Name: Yash Gandhi BE IT Batch A 2017140014

Aim:
To implement basic logic gates using a single neuron network

Problem Statement:
Implement AND, OR, NOT Gates using a single perceptron with the following activation functions:

a) Unipolar Binary Activation functions
b) Bipolar Binary Activation functions

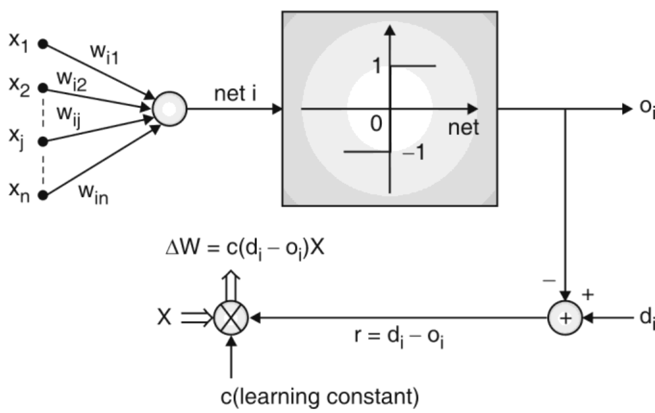
Hint:

Construct the truth table for these gates.

Assume the entries in the truth table as examples for your training.

Theoretical Aspects:

Perceptron Model



Algorithm for Training a Single Perceptron Network

- A single perceptron can solve a classification problem with 'n' input features and two output classes (0/1).

SDPTA (Single Discrete Perceptron Training Algorithm)

- Given are P training pairs
  $\{X_1, d_1, X_2, d_2, X_3, d_3, ..., X_p, d_p\}$ where X_i is $(n \times 1)$, d_i is (1×1), i = 1, 2, ... P
- Note that augmented input vectors are used.

$$Y_i = \begin{bmatrix} X_i \\ 1 \end{bmatrix} \text{ for } i = 1, 2, \ldots, P$$

i.e. there is an additional input for the bias and it is 1.

In the following, *k* denotes the training steps and *p* denotes the step counter within a cycle.

Step 1 : $c > 0$ is chosen

Step 2 : Weights are initialized as *W* at small random values.

W is $(n + 1) \times 1$

Counters are initialized

$k \leftarrow 1, p \leftarrow 1, \quad E \leftarrow 0$

Step 3 : The training cycle begins here. Input is presented and output is computed

$Y \leftarrow Y_p, \qquad d \leftarrow d_p$

$o \leftarrow \mathrm{sgn}\,(W^t\, Y)$

Step 4 : Weights are updated

$W \leftarrow W + \frac{1}{2}\, c\,(d - o)\, Y$

Step 5 : Cycle error is computed

$E \leftarrow \frac{1}{2}\,(d - o)^2 + E$

Step 6 : If $p < P$ then

$p \leftarrow p + 1 \qquad\qquad k \leftarrow k + 1$

and go to step 3 ; otherwise go to step 7

Step 7 : The training cycle is completed.

For $E = 0$, terminate the training session. Display weights and *k*.

If $E > 0$, then $E \leftarrow 0$, $p \leftarrow 1$

and enter the new training cycle by going to step 3

Weight Update Equations:

$W \leftarrow W + c*(d - o)*X$

C=1

Cycle error is computed as:

$E = \frac{1}{2}\,(d - o)^2 + E$

Tool/Language:

C

Code:

Activation function: Unipolar Binary function

For And Gate:

```c
#include <stdio.h>

int activation(int O)
{
    if(O>0)
    O=1;
    else
    O=0;

    return O;
}


int main()
{
    int a[4][4] = {
        {0,0,1,0},
        {0,1,1,0},
        {1,0,1,0},
        {1,1,1,1}
    };
    float w[1][3]={
        {0,0,0}
    };

    int c= 1;
    float e=1;
    int O=0;
    int count=0;
    while(e!=0)
    {
        e=0;
        count=count +1;
        printf("Epoch = %d\n",count);
      for(int i=0;i<4;i++)
      {

          O=a[i][0]*w[0][0]+a[i][1]*w[0][1]+a[i][2]*w[0][2];
          O=activation(O);
          w[0][0]=w[0][0]+c*(a[i][3]-O)*a[i][0];
          w[0][1]=w[0][1]+c*(a[i][3]-O)*a[i][1];
          w[0][2]=w[0][2]+c*(a[i][3]-O)*a[i][2];
          float k=(a[i][3]-O)*(a[i][3]-O);
          e=e+k/2;
          printf("w1= %.2f , w2= %.2f ,w3= %.2f ,Output = %d,Error=%.2f \n",w[0][0],w[0][1],w[0][2],O,e);
```

```
        }
    }

printf("\nTesting with input 1 , 1\n");
int output=w[0][0]*1+w[0][1]*1+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 0 , 1\n");
output=w[0][0]*0+w[0][1]*1+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 1 , 0\n");
output=w[0][0]*1+w[0][1]*0+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 0 , 0\n");
output=w[0][0]*0+w[0][1]*0+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);
    return 0;
}
```

Results: For each step: weight vector,output and error :

```
Epoch = 1
w1= 0.00 , w2= 0.00 ,w3= 0.00 ,Output = 0,Error=0.00
w1= 0.00 , w2= 0.00 ,w3= 0.00 ,Output = 0,Error=0.00
w1= 0.00 , w2= 0.00 ,w3= 0.00 ,Output = 0,Error=0.00
w1= 1.00 , w2= 1.00 ,w3= 1.00 ,Output = 0,Error=0.50
Epoch = 2
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.50
w1= 1.00 , w2= 0.00 ,w3= -1.00 ,Output = 1,Error=1.00
w1= 1.00 , w2= 0.00 ,w3= -1.00 ,Output = 0,Error=1.00
w1= 2.00 , w2= 1.00 ,w3= 0.00 ,Output = 0,Error=1.50
Epoch = 3
w1= 2.00 , w2= 1.00 ,w3= 0.00 ,Output = 0,Error=0.00
w1= 2.00 , w2= 0.00 ,w3= -1.00 ,Output = 1,Error=0.50
w1= 1.00 , w2= 0.00 ,w3= -2.00 ,Output = 1,Error=1.00
w1= 2.00 , w2= 1.00 ,w3= -1.00 ,Output = 0,Error=1.50
Epoch = 4
w1= 2.00 , w2= 1.00 ,w3= -1.00 ,Output = 0,Error=0.00
w1= 2.00 , w2= 1.00 ,w3= -1.00 ,Output = 0,Error=0.00
w1= 1.00 , w2= 1.00 ,w3= -2.00 ,Output = 1,Error=0.50
w1= 2.00 , w2= 2.00 ,w3= -1.00 ,Output = 0,Error=1.00
Epoch = 5
w1= 2.00 , w2= 2.00 ,w3= -1.00 ,Output = 0,Error=0.00
w1= 2.00 , w2= 1.00 ,w3= -2.00 ,Output = 1,Error=0.50
w1= 2.00 , w2= 1.00 ,w3= -2.00 ,Output = 0,Error=0.50
w1= 2.00 , w2= 1.00 ,w3= -2.00 ,Output = 1,Error=0.50
Epoch = 6
w1= 2.00 , w2= 1.00 ,w3= -2.00 ,Output = 0,Error=0.00
w1= 2.00 , w2= 1.00 ,w3= -2.00 ,Output = 0,Error=0.00
w1= 2.00 , w2= 1.00 ,w3= -2.00 ,Output = 0,Error=0.00
w1= 2.00 , w2= 1.00 ,w3= -2.00 ,Output = 1,Error=0.00

Testing with input 1 , 1
Output = 1

Testing with input 0 , 1
Output = 0

Testing with input 1 , 0
Output = 0

Testing with input 0 , 0
Output = 0
```

Activation function: Unipolar Binary function
For OR Gate:

```c
#include <stdio.h>

int activation(int O)
{
    if(O>0)
    O=1;
    else
    O=0;

    return O;
}


int main()
{
    int a[4][4] = {
        {0,0,1,0},
        {0,1,1,1},
        {1,0,1,1},
        {1,1,1,1}
    };
    float w[1][3]={
        {0,0,0}
    };

    int c= 1;
    float e=1;
    int O=0;
    int count=0;
    while(e!=0)
    {
        e=0;
        count=count +1;
        printf("Epoch = %d\n",count);
      for(int i=0;i<4;i++)
      {

          O=a[i][0]*w[0][0]+a[i][1]*w[0][1]+a[i][2]*w[0][2];
          O=activation(O);
          w[0][0]=w[0][0]+c*(a[i][3]-O)*a[i][0];
          w[0][1]=w[0][1]+c*(a[i][3]-O)*a[i][1];
          w[0][2]=w[0][2]+c*(a[i][3]-O)*a[i][2];
          float k=(a[i][3]-O)*(a[i][3]-O);
          e=e+k/2;
          printf("w1= %.2f , w2= %.2f ,w3= %.2f ,Output = %d,Error=%.2f \n",w[0][0],w[0][1],w[0][2],O,e);
```

```
        }
    }

printf("\nTesting with input 1 , 1\n");
int output=w[0][0]*1+w[0][1]*1+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 0 , 1\n");
output=w[0][0]*0+w[0][1]*1+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 1 , 0\n");
output=w[0][0]*1+w[0][1]*0+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 0 , 0\n");
output=w[0][0]*0+w[0][1]*0+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);
    return 0;
}
```

Results: For each step: weight vector,output and error :

```
Epoch = 1
w1= 0.00 , w2= 0.00 ,w3= 0.00 ,Output = 0,Error=0.00
w1= 0.00 , w2= 1.00 ,w3= 1.00 ,Output = 0,Error=0.50
w1= 0.00 , w2= 1.00 ,w3= 1.00 ,Output = 1,Error=0.50
w1= 0.00 , w2= 1.00 ,w3= 1.00 ,Output = 1,Error=0.50
Epoch = 2
w1= 0.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.50
w1= 0.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.50
w1= 1.00 , w2= 1.00 ,w3= 1.00 ,Output = 0,Error=1.00
w1= 1.00 , w2= 1.00 ,w3= 1.00 ,Output = 1,Error=1.00
Epoch = 3
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.50
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.50
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.50
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.50
Epoch = 4
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 0,Error=0.00
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.00
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.00
w1= 1.00 , w2= 1.00 ,w3= 0.00 ,Output = 1,Error=0.00

Testing with input 1 , 1
Output = 1

Testing with input 0 , 1
Output = 1

Testing with input 1 , 0
Output = 1

Testing with input 0 , 0
Output = 0
```

Activation function: Unipolar Binary function
For NOT Gate:

```
#include <stdio.h>

int activation(int O)
{
    if(O>0)
    O=1;
    else
    O=0;

    return O;
}


int main()
{
    int a[2][3] = {
        {0,1,1},
        {1,1,0}
    };
    float w[1][2]={
        {0,0}
    };

    int c= 1;
    float e=1;
    int O=0;
    int count=0;
    while(e!=0)
    {
        e=0;
        count=count +1;
        printf("Epoch = %d\n",count);
      for(int i=0;i<2;i++)
      {

          O=a[i][0]*w[0][0]+a[i][1]*w[0][1];
          O=activation(O);
          w[0][0]=w[0][0]+c*(a[i][3]-O)*a[i][0];
          w[0][1]=w[0][1]+c*(a[i][3]-O)*a[i][1];

          float k=(a[i][2]-O)*(a[i][2]-O);
          e=e+k/2;
          printf("w1= %.2f , w2= %.2f ,Output = %d,Error=%.2f \n",w[0][0],w[0][1],O,e);

      }
    }

printf("\nTesting with input 1 \n");
int output=w[0][0]*1+w[0][1]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 0 \n");
output=w[0][0]*0+w[0][1]*1;
output=activation(output);
printf("Output = %d\n",output);

}
```

Results: For each step: weight vector,output and error :

```
Epoch = 1
w1= 0.00 , w2= 1.00 ,Output = 0,Error=0.50
w1= -1.00 , w2= 0.00 ,Output = 1,Error=1.00
Epoch = 2
w1= -1.00 , w2= 1.00 ,Output = 0,Error=0.50
w1= -1.00 , w2= 1.00 ,Output = 0,Error=0.50
Epoch = 3
w1= -1.00 , w2= 1.00 ,Output = 1,Error=0.00
w1= -1.00 , w2= 1.00 ,Output = 0,Error=0.00

Testing with input 1
Output = 0

Testing with input 0
Output = 1
```

Activation function: Bipolar Binary function

For AND Gate:

```c
#include <stdio.h>

int activation(int O)
{
    if(O>0)
    O=1;
    else
    O=-1;

    return O;
}


int main()
{
    int a[4][4] = {
        {-1,-1,1,-1},
        {-1,1,1,-1},
        {1,-1,1,-1},
        {1,1,1,1}
    };
    float w[1][3]={
        {0,0,0}
    };

    int c= 1;
    float e=1;
    int O=0;
    int count=0;
    while(e!=0)
    {
        e=0;
        count=count +1;
        printf("Epoch = %d\n",count);
      for(int i=0;i<4;i++)
      {

          O=a[i][0]*w[0][0]+a[i][1]*w[0][1]+a[i][2]*w[0][2];
          O=activation(O);
          w[0][0]=w[0][0]+c*(a[i][3]-O)*a[i][0];
          w[0][1]=w[0][1]+c*(a[i][3]-O)*a[i][1];
          w[0][2]=w[0][2]+c*(a[i][3]-O)*a[i][2];
          float k=(a[i][3]-O)*(a[i][3]-O);
          e=e+k/2;
          printf("w1= %.2f , w2= %.2f ,w3= %.2f ,Output = %d,Error=%.2f \n",w[0][0],w[0][1],w[0][2],O,e);
```

```c
        }
    }

printf("\nTesting with input 1 , 1\n");
int output=w[0][0]*1+w[0][1]*1+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input -1 , 1\n");
output=w[0][0]*(-1)+w[0][1]*1+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 1 , -1\n");
output=w[0][0]*1+w[0][1]*(-1)+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input -1 , -1\n");
output=w[0][0]*(-1)+w[0][1]*(-1)+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);
    return 0;
}
```

Results: For each step: weight vector,output and error :

```
Epoch = 1
w1= 0.00 , w2= 0.00 ,w3= 0.00 ,Output = -1,Error=0.00
w1= 0.00 , w2= 0.00 ,w3= 0.00 ,Output = -1,Error=0.00
w1= 0.00 , w2= 0.00 ,w3= 0.00 ,Output = -1,Error=0.00
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = -1,Error=2.00
Epoch = 2
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = -1,Error=0.00
w1= 4.00 , w2= 0.00 ,w3= 0.00 ,Output = 1,Error=2.00
w1= 2.00 , w2= 2.00 ,w3= -2.00 ,Output = 1,Error=4.00
w1= 2.00 , w2= 2.00 ,w3= -2.00 ,Output = 1,Error=4.00
Epoch = 3
w1= 2.00 , w2= 2.00 ,w3= -2.00 ,Output = -1,Error=0.00
w1= 2.00 , w2= 2.00 ,w3= -2.00 ,Output = -1,Error=0.00
w1= 2.00 , w2= 2.00 ,w3= -2.00 ,Output = -1,Error=0.00
w1= 2.00 , w2= 2.00 ,w3= -2.00 ,Output = 1,Error=0.00

Testing with input 1 , 1
Output = 1

Testing with input -1 , 1
Output = -1

Testing with input 1 , -1
Output = -1

Testing with input -1 , -1
Output = -1
```

Activation function: Bipolar Binary function
For OR Gate:

```c
#include <stdio.h>

int activation(int O)
{
    if(O>0)
    O=1;
    else
    O=-1;

    return O;
}


int main()
{
    int a[4][4] = {
        {-1,-1,1,-1},
        {-1,1,1,1},
        {1,-1,1,1},
        {1,1,1,1}
    };
    float w[1][3]={
        {0,0,0}
    };

    int c= 1;
    float e=1;
    int O=0;
    int count=0;
    while(e!=0)
    {
        e=0;
        count=count +1;
        printf("Epoch = %d\n",count);
      for(int i=0;i<4;i++)
      {

          O=a[i][0]*w[0][0]+a[i][1]*w[0][1]+a[i][2]*w[0][2];
          O=activation(O);
          w[0][0]=w[0][0]+c*(a[i][3]-O)*a[i][0];
          w[0][1]=w[0][1]+c*(a[i][3]-O)*a[i][1];
          w[0][2]=w[0][2]+c*(a[i][3]-O)*a[i][2];
          float k=(a[i][3]-O)*(a[i][3]-O);
          e=e+k/2;
          printf("w1= %.2f , w2= %.2f ,w3= %.2f ,Output = %d,Error=%.2f \n",w[0][0],w[0][1],w[0][2],O,e);

      }
    }

printf("\nTesting with input 1 , 1\n");
int output=w[0][0]*1+w[0][1]*1+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input -1 , 1\n");
output=w[0][0]*(-1)+w[0][1]*1+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input 1 , -1\n");
output=w[0][0]*1+w[0][1]*(-1)+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input -1 , -1\n");
output=w[0][0]*(-1)+w[0][1]*(-1)+w[0][2]*1;
output=activation(output);
printf("Output = %d\n",output);
    return 0;
}
```

Results: For each step: weight vector,output and error :

```
Epoch = 1
w1= 0.00 , w2= 0.00 ,w3= 0.00 ,Output = -1,Error=0.00
w1= -2.00 , w2= 2.00 ,w3= 2.00 ,Output = -1,Error=2.00
w1= 0.00 , w2= 0.00 ,w3= 4.00 ,Output = -1,Error=4.00
w1= 0.00 , w2= 0.00 ,w3= 4.00 ,Output = 1,Error=4.00
Epoch = 2
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = 1,Error=2.00
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = 1,Error=2.00
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = 1,Error=2.00
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = 1,Error=2.00
Epoch = 3
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = -1,Error=0.00
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = 1,Error=0.00
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = 1,Error=0.00
w1= 2.00 , w2= 2.00 ,w3= 2.00 ,Output = 1,Error=0.00

Testing with input 1 , 1
Output = 1

Testing with input -1 , 1
Output = 1

Testing with input 1 , -1
Output = 1

Testing with input -1 , -1
Output = -1
```

Activation function: Bipolar Binary function
For NOT Gate:

```
#include<stdio.h>

int activation(int O)
{
    if(O>0)
    O=1;
    else
    O=-1;

    return O;
}


int main()
{
    int a[2][3] = {
        {-1,1,1},
        {1,1,-1}
    };
    float w[1][2]={
        {0,0}
    };

    int c= 1;
    float e=1;
    int O=0;
    int count=0;
    while(e!=0)
    {
        e=0;
        count=count +1;
        printf("Epoch = %d\n",count);
      for(int i=0;i<2;i++)
      {

          O=a[i][0]*w[0][0]+a[i][1]*w[0][1];
          O=activation(O);
          w[0][0]=w[0][0]+c*(a[i][2]-O)*a[i][0];
          w[0][1]=w[0][1]+c*(a[i][2]-O)*a[i][1];

          float k=(a[i][2]-O)*(a[i][2]-O);
          e=e+k/2;
          printf("w1= %.2f , w2= %.2f ,Output = %d,Error=%.2f \n",w[0][0],w[0][1],O,e);

      }
```

```c

printf("\nTesting with input 1 \n");
int output=w[0][0]*1+w[0][1]*1;
output=activation(output);
printf("Output = %d\n",output);

printf("\nTesting with input -1 \n");
output=-w[0][0]+w[0][1]*1;
output=activation(output);
printf("Output = %d\n",output);

}
```

Results: For each step: weight vector,output and error :

```
Epoch = 1
w1= -2.00 , w2= 2.00 ,Output = -1,Error=2.00
w1= -2.00 , w2= 2.00 ,Output = -1,Error=2.00
Epoch = 2
w1= -2.00 , w2= 2.00 ,Output = 1,Error=0.00
w1= -2.00 , w2= 2.00 ,Output = -1,Error=0.00

Testing with input 1
Output = -1

Testing with input -1
Output = 1


...Program finished with exit code 0
Press ENTER to exit console.
```

Conclusion:

The perceptron model is a more general computational model than McCulloch-Pitts neuron. It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0.A simple perceptron model was used to implement AND, OR and NOT Logic Gates. Unipolar and Bipolar Binary Activation function was used.

GATE	UNIPOLAR BINARY Epochs	BIPOLAR BINARY Epochs
AND	6	3
OR	4	3
NOT	3	2

It is evident from the table that Bipolar Binary function adjusts the weight to solve Basic Logic Gates in less number of epochs i.e. in less time than Unipolar Binary.