|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Neural Networks**

**“Red Perceptrón Simple”**

Resumen

Perceptrón Simple en MATLAB para poder clasificar a lo sumo 4 clases distintas de datos por medio del método gráfico y de la regla de aprendizaje.

**Por:**

**Joel Mauricio Romero Gamarra**

Profesor:

MARCO ANTONIO MORENO ARMENDÁRIZ

Noviembre 2017

**Índice**

Contenido

[Introducción: 1](#_Toc476313175)

[Análisis Teórico: 1](#_Toc476313176)

[Software (librarias, paquetes, herramientas): 1](#_Toc476313177)

[Procedimiento: 1](#_Toc476313178)

[Resultados 1](#_Toc476313179)

[Discusión: 1](#_Toc476313180)

[Conclusiones: 1](#_Toc476313181)

[Referencias: 1](#_Toc476313182)

[Código 1](#_Toc476313183)

# Introducción:

El Perceptrón Simple es un tipo de red FeedForward con 1 sola capa (eso hace que sea perceptrón simple y no multicapa), cuenta con una función de activación Hardlims que se muestra en la Figura 1.1

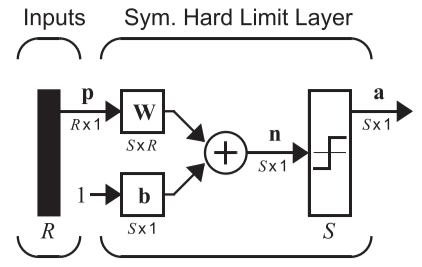


Figura 1. Arquitectura del Perceptrón Simple

Como se puede observar en la Figura 1, esta red cuenta con bias y al igual que todas las otras redes neuronales anteriores, las dimensiones están debajo de cada elemento, donde:

* a: Salida del Perceptrón Simple
* p: Vector de entrada
* W: Matriz de pesos
* b: Bias
* S: Número de neuronas
* R: Dimensión del vector de entrada

El Perceptrón Simple puede clasificar a los vectores de entrada en 2 clases, sin embargo, primero que nada, debemos conocer su modelo matemático descrito a continuación:

a = hardlims (W·p + b)

Podemos notar que el producto interno de la matriz de pesos con el vector de entrada es mayor o igual a -b, entonces la salida del perceptrón simple sería 1 (clase +1), en caso contrario, la salida sería -1 (clase -1).

Como podemos ver la función de activación hardlims es una función por partes donde solamente clasifica entre las clases +1 y -1, mostrada anteriormente.

# Software (librarías, paquetes, herramientas):

* MATLAB R2016a2
* Sublime Text 33
* Notepad ++4

# Procedimiento:

Como se puede observar en la arquitectura, primero debemos hacer el cálculo de la salida de la primera capa (FeedForward), ya que nos servirá para comenzar a hacer las iteraciones de la segunda capa. Para realizar el programa en MATLAB, se escribirá en 1 archivo de texto la matriz de pesos W y en otro archivo de texto el vector de entrada p.

Para el caso del Perceptrón Simple existen 2 métodos, el método gráfico y el método por regla de aprendizaje. Para el caso del método gráfico debemos tener algunos puntos que representarán a los vectores de entrada y un target (podemos no tenerlo), el algoritmo para el método gráfico es el siguiente:

* Graficar los vectores de entrada (los datos)
* Dibujar la “frontera de decisión”, que es una línea que separa a un conjunto de datos en 2 clases, cada frontera de decisión es capaz de separar en 2 clases, y cada neurona dibuja 1 sola frontera de decisión
* Identificar a la clase +1 (si es que tenemos un target)
* Dibujar una flecha apuntando hacia la clase identificada (clase +1), partiendo del origen y ortogonal a la frontera de decisión dibujada (a esta flecha la llamaremos matriz de pesos), la magnitud de dicha matriz no es importante

Posteriormente, debemos escoger un par de puntos, el primero es un punto **sobre** la matriz de pesos que dibujamos, el segundo, es un punto **sobre** la frontera de decisión que acabamos de trazar y ya que lo tenemos, debemos obtener el bias, para ello, ocupamos los valores propuestos de pesos (W) y de la frontera de decisión (p) con la siguiente fórmula:

b = -W·p

Ya que obtuvimos el bias, se realiza la propagación hacia delante de todos los datos con los valores obtenidos de W y b (usando cada dato como vector de entrada), y si se clasifican correctamente entonces los valores obtenidos son correctos y clasifican correctamente todos los datos.

En cuanto a la regla de aprendizaje del perceptrón simple, tenemos un nuevo concepto llamado “señal del error”, que nos ayudará a la clasificación, los siguientes elementos son los que se utilizarán para realizar el aprendizaje:

* Señal del error: e = t – a t es target y a es la salida de la red
* W (k + 1) = W (k) + e·p
* b (k + 1) = b (k) + e

Ahora, esta red al poseer un método de aprendizaje debe tener criterios de finalización, para evitar que un procesamiento infinito, los criterios son los siguientes:

* El valor de **it\_max** se alcanza (iteraciones máximas)
* Todos los datos son clasificados correctamente con los mismos valores de pesos y bias
* Eit < Error proporcionado por el usuario

Los valores pedidos al usuario son **it\_max** y **eit,** que son de ayuda para los criterios de finalización y poder obtener el aprendizaje correctamente de la red.

# Discusión:

En la sección donde presentamos los resultados de 3 matrices de pesos y vectores de entrada totalmente distintos, pudimos observar cómo se lleva a cabo las iteraciones de la red de Hamming hasta converger a una de las clases y poder clasificar correctamente el vector de entrada p.

# Conclusiones:

Como ya se había platicado, el manejo de archivos en MATLAB no es complicado, de hecho, es bastante similar al lenguaje C, utilizando funciones con una sintaxis muy similar, además, el hacer el uso de ellos pone a prueba un poco de la lógica de programación en MATLAB, ya que no es tan parecido en ciertas cosas, por ejemplo, multiplicar las matrices es demasiado simple.

# Referencias:

**[1]** “Capítulo 4. Perceptrón Simple”, class notes for Neural Networks, Department of Engineering in Computer Systems, Escuela Superior de Cómputo, 2017.

**[2]** Math Works, ‘MATLAB’, [Online]. Disponible en: <https://es.mathworks.com/products/matlab>.

**[3]** Sublime HQ, ‘Download’, [Online]. Disponible en:<https://www.sublimetext.com/3>

**[4]** Edgardo Adrián Franco Martínez, ‘Software de Programación GNU’ [Online]. Disponible en: <http://www.eafranco.com/?p=software/programacion/index.htm>

# Código

**Hamming.m**

%Limpiamos la pantalla y borramos todas las variables creadas anteriormente

clc;

clear;

%Recibimos el nombre del archivo como un String (por eso la 's')

archivo = input ('Introduce el nombre del archivo que contiene a W: ', 's');

%Abrimos el archivo que contiene la matriz a usar en modo lectura

archivo\_matriz = fopen (archivo, 'r');

%Recibimos el n�mero de filas y columnas que tiene la matriz para lectura

num\_filas = input ('\nIngresa el n�mero de patrones prototipo (filas) que tiene la matriz: ');

num\_col = input ('\nIngresa el n�mero de rasgos de cada patron (columas) que tiene la matriz: ');

%Creamos una matriz para el bias

bias = zeros (num\_filas,1);

bias = bias + num\_col;

**while** ~feof (archivo\_matriz)

[W, cont] = fscanf (archivo\_matriz, '%f', [num\_col num\_filas]);

**end**

W = W';

%Recibimos el nombre del archivo como un String (por eso la 's')

archivo = input ('\nIntroduce el nombre del archivo que contiene el vector de entrada: ', 's');

%Abrimos el archivo que contiene la matriz a usar en modo lectura

archivo\_matriz = fopen (archivo, 'r');

%Guardamos el vector de entrada p en una matriz

p = fscanf (archivo\_matriz, '%f');

%COMIENZA LA CAPA FEEDFORWARD (1 vez)

a = purelin((W \* p) + bias);

clc

%Calculamos un valor para epsilon aleatorio

epsilon = (rand() \* (1/(num\_filas - 1)) \* -1);

fprintf ('\n\nValor de epsilon: %d\n\n', epsilon);

%Calculamos la matriz de pesos a usar en la capa Recurrente

W = zeros (num\_filas);

W = W + epsilon;

**for** i = 1:num\_filas

**for** j = 1:num\_filas

**if** i == j

W (i, j) = 1;

**end**

**end**

**end**

%COMIENZA LA CAPA RECURRENTE (n veces)

flag = ones (num\_filas, 1);iteracion = 1;

%ESCRIBIR LA MATRIZ EN UN ARCHIVO DE TEXTO LLAMADO SalidaHamming

nuevo = fopen ('SalidaHamming.txt', 'w');

**for** i = 1:num\_filas

fprintf (nuevo, '%f\n', a(i, 1));

**end**

cont = 0;

**while** cont ~= 1

cont = 0;

p = a;

a = poslin (W \* p);

**if** iteracion == 1

aux = a;

**else**

%Agregamos un vector columna a la matriz aux

aux = [aux, a];

%Restamos los 2 vectores columna y lo guardamos en la bandera

flag = aux (:,iteracion) - aux (:, iteracion - 1);

**if** flag == zeros (num\_filas, 1)

**for** i = 1:num\_filas

clase = a (i, 1);

**if** clase ~= 0

cont = cont + 1;

**end**

**end**

**end**

**end**

%Escribimos los datos en un archivo

**for** i = 1:num\_filas

fprintf (nuevo, '%f\n', a(i, 1));

**end**

iteracion = iteracion + 1;

**end**

**for** i = 1:num\_filas

clase = a (i, 1);

**if** clase ~= 0

**break**;

**end**

**end**

fprintf ('\nLa RNA convergi� en la iteraci�n %d a la clase numero %d\n\n', iteracion - 1, i);

%GRAFICAMOS LOS VALORES DE SALIDA DE LA RED

nuevo = fopen ('SalidaHamming.txt', 'r');

Graph = figure('Name','Red Hamming','NumberTitle','off');

**while** ~feof (nuevo)

[r, cont] = fscanf (nuevo, '%f', [num\_filas 1]);

figure (Graph);

hold on;

plot (r, 'o-');

grid, ylabel('Rasgo'), xlabel('Clase');

**end**

%Cerramos los archivos

fclose (archivo\_matriz);

fclose (nuevo);

%Borramos todas las variables creadas durante la ejecuci�n del programa

clear;