

Redes Neuronales Artificiales (Parte 2)

clase 5

Definición : Una neurona artificial es una unidad de procesamiento de información de una red neuronal. Los elementos que conforman una red neuronal son :

1) Un conjunto de líneas/conexiones / sinapsis , caracterizados por un peso/fuerza θ_{ij} .

OBS : "W" de weight usualmente , es decir $\theta_{ij} \equiv W_{ij}$.

2) Un sumador de señales ponderadas por los pesos θ_{ij} (Combinación Lineal)

3) Una función de activación que limita la amplitud de salida. Generalmente , esta transformación es no-lineal. Ejemplo :

i) Función signo

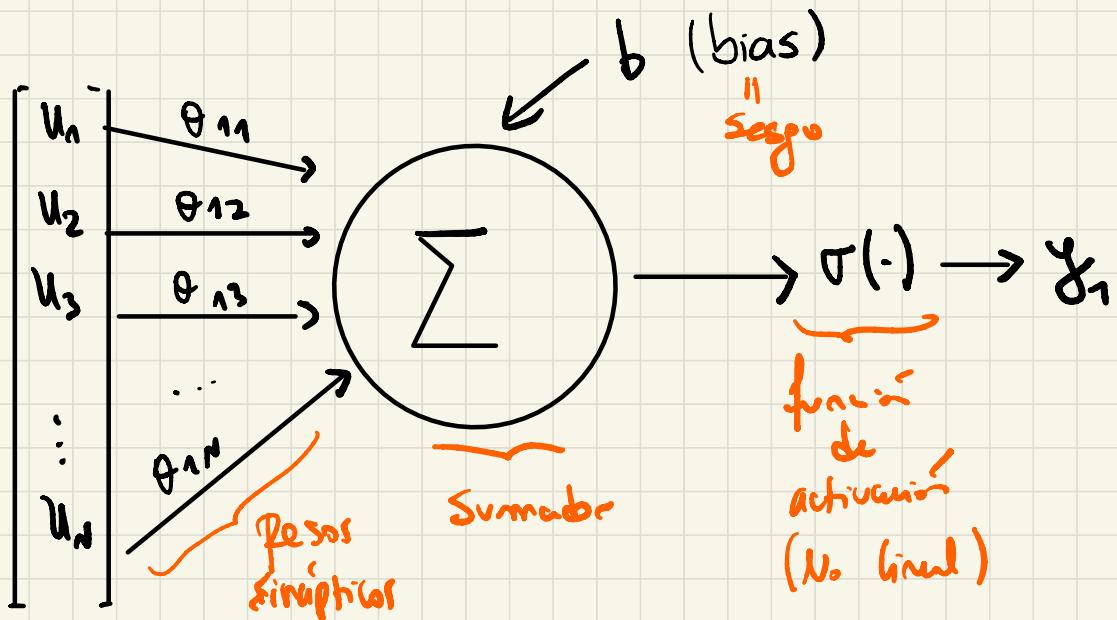
etc |
~

ii) Función arco tangente

iii) Función sigmoidal

iv) Función ReLU

Ilustremos una sola neurona :



Matemáticamente :

$$\sigma(\theta_{11}u_1 + \theta_{12}u_2 + \dots + \theta_{1n}u_n) = y_1$$

\Leftrightarrow

$$\sigma\left(\sum \theta_{1i}u_i\right) = y_1$$

$$\Leftrightarrow \sigma\left(\langle \vec{\theta}_{11}, u \rangle\right) = y_1$$

Si repetimos el modelo, considerando las salidas de un modelo como entradas de otro hablaremos de "redes".

En notación matemática (sumatorias) la

neurona k -ésima puede ser escrita como:

$$u_k = \sum_{j=1}^m \theta_{kj} x_j \quad (1)$$

Podemos conectar la notación anterior a matrices, de la siguiente forma:

$$u_1 = \sum_{j=1}^m \theta_{1j} x_j = \theta_{11} x_1 + \theta_{12} x_2 + \dots + \theta_{1m} x_m$$

$$u_2 = \sum_{j=1}^m \theta_{2j} x_j = \theta_{21} x_1 + \theta_{22} x_2 + \dots + \theta_{2m} x_m$$

⋮ ⋮ ⋮

Es decir:

$$\underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_m \end{bmatrix}}_{m \times 1} = \underbrace{\begin{bmatrix} \theta_{11} & \theta_{12} & \dots & \theta_{1m} \\ \theta_{21} & \theta_{22} & \dots & \theta_{2m} \\ \vdots & \ddots & \ddots & \vdots \\ \theta_{m1} & \dots & \theta_{mm} \end{bmatrix}}_{m \times m} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_m \end{bmatrix}}_{m \times 1}$$

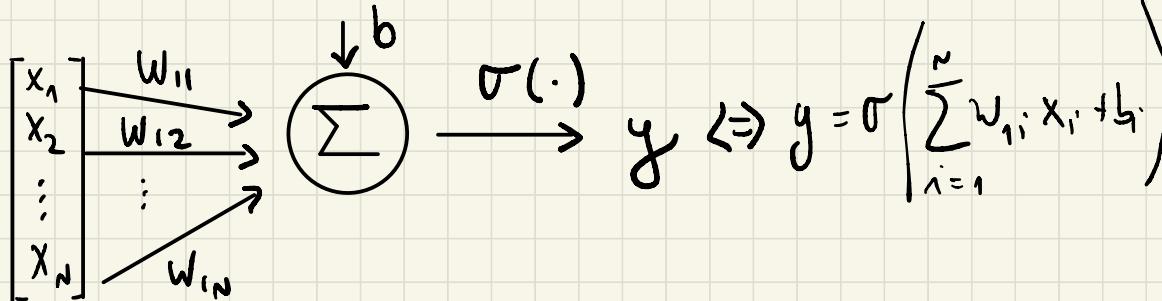
Una Neuron
artificial

Pero!

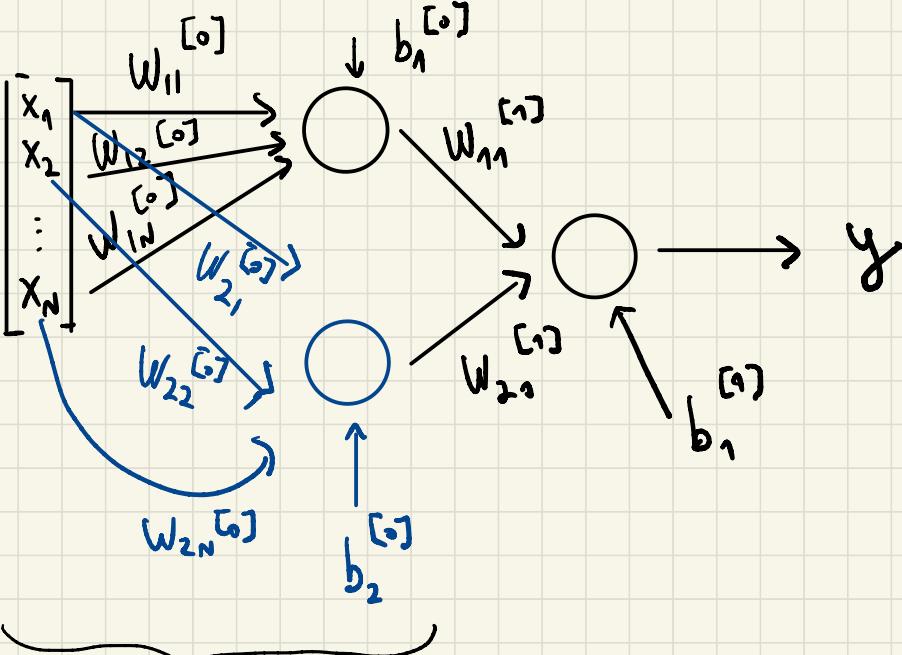
Generalmente
veremos
modelos que
comprimen la
información.

$$u = \theta \times \left\{ \begin{array}{l} \text{es un producto} \\ \text{matriz \times vector!} \end{array} \right.$$

Modelos de una neurona :



Modelo de red neuronal :



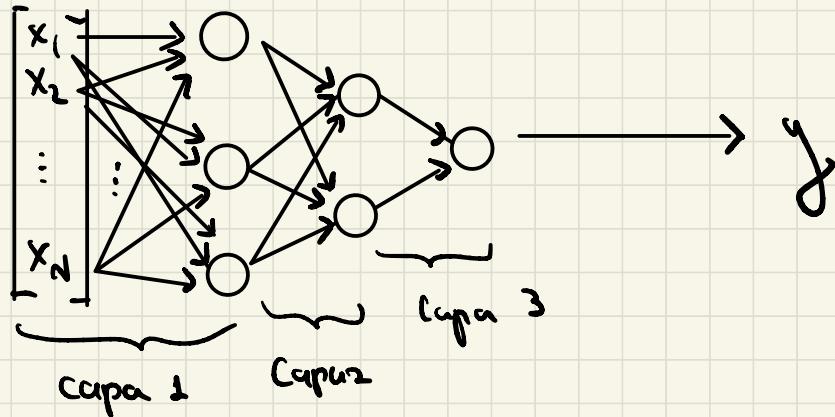
2 Neuronas, Matricialmente :

$$\sigma \left(\begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1N} \\ W_{21} & W_{22} & \cdots & W_{2N} \end{bmatrix}_{2 \times N} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{N \times 1} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}_{2 \times 1} \right) = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}_{2 \times 1}$$

Luego :

$$y = \sigma \left(w_{11}^{[1]} z_1 + w_{21}^{[1]} z_2 + b_1^{[1]} \right)$$

Ejercicio : Exprese matemáticamente siguiendo la notación anterior el siguiente diagrama



Señale las dimensiones de las matrices y vectores que aparecen!

Geométricamente, pode mos pensar en un hiperplano donde b es la translación de sufr dicho hiperplano, es decir :

$$a_k = \sum_{i=1}^n \theta_{ki} u_i + b_k$$

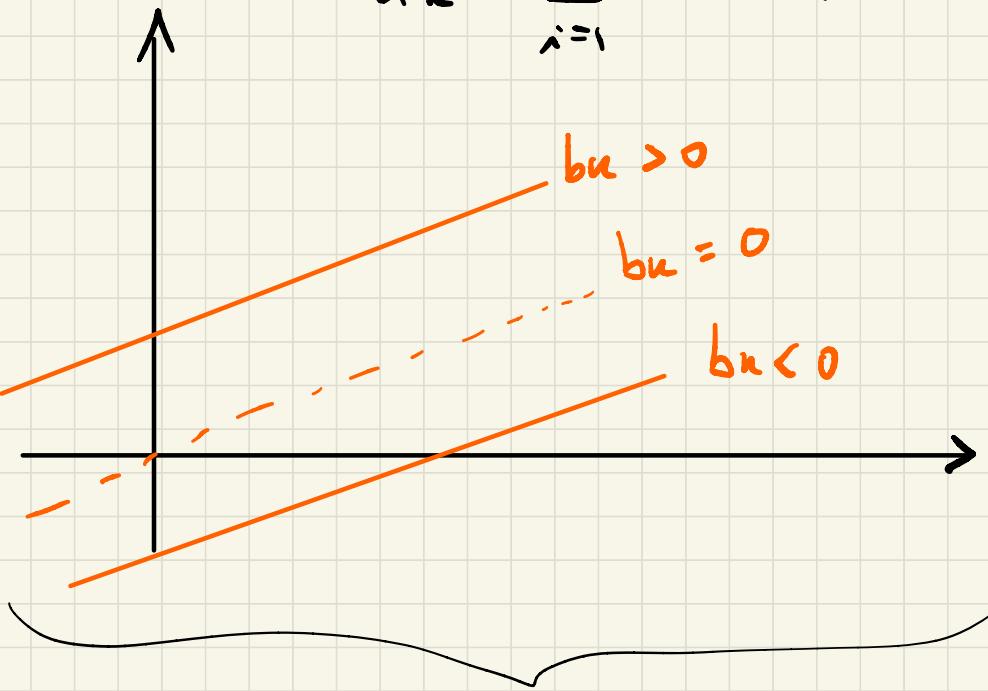
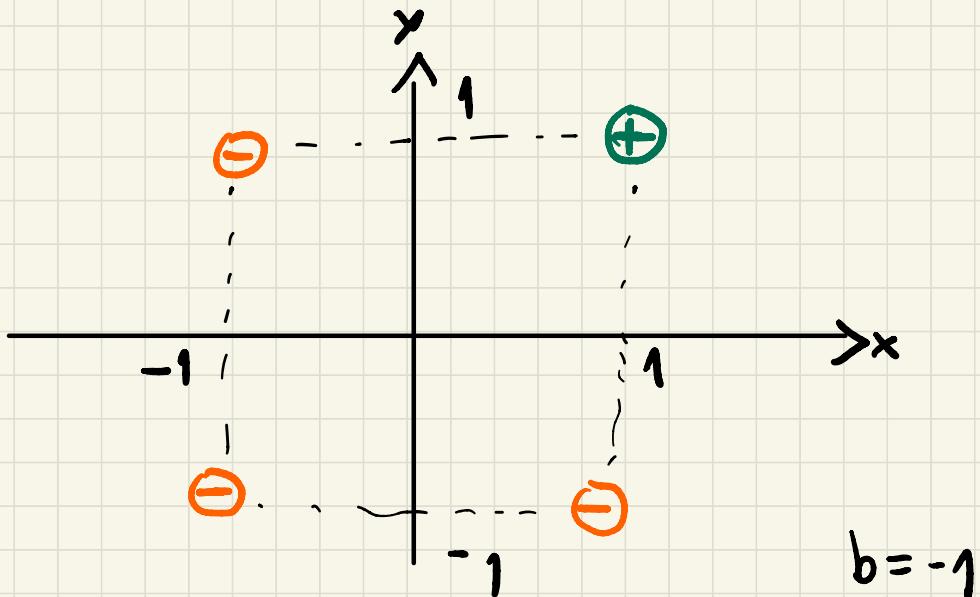


Ilustración reducida a 2 dimensiones !

- a_k es llamado "Potencial de activación"

Inicialmente, la motivación de estos modelos consistía en imitar fáscies, ejemplo (Hornik 1989) :



x	y	Clase
-	-	-
-	+	-
+	-	-
+	+	-

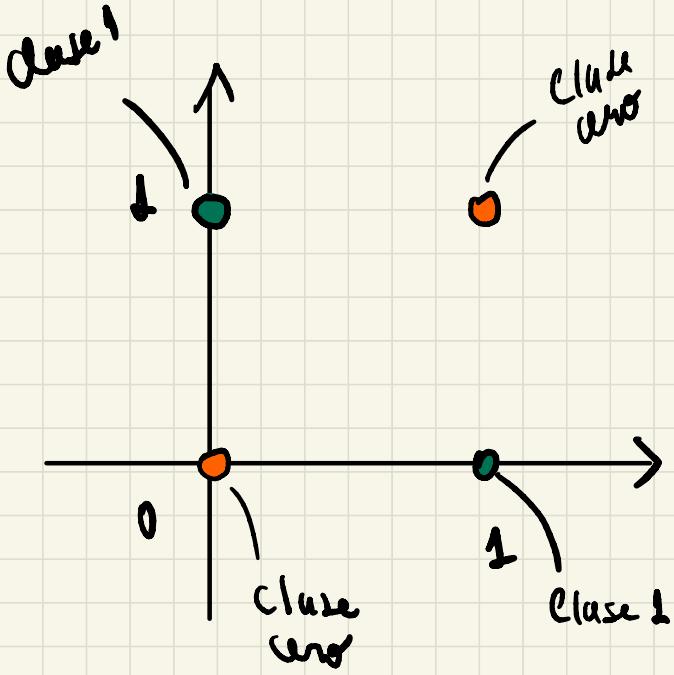
input
" " $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$ $w_{11} = 2$ \rightarrow Σ
 $w_{12} = 2$ " "
 $\sigma(-2 + -2 - 1)$

$T = \text{fórmula signo} = \frac{x}{|x|} \text{ con } x \neq 0$ -1

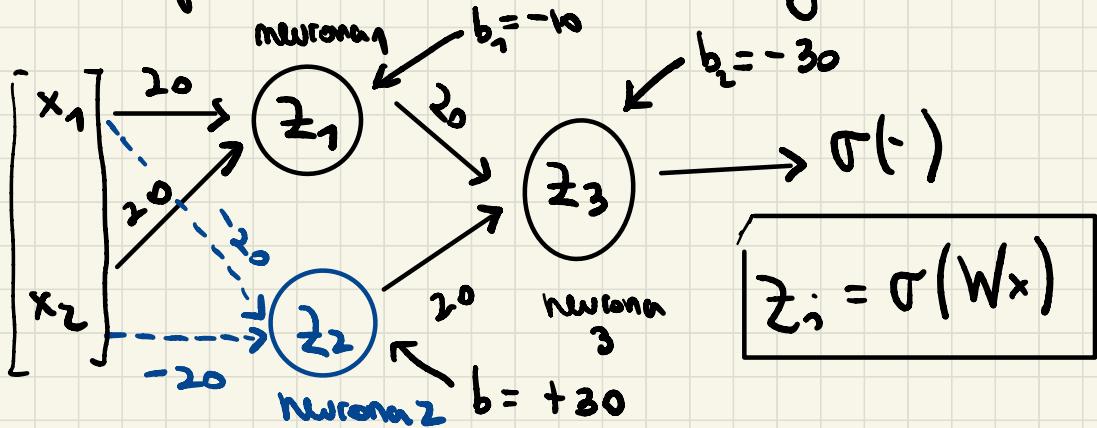
Un problema clásico al que se vio
enfrentado este modelo fue a la función

XOR :

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Para poder resolverlo fue necesario agregar
otra capa de neuronas o "layers":



Calculemos :

$$z_1 = \sigma(20x_1 + 20x_2 - 10)$$

$$z_2 = \sigma(-20x_1 - 20x_2 + 30)$$

$$z_3 = \sigma(20\sigma(z_1) + 20\sigma(z_2) - 30)$$

1) $x_1 = 0, x_2 = 0$:

$$z_1 = -1$$

$$z_2 = +1$$

$$z_3 = \sigma(-20 + 20 - 30)$$

$$z_3 = -1 \rightarrow \text{Clase 0} \checkmark$$

2) $x_1 = 0, x_2 = 1$:

$$z_1 = +1$$

$$z_2 = +1$$

$$z_3 = \sigma(20 + 20 - 30)$$

$$z_3 = +1 \rightarrow \text{Clase 1}$$

3) $x_1 = 1, x_2 = 0$:

$$z_1 = +1$$

$$z_2 = +1$$

$$z_3 = +1 \rightarrow \text{Clase 1}$$

$$4) \quad x_0 = 1 \quad , \quad x_2 = 1 : \quad$$

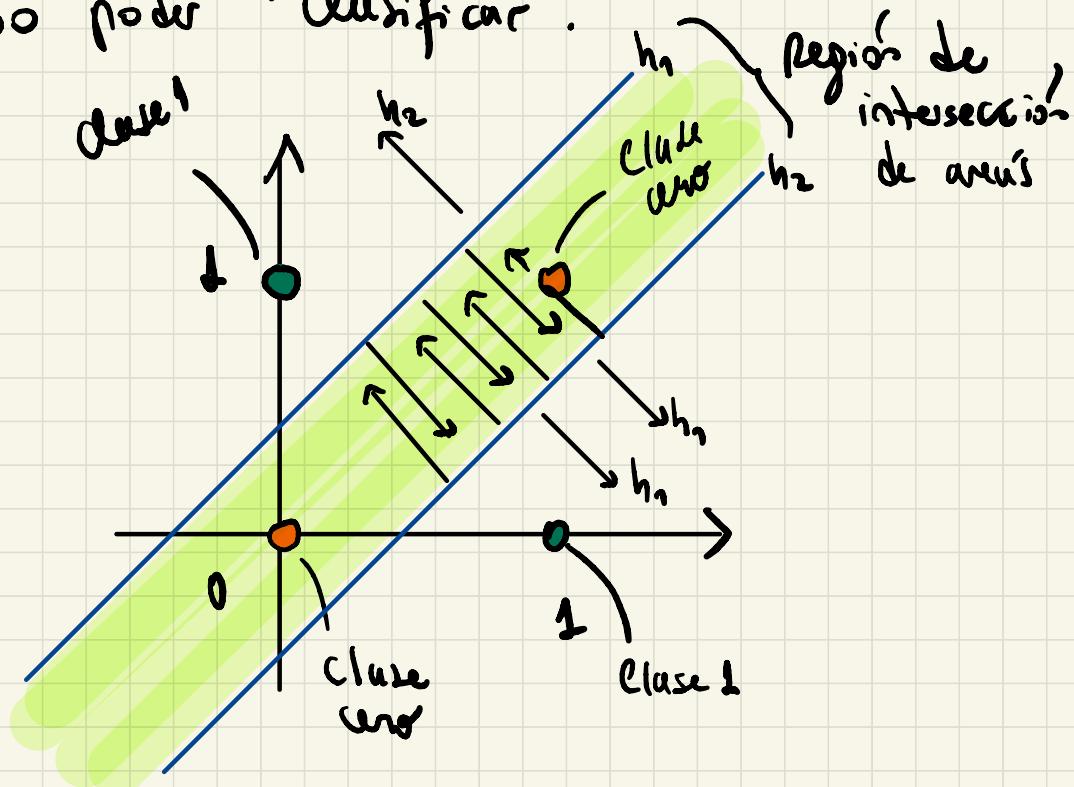
$$z_1 = +1$$

$$z_2 = -1$$

$$z_3 = -1 \rightarrow \text{clase } 0$$



- Es decir, el modelo reavivó de varios hiperpálanos como si buscara operariera su intersección y a partir de eso poder "clasificar".



- Sin embargo, devísemos encontrar tales parámetros a través de un algoritmo de optimización.
- Pues ya se imaginarán que el gradiente descendiente nos permitiría resolver dicho problema.
- Si el problema es una Regresión utilizaremos el error cuadrático medio como posible candidato a función de costo.
- Si el problema es una clasificación utilizaremos el costo asociado a una función logística.

$$J(\theta) = \sum_{i=0}^m (\hat{y}_\theta(x_i) - y_i)^2 \quad (\text{Regresión})$$

$$J(\theta) = \sum_{i=0}^m -y_i \log(\sigma_\theta(x_i)) - (1-y_i) \log(1 - \sigma_\theta(x_i)) \quad (\text{Classification})$$

Clasificando con un Perceptrón (No visto en clases) vars!

Analicemos un caso sencillo

$$\hat{y} = \Gamma(\theta, b)$$

$$x_1 \xrightarrow{\theta_1}$$

$$x_2 \xrightarrow{\theta_2}$$

$$z = \theta_1 x_1 + \theta_2 x_2 + b \longrightarrow$$

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

Vamos al código! Desafío: Programa un Perceptrón!

Antiguamente el perceptrón utilizaba otra regla para actualizar los pesos, dada por:

$$\theta = \theta + \Delta\theta$$

$$\theta_0 = \theta_0 + \Delta\theta_0$$

$$\Delta\theta = \alpha \cdot (y_i - \hat{y}_i) \cdot x_i$$

$$\Delta\theta_0 = \alpha \cdot (y_i - \hat{y}_i)$$

Regla
Antigua

Con $\alpha \in [0, 1]$ "learning rate".

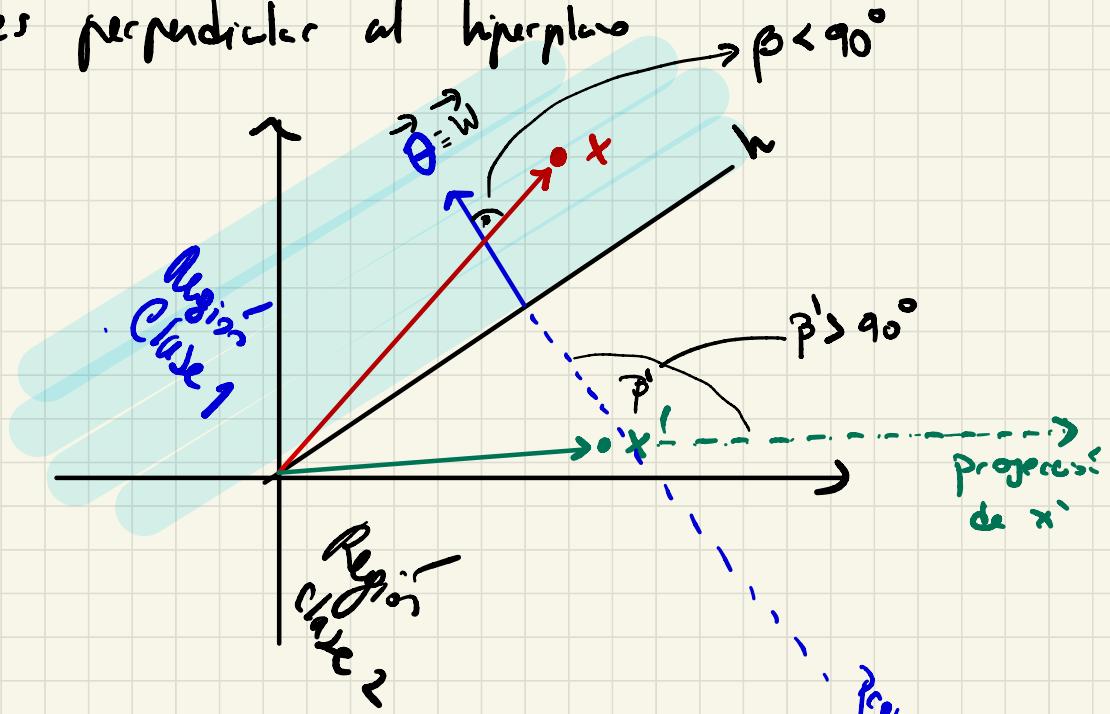
¿Por qué funciona esta regla de actualización?

y	\hat{y}	$y - \hat{y}$
1	1	0
0	0	0
1	0	1
0	1	-1

} no hay error!

} Peso muy pequeño
} Peso muy grande

- La regla del perceptrón se basa en $\sigma(w^T x + b)$
- Si: $\sigma(w^T x + b) > 0$ (Clase A) si: $\sigma(w^T x + b) \leq 0$ (Clase B).
- En la actualización de $\Delta\theta$ se multiplica por x_i . El vector de pesos $\theta = [\theta_1, \theta_2, \dots, \theta_m]$ es perpendicular al hiperplano

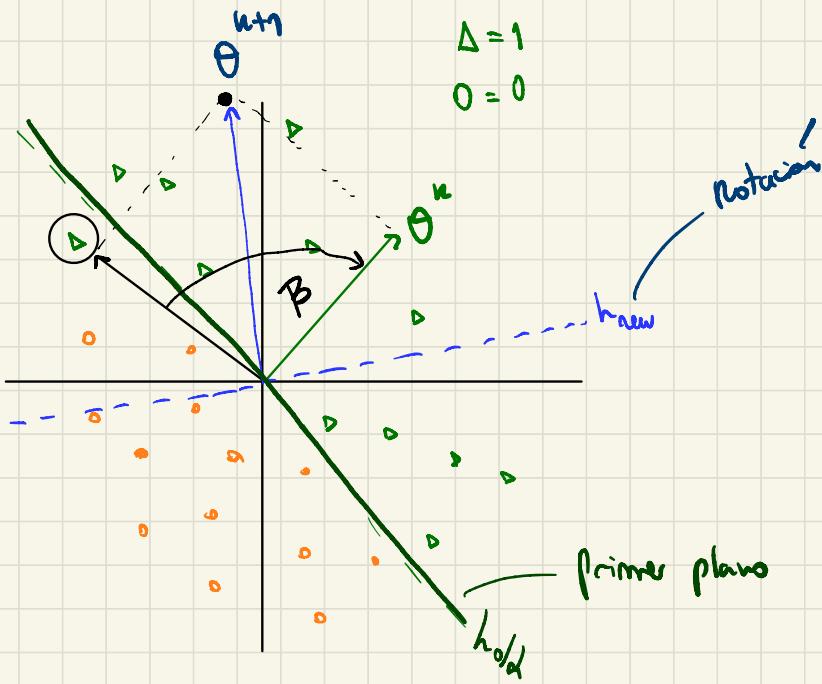


Existen 3 posibilidades :

- i) $y_i - \hat{y}_i = 0$ (Clasificamos correctamente) $\Rightarrow \Delta\theta \equiv 0 \quad \Delta\theta_0 \equiv 0$
- ii) $y_i - \hat{y}_i = 1 \Rightarrow \hat{y}_i = 0$ es decir $\langle w, x \rangle \leq 0$, debería ser $\langle w, x \rangle > 0$, Geometricamente:

$$\Delta = \text{clase } 1$$

$$x = \text{clase } 0$$



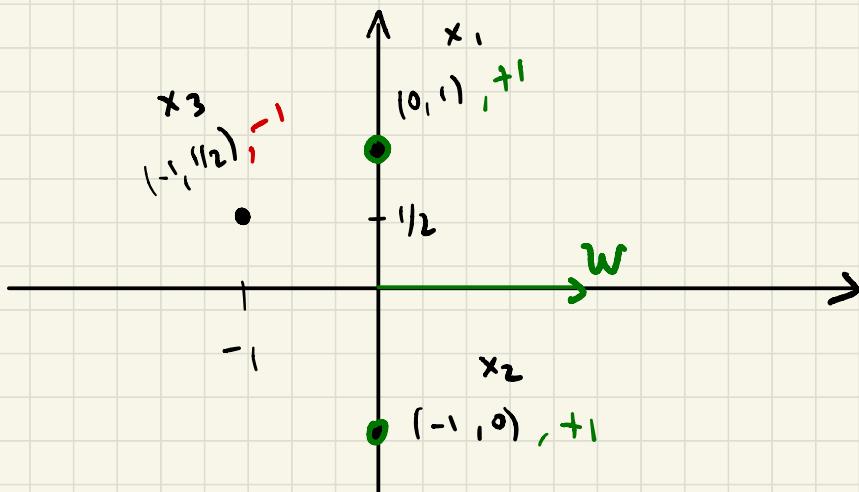
- En este caso el plano " h " tiene pesos que no están definiendo bien la frontera de decisión.
- β debería ser menor que 90° .
- La regla de actualización "suma" x_i pesos :

$$\Delta\theta = \alpha (y_i - 0) x_i = \alpha (1) \cdot x_i$$

$$\Rightarrow \theta^{u+1} = \theta^u + \alpha x_i \quad (\text{si } \alpha = 1 \text{ el dibujo } \textcolor{orange}{\text{malo}})$$

- Observe que mejoró la situación del Δ pivote y emparejó para otros.

Son estos datos linealmente separables? : (Esto
y es
mucho!)



$$w^T x > 0 \stackrel{?}{\Rightarrow} y = +1$$

$$w^T [-1, 1/2] < 0 \stackrel{?}{\Rightarrow} y = -1$$

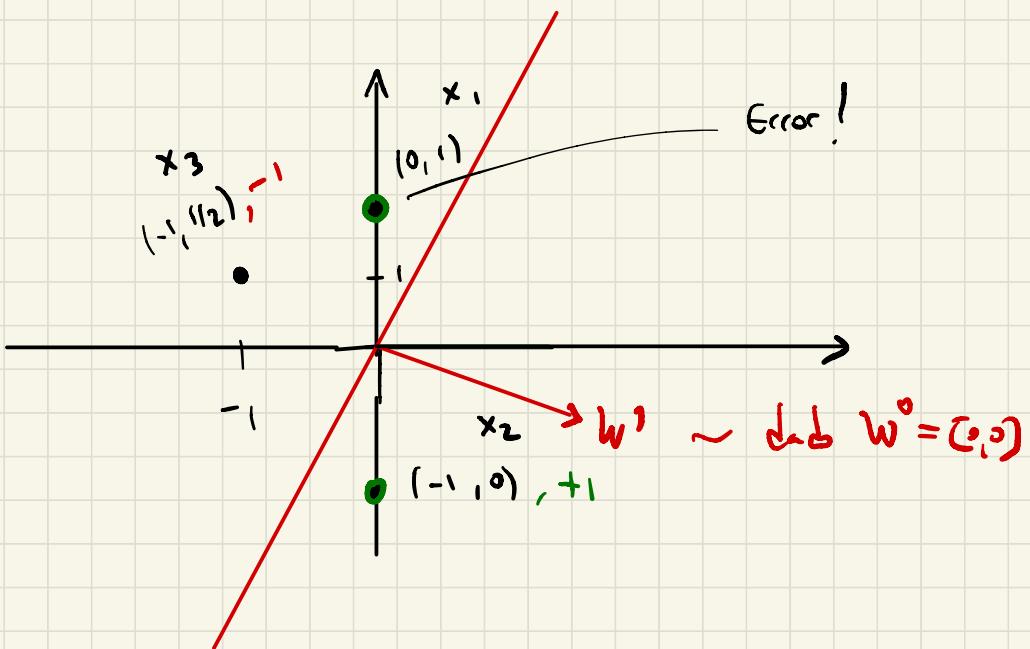
} Estos 3 puntos son linealmente separables

Por ejemplo, si $w^0 = [0, 0]$

Iteremos:

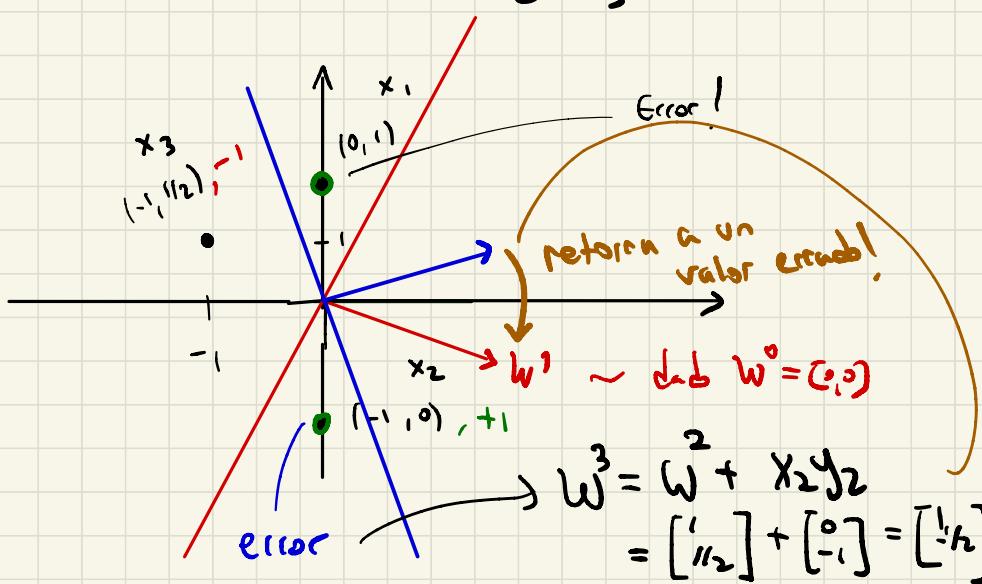
- $w^{[0]T} x_i = 0 \quad \forall x_i, i=1, 2, 3$
 - Cometimos solo 1 error x_3
- $$\Rightarrow w^1 = w^0 + x_3 y_3 = w^0 + x_3 y_3$$
- $$= [0, 0] + [-1, 1/2]$$

*** $w^1 = [-1, 1/2]$ ***



$$w^2 = w^1 + x_1 \cdot y_1 = \begin{bmatrix} 1 \\ -1/2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot +1$$

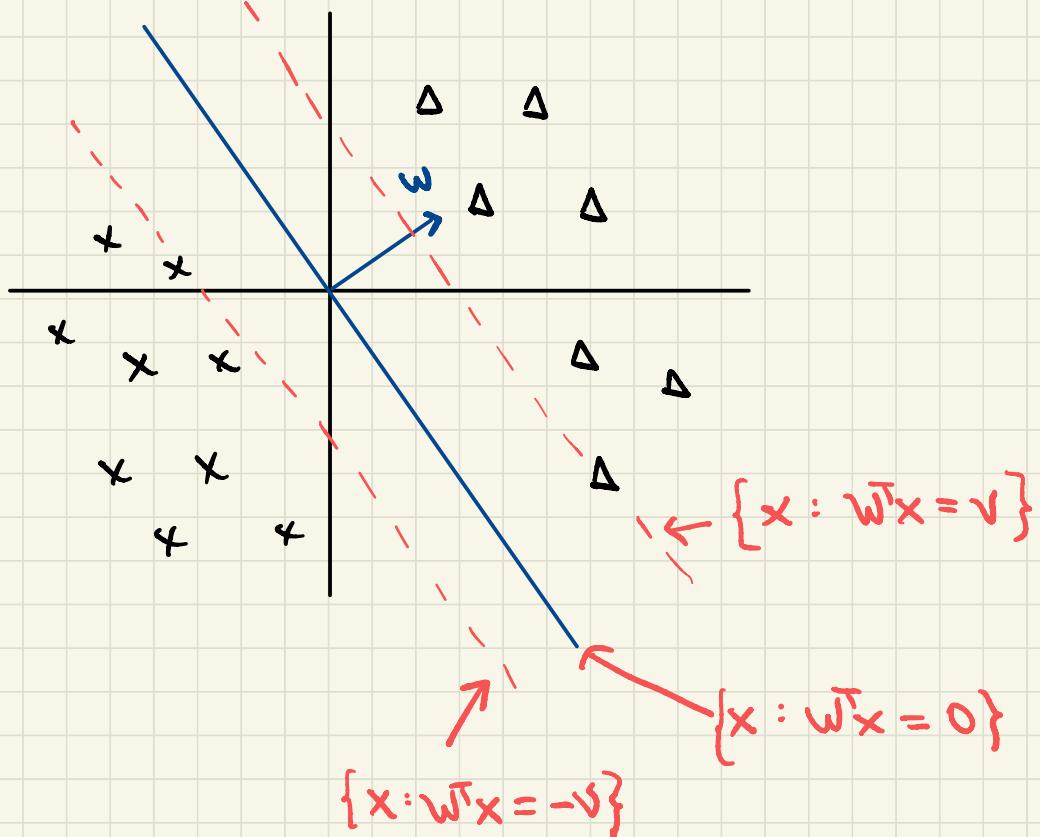
$$w^2 = \begin{bmatrix} 1 \\ 1/2 \end{bmatrix}$$



$$\begin{aligned} w^3 &= w^2 + x_2 y_2 \\ &= \begin{bmatrix} 1 \\ 1/2 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1/2 \end{bmatrix} \end{aligned}$$

Para evitar este problema de convergencia, debemos asumir que :

Separabilidad Lineal con Margen - γ (Gamma)



Un conjunto de datos es **linealmente separable**

con v -margen si: $\exists v \in \mathbb{R}^d$ sujeto a:

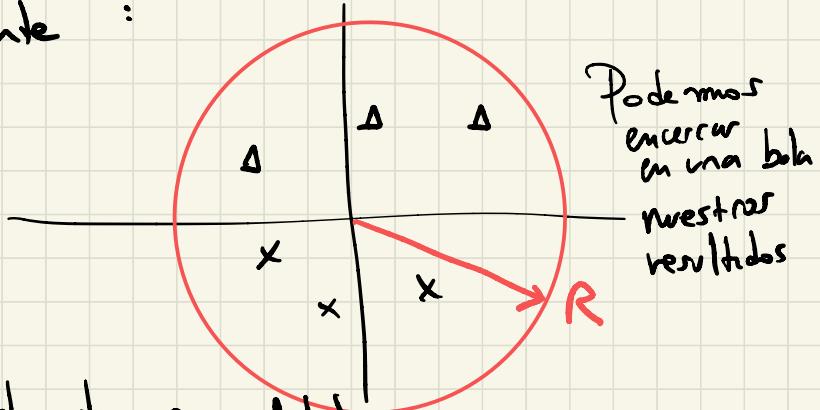
$(w^T x_i) y_i \geq v \quad \forall i$ para algún $v > 0$

- Es decir asumimos que no hay puntos en ese margen.
- Asumiendo que el dataset es linalmente separable con γ -margen \Rightarrow podemos probar convergencia!

Otras cosas que asumimos :

$$2) \forall i \in D \quad \|x_i\|_2 < R \quad \text{para algún } R > 0$$

Geometricamente :



$$3) \text{ Sin pérdida de generalidad} \quad \text{asumimos:} \quad \|w^*\| = 1$$

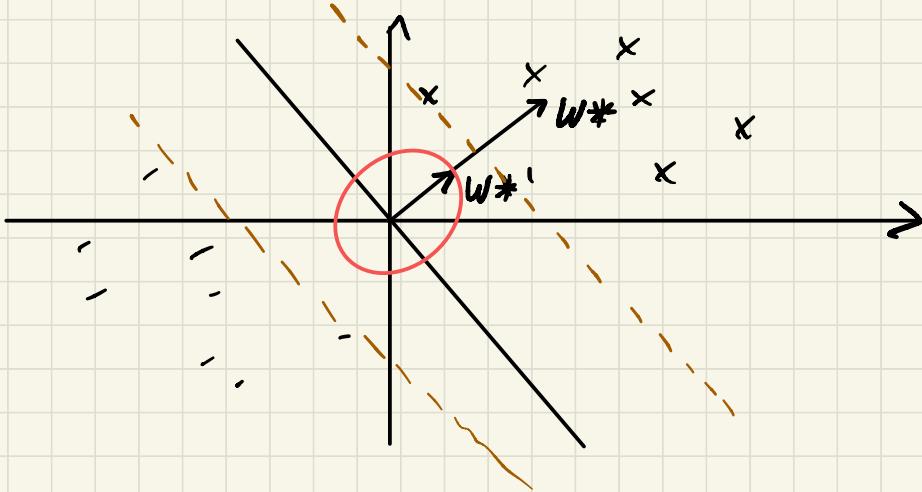
w^* : solución del problema.

Esto cambia la definición de los márgenes :

$$\left\{ x : \frac{w^T x}{\|w^*\|} = \frac{\gamma}{\|w^*\|} \right\} \quad \text{Si} \quad \gamma = \frac{\gamma}{\|w^*\|} \quad \Rightarrow \quad w^* = \frac{w^*}{\|w^*\|}$$

$$\Rightarrow \left\{ x : w^{*,T} x = \gamma' \right\}$$

Geometricamente :



- Podemos reescalar arbitrariamente para mantener $\sqrt{v} > 0$ (mientras $v' > 0$).
- Cuándo el algoritmo erra :

$$w^{l+1} = w^l + x \cdot y$$

$$\begin{aligned}\|w^{l+1}\|^2 &= \|w^l + xy\|^2 = (w^l + xy)^T (w^l + xy) \\ &= \|w^l\|^2 + 2 \underbrace{(w^l)^T x}_\text{error} y + \|x\|^2 y^2\end{aligned}$$

$$\leq \|w^l\|^2 + 0 + R^2$$

Asumimos esp.
finitos

$$\|w^{l+1}\|^2 \leq \|w^l\|^2 + R^2$$

Diferente signo!

- Quiere decir que el error puede crecer a lo más en R^2 .
- Si asumimos una cadena de errores, recursivamente:

$$\|w^{l+1}\|^2 \leq (\|w^{l-1}\|^2 + R^2) + R^2$$

$$\|w^{l+1}\|^2 \leq \|w^0\|^2 + l \cdot R^2$$

$\text{Si: } w^0 = \vec{0}$

$$\boxed{\|w^{l+1}\|^2 \leq l \cdot R^2}$$

} después de corregir l errores (1)

Segunda parte:

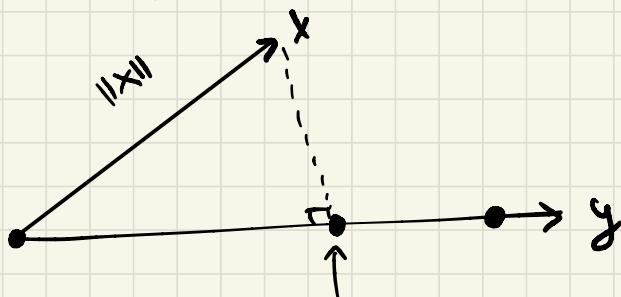
$$\begin{aligned}
 (w^{l+1})^T w^* &= (w^l + x y)^T w^* \\
 &= w^{l T} w^* + y^T x^T w^* \quad \downarrow (?) \\
 &\quad + (w^* x)^T y \quad \} > \sqrt{(\text{Junquies con separación})}
 \end{aligned}$$

$$\begin{aligned}
 (w^{l+1})^T w^* &> w^{l T} w^* + v \quad \downarrow \text{Recursivamente} \\
 (w^{l+1})^T w^* &> (w^{l-1 T} w^* + v) + v
 \end{aligned}$$

$$\Rightarrow (w^{l+1})^T w^* > l \vee \quad (2)$$

- En la medida que cometes errores el producto $(w^{l+1})^T w^*$ crece.
- En (1) sabemos que hay un límite para el error.
- Hay que relacionar (1) y (2).

Para Walquier x, y :



$$\left(\frac{x^T y}{\|y\|} \right) y$$

$$\left\| \left(\frac{x^T y}{\|y\|} \right) y \right\|^2 \leq \|x\|^2 \quad (\text{Pitágoras})$$

\Leftrightarrow

$$\left(x^T y \right)^2 \frac{\|y\|^2}{\|y\|^2} \leq \|x\|^2$$

$$\Leftrightarrow (x^T y)^2 \leq \|x\|^2 \|y\|^2 \quad \text{Cauchy-Schwarz} \heartsuit$$

Luego, desde (2) :

$$l \vee \leq (w^{l+1})^T w^*$$

$$\Rightarrow l^2 \vee^2 \leq [(w^{l+1})^T w^*]^2 \stackrel{\text{Cauchy-Schmitz}}{\leq} \underbrace{\|w^{l+1}\|^2 \|w^*\|^2}_1$$

$$\Rightarrow \|w^{l+1}\|^2 \geq l^2 \vee^2 \quad (3)$$

Es decir :

$$l^2 \vee^2 \leq \underbrace{\|w^{l+1}\|^2}_{\substack{\text{Error total} \\ l+1 \\ \text{iteraciones}}} \leq l R^2 \quad (2)$$

$$\Rightarrow l^2 \vee^2 \leq l R^2$$

$$\Rightarrow l \leq R^2 / \vee^2$$

l : número de errores

↳ Está limitado.

↳ Límite del radio del error.

⇒ El perceptrón converge.

- R es calculable
- $\sqrt{2}$ es arbitrario
- \therefore Dado un grupo de datos
es posible cometer un número
finito de errores.

□

Esta información quedó pendiente!

Entrenando a través del Gradiente

(Visto en
clases)

$$\begin{array}{ccc} x_1 & \xrightarrow{\theta_1} & \\ x_2 & \xrightarrow{\theta_2} & z = \theta_1 x_1 + \theta_2 x_2 + b \xrightarrow{\sigma(\cdot)} \hat{y} \\ b & \xrightarrow{\quad} & \sigma(z) = \frac{1}{1 + e^{-z}} \end{array}$$

* Recordando que $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Queremos medir como cambia nuestro $L(\cdot)$ dados pequeñas perturbaciones en w_1, w_2 y b .

Sea :

$$L(\hat{y}, y) = -[y \log(\hat{y}) + (1-y) \log(1-\hat{y})]$$

Para ello debemos calcular las derivadas parciales :

$$\frac{\partial L}{\partial b} = \underbrace{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial b}}_{\frac{\partial L}{\partial \hat{y}}} = - \left[y \frac{1}{\hat{y}} + \frac{(1-y)}{1-\hat{y}} \cdot -1 \right] \cdot \hat{y}^{(1-\hat{y})}$$

$\frac{\partial \hat{y}}{\partial z}$... o 1 ($\partial z / \partial b$)

Un poquito de álgebra :

$$\frac{\partial L}{\partial b} = - (y - \hat{y}) = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial b}$$

$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial \theta_1} = -(y - \hat{y}) x_1$$

$$\frac{\partial L}{\partial \theta_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial \theta_2} = -(y - \hat{y}) x_2$$

Vía Gradiente Descendiente :

$$\theta_1^{k+1} = \theta_1^k - \alpha \frac{\partial L}{\partial \theta_1}$$

$$\theta_2^{k+1} = \theta_2^k - \alpha \frac{\partial L}{\partial \theta_2}$$

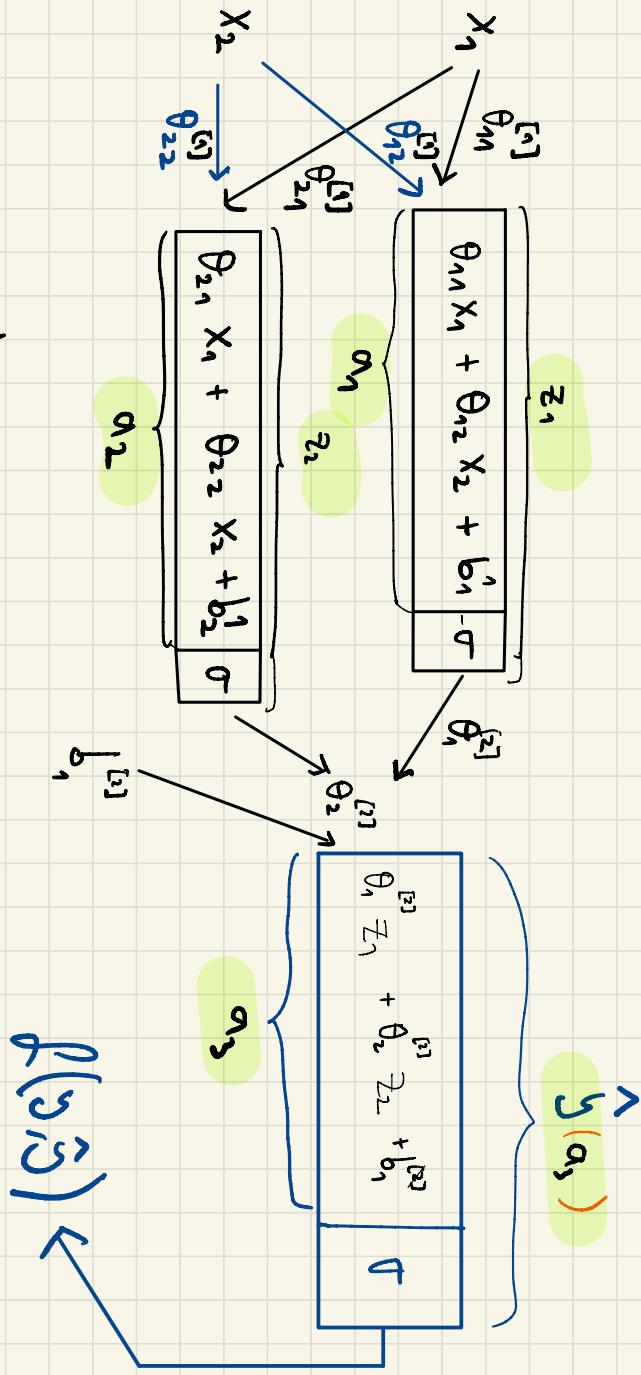
$$b^{k+1} = b^k - \alpha \frac{\partial L}{\partial b}$$

Complejaremos un poco más las cuentas :

$\Theta_{i,j}^{[k]}$ → i : Neurona
 j : Componente

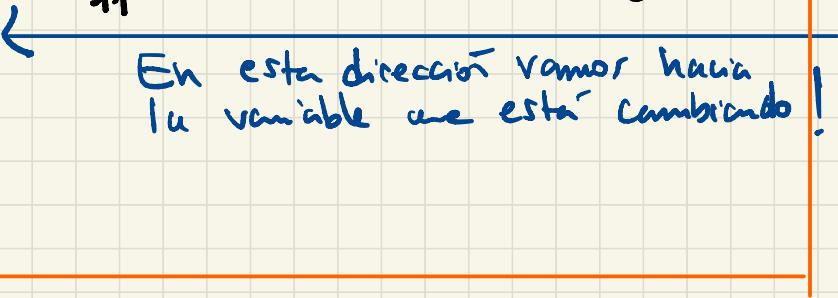
$\sigma(\cdot)$: capa o layer

\hat{y} : Binary Cross Entropy Loss .



Como ejemplo calculemos un par de derivadas :

$$\frac{\partial L}{\partial \theta_{11}} = \frac{\partial a_1}{\partial \theta_{11}} \cdot \frac{\partial z_1}{\partial a_1} \cdot \frac{\partial a_3}{\partial z_1} \cdot \frac{\partial \hat{y}}{\partial a_3} \cdot \frac{\partial L}{\partial \hat{y}}$$


En esta dirección vamos hacia la variable que está cambiando!

$$\frac{\partial L}{\partial \hat{y}} = -\frac{(y - \hat{y})}{\hat{y}(1 - \hat{y})}$$

$$\frac{\partial v_1}{\partial \theta_{11}} = x_1$$

$$\frac{\partial \hat{y}}{\partial a_3} = \hat{y}(1 - \hat{y}) \quad \therefore \frac{\partial L}{\partial \theta_{11}} = -x_1 \Theta_1^{[2]} z_1 (1 - z_1) (y - \hat{y})$$

$$\frac{\partial a_1}{\partial a_1} = \Theta_1^{[2]}$$

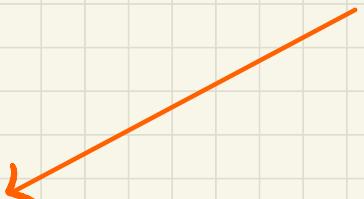
$$\frac{\partial z_1}{\partial a_1} = z_1(1 - z_1)$$

Observe que para otra derivada parcial tenemos :

$$\frac{\partial L}{\partial \theta_{12}} =$$

$$\frac{\partial a_1}{\partial \theta_{12}}$$

$$\boxed{\frac{\partial z_1}{\partial a_1} \quad \frac{\partial a_3}{\partial z_1} \quad \frac{\partial \hat{y}}{\partial a_3} \cdot \frac{\partial L}{\partial \hat{y}}}$$



Sin embargo, todas estas derivadas ya las calculamos. El algoritmo llamado Backpropagation o retropropagación almacena estos cálculos para evitar recalcular todo nuevamente.

- Técnicamente se denomina "Forward Pass" al cálculo de \hat{y} ("hacia adelante").
- Luego, calculamos el costo J
- Finalmente, viene el "Backward Pass" o hacia atrás donde calculamos los gradientes. Almacenando,

aquellos que no tenemos. Esto "propaga" los gradientes hacia atrás.

Resumen

- Hemos revisado la definición de Neurona artificial.
- Además, definimos lo que es una red neuronal.
- Revisamos parte de la historia y evolución del modelo.
- Quedó pendiente establecer la antigua regla para entrenar un Perceptrón.
- Revisamos el entrenamiento basado en el gradiente. El grafo del modelo permite computacionalmente almacenar el circuito de gradientes que podemos reutilizar.

Si M es un modelo, $x \in \mathbb{R}^d$ es un vector de entrada (información) entonces llamaremos de:

i) "Forward Pass" a evaluar el modelo en x , es decir :

$$M(x) = \hat{y}$$

donde \hat{y} es la salida del modelo.

ii) Luego calculamos los gradientes y almacenarlos

gradientes repetidos : esto se conoce como
Backpropagation.

iii) luego , actualizamos los pesos ($W \cdot \Theta$)
y calculamos los gradientes para repetir el
proceso.

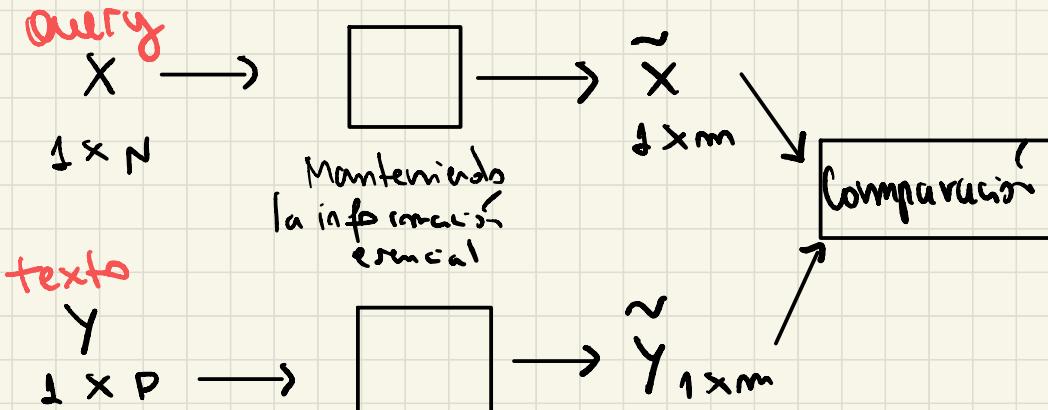
- Revisamos la implementación de una
neurona artificial o de una red multicapa
utilizando **torch**.
- Quedó de tarea revisar vía **Skearn**.
- Fueron enviados dos pequeños tutoriales
OSX / Linux para manejo de ambientes en
python.

Otra mirada de la red neuronal: Como un bloque

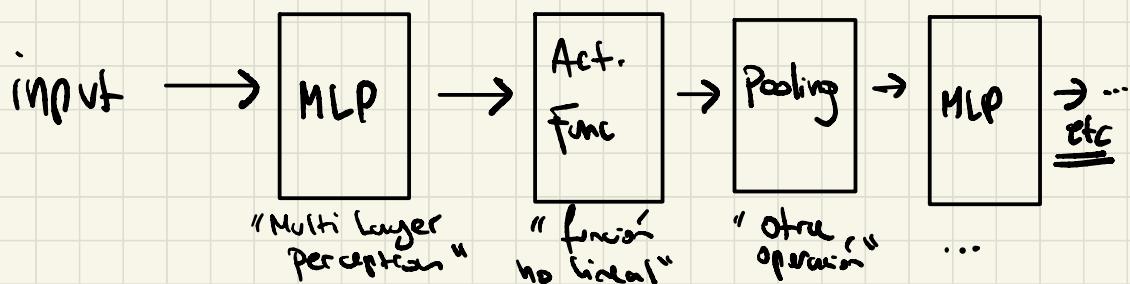
Una perspectiva interesante es pensar la red neuronal como un conversor de la data a un espacio vectorial determinado o de interés.

Por ejemplo, en el caso de un clasificador, la tendencia del modelo es a ~~comprimir~~ los datos.

En otras aplicaciones, como modelos de lenguaje para búsqueda semántica, queremos convertir un texto y una query o búsqueda del usuario, a vectores en un mismo espacio vectorial:



Podemos pensar intuitivamente a las redes como Extractores → compresores de información que permiten unir diferentes componentes o funciones:



* Explicar proyecto + Mostrar notebook t-SNE

* Construir notebook Perceptrón antiguo (Proyecto).

* Construir notebook Perceptrón back propagation

Mostrar resultados sintéticos !.

Otro notebook :

Dentro de assignments :

① t-SNE - audio.ipynb

② PDF torch

③ neural-network - scratch.ipynb

Luego : one-single-neural-torch.ipynb (Hasta Entrenamiento)

Entregar
de assignments :

new

(con audio)

Hasta

Entrenamiento