Red Perceptron (clasificación)

La red perceptrón es una red de propagación hacia delante (feedforward), supervisada. En esta red, la salida se calcula directamente a partir de la entrada en un solo paso y no se requiere retropropagación (feedback). Normalmente se utiliza en reconocimiento de patrones y como aproximador a funciones. Lo que es util para calcular filtros adaptativos y control automático.

Comportamiento básico

La siguiente técnica es valida para pocas neuronas de entrada, pero es suficiente para entender como funciona esta red. Solo se puede usar en datos linealmente separables.

Diseño

- Definición de la red perceptrón:

$n = wp + b$

- Si son 2 entradas y una neurona

$n = w_{1,1} p_1 + w_{2,1} p_2 + b;$

En la frontera n=0 entonces se da el cambio.

$0 = w_{1,1} p_1 + w_{2,1} p_2 + b;$

- Cruces por cero, es decir

si $P1=0 \rightarrow w_{2,1} = -b/P2$

si $P2=0 \rightarrow w_{1,,1} = -b/P2$

- Para verificar hay que checar que se clasifique bien.

Fronteras de decisión, es donde se da el cambio de clase, si se usa una hardlimit es cuando se da el cambio de 0 a 1 o de 1 a 0, en el valor de n.

Entrenamiento [2]

- w nueva= w anterior + e pT
- b nueva= b anterior + e
- donde e es el error :

e = target – salida anterior

Reglas de aprendizaje generalizado para una red perceptrón.

Para poder generalizar el comportamiento de la red perceptrón es importante que es una regla de aprendizaje.

Una regla de aprendizaje, es un procedimiento para modificar los pesos y bias de una red y también se le conoce como algoritmo de entrenamiento. EL propósito de la regla de aprendizaje es estrenar a la red para realizar una tarea. Existen diversos tipos de algoritmos, pero en general caen en tres categorías:

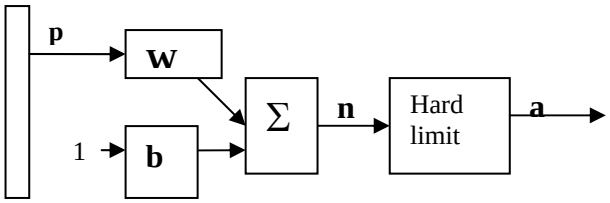
1. Aprendizaje supervisado, en este tipo de algoritmos se proporcionan ejemplos denominado set de entrenamiento (training set) y el conjunto de respuestas que estamos esperando a la salida y que normalmente denominamos set de salida objetivo (target set). El algoritmo se usa para ajustar los pesos y los bias para ajustar la respuesta de la neurona a lo que se desea obtener. La regla que usa la red perceptrón es un caso típico de este tipo de algoritmo.

2. Aprendizaje no supervisado, en este caso los pesos y los bias se ajustan solamente con las entradas, ya que no se cuenta con los set de salidas esperadas. Parecería no tener sentido, pero casi todas las redes que usan esta técnica usan alguna técnica de agrupamiento. Aprenden a clasificar en un número finito de clases. Útil en la cuantización vectorial.

3. Aprendizaje reforzado, es parecido al aprendizaje supervisado, excepto que en lugar de dar un conjunto de salida deseada u objetivo, se le da un grado o medida, que está asociado a el desempeño de la red. Generalmente se usa en sistemas de control.

Para generalizar podemos proponer la red perceptrón que vimos al inicio, con una función de activación tipo Hard limit, es decir

f(n) = 1 si n>0 y f(n)=0 si n<0;



a= hardlim (Wp+b)

donde $\mathbf{W}_{RxS}$ es la matriz de pesos, R es el numero de entradas (numero de columnas) y S es el numero de neuronas(numero de filas).

Por filas, se puede encontrar todos los pesos asociados a una neurona y podemos definir un vector columna ${}_i\mathbf{w} = [w_{i,1}; w_{i,2}; w_{i,3} \ldots ; w_{i,R}]$

Entonces podemos redefinir a la matriz a partir de los nuevos vectores:
$\mathbf{W} = [{}_1\mathbf{w}^T; {}_2\mathbf{w}^T; {}_3\mathbf{w}^T; \ldots; {}_S\mathbf{w}^T;];$

Por lo que entonces el algoritmo se puede redefinir como

$a_i = \text{hardlim}(n_i) = \text{hardlim}({}_i\mathbf{w}^T\mathbf{p} + b_i)$

donde i es el numero de filas, es decir el numero de neuronas.

Para evaluar el programa, proponemos patrones de prueba, dentro del rango de trabajo de la red y le solicitaremos que los clasifique utilizando los w´s y b´s calculados.

Ejemplo de Algoritmo de entrenamiento básico.

```
%entrenamiento de una red perceptron para clasificar 8 puntos
close all;clear all;clc;
P=[-1 -2 -3 -3 -1 1 2 1;3 2 0 -2 -3 -2 0 1];
T=[1 1 1 1 0 0 0 0;1 1 0 0 0 0 1 1];
W=rand(2,2);
b=rand(2,1);
epocas=input('Ingrese el numero de èpocas de entrenamiento:');
[m,n]=size(P);
for x=1:epocas
    for y=1:n
        a=hardlim(W*P(:,y)+ b);
        e=T(:,y)-a;
        W=W+e*P(:,y)';
        b=b+e;
    end
end
%obtener las fronteras;Puntos de cruce
Ph(1)=-b(1)/W(1,1);
Ph(2)=-b(1)/W(1,2);
Pv(1)=-b(2)/W(2,1);
Pv(2)=-b(2)/W(2,2);
%si se utiliza la forma general de recta para cada fila del vector n
x1=-3:0.1:3;x2=-3:0.1:3;n=length(x1); z=zeros(n);
n1=-W(1,1)/W(1,2)*x1- b(1)/W(1,2);
n2=-W(2,1)/W(2,2)*x1- b(2)/W(2,2);
%graficaciòn
hold on;grid on;
plot(P(1,:),P(2,:),'*'); title('datos y fronteras');
plot(x1,z,'r');plot(z,x2,'r');
plot(x1,n1,'g',x1,n2,'g');
```


[1] Jyh-Shing Roger Jang,“Neuro-Fuzzy and soft computing”, Prentice Hall,1997.
[2] Yesenia Gonzalez, “RNA01”, Julio, 2010.diapositivas 10-16.
[3]Hagan Martin, Neural Network design, city Publishing house, 1996.