



**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
Análisis Teórico: .....	2
Software (librerías, paquetes, herramientas): .....	4
Procedimiento: .....	5
Resultados .....	6
Discusión: .....	10
Conclusiones:.....	10
Referencias: .....	10
Código .....	11

## 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.<sup>1</sup>

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.

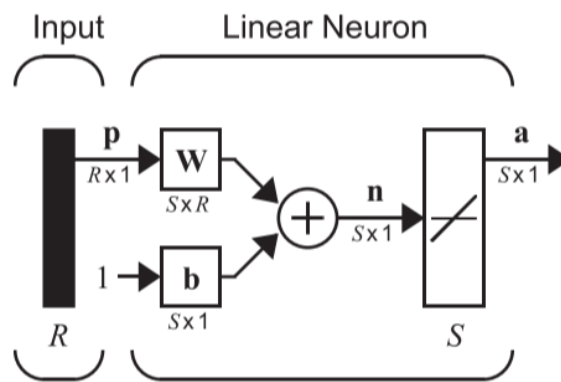


Figura 1. Arquitectura de la red Adaline

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:

$$a = \text{purelin}(W \cdot p + b)$$

El diseño de la regla de aprendizaje (regla delta, mencionada anteriormente) consiste en diseñar el elemento  $\Delta W_i$ .

## Análisis Teórico:

El elemento  $W_i$  se calcula de la siguiente forma:

$$\Delta W_i = \alpha \frac{de}{dW_i}$$

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

El valor de  $\alpha$  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.<sup>1</sup>

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

$$W_i(k+1) = W_i(k) + 2 \cdot \alpha \cdot (t - a) \cdot p$$

$$b_i(k+1) = b_i(k) + 2 \cdot \alpha \cdot (t - a)$$

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
- $e_{it}$ : Valor pequeño al cual el usuario quiere que se aproxime la señal del error
- $\alpha$ : 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:

$$E_{it} = \frac{1}{N} \sum_{j=1}^N e_j$$

Donde:

- $e_j$ : 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  $E_{ut} = 0$
- 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  $E_{ut} < e_{it}$  que es el valor introducido por el usuario (Garantiza aproximadamente 2/3 de un aprendizaje exitoso)

#### **Software (librerías, paquetes, herramientas):**

- MATLAB R2016a<sup>2</sup>
- Sublime Text 3<sup>3</sup>
- Notepad ++<sup>4</sup>

#### **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 la bias  $a$  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.

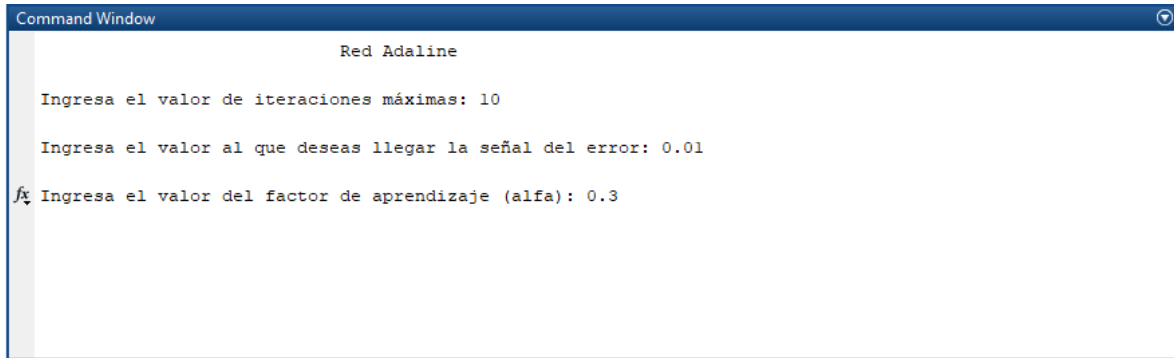
Para no estar ocupando espacio en la memoria (debido a que no sabemos de cuantos bits quiera el usuario el decodificador), escribiremos cada uno de los datos en un archivo y lo estaremos abriendo y cerrando, para posteriormente hacer la gráfica de la evolución tanto de la matriz de pesos (que va cambiando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



```
Command Window

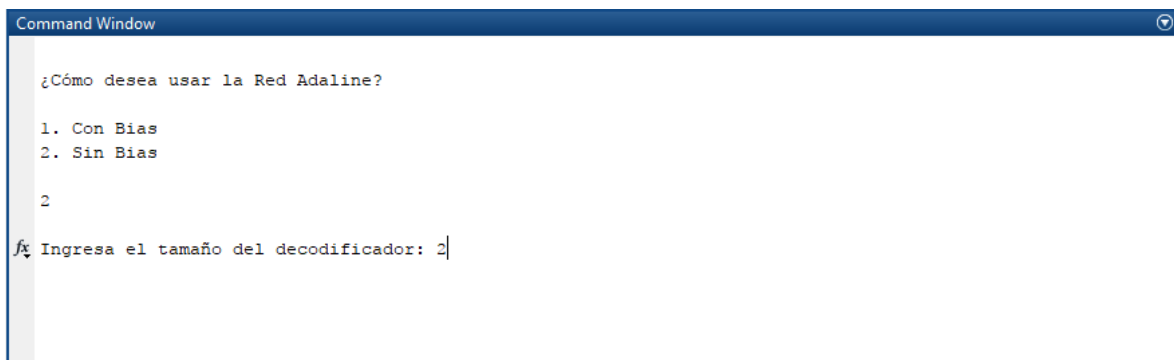
Red Adaline

Ingresa el valor de iteraciones máximas: 10

Ingresa el valor al que deseas llegar la señal del error: 0.01

fx Ingresa el valor del factor de aprendizaje (alfa): 0.3
```

Figura 2. Valores iniciales pedidos al usuario (1)



```
Command Window

¿Cómo desea usar la Red Adaline?

1. Con Bias
2. Sin Bias

2

fx Ingresa el tamaño del decodificador: 2
```

Figura 3. 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.



```
Command Window

¿Cómo desea usar la Red Adaline?

1. Con Bias
2. Sin Bias

2

Ingresa el tamaño del decodificador: 2

Se alcanzó la señal del error en la iteración 3

fx .>> |
```

Figura 4. 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.

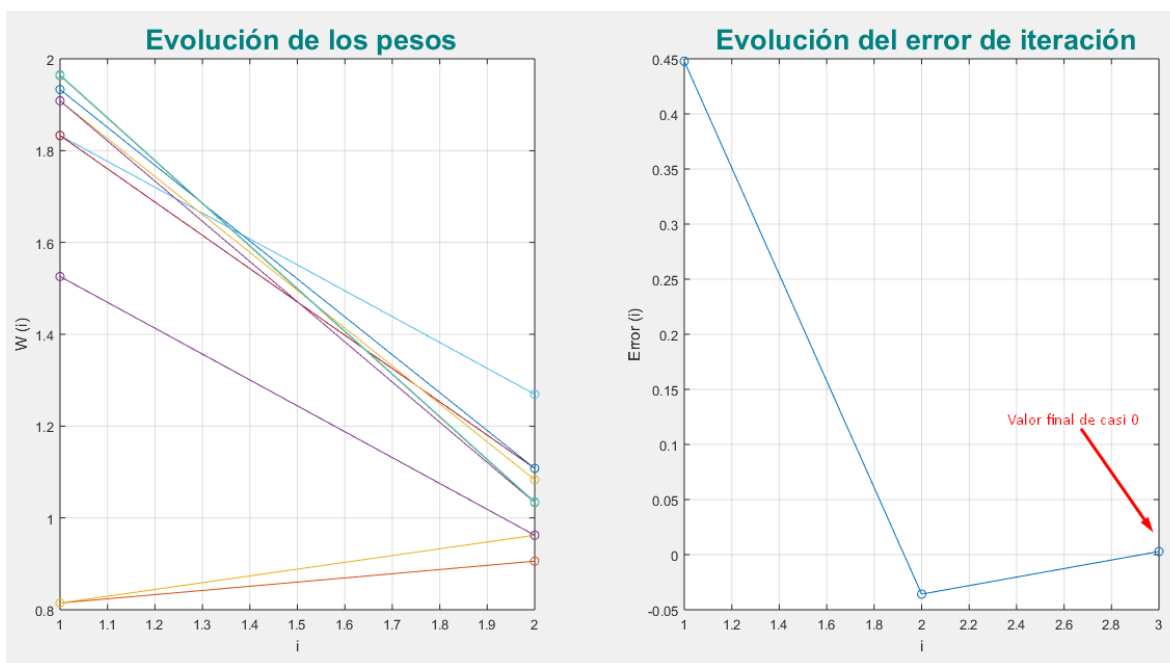


Figura 5. Gráficas para decodificador de 2 bits

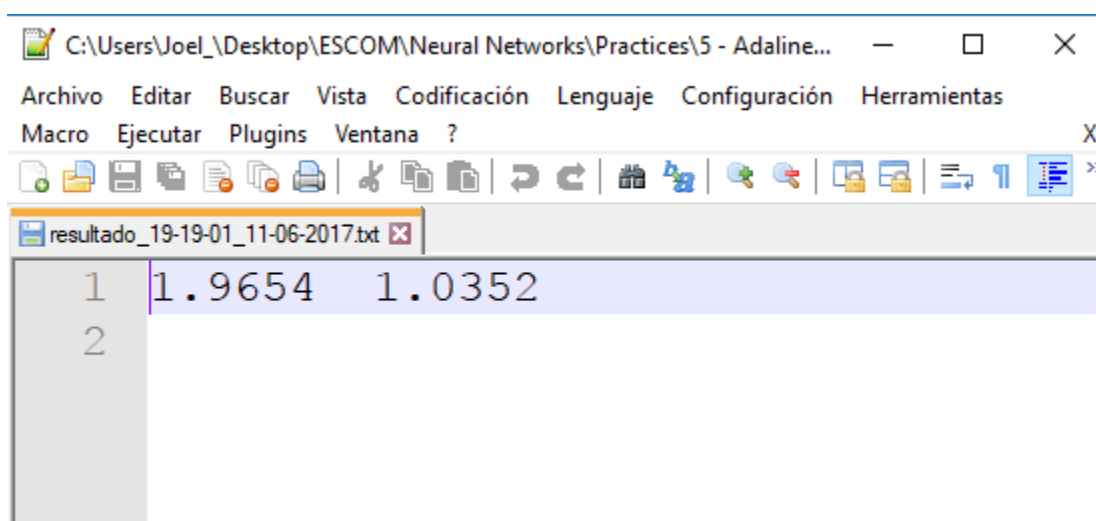


Figura 6. Valor final de los pesos para clasificar correctamente

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).

```
Command Window

Red Adaline

Ingresa el valor de iteraciones máximas: 15

Ingresa el valor al que deseas llegar la señal del error: 0.01

fx Ingresa el valor del factor de aprendizaje (alfa): 0.3
```

*Figura 7. Valores iniciales pedidos al usuario (1)*

```
Command Window

¿Cómo desea usar la Red Adaline?

1. Con Bias
2. Sin Bias

2

fx Ingresa el tamaño del decodificador: 4
```

*Figura 8. 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.

```
Command Window

¿Cómo desea usar la Red Adaline?

1. Con Bias
2. Sin Bias

2

Ingresa el tamaño del decodificador: 4

Se alcanzó la señal del error en la iteración 3

fx .>>
```

*Figura 9. 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.



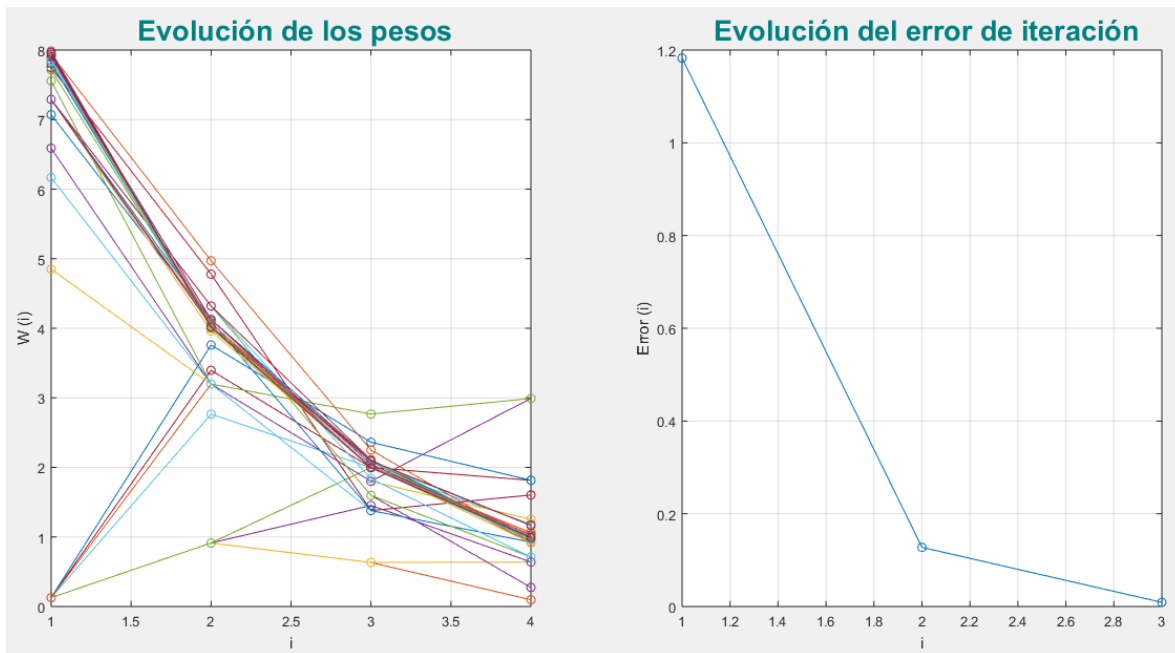


Figura 10. Gráficas para decodificador de 4 bits

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

The image shows a screenshot of a text editor window titled 'C:\Users\Joel\Desktop\ESCOM\Neural Networks\Practices\5 - Adaline...'. The file 'resultado\_19-29-23\_11-06-2017.txt' is open, displaying a table of final weights. The table has two rows and four columns of numerical values.

1	7.9678	4.019	1.996	0.99154
2				

Figura 11. 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.

```
Command Window

Red Adaline

Ingresa el valor de iteraciones máximas: 20

Ingresa el valor al que deseas llegar la señal del error: 0.001

fx Ingresa el valor del factor de aprendizaje (alfa): 0.3
```

Figura 12. Valores iniciales pedidos al usuario (1)

```
Command Window

¿Cómo desea usar la Red Adaline?

1. Con Bias
2. Sin Bias

2

fx Ingresa el tamaño del decodificador: 6
```

Figura 13. 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.

```
Command Window

¿Cómo desea usar la Red Adaline?

1. Con Bias
2. Sin Bias

2

Ingresa el tamaño del decodificador: 6

Se alcanzó la señal del error en la iteración 6

fx .>>
```

Figura 14. Mensaje final al usuario

Ahora, procedemos a ver las gráficas que se generaron.

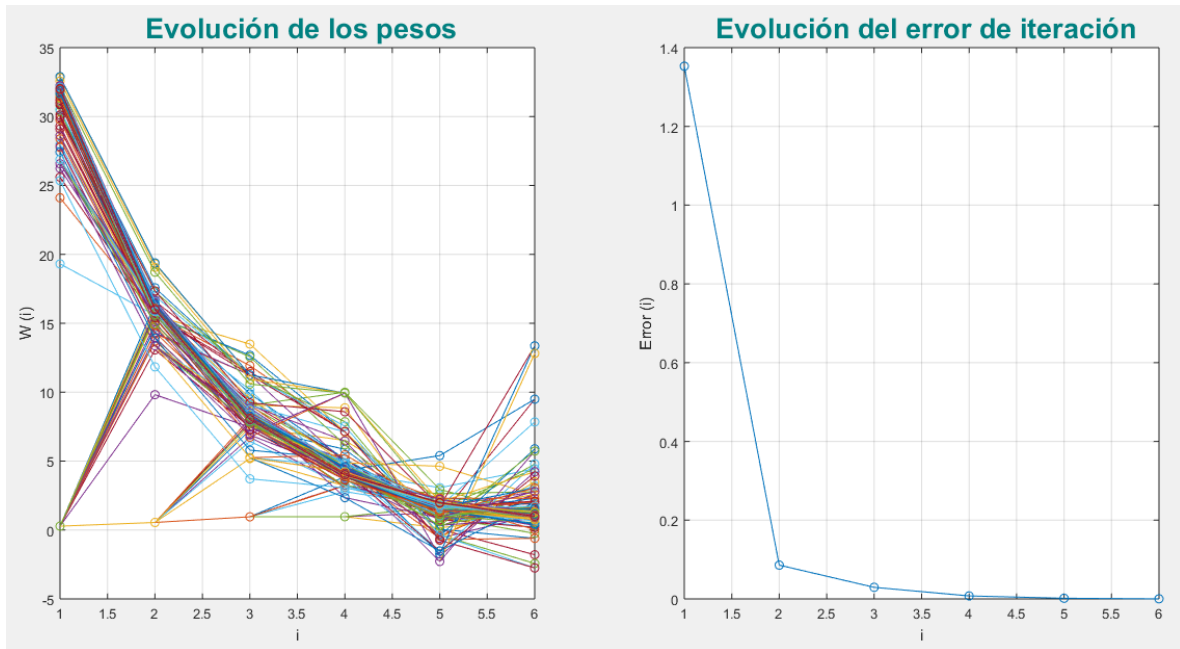


Figura 15. Gráficas para decodificador de 6 bits

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  $2^n$  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.

The image shows a screenshot of a text editor window. The title bar indicates the file path: 'C:\Users\Joel\Desktop\ESCOM\Neural Networks\Practices\5 - Adaline\_2\resultado\_19-52-38\_11...'. The menu bar includes 'Archivo', 'Editar', 'Buscar', 'Vista', 'Codificación', 'Lenguaje', 'Configuración', 'Herramientas', 'Macro', 'Ejecutar', and 'Plugins'. The toolbar contains various icons for file operations. The active window is 'resultado\_19-52-38\_11-06-2017.txt'. The content of the file is as follows:

1	31.999	16	7.9995	3.9997	1.9991	1.0009
2						

Figura 16. 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:**

La Red Adaline es como una mejora del perceptrón simple, ya que como habíamos visto el perceptrón necesita  $S$  neuronas para separar  $2^S$  clases, y en el caso de Adaline no, además el proceso de aprendizaje es casi igual, solamente que el factor de aprendizaje Alpha ayuda mucho a que los pesos y bias no crezcan demasiado ya que éste, suele tener valores entre 0 y 1.

La realización de esta red fue demasiado sencilla, sin embargo, creo que la graficación de los pesos no la hice como se requería, ya que yo grafiqué cada cambio que existía en el vector, y como se puede observar para muchos bits se distorsiona demasiado, así que probablemente lo que se había que graficar era el vector de pesos, pero en lugar de graficarlo cada que se actualiza con un dato, graficarlo cada que era una nueva iteración.

### **Conclusiones:**

Escribir los datos en un archivo es mucho más fácil ya que no tenemos que estar utilizando memoria a cada momento, porque, posteriormente se puede hacer la graficación leyendo de  $n$  datos en  $n$  datos en vez de estar almacenando todo el tiempo esos  $n$  datos.

Ese principio fue el que utilicé para la graficación de los pesos ya que, en el caso de 6 bits, si se hubiera puesto un factor de escala no muy adecuado o una señal del error demasiado pequeña o ambas, el llegar hasta el número de iteraciones máximas sería estar almacenando  $63 * 6 *$  iteraciones datos en memoria, que es un computacionalmente muy costoso.

### **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>

11

```

        end

        if error ~= 0
            %Agregamos el valor del error al error de iteración
            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 cumplió alguno de los criterios de finalización
    if flag == 1
        fprintf ('\nTodos los datos fueron bien clasificados en la iteración %d.\n\n',
k);
        break;
    elseif Error_iteracion (1, k) < e_it && Error_iteracion (1, k) > 0
        fprintf ('\nSe alcanzó la señal del error en la iteración %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 iteración
    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 .5}Evolución del error de iteración');
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 iteración
    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 .5}Evolución del error de iteración');
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 .5}Evolución de los pesos');
end
else
    fprintf ('\nOpción inválida.\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 última columna, ponemos el valor decimal
    conjunto ((i + 1), (numero_bits + 1)) = i;
end
end

```