INSTITUTO TECNOLOGICO DE CIUDAD GUZMAN

UNIDAD 2
PERCEPTRON

Alumnos:

+ Angel Francisco Sanches Noriega
+ Juan Carlos Arias Guzman
+ Juan Carlos Guzman Rosales

INDICE

1. Problema a resolver

Programar la estructura de red neuronal artificial "perceptron", creando los componentes de la interfaz grafica de usuario, necesarios para la representación neurobiológica de una neurona, las partes de la interfaz grafica, se enumeran a continuación.

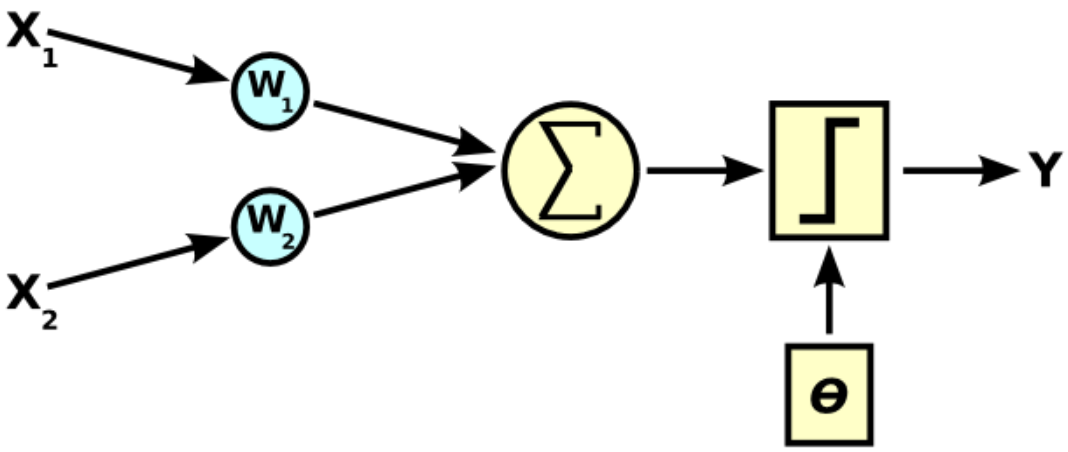

Figura 1.- Representación Grafica de Perceptrón.

a) **X1, X2,…xn,** xn corresponde a la entrada presináptica de aferentes i.
b) **W1, W2,…WN**, peso **wi** encapsula la fuerza sináptica correspondiente.
c) **SUMATORIA,** El **wix**i producto es similar a la del potencial post-sináptico (PSP), que es inhibitorio / excitatorio en función de si es wi negativo / positivo
   - La integración del potencial post-sináptico sobre el eje de dendritas con el soma es representado por una simple suma aritmética y la cantidad de *α* al potencial de la membrana somatica

d) Esta información es transformada por la función de aplastamiento que nos ayuda a disminuir el rango de la salida, usualmente en un rango de [0,1]
e) La constante Θ "bias" define el punto en el cual ***y*** tiene su valor medio, además es el punto donde la función está cambiando más rápidamente y es por lo tanto, el valor de la activación en el cual el nodo es más sensible a pequeños cambios en las entradas.

2. Propuesta de solución con software

Figura 2. Especificación de Interfaz Grafica de Usuario Programa Perceptrón 1.

a) Las entradas, los pesos pre sinápticos y la constante bias se capturan mediante JTextField.

Figura 3. Especificación de Interfaz Grafica de Usuario Programa Perceptrón 2.

b) Los resultados se muestran mediante JTextField al presionar el botón entrenar

3. Explicación del programa

A. Cuando se presiona el botón entrenar manda a llamar los siguientes métodos que capturan los valores, realizan la sumatoria de la multiplicación de las entradas y pesos sinápticos mas el valor de bias, el valor resultante del método **ent(),** lo pasan como parámetro a la función de activación **hardlim(),** y si existe **error(),** someten el resultado a la función de entrenamiento **traine()** hasta que el valor de **error()** sea igual a o. Lo mencionado anteriormente se muestra en código en la **figura 4.**

```
216    private void EntrenarActionPerformed(java.awt.event.ActionEvent evt) {
217
218            getX1(); //obtener datos de x1
219            getX2(); //obtener datos de x2
220            getW1(); //obtener datos de w1
221            getW2(); //obtener datos de w2
222            getBias(); //obtener datos de bias
223
224                    net(); //Calcular suma aritmetica
225                    setU(); //Settear valor resultante a JTextField u
226                    hardlim(net); //Clasificar valor Resultante de net()
227                    setyResultante(yResultante); //settear valor resultante a JtextField YR
228                    error(); //Calcular Error
229
230                    do{ //si existe error arrojado por el metodo error(, mandar llamar a traine
231                        traine(); // recalcular pesos y reasignarlos
232                    }
233                    while(error!=0); //hasta que no haya error
234
235    }
```

Figura 4.- acciones del evento de botón entrenar, Primera iteración.

B. A continuación se presenta una serie de imágenes que detalla mediante código lo que realiza cada método mencionado en la descripción anterior.

```
353    public int net(){
354        net=(int)getX1()*(int)getW1()+(int)getX2()*(int)getW2()+(int)getBias();
355    return net;
356    }
```

Figura 5.- Método que Calcula suma aritmética.

```
344      public int hardlim(int net){
345          if(net()<=0){
346          setyResultante(0);
347          }
348          else{
349          setyResultante(1);
350          }
351      return yResultante;
352      }
```

Figura 6.- Método que Clasifica el resultado del método net(), función de activación Hardlim.

```
357      public int error(){
358          error=getYDeseada()-getYResultante();
359      return error;
360      }
```

Figura 7.- Método que evalúa el error, utilizando métodos que obtienen los valores arrojados por el método hardlim(int net).

```
361      public int traine(){
362          setIncrementoBias(incrementoBias); //utilza como parametro el valor actual de bias para recalcular incremento
363          setIncrementoPesoSinapticoW1(incrementoPesoSinapticoW1);
364          //utilza como parametro el valor actual de w1 para recalcular incremento
365          setIncrementoPesoSinapticoW2(incrementoPesoSinapticoW2);
366          //utilza como parametro el valor actual de w2 para recalcular incremento
367          setW1(incrementoPesoSinapticoW1);
368          //utilza como parametro el valor de retorno del metodo setIncrementoPesoSinapticoW1 para recalcular incremento
369          setW2(incrementoPesoSinapticoW2);
370          //utilza como parametro el valor de retorno del metodo setIncrementoPesoSinapticoW2 para recalcular incremento
371          setBias(incrementoBias);
372          //utilza como parametro el valor de retorno del metodo setIncrementoBias para recalcular incremento
373          net();
374          hardlim(yResultante);
375          error();
376
377          numeroIteraciones+=numeroIteraciones;
378          //contador de numero de iteraciones
379      return error;
380      }
```

Figura 8.- Método que se encarga de reasignar valores a los elementos x1,x2,w1,w2,bias y net(), si existe error y como valor de retorno es el error que resulta para su próxima evaluación fuera de este método.

```
334     public double setIncrementoBias(double incrementoBias) {
335         return this.incrementoBias = incrementoBias+getU()*error();
336     }
337     public double setIncrementoPesoSinapticoW1(double incrementoPesoSinapticoW1) {
338         return this.incrementoPesoSinapticoW1 = incrementoPesoSinapticoW1+getU()*error()*getX1();
339     }
340     public double setIncrementoPesoSinapticoW2(double incrementoPesoSinapticoW2) {
341      return   this.incrementoPesoSinapticoW2 = incrementoPesoSinapticoW2+getU()*error()*getX2();
342     }
```

Figura 9.- Métodos utilizados en el método traine().

4. Conclusiones

Al realizar la programación de esta práctica, logramos entender en su totalidad la función, y algunos posibles usos de la estructura de red neuronal "perceptron", ya que identificamos como representar mediante algoritmo y interfaz de usuario cada una de las partes de esta estructura básica.