UNIVERSIDAD POLITECNICA SALESIANA

Nombre: María José Peláez

Asignatura: Inteligencia Artificial II

Red Neuronal ADALINE

La arquitectura de Adaline (Adaptive Linear Neuron) fue creada por Bernard Widrow en 1959. Utiliza un dispositivo lógico que realiza una suma lineal de las entradas y genera una función umbral para el resultado de dicha suma.

La red Adaline es similar al Perceptron, excepto en su función de transferencia, la cual es una función de tipo lineal en lugar de un limitador fuerte como en el caso del Perceptron. La red Adaline presenta la misma limitación del Perceptron en cuanto al tipo de problemas que pueden resolver, ambas redes pueden solo resolver problemas linealmente separables, sin embargo el algoritmo LMS es más potente que la regla de aprendizaje del Perceptrón ya que minimiza el error medio cuadrático, la regla sirvió de inspiración para el desarrollo de otros algoritmos, este es el gran aporte de esta red. El término Adaline es una sigla, sin embargo su significado cambió ligeramente a finales de los años sesenta cuando decayó el estudio de las redes neuronales, inicialmente se llamaba ADAptive LInear NEuron (Neurona Lineal Adaptiva), para pasar después a ser Adaptive LInear Element (Elemento Lineal Adaptivo), este cambio se debió a que la Adaline es un dispositivo que consta de un único elemento de procesamiento, como tal no es técnicamente una red neuronal (Velandia y otros).
Características

- **Aprendizaje OFF Line**. - Requiere fase de entrenamiento, debe existir un conjunto de datos de entrenamiento y un conjunto de datos de test o prueba.
- **Aprendizaje por corrección de error**. - Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos en la salida.
- **LMS**. - Minimizar el error cuadrático
- **Aprendizaje Supervisado**. - Se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor, maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada.

El Adaline se entrena por medio de un proceso de minimización de errores que garantiza la convergencia a una solución. ###
Estructura de la red ![Texto Alternativo](ann_neurona.png)
El elemento de procesamiento realiza la suma de los productos de los vectores de entrada y de pesos, y aplica una función de salida para obtener un único valor de salida, el cual debido a su función de transferencia lineal será +1 si la sumatoria es positiva o –1 si la salida de la sumatoria es negativa.

$$y = \sum_{i=1}^{n} w_i * x_i + \theta$$

$$f(y) = \left\{ \begin{matrix} +1, si y \geq 0 \\ -1, si y < 0 \end{matrix} \right\}$$

La regla de aprendizaje de ADALINE considera el error entre la salida lograda **y** versus la salida deseada **d**.

$$| \quad \vec{y} \quad - \quad \vec{d} \quad |$$

Esta regla se conoce como **REGLA DELTA**

$$\Delta w_i = \alpha \sum_{\forall p} (dp - y_p) x_i$$

La constante α se denomina **TAZA DE APRENDIZAJE**

Regla de Aprendizaje

El ALC realiza el cálculo de la suma ponderada de las N entradas:

$$y = b + \sum_{j=1}^{N} w_j * x_j$$

Para realizar una simplificación en la función de salida, vamos a considerar el valor umbral bcomo una conexión ficticia de peso

w0. Si tenemos en cuenta que para esta entrada se toma el valor de x_0=1, la ecuación anterior se puede escribir de la forma:

$$y = w_0 x_0 + \sum_{j=1}^{N} w_j x_j = \sum_{j=0}^{N} w_j x_j$$

La regla de aprendizaje de mínimo error cuadrado medio LMS (Least Mean Squared) es conocida también como regla delta porque trata de minimizar un delta o diferencia entre valor observado y deseado en la salida de la red. Entonces, la regla delta, es un método para hallar el vector de pesos W deseado, el cual deberá ser único y asociar con éxito cada vector del conjunto de vectores o patrones de entrada $x_1, x_2, x_3, \ldots, x_Q$ con su correspondiente valor de salida correcto o deseado $d_1, d_2, d_3, \ldots, d_Q$.

Concretamente, la regla de aprendizaje LMS minimiza el error cuadrado medio, definido como:

$$< E_k^2 >= \frac{1}{2} \sum_{k=1}^{Q} E_k^2$$

donde Q es el número de vectores de entrada (patrones) que forman el conjunto de entrenamiento, y εkla diferencia entre la salida deseada y la obtenida cuando se introduce el patrón k-ésimo, que en el caso del Adaline, se expresa como $E_k = (dp - y_p)$, siendo y_k la salida del ALC; es decir:

$$y_k = x_k W_T = \sum_{j=0}^{N} w_j x_k$$

La función error es una función matemática definida en el espacio de pesos multidimensionales para un conjunto de patrones dados. Es una superficie que tendrá muchos mínimos (global y locales), y la regla de aprendizaje va en busca del punto en el espacio de pesos donde se encuentra el mínimo global de esta superficie. Aunque la superficie de error es desconocida, el método de gradiente decreciente consigue obtener información local de dicha superficie a través del gradiente. Con esta información se decide qué dirección tomar para llegar al mínimo global de dicha superficie. Entonces las modificaciones de los pesos son proporcionales al gradiente decreciente de la función error, siendo α la constante de proporcionalidad o tasa de aprendizaje, en notacion matricial queda:

$$W(t+1) = W(t) + \alpha E_k X_k = W(t) + \alpha(dp - y_p) x_k$$

Esta expresión representa la modificación de los pesos obtenida a partir del algoritmo LMS, y es parecida a la obtenida anteriormente para el caso del Perceptrón. α es el parámetro que determina la estabilidad y la velocidad de convergencia del vector de pesos hacia el valor de error mínimo. Los cambios en dicho vector deben hacerse relativamente pequeños en cada iteración, sino podría ocurrir que no se encontrase nunca un mínimo, o se encontrase solo por accidente, en lugar de ser el resultado de una convergencia sostenida hacia él. Aunque a simple vista no parece que exista gran diferencia entre los mecanismos de aprendizaje del Perceptrón y del Adaline, este último mejora al del Perceptrón ya que va a ser más sencillo alcanzar el mínimo error, facilitando la convergencia del proceso de entrenamiento.

Algoritmo de Aprendizaje

Paso 1: Se inicializa la matriz de pesos y el valor de la ganancia, por lo general se asignan valores aleatorios pequeños a cada uno de los pesos $w_i yelvalorb$ El valor de bias b puede tomarse como el valor de un peso adicional w_0 asociado con una entrada adicional siempre en 1.

Paso 2: Se aplica un vector o patrón de entrada p_k en las entradas del Adaline.

Paso 3: Se obtine la salida lineal $a_k = x_k * W_T = \sum_{j=0}^{N} W_j X_k$ y se calcula la diferenciación respecto a la deseada $E_k = (dp - y_p)$.

Paso 4: Se actualizan los pesos: $W(t+1) = W(t) + \alpha E_k X_k = W(t) + \alpha(dp - y_p) x_k$

$$\Delta w_j = \gamma(dp - y_p) x_j$$

$$\Delta_p \theta = \gamma(dp - y_p)$$

Paso 5: Se repiten los pasos 2 al 4 con todos los vectores de entrada (Q)

Paso 6: Si el error cuadrático medio $ = \frac{1}{2} \sum_{k=1}^{Q} E_{k}^{2} $ es un valor reducido aceptable, termina el proceso de aprendizaje; sino, se repite otra vez desde el paso 2 con todos los patrones.

Parada de Aprendizaje

Criterio 1: Fijar un número de ciclos máximo. Dicho número debe garantizar que el error cuadrático para los patrones de entrenamiento se haya estabilizado.

Criterio 2: Cuando el error cuadrático sobre los patrones de entrenamiento no cambia durante x ciclos

Criterio 3: Sí se puede utilizar un conjunto de validación, que correspondería con una porción aleatoria del conjunto de entrenamiento. En este caso, el criterio sería: cuando el error cuadrático sobre los patrones de validación no aumenta o se mantiene estable a lo largo de x ciclos.

Aplicaciones

Esta red a sido utilizada en ciencia, Estadística: en el análisis de regresion lineal, procesamiento adaptativo de senales, control de sistemas, fitrado lineal de datos, Reconocimiento de voz y caracteres, filtros adaptativos que eliminan ecos de las líneas telefónicas.

La aplicación más común de la red Adaline es el filtro adaptativo, para eliminar el ruido de una señal de entrada. Cuando se diseña un filtro digital por medio de software; con un programa normal, el programador debe saber exactamente como se específica el algoritmo de filtrado y cuales son los detalles de las características de las señales; si se necesitaran modificaciones, o si cambian las características de la señal, es necesario reprogramar; cuando se emplea una red tipo Adaline, el problema se convierte, en que la red sea capaz de especificar la señal de salida deseada, dada una señal de entrada específica.

La red Adaline toma la entrada y la salida deseada, y se ajusta a sí misma para ser capaz de llevar a cabo la transformación deseada. Además, si cambian las características de la señal, la red Adaline puede adaptarse automáticamente.

Ejemplo 2

Decodificador de Decimal a Binario

Pesos iniciales:

$\vec{w} = [-0.23,\ -1.87]$

x1	x2	d(salida deseada)
0.73	0.21	1
1.2	1.7	1
2.07	3.7	0
2.33	2.7	0

$\theta = 1.31\ \alpha = 0.91$

1) Primera entrada

$\boldsymbol{y = b + \sum_{j=1}^{N} w_j * x_j = b + w_1 * x_1 + w_2 * x_2}$

$= 1.31 + (-0.23 * 0.73) + (-1.87 * 0.21)$

$= 0.75$

$(d_p - y_p) = (1 - 0.75) = 0.25$

Actualizar pesos:

W1 nuevo

$W(t+1) = W(t) + \alpha E_k X_k = W(1) + \alpha(dp - y_p)x_1$

$W(t+1) = -0.23 + 0.91 * (0.25) * 0.73 = -0.06$

W2 nuevo

$W(t+1) = W(t) + \alpha E_k X_k = W(2) + \alpha(dp - y_p)x_2$

$W(t+1) = -1.87 + 0.91 * (0.25) * 0.21 = -1.82$

b nuevo

$b(t+1) = b_anterior + \alpha(dp - y_p)$

$b(t+1) = 1.31 + 0.91 * (0.25) = 1.54$

Error Cuadratico

$E = (d_p - y_p)^2 = (0.25)^2 = 0.06$

2) Segunda entrada

$$y = b + \sum_{j=1}^{N} w_j * x_j = b + w_1 * x_1 + w_2 * x_2$$

$$= 1.54 + (-0.06 * 1.2) + (-1.82 * 1.7)$$

$$= -1.63$$

$(d_p - y_p) = (1 + 1.63) = 2.63$

Actualizar pesos:

W1 nuevo

$W(t+1) = W(t) + \alpha E_k X_k = W(1) + \alpha(dp - y_p)x_1$

$W(t+1) = -0.06 + 0.91 * (2.63) * 1.2 = 2.81$

W2 nuevo

$W(t+1) = W(t) + \alpha E_k X_k = W(2) + \alpha(dp - y_p)x_2$

$W(t+1) = -1.82 + 0.91 * (2.63) * 1.7 = 2.24$

b nuevo

$b(t+1) = b_anterior + \alpha(dp - y_p)$

$b(t+1) = 1.54 + 0.91 * (2.63) = 3.93$

Error Cuadratico

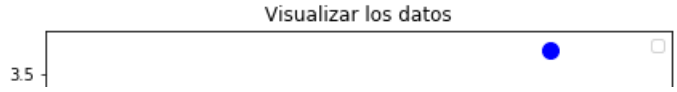
$E = (d_p - y_p)^2 = (2.63)^2 = 6.91$

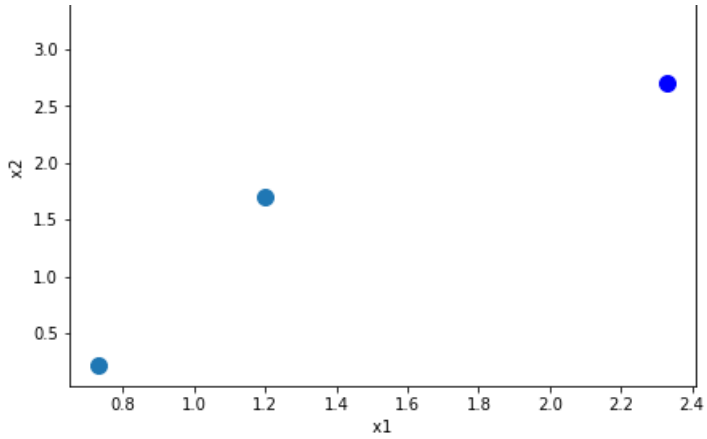
In [27]:

```
import matplotlib.pyplot as plt

%matplotlib inline
x1=[1.0,1.0,-1.0,-1.0]
x2=[1.0,-1.0,1.0,-1.0]

xx1=[0.73,1.2]
xx2=[0.21,1.7]

xx3=[2.07,2.33]
xx4=[3.7,2.7]

print("Entrada X1",x1)
print("Entrada X2",x2)

plt.figure(figsize=(7,5))
#plt.plot(x1,x2,'o',markersize=10)
plt.plot(xx1,xx2,'o',markersize=10)
plt.plot(xx3,xx4,'o',markersize=10, color='b')
plt.legend()
plt.title ( ' Visualizar los datos ' )
plt.xlabel ( ' x1 ' )
plt.ylabel ( ' x2 ' )
plt.show()
```

```
No handles with labels found to put in legend.
```

```
Entrada X1 [1.0, 1.0, -1.0, -1.0]
Entrada X2 [1.0, -1.0, 1.0, -1.0]
```





Ejemplo de una Red Neuronal Adaline

Pesos iniciales:

$\vec{w} = [0.2]$

x1	x2	d(salida deseada)
1	1	-1
1	-1	1
-1	1	-1
-1	-1	-1

$\theta = 0.2 \; \alpha = 0.2$

1) Primera entrada

$$\boldsymbol{y = b + \sum_{j=1}^{N} w_j * x_j = b + w_1 * x_1 + w_2 * x_2}$$

$$= 0.2 + (-0.2 * 1) + (0.2 * 1)$$

$$= 0.6$$

$$(d_p - y_p) = (-1 - 0.6) = -1.6$$

Actualizar pesos:

W1 nuevo

$$W(t+1) = W(t) + \alpha E_k X_k = W(1) + \alpha(dp - y_p)x_1$$

$$W(t+1) = -0.2 + 0.2 * (-1 - 0.6) * 1 = -0.12$$

W2 nuevo

$$W(t+1) = W(t) + \alpha E_k X_k = W(2) + \alpha(dp - y_p)x_2$$

$$W(t+1) = 0.2 + 0.2 * (-1 - 0.6) * 1 = -0.12$$

b nuevo

$$b(t+1) = b_anterior + \alpha(dp - y_p)$$

$$b(t+1) = 0.2 + 0.2 * (-1 - 0.6) = -0.12$$

Error Cuadratico

$$E = (d_p - y_p)^2 = (-1.6)^2 = 2.56$$

2) Segunda entrada

$$\boldsymbol{y = b + \sum_{j=1}^{N} w_j * x_j = b + w_1 * x_1 + w_2 * x_2}$$

$$= -0.12 + (-0.12 * 1) + (-0.12 * (-1))$$

$= -0.12$

$(d_p - y_p) = (1 - (-0.12)) = 1.12$

Actualizar pesos:

W1 nuevo

$W(t+1) = W(t) + \alpha E_k X_k = W(1) + \alpha(dp - y_p)x_1$

$W(t+1) = -0.12 + 0.2 * (1 - (-0.12)) * 1 = 0.10$

W2 nuevo

$W(t+1) = W(t) + \alpha E_k X_k = W(2) + \alpha(dp - y_p)x_2$

$W(t+1) = -0.12 + 0.2 * (1 - (-0.12)) * (-1) = -0.34$

b nuevo

$b(t+1) = b_anterior + \alpha(dp - y_p)$

$b(t+1) = -0.12 + 0.2 * (1 - (-0.12)) = 0.10$

Error Cuadratico

$E = (d_p - y_p)^2 = (1.12)^2 = 1.25$

3) Tercera Entrada

$\boldsymbol{y = b + \sum_{j=1}^{N} w_j * x_j = b + w_1 * x_1 + w_2 * x_2}$

$= 0.10 + (0.10 * (-1)) + (-0.34 * 1)$

$= -0.34$

$(d_p - y_p) = (-1 - (-0.34)) = -0.66$

Actualizar pesos:

W1 nuevo

$W(t+1) = W(t) + \alpha E_k X_k = W(1) + \alpha(dp - y_p)x_1$

$W(t+1) = 0.10 + 0.2 * (-1 - (-0.34)) * (-1) = 0.23$

W2 nuevo

$W(t+1) = W(t) + \alpha E_k X_k = W(2) + \alpha(dp - y_p)x_2$

$W(t+1) = -0.34 + 0.2 * (-1 - (-0.34)) * (1) = -0.47$

b nuevo

$b(t+1) = b_anterior + \alpha(dp - y_p)$

$b(t+1) = 0.10 + 0.2 * (-0.66) = -0.03$

Error Cuadratico

$E = (d_p - y_p)^2 = (-0.66)^2 = 0.43$

2) Cuarta entrada

$\boldsymbol{y = b + \sum_{j=1}^{N} w_j * x_j = b + w_1 * x_1 + w_2 * x_2}$

$= -0.03 + (0.23 * (-1)) + (-0.47 * (-1))$

$= 0.21$

$(d_p - y_p) = (-1 - 0.12) = -1.21$

Actualizar pesos:

W1 nuevo

$$W(t+1) = W(t) + \alpha E_k X_k = W(1) + \alpha(dp - y_p)x_1$$

$$W(t+1) = 0.23 + 0.2 * (-1.21) * (-1) = 0.47$$

W2 nuevo

$$W(t+1) = W(t) + \alpha E_k X_k = W(2) + \alpha(dp - y_p)x_2$$

$$W(t+1) = -0.47 + 0.2 * (-1.21) * (-1) = -0.23$$

b nuevo

$$b(t+1) = b_anterior + \alpha(dp - y_p)$$

$$b(t+1) = -0.03 + 0.2 * (-1.21) = -0.27$$

Error Cuadratico

$$E = (d_p - y_p)^2 = (-1.21)^2 = 1.47$$

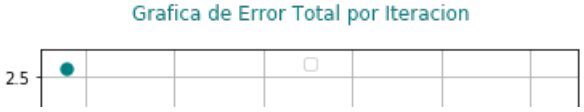
In [32]:

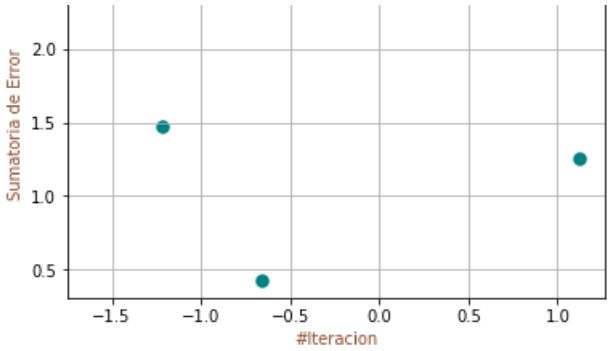
```
import numpy as np
import matplotlib.pyplot as plt

class Adaline():
    def __init__(self):
        self.Matriz = [[1.0, 1.0], [1.0, -1.0], [-1.0, 1.0], [-1, -1]]
        self.W = np.array([0.2, 0.2])
        self.bias =0.2
        self.alfa = 0.2
        self.delta = [-1.0, 1.0, -1.0, -1.0]
        self.cont0 = 0
        self.estado = False
        self.MatrizResultados=[]
        self.MatrizResultadoss=[]
    def calcular(self):


        while(self.estado == False):
            cont = 0
            for i in self.Matriz:
                self.X = np.array(i)
                n = np.sum(np.multiply(self.X, self.W)) + self.bias

                print("Entrada X1 X2", self.X)
                print("Pesos W1 W2", self.W)
                print("Salida Deseada d: ", self.delta[cont])

                self.error = (self.delta[cont] - n)
                cont += 1
                print("Salida Obtenida: ", n)
                self.error2=self.error*self.error
                print("Error Cuadratico: ", self.error2)
                self.MatrizResultados.append(self.error2)
                self.MatrizResultadoss.append(self.error)
                if (self.error != 0):
                    self.cont0=0
                    print("Actualizar pesos")
                    print("_____________________________________\n")
                    self.W = (self.W + np.multiply(self.alfa* self.error, self.X))
                    print("Nuevo Peso W1 y W2", self.W)
                    self.bias = self.bias + (self.alfa * self.error)
                    print("Nuevo bias", self.bias)
                    self.estado=True
                else:
                    self.cont0+=1

                    if(self.cont0==len(self.delta)):
                        print("Acabo")
                        print("Pesos Finales",self.W)
                        print("Bias Final",self.bias)
```

```
                        self.estado=True
                        break

                print("#######################################\n")

    def crearGrafica(self):
        pp.scatter(self.MatrizResultadoss, self.MatrizResultados, color="teal", linewidth=2.5, linestyl
e="-")
        pp.suptitle("Grafica de Error Total por Iteracion", color="teal")
        #pp.plot(x, y, color="teal", linewidth=2.5, linestyle="-", label="a")
        pp.xlabel("#Iteracion", color="sienna")
        pp.ylabel("Sumatoria de Error", color="sienna")
        pp.legend(loc='upper center')
        pp.grid(True)
        # pp.set_title('Grafica Final')
        pp.savefig('grafico.png')
        pp.show()

if __name__ == '__main__':
    ada = Adaline()
    ada.calcular()
    ada.crearGrafica()
```

```
No handles with labels found to put in legend.
```

```
Entrada X1 X2 [1. 1.]
Pesos W1 W2 [0.2 0.2]
Salida Deseada d:  -1.0
Salida Obtenida:  0.6000000000000001
Error Cuadratico:  2.5600000000000005
Actualizar pesos
_____________________________________

Nuevo Peso W1 y W2 [-0.12 -0.12]
Nuevo bias -0.12000000000000005
#####################################

Entrada X1 X2 [ 1. -1.]
Pesos W1 W2 [-0.12 -0.12]
Salida Deseada d:  1.0
Salida Obtenida:  -0.12000000000000005
Error Cuadratico:  1.2544000000000002
Actualizar pesos
_____________________________________

Nuevo Peso W1 y W2 [ 0.104 -0.344]
Nuevo bias 0.10399999999999998
#####################################

Entrada X1 X2 [-1.  1.]
Pesos W1 W2 [ 0.104 -0.344]
Salida Deseada d:  -1.0
Salida Obtenida:  -0.3440000000000001
Error Cuadratico:  0.4303359999999999
Actualizar pesos
_____________________________________

Nuevo Peso W1 y W2 [ 0.2352 -0.4752]
Nuevo bias -0.027200000000000002
#####################################

Entrada X1 X2 [-1 -1]
Pesos W1 W2 [ 0.2352 -0.4752]
Salida Deseada d:  -1.0
Salida Obtenida:  0.2128000000000001
Error Cuadratico:  1.4708838400000002
Actualizar pesos
_____________________________________

Nuevo Peso W1 y W2 [ 0.47776 -0.23264]
Nuevo bias -0.26976
#####################################
```





Referencias

Velandia & Yudi, V. Y. (s.f.). Adaline Red Nuronal. Bogotá: Universidad Panamericana.
Jonathan Arana. (2014). Adaline Inteligencia Artificial . 23/042019, de SlideShare Sitio web: https://es.slideshare.net/JonathanArana1/adaline

Luis Federico Bertona . (2005). Entrenamiento de Redes Neuronales dasado En Algoritmos Evolutivos. 23/04/2019, de Universidad de Buenos Aires Sitio web: http://laboratorios.fi.uba.ar/lsi/bertona-tesisingenieriainformatica.pdf

Xabier Basogain Olabe. (2010). Redes Neuronales Artificiales y sus aplicaciones. 23/04/2019, de Escuela Superior de Ingeniería de Bilbao Sitio web: https://ocw.ehu.eus/file.php/102/redes_neuro/contenidos/pdf/libro-del-curso.pdf

Fernando Tanco. (2011). Introducción a las Redes Neuronales Artificiales. 23/04/2019, de Universidad Tecnológica Nacional Facultad Regional Buenos Aires Sitio web: http://www.secyt.frba.utn.edu.ar/gia/RNA.pdf

Carlos Regueiro. (2008). Modelos básicos de Redes Neuronales Artificiales. 23/04/2019, de Universidad de A Coruña Sitio web: https://minerva.usc.es/xmlui/bitstream/handle/10347/12132/8cc86neuronal.pdf?sequence=1&isAllowed=y

ADALINE for Pattern Classification, Prof. de la Universidad de NuevaYork (NYU Tandon School of Engineering).http://cis.poly.edu/~mleung/CS6673/s09/ADALINE.pdf

Inés Galván. (2010). Primeros Modelos Computacionales. 23/04/2019, de Universidad Carlos III de Madrid Sitio web: http://ocw.uc3m.es/ingenieria-informatica/redes-de-neuronas/transparencias/Tema2%20PerceptronAdalineRN.pdf

Eliana Cordoba. (2015). Redes Adaline. 23/04/2019, de prezi Sitio web: https://prezi.com/j82sylwvm0a5/redes-adaline/