El Perceptron de Rosenblatt

2

Profesor: Ricardo Rodriguez Bustinza
Email: robust@uni.edu.pe

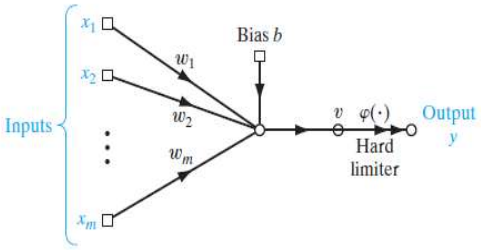
Introducción

In the formative years of neural networks (1943–1958), several researchers stand out for their pioneering contributions:

- McCulloch and Pitts (1943) for introducing the idea of neural networks as computing machines.
- Hebb (1949) for postulating the first rule for self-organized learning.
- Rosenblatt (1958) for proposing the perceptron as the first model for learning with a teacher (i.e., supervised learning).

Modelo Rosenblatt (3/1)

- El perceptron de Rosenblatt está construido alrededor de una neurona no lineal, es decir, el modelo de McCulloch-Pitts de una neurona. Recordemos que tal modelado neuronal consiste en un combinador lineal seguido de un limitador duro (que realiza la función signo). El nodo sumador del modelo neural calcula un lineal combinación de las entradas aplicadas a sus sinapsis, así como también incorpora un externo sesgo aplicado.
- La suma resultante, es decir, el campo local inducido, se aplica a un limitador. En consecuencia, la neurona produce una salida igual a 1 si el limitador duro es positivo y -1 si es negativo.
- En el modelo de gráfico de flujo de señal de la figura, los pesos sinápticos del perceptron se denotan por $\omega_1, \omega_2, ..., \omega_m$. En consecuencia, las entradas aplicadas al perceptron son denotados por $x_1, x_2, ..., x_m$.



Modelo Rosenblatt (3/2)

- El sesgo aplicado externamente se denota por b. Del modelo, encontramos que la entrada del limitador duro, o campo local inducido, de la neurona es:

$$v = \sum_{i=1}^{m} w_i x_i + b$$

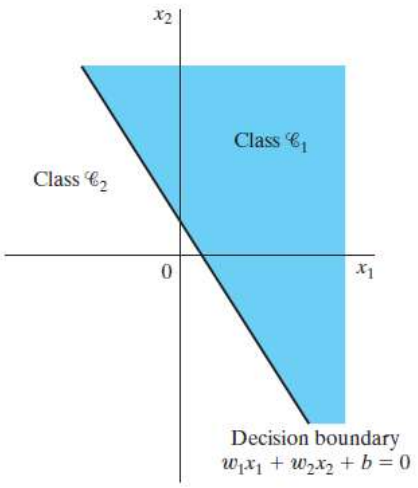
- El objetivo del perceptron es clasificar correctamente el conjunto de estímulos aplicados externamente $x_1, x_2, ..., x_m$ en una de dos clases, c_1 o c_2. La regla de decisión para la clasificación es asignar el punto representado por las entradas $x_1, x_2, ..., x_m$ a la clase c_1 si la salida del perceptron y es $+1$ y para la clase c_2 si la salida y es -1.
- Para desarrollar información sobre el comportamiento de un clasificador de patrones, es costumbre trazar un mapa de las regiones de decisión en el espacio de señal m-dimensional abarcado por la entrada m variables $x_1, x_2, ..., x_m$. En la forma más simple del perceptron, hay dos regiones de decisión. separados por un hiperplano, que se define por:

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

Modelo Rosenblatt (3/3)

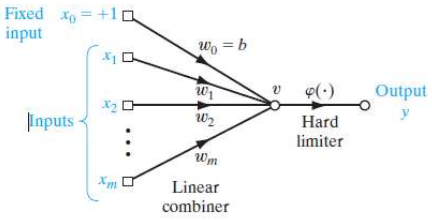
- La ecuación del limite de decision, es mostrado en la Figura para el caso de dos variables de entrada x_1 y x_2, para las cuales el límite de decisión toma la forma de una línea recta. Un punto (x_1, x_2) que se encuentra sobre el la línea de límite se asigna a la clase c_1 y un punto (x_1, x_2) que se encuentra debajo de la línea de límite se asigna a la clase c_2. Tenga en cuenta también que el efecto del sesgo b es simplemente cambiar la decisión límite lejos del origen.

- Los pesos sinápticos ω_1, ω_2, ..., ω_m del perceptron se adapta mediante iteraciones. Para la adaptación, podemos usar una regla de corrección de errores conocida como **Algoritmo de Convergencia perceptron.**



Teorema Convergencia del Perceptron (4/1)

- Para derivar el algoritmo de aprendizaje de corrección de errores para el perceptron, lo encontramos más conveniente para trabajar con el modelo de gráfico de flujo de señal modificado en la Figura.



- En este segundo modelo, es equivalente al 1943 debido a que tiene el sesgo $b(n)$ se trata como un peso sináptico impulsado por una entrada fija igual a 1.

$$x(n) = [+1, x_1(n), x_2(n), \cdots, x_m(n)]^T$$

- donde n denota el paso de tiempo en la aplicación del algoritmo. En consecuencia, definimos el vector de peso como:

$$w(n) = [b, w_1(n), w_2(n), \cdots, w_m(n)]^T$$

Teorema Convergencia del Perceptron (4/2)

- En consecuencia, la salida del combinador lineal se escribe en forma compacta

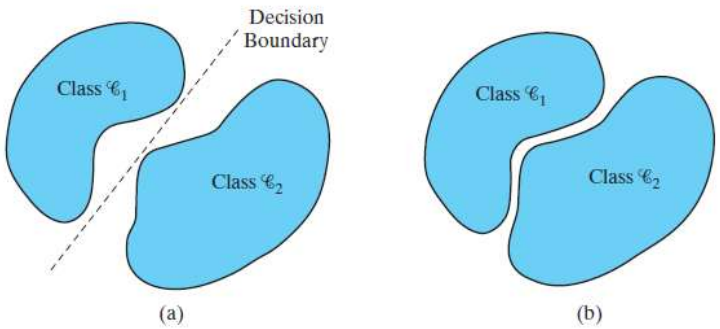
$$v(n) = \sum_{i=0}^{m} w_i(n)x_i(n) = w^T(n)x(n)$$

- Para n fijo, la ecuación $\omega^T x = 0$, trazada en un espacio m-dimensional (y para algunos sesgos prescritos) con coordenadas x_1, x_2, ..., x_m, define un hiperplano como la superficie de decisión entre dos diferentes clases de entradas.

- Para que el perceptron funcione correctamente, las dos clases c_1 y c_2 deben ser linealmente separable esto, a su vez, significa que los patrones a clasificar deben estar suficientemente separados uno del otro para asegurar que la superficie de decisión consista en un hiperplano.

Teorema Convergencia del Perceptron (4/3)

- El requisito se ilustra en la figura para el caso de un perceptron bidimensional. En la Figura (a) las dos clases c_1 y c_2 están suficientemente separadas entre sí para que podamos dibujar un hiperplano (en este caso, una línea recta) como límite de decisión. Sin embargo, los dos clases c_1 y c_2 se mueven demasiado cerca una de la otra, como en la Figura (b), se convierten en no linealmente separable, una situación que está más allá de la capacidad informática del perceptron



Teorema Convergencia del Perceptron (4/4)

- de los vectores para entrenar al clasificador, el proceso de entrenamiento implica el ajuste del vector de peso ω de tal manera que las dos clases c_1 y c_2 son linealmente separables. Es decir, existe un vector de peso ω tal que podamos indicar

$\omega^T x > 0$ para cada vector de entrada x perteneciente a la clase C_1
$\omega^T x \leq 0$ para cada vector de entrada x perteneciente a la clase C_2

- En la segunda línea de la ecuación, hemos elegido arbitrariamente decir que el vector de entrada x pertenece a la clase c_2 si $\omega^T x = 0$. Sean los subconjuntos de vectores de entrenamiento h_1 y h_2, el entrenamiento El problema para el perceptron es encontrar un vector de peso ω tal que las dos desigualdades de la ecuación están satisfechos. El algoritmo para adaptar el vector de peso del perceptron elemental ahora se formulará de la siguiente manera:

1. If the nth member of the training set, $\mathbf{x}(n)$, is correctly classified by the weight vector $\mathbf{w}(n)$ computed at the nth iteration of the algorithm, no correction is made to the weight vector of the perceptron in accordance with the rule:

$$\mathbf{w}(n+1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2$$

2. Otherwise, the weight vector of the perceptron is updated in accordance with the rule

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) > 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2$$
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n) \quad \text{if } \mathbf{w}^T(n)\mathbf{x}(n) \leq 0 \text{ and } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1$$

where the *learning-rate parameter* $\eta(n)$ controls the adjustment applied to the weight vector at iteration n.

If $\eta(n) = \eta > 0$, where η is a constant independent of the iteration number n, then we have a *fixed-increment adaptation rule* for the perceptron.

In the sequel, we first prove the convergence of a fixed-increment adaptation rule for which $\eta = 1$. Clearly, the value of η is unimportant, so long as it is positive. A value of $\eta \neq 1$ merely scales the pattern vectors without affecting their separability. The case of a variable $\eta(n)$ is considered later.

Algoritmo del Perceptron

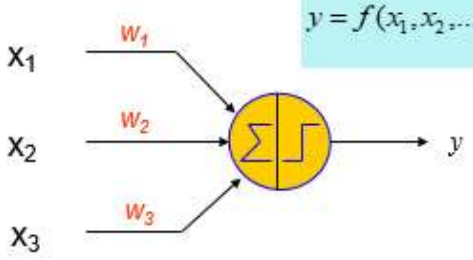
Variables and Parameters:

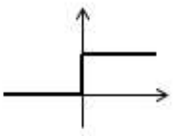
$\mathbf{x}(n)$ = $(m+1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as
$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$
where sgn(·) is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
where
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$
5. *Continuation.* Increment time step n by one and go back to step 2.

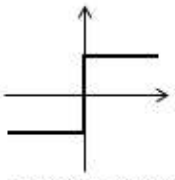
Miscelánea (3/1)

$$y = f(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{si} \quad w_1 x_i + w_2 x_2 + \ldots + w_n x_n \geq \theta \\ -1 & \text{si} \quad w_1 x_i + w_2 x_2 + \ldots + w_n x_n < \theta \end{cases}$$

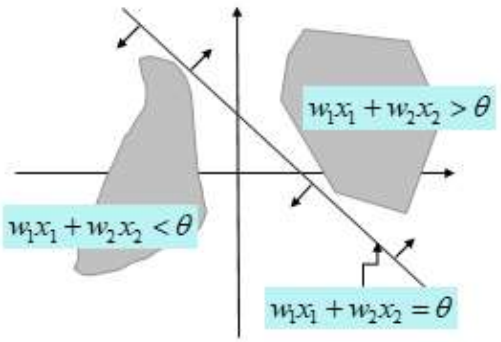




Función paso o *De Heaviside*



Función signo



Miscelánea (3/2)

Se dispone de la siguiente **información:**

- Conjunto de patrones $\{x^k\}$, $k = 1,2,\ldots,p_1$, de la clase C_1 $(z^k = 1)$
- Conjunto de patrones $\{x^r\}$, $k = p_1+1,\ldots,p$, de la clase C_2 $(z^r = -1)$
  - Se pretende que el perceptron asigne a cada entrada (patrón x^k) la salida deseada z^k siguiendo un proceso de corrección de error (aprendizaje) para determinar los pesos sinápticos apropiados.

Regla de aprendizaje del Perceptrón: $\Delta w_j(k) = \eta(k)[z(k) - y(k)]x_j(k)$

error

tasa de aprendizaje

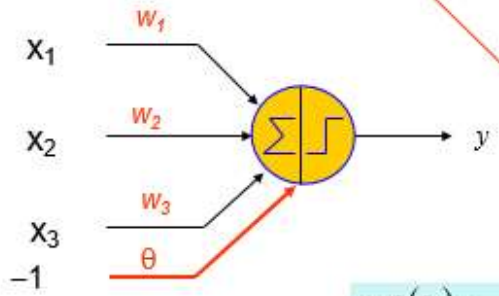
$$w_j(k+1) = \begin{cases} w_j(k) + 2\eta x_j(k) & \text{si } y(k) = -1 \text{ y } z(k) = 1, \\ w_j(k) & \text{si } y(k) = z(k) \\ w_j(k) - 2\eta x_j(k) & \text{si } y(k) = 1 \text{ y } z(k) = -1 \end{cases}$$

Miscelánea (3/3)

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n \geq \theta \quad \Leftrightarrow \quad w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_{n+1}x_{n+1} \geq 0$$

θ -1

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n + \theta(-1) \geq 0$$



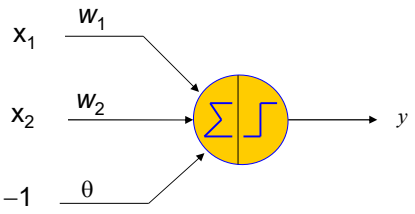
$$\Delta\theta(k) = -\eta(k)[z(k) - y(k)]$$

Ejemplo

- Diseñe una red de tipo perceptron simple que implemente la función **lógica AND.**

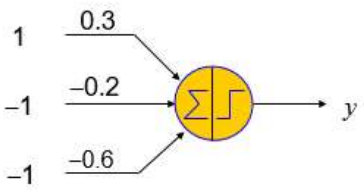
AND

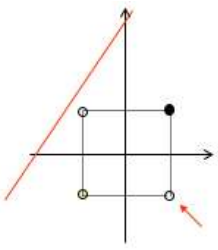
Entradas	Salidas
(1, 1)	1
(1, –1)	–1
(–1, 1)	–1
(–1,–1)	–1



- Inicialización pesos aleatorios pequeños, por ejemplo $\omega_1 = 0.3,\ \omega_2 = -0.2,\ \theta = -0.6$. Asuma un factor de aprendizaje de 0.5.

- En este caso a la red se ingresan los vectores de datos en forma aleatoria. No necesariamente que guarden cierto orden en las formas de la tablas.





Paso 1:
Patrón de entrada (1,–1): $h = 0.3(1) - 0.2(-1) - 0.6(-1) = 1.1$

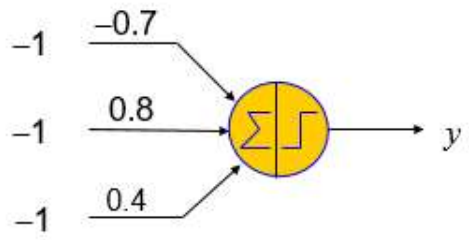
$y = 1$

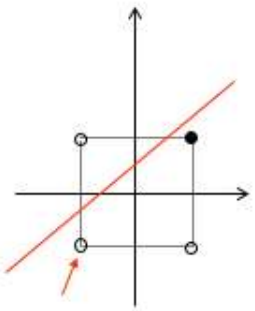
Paso 2: Corrección de los pesos sinápticos

$w_1(1) = w_1(0) - 2\eta 1 = 0.3 - 1 = -0.7$

$w_2(1) = w_2(0) - 2\eta(-1) = -0.2 + 1 = 0.8$

$\theta(1) = \theta(0) - 2\eta(-1) = -0.6 + 1 = 0.4$

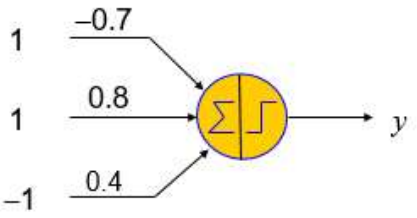


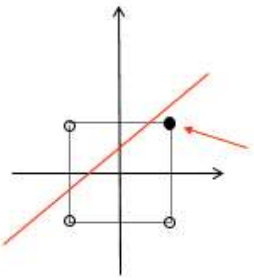


Paso 1:
Patrón de entrada (–1,–1): $h = -0.7(-1) + 0.8(-1) + 0.4(-1) = -0.5$

Como $y = -1$ y $z = -1$ la clasificación es **correcta**

$y = -1$





Paso 1:
Patrón de entrada (1,1): $h = -0.7(1) + 0.8(1) + 0.4(-1) = -0.3$

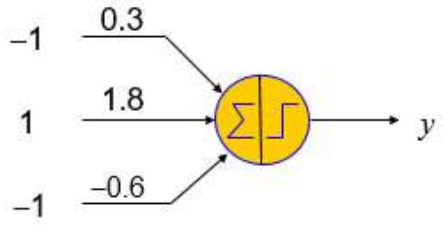
$y = -1$

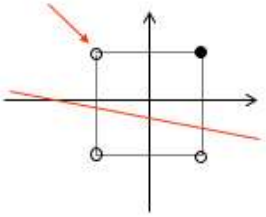
Paso 2: Corrección de los pesos sinápticos

$w_1(2) = w_1(1) + 2\eta(1) = -0.7 + 1 = 0.3$

$w_2(2) = w_2(1) + 2\eta(1) = 0.8 + 1 = 1.8$

$\theta(2) = \theta(1) + 2\eta(-1) = 0.4 - 1 = -0.6$





Paso 1:
Patrón de entrada (–1,1): $h = 0.3(-1) + 1.8(1) - 0.6(-1) = 2.1$

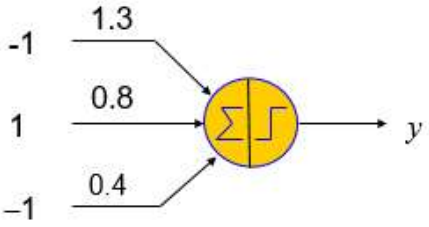
$y = 1$

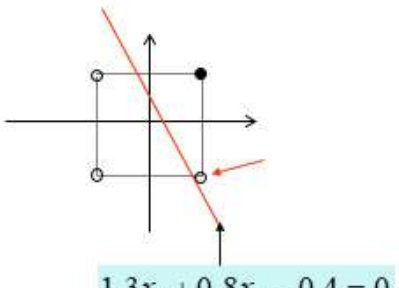
Paso 2: Corrección de los pesos sinápticos

$w_1(3) = w_1(2) - 2\eta(-1) = 0.3 + 1 = 1.3$

$w_2(3) = w_2(2) - 2\eta(1) = 1.8 - 1 = 0.8$

$\theta(3) = \theta(2) - 2\eta(-1) = -0.6 + 1 = 0.4$





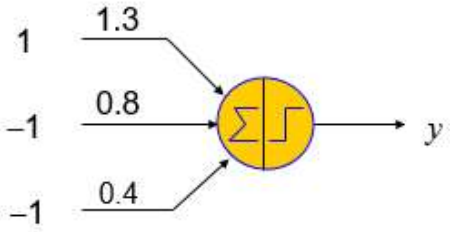
$1.3x_1 + 0.8x_2 - 0.4 = 0$

Patrón (1,1): $h = 1.3(1) + 0.8(1) + 0.4(-1) = 2.7$

Patrón (1,–1): $h = 1.3(1) + 0.8(-1) + 0.4(-1) = 0.1$ ←

Patrón (–1,–1): $h = 1.3(-1) + 0.8(-1) + 0.4(-1) = -2.5$

Patrón (–1,1): $h = 1.3(-1) + 0.8(1) + 0.4(-1) = -0.9$



Paso 1:
Patrón de entrada (1,–1): $h = 1.3(1) + 0.8(-1) + 0.4(-1) = 0.1$

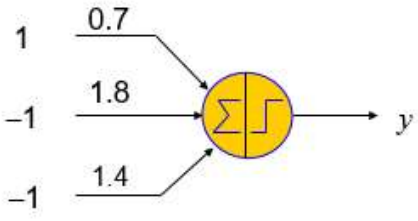
$y = 1$

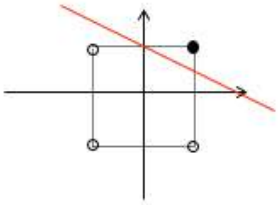
Paso 2: Corrección de los pesos sinápticos

$w_1(3) = w_1(2) - 2\eta(1) = 1.3 - 1 = 0.3$

$w_2(3) = w_2(2) - 2\eta(-1) = 0.8 + 1 = 1.8$

$\theta(3) = \theta(2) - 2\eta(-1) = 0.4 + 1 = 1.4$





Paso 1:
Patrón de entrada (1,–1): $h = 1.3(1) + 0.8(-1) + 0.4(-1) = 0.1$

$y = 1$

Paso 2: Corrección de los pesos sinápticos

$w_1(3) = w_1(2) - 2\eta(1) = 1.3 - 1 = 0.3$

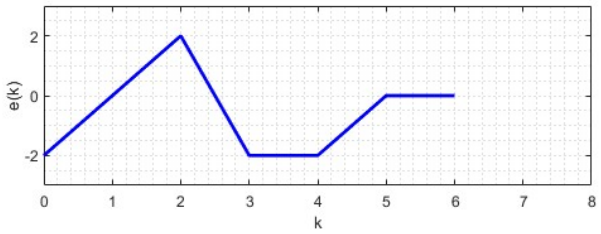
$w_2(3) = w_2(2) - 2\eta(-1) = 0.8 + 1 = 1.8$

$\theta(3) = \theta(2) - 2\eta(-1) = 0.4 + 1 = 1.4$

```
clear; close all; clc
P=[-1 -1 -1 -1
   -1 -1 1 1
   -1 1 -1 1];
T=[-1 -1 -1 1];
w0=[-0.6 0.3 -0.2];
alpha=0.5;
```

```
% ------------------------------------------
% patrón (1,-1)
% ------------------------------------------
% activación (n)
n1=w0*P(1:3,3);
a1=hardlims(n1);
e1=T(3)-a1;
if e1~=0
   w1=w0+alpha*e1*P(1:3,3)';
else
   w1=w0;
end
w1; % 0.4000 -0.7000 0.8000
```

```
k=0:6;
error=[e1 e2 e3 e4 e5 e6 e7];
subplot(211)
plot(k,error)
axis([0 8 -3 3])
grid minor
xlabel('k')
ylabel('e(k)')
```

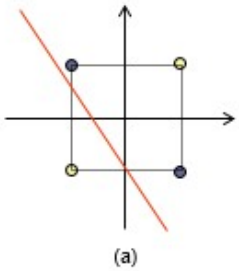


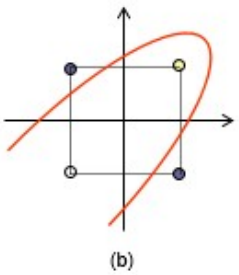
Limitaciones del Perceptron

- ¿Un conjunto cualquiera de patrones de entrenamiento, el Perceptron aprenderá a clasificarlos correctamente? Por ejemplo:

Problema XOR

Entradas	Salidas
(1, 1)	−1
(1, −1)	1
(−1, 1)	1
(−1,−1)	−1





Deducción del Algoritmo (1/3)

- Ajuste del valor del valor de umbral como un 'peso' más.

$$w_1x_1 + w_2x_2 = \theta$$

$$w_1x_1 + w_2x_2 - \theta = 0$$

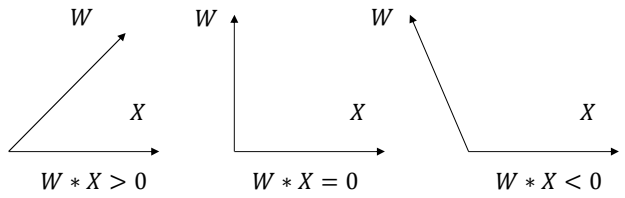
$$w_1x_1 + w_2x_2 + \theta(-1) = 0$$

- Para entender mejor al procedimiento de entrenamiento, vamos a introducir una representación vectorial.

$$W * X = 0$$

Deducción del Algoritmo (2/3)

- (producto punto = qué tan alineados están)

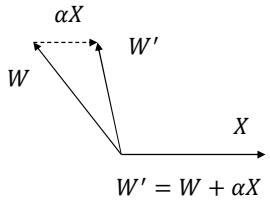


$W * X \geq 0, \quad y = 1$ $\quad\quad$ $W * X < 0, \quad y = 0$

- ¿Qué pasa si no obtenemos el resultado deseado?, entonces se requiere de un ajuste. ¿Cómo se realiza?.

Deducción del Algoritmo (3/3)

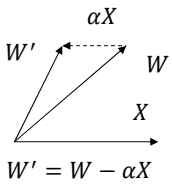
- Si el resultado es 0 en lugar de 1:



Donde

$0 < \alpha < 1$

- Si el resultado es 1 en lugar de 0:



- En resumen: $W' = W + \alpha(t - y)X$ $\Delta W = \alpha(t - y)X$

- A esto se le llama la regla de aprendizaje del perceptron simple. El parámetro α es la razón de aprendizaje.

Medición del Desempeño

- Qué es el factor $(t - y)$? Para el perceptron, hay corrección de error, o sólo se detecta la presencia del error.
- En el entrenamiento se logra encontrar la convergencia del error?
- En general, en el entrenamiento es usual encontrar el error 'SSE' o suma de los cuadrados de los errores:

$$sse = \sum_{k=1}^{Q} e(k)^2 = \sum_{k=1}^{Q} (t(k) - y(k))^2$$

- Cuando se da un entrenamiento en 'batch' o por lotes. Se dice que un lote es el conjunto de vectores de entrada que se muestran a la red para que ésta los procese en conjunto (en lugar de presentarlos vector por vector).

Época y Umbral

- Se le llama época a cada iteración de la red por el lote de entradas en la que haya ajuste de variables. El ajuste de variables se hace después de la presentación de vectores de entrada individuales o por lotes.
- La variable θ también es llamada elemento de tendencia o 'threshold' porque, es el que mueve el hiperplano de decisión a lo largo del eje 'x'. A esta variable se le denomina en muchas ocasiones con el símbolo 'b' denominándose 'Bias' que a diferencia del threshold es de característica de pesos excitatorio.

Ejercicio

1. Considere el conjunto de pares de entrenamiento:

$$Q = \left\{ \left(\begin{bmatrix} -1 \\ 1 \end{bmatrix}, 1 \right), \left(\begin{bmatrix} -1 \\ -1 \end{bmatrix}, 1 \right), \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, 0 \right), \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, 0 \right) \right\}$$

Siendo los vectores de pesos iniciales y bias es: $W = \begin{bmatrix} -5 & 8 \end{bmatrix}$ y $b = \begin{bmatrix} -2 \end{bmatrix}$.

a) Dibuje el estado inicial de la topología de la red modificada.

b) Simule el estado inicial de la red y dibuje su espacio de patrones.

c) La red debe ser entrenada, para ello necesita procesar el algoritmo de aprendizaje supervisado *perceptron aprende la regla* del TRN. ¿Cuál es el número mínimo de épocas?. ¿Qué pesos finales se obtienen del entrenamiento?.

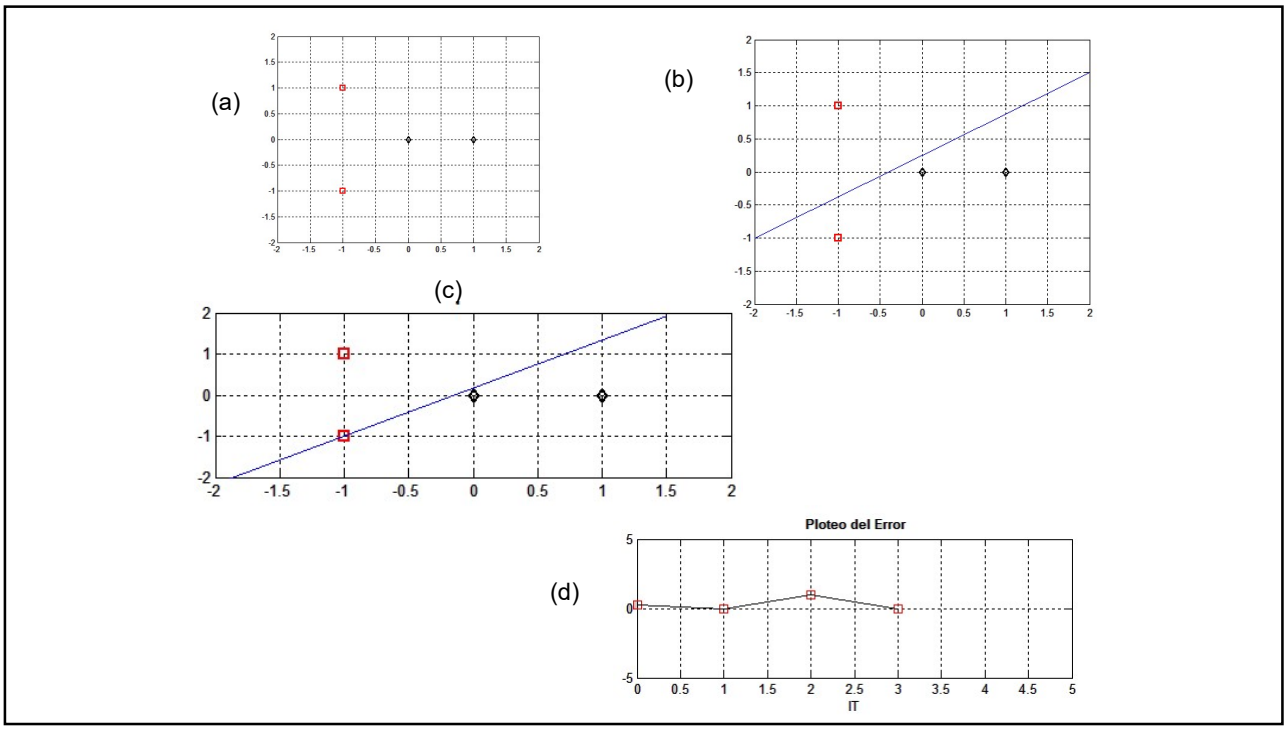
d) Asumiendo un entrenamiento exitoso. Dibujar la evolución del error vs épocas.

e) Ahora se desea validar la performance de la red, para ello se le presentan el vector de prueba:

$$P_{\text{test}} = \begin{bmatrix} -2 & 1 & 0 & -1 \\ 0 & 1 & 1 & -2 \end{bmatrix}$$

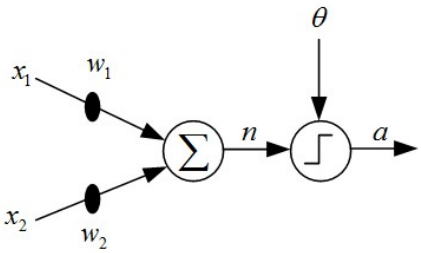
¿Cuál es el estado final de la red?. Justificar su respuesta en forma clara.

f) Escriba un algoritmo computacional usando el TRN de `MATLAB`, de tal modo que verifique la solución a su problema en los items anteriores.



Ejercicio

- Supongamos que un Perceptron (ver figura) de dos entradas tiene pesos iniciales 0, 0.4 y umbral 0.3. Se requiere que este Perceptron aprenda la función lógica AND. Suponga una razón de aprendizaje α de 0.25. Usando el algoritmo anterior complete la tabla hasta que encuentre convergencia.



- ¿Cuántas iteraciones se requieren para la convergencia?
- ¿Es lo mismo iteraciones y épocas?
- ¿Cuáles son los valores de convergencia de los pesos y el umbral?
- Defina algebraicamente el hiperplano de decisión.
- Demuestre gráficamente que éste hiperplano es un límite apropiado para la distinción de clases y que el vector de pesos y el hiperplano son ortogonales.
- Entrenar con diferentes umbrales implica una modificación en la característica de los pesos. En este caso es indiferente para el entrenamiento de la red perceptron?.

$w_1 = 0.0$
$w_2 = 0.4$
$\theta = 0.3$
$\alpha = 0.25$
t = Función AND

$w' = w + \alpha(t-y)x$
ó $\Delta w_i = \alpha(t-y)x_i$
$a = 1$, si $(x_1 w_1 + x_2 w_2) \geq \theta$
$a = 0$, si $(x_1 w_1 + x_2 w_2) < \theta$

x_1	x_2	w_1	w_2	n	θ	$n \geq \theta$	y	T	$T-y$	Δw_1	Δw_2	$\Delta\theta$
0	0	0	0.4	0	0.3	F	0	0	0	0	0	0
0	1	0	0.4	0.4	0.3	T	1	0	-1	0	-0.25	0.25
1	0	0	0.15		0.55			0				
1	0							1				

w_1	w_2	θ	x_1	x_2	n	a	t	α(t-y)	Δw_1	Δw_2	$\Delta\theta$
0	0.4	0.3	0	0	0	0	0	0	0	0	0
0	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	0.25
0	0.15	0.55	1	0	0	0	0	0	0	0	0
0	0.15	0.55	1	1	0.15	0	1	0.25	0.25	0.25	-0.25

it	w_1	w_2	θ	x_1	x_2	n	y	t	$\alpha(t-y)$	Δw_1	Δw_2	$\Delta\theta$
1	0	0.4	0.3	0	0	0	0	0	0	0	0	0
2	0	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	0.25
3	0	0.15	0.55	1	0	0	0	0	0	0	0	0
4	0	0.15	0.55	1	1	0.15	0	1	0.25	0.25	0.25	-0.25
5	0.25	0.4	0.3	0	0	0	0	0	0	0	0	0
6	0.25	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	0.25
7	0.25	0.15	0.55	1	0	0.25	0	0	0	0	0	0
8	0.25	0.15	0.55	1	1	0.4	0	1	0.25	0.25	0.25	-0.25
9	0.5	0.4	0.3	0	0	0	0	0	0	0	0	0
10	0.5	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	0.25
11	0.5	0.15	0.55	1	0	0.5	0	0	0	0	0	0
12	0.5	0.15	0.55	1	1	0.65	1	1	0	0	0	0
13	0.5	0.15	0.55	0	0	0	0	0	0	0	0	0
14	0.5	0.15	0.55	0	1	0.15	0	0	0	0	0	0
15	0.5	0.15	0.55	1	0	0.5	0	0	0	0	0	0
16	0.5	0.15	0.55	1	1	0.65	1	1	0	0	0	0

Valores finales: w1=0.5, w2=0.15, theta=0.55.
Eq. recta: x2 = -3.333*x1+3.667