PONTIFICIA
UNIVERSIDAD
CATÓLICA DE
VALPARAÍSO

Introducción al Deep Learning

Dr. Ing. Gabriel Hermosilla Vigneau

Redes Neuronales Artificiales
Parte 2

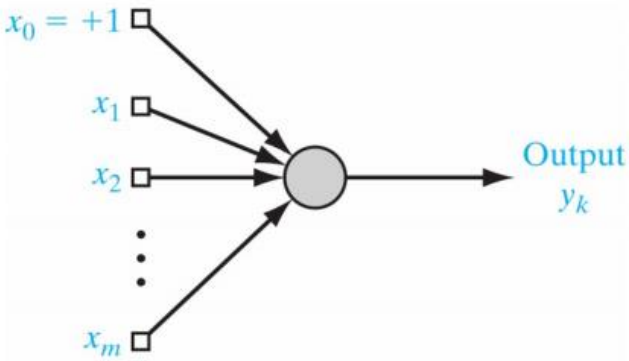
Modelo Perceptrón

- El perceptrón posee un algoritmo simple que, dado un vector de entrada x (x1, x2, ..., xn) a menudo llamado características de entrada o "características", produce salidas 1 (sí) ó 0 (no). Matemáticamente se define como :

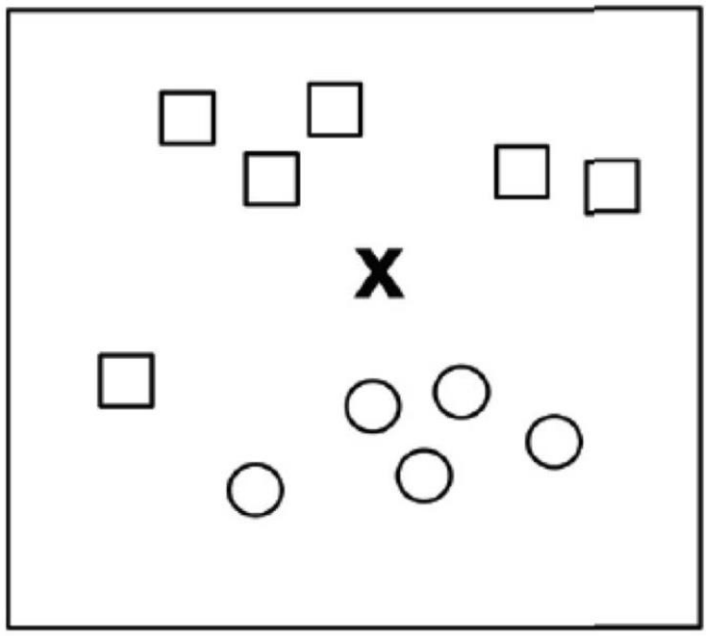
$$f(x) = \begin{cases} 1 & wx + b > 0 \\ 0 & otherwise \end{cases}$$

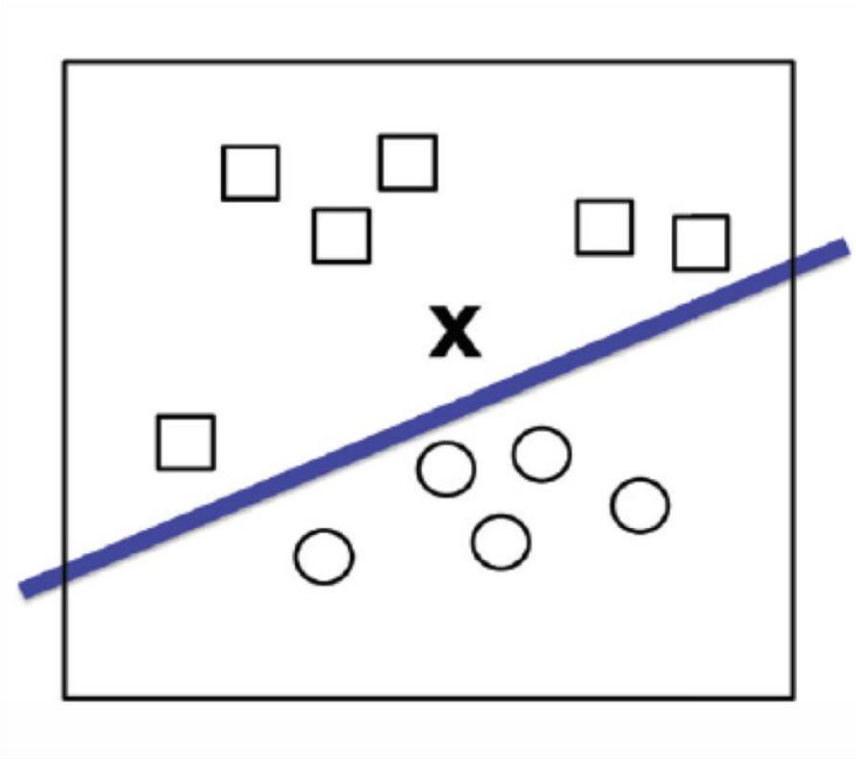
- Aquí, w es un vector de pesos neuronales, ωx es el producto punto $\sum_{j=1}^{m} w_j x_j$, *b* es un sesgo o bias. El término, ωx + *b* define un hiperplano separador que cambia de posición de acuerdo con la valores asignados a ω y *b*.
- Si x se encuentra sobre la línea recta, entonces la respuesta es positiva, de lo contrario es negativa. Un algoritmo muy simple!
- El perceptrón no puede expresar una respuesta "tal vez". Puede responder sí (1) o no (0 ó -1), si entendemos cómo definir ω y *b,* proveniente de un proceso de entrenamiento.

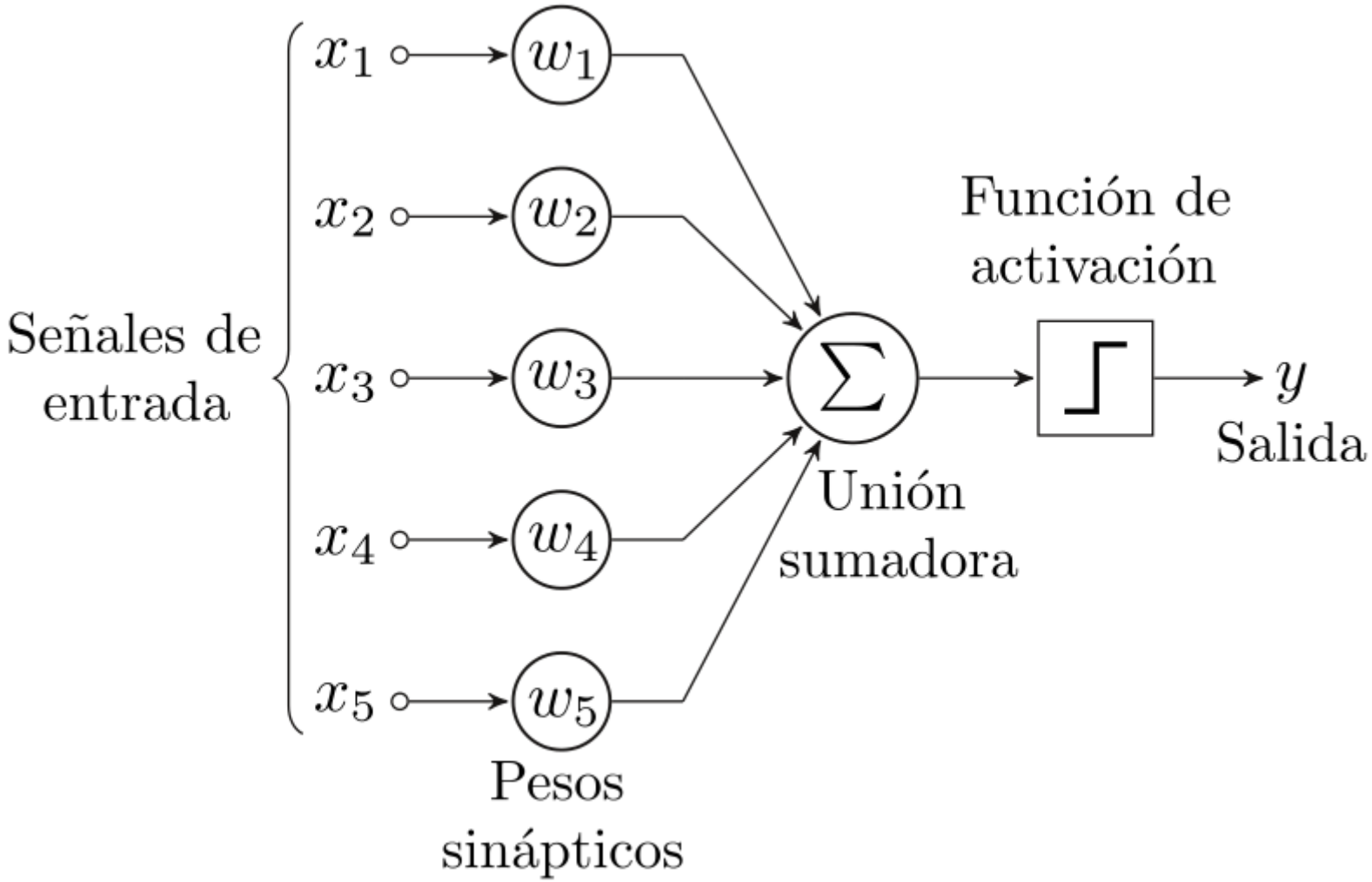
Modelo Perceptrón



$$y = w_1x_1 + w_2x_2 + b$$

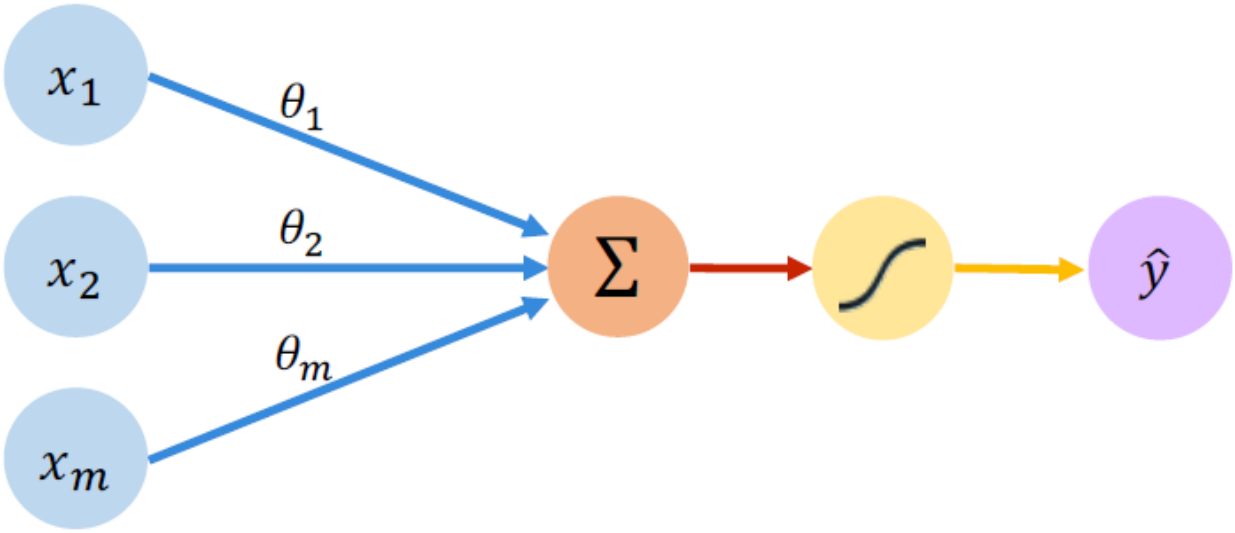






Modelo Perceptrón para 5 señales de entrada

Modelo Perceptrón



Inputs Weights Sum Non-Linearity Output

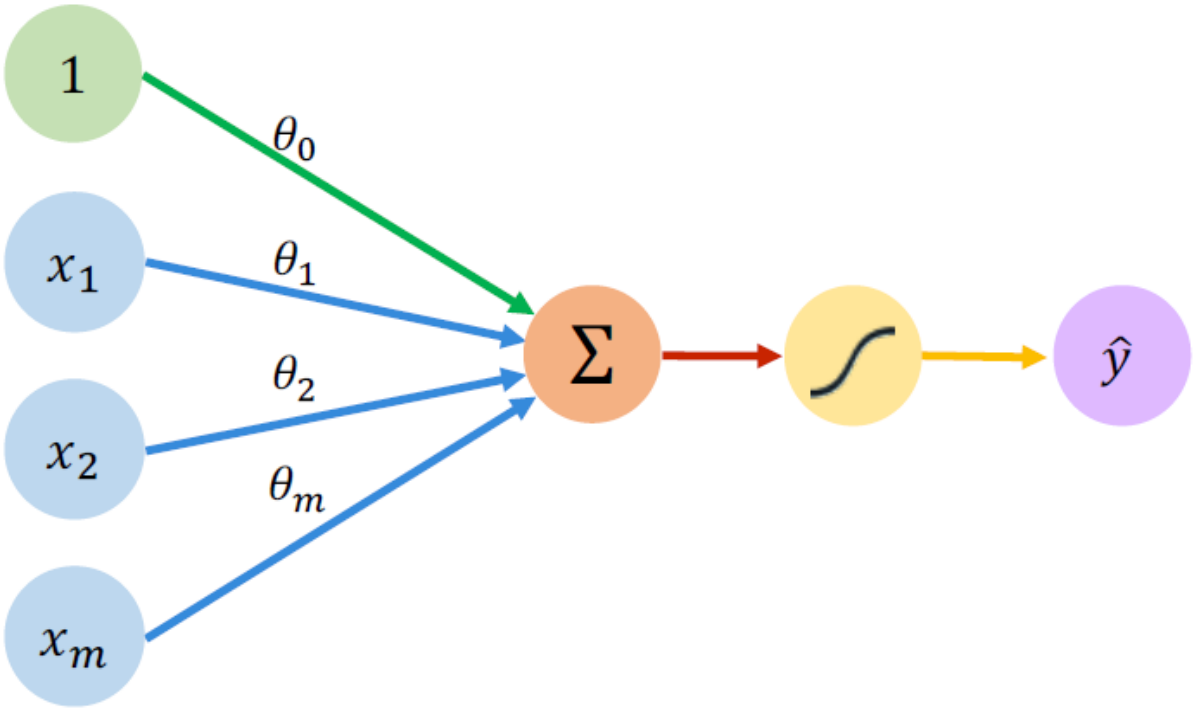
Output

Linear combination of inputs

$$\hat{y} = g\left(\sum_{i=1}^{m} x_i \; \theta_i\right)$$

Non-linear activation function

Modelo Perceptrón



Inputs Weights Sum Non-Linearity Output

Output

Linear combination of inputs

$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i \, \theta_i\right)$$

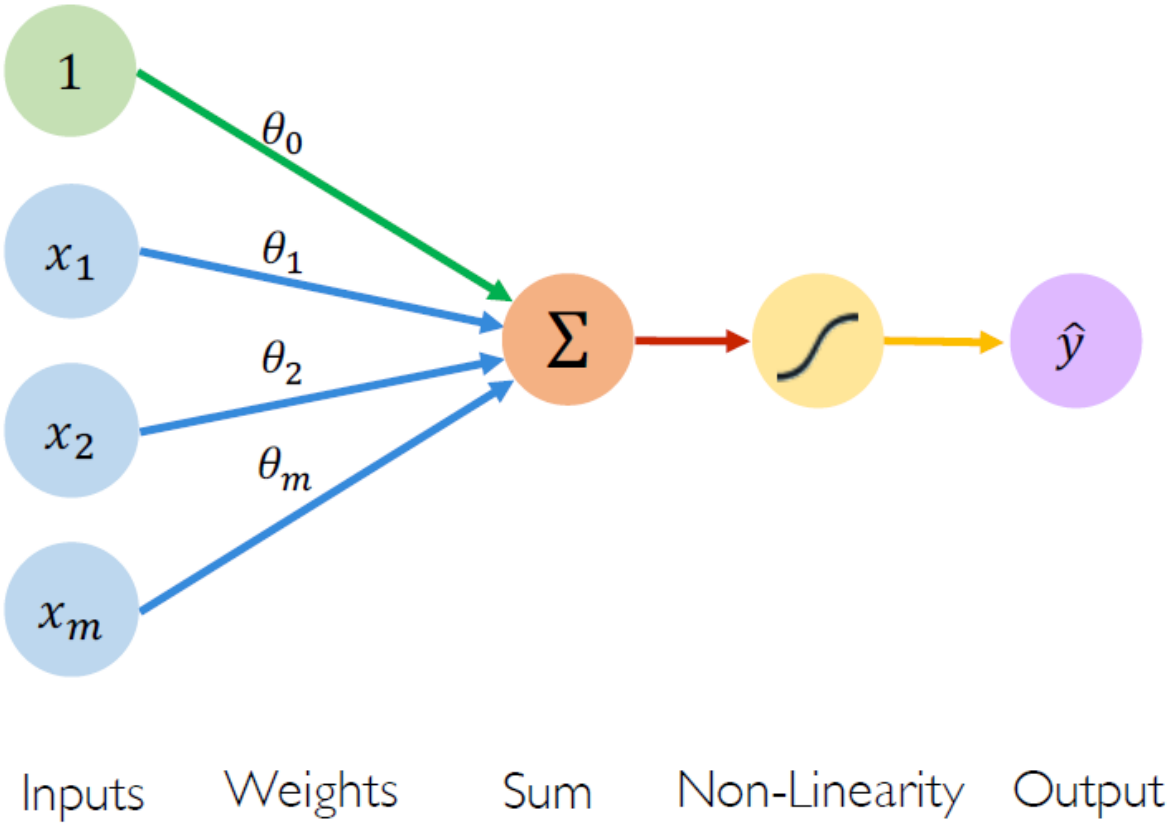
Non-linear activation function

Bias

$$\hat{y} = g\left(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$

Modelo Perceptrón

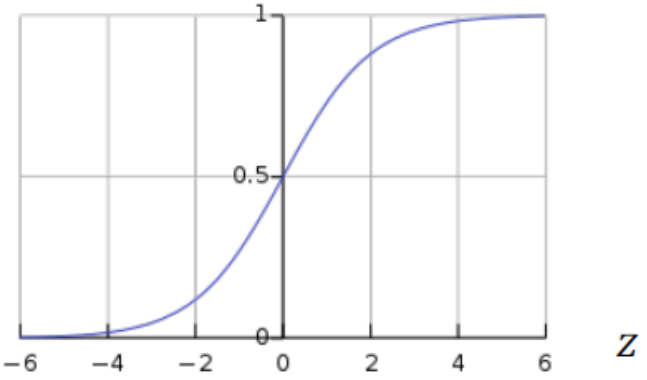


Activation Functions

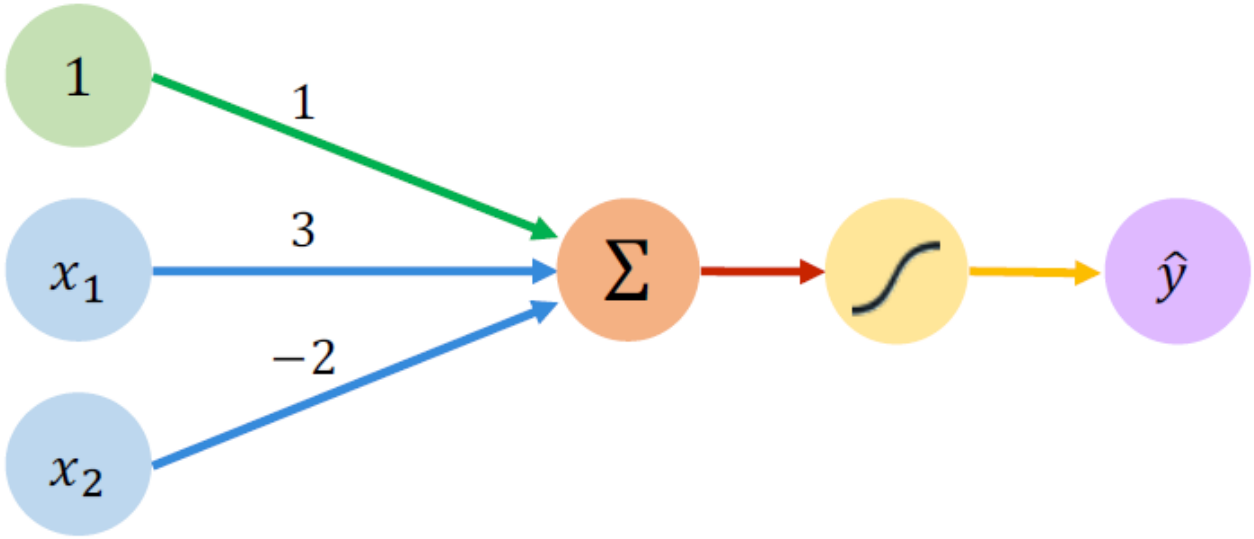
$$\hat{y} = g(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta})$$

- Example: sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



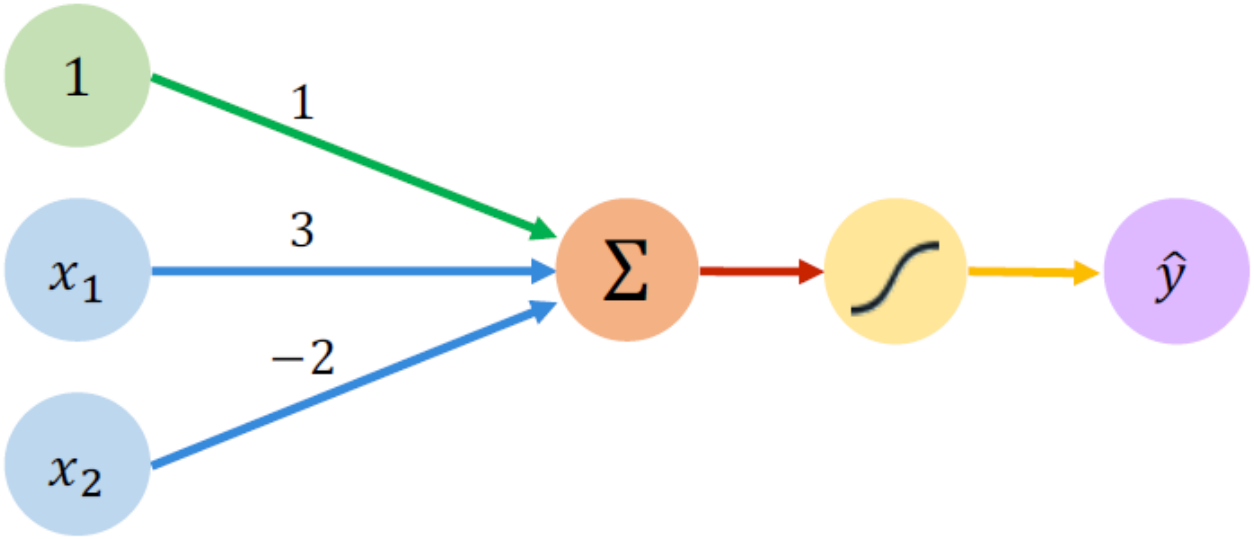
Ejemplo: Modelo Perceptrón



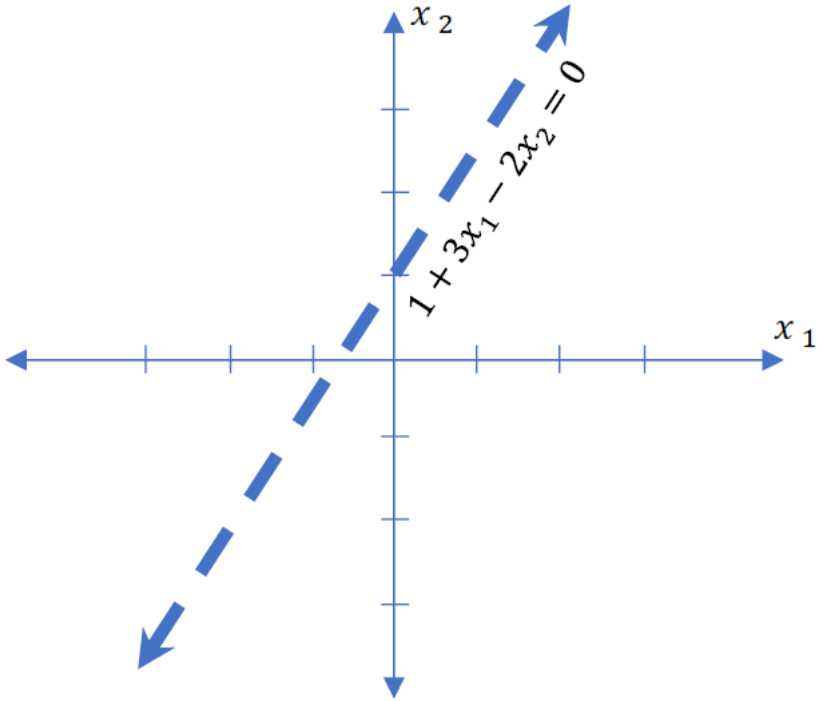
$$\theta_0 = 1 \text{ and } \boldsymbol{\theta} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$$

$$\begin{aligned} \hat{y} &= g\left(\theta_0 + \boldsymbol{X}^T \boldsymbol{\theta}\right) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g\left(1 + 3x_1 - 2x_2\right) \end{aligned}$$

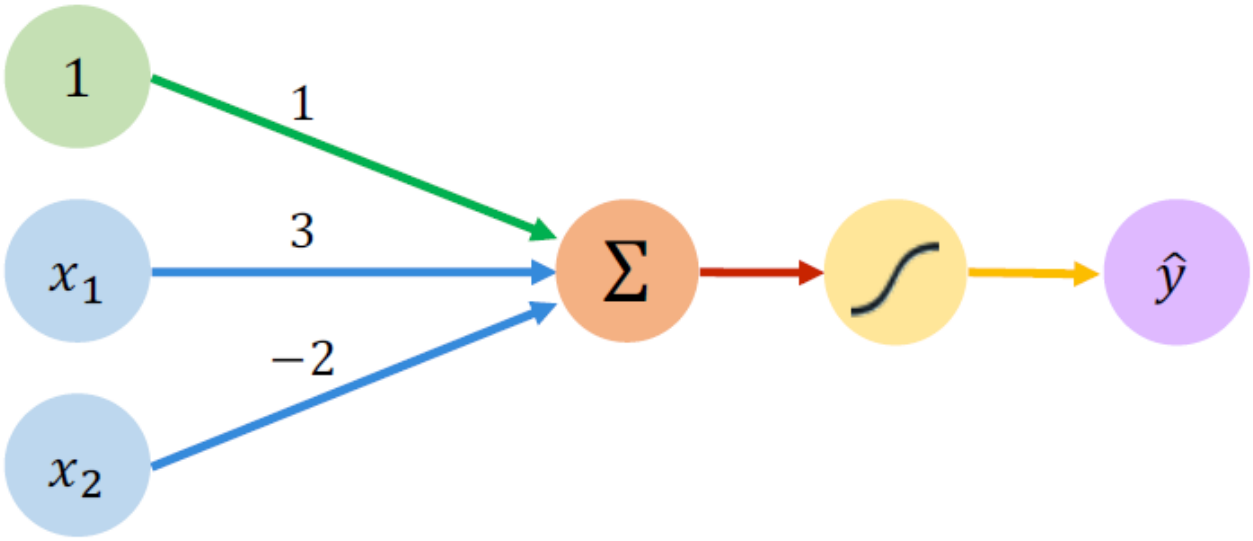
Ejemplo: Modelo Perceptrón

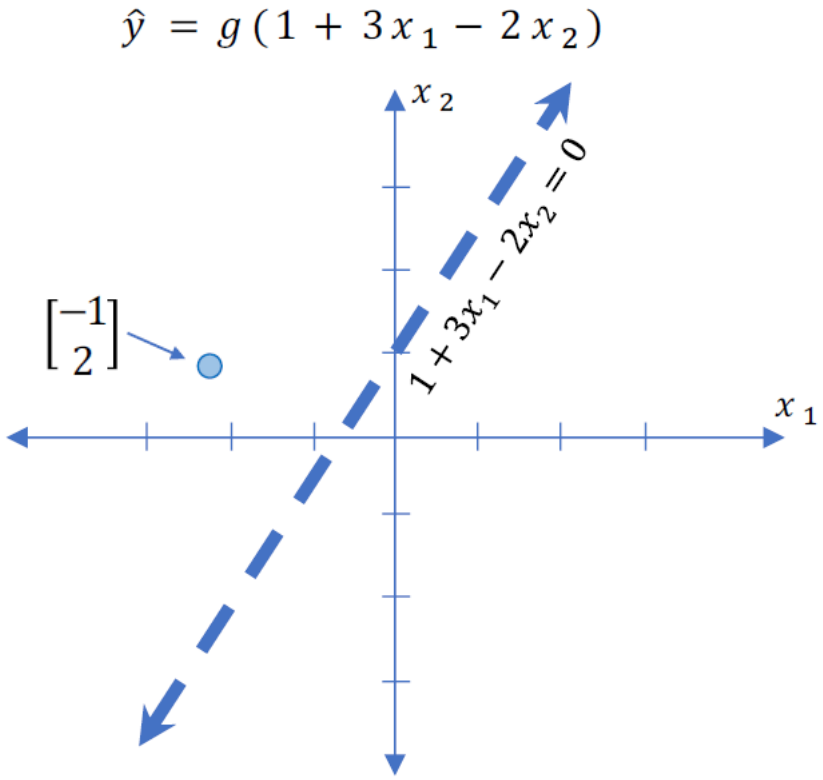


$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



Ejemplo: Modelo Perceptrón

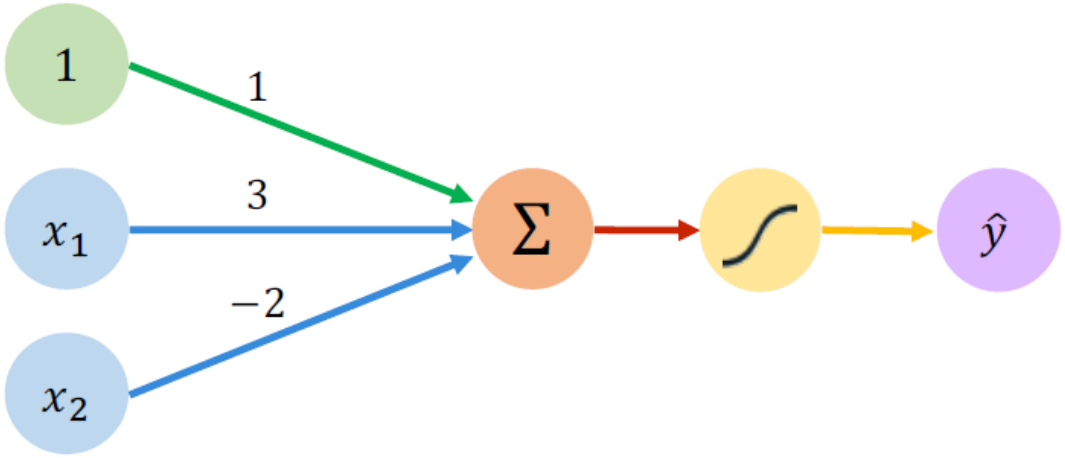


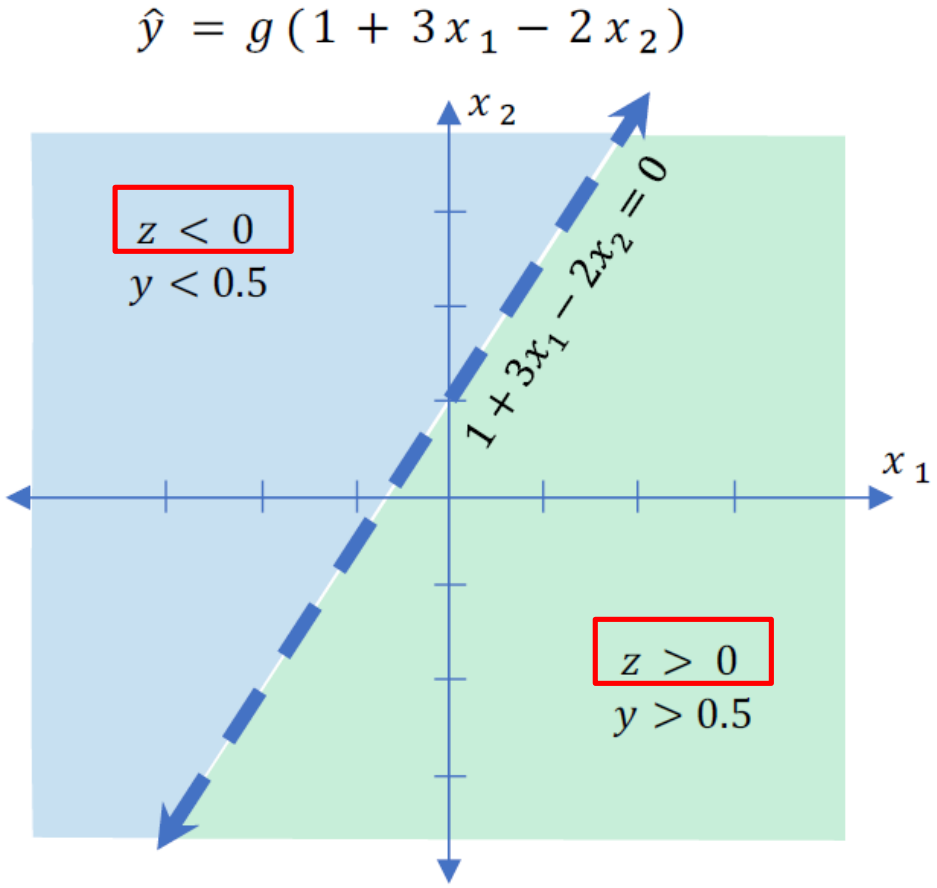


Si la entrada es $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$\hat{y} = g\left(1 + (3 * -1) - (2 * 2)\right)$$
$$= g(-6) \approx 0.002$$

Ejemplo: Modelo Perceptrón





Note que el Perceptrón analiza el signo de z para ver el lado del plano separador que le corresponde.

Del mismo modo, el Perceptrón puede evaluar la sigmoide para encontrar el plano separador mediante su umbral (0.5).

Modelo Perceptrón

- Perceptrón: Una neurona con pesos sinápticos y nivel de umbral ajustables.
- Propósito: Clasificar estímulos externos en una de dos clases (C1 y C2).
- Aprendizaje: Calcular el vector de pesos W a partir de un conjuntos de P ejemplos (conjunto de entrenamiento E).
  - Vector de pesos: $W = \langle w_0, w_1, \dots, w_n \rangle$
  - Conjunto de entrenamiento: E= $\{\langle \vec{x}^1, t^1 \rangle, \langle \vec{x}^2, t^2 \rangle, \dots, \langle \vec{x}^P, t^P \rangle\}$, donde:
  - $\vec{x}^u = \langle +1, x_1^u, x_2^u, \dots, x_n^u \rangle$ es el vector de entradas en $\{-1, 1\}$
  - t^u salida deseada en $\{-1, 1\}$
- El perceptrón debe satisfacer la ecuación $sign(\vec{w} \cdot \vec{x}^u) = t^u, \quad \forall u$
- Note que $\vec{w} \cdot \vec{x} = 0$ define un hiperplano que separa el espacio de entradas en puntos positivos y negativos

Algoritmo Perceptrón

1. Inicializar el vector de pesos $\vec{w} = 0$
2. Seleccionar un ejemplo $\langle \vec{x}^u, t^u \rangle$ en orden cíclico o al azar.
3. Si $\vec{w}$ clasifica correctamente $\vec{x}^u$, es decir

$$(\vec{w} \cdot \vec{x}^u) \geq 0 \; y \;\; t^u = \; +1, \;\; \text{o}$$
$$(\vec{w} \cdot \vec{x}^u) < 0 \; y \;\; t^u = -1$$

   Entonces no se realiza ninguna acción, de otro modo modificar $\vec{w}$,

   utilizando la regla correctora del error :

$$w' = \vec{w} + t^u \vec{x}^u$$

4. Volver al punto 2.

Modelo Perceptrón

- Ejemplo AND

	x_0	x_1	x_2	y	
x^1	+1	-1	-1	-1	t^1
x^2	+1	-1	+1	-1	t^2
x^3	+1	+1	-1	-1	t^3
x^4	+1	+1	+1	+1	t^4

- 1. Inicializar los pesos en 0. $\vec{w}^t = [w_0\ w_1\ w_2] = [0\ 0\ 0]$
- 2. Calculamos $\vec{w} \cdot \vec{x}^1 = (0 \cdot 1 + 0 \cdot -1 + 0 \cdot -1) \geq 0$ y como $t^1 = -1$, se deben corregir los pesos utilizando $w' = \vec{w} + t^1 \vec{x}^1$. Así $w' = [0\ 0\ 0] + (-1) \cdot [1\ -1\ -1]$ =[-1 1 1]
- 3. Seleccionamos la segunda entrada y repetimos el punto anterior con los nuevos pesos obtenidos. Así $\vec{w} \cdot \vec{x}^2 = (-1 \cdot 1 + 1 \cdot -1 + 1 \cdot 1) \leq 0$ y como $t^2 = -1$ no hay corrección. Por lo tanto el vector de pesos se mantiene w'=[−1 1 1].
- 4. Se repite hasta que no exista más variación en los pesos obtenidos.

Modelo Perceptrón

- Tarea: Aplique el modelo de perceptrón para la clasificación de una tabla OR.

	x_0	x_1	x_2	y	
x^1	+1	-1	-1	-1	t^1
x^2	+1	-1	+1	+1	t^2
x^3	+1	+1	-1	+1	t^3
x^4	+1	+1	+1	+1	t^4

- ¿Es posible realizar la clasificación?

Modelo Perceptrón

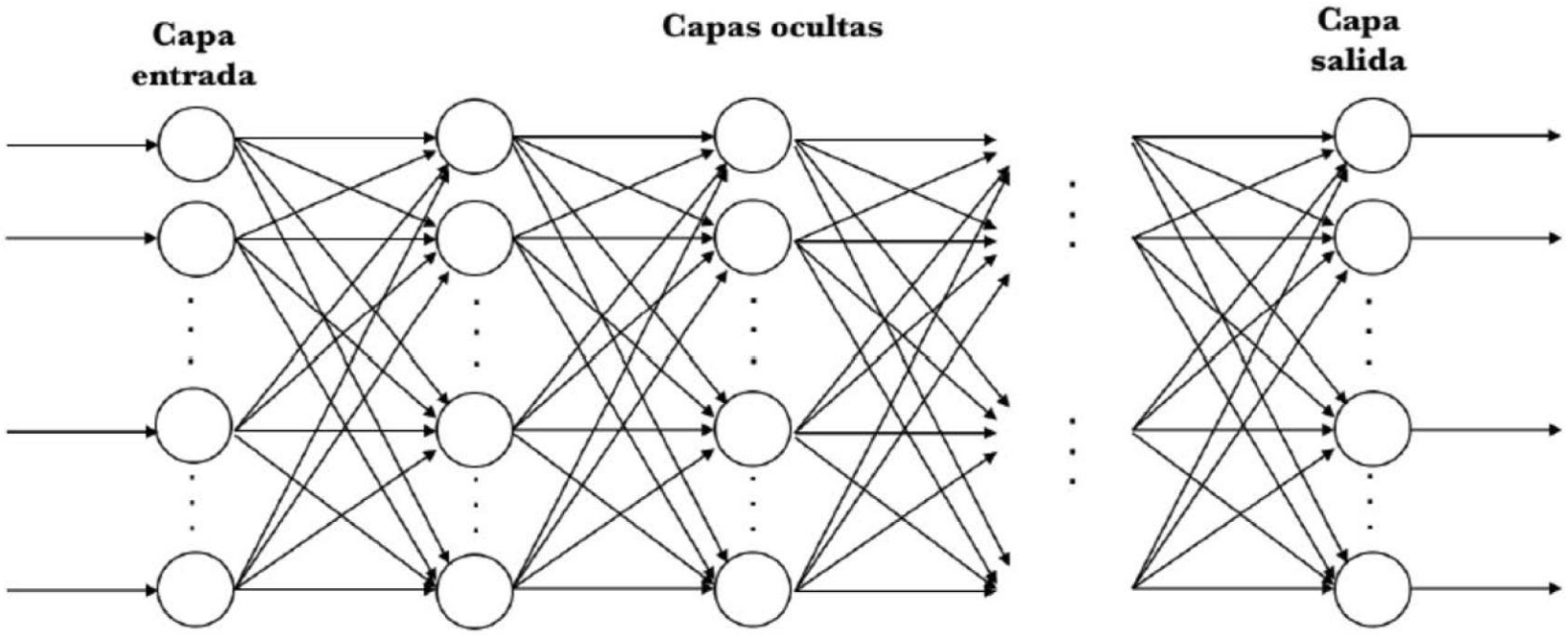
- Tarea: Aplique el modelo de perceptrón para la clasificación de una tabla XOR.

	x_0	x_1	x_2	y	
x^1	+1	-1	-1	-1	t^1
x^2	+1	-1	+1	+1	t^2
x^3	+1	+1	-1	+1	t^3
x^4	+1	+1	+1	-1	t^4

- ¿Es posible realizar la clasificación?

Modelo Perceptrón Multi Capas

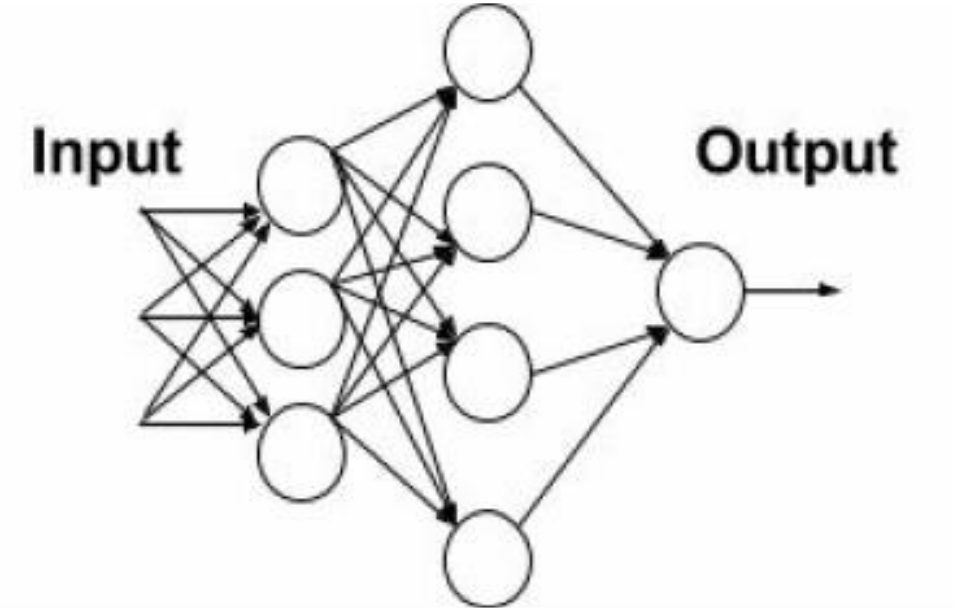
Nos referimos a un Multi-Layer Perceptron (MLP) cuando nos encontramos con redes neuronales que tienen una capa de entrada (input layer), una o más capas compuestas por perceptrones, llamadas capas ocultas (hidden layers), y una capa final con varios perceptrones llamada la capa de salida (output layer). En general nos referimos a Deep Learning cuando el modelo basado en redes neuronales está compuesto por múltiples capas ocultas. Visualmente se puede presentar con el siguiente esquema:



Modelo Perceptrón Multi Capas

Por ejemplo, en la siguiente red, cada nodo en la primera capa recibe una entrada y se activa de acuerdo a sus funciones de decisión local predefinidos. Luego, la salida de la primera capa se pasa a la segunda capa, cuyos resultados continúan a la capa de salida final que consiste en una sola neurona. Es interesante notar que esta organización en capas se asemeja vagamente a los patrones de visión humana que discutimos anteriormente.

La red es **densa**, lo que significa que cada neurona en una capa está conectada a todas las neuronas ubicadas en la capa anterior y a todas las neuronas en la siguiente capa.



Modelo Perceptrón Multi Capas

Consideremos una sola neurona:

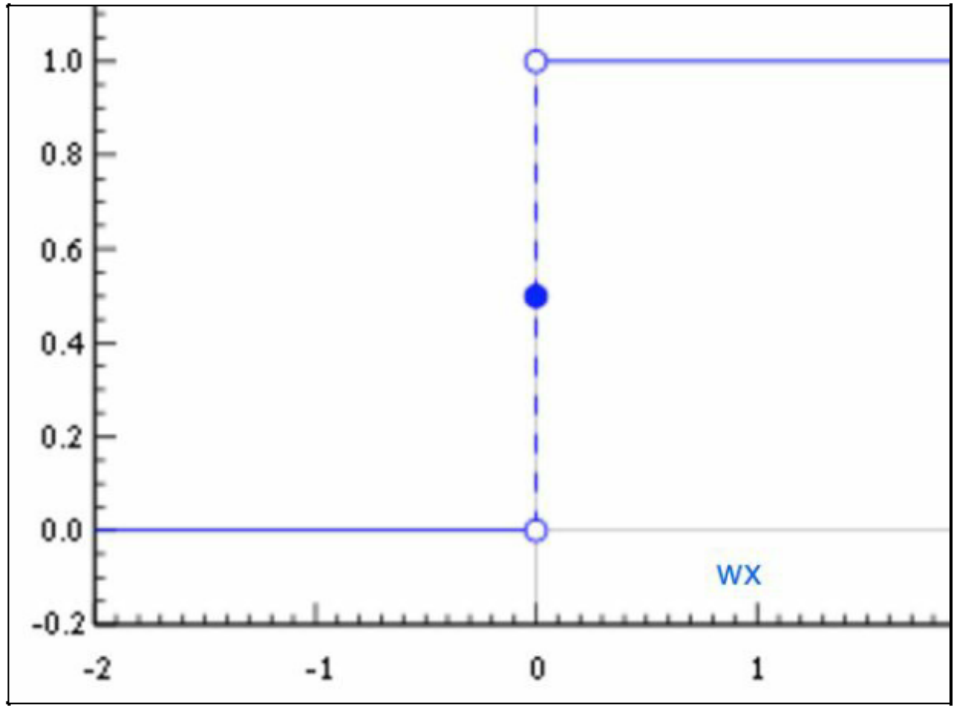
¿Cuáles son las mejores opciones para el peso *w* y el sesgo *b*?

Idealmente, nos gustaría proporcionar un conjunto de ejemplos de entrenamiento y dejar que el computador ajuste el peso y el sesgo de tal manera que se minimicen los errores producidos en la salida. Para hacer esto un poco más concreto, supongamos que tenemos un conjunto de imágenes de gatos y otro conjunto separado de imágenes que no contienen gatos. Por simplicidad, suponga que cada neurona mira solo un valor de píxel de entrada único.

Mientras el computador procesa estas imágenes, nos gustaría que nuestra neurona ajustará sus pesos y sesgos para que tengamos cada vez menos imágenes incorrectamente reconocidas como no gatos. Este enfoque parece muy intuitivo, pero requiere que un pequeño cambio en los pesos (y / o sesgo) cause solo un pequeño cambio en los resultados.

Modelo Perceptrón Multi Capas

Si tenemos un gran salto de salida no podemos aprender progresivamente. Un perceptrón posee salidas 0 o 1 y debido a este salto en su respuesta no le ayudará a aprender, como se muestra en el siguiente gráfico:

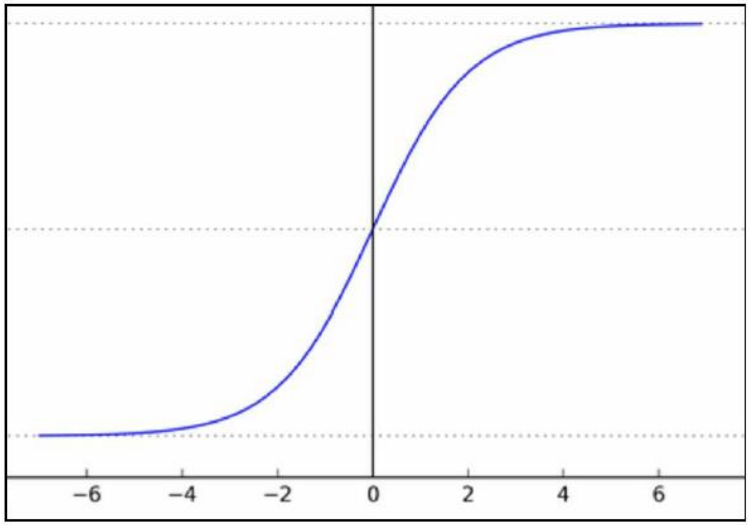


Modelo Perceptrón Multi Capas

Necesitamos algo diferente, más suave. Necesitamos una función que cambie progresivamente de 0 a 1 sin discontinuidad. Matemáticamente, esto significa que necesitamos una función continua que nos permita calcular la derivada. Acá recurrimos a las funciones de activación.

Una función sigmoide es definida de la siguiente manera: $$\sigma(x) = \frac{1}{1+e^{-x}}$$

Como se representa en el siguiente gráfico, tiene pequeños cambios de salida en (0, 1) cuando la entrada varía. Matemáticamente, la función es continua. Una función sigmoidea típica se representa en el siguiente gráfico:



Modelo Perceptrón Multi Capas

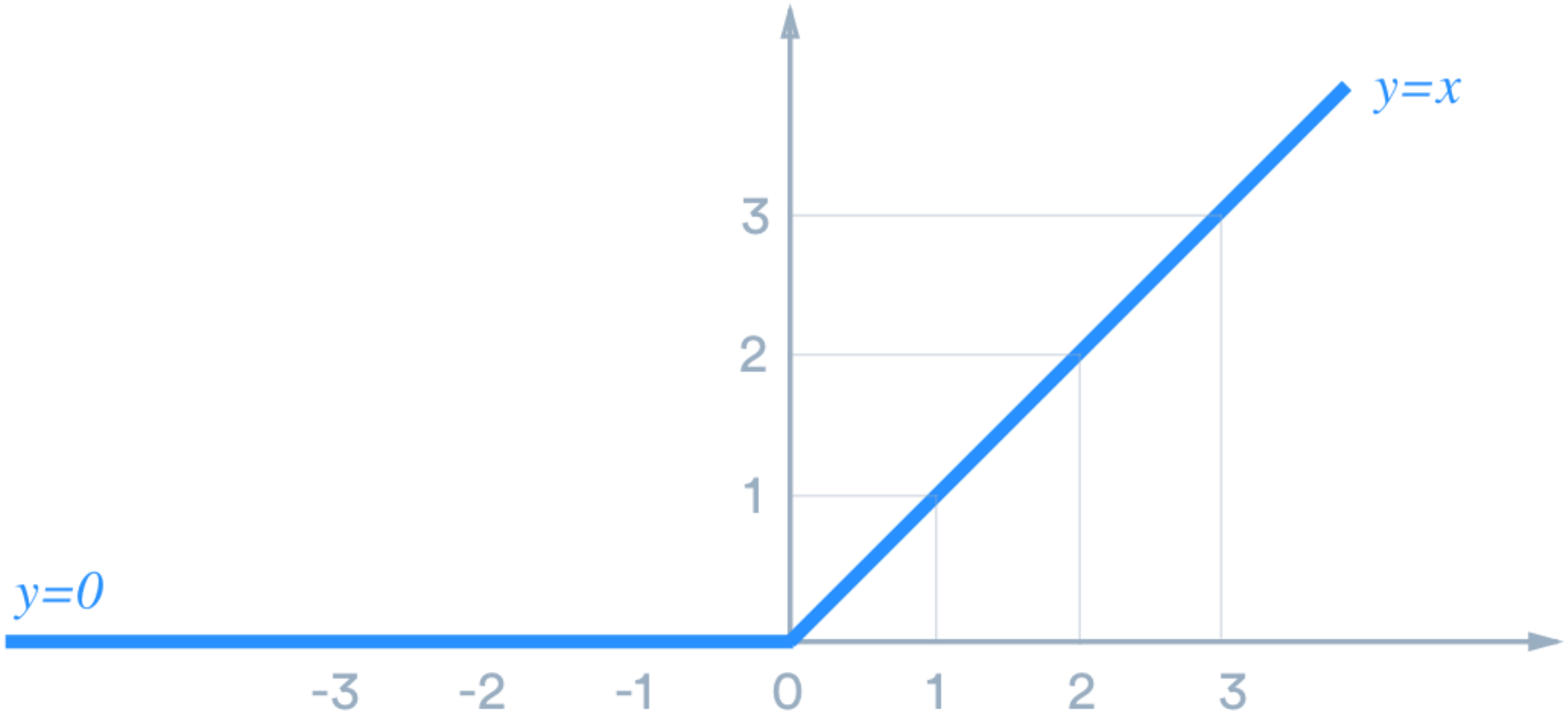
Una neurona puede usar el sigmoide para calcular la función no lineal $\sigma(z = wx + b)$. Tenga en cuenta que, si $z = wx + b$ es muy grande y positivo, entonces $e^{-z} \to 0$, así $\sigma(z) \to 1$, mientras que si $z = wx + b$ es muy grande y negativo, entonces entonces $e^{-z} \to \infty$, así $\sigma(z) \to 0$.

En otras palabras, una neurona con activación sigmoidea tiene un comportamiento similar al perceptrón, pero los cambios son graduales y los valores de salida, como 0.5539 ó 0.123191, son perfectamente legítimos. En este sentido, una neurona sigmoide puede responder “tal vez”.

Por otra parte, existen otras funciones de activación útiles. El sigmoide no es el único tipo de función de activación suave utilizada para las redes neuronales. Una función muy simple llamada unidad lineal rectificada (ReLU) se volvió muy popular porque genera muy buenos resultados experimentales. En el siguiente gráfico se muestra la función, que es cero para los valores negativos y crece linealmente para los valores positivos.

Modelo Perceptrón Multi Capas

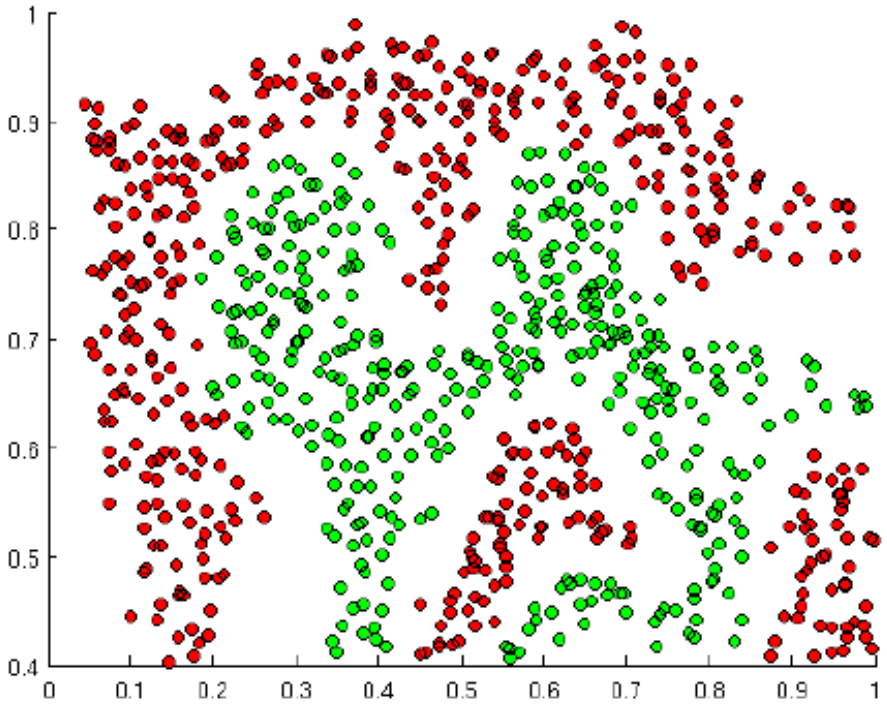
ReLU



Importancia de las funciones de activación

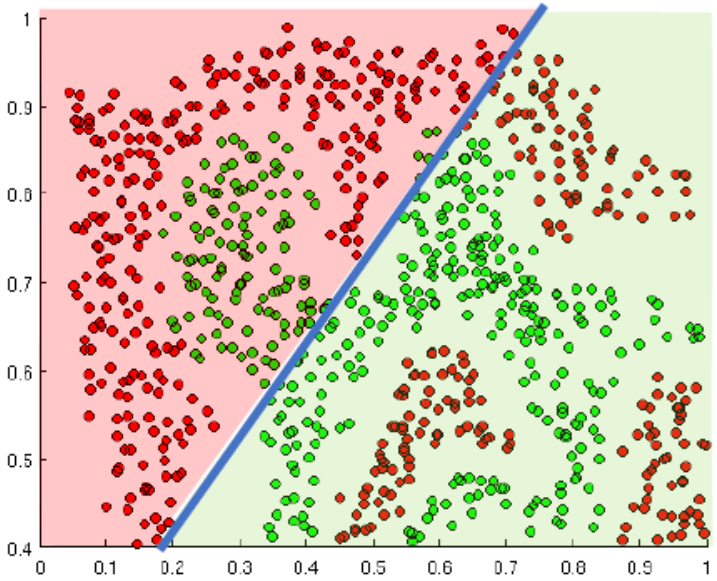
El propósito de las funciones de activación es introducir no linealidades a la red.

¿Qué sucede si quisiéramos construir una red neuronal para clasificar los puntos verdes de los rojos?

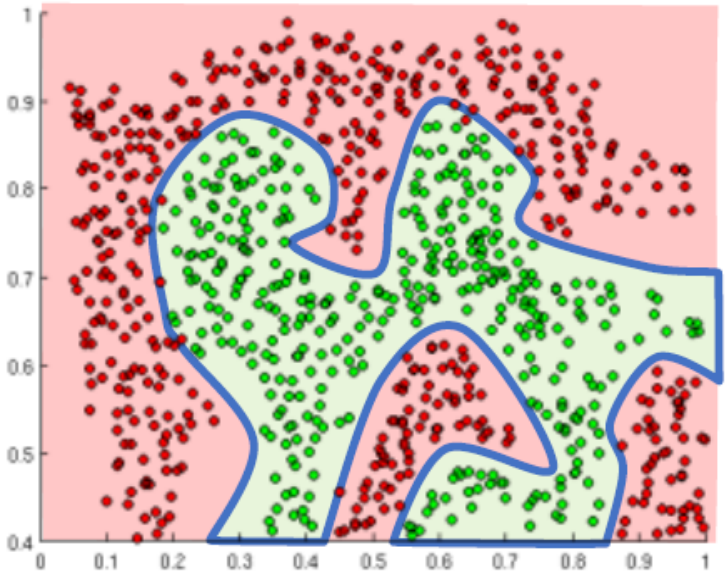


Importancia de las funciones de activación

¿Qué sucede si quisiéramos construir una red neuronal para clasificar los puntos verdes de los rojos?



Funciones de activación lineal producen decisiones lineales no importando el tamaño de la red



Funciones de activación no lineal permiten aproximar funciones complejas