|  |  |  |
| --- | --- | --- |
| **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: 2](#_Análisis_Teórico:)

[Software (librarias, paquetes, herramientas): 4](#_Toc476313177)

[Procedimiento: 5](#_Toc476313178)

[Resultados 6](#_Resultados:)

[Discusión: 15](#_Discusión:)

[Conclusiones: 16](#_Toc476313181)

[Referencias: 16](#_Toc476313182)

[Código 17](#_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 Hardlim que se muestra en la Figura 1.1

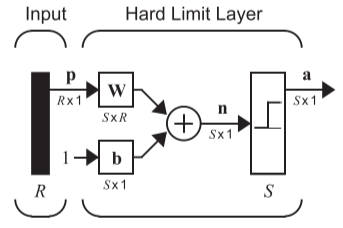


Figura . Arquitectura del Perceptrón Simple

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 = hardlim (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 hardlim es una función por partes donde solamente clasifica entre las clases +1 y 0, mostrada anteriormente.

# Análisis Teórico:

Los valores iniciales de la matriz de pesos se asignan de manera aleatoria al igual que el valor que tiene el bias, posteriormente ya que tenemos ambos valores (recordemos que los vectores de entrada p a clasificar son introducido por el usuario) procedemos a realizar el algoritmo de aprendizaje mostrado en la sección de introducción.

Para que quede un poco más claro se muestra un ejemplo a continuación:1

**it\_max**: 3 **eit = 0.1** Introducidos por el usuario

y

Ahora que tenemos todos los datos para poder comenzar el aprendizaje, procedemos a realizar la iteración número 1 propagando los datos hacia adelante (**p1 es naranja y p2 es manzana**).

**ITERACIÓN 1:**

Dato 1

Con k = 0; e = t – a = -1

W (1) = W (0) + e · p1 b (1) = b (0) + e

Dato 2

Con k = 1; e = t – a = 1

W (2) = W (1) + e · p2 b (2) = b (1) + e

Eit = = = 0

**ITERACIÓN 2:**

Dato 1

Con k = 0; e = t – a = -1

W (1) = W (0) + e · p1 b (1) = b (0) + e

Dato 2

Con k = 1; e = t – a = 0

W (2) = W (1) + e · p2 b (2) = b (1) + e

Eit = = = -

**ITERACIÓN 3:**

Dato 1

Con k = 0; e = t – a = 0

W (1) = W (0) + e · p1 b (1) = b (0) + e

Dato 2

Con k = 1; e = t – a = 0

Como ya alcanzamos la iteración máxima, vemos que los errores individuales por dato son 0 ambos, por lo tanto, los datos se clasificaron correctamente, así que, utilizamos el valor de pesos que sirvió para hacer la correcta clasificación en la siguiente fórmula:

-0.5P1 + 2P2 + 0.5P3 – 0.5 = 0

* Si P2 y P3 = 0  
    
  P1 = -1
* Si P1 y P3 = 0  
    
  P2 = 0.25
* Si P1 y P2 = 0  
    
  P3 = 1

Como se puede apreciar en el procedimiento de la regla de aprendizaje, es un poco extenso sin embargo bastante entendible y sencillo, sin embargo, puede ser que la RNA converja hasta la última iteración y el problema en hacer el programa, sería que el valor de iteraciones máximo sea muy grande ya que puede que no converja tan rápido.

Computacionalmente, hacer la programación del perceptrón por regla de aprendizaje es sencillo, sin embargo, el reto es hacer la programación del método gráfico ya que a simple vista es muy fácil ver dónde poner la frontera de decisión para que separe a las clases, pero el hacer que la computadora lo haga tiene un grado de dificultad, y aún más si son 4 clases o más.

# 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.

# Resultados:

Las Figuras 2 a 6 muestran el comportamiento del perceptrón simple para la compuerta AND con esta estructura del conjunto de entrenamiento en el archivo:

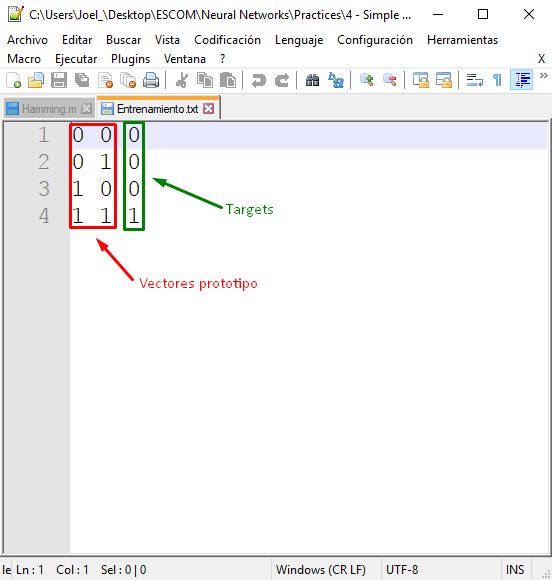


Figura . Conjunto de entrenamiento (Entrenamiento.txt)

Procedemos a ejecutar el programa en la Figura 3 y pedir datos iniciales al usuario:

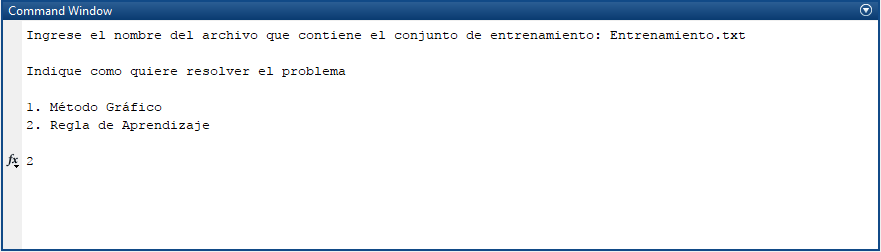


Figura . Valores iniciales pedidos al usuario (1)

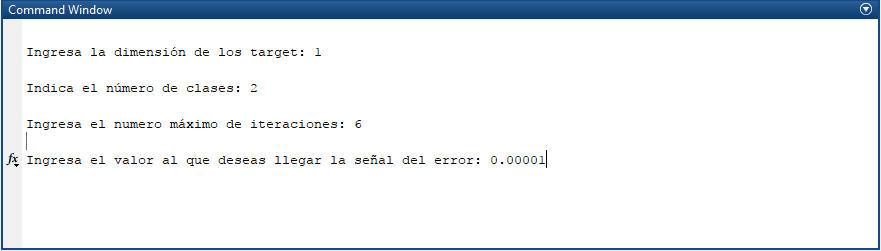


Figura . Valores iniciales pedidos al usuario (2)

A continuación, en la Figura 5 se muestra el mensaje que nos muestra el programa ya que se cumplió alguno de los criterios de finalización.

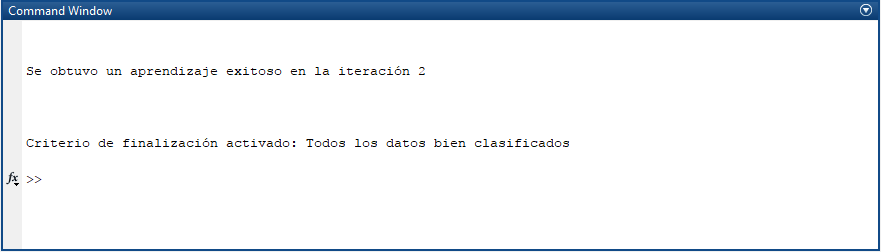


Figura . Mensaje final al usuario al terminar el aprendizaje

Procedemos a ver el resultado de los pesos y bias finales en el archivo nombrado ‘resultado\_hora\_fecha’, sin embargo, a la hora de poner el nombre no era posible poner el carácter “:” para las horas, por lo tanto, lo cambié por guiones medios.

Como podemos ver, el criterio de finalización activado fue que los datos estaban bien clasificados, y esto se da cuando todos los errores individuales son 0.

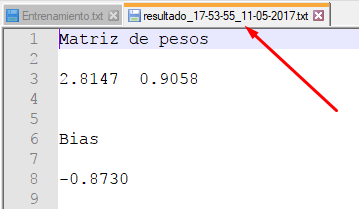


Figura . Archivo con pesos y bias que clasifican correctamente

Como podemos observar, la red perceptrón simple convergió correctamente en la segunda iteración y nos muestra en el mismo archivo los pesos y bias correctos para hacer la clasificación.

En las Figuras 7 a 11 se muestra otro ejemplo, ahora para la compuerta OR de 2 entradas, el conjunto de entrenamiento está descrito en la Figura 7:

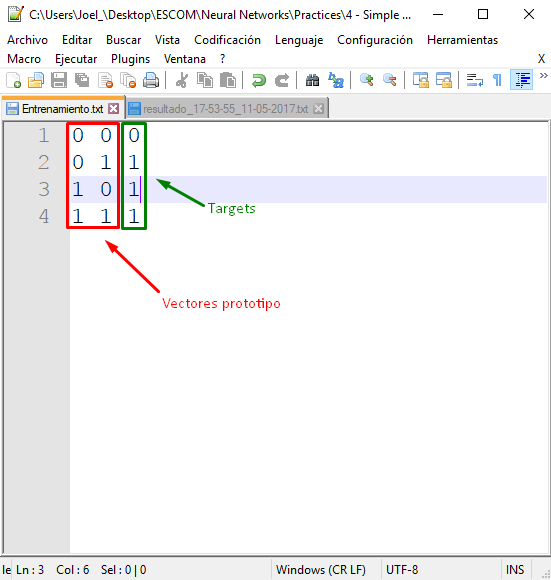


Figura . Conjunto de entrenamiento (Entrenamiento.txt)

Procedemos a ejecutar el programa en la Figura 8 y pedir datos iniciales al usuario:

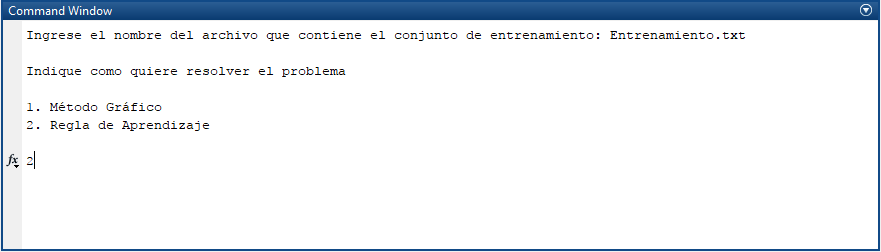


Figura . Valores iniciales pedidos al usuario (1)

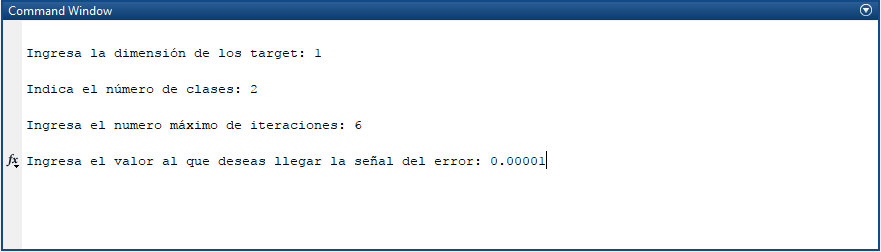


Figura . Valores iniciales pedidos al usuario (2)

A continuación, en la Figura 10 se muestra el mensaje que nos muestra el programa ya que se cumplió alguno de los criterios de finalización.

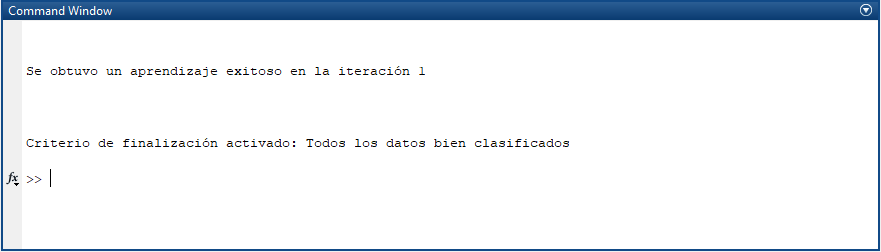


Figura . Mensaje final al usuario al terminar el aprendizaje

Como podemos ver, el criterio de finalización activado fue que los datos estaban bien clasificados, y esto se da cuando todos los errores individuales son 0.

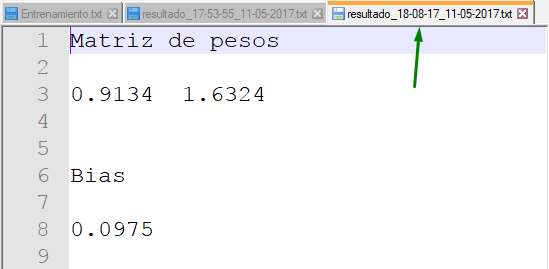


Figura . Archivo con pesos y bias que clasifican correctamente

Como podemos observar, la red perceptrón simple convergió correctamente en la primera iteración y nos muestra en el mismo archivo los pesos y bias correctos para hacer la clasificación.

En las Figuras 12 a 16 se muestra otro ejemplo, ahora para la compuerta XOR de 2 entradas, el conjunto de entrenamiento está descrito en la Figura 12:

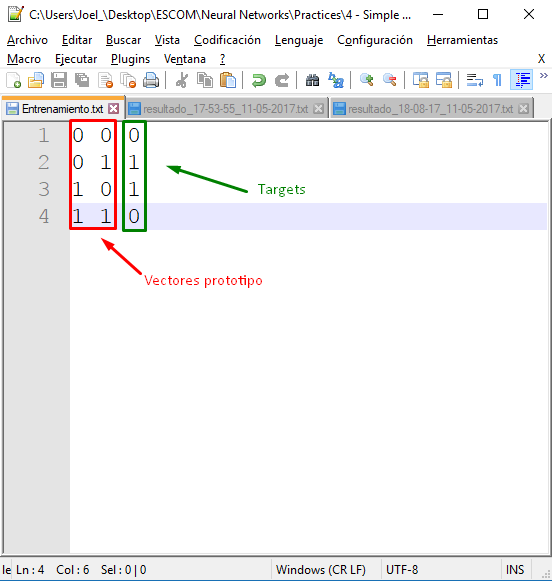


Figura . Conjunto de entrenamiento (Entrenamiento.txt)

Procedemos a ejecutar el programa en la Figura 13 y pedir datos iniciales al usuario:

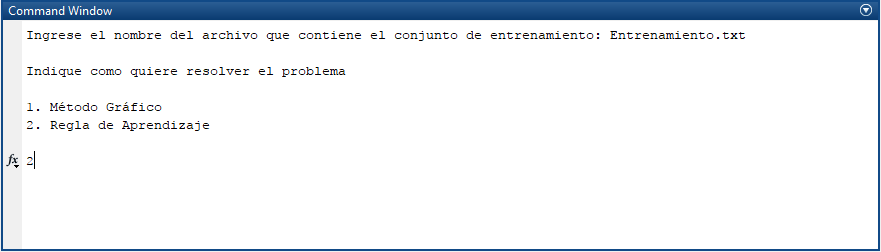


Figura . Valores iniciales pedidos al usuario (1)

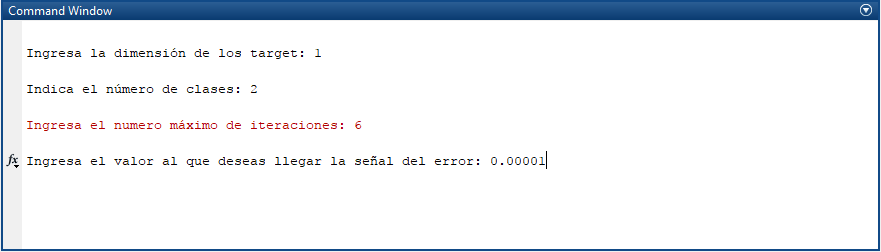


Figura . Valores iniciales pedidos al usuario (2)

A continuación, en la Figura 15 se muestra el mensaje que nos muestra el programa ya que se cumplió alguno de los criterios de finalización.

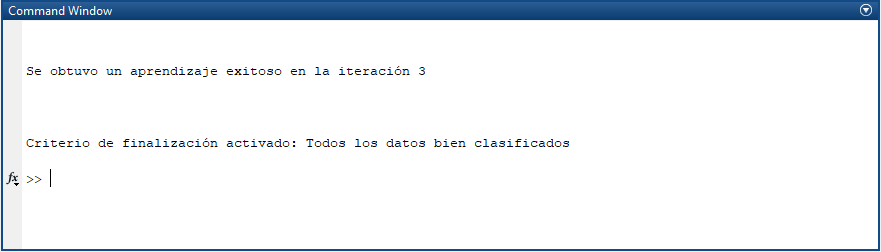


Figura . Mensaje final al usuario al terminar el aprendizaje

Como podemos ver, el criterio de finalización activado fue que los datos estaban bien clasificados, y esto se da cuando todos los errores individuales son 0.

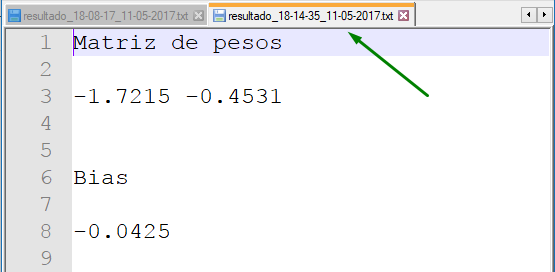


Figura . Archivo con pesos y bias que clasifican correctamente

A continuación, se muestra un ejemplo para target de 2 dimensiones, descrito en la Figura 17:

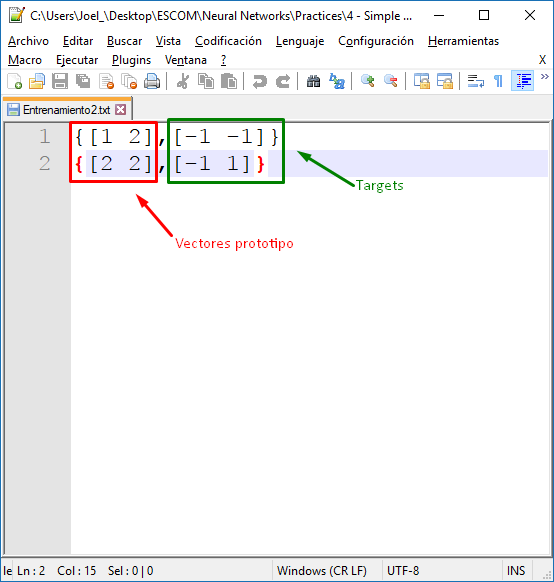


Figura . Conjunto de entrenamiento (Entrenamiento2.txt)

Procedemos a ejecutar el programa en la Figura 13 y pedir datos iniciales al usuario:

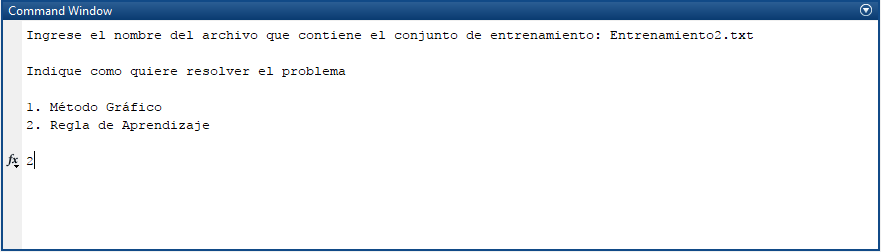


Figura . Valores iniciales pedidos al usuario (1)

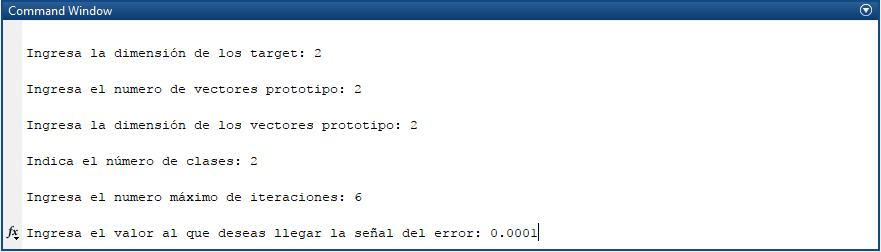


Figura . Valores iniciales pedidos al usuario (2)

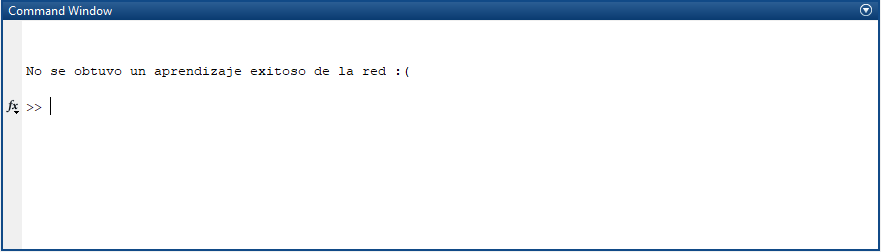


Figura . Mensaje final al usuario al terminar el aprendizaje

Como podemos ver no se obtuvo un aprendizaje exitoso, quiere decir que se llegó a las iteraciones máximas y no se cumplió ningún criterio de finalización, por lo tanto, no hay un archivo con pesos y bias finales ya que no se cumplió el aprendizaje.

A continuación, se muestra un ejemplo con una dimensión distinta:

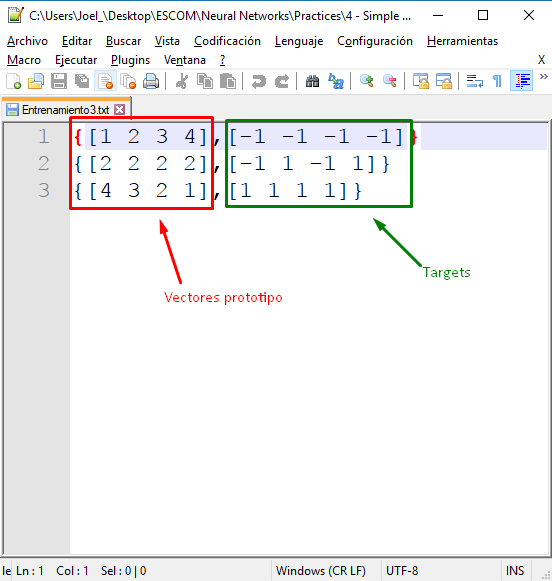


Figura . Conjunto de entrenamiento (Entrenamiento3.txt)

Procedemos a ejecutar el programa en la Figura 13 y pedir datos iniciales al usuario:

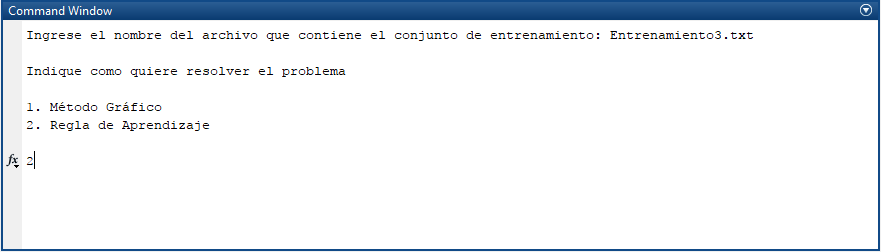


Figura . Valores iniciales pedidos al usuario (1)

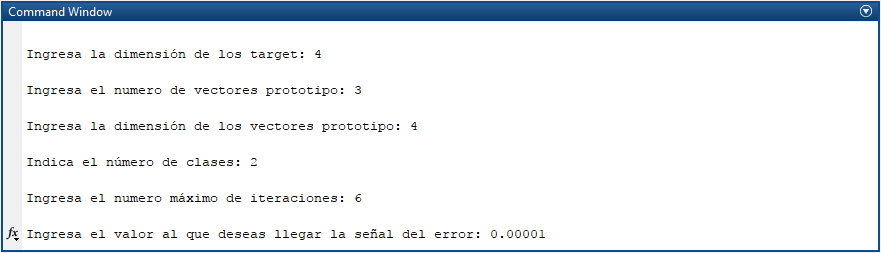


Figura . Valores iniciales pedidos al usuario (2)

A continuación, en la Figura 24 se muestra el mensaje que nos muestra el programa ya que se cumplió alguno de los criterios de finalización.

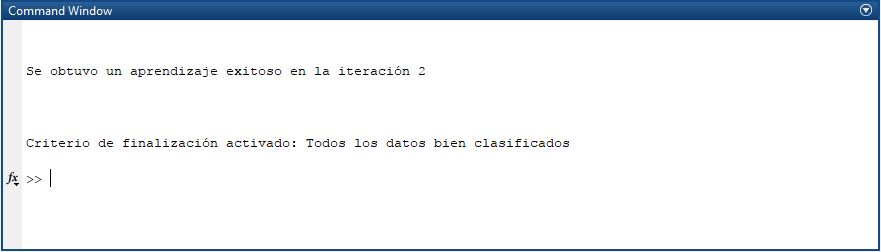


Figura . Mensaje final al usuario al terminar el aprendizaje

Como podemos ver, el criterio de finalización activado fue que los datos estaban bien clasificados, y esto se da cuando todos los errores individuales son 0.

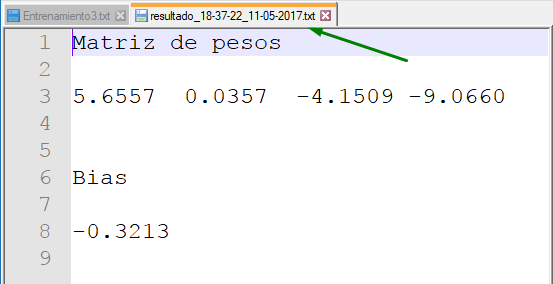


Figura . Archivo con pesos y bias que clasifican correctamente

# Discusión:

Desgraciadamente, no me fue posible realizar el programa con el método gráfico por falta de tiempo y se me hizo algo complicado, sin embargo, la regla de aprendizaje se cumplió correctamente para n dimensiones como se puede observar la evolución de los pesos, bias y la señal del error.

Como se puede ver en la sección anterior (la sección de resultados), la red aprendió correctamente y se observa como la señal del error se fue aproximando a 0 con cada iteración como era necesario para que se cumpliera un correcto aprendizaje.

La Red Perceptrón Simple tiene un gran potencial, sin embargo, la única restricción que tiene es que cada neurona puede dibujar únicamente 1 frontera de decisión, y a su vez cada frontera de decisión puede separar a los datos en 2 clases, por lo tanto, mientras más datos haya, se van a necesitar log2(#\_de\_Clases) neuronas.

En cuanto a la programación del perceptrón, fue un poco complicado realizar la lectura de los prototipos y targets cuando los targets tienen una dimensión mayor a 1, ya que en ciertas ocasiones estaba dando errores, sin embargo, al final quedó correcto y es por eso que en último ejemplo mostrado los targets tienen una dimensión de 4 al igual que los vectores prototipo y se clasificó correctamente.

Para la parte del método gráfico fue bastante complicado y aunque lo intenté, al final no quedó funcionando para todos los casos, solo para algunos, por esa razón opté por eliminar esa parte del código y dejar únicamente lo que funciona correctamente para cualquier dimensión (pidiendo algunos datos al usuario para hacer un poco más fácil la lectura de los datos).

# 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.

Los usos de algunas funciones parecen simples, sin embargo, tienen algunas restricciones un poco extrañas, por ejemplo, una muy útil a la hora de leer las matrices de un archivo de texto, fue dlmread, sin embargo, para la escritura de los pesos y bias finales lo sobrescribía (la función dlmwrite) así que tuve que hacer la impresión en el archivo dato por dato utilizando ciclos for (ya que ya conocemos las dimensiones de las matrices de pesos y bias).

Creo que MATLAB facilita bastante muchas cosas, sin embargo su sintaxis es un poco compleja en ciertas funciones y un poco difícil de entender al tener tantos parámetros posibles ya que tienen que tener ciertas cosas que lo hacen extraño, como ejemplo la función plot y/o subplot para poder ponerle un nombre distinto a cada subplot es un poco complicado, y además dibujarlo con un rango de valores específico a veces parece algo difícil ya que los vectores a graficar tienen que tener la misma longitud.

Esta práctica fue compleja sobre todo por la parte de hacer el método gráfico, ya que, visualmente una persona puede identificar de manera muy simple en donde colocar la frontera de decisión para poder separar las clases correctamente, y si ya es complicado hacerlo para 2 clases distintas, hacerlo para 4 es mucho más complejo.

# 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

**Perceptron.m**

%Comenzamos limpiando la pantalla y todas las variables

clearvars

clc

%Para guardar valores finales de pesos y bias

nombre\_arch = strcat ('resultado\_', datestr(now,'HH-MM-SS'), '\_', datestr (now, 'mm-dd-yyyy'), '.txt');

nombre\_archivo = input ('Ingrese el nombre del archivo que contiene el conjunto de entrenamiento: ', 's');

opcion = input ('\nIndique como quiere resolver el problema\n\n1. M�todo Gr�fico\n2. Regla de Aprendizaje\n\n');

clc

**if** opcion == 1

%AQUI VA EL METODO GR�FICO

**elseif** opcion == 2

%Pedimos la dimensi�n de los target al usuario

dim\_target = input ('\nIngresa la dimensi�n de los target: ');

**if** dim\_target > 1

num\_prototipos = input ('\nIngresa el numero de vectores prototipo: ');

R = input ('\nIngresa la dimensi�n de los vectores prototipo: ');

arch = fopen (nombre\_archivo, 'r');

prototipos = zeros (num\_prototipos, R);

targets = zeros (num\_prototipos, dim\_target);

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

fscanf (arch, '{[');

prototipos (i, :) = fscanf (arch, '%f');

fscanf (arch, '],[');

targets (i, :) = fscanf (arch, '%f');

fscanf (arch, ']}\n');

**end**

fclose (arch);

%Imprimimos los valores de los prototipos y targets

%prototipos

%targets

clases = input ('\nIndica el n�mero de clases: ');

%Calculamos el n�mero de neuronas a utilizar

S = ceil (log2 (clases));

**else**

%Abrimos el archivo y lo guardamos directamente en una matriz

conjunto\_entrenamiento = dlmread (nombre\_archivo);

aux = size (conjunto\_entrenamiento);

%La dimensi�n del vector de entrada (R) es el n�mero de columnas

%pero restando la columna que contiene a los target

R = aux (1, 2) - 1;

num\_prototipos = aux (1, 1);

clases = input ('\nIndica el n�mero de clases: ');

%Calculamos el n�mero de neuronas a utilizar

S = ceil (log2 (clases));

%Obtenemos los target y vectores prototipo en matrices distintas

prototipos = conjunto\_entrenamiento (:, 1: R);

targets = conjunto\_entrenamiento (:, R + 1);

**end**

%Pedimos los valores necesarios para los criterios de finalizaci�n

it\_max = input ('\nIngresa el numero m�ximo de iteraciones: ');

e\_it = input ('\nIngresa el valor al que deseas llegar la se�al del error: ');

%Limpiamos la pantalla

clc

%Se asignan valores aleatorios entre 0 y 1 a los pesos y bias

W = rand (S, R);

bias = rand (S, 1);

%Para escribir el error

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

%Ciclo que controla las iteraciones

**for** i = 1:it\_max

Eit = 0;

k = 0;

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

%p es igual a la fila j de los prototipos transpuesta

p = prototipos (j, :)';

%Calculamos la salida de la red para el prototipo j

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

%Calculamos la se�al del error para el prototipo j

signal\_error = targets (j, :) - a;

aux = size (signal\_error);

signal\_error = (sum(signal\_error) / aux (1, 2));

%Si el error es 0, la matriz de pesos y bias quedan igual

**if** signal\_error ~= 0

%Calculamos la nueva matriz de pesos para el siguiente

%vector prototipo

W = W + (signal\_error \* p');

%Calculamos el nuevo bias para el siguiente vector

%prototipo

bias = bias + signal\_error;

%Sumamos el error obtenido

Eit = Eit + ((1 / num\_prototipos) \* signal\_error);

**end**

k = k + 1;

**end**

fprintf (nuevo, '%f\n', Eit);

%Verificamos si se cumpli� alguno de los criterios de

%finalizaci�n, si no, se realiza otra iteraci�n

**if** Eit < e\_it && Eit > 0

fprintf ('\n\nSe obtuvo un aprendizaje exitoso en la iteraci�n %d\n\n', i);

fprintf ('\n\nCriterio de finalizaci�n activado: Eit < %f\n\n', e\_it);

i = -1;

**break**;

**elseif** Eit == 0

fprintf ('\n\nSe obtuvo un aprendizaje exitoso en la iteraci�n %d\n\n', i);

fprintf ('\n\nCriterio de finalizaci�n activado: Todos los datos bien clasificados\n\n');

i = -1;

**break**;

**end**

**end**

**if** i >= it\_max

fprintf ('\n\nNo se obtuvo un aprendizaje exitoso de la red :(\n\n');

**else**

pesos\_finales = fopen (nombre\_arch, 'w');

fprintf (pesos\_finales, 'Matriz de pesos\n\n');

**for** i = 1:S

**for** j = 1:R

fprintf (pesos\_finales, '%.4f\t', W (i, j));

**end**

fprintf (pesos\_finales, '\n');

**end**

fprintf (pesos\_finales, '\n\nBias\n\n');

**for** i = 1:S

fprintf (pesos\_finales, '%.4f\n', bias (i, 1));

**end**

fclose (pesos\_finales);

**end**

**else**

fprintf ('\n\nOPCION INVALIDA\n\n');

**end**

clearvars