1. McCulloch-Pitts neuron XOR gate.

PROGRAM:

```
x1=[0,0,1,1]

x2=[0,1,0,1]

z=[0,1,1,0]

con=True

while con:

    w11=float(input("weight w11="))

    w12=float(input("weight w12="))

    w21=float(input("weight w21="))

    w22=float(input("weight w22="))

    v1=float(input("weight v1="))

    v2=float(input("weight v2="))

    theta=float(input("theta="))

    zin1=[x1[i]*w11+x2[i]*w21 for i in range(4)]

    zin2=[x1[i]*w21+x2[i]*w22 for i in range(4)]

    y1=[1 if zin1[i]>=theta else 0 for i in range(4)]

    y2=[1 if zin2[i]>=theta else 0 for i in range(4)]

    yin=[y1[i]*v1+y2[i]*v2 for i in range(4)]

    y=[1 if yin[i]>=theta else 0 for i in range(4)]

    print("output of net=")
```

```
        print(y)
        if y==z:
            con=False
        else:
            print("net is not learning. enter another set of weights and threshold value")
print("mcculloch pitts neuron for xor function")
print("weights of neuron z1:")
print(w11)
print(w21)
print("weights of neuron z2:")
print(w12)
print(w22)
print("weights of neuron y:")
print(v1)
print(v2)
print("threshold value:")
print(theta)
```

OUTPUT:

weight w11=1

weight w12=-1

weight w21=-1

weight w22=1

weight v1=1

weight v2=1

theta=1

output of net=

[0, 1, 1, 0]

mcculloch pitts neuron for xor function

weights of neuron z1:

1.0

-1.0

weights of neuron z2:

-1.0

1.0

weights of neuron y:

1.0

1.0

threshold value:

1.0

2. perceptron for an AND function with bipolar inputs and targets

PROGRAM:

```
def mcculloch_pitts_neuron():
    y=[0,0,0,0]
    x1=[1,1,0,0]
    x2=[1,0,1,0]
    z=[1,0,0,0]
    con=True
    print('enter the weights')
    w1=float(input('weight w1='))
    w2=float(input('weight w2='))
    theta=float(input('enter the threshold value theta='))
    while con:
        zin=[x1[i]*w1+x2[i]*w2 for i in range(4)]
        for i in range(4):
            if zin[i]>=theta:
                y[i]=1
            else:
                y[i]=0
                print('output of net=')
                print(y)
```

```
                if y==z:
                    con=False
                else:
                    print('net is not learning. enter another ser of weights and threshold value')
                    w1=float(input('weight w1='))
                    w2=float(input('weight w2='))
                    theta=float(input('enter threshold value theta'))
        print('mcculloch pitts neuron for and function')
        print('weights of neuron')
        print(w1)
        print(w2)
        print('threshold value=')
        print(theta)
mcculloch_pitts_neuron()
```

Output:

enter the weights

weight w1=1

weight w2=1

enter the threshold value theta=2

output of net=

[1, 0, 0, 0]

output of net=

[1, 0, 0, 0]

output of net=

[1, 0, 0, 0]

mcculloch pitts neuron for and function

weights of neuron

1.0

1.0

threshold value=

2.0

3. XOR function with binary input and output

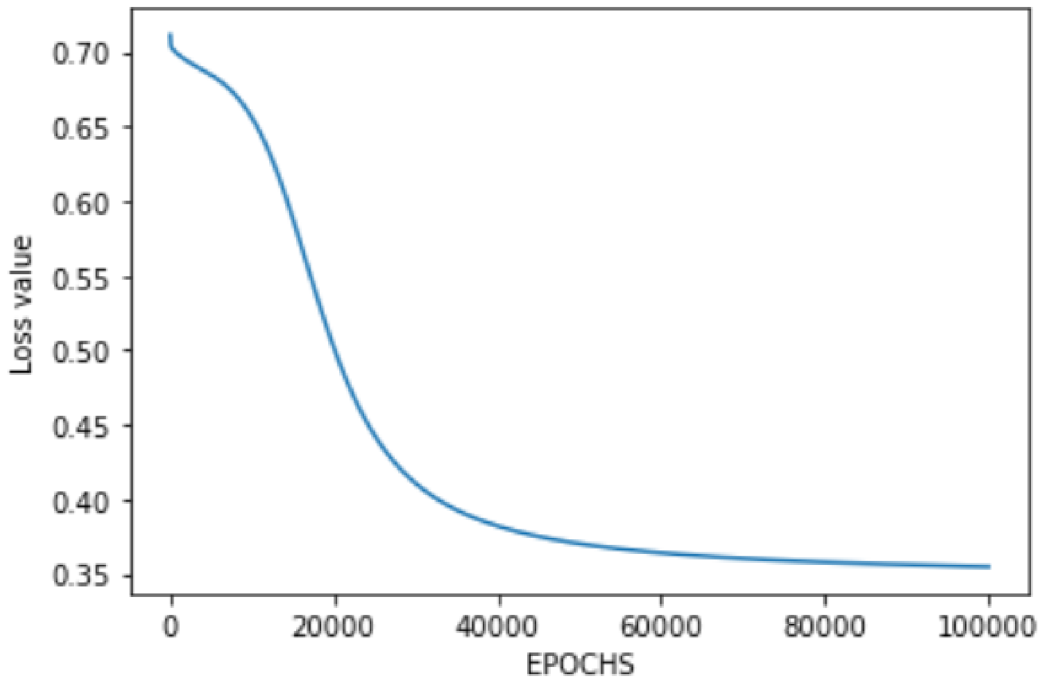
PROGRAM:

```
import numpy as np
from matplotlib import pyplot as plt
def sigmoid(z):
    return 1/(1+np.exp(-z))
def initializeParameters(inputFeatures,neuronsInHiddenLayers,outputFeatures):
    W1=np.random.randn(neuronsInHiddenLayers,inputFeatures)
    W2=np.random.randn(outputFeatures,neuronsInHiddenLayers)
    b1=np.zeros((neuronsInHiddenLayers,1))
    b2=np.zeros((outputFeatures,1))
    parameters={"W1":W1,"b1":b1,"W2":W2,"b2":b2}
    return parameters
def forwardPropagation(x,y,parameters):
    m=x.shape[1]
    W1=parameters["W1"]
    W2=parameters["W2"]
    b1=parameters["b1"]
    b2=parameters["b2"]
    Z1=np.dot(W1,x)+b1
    A1=sigmoid(Z1)
    Z2=np.dot(W2,A1)+b2
```

```
    A2=sigmoid(Z2)
    cache=(Z1,A1,W1,b1,Z2,A2,W2,b2)
    logprobs=np.multiply(np.log(A2),y)+np.multiply(np.log(1-A2),(1-Y))
    cost=-np.sum(logprobs)/m
    return cost,cache,A2
def backwardPropagation(X,Y,cache):
    m=X.shape[1]
    (Z1,A1,W1,b1,Z2,A2,W2,b2)=cache
    dz2=A2-Y
    dW2=np.dot(dz2,A1.T)/m
    db2=np.sum(dz2,axis=1,keepdims=True)
    dA1=np.dot(W2.T,dz2)
    dZ1=np.multiply(dA1,A1*(1-A1))
    dW1=np.dot(dZ1,X.T)/m
    db1=np.sum(dZ1,axis=1,keepdims=True)/m
gradients={"dz2":dz2,"dW2":dW2,"db2":db2,"dZ1":dZ1,"dW1":dW1,"db1":db1}
    return gradients
def updateParameters(parameters,gradients,learningRate):
    parameters["W1"]=parameters["W1"]-learningRate*gradients["dW1"]
    parameters["W2"]=parameters["W2"]-learningRate*gradients["dW2"]
    parameters["b1"]=parameters["b1"]-learningRate*gradients["db1"]
    parameters["b2"]=parameters["b2"]-learningRate*gradients["db2"]
    return parameters
```

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

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

neuronsInHiddenLayers=2

inputFeatures=X.shape[0]

outputFeatures=Y.shape[0]

parameters=initializeParameters(inputFeatures,neuronsInHiddenLayers,outputFeatures)

epoch=100000

learningRate=0.01

losses=np.zeros((epoch,1))

for i in range(epoch):

    losses[i,0],cache,A2=forwardPropagation(X,Y,parameters)

    gradients=backwardPropagation(X,Y,cache)

    parameters=updateParameters(parameters,gradients,learningRate)

plt.figure()

plt.plot(losses)

plt.xlabel("EPOCHS")

plt.ylabel("Loss value")

plt.show()

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

cost,_,A2=forwardPropagation(X,Y,parameters)

prediction=(A2>0.5)*1.0

print(prediction)
```

OUTPUT:


0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
Loss value
0
20000
40000
60000
80000
100000
EPOCHS

```
[[0. 1. 0. 1.]]
```

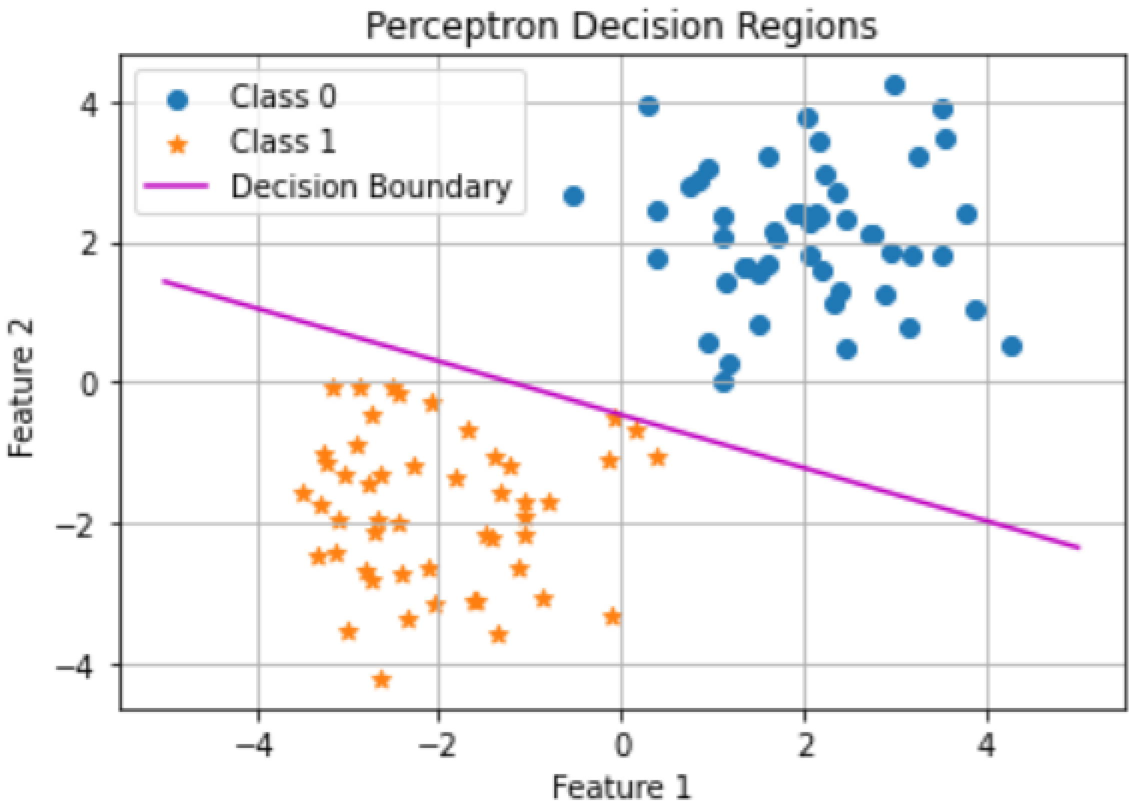
4) Perceptron learning law

PROGRAM:

```
import numpy as np

import matplotlib.pyplot as plt

np.random.seed(0)

class_0=np.random.randn(50,2)+np.array([2,2])

class_1=np.random.randn(50,2)+np.array([-2,-2])

X=np.vstack((class_0,class_1))

y=np.hstack((np.zeros(50),np.ones(50)))

X_augmented=np.c_[X,np.ones(100)]

np.random.seed(1)

w = np.random.randn(3)

def perceptron_learning(X,y,w,epochs=50,learning_rate=0.1):

    errors=[]

    for _ in range(epochs):

        error_count=0

        for xi,target in zip(X,y):

            output=np.dot(xi,w)>=0

            error=target - output

            if error !=0:

                w+=learning_rate*error*xi
```

```
                error_count+=1
        errors.append(error_count)
    return w,errors
trained_weights,errors= perceptron_learning(X_augmented,y,w)
plt.scatter(class_0[:,0],class_0[:,1],marker='p',label='class 0')
plt.scatter(class_1[:,0],class_1[:,1],marker='*',label='class 1')
x_boundary=np.linspace(-5,5,100)
y_boundary=(-trained_weights[0]*
x_boundary-trained_weights[2])/trained_weights[1]
plt.plot(x_boundary,y_boundary,'p-',label='Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid()
plt.title('Perceptron Decision Regions')
plt.show()
```

OUTPUT:



Perceptron Decision Regions
Class 0
Class 1
Decision Boundary
Feature 2
Feature 1
4
2
0
−2
−4
−4
−2
0
2
4

5) XOR function with Bipolar input and output

PROGRAM:

```
import numpy as np
def sigmoid(x):
    return 2/(1+np.exp(-x))-1
def sigmoid_derivative(x):
    return 0.5*(1+x)*(1-x)
x = np.array([[-1,-1],[-1,1],[1,-1],[1,1]])
t = np.array([[-1],[1],[1],[-1]])
input_size = 2
hidden_size = 2
output_size = 1
np.random.seed(0)
w_input_hidden = np.random.rand(input_size,hidden_size)
b_hidden = np.random.rand(1,hidden_size)
w_hidden_output = np.random.rand(hidden_size,output_size)
b_output = np.random.rand(1,output_size)
learning_rate = 0.1
epochs = 10000
for epoch in range(epochs):
    hidden_input = np.dot(x,w_input_hidden)+b_hidden
    hidden_output = sigmoid(hidden_input)
```

```
    output = np.dot(hidden_output,w_hidden_output)+b_output
    output = sigmoid(output)
    output_error = t-output
    hidden_error =
output_error.dot(w_hidden_output.T)*sigmoid_derivative(hidden_output)
    w_hidden_output+=hidden_output.T.dot(output_error)*learning_rate
    b_output+=np.sum(output_error)*learning_rate
    w_input_hidden+=x.T.dot(hidden_error)*learning_rate
    b_hidden+=np.sum(hidden_error)*learning_rate
    if epoch % 1000==0:
        print(f"Epoch:{epoch},Error:{np.mean(np.abs(output_error))}")


print("\nTesting the trained network:")
for i in range(3):
    hidden_input = np.dot(x[i],w_input_hidden)+b_hidden
    hidden_output = sigmoid(hidden_input)
    output = np.dot(hidden_output,w_hidden_output)+b_output
    output = sigmoid(output)
    print(f"input:{x[i]},Predicted Output:{output[0][0]},Target Output:{t[i][0]}")
```

OUTPUT:

Epoch:0,Error:0.9612435747420139
Epoch:1000,Error:0.5040833500390071
Epoch:2000,Error:0.5017794479227115
Epoch:3000,Error:0.5011257955648463
Epoch:4000,Error:0.50081982834948
Epoch:5000,Error:0.5006431517612333
Epoch:6000,Error:0.5005283743537952
Epoch:7000,Error:0.500447932412263
Epoch:8000,Error:0.5003884828374154
Epoch:9000,Error:0.5003427898178361

Testing the trained network:
input:[-1 -1],Predicted Output:-0.9996290483699555,Target Output:-1
input:[-1 1],Predicted Output:-0.00022888265647258432,Target Output:1
input:[1 -1],Predicted Output:0.999615535881984,Target Outp

6) Write a program to load an image and apply the Convolution layer, Activation layer and polling layer Operation to extract the inside feature

PROGRAM:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from itertools import product

# set the param
plt.rc('figure', autolayout=True)
plt.rc('image', cmap='magma')

kernel = tf.constant([[-1, -1, -1],
                      [-1, 8, -1],
                      [-1, -1, -1],
                     ])

#Load the image

image_path =r"C:\Users\STUDENT.HCAS-PGLAB-4\Pictures\Saved Pictures\ga
nesh.jpg"
image = tf.io.read_file(image_path)
image = tf.io.decode_jpeg(image, channels=1)
image = tf.image.resize(image, size=[300, 300])

#Load the image

image_path =r"C:\Users\STUDENT.HCAS-PGLAB-4\Pictures\Saved Pictures\ga
nesh.jpg"
image = tf.io.read_file(image_path)
image = tf.io.decode_jpeg(image, channels=1)
image = tf.image.resize(image, size=[300, 300])

#plot the image
```

```
img = tf.squeeze(image).numpy()
plt.figure(figsize=(5, 5))
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.title('Original Gray Scale image')
plt.show();

#Reformat

image = tf.image.convert_image_dtype(image, dtype=tf.float32)
image = tf.expand_dims (image, axis=0)
kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel = tf.cast (kernel, dtype=tf.float32)

conv_fn = tf.nn.conv2d#Reformat

image = tf.image.convert_image_dtype(image, dtype=tf.float32)
image = tf.expand_dims (image, axis=0)
kernel = tf.reshape(kernel, [*kernel.shape, 1, 1])
kernel = tf.cast (kernel, dtype=tf.float32)

image_filter = conv_fn(
    input = image,
    filters = kernel,
    strides = 1,
    padding ='SAME',
)
plt.figure(figsize=(15,5))

# Plot the convolved image
plt.subplot(1, 1, 1)

plt.imshow(
    tf.squeeze(image_filter)
)

plt.axis('off')
plt.title('Convolution')
```

```
#activation layer
tanh_fn = tf.nn.tanh


# Image detection
image_detect = relu_fn(image_filter)

plt.subplot(1, 1, 1)
plt.imshow(
    # Reformat for plotting
    tf.squeeze(image_detect)
)

plt.axis('off')
plt.title('Activation')

pool = tf.nn.pool

image_condense = pool(input=image_detect,
                      window_shape=(2,2),
                      pooling_type='MAX',
                      strides=(2,2),
                      padding='SAME',
                      )

plt.subplot(1,1,1)
plt.imshow(tf.squeeze(image_condense))
plt.axis('off')
plt.title('pooling')
plt.show()
```

OUTPUT:

Original Gray Scale image

Convolution

Activation

pooling