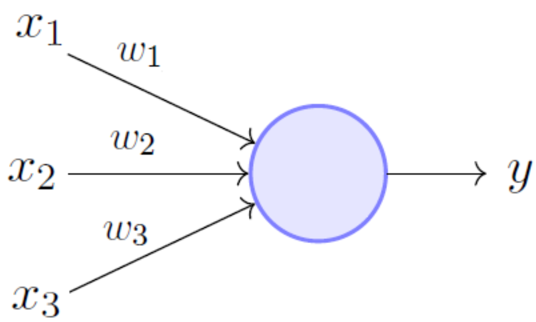
Assignment 1: Implementation of Perceptron from scratch

Rakesh K P

05 April 2022

1 Perceptron

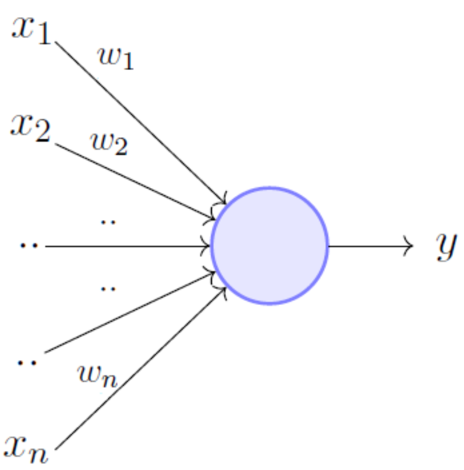
The perceptron model, proposed by Minsky-Papert, is a more general computational model than McCulloch-Pitts neuron. It overcomes some of the limitations of the M-P neuron by introducing the concept of numerical weights (a measure of importance) for inputs, and a mechanism for learning those weights. Inputs are no longer limited to boolean values like in the case of an M-P neuron, it supports real inputs as well which makes it more useful and generalized.



Perceptron Model (Minsky-Papert in 1969)

The perceptron model, proposed by Minsky-Papert, is a more general computational model than McCulloch-Pitts neuron. It overcomes some of the limitations of the M-P neuron by introducing the concept of numerical weights (a measure of importance) for inputs, and a mechanism for learning those weights. Inputs are no longer limited to boolean values like in the case

of an M-P neuron, it supports real inputs as well which makes it more useful and generalized.



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

A single perceptron can only be used to implement linearly separable functions. It takes both real and boolean inputs and associates a set of weights to them, along with a bias (the threshold thing I mentioned above). We learn the weights, we get the function. Let's use a perceptron to learn an OR function.

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 > -w_0$$

One possible solution is

$w_0 = -1, \; w_1 = 1.1, \; w_2 = 1.1$

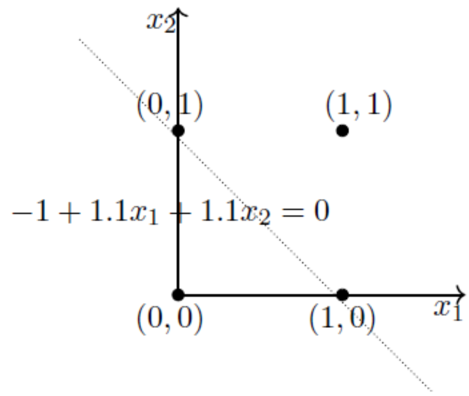


Figure 1: OR function using Perceptron

2 Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while !$convergence$ **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

Figure 2: Perceptron Learning Algorithm

3 Source Code

3.1 Class declaration

```
#ifndef __PERCEPTRON_
#define __PERCEPTRON_

class Perceptron
{
    float w1,w2,theta,lr;

    int X[4][3];
    int Y[4];

        public:
        Perceptron();

                int random(int min, int max);
                float getRandom();
                void initialise(int Y[4]);
                void train();
                int threshold(float y);
                void display();
};

#endif
```

3.2 Class definition

```
#include "Perceptron.h"
#include <iostream>
#include <string.h>
#include <time.h>

using namespace std;

Perceptron::Perceptron():
    X{{0,0,1},{0,1,1},{1,0,1},{1,1,1}}
{
    w1 = getRandom();
```

```
    w2 = getRandom();

    theta = getRandom();
}

int Perceptron::random(int min, int max)
{
        static bool first = true;

        if(first)
        {
                srand(time(NULL));
                first = false;
        }
        return min+rand()%((max+1)-min);
}

float Perceptron::getRandom()
{
        return random(-100,100)/100.0;
}

void Perceptron::initialise(int Y[4])
{
    this->Y[0] = Y[0];
    this->Y[1] = Y[1];
    this->Y[2] = Y[2];
    this->Y[3] = Y[3];
}

void Perceptron::train()
{
    bool updated;
    int epochs = 0;
    cout << "Weights initialised: " <<"w1: "<< w1 <<","<<"w2: "
         << w2<<","<<"theta: "<< theta<<endl;
    do
    {
        float prev_w1 = w1;
        float prev_w2 = w2;
        float prev_theta = theta;
```

```
        for(int i=0; i < 4; i++)
        {
            float y = w1*X[i][0] + w2*X[i][1] + theta*X[i][2];

            if(Y[i] == 0 && y > 0)
            {
                w1=w1-X[i][0];
                w2=w2-X[i][1];
                theta = theta-X[i][2];
                updated = true;
            }

            if(Y[i] == 1 && y <= 0)
            {
                w1=w1+X[i][0];
                w2=w2+X[i][1];
                theta = theta+X[i][2];
                updated = true;
            }
        }

        epochs++;

        if( prev_w1 == w1 &&
            prev_w2 == w2 &&
            prev_theta == theta )
        {
            break;
        }
    }
    while(1);

    cout << "Updated Weights: " <<"w1: "<< w1 <<","<<"w2: "
         << w2<<","<<"theta: "<< theta<<endl;
    cout << "Epochs: " << epochs <<endl;
}

int Perceptron::threshold(float y)
{
    return (y>0)?1:0;
```

```
}

void Perceptron::display()
{
        for(int i = 0; i < 4; i++)
        {
        float y = w1*X[i][0] + w2*X[i][1] + theta*X[i][2];
                int output = threshold(y);
                cout << "(" << X[i][0] <<","
            <<X[i][1]<<") ---> ("<<output<<")\n";
        }
    cout << "------------------------------"<<endl;
}
```

3.3 Driver code

```
#include <iostream>
#include "Perceptron.h"

using namespace std;

int main()
{
    Perceptron AND_Gate;

    int AND_output[4] = {0,0,0,1};

    cout << "AND_GATE"<<endl;
    cout << "********"<<endl;

    AND_Gate.initialise(AND_output);
    AND_Gate.train();
    AND_Gate.display();

    Perceptron OR_Gate;

    int OR_output[4] = {0,1,1,1};

    cout << "OR_GATE" <<endl;
    cout << "********" << endl;
```

```
    OR_Gate.initialise(OR_output);
    OR_Gate.train();
    OR_Gate.display();

    Perceptron NAND_Gate;

    int NAND_output[4] = {1,1,1,0};

    cout << "NAND_GATE"<<endl;
    cout << "********"<<endl;

    NAND_Gate.initialise(NAND_output);
    NAND_Gate.train();
    NAND_Gate.display();

    Perceptron NOR_Gate;

    int NOR_output[4] = {1,0,0,0};

    cout << "NOR_GATE"<<endl;
    cout << "********"<<endl;

    NOR_Gate.initialise(NOR_output);
    NOR_Gate.train();
    NOR_Gate.display();

    Perceptron XOR_Gate;

    int XOR_output[4] = {0,1,1,0};

    cout << "XOR_GATE"<<endl;
    cout << "********"<<endl;

    XOR_Gate.initialise(XOR_output);
    XOR_Gate.train();
    XOR_Gate.display();


        return 0;
}
```

3.4 Output of Logic gate realisation

3.4.1 AND Gate

```
AND GATE
********
Weights initialised: w1: 0.13,w2: -0.99,theta: 0.88
Updated Weights: w1: 1.13,w2: 1.01,theta: -2.12
Epochs: 7
(0,0) --> (0)
(0,1) --> (0)
(1,0) --> (0)
(1,1) --> (1)
```

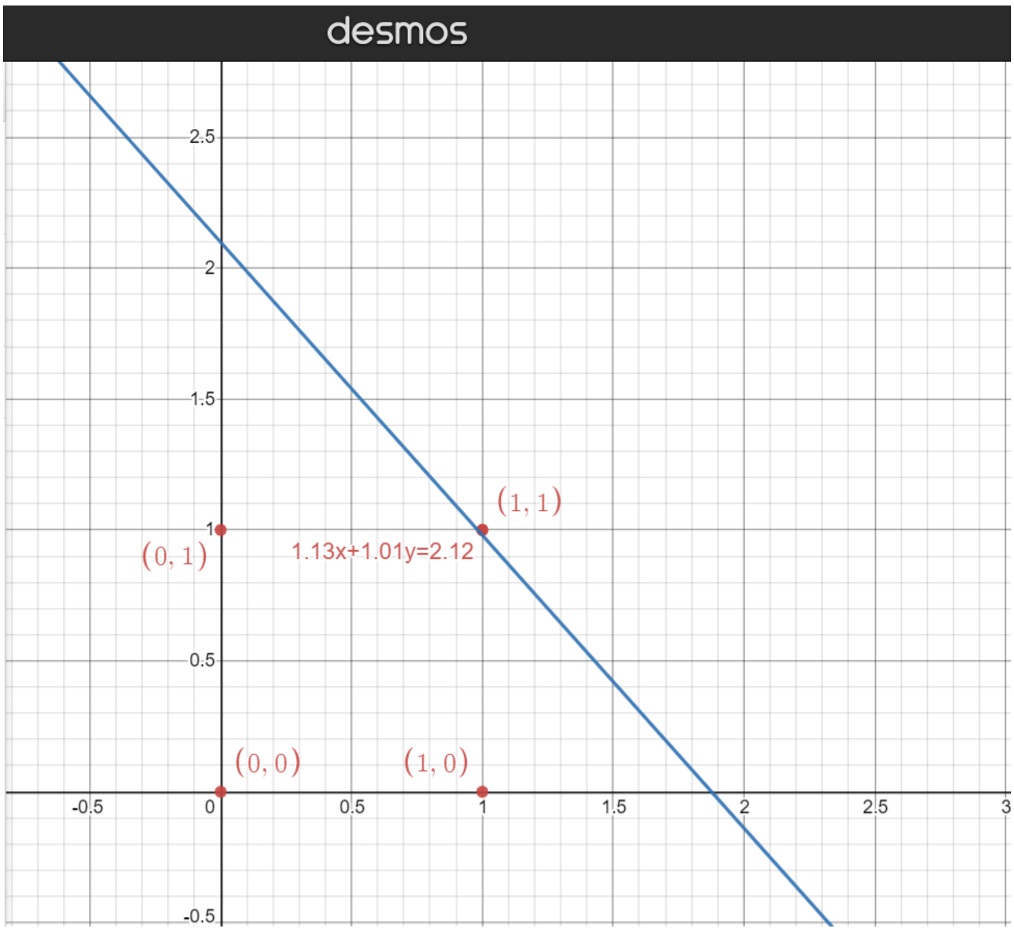


Figure 3: AND Gate

3.4.2 OR Gate

```
OR GATE
********
Weights initialised: w1: -0.89,w2: 0.03,theta: -0.12
Updated Weights: w1: 1.11,w2: 1.03,theta: -0.12
Epochs: 5
(0,0) --> (0)
(0,1) --> (1)
(1,0) --> (1)
(1,1) --> (1)
_____________________________
```

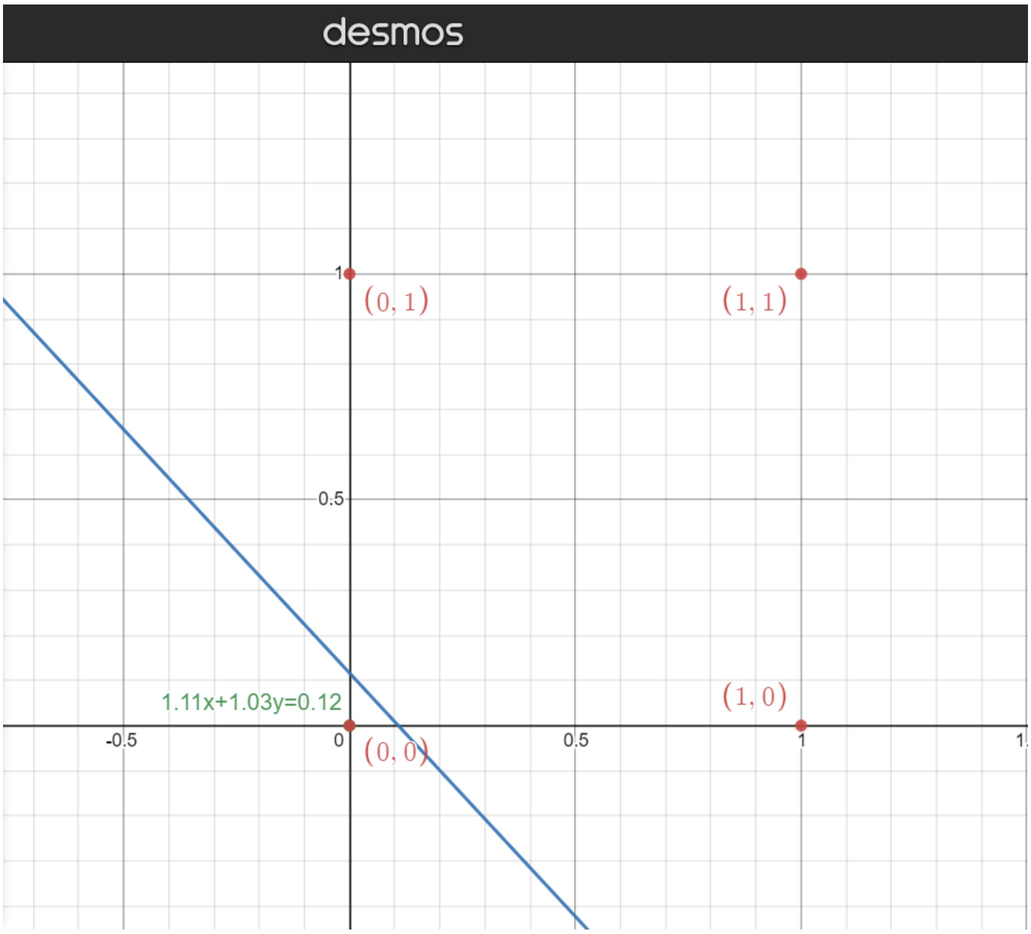


Figure 4: OR Gate

3.4.3 NAND Gate

```
NAND GATE
********
Weights initialised: w1: 0.79,w2: 0.36,theta: -0.79
Updated Weights: w1: -2.21,w2: -1.64,theta: 3.21
Epochs: 9
(0,0) --> (1)
(0,1) --> (1)
(1,0) --> (1)
(1,1) --> (0)
______________________________
```

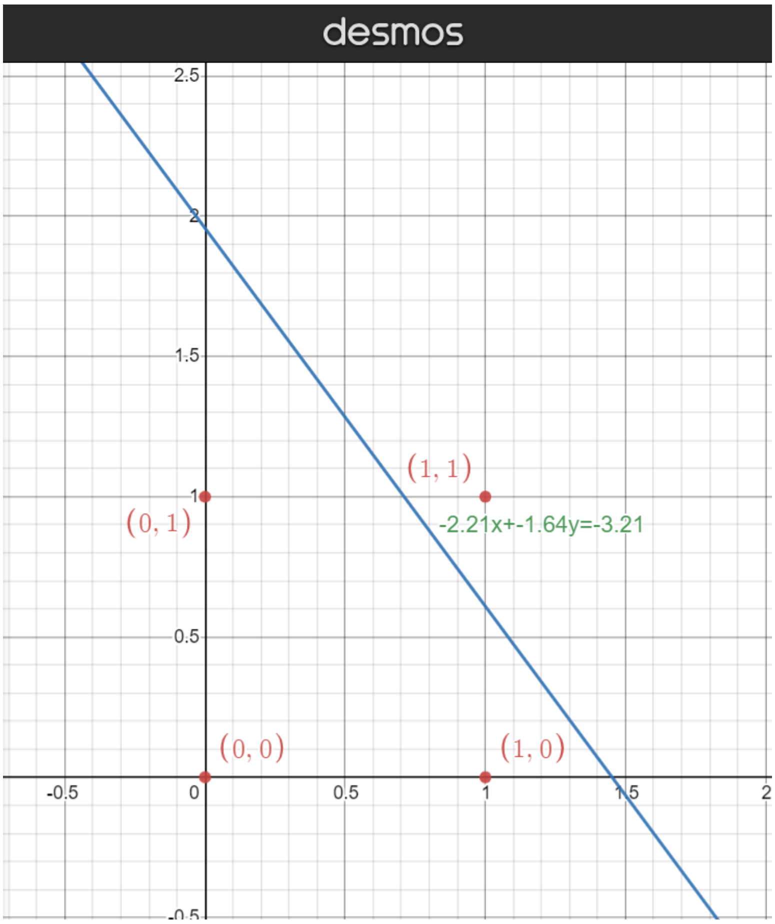


Figure 5: NAND Gate

3.4.4 NOR Gate

```
NOR GATE
********
Weights initialised: w1: -0.36,w2: 0.85,theta: 0.85
Updated Weights: w1: -1.36,w2: -1.15,theta: 0.85
Epochs: 5
(0,0) --> (1)
(0,1) --> (0)
(1,0) --> (0)
(1,1) --> (0)
```

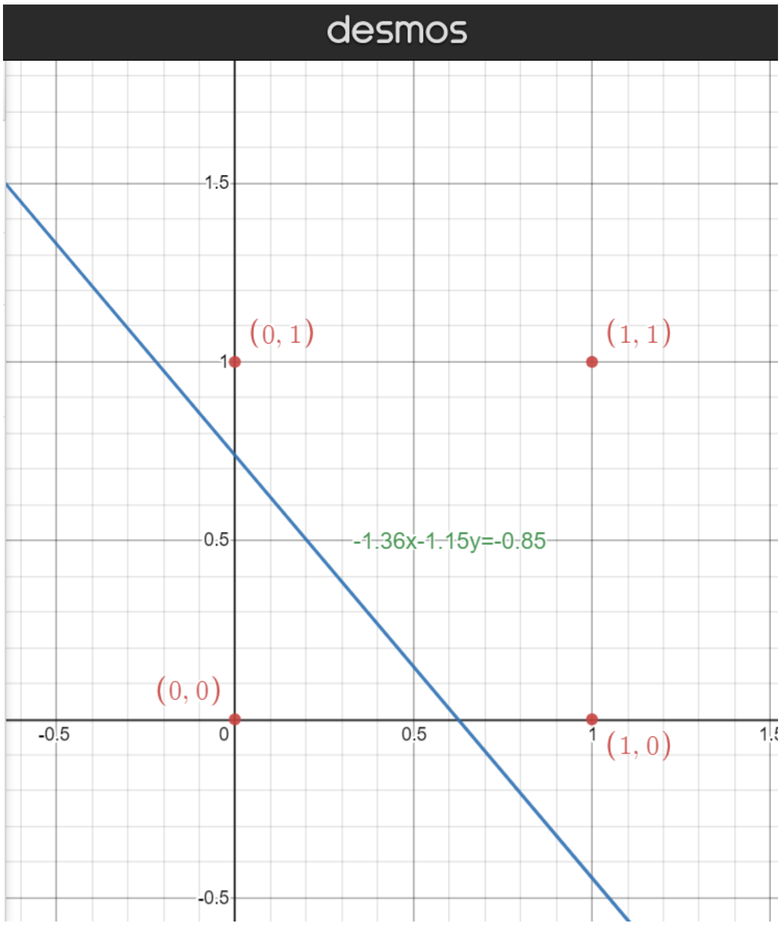


Figure 6: NOR Gate

3.4.5 XOR Gate

```
XOR GATE
********
Weights initialised: w1: 0.01,w2: 0.41,theta: -0.2
Updated Weights: w1: -0.99,w2: -0.59,theta: 0.8
Epochs: 4
(0,0) --> (1)
(0,1) --> (1)
(1,0) --> (0)
(1,1) --> (0)
```

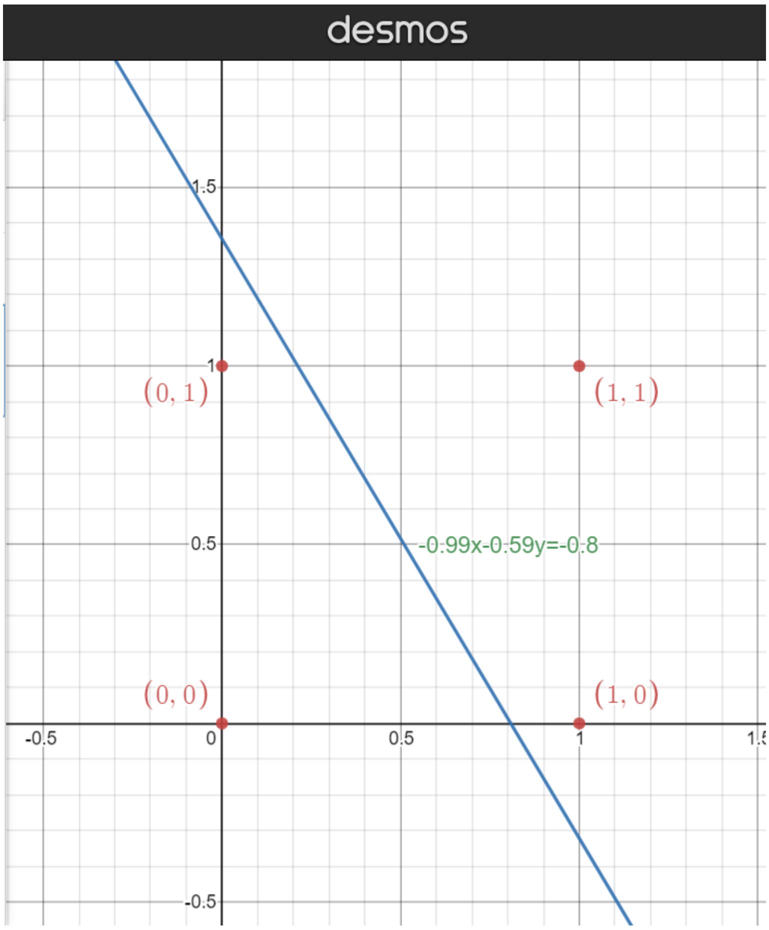


Figure 7: XOR Gate

3.5 Conclusion from the experiment

Realisation of gates AND,OR,NAND and NOR is possible with a single layer perceptron.But realisation of XOR Gate failed. It cannot be implemented using single layer perceptron as it is not a non-linear problem. Need to realise using a hidden layer.