(2, 1)
self.b2 = np.random.randn(1)     def
sigmoid(self, x):         return 1 / (1 +
np.exp(-x))     def
sigmoid_derivative(self, x):
       return x * (1 - x)     def forward(self, X):
self.z1 = np.dot(X, self.W1) + self.b1
self.a1 = self.sigmoid(self.z1)         self.z2 =
np.dot(self.a1, self.W2) + self.b2         self.a2 =
self.sigmoid(self.z2)         return self.a2     def
backward(self, X, y, output):
self.output_error = y - output
       self.output_delta = self.output_error * self.sigmoid_derivative(output)
self.z1_error = self.output_delta.dot(self.W2.T)
       self.z1_delta = self.z1_error * self.sigmoid_derivative(self.a1)
self.W1 += X.T.dot(self.z1_delta)         self.b1 +=
np.sum(self.z1_delta, axis=0)         self.W2 +=
self.a1.T.dot(self.output_delta)         self.b2 +=
np.sum(self.output_delta, axis=0)     def train(self, X, y, epochs):
for _ in range(epochs):
          output = self.forward(X)
self.backward(X, y, output)     def
predict(self, X):         return
self.forward(X) xor_nn =
XORNetwork()
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) y
= np.array([[0], [1], [1], [0]])
xor_nn.train(X, y, epochs=10000)
predictions = xor_nn.predict(X)
print(predictions)
```

Output:

```
[[0.63075848]
 [0.98777524]
 [0.9877705 ]
 [0.63148649]]
```

Assignment No 08

Name- Thorve Avishkar Shrikrushna

Roll No-63

Title- Program for creating a back propagation feed-forward neural network.

Program:

```
import numpy as np def
sigmoid(x):
    return 1 / (1 + np.exp(-x)) def
sigmoid_derivative(x):
    return x * (1 - x)

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) y
= np.array([[0], [1], [1], [0]])

learning_rate = 0.1
num_epochs = 100000

hidden_weights = 2 * np.random.random((2, 2)) - 1
output_weights = 2 * np.random.random((2, 1)) - 1

for _ in range(num_epochs):
    hidden_layer = sigmoid(np.dot(X, hidden_weights))
    output_layer = sigmoid(np.dot(hidden_layer, output_weights))

    output_error = y - output_layer
    output_delta = output_error * sigmoid_derivative(output_layer)

    hidden_error = output_delta.dot(output_weights.T)
    hidden_delta = hidden_error * sigmoid_derivative(hidden_layer)


    output_weights += hidden_layer.T.dot(output_delta) * learning_rate
hidden_weights += X.T.dot(hidden_delta) * learning_rate

print("Input:") print(X)
print("Output:")
print(output_layer)
```

Output:

Input:

[[00]

[0 1]

[1 0]

[1 1]]

Output:

[[0 61385986]

[0.63944088]

[0.8569871] [0.11295854]]