REDES NEURONALES ARTIFICIALES: TEORIA, APLICACIONES E IMPLEMENTACIONES.

Capítulo 2: El 'Perceprton' y la Separabilidad Lineal.

Vamos a tomar el 'modelo artificial más real' del capítulo anterior y verlo con más detalle, pero antes haremos unas notas históricas.

El modelo que de la neurona real que vimos en el capítulo anterior se debe a Warren McCulloch (neurofisiólogo) y Walter Pitts (matemático) de 1943. Este se originó como un modelo de neuronas reales que era plausible aunque simple. Introdujo los conceptos de pesos que implicaban una influencia continua de la sinapsis en la neurona. Algo también inovador fue la idea de pesos negativos y su influencia inibidora. Después de esto hubo varias contribuciones al modelo, principalmente en la manera de 'entrenar' la red. Contribuciones importantes se deben a el Biólogo Donald Hebb en 1949 en aspectos de bases biológicas; Bernard Widrow en 1962 propuso una regla de aprendizaje importante conocida como la regla Widrow-Hoff. Una prueba de convergencia importante para el modelo MCP fue desarrollada y publicada por Frank Rosenblatt en 1958 y 1962 en su obra 'Perceptrons'. Este teorema se llama el teorema de convergencia del perceptron. Rosenblatt no sólo presentó un algoritmo mediante el cual su red podía ser entrenada, sino ademas probó que si el problema estaba dentro del conjunto de problemas 'resolvibles' por su red, su algoritmo convergería a una solución. Este trabajo impulsó mucho el estudio de las RNAs en sus inicios porque además se presentó como un método de reconocimiento de patrones (herramineta computacional) con logros importantes, no sólo como modelo neuronal.

Una de las características importantes de esta red es que aunque es capaz de resolver problemas interesantes, existe un grupo importante de problemas que no puede resolver. Por esta razón este tipo de redes está casi en desuso ahora pero no se debe menospreciar la contribución de Rosenblatt a esta área de conocimiento. La gran limitante del perceptron es que sólo se puede usar para problemas de clasificación que sean separables linealmente en la salida. Esto lo vamos a explicar a detalle y con ejemplos pero primero vamos a requerir de unas definiciones:

Un **hiperplano** es un objeto de dimensión $n-1$ que actúa en un espacio de dimensión n.

En general, un perceprton de *n* entradas puede ejecutar cualquier función qu esté definida por un hiperplano que corte el espacio de dimensión *n*.

Ejemplos:

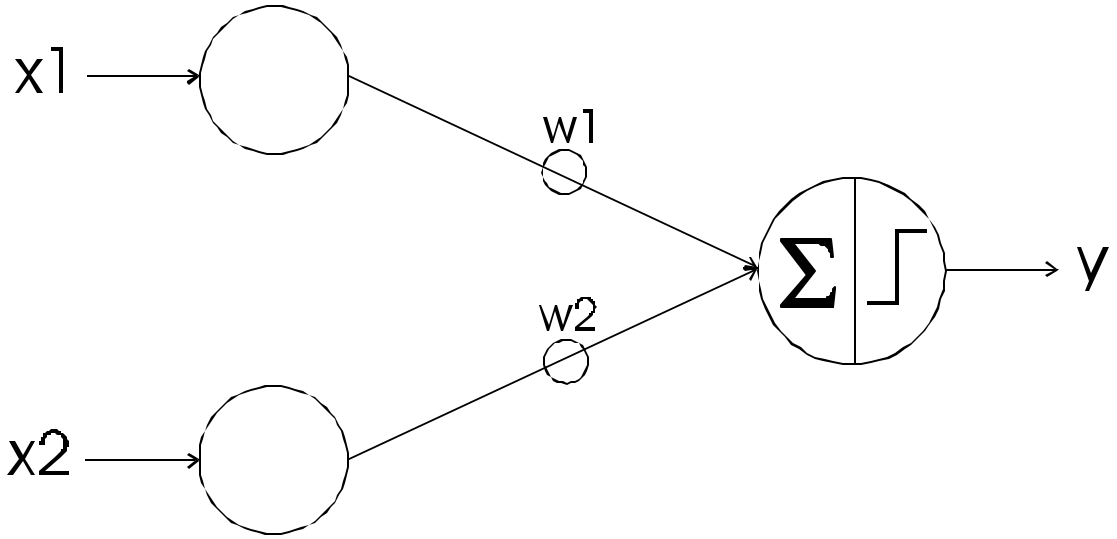

Fig. 2. 1 Ejemplo gráfico de un perceptron simple.

$$y = 1 \text{ si } (x_1w_1 + x_2w_2) \geq \theta$$

$$y = 0 \text{ si } (x_1w_1 + x_2w_2) < \theta$$

sindo θ el valor del umbral.

Supongamos que tenemos los siguientes vectores X1 y X2 y una función propuesta Y como sigue:

X1: 0 0 1 1

X2: 0 1 0 1

Y: 0 1 1 1

Para ver si esta función es realizable, buscamos el hiperplano que separe las clases de salida. En este caso, como son salidas binarias, solo hay dos clases que bien pueden verse como activación (1) o no activación (0) de la neurona. En la siguiente figura vemos que esta función sí es realizable, porque las clases se pueden separar linealmente.

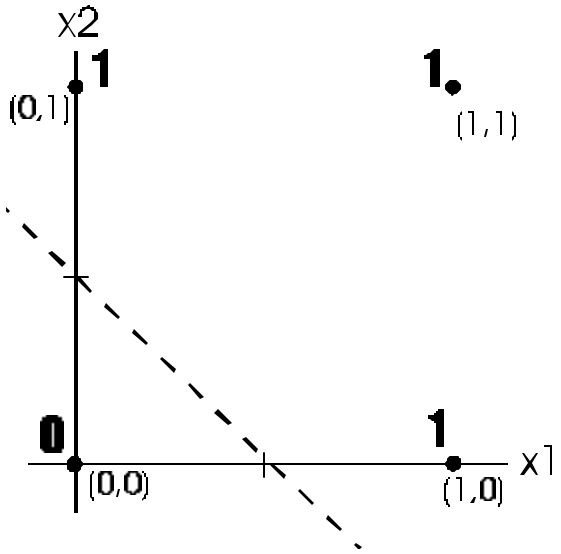

Fig. 2. 2 Ejemplo de una función separable linealmente.

donde el cero o uno representan la tercera dimensión (la función Y), que es binaria.

Ya que una condición crítica para la clasificación ocurre cuando la activación es igual al valor de umbral, resulta útil analizar esta condición. Poniendo la activación igual al umbral nos dá, en el caso general,

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

que para el caso que estamos manejando resulta,

$$w_1 x_1 + w_2 x_2 = \theta$$

que, arreglando, queda como,

$$x_2 = -\left(\frac{w_1}{w_2}\right) x_1 + \left(\frac{\theta}{w_2}\right)$$

y esto está en la forma,

$$x_2 = m x_1 + b$$

Esta, como sabemos es la ecuación general de una recta con pendiente m y cruce con el eje y (x_2) igual a b. Si hacemos w_1=1, w_2=1 y b=0.5 tenemos la línea recta que dibujamos anteriormente

(con theta = 0.5). Otra característica interesante es que el vector de pesos (w1,w2) es perpendicular a esta línea y apunta en el sentido del área de activación de la neurona.

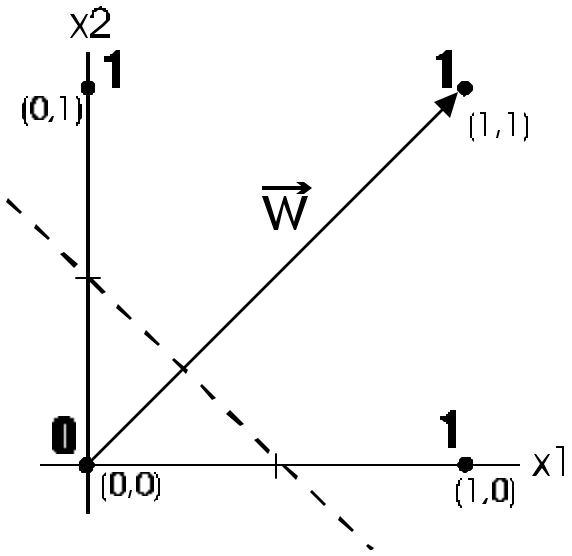

Fig. 2. 3 Ilustración del vector de pesos, junto con las areas de clasificación.

Las dos clases estan separadas por esta línea de decisión. Si estamos en un espacio de dimensión mayor (n>2), la forma geométrica en el límite de decisión sera un hiperplano de dimensión n-1. Esta restricción hace que este clasificador sea llamado un clasificador lineal.

Entrenamiento.

Hemos deducido que las variables que definen la convergencia de la 'red' son el vector de pesos (w1,w2) y el valor del umbral. Estas variables son de distinta naturaleza por lo que habría que buscar metodos apropiados para encontrar los valores de convergencia para cada una. Sin embargo esto se puede facilitar si reordenamos nuestras ecuaciones de la siguiente manera, tomando el caso bidimensional:

$$w_1 x_1 + w_2 x_2 - \theta = 0$$

En este caso podemos pensar que existe otra entrada x3 fija a 1 y que se tiene un peso 'theta' por definir. Asi, se puede diseñar un algoritmo de entrenamiento que sólo considere buscar los pesos adecuados para la convergencia, y se tiene:

$$w_1 x_1 + w_2 x_2 + \theta(-1) = 0$$

Representación y álgebra vectorial.

Para entender mejor el procedimiento de entrenamiento, vamos a introducir un sistema de representación vectorial del problema.

Nuestra ecuación,

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

se puede representar en términos vectoriales de la siguiente manera,

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}=\mathbf{q}$$

La parte izquierda de la ecuación es el producto punto entre dos vectores, lo cual tiene implicaciones gráficas muy ilustrativas en este caso. En pocas palabras el producto punto es una indicación de que tan alineados están los vectores el uno con el otro.

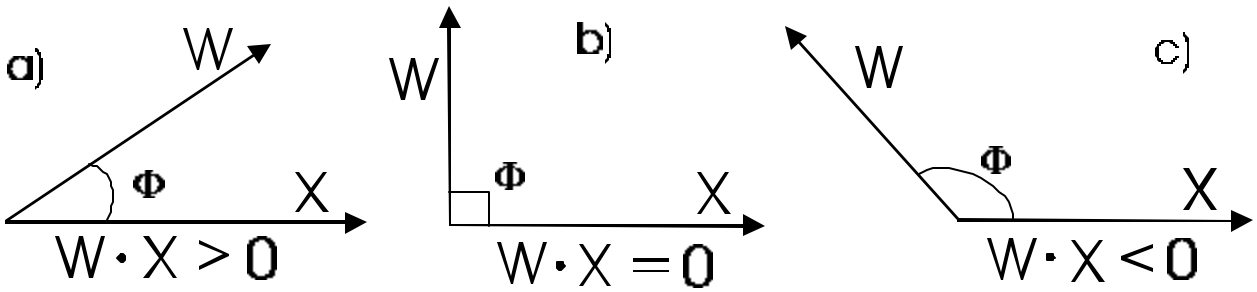

Fig. 2. 4 Análisis del resultado del producto punto, vectorialmente.

El entrenar a una neurona, como hemos visto, consiste en ajustar los valores de los pesos (o el vector de pesos) y el valor del umbral de tal manera que se realice la clasificación deseada. Para la neurona a entrenar,

$$w \cdot x \geq 0 \quad \Rightarrow \quad y=1$$

$$w \cdot x < 0 \quad \Rightarrow \quad y=0$$

Haciendo $w \cdot x = 0$ define el hiperplano de decisión. Este plano es ortogonal al vector de pesos que ahora incluye la adición del valor de umbral.

En términos vectoriales, el conjunto de entrenamiento de la neurona, consiste de un conjunto de pares {**v**,***t***}, donde **v** es un vector de entrada y ***t*** es la clase deseada o salida (‘1’ o ‘0’) a la que **v** pertenece. Este tipo de entrenamiento es de tipo supervisado porque se indica cual es el valor deseado

de la salida de la red. Supongamos, en un primer caso, que se tiene un vector inicial de pesos, **w**, con el cual la salida es '0', y que el valor de la salida deseada para esta entrada es '1'. De acuerdo con la ecuaciones de arriba, para producir un '0' la activación debió haber sido negativa cuando tenía que ser positiva. Aquí se tiene el caso de la figura 4 c). Para corregir este problema, se debe afectar el vector de pesos de tal manera que quede apuntando más hacia el vector de entradas. Esto debe hacerse con cuidado ya que no queremos hacer una rotación muy drástica porque se perdería todo el terreno ganado con el aprendizaje acumulado hasta ese momento. Podemos obtener el resultado que queremos si añadimos una fracción de **v** a **w** para producir un nuevo vector **w'**, por lo que tenemos,

$\mathbf{w'} = \mathbf{w} + \alpha\mathbf{v}$

donde $0<\alpha<1$

Esto se ilustra en la siguiente figura:

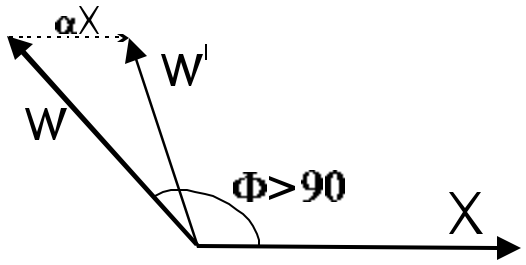

Fig. 2. 5 Adaptación mediante suma del vector de pesos por una fracción del vector de entradas.

Es importante notar que los vectores v y w son vectores que incluyen a TODAS las entradas y TODOS los pesos respectivamente y que esta ilustración vectorial es útil por facilitar entender el concepto de adaptación de los pesos unicamente. Esta operación se repite iterativamente hasta lograr que el vector de pesos sea ortogonal al hiperplano de decisión.

Ahora supongamos que el caso es que t = 0 pero y = 1. Esto significa que la activación era positiva cuano debió ser negativa. El caso de ahora es el que se ilustra en la fig. 2.4 a). Ahora lo que necesitamos es rotar a **w** alejándolo de **v**, lo que se logra restando de **w** una fracción de **v**. Esto se ejemplifica gráficamente en la figura 2.6.

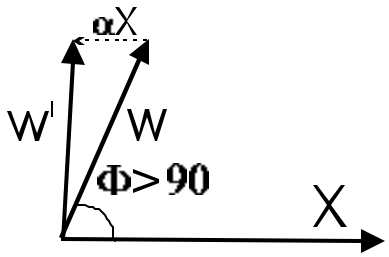

Fig. 2. 6 Adaptación mediante resta del vector de pesos por una fracción del vector de entradas.

En ambos casos los esfectos de suma y resta se pueden resumir en la siguiente ecuación:

$$\mathbf{w'}=\mathbf{w}+\alpha(t-y)\mathbf{v}$$

Esto puede ser escrito en términos del cambio en el vector de pesos Dw o en términos de sus componentes,

$$Dw_i = a(t-y)v_i$$

Esto es el primer ejemplo que vemos de una regla de aprendizaje o regla de entrenamiento. Al parámetro se le llama razón de aprendizaje. Esta regla de aprendizaje puede ser incorporada en un algoritmo de entrenamiento como sigue (en pseudo-código):

repetir
    para cada par en los vectores de entrenamiento (**v**,*t*)
        evaluar la salida y_i cuando v_i es la entrada al perceptron
        si $y \neq t$, entonces
            forme un nuevo vector de pesos **w'** de acuerdo a la ecuación correspondiente
        de otra manera,
            no haga nada
        fin (del si)
    fin (del para)
hasta que $y = t$ para todos los vectores.

A este algoritmo se le conoce normalmente como el algoritmo de aprendizaje 'Perceptron'. Originalmente los valores de los pesos fueron limitados a valores entre -1 y 1.

EJEMPLO #1:Veamos ahora un ejemplo. Supongamos que un perceptron 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 1.1 hasta que encuentre convergencia. El percepton usado es el que muestra la figura 2.7.

$w_1 = 0.0$

$w_2 = 0.4$

$q = 0.25$

t = Función AND

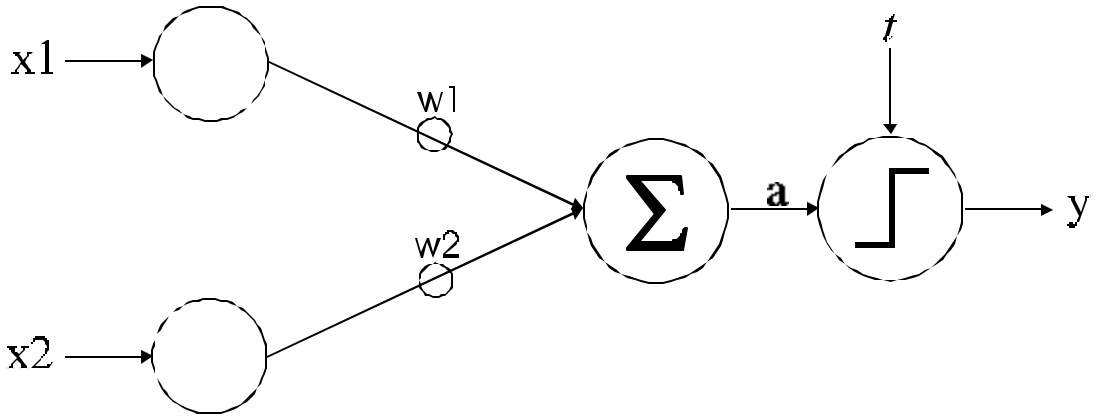

Fig. 2. 7 Perceptron de demostracioón de entrenamiento con punto de prueba 'a' y entrada de señal deseada 't'.

Tabla 1. 1 Ejemplo del proceso de aprendizaje del perceptron de la figura 2.7.

w_1	w_2	q	x_1	x_2	a	y	t	$a(t-y)$	dw_1	dw_2	δθ
0.0	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
0.25	0.4	0.3	0	0	0	0	0	0	0	0	0
0.25	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	-0.25
0.25	0.15	0.55	1	0	0.25	0	0	0	0	0	0
0.25	0.15	0.55	1	1	0.4	0	1	0.25	0.25	0.25	-0.25
0.5	0.4	0.3	0	0	0	0	0	0	0	0	0
0.5	0.4	0.3	0	1	0.4	1	0	-0.25	0	-0.25	0.25
0.5	0.15	0.55	1	0	0.5	0	0	0	0	0	0
0.5	0.15	0.55	1	1	0.65	1	1	0	0	0	0
0.5	0.15	0.55	0	0	0	0	0	0	0	0	0
0.5	0.15	0.55	0	1	0.15	0	0	0	0	0	0
0.5	0.15	0.55	1	0	0.5	0	0	0	0	0	0
0.5	0.15	0.55	1	1	0.65	1	1	0	0	0	0

Después de haber terminado, responda a las siguientes preguntas:

1) Cuantas iteraciones se requieren para la convergencia? [10]

2) Cuales son los valores de convergencia de los pesos y el umbral? [w1=0.5, w2=0.15, theta=0.55]

3) Defina algebráicamente el hiperplano de decisión. [x2 = -(3.333)x1+3.667]

4) Demuestre gráficamente que éste hiperplano es un límite apropiado para la distinción de clases.[Ver gráfica]

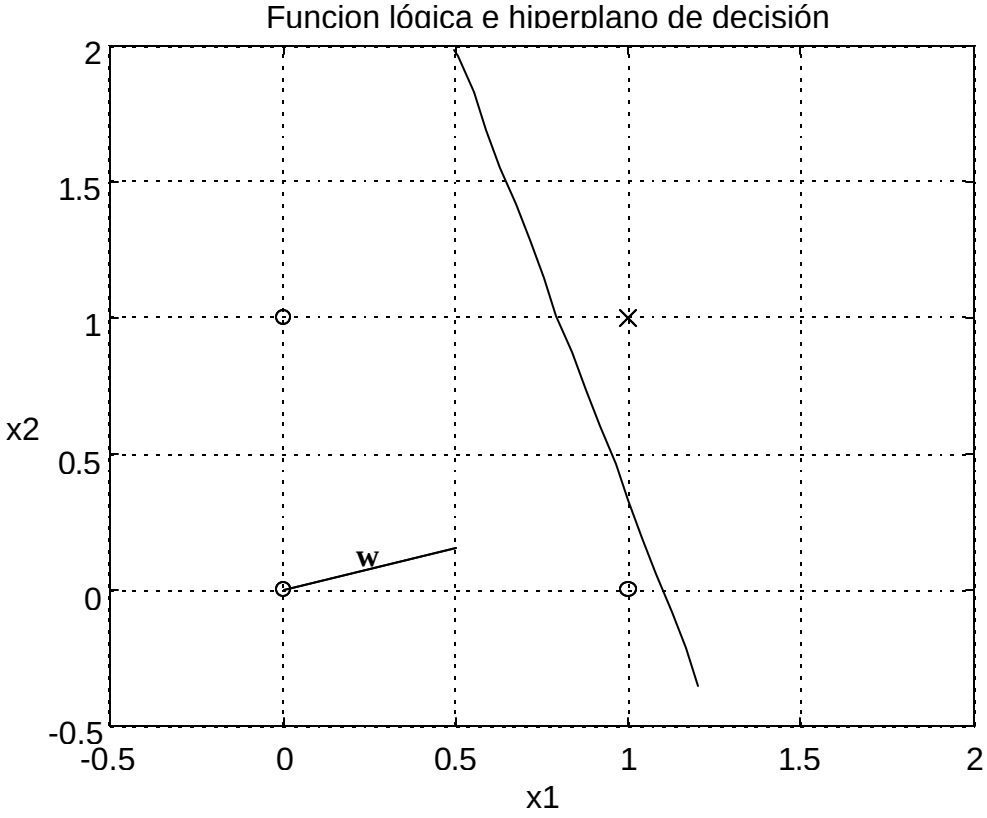

Fig. 2. 8 Gráfica del hiperplano de decisión resultante y el vector de pesos para el ejemplo #1.

EJEMPLO #2: Un ejemplo alternativo, e interesante en vista de que el hiperplano resultante cae sobre puntos a clasificar, es el de aprender la función siguiente:

x_1	x_2	t
0	0	0
0	1	1
1	0	0
1	1	0

Usando w1 = 0.25, w2 = -0.25, θ = 0.25 y α = 0.25 (valores al azar !?) determinar los valores de convergencia y el hiperplano de decisión.

Respuesta: Después de 11 iteraciones a mano, el algoritmo converge a w1=-0.25, w2=0.25 y θ=0. Qué pasa con los puntos por los que pasa la recta?, Cual es la implicación de que θ=0?

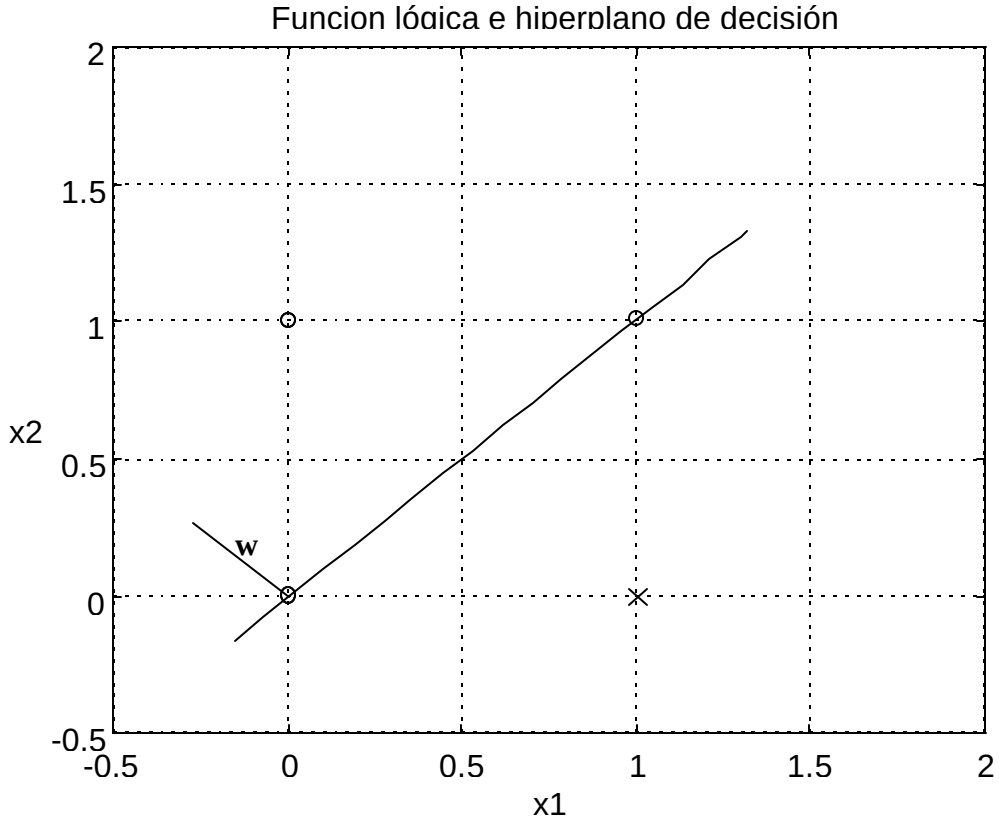

Fig. 2. 9 Gráfica del hiperplano de decisión resultante y el vector de pesos para el ejemplo #2.

En realidad los perceptrones pueden ser usados para entradas continuas no sólo digitales, como lo hemos estado usando. Esta se ha hecho sólo por claridad pero como veremos en las demostraciones de MATLAB, las aplicaciones son más amplias.

AHORA SE VERAN VARIOS EJEMPLOS EN BASE A LOS 'DEMOS' DE **MATLAB**. lAS OPCIONES SON DEMOP1, DEMOP2, DEMOP3, DEMOP4, DEMOP5, DEMOP6, DEMOP7, DEMOP8 Y DEMOP9.

PERCEPTRON Basic functions.

simup - Simulate perceptron layer.

initp - Inititialize perceptron layer.

trainp - Train perceptron layer with perceptron rule.

trainpn - Train perceptron with normalized perceptron rule.

Transfer functions.

hardlim - Hard limit transfer function.

hardlims - Symmetric hard limit transfer function.

Learning rules.

learnp - Perceptron learning rule.

learnpn - Normalized perceptron learning rule.

Plotting functions.

plotpv - Plot 2-input vectors.

plotpc - Add perceptron classification line to existing plot.

Demonstrations

demop1 - Classification with a 2-input perceptron.

demop2 - Classification with a 3-input perceptron.

demop3 - Classification with a 2-neuron perceptron.

demop4 - Outlier input vectors.

demop5 - Normalized perceptron rule.

demop6 - Linearly non-seperable vectors.

demop7 - Classification with a two-layer perceptron.