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

**Neural Networks**

**“Red Adaline”**

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: 1](#_Discusión:)

[Conclusiones: 1](#_Toc476313181)

[Referencias: 1](#_Toc476313182)

[Código 1](#_Toc476313183)

# Introducción:

La Red Adaline es una RNA algo parecida al perceptrón simple, sin embargo, para separar a los datos de entrada en más de 2 clases con un perceptrón simple, se deben usar más neuronas ya que cada neurona dibuja una sola frontera de decisión.1

Para evitar estos problemas, se propuso una RNA conocida como **adaptive linear neuron**, (ADALINE), el proceso de aprendizaje de esta red es gracias a una técnica llamada “Regla Delta” actualizando los valores de la matriz de pesos y el bias automáticamente al calcular las derivadas parciales de cada matriz de pesos y cada bias con respecto a la señal del error de la RNA. En la Figura 1, se muestra la arquitectura de la red Adaline.

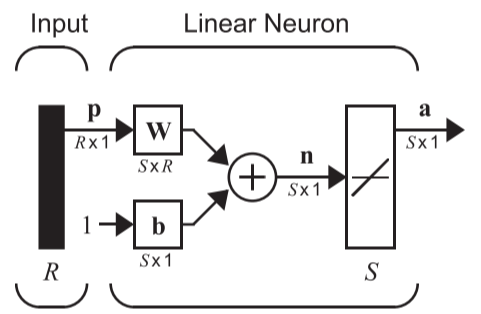


Figura . Arquitectura de la red Adaline

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 de la red Adaline
* p: Vector de entrada
* W: Matriz de pesos
* b: Bias
* S: Número de neuronas
* R: Dimensión del vector de entrada

La Red Adaline puede clasificar a los vectores de entrada en más de 2 clases, sin embargo, primero que nada, debemos conocer su modelo matemático descrito a continuación:

a = purelin (W·p + b)

El diseño de la regla de aprendizje (regla delta, mencionada anteriormente) consiste en diseñar el elemento ΔWi.

# Análisis Teórico:

El elemento Wi se calcula de la siguiente forma:

ΔWi = α

Donde la derivada nos indica la dirección en la cual se va a obtener un valor más pequeño de la señal del error, donde α se le conoce factor de aprendizaje donde comúnmente toma valores entre 0 y 1.

El valor de α es muy importante ya que permite regular el tamaño de incremento o decremento que se aplicará a cada peso sináptico, evitando que los valores de pesos y bias que se van actualizando oscilen demasiado durante el aprendizaje.1

Después de calcular las derivadas parciales, las ecuaciones finales para calcular el valor de pesos y bias son las siguientes:

Los valores introducidos por el usuario son los siguientes:

* It\_max: Número máximo de iteraciones a llevar a cabo para que no se cicle el programa
* : Valor pequeño al cual el usuario quiere que se aproxime la señal del error
* : Es el factor de aprendizaje

El algoritmo para llevar a cabo la regla de aprendizaje es el siguiente:

1. Inicializar los valores de pesos y bias con valores aleatorios entre 0 y 1 (A estos valores de inicialización se le llaman condiciones iniciales)
2. Se lleva a cabo la propagación hacia delante de cada uno de los datos de entrada (el conjunto de entrenamiento) y para cada uno de estos datos se calcula su señal del error y se aplica la regla de aprendizaje correspondiente.
3. Una vez terminada una iteración (Se llama iteración cuando todos los datos ya pasaron fueron propagados hacia adelante) se calcula el **error de iteración** de la siguiente forma:

Donde:

* ej: Es el error individual de cada uno de los datos
* N: Es el número de datos en el conjunto de entrenamiento

Posteriormente, se verifica si se cumplió alguno de los siguientes criterios de finalización, y si no, se realiza otra iteración hasta que alguno de los siguientes criterios se cumpla:

* Que todos los elementos del conjunto de entrenamiento estén correctamente clasificados, es decir, que Eut = 0
* Que se llegue al valor de it\_max, es decir que las iteraciones máximas se cumplan (Este criterio de finalización no garantiza el aprendizaje)
* Que el valor de Eut < eit que es el valor introducido por el usuario (Garantiza aproximadamente 2/3 de un aprendizaje exitoso)

# Software (librarías, paquetes, herramientas):

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

# Procedimiento:

Para hacer la programación de la red Adaline, debemos comenzar a ver el modelo matemático expuesto en la sección de introducción.

Primero, debemos pedir al usuario los valores que tiene que introducir para poder tener los criterios de finalización bien definidos y que no se cicle el programa (muy importante).

Después, comenzamos asignando valores aleatorios con una función rand que nos da valores entre 0 y 1, para darle valores a la matriz de pesos W y a l bias a utilizar. Para resolver el problema del codificador de binario a decimal, haremos uso de una función para convertir de decimal a binario (dentro de un ciclo for, para hacerlo dinámico a cualquier valor), y guardarlo dentro de una matriz (sería el vector de entrada p) y posteriormente hacer la multiplicación de matrices con el vector de pesos calculado en la primera iteración de manera aleatoria.

Para no estar ocupando espacio en la memoria (debido a que no sabemos de cuantos bits quiera el usuario el decodificador), escribiremos cada uno de los datos en un archivo y lo estaremos abriendo y cerrando, para posteriormente hacer la gráfica de la evolución tanto de la matriz de pesos (que va cambiando junto con el bias debido a la regla de aprendizaje), y la gráfica de la evolución de la señal del error.

NOTA: Esta red puede trabajar con o sin bias, para este programa al usuario se le da la opción de manejarla como quiera.

En la siguiente sección se presentan los resultados obtenidos por el programa para el problema del decodificador y algún otro conjunto de entrenamiento dado por el usuario mediante un archivo de texto.

# Resultados:

Comenzamos pidiendo algunos datos al usuario para saber si desea utilizar la red con bias o sin bias, sin embargo, siempre se necesitan los criterios de finalización por lo que se piden al principio, en las Figuras 2 a 6 se muestra el ejemplo de la red Adaline sin bias (es decir, resolviendo el problema del codificador) para 2 bits.

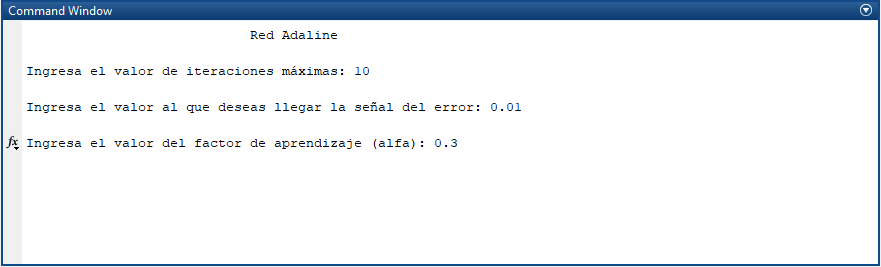


Figura . Valores iniciales pedidos al usuario (1)

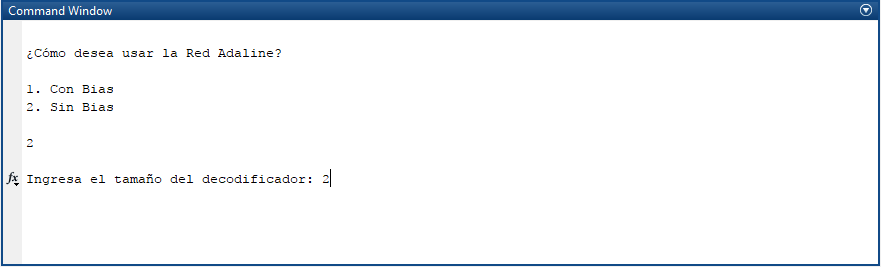


Figura . Valores iniciales pedidos al usuario (2)

A continuación, se muestra el mensaje final para el usuario, indicando en que iteración se obtuvo un aprendizaje exitoso.

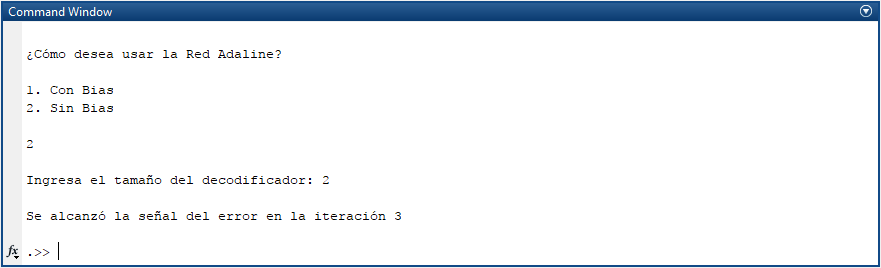


Figura . Mensaje final al usuario

Procedemos a ver las gráficas de la evolución de la RNA y el archivo donde se guardó el valor final de los pesos para realizar la correcta clasificación.

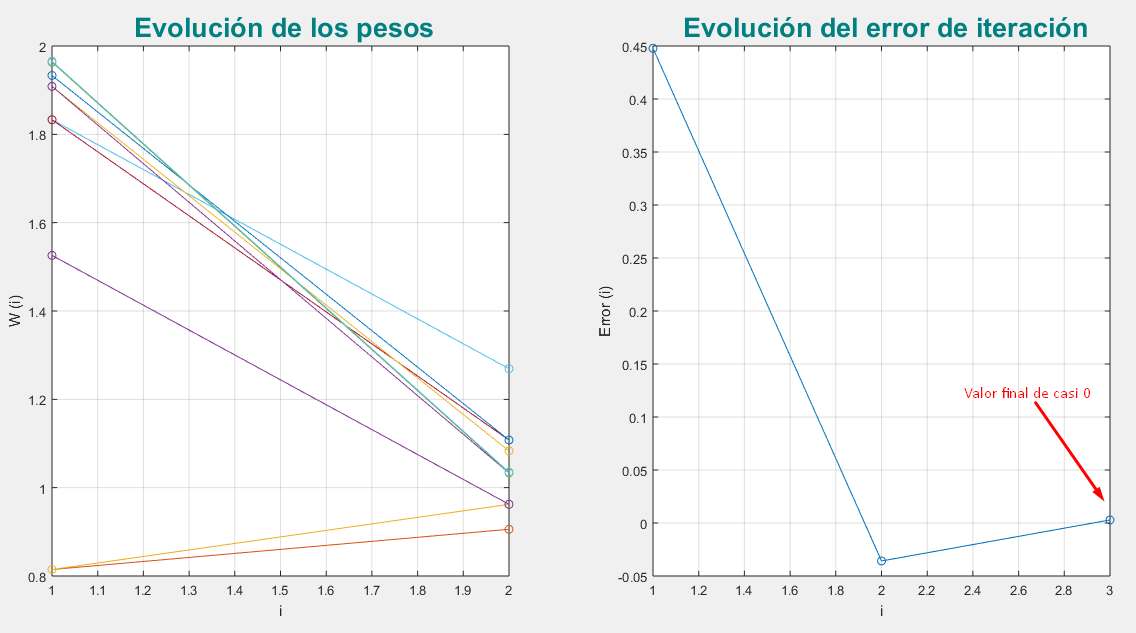


Figura . Gráficas para decodificador de 2 bits

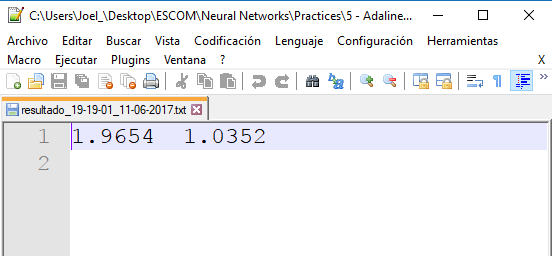


Figura . Valor final de los pesos para clasificar correctamente

Como podemos ver, las gráficas y los pesos están correctos, sin embargo, este problema es algo fácil, así que procedemos a aumentar el número de bits del decodificador a 4 (se muestra en las Figuras 7 a 11).

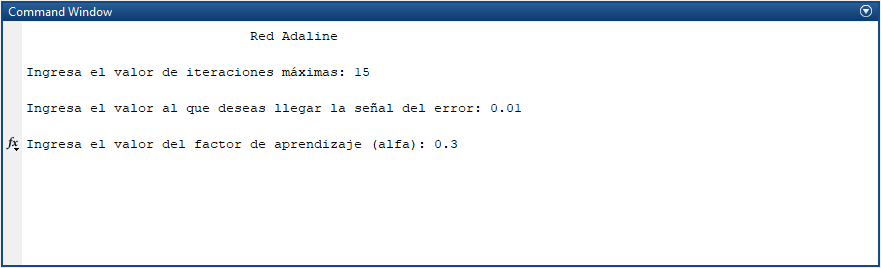


Figura . Valores iniciales pedidos al usuario (1)

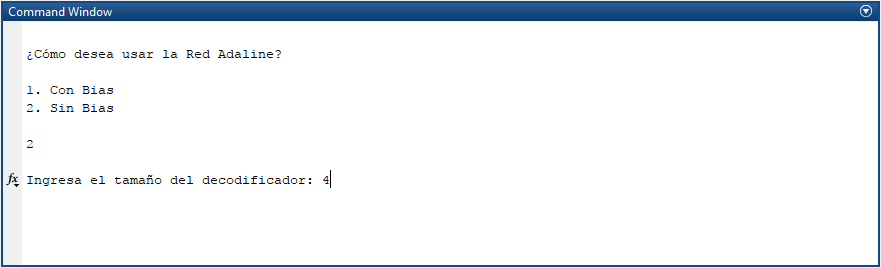


Figura . Valores iniciales pedidos al usuario (2)

A continuación, se muestra un mensaje final al usuario indicando si se pudo realizar o no el correcto aprendizaje de la red, si fue exitoso, mostrando en que iteración.

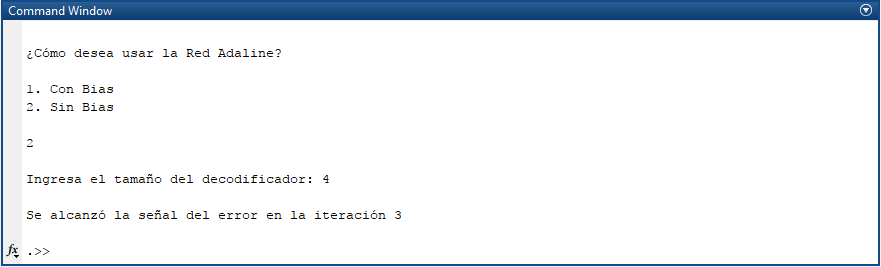


Figura . Mensaje final al usuario

Procedemos a ver las gráficas de la evolución de la RNA y el archivo donde se guardó el valor final de los pesos para realizar la correcta clasificación.

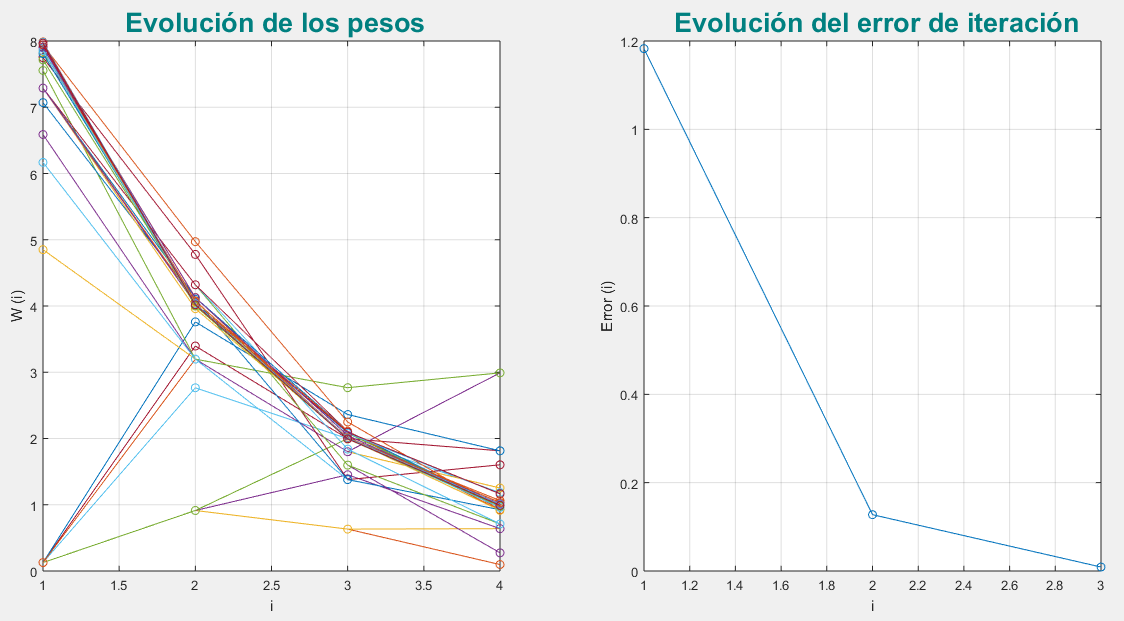


Figura . Gráficas para decodificador de 4 bits

Ahora nos vamos al archivo de texto en donde se guardó el valor final de los pesos que clasifican correctamente a todos los datos.

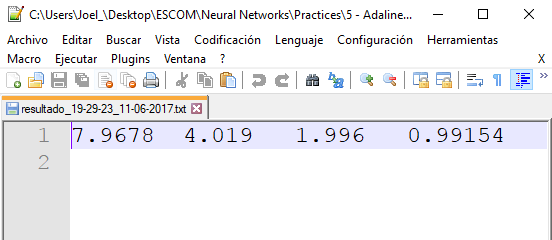


Figura . Valor final de los pesos para clasificar correctamente

Podemos observar que en esta ocasión el vector de pesos tiene 4 elementos, y esto es correcto ya que el número de bits que escogimos para el decodificador es 4, por lo tanto, tiene que ser un vector de 4 elementos.

A continuación, se muestra el último ejemplo del decodificador, pero ahora para 6 bits, en las figuras 12 a 16.

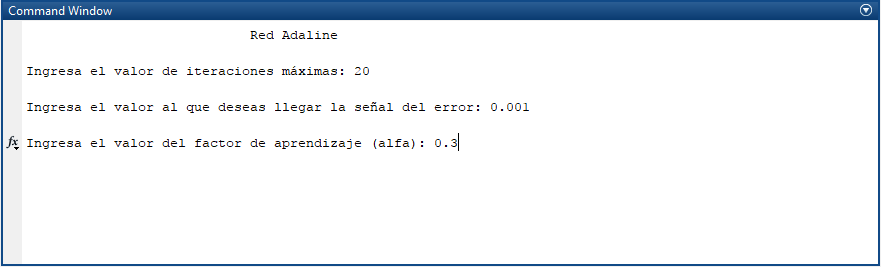


Figura . Valores iniciales pedidos al usuario (1)

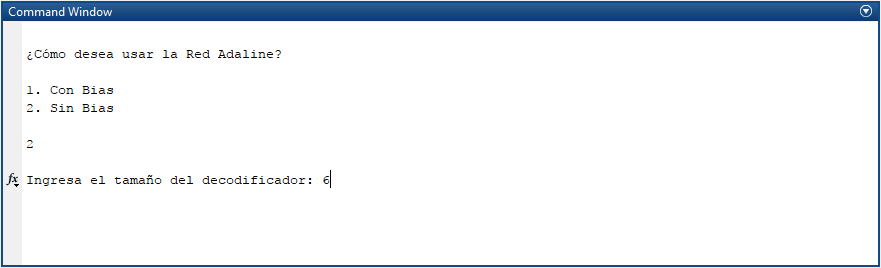


Figura . Valores iniciales pedidos al usuario (2)

Ahora, procedemos a mostrar el mensaje que se envía al usuario para saber si se logró o no un aprendizaje exitoso y en caso de que sí, en que iteración ocurrió.

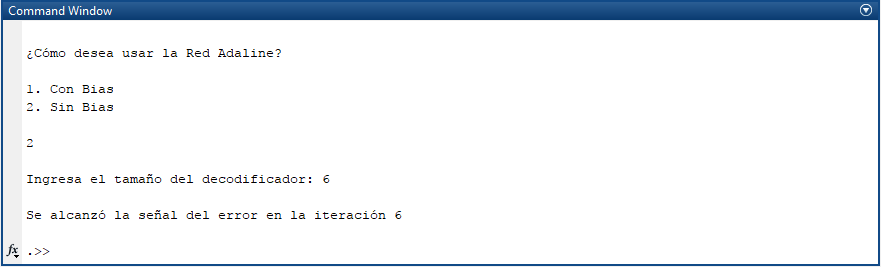


Figura . Mensaje final al usuario

Ahora, procedemos a ver las gráficas que se generaron.

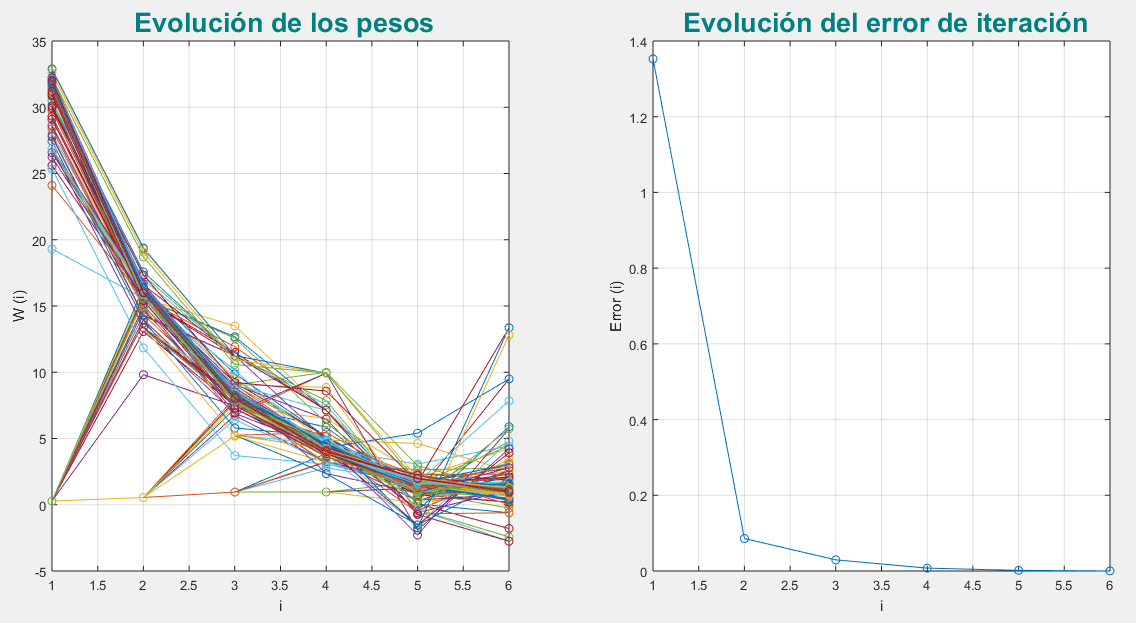


Figura . Gráficas para decodificador de 6 bits

Como se puede observar en la Figura 15, la matriz de pesos tuvo distintos cambios hasta llegar a un valor más o menos estable (donde se encuentran casi todas las líneas), sin embargo, no se puede apreciar tan bien debido a que en cada iteración se hacen 2n actualizaciones en la matriz de pesos.

Por lo tanto, procedemos a abrir el archivo que contiene el valor de los pesos finales y saber cuáles nos pueden servir para hacer la correcta clasificación.

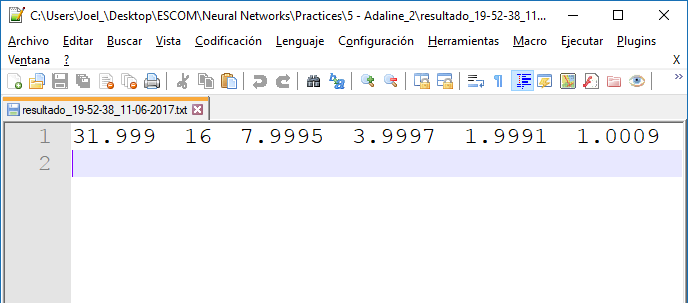


Figura . Valor final de los pesos para clasificar correctamente

Como podemos ver, en esta ocasión 2 de los elementos de la matriz de pesos crecieron demasiado (los primeros 2 elementos), debido a que el número más grande que debemos formar con 6 bits es el número 63.

# 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 10. Red Adaline”, 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

**Adaline.m**

%Comenzamos limpiando la pantalla y todas las variables

clearvars

clc

%Pedimos los valores para los criterios de finalizaci�n

fprintf ('\t\t\t\t\t\t\tRed Adaline\n');

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

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

alpha = input ('\nIngresa el valor del factor de aprendizaje (alfa): ');

%Preguntamos si quiere usar la red con o sin bias

clc

opcion = input ('\n�C�mo desea usar la Red Adaline?\n\n1. Con Bias\n2. Sin Bias\n\n');

**if** opcion == 1

%C�digo para perceptron simple

**elseif** opcion == 2

numero\_bits = input ('\nIngresa el tama�o del decodificador: ');

numero\_datos = power (2, numero\_bits);

%Generamos el conjunto de entrenamiento

entrenamiento = genera\_conjunto (numero\_bits, numero\_datos);

%Abrimos archivos para escribir datos de los pesos

pesos = fopen ('Pesos.txt', 'w');

%Creamos un figure para hacer las gr�ficas

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

rango = 1:numero\_bits;

%Asignamos valores aleatorios a los pesos

W = rand (1, numero\_bits);

**for** j = 1:numero\_bits

%Escribimos los pesos en un archivo

fprintf (pesos, '%.4f\n', W (1, j));

**end**

Error\_iteracion = zeros (1, it\_max);

flag = 1;

%Regla de aprendizaje

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

%El error se hace 0 en cada iteraci�n nueva

error = 0;

**for** i = 1:numero\_datos

%Obtenemos cada uno de los datos

p = (entrenamiento (i, 1:numero\_bits))';

a = purelin (W \* p);

error = ((entrenamiento (i, (numero\_bits + 1))) - a);

%Actualizamos el valor de los pesos

W = (W + (2 \* alpha \* error \* p'));

**for** j = 1:numero\_bits

%Escribimos los pesos en un archivo

fprintf (pesos, '%.4f\n', W (1, j));

**end**

**if** error ~= 0

%Agregamos el valor del error al error de iteraci�n

Error\_iteracion (1, k) = Error\_iteracion (1, k) + error;

flag = 0;

**end**

**end**

Error\_iteracion (1, k) = (Error\_iteracion (1, k) / numero\_datos);

%Verificamos si se cumpli� alguno de los criterios de finalizaci�n

**if** flag == 1

fprintf ('\nTodos los datos fueron bien clasificados en la iteraci�n %d.\n\n', k);

**break**;

**elseif** Error\_iteracion (1, k) < e\_it && Error\_iteracion (1, k) > 0

fprintf ('\nSe alcanz� la se�al del error en la iteraci�n %d\n\n.', k);

**break**;

**end**

**end**

%Cerramos el archivo

fclose (pesos);

**if** k >= it\_max

fprintf ('No hubo un aprendizaje exitoso :(\n\n');

%Graficamos el error de iteraci�n

subplot (1, 2, 2), plot (1:it\_max, Error\_iteracion, 'o-');

grid, ylabel('Error (i)'), xlabel('i');

title ('\fontsize{20} \color[rgb]{0 .5 .5}Evoluci�n del error de iteraci�n');

**else**

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

dlmwrite (nombre\_arch, W, 'delimiter', '\t');

%Graficamos el error de iteraci�n

subplot (1, 2, 2), plot (1:k, Error\_iteracion (1, 1:k), 'o-');

grid, ylabel('Error (i)'), xlabel('i');

title ('\fontsize{20} \color[rgb]{0 .5 .5}Evoluci�n del error de iteraci�n');

**end**

pesos = fopen ('Pesos.txt', 'r');

frewind (pesos);

**while** ~feof (pesos)

W = fscanf (pesos, '%f\n', [1 numero\_bits]);

%Graficamos los pesos

hold on;

subplot (1, 2, 1), plot (rango, W, 'o-');

grid, ylabel('W (i)'), xlabel('i');

title ('\fontsize{20} \color[rgb]{0 .5 .5}Evoluci�n de los pesos');

**end**

**else**

fprintf ('\nOpci�n inv�lida.\n\n');

**end**

%Limpiamos todas las variables utilizadas

clearvars

**genera\_conjunto.m**

**function** conjunto = genera\_conjunto (numero\_bits, numero\_datos)

conjunto = zeros (numero\_datos, numero\_bits + 1);

**for** i = 0:(numero\_datos - 1)

%Convertimos cada numero en un numero binario de n bits

binario = dec2bin (i, numero\_bits);

**for** j = 1:numero\_bits

%Agregamos el numero binario a cada fila

conjunto ((i + 1), j) = str2num (binario (:, j));

**end**

%En la �ltima columna, ponemos el valor decimal

conjunto ((i + 1), (numero\_bits + 1)) = i;

**end**

**end**