ALGORITMO DE ENTRENAMIENTO PARA PERCEPTRÓN DE DOS ENTRADAS.

Juan Guillermo Patiño V
Universidad Pontificia Bolivariana, Cq. 1 #70-01, Medellín, Colombia,
juan.patino@pascualbravo.edu.co,
Tel. (+574) 4911033.

Resumen: *en este artículo se muestran los resultados obtenidos al utilizar un algoritmo de entrenamiento para un perceptrón de dos entradas con el fin de comprobar su eficiencia, tomando el toolbox de redes neuronales diseñado en Matlab. También se describen los aspectos más relevantes del funcionamiento del algoritmo. Copyright © 2011 ITPB.*

Abstract: *This paper presents the results obtained by using a training algorithm in a two-input perceptron, using the neural network toolbox designed in Matlab. Also, the most important algorithm's behaviour aspects are described.*

Keywords: *Neural Network, Two-Input Perceptron, Training Algorithm.*

1. INTRODUCCIÓN

En el siguiente artículo se presentan los resultados del uso de un algoritmo para el entrenamiento de un perceptrón de dos entradas. Para el diseño del algoritmo se utilizó el toolbox de redes neuronales desarrollado en Matlab utilizando el método de ponderación (Demuth Howard, 1998).

En la primera sección se muestran los aspectos generales que constituyen la base del algoritmo. En la segunda sección se explica en detalle el algoritmo y se presentan los resultados obtenidos a partir de la realización de diversas pruebas. Por último se analizan los resultados obtenidos y se hacen las respectivas conclusiones del trabajo desarrollado.

2. ASPECTOS GENERALES DEL PERCEPTRÓN

El perceptrón se define como una neurona artificial, la cual posee como función de evaluación una función de tipo umbral, razón por la cual la salida de esta sólo puede ser binaria, 0 o 1. (Del Brio,2005).

La idea de entrenar el perceptrón está basada en la modificación de los pesos de cada una de las entradas (Wp) con el objetivo de que la salida (Y) de la neurona arroje un conjunto de valores definidos por el usuario ante un juego de entradas determinado.

La estructura de un perceptrón de dos entradas se muestra en la figura 1.

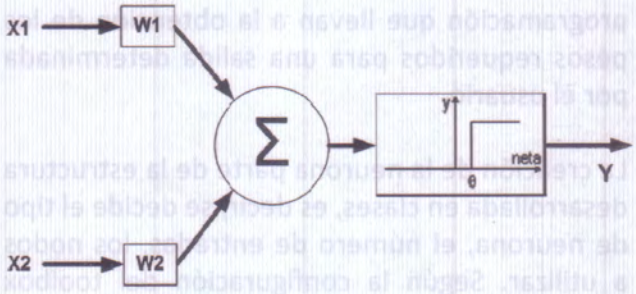

Fig. 1. Perceptrón de dos entradas.

El proceso básico del entrenamiento consiste en la comparación entre cada uno de los miembros de dos vectores, uno de ellos contiene la salida ideal del perceptrón y el otro contiene su salida real. Cuando dos miembros en una misma

posición en los dos vectores no son iguales se procede a sumar un valor específico a cada uno de los pesos del perceptrón. La forma de sumar los pesos depende de un factor que es ajustado por el usuario de la red. Entonces los nuevos pesos de la neurona quedan como sigue:

$$Wi = w + \alpha(tv - Y)X \qquad (1)$$

donde:
Wi, vector con los nuevos pesos
w, vector con los pesos anteriores
α, factor de pesos
tv, vector de entrenamiento
Y, vector de salidas
X, vector con el valor de las entradas

Cuando se ha logrado la igualdad del vector de entrenamiento con el vector de salidas, entonces se ha terminado la modificación de los pesos y por lo tanto el entrenamiento del perceptrón (Norgaard M, 2000).

3. ALGORITMO DE ENTRENAMIENTO Y RESULTADOS

El algoritmo de entrenamiento para el perceptrón usa como base el toolbox de algoritmos genéticos desarrollado (Betancur, 2007), el cual se anexa al final de este artículo. Teniendo en cuenta las bases teóricas del entrenamiento de un perceptrón se construyeron estructuras de programación que llevan a la obtención de los pesos requeridos para una salida determinada por el usuario.

La creación de la neurona parte de la estructura desarrollada en clases, es decir, se decide el tipo de neurona, el número de entradas, los nodos a utilizar. Según la configuración del toolbox desarrollado la neurona es de tipo 1 (umbral sigmoidal), sin embargo la función de evaluación que se estaba utilizando era la función signo, la cual arroja valores de -1 para un valor menor al umbral y 1 para un valor mayor al umbral (Norgaard M, 2000). Entonces ésta se modificó y se utilizó una función sigmoide, cuyos parámetros son el punto de inflexión, que en este caso se usa como valor umbral y el radio de curvatura, el cual se pone como un valor grande para que se forme una especie de punto discontinuo en la función. La función utilizada es

$$y = \frac{1}{1 + e^{-1x10^{29}(x-u)}}, \qquad (2)$$

donde:
x, entrada
u, umbral
y, salida

La parte de entrenamiento de los pesos se hace con un ciclo *while,* el cual itera hasta que la diferencia entre los vectores de salida y de entrenamiento sea cero. Dentro del ciclo *while*, hay un ciclo *for*, que evalúa cada una de las entradas para la neurona. Al final del *for* hay un condicional, que evalúa la igualdad elemento a elemento y si no son iguales, aplica la ecuación 1, para recalcular los pesos. Cuando termina el ciclo, el algoritmo arroja los nuevos pesos y el vector de salidas. Figura 2.

Se hicieron varias pruebas para el algoritmo de entrenamiento. El resumen de las pruebas se muestra en las siguientes tablas.

Tabla 1. Función Lógica OR

Umbral	Pesos	α	Iteraciones
2	2.0000 2.0000	0.01	161
5	5.0100 5.0100	0.01	387
2	2.0001 2.0001	0.0001	16002
5	5.0000 5.0000	0.0001	38501

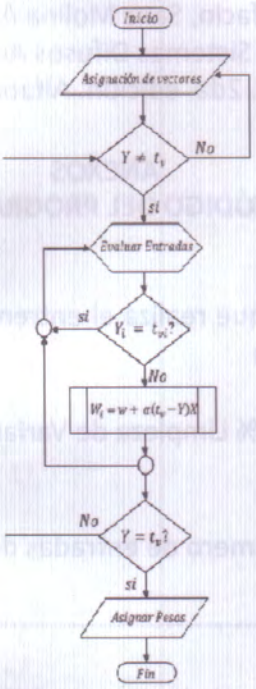

Fig. 2. Diagrama de flujo entrenamiento de pesos.

Tabla 2. Función Lógica AND

Umbral	Pesos	α	Iteraciones
2	1.1000 0.9000	0.01	101
5	2.6100 2.4100	0.01	252
2	1.1001 0.9001	0.0001	10002
5	2.6000 2.4000	0.0001	25001

Para las siguientes pruebas los vectores de entrenamiento utilizados fueron:

Tabla 3. Valores de entrenamiento pruebas

Prueba 1	Prueba 2
0	0
1	0
0	1
1	1

Para la prueba 1, se tiene:

Tabla 4. Prueba 1

Umbral	Pesos	α	Iteraciones
1.5	0.6000 1.5000	0.05	23
7	2.4500 7.0500	0.05	97
1.5	0.6000 1.5005	0.0005	2202
7	2.4335 7.0005	0.0005	9535

Para la prueba 2, se tiene:

Tabla 5. Prueba 2

Umbral	Pesos	α	Iteraciones
1.5	1.5100 0.4100	0.03	31
7	7.0000 2.2400	0.03	153
1.5	1.5001 0.4001	0.0003	3001
7	7.0000 2.2334	0.0003	15223

Para todos los ensayos los pesos por defecto fueron 0.1 y -0.1 (escogidos arbitrariamente) para cada entrada respectivamente.

4. ANÁLISIS DE LOS RESULTADOS

Los resultados obtenidos del algoritmo de entrenamiento del perceptrón fueron satisfactorios, ya que se lograron los resultados esperados y de forma rápida.

Cuando se utilizaron valores de α más pequeños se obtuvieron valores de los pesos con más cifras decimales. Situación que indica que la búsqueda se realiza de forma más exhaustiva.

También se nota que a valores de α más pequeños se realizan mayor número de iteraciones, debido a que el ajuste de los pesos se hace en pasos más pequeños.

El número de iteraciones aumenta significativamente cuando se aumenta el valor del umbral de la función, ya que se requieren más pasos para lograr la elección de los pesos adecuados.

Este número de iteraciones también depende del valor por defecto de los pesos, ya que si los pesos están muy alejados del valor que realmente deben tener, se deben hacer una mayor cantidad de cálculos para lograr su ajuste.

El algoritmo no fue capaz de encontrar los pesos de ciertas combinaciones, como las funciones lógicas XOR y XNOR, ya que estas no son linealmente separables (LIDI, 2007).

5. CONCLUSIONES

El algoritmo desarrollado es eficiente, ya que encuentra de manera rápida los pesos más adecuados para el perceptrón.

Por otro lado, el software desarrollado es muy útil ya que puede ser utilizado con cualquier valor umbral y valor de α.

Además, éste posee capacidad de ampliación, ya que puede ser empleado para entrenar perceptrones con mayor cantidad de entradas, aumentando las dimensiones de los vectores del algoritmo.

El software es capaz de realizar el entrenamiento del perceptrón para combinaciones en la salida que no fueron presentadas en este trabajo, las cuales deben ser linealmente separables.

REFERENCIAS

LIDI (2007). *Presentación de Neuronas Artificiales* En 02_Neurona Artificial.ppt, en línea [http://weblidi.info.unlp.edu.ar/catedras/neuronale
s.html] consultado en Diciembre de 2007.

Betancur, M.J. (2007). *Toolbox de Redes Neuronales. E*n toolbox_RN.zip, en línea [http://amasd.upb.edu.co/postautomatica/controll
nteligente.html] consultado en Diciembre de 2010.

Norgaard M, Ravn O. Neuronal Networks for Modelling and Control of Dinamic Systems: A Practitioner's Handbook. Springer-Verlag, New York LLC. 2000.

Demuth Howard, Beale Mark. Neural Network Toolbox for Use With MATLAB. The Math Works. (1998)

Del Brio Bonifacio, Sanz Molina Alfredo. Redes Neuronales y Sistemas Difusos Ampliada y Actualizada. 2da edición. Alfaomega. 2005.

ANEXOS
CÓDIGO DEL PROGRAMA

```
% Programa que realiza el entrenamiento de un
   Perceptrón

clear all       % Limpieza de Variables
close all
ini_RN(2)
% Máximo número de entradas de la neurona

%-------------------------------------------------------

% Inicialización de las características por defecto
   del perceptrón
typ  = 1;
% Tipo de neurona umbral, 1
numIn = 2;
 % Número de entradas del perceptrón
val0  = 0;
% valor inicial para el nodo de salida del
perceptrón
w0   = 0;
% Bias del perceptrón

%-------------------------------------------------------

% Definición de los valores para el perceptrón
Wi=[0.1
   -0.1];
 %Pesos iniciales del perceptrón
tv =[1 0 0 1];
 % Vector de entrenamiento (Contiene las salidas
ideales del perceptrón)
alfa=0.03;
 % Factor para la modificación de los pesos del
perceptrón
Xi=[0 0;0 1;1 0;1 1];
% Vector de combinación de entradas del
perceptrón
```

```
Y=0;
 %Valor por defecto de la salida del perceptrón

 %-------------------------------------------------------

 % Ciclo de Entrenamiento del perceptrón
 iteraciones=0;
 while norm(tv-Y)~=0,
% Se evalúa la igualdad de los vectores de salida
y de entrenamiento
for i=1:length(tv),
   X=Xi(i,:);
% Se asignan a las entradas la combinación
predeterminada
   w = Wi;
% Asignación de los pesos calculados a los pesos
iniciales del ciclo.
   inNod = [-1;-2];
 % Nodos de conexión a la entrada respectiva.

   ini_neu(1,typ,numIn,inNod,val0,w,w0)
   %Se inicia la creación del perceptrón
   Y(i) = eval_RN(X,1);
  %Se Asigna a Y el valor de evaluación del
perceptrón
   Wi=w;
%Se reasigna Wi el valor de los pesos actuales

   if (tv(i)-Y(i))==1;
%Se evalúa si son iguales las salidas miembro a
miembro
   Wi=w+alfa*(tv(i)-Y(i))*X';
%Si no son iguales, se hace el cambio en los
pesos
   end
  end
 iteraciones=iteraciones+1;
 end
iteraciones
w
Y
%
% eval_neu(k)  : Evaluar la neurona k-ésima
%
function eval_neu(kNeu)
global_RN

for kIn = 1:numInX,
% c/u de las entradas de la neurona k
     nodo = neu.inNod(kNeu,kIn);
 % lea de que nodo toma la señal
  if  nodo > 0,
% si ese número de nodo es positivo
     v = neu.val(nodo);
% asigne el valor de nodo a la entrada
   else
     if  nodo < 0,
 % pero si el número de nodo es negativo
        v = X_RN(-nodo);
% asigne a la entrada el valor externo
     else v = 0;
 % No hay entrada asignada, es nula
     end
   end
   inNeu(kIn) = v;
  % Este es el valor de la entrada
end

sp = ponderacion(kNeu,inNeu);
 % combinación lineal (suma ponderada)

% Función Umbral
 switch neu.typ(kNeu),
   case 1,
   y=sigmf(sp, [1e29 7]);
%Función umbral

 end

neu.val(kNeu) = y;
 % Escriba el valor del nodo de salida
```