

Ejercicio



Neuronas (Compuertas Lógicas)

Integrantes del equipo

César Mauricio Arellano Velásquez

Profesor

César Arturo Ángeles Ruiz

Materia

Taller de Desarrollo de Aplicaciones

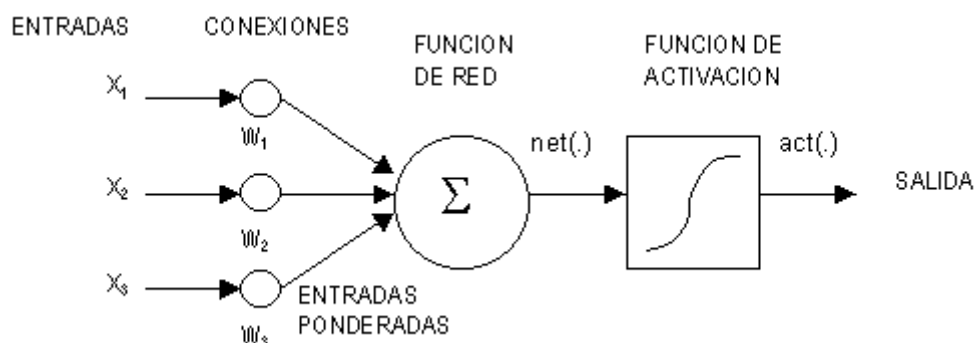
Introducción:

La llamada ley de Hebb, propuesta por el neuro psicólogo Donald Hebb, afirma que las conexiones sinápticas se fortalecen cuando dos o más neuronas se activan de forma contigua en el tiempo y en el espacio. Al asociarse el disparo de la célula presináptica con la actividad de la postsináptica tienen lugar cambios estructurales que favorecen la aparición de ensamblajes o redes neuronales.

La propuesta de Hebb tuvo un fuerte impacto en la neuropsicología, llegando a constituir el núcleo de muchos planteamientos desarrollados en las décadas posteriores, y sigue siendo una referencia muy importante en este campo en la actualidad.

A principios de los años 70 se descubrió la existencia de un mecanismo muy relevante para el aprendizaje: la potenciación a largo plazo, que consiste en la consolidación de los recuerdos a través de la experiencia repetida. Así, la memoria a corto plazo se asienta por cambios estructurales (expresión de genes, síntesis de proteínas y cambios en las sinapsis).

$$\Delta w_{ij}(t) = \varepsilon \cdot (t^{\mu_i} - y^{\mu_i}) x^{\mu_j} \quad \rightarrow \text{regla del perceptrón}$$



Objetivos:

- Comprender el funcionamiento del aprendizaje de una red neuronal.
- Entender la lógica básica detrás de la inteligencia artificial.
- Saber cómo se comporta la ley de Hebb a nivel de software.

Descripción del Problema.

Entrenar 3 neuronas por separado que sean capaces de realizar las funciones de compuertas lógicas AND, OR, NOT a través de la regla de Hebb.

Análisis

Pseudocódigo:

```
#define return -> retorna el valor de alguna operación o variable.
#define fopen() -> Abre un archivo en un modo determinado
(Lectura, Escritura, Añadir).
#define fprintf() -> Escribe en un archivo de texto determinado.
#define fclose() -> Cierra un archivo.
#define exit(0) -> Finaliza la ejecución del programa.
#define drand48() -> Genera número double del 0 al 1.
```

```
Principal()
{
    DatosAND[4][3] = {{-1, -1, -1}, {-1, 1, -1}, {1, -1, -1}, {1, 1, 1}};
    DatosOR[4][3] = {{-1, -1, -1}, {-1, 1, 1}, {1, -1, 1}, {1, 1, 1}};
    DatosNOT[2][2] = {{-1, 1}, {1, -1}};
    InicializarNeuronasAND_OR( | NeuronaAND);
    InicializarNeuronasAND_OR( | NeuronaOR);
    InicializarNeuronasNOT( | NeuronaNOT);
    EntrenamientoAND_OR(DatosAND, NombreArchivoAND | NeuronaAND);
    EntrenamientoAND_OR(DatosOR, NombreArchivoOR | NeuronaOR);
    EntrenamientoNOT(DatosNOT, NombreArchivoNOT | NeuronaNOT);
    Imprimir("Entrenamiento Finalizado.");
    Imprimir("Presione Enter para continuar...");
    Haz
    {
        Imprimir("Menú Implementación de generador de gráficos");
        Imprimir("1.- AND");
        Imprimir("2.- OR");
        Imprimir("3.- NOT");
        Imprimir("4.- Salir");
        Imprimir("Escoge una opción");
        Leer(Opcion);
        Seleccion(Opcion)
        {
            caso 1:
                Resultados(NeuronaAND, Tipo | );
                romper;
            caso 2:
                Resultados(NeuronaOR, Tipo | );
                romper;
            caso 3:
                Resultados(NeuronaNOT, Tipo | );
                romper;
            caso 4:
```

```

        Imprimir("Saliendo del programa...");
        exit(0);
        romper;
    Por defecto:
        Imprimir("Opción incorrecta, intente de nuevo.");
        romper;
    }
    Imprimir("Presione Enter para continuar.");
}mientras(Opcion<>4);
}

InicializarNeuronasAND_OR( | Neurona)
{
    ↑ Neurona.W[0] = drand48();
    ↑ Neurona.W[1] = drand48();
    ↑ Neurona.Bias = drand48();
}
InicializarNeuronasNOT( | Neurona)
{
    ↑ Neurona.W[0] = drand48();
    ↑ Neurona.Bias = drand48();
}
EntrenamientoAND_OR(Datos, NombreArchivo | Neurona)
{
    Archivo = fopen(NombreArchivo, "wt");
    N = 0.01;
    Posicion = 0;

    Desde i = 0; Hasta i < 1000; i++
    {
        Si (Posicion == 4)
            Posicion = 0;
        ↑ Neurona.Error=Datos[Posicion][2]-ObtenerForwardPass(Neurona,Datos[Posicion],Tipo | );

        Desde j = 0; Hasta j < 2; j++
            ↑ Neurona.W[j] = ↑ Neurona.W[j] + N * ↑ Neurona.Error *
Datos[Posicion][j];

        ↑ Neurona.Bias += ↑ Neurona.Error * N;
        fprintf(Archivo, "%lf", %lf, %lf\n",
↑ Neurona.W[0], ↑ Neurona.W[1], ↑ Neurona.Error);
        Posicion++;
    }

    fclose(Archivo);
}

EntrenamientoNOT(Datos, NombreArchivo | Neurona)
{

```

```

Archivo = fopen(NombreArchivo, "wt");
N = 0.01;
Posicion = 0;

Si (Posicion == 4)
    Posicion = 0;
    ↑ Neurona.Error=Datos[Posicion][1]-ObtenerForwardPass(Neurona,Datos[Posicion],Tipo | );
    ↑ Neurona.W[0] = ↑ Neurona.W[0] + N * ↑ Neurona.Error *
Datos[Posicion][0];

    ↑ Neurona.Bias += ↑ Neurona.Error * N;
    fprintf(Archivo, "%lf, %lf, %lf\n",
↑ Neurona.W[0], ↑ Neurona.W[1], ↑ Neurona.Error);
    Posicion++;
}

fclose(Archivo);
}

Resultados(Neurona,Tipo | )
{
    Si(Tipo == 2)
    {
        Imprimir("Ingrese A");
        Leer(EntradasX[0]);
        printf("Ingrese B");
        Leer(EntradasX[1]);
        Imprimir("El resultado es: " + Verificacion(Resultado | ));
    }
    Si no
    {
        Imprimir("Ingrese A");
        Leer(EntradasX[0]);
        Imprimir("El resultado es: " + Verificacion(Resultado | ));
    }
}

ObtenerForwardPass(Neurona, EntradasX[], int Tipo | )
{
    Suma = 0;
    Desde i = 0; Hasta i < Tipo; i++
        Suma += EntradasX[i] * ↑ Neurona.W[i];
    Suma += ↑ Neurona.Bias;
    return Suma;
}

Verificacion(Resultado | )
{
    Si(Resultado < 0)
        return -1;
    Si no

```

```
    return 1;
}
```

Código

Archivo neuronas.c

```
/*
 * @author: César Mauricio Arellano Velásquez
 * @date: 19/Noviembre/2019
 * @file: neuronas.c
 * @brief: Implementación de la ley de Hebb en una neurona para simular
compuertas lógicas AND/OR/NOT.
 */

// INCLUYENDO LAS BIBLIOTECAS A UTILIZAR
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

// DEFINIENDO LA ESTRUCTURA DE CADA NEURONA
typedef struct Neuron
{
    double W[2], Bias, Error;
    int X[2];
}TipoNeurona;

// Estructuras de las funciones
void InicializarNeuronasAND_OR(TipoNeurona *Neurona);
void InicializarNeuronasNOT(TipoNeurona *Neurona);
void EntrenamientoAND_OR(TipoNeurona* Neurona, double Datos[4][3], char
NombreArchivo[]);
void EntrenamientoNOT(TipoNeurona* Neurona, double Datos[2][2], char
NombreArchivo[]);
double ObtenerForwardPass(TipoNeurona *Neurona, double EntradasX[], int Tipo);
void Resultados(TipoNeurona *AND, int Tipo);
int Verificacion(double Resultado);

int main (void)
{
    TipoNeurona NeuronaAND, NeuronaOR, NeuronaNOT;
    int Opcion;
    double DatosAND[4][3] = {{-1, -1, -1}, {-1, 1, -1}, {1, -1, -1}, {1, 1, 1}};
    double DatosOR[4][3] = {{-1, -1, -1}, {-1, 1, 1}, {1, -1, 1}, {1, 1, 1}};
    double DatosNOT[2][2] = {{-1, 1}, {1, -1}};
    InicializarNeuronasAND_OR(&NeuronaAND);
    InicializarNeuronasAND_OR(&NeuronaOR);
    InicializarNeuronasNOT(&NeuronaNOT);
    EntrenamientoAND_OR(&NeuronaAND, DatosAND, "NeuronaAND.txt");
    EntrenamientoAND_OR(&NeuronaOR, DatosOR, "NeuronaOR.txt");
    EntrenamientoNOT(&NeuronaNOT, DatosNOT, "NeuronaNOT.txt");
    printf("\nEntrenamiento Finalizado.\nPresione Enter para continuar...\n");
    __fpurge(stdin);
    getchar();
}
```

```

do
{
    system("clear");
    printf("Menú Implementacion de generador de gráfos\n");
    printf("1.- AND\n");
    printf("2.- OR\n");
    printf("3.- NOT\n");
    printf("4.- Salir\n");
    printf("Escoge una opción\n");
    scanf(" %d",&Opcion);
    system("clear");
    switch(Opcion)
    {
        case 1:
            Resultados(&NeuronaAND,2);
            break;
        case 2:
            Resultados(&NeuronaOR,2);
            break;
        case 3:
            Resultados(&NeuronaNOT,1);
            break;
        case 4:
            printf("Saliendo del programa...\n");
            exit(0);
            break;
        default:
            printf("Opción incorrecta, intente de nuevo.\n");
            break;
    }
    printf("\nPresione Enter para continuar.");
    __fpurge(stdin);
    getchar();
}while(Opcion != 4);
}

/* * Funcion que inicializa los valores de los pesos y el bias de la neurona.
 * @param TipoNeurona* Neurona actualiza la estructura de la neurona AND y OR.
 */
void InicializarNeuronasAND_OR(TipoNeurona *Neurona)
{
    srand48(time(NULL));
    Neurona->W[0] = drand48();
    Neurona->W[1] = drand48();
    Neurona->Bias = drand48();
}

/* * Funcion que inicializa los valores del peso y el bias de la neurona.
 * @param TipoNeurona* Neurona actualiza la estructura de la neurona NOT.
 */
void InicializarNeuronasNOT(TipoNeurona *Neurona)
{
    srand48(time(NULL));
    Neurona->W[0] = drand48();

```

```

    Neurona->Bias = drand48();
}
/* * Funcion que realiza el entrenamiento de las neuronas AND y OR.
    * @param TipoNeurona* Neurona actualiza los pesos para interpretar las
    entradas y así arrojar la salida deseada.
    * @param double Datos[][] Arreglo bidimensional con las entradas y salidas
    deseadas
    * @param char NombreArchivo Cadena que indica el nombre del archivo a
    registrar los pesos y el error.
*/
void EntrenamientoAND_OR(TipoNeurona* Neurona, double Datos[4][3], char
NombreArchivo[])
{
    FILE *Archivo;
    Archivo = fopen(NombreArchivo, "wt");
    double N = 0.01;
    int Posicion = 0;

    for(int i = 0; i < 1000; i++)
    {
        if(Posicion == 4)
            Posicion = 0;

        Neurona->Error = Datos[Posicion][2] -
ObtenerForwardPass(Neurona,Datos[Posicion],2);

        for(int j = 0 ; j < 2;j++)
            Neurona->W[j] += N * Neurona->Error * Datos[Posicion][j];

        Neurona->Bias += Neurona->Error * N;
        fprintf(Archivo, "%lf, %lf, %lf\n",
Neurona->W[0],Neurona->W[1],Neurona->Error);
        Posicion++;
    }

    fclose(Archivo);
}

/* * Funcion que realiza el entrenamiento de las neuronas NOT.
    * @param TipoNeurona* Neurona actualiza el peso para interpretar la entrada y
    así arrojar la salida deseada.
    * @param double Datos[][] Arreglo bidimensional con las entradas y salidas
    deseadas
    * @param char NombreArchivo Cadena que indica el nombre del archivo a
    registrar los pesos y el error.
*/
void EntrenamientoNOT(TipoNeurona* Neurona, double Datos[2][2], char
NombreArchivo[])
{
    FILE *Archivo;
    Archivo = fopen(NombreArchivo, "wt");
    double N = 0.01;
    int Posicion = 0;

```



```

for(int i = 0; i < 1000; i++)
{
    if(Posicion == 2)
        Posicion = 0;

        Neurona->Error = Datos[Posicion][1] -
ObtenerForwardPass(Neurona,Datos[Posicion],1);
        Neurona->W[0] += N * Neurona->Error * Datos[Posicion][0];
        Neurona->Bias += Neurona->Error * N;
        fprintf(Archivo, "%lf, %lf\n", Neurona->W[0],Neurona->Error);
        Posicion++;
    }

    fclose(Archivo);
}

/* * Funcion que da los resultados de la compuertas lógicas AND, OR y NOT.
    * @param TipoNeurona* Neurona al terminar el entrenamiento, esta se usa para
desplegar la salida esperada dependiendo la combinación de la compuerta lógica.
    * @param int Tipo Indica que tipo de compuerta es: AND, OR o NOT.
*/
void Resultados(TipoNeurona *Neurona, int Tipo)
{
    double EntradasX[2];
    if(Tipo == 2)
    {
        printf("Ingrese A\n");
        scanf(" %lf",&EntradasX[0]);
        printf("Ingrese B\n");
        scanf(" %lf",&EntradasX[1]);
        printf("El resultado es: %d",
Verificacion(ObtenerForwardPass(Neurona,EntradasX,Tipo)));
    }
    else
    {
        printf("Ingrese A\n");
        scanf(" %lf",&EntradasX[0]);
        printf("El resultado es: %d",
Verificacion(ObtenerForwardPass(Neurona,EntradasX,Tipo)));
    }
}

/* * Funcion que obtiene la salida estimada de las neuronas AND, OR. NOT
    * @param TipoNeurona* Neurona calcula los pesos para interpretar las entradas
y así arrojar la salida deseada.
    * @param double EntradasX[] Arreglo bidimensional con las entradas.
    * @param int Tipo Indica que tipo de compuerta es: AND, OR o NOT.
*/
double ObtenerForwardPass(TipoNeurona *Neurona, double EntradasX[], int Tipo)
{
    double Suma = 0;
    for(int i = 0; i < Tipo; i++)

```

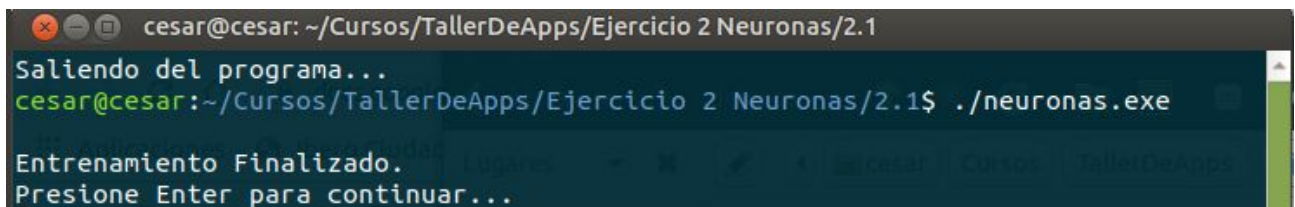
```

        Suma += EntradasX[i] * Neurona->W[i];
        Suma += Neurona->Bias;
        return Suma;
    }

    /* * Funcion que verifica la salida estimada para obtener la salida real.
        * @param double Resultado es threshold que nos indica que salida arrojar
        dependiendo de la salida estimada.
    */
    int Verificacion(double Resultado)
    {
        if(Resultado < 0)
            return -1;
        else
            return 1;
    }
}

```

Ejecución del programa

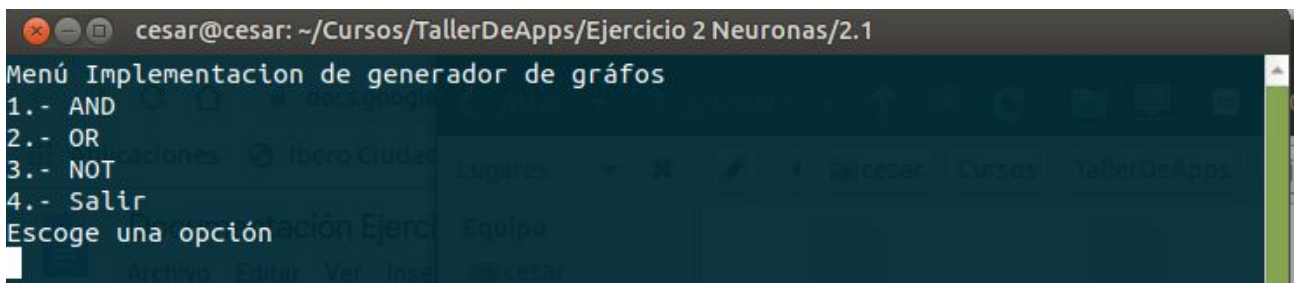


```

cesar@cesar: ~/Cursos/TallerDeApps/Ejercicio 2 Neuronas/2.1
Saliendo del programa...
cesar@cesar:~/Cursos/TallerDeApps/Ejercicio 2 Neuronas/2.1$ ./neuronas.exe

Entrenamiento Finalizado.
Presione Enter para continuar...

```

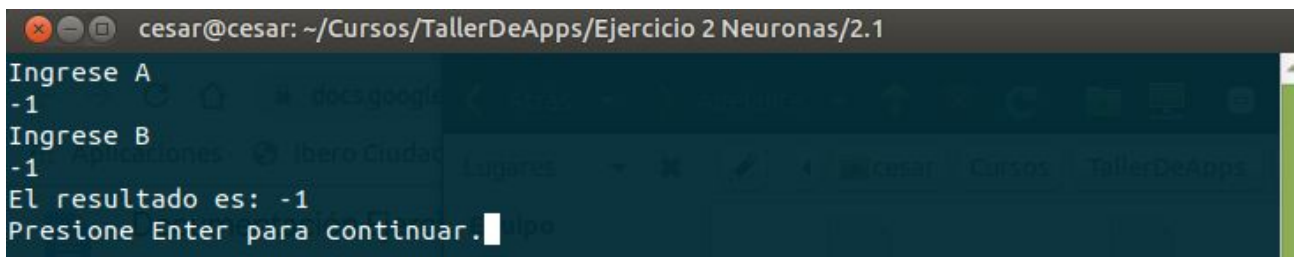


```

cesar@cesar: ~/Cursos/TallerDeApps/Ejercicio 2 Neuronas/2.1
Menú Implementacion de generador de gráficos
1.- AND
2.- OR
3.- NOT
4.- Salir
Escoge una opción

```

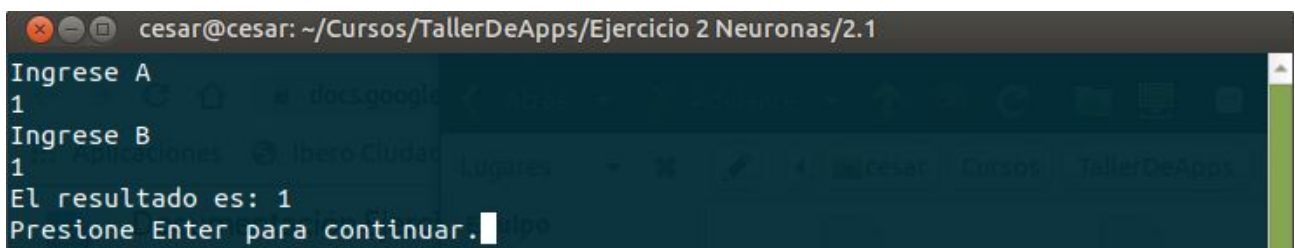
AND



```

cesar@cesar: ~/Cursos/TallerDeApps/Ejercicio 2 Neuronas/2.1
Ingrese A
-1
Ingrese B
-1
El resultado es: -1
Presione Enter para continuar.

```



```

cesar@cesar: ~/Cursos/TallerDeApps/Ejercicio 2 Neuronas/2.1
Ingrese A
1
Ingrese B
1
El resultado es: 1
Presione Enter para continuar.

```

OR

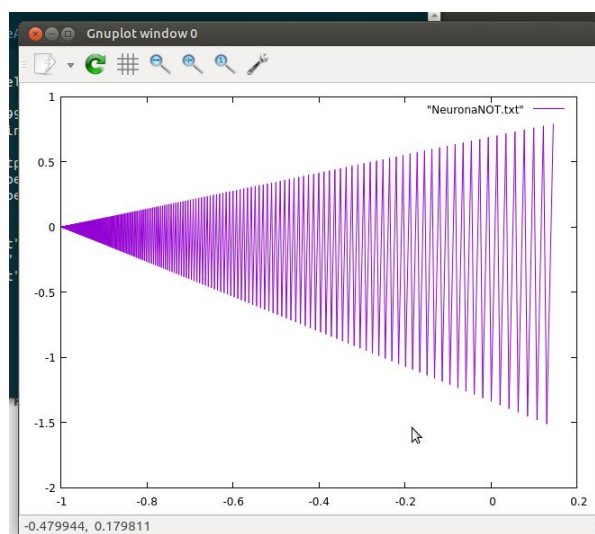
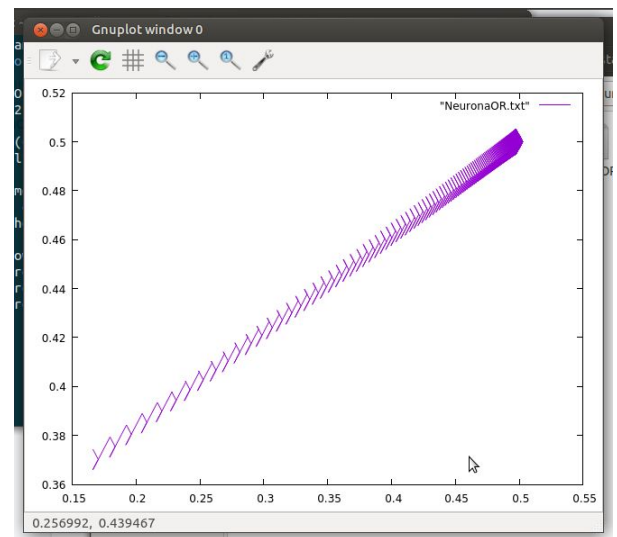
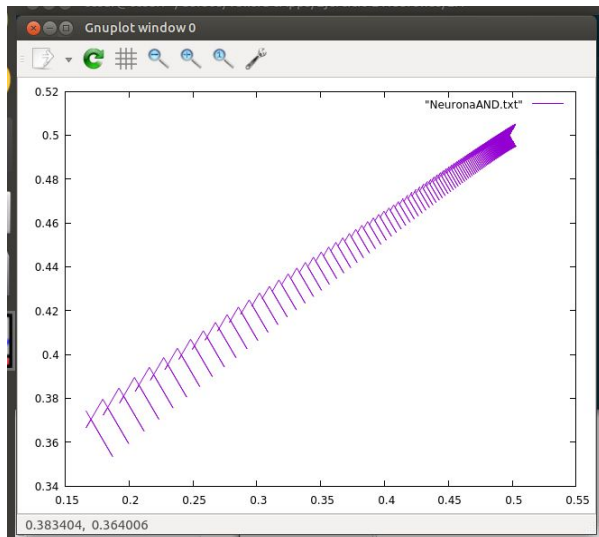
```
cesar@cesar: ~/Cursos/TallerDeApps/Ejercicio 2 Neuronas/2.1
Ingrese A
-1
Ingrese B
1
El resultado es: 1
Presione Enter para continuar.
```

NOT

```
cesar@cesar: ~/Cursos/TallerDeApps/Ejercicio 2 Neuronas/2.1
Ingrese A
-1
El resultado es: 1
Presione Enter para continuar.
```

```
cesar@cesar: ~/Cursos/TallerDeApps/Ejercicio 2 Neuronas/2.1
Ingrese A
1
El resultado es: -1
Presione Enter para continuar.
```

Gráficas -> AND, OR, NOT



Conclusión:

Gracias a este ejercicio comprendí al menos una base de cómo es que se puede programar el aprendizaje de una neurona, para que esta arroje los resultados deseados dependiendo de las entradas dadas, lo que nos permite sacar provecho de esto y desatar la creatividad para desarrollar proyectos interesantes con este nuevo conocimiento.