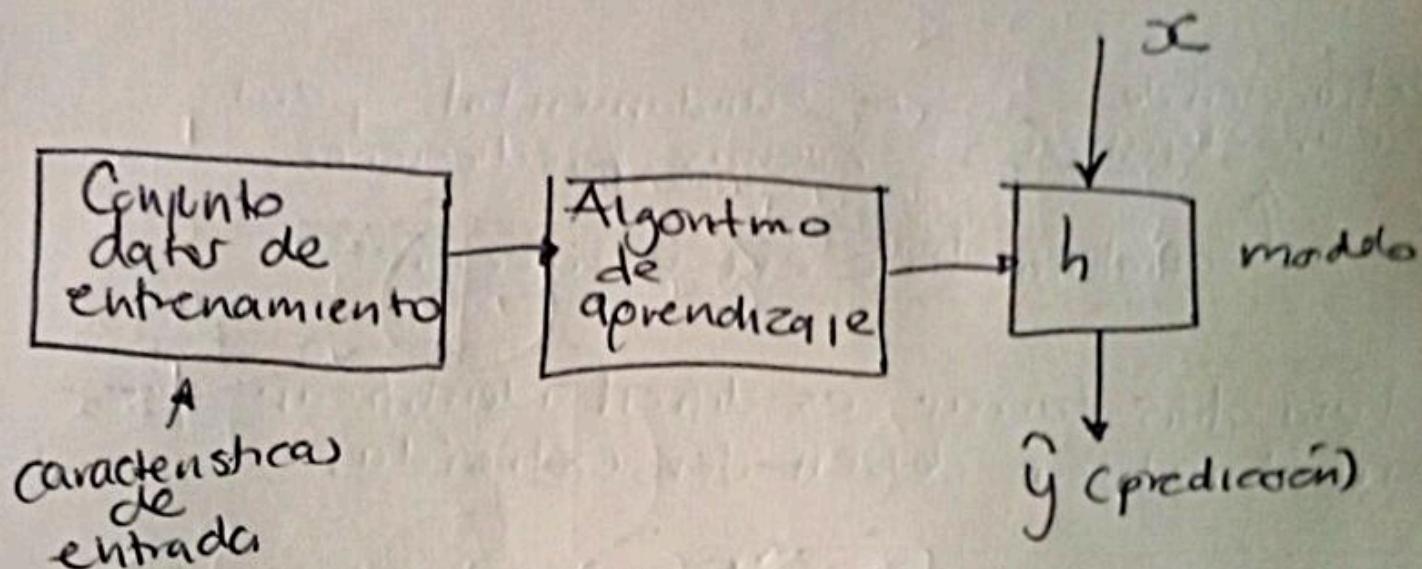
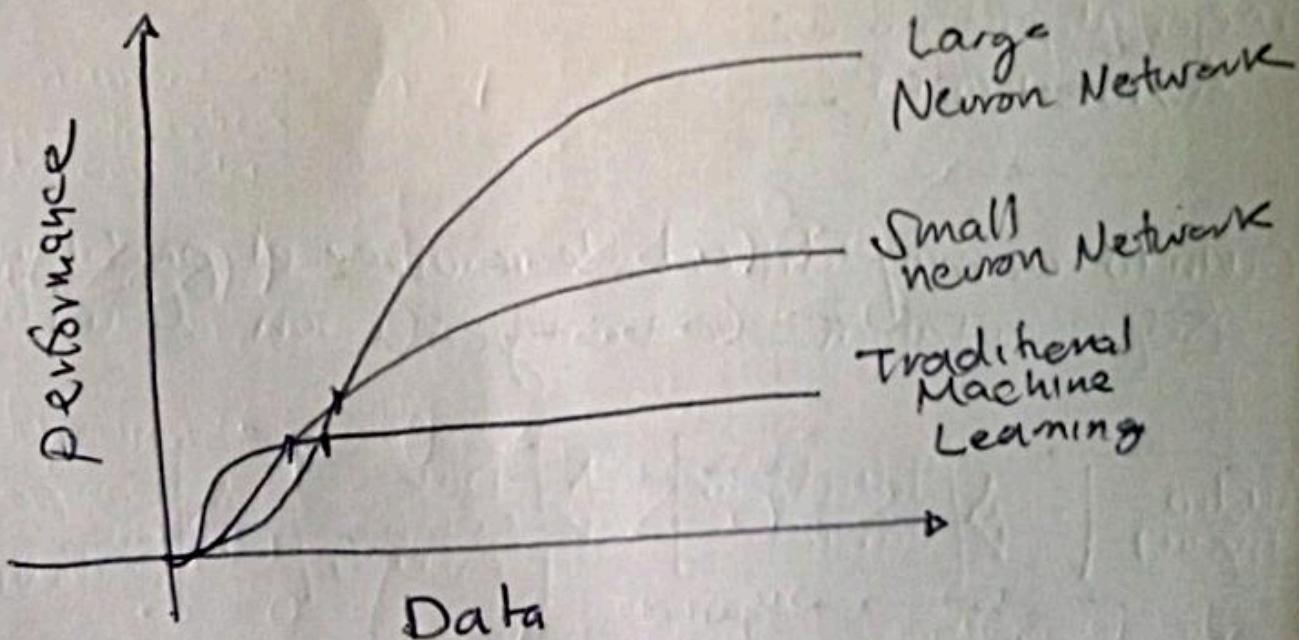


¿Qué es Deep Learning?

Es un subdominio del Machine Learning que resuelven el problema de las técnicas de Representación Learning introduciendo representaciones que se expresan en términos con representación.



Aplicación 1 clasificación de deep learning



Deep learning ha conseguido resultados excelentes en áreas complejas que no podían resolverse con técnicas de ML

- Clasificación de imágenes
- Conducción autónoma
- Reconocimiento de audio.
- Simulación de fotos y video

- Redes Neuronal Artificial Profunda
- Redes Neuronal Convencional
- Redes Neuronal Recurrente
- " " " Generativas Antagonistas

[Introducción a las Redes Neuronal Artificial]

- Tipo de algoritmo de ML inspirado en las redes neuronales biológicas
- Componente principal de Deep Learning
- Se introducen por primera vez en 1943 por Warren McCulloch y Walter Pitts.

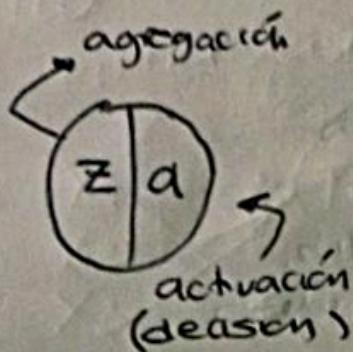
Neurona McCulloch & Pitts

Es la primera neurona artificial inventada en 1943. Se caracteriza porque recibe una o más señales binarias ($1, 0 \dots$) y retorna otra señal binaria ($1, 0 \dots$).

Se activa su salida cuando más de un número de valores de valores de entradas se encuentran activos.

Debe establecerse un threshold manual.
o límite

Funcionamiento Neurona MP



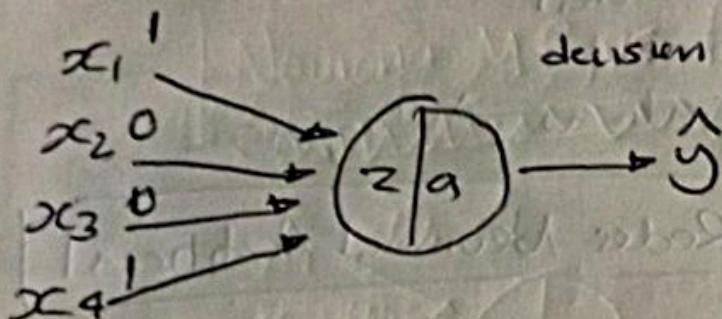
→ Voy al cine hoy?

$x_1 \Rightarrow$ Fin de semana?

$x_2 \Rightarrow$ Tengo tareas a realizar?

$x_3 \Rightarrow$ Está el cine cerrado?

$x_4 \Rightarrow$ Estrenan película decisión S: Si voy a ir hoy?
D: No voy a ir.



$$z = (x_1, x_2, x_3, x_4) = z(x)$$

$$z(x) = \sum_{i=1}^n x_i = x_1 + x_2 + x_3 + x_4$$

n = numero de
Características

de entrada

threshold

$$a(z(x)) = \begin{cases} 1 & \text{si } z(x) \geq \theta \\ 0 & \text{si } z(x) < \theta \end{cases}$$

Suponiendo que $\theta = 2$

$$\begin{aligned} x_1 &= 1 \\ x_2 &= 0 \\ x_3 &= 0 \\ x_4 &= 1 \end{aligned}$$

$$z(x) = 2$$

$$\text{así que } a(z(x)) = 1, \quad \hat{y} = 1$$

2 tipos de inputs que definen

Inputs → inhibidores ($-x_1, -x_2, -x_3$)

Inputs → excitadores (x_3)

Función lógica AND

$$x_1 > \Theta = 0 \rightarrow y$$

$$x_2 > \Theta = 0 \rightarrow y$$

excitadores

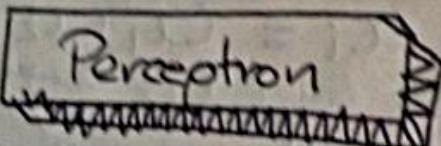
función lógica OR

$$x_1 > \Theta = 1 \rightarrow y$$

$$x_2 > \Theta = 1 \rightarrow y$$

Limitaciones

- Requiere la selección de threshold manual
- Todas las entradas son iguales
- No son capaces de resolver problemas que no sean linearmente separables, por ejemplo la operación XOR

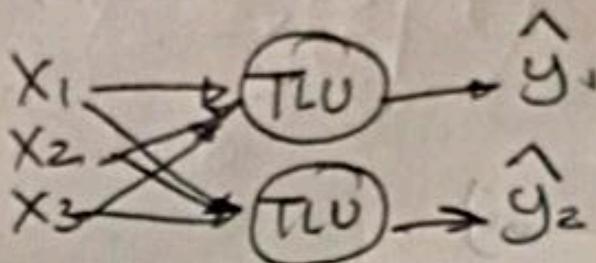


RNA → Perceptron

- Fue propuesto por Frank Rosenblatt en 1958.
- Mejora el planteamiento de McCullough Pitts añadiendo el concepto de "peso" numérico a las entradas y plantando un mecanismo para ajustarlo.
- No recibe únicamente valores de entrada binaria, permite valores de entrada reales.
- Se basa en un tipo de neurona conocida TLU

Threshold Logic Unit (TLU)

- La TLU computa una suma parametrizada las entradas



Neurona MP vs Percepción

$$\begin{array}{l} 1 \ x_1 \ w_1=2 \\ 1 \ x_2 \ w_2=1 \\ 0 \ x_3 \ w_3=1 \end{array}$$

$\theta = 2$

threshold

$x_1 \rightarrow$ d Fin de semana?
 $x_2 \rightarrow$ d Estrenan película?
 $x_3 \rightarrow$ d No tengo
tareas pendientes?

$$y = 1$$

$$z(x_1, x_2, x_3) = \sum_{i=1}^n x_i = (x_1 + x_2 + x_3)$$

$$a(z(x)) = \begin{cases} 0 & \text{si } z(x) \geq \theta \\ 1 & \text{si } z(x) < \theta \end{cases}$$

Introduciendo un peso.

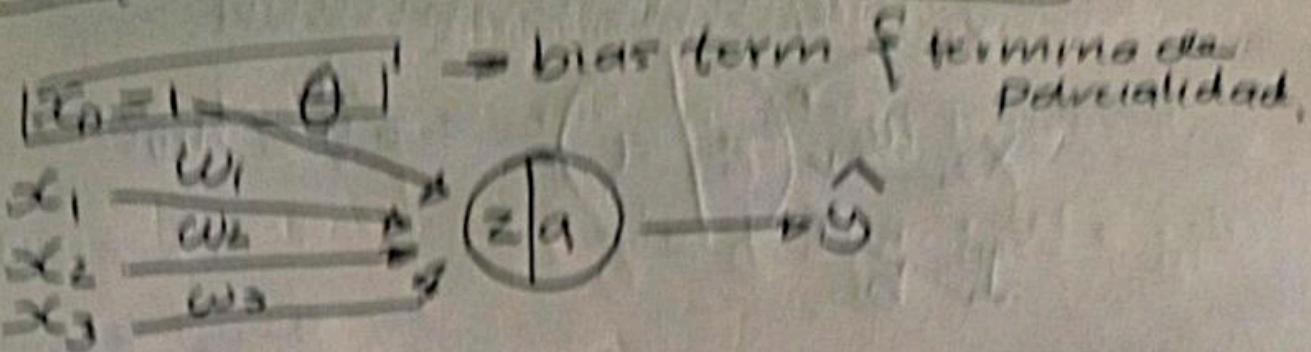
$$z(x_1, x_2, x_3) = \sum_{i=1}^n x_i \cdot w_i$$

$$= x_1 w_1 + x_2 w_2 + x_3 w_3 + \dots + x_n w_n$$

$$z(x_1, x_2, x_3) = (1)(z) + (0)(1) + (0)(1)$$

$$= 2 \quad \therefore \hat{y} = 1$$

Recepción y eliminación del threshold



$$Z(x) = \sum_{i=0}^n x_i \cdot w_i = x_0 w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3 + \dots + x_n w_n$$

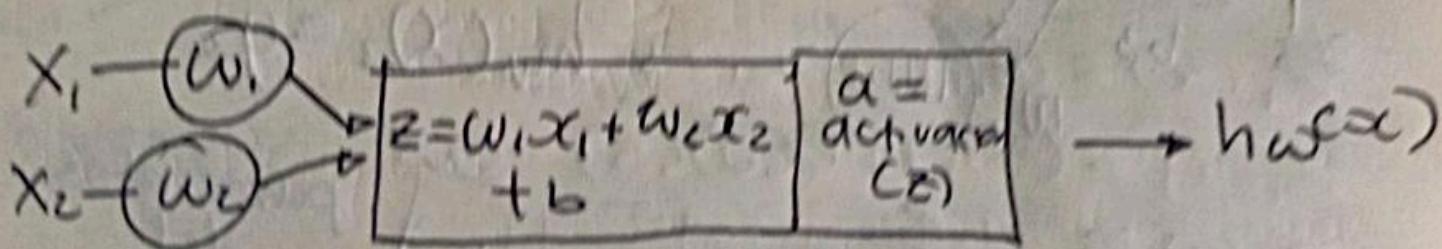
dónde

$$Z(x) = \sum_{i=0}^n x_i \cdot w_i = \theta + x_1 w_1 + x_2 w_2 + \dots + x_n w_n$$

$$a(Z(x)) = \begin{cases} 1 & \text{si } Z(x) \geq 0 \\ 0 & \text{si } Z(x) < 0 \end{cases}$$

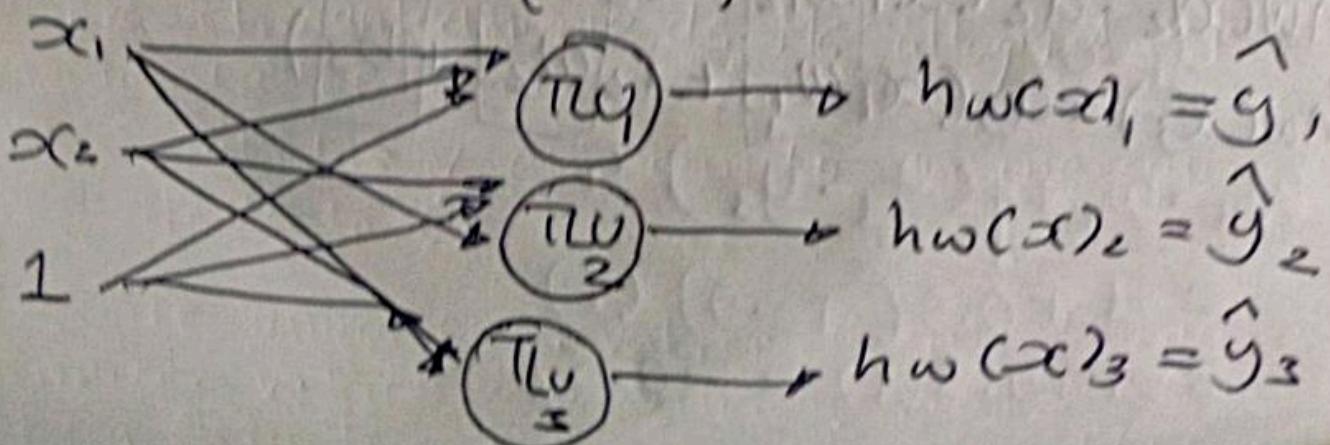
TLU computa una suma parametrizada de las entradas

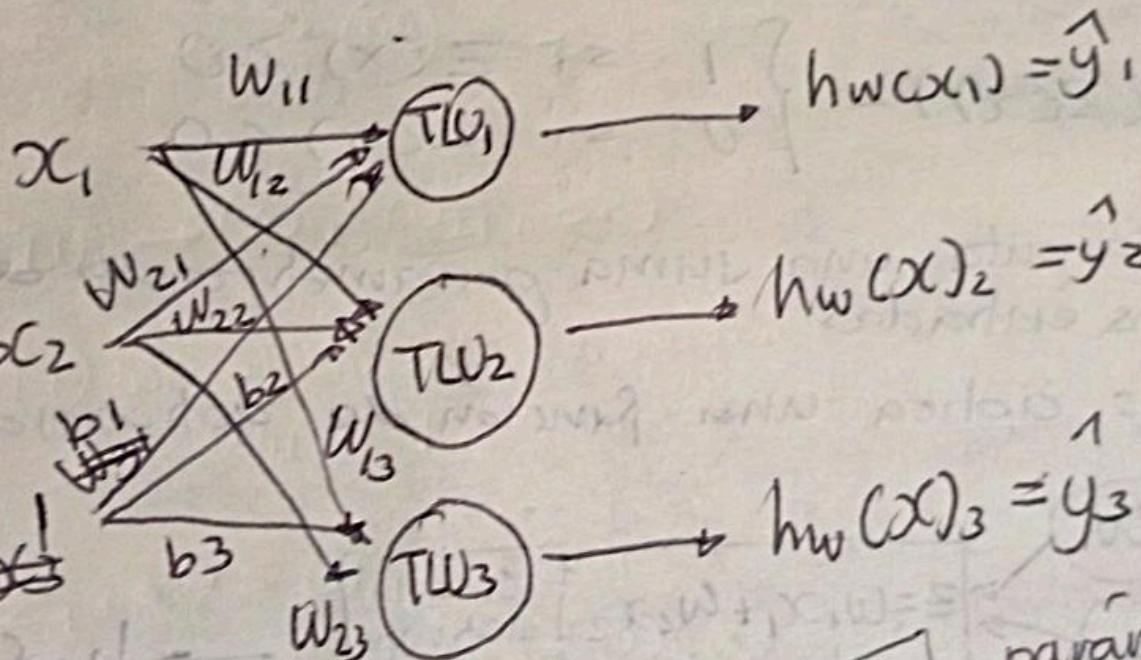
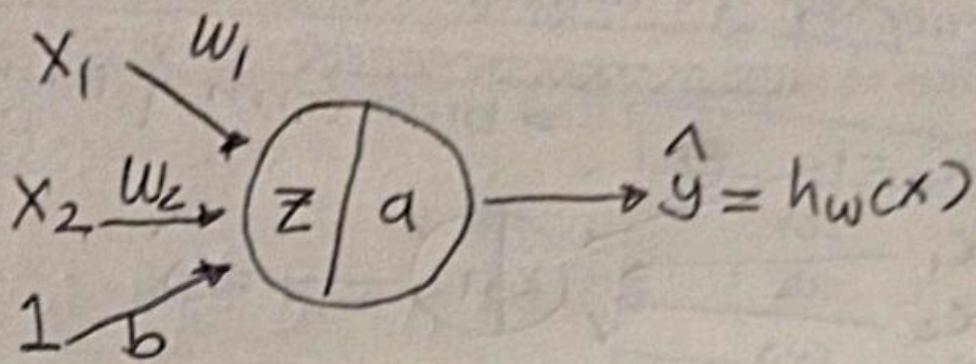
después aplica una función de activación.



Input layer

Output layer





$$h_w(x)_1 = a_1(x_1w_{11} + x_2[w_{21} + b_1])$$

$$h_w(x)_2 = a_2(x_1w_{12} + x_2[w_{22} + b_2])$$

$$h_w(x)_3 = a_3(x_1w_{13} + x_2[w_{23} + b_3])$$

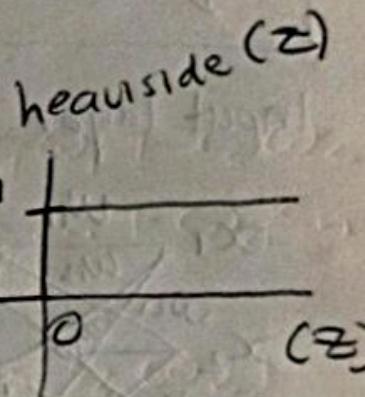
Funciones de activación

El perceptrón permite la clasificación de instancias en clases binarias de manera simultánea.

Funciones más comunes son:

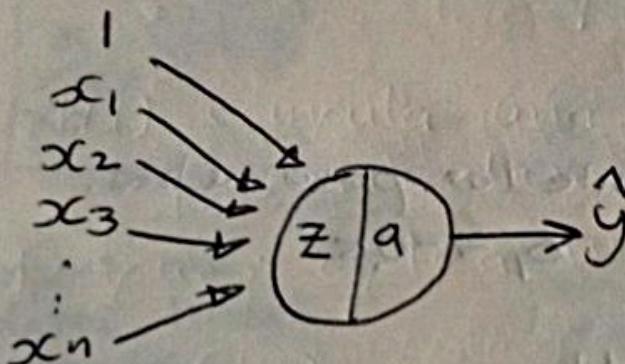
- Heaviside step function

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$



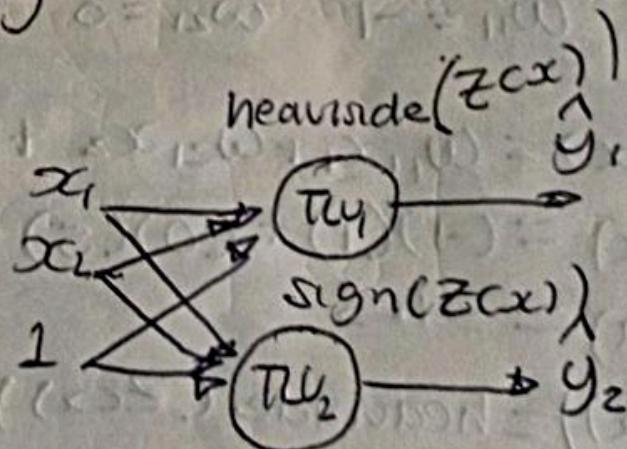
- Sign function

$$\text{sign}(z) = \begin{cases} 0 & \text{if } z = 0 \\ 1 & \text{if } z > 0 \\ -1 & \text{if } z < 0 \end{cases}$$



$$z(x) = x_1 w_1 + x_2 w_2 + \dots + x_n w_n + b$$

$n = \# \text{ inputs features}$

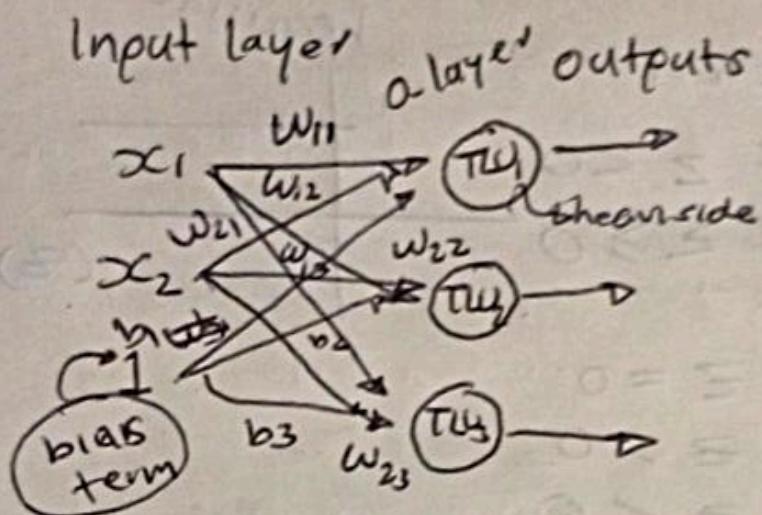


Perceptron : Construcción de modelo

Entrenamiento del perceptrón

$$w_{i,j} = w_{i,j} + \eta (y_j - \hat{y}_j) x_i$$

(x_i) (x_e) y
#faulty #html y



<u>email 1</u>	1	5	1
email 2	1	0	0
:	=	(=)	:
email m	:	:	:

$$T_{LU_1} \rightarrow (\omega_{11}, \omega_{21}, b_1)$$

- ① Inicializar aleatoriamente los parámetros.

$$\omega_{11} = -1 \quad \omega_{21} = 0 \quad \delta_1 = -0.5$$

$$\tilde{z}_1(x) = w_{11}x_1 + w_{21}x_2 + b_1$$

$$\Xi_1(\alpha) = (-1)(1) + (0)(5) - 0.5$$

$$z_1(x) = -1.5$$

$$a_1(z\alpha)) = \text{height}(z\alpha) = 0$$

$\hat{y}_1 = 0.1$ asignar $\eta = 0.5$ η = learning rate

$$W_{11} = w_{11} + \eta (y_1 - \hat{y}_1) x_1$$

$$= -1 + 0.8(1-0)(1)$$

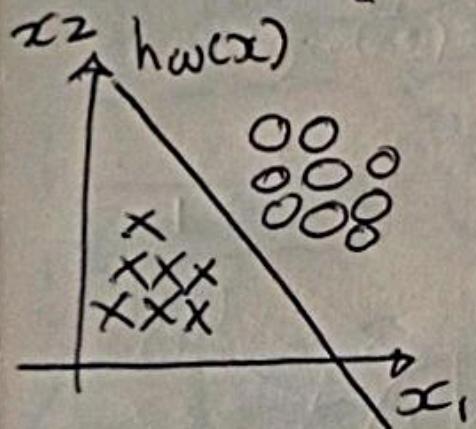
$$= -1 + 0, S(1) = \underline{-0,5}$$

parámetros que determinan la velocidad.

$$z_1(x) = 0.5(1) + 0(-5) + (-0.5) = -1$$

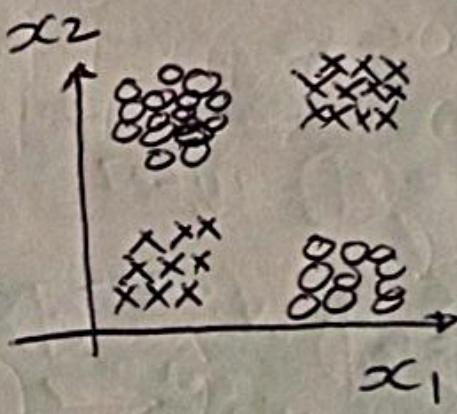
Limitaciones

- El perceptrón no proporciona como resultado una probabilidad, sino con un threshold estático.
- Construye límites de decisiones lineales.
- El uso del perceptrón tiene grandes limitaciones no permite resolver problemas tan sencillos como el problema de clasificación XOR.
- Muchas limitaciones del perceptrón se solucionan añadiendo capas TLUs apareciendo Multi-Layer-Perceptron.



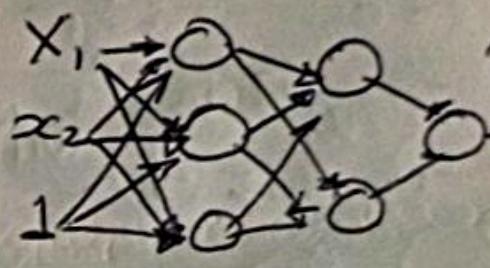
conjunto de datos lineal separable

Se puede resolver con perceptron



conjunto de datos NO linealmente separable

No se puede resolver con perceptron.

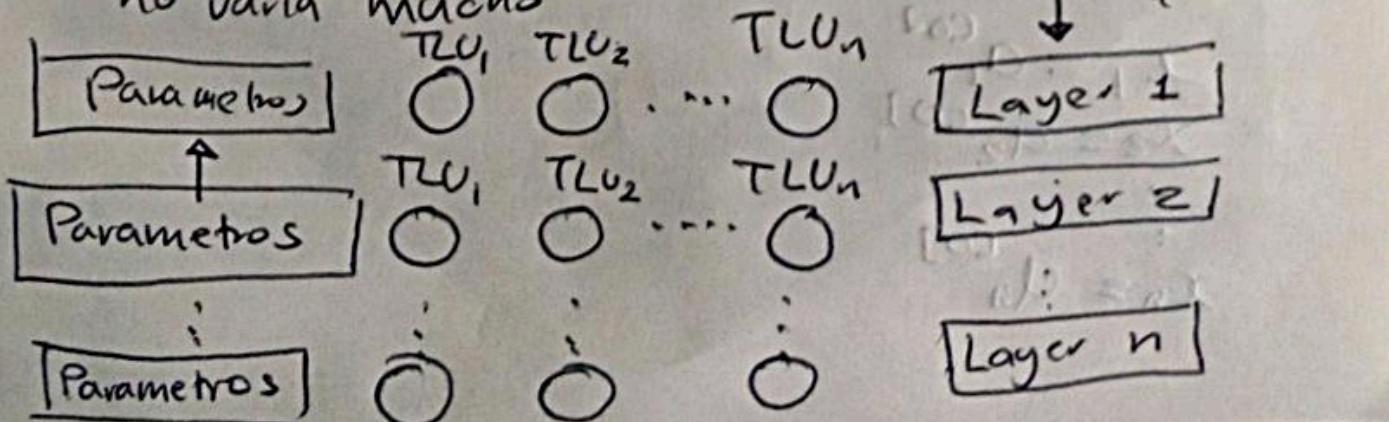
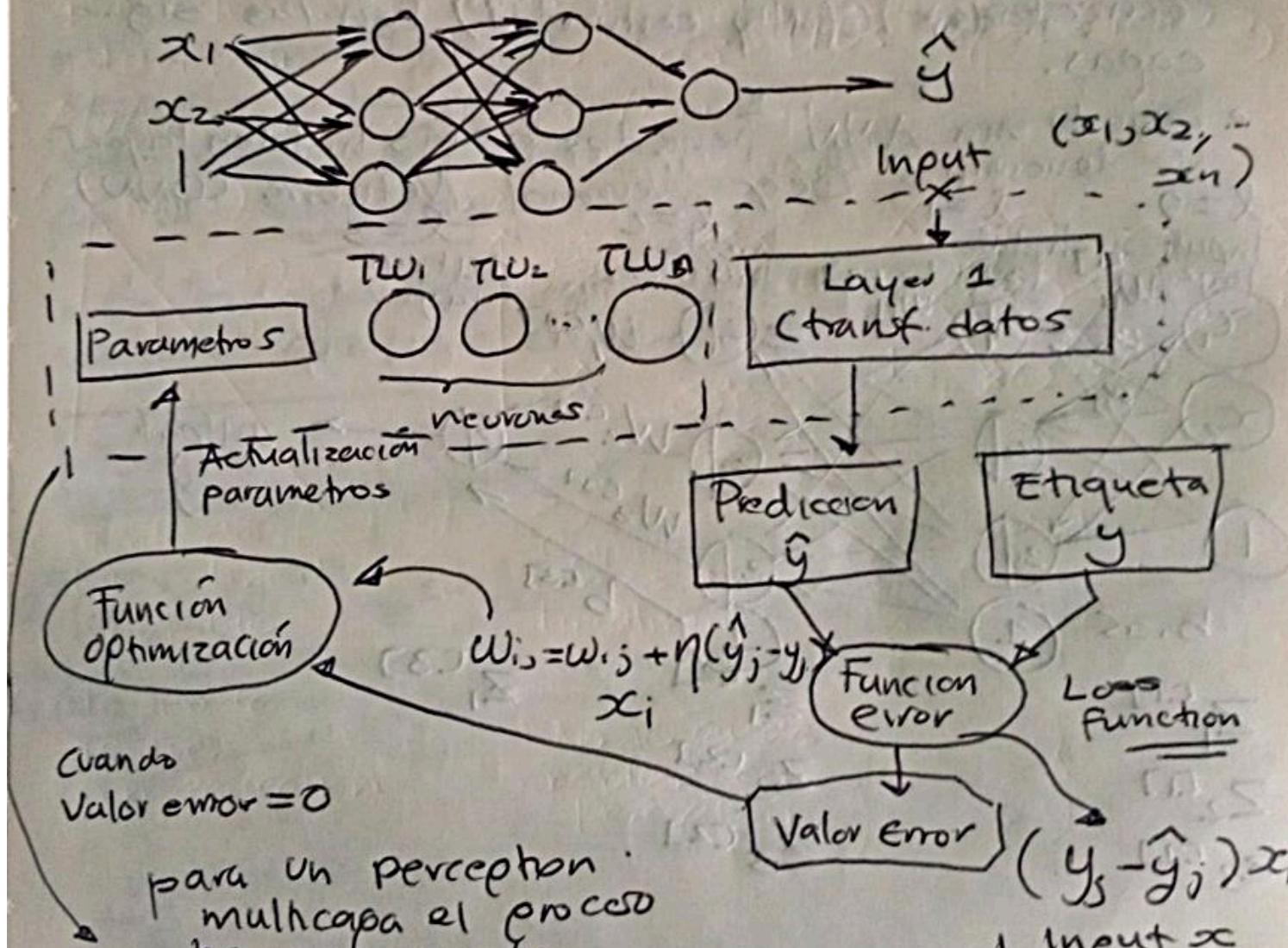


Perceptron Multi capa.

Perceptron Multicapa

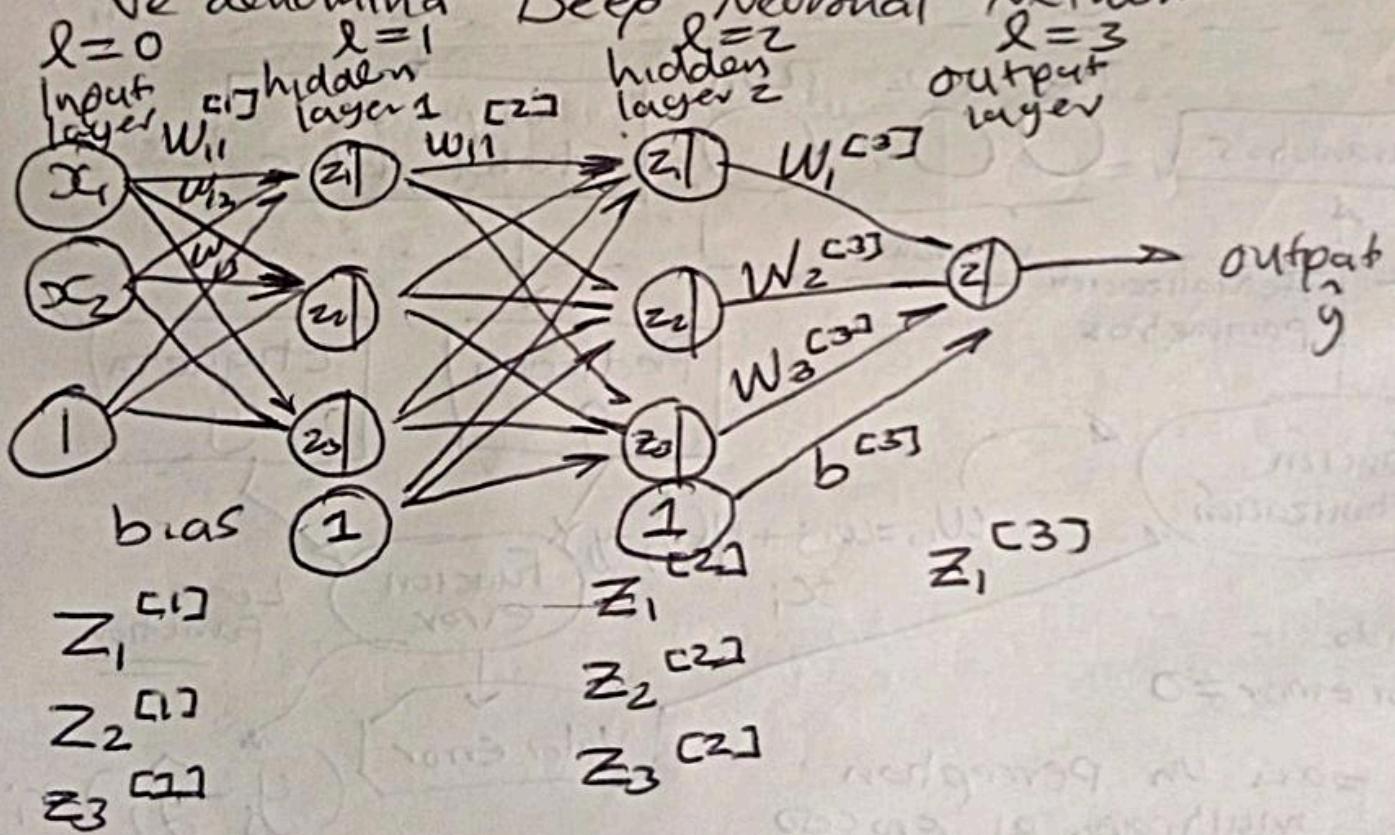
Deep Feedforward network es un modelo que ha sido muy popular en el Deep Learning.

Se denomina feedforward porque la información influye desde las entradas hasta la salida sin conexiones de feedback.



- El perceptrón Multicapa se compone de una input layer y una o más capas de TLUs
- Cada capa de TLUs intermedia se denomina hidden layer.
- La capa final TLUs se denomina output layer.
- Todas las capas excepto la output layer incluyen las bias neuron.
- Todas las capas se encuentran totalmente conectadas (fully connected) con las sig capas.

• Cuando una ANN tiene dos o más hidden layers se denomina Deep Neural Network (DNN),



$$x_1 = a_1^{[0]}$$

$$x_2 = a_2^{[0]}$$

$$\vdots$$

$$x_n = a_n^{[0]}$$

layer 1

$$z_1^{[1]} = W_{11}^{[1]} a_1^{[0]} + W_{21}^{[1]} a_2^{[0]} + b_1^{[1]}$$

$a_1^{[1]}$ = activación ($z_1^{[1]}$) ← input de layer 2

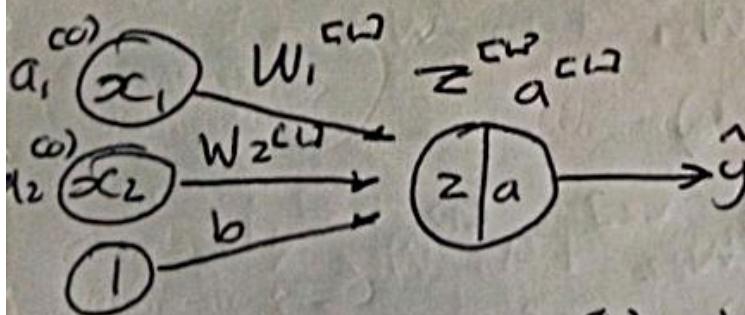
$$z_1^{[2]} = W_{11}^{[2]} a_1^{[1]} + W_{21}^{[2]} a_2^{[1]} + W_{31}^{[2]} a_3^{[1]} + b_1^{[2]}$$

$a_1^{[2]}$ = activación ($z_1^{[2]}$)

• El cambio clave del algoritmo back propagation respecto a los utilizados anteriormente fue reemplazar la función Heaviside step function por la sigmoide

• en la actualidad existen funciones las activaciones populares como tanh(z) o ReLU(z)

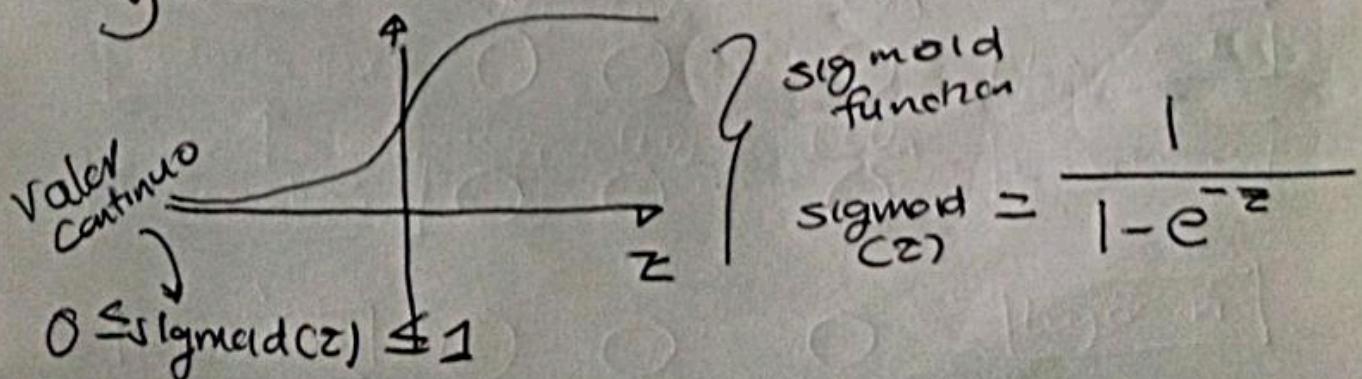
Función de activación Perceptron multicapa.



$$z = x_1 w_1 + x_2 w_2 + b = a_1^{[0]} w_1^{[1]} + a_2^{[0]} w_2^{[1]} + b^{[1]}$$

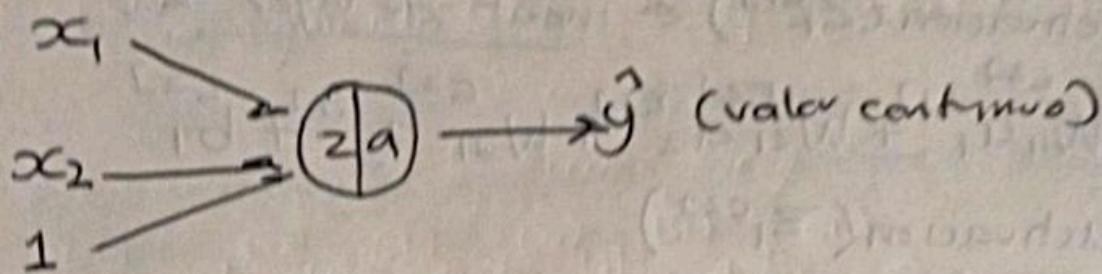
\hat{y} = activación ($z^{[1]}$) = heaviside ($z^{[1]}$) ✗

\hat{y} = activación ($z^{[1]}$) = sigmoid ($z^{[1]}$) ✓



Probabilidad para función sigmoidal

$$0 \leq \text{sigmoid}(z) \leq 1$$



$$\hat{y} = a(z(x)) = \text{sigmoid}(x_1w_1 + x_2w_2 + b)$$

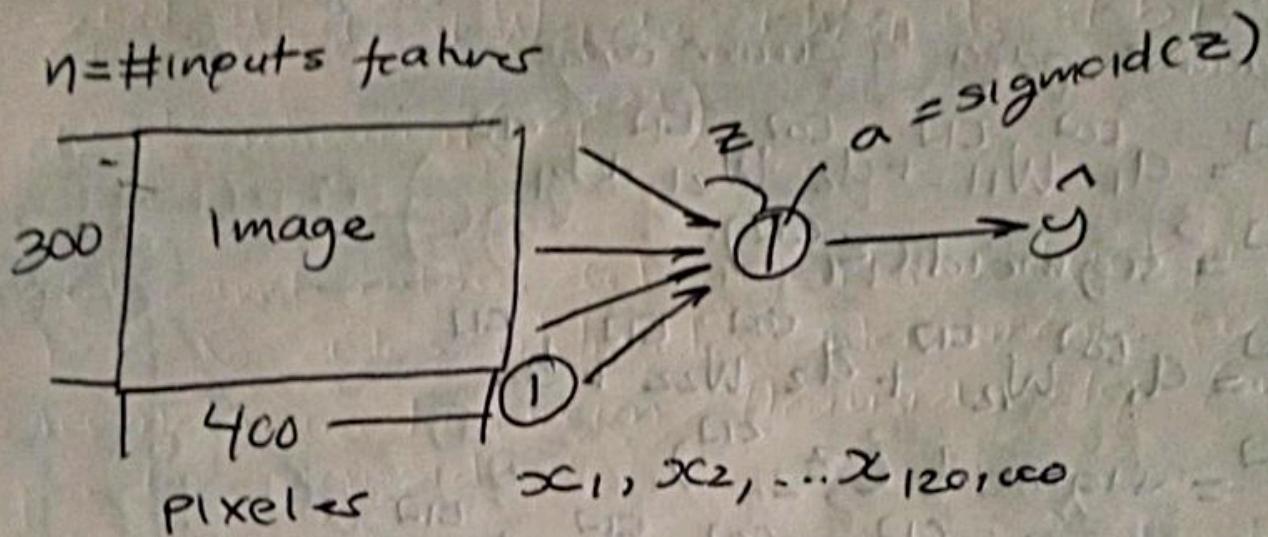
$$\boxed{\hat{y} = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}}$$

↳ probabilidad que pertenezca a la clase positiva ($y=1$)

Forward Propagation

Fluye de las entradas a las salidas.
Se utiliza para entregar una red neuronal.

$n = \# \text{ inputs features}$

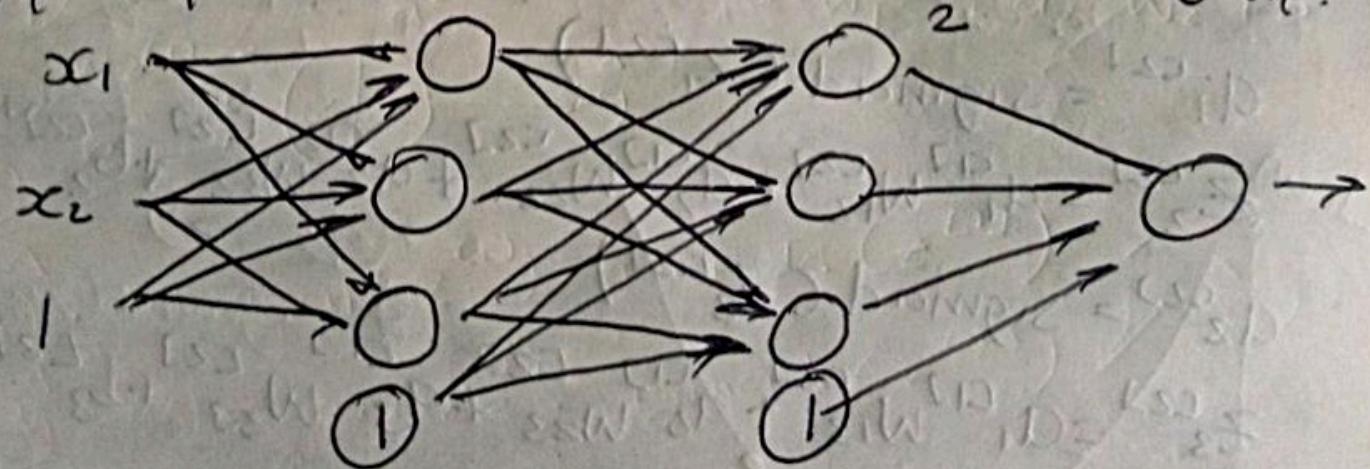


$$n = 300 \times 400 = 120,000$$

$$z = x_1 w_1 + x_2 w_2 + \dots + x_{120,000} w_{120,000} + b$$

$$\hat{y} = a = \text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

Input layer Hidden layer 1 Hidden layer 2 Outp.



El cálculo de todos los términos, agrupación y activación corresponde al Forward Propagation.

para la primera neurona de la capa c_1 .

$$\text{TLU}_1 = z_1^{[1]} = W_{11} a_1^{[0]} + W_{21} a_2^{[0]} + b_1^{[1]}$$

$$a_1^{[1]} = \text{sigmoid}(z_1^{[1]}) = \frac{1}{1 + e^{-z_1^{[1]}}}$$

ecuaciones para la primera capa

$$z_1^{[1]} = a_1^{[0]} W_{11}^{[1]} + a_2^{[0]} W_{21}^{[1]} + b_1^{[1]}$$

$$a_1^{[1]} = \text{sigmoid}(z_1^{[1]})$$

$$z_2^{[1]} = a_1^{[1]} W_{12}^{[1]} + a_2^{[1]} W_{22}^{[1]} + b_2^{[1]}$$

$$a_2^{[1]} = \text{sigmoid}(z_2^{[1]})$$

$$z_3^{[1]} = a_1^{[1]} W_{13}^{[1]} + a_2^{[1]} W_{23}^{[1]} + b_3^{[1]}$$

$$a_3^{[1]} = \text{sigmoid}(z_3^{[1]})$$

— ecuaciones para la segunda capa.

$$z_1^{[2]} = a_1^{[1]} W_{11}^{[2]} + a_2^{[1]} W_{21}^{[2]} + a_3^{[1]} W_{31}^{[2]} + b_1^{[2]}$$

$$a_1^{[2]} = \text{sigmoid}(z_1^{[2]})$$

$$z_2^{[2]} = a_1^{[2]} W_{12}^{[2]} + a_2^{[2]} W_{22}^{[2]} + a_3^{[2]} W_{32}^{[2]} + b_2^{[2]}$$

$$a_2^{[2]} = \text{sigmoid}(z_2^{[2]})$$

$$z_3^{[2]} = a_1^{[2]} W_{13}^{[2]} + a_2^{[2]} W_{23}^{[2]} + a_3^{[2]} W_{33}^{[2]} + b_3^{[2]}$$

$$a_3^{[2]} = \text{sigmoid}(z_3^{[2]})$$

equaciones para la última capa.

$$z^{(3)} = a_1^{(2)} w_1^{(3)} + a_2^{(2)} w_2^{(3)} + a_3^{(2)} w_3^{(3)} + b^{(3)}$$

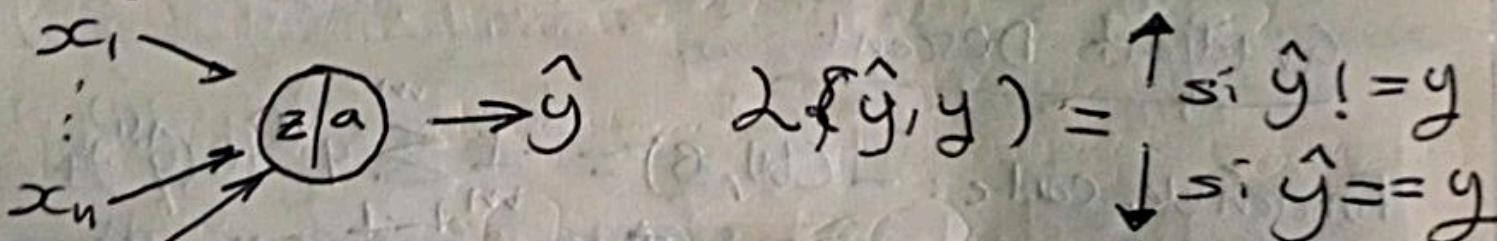
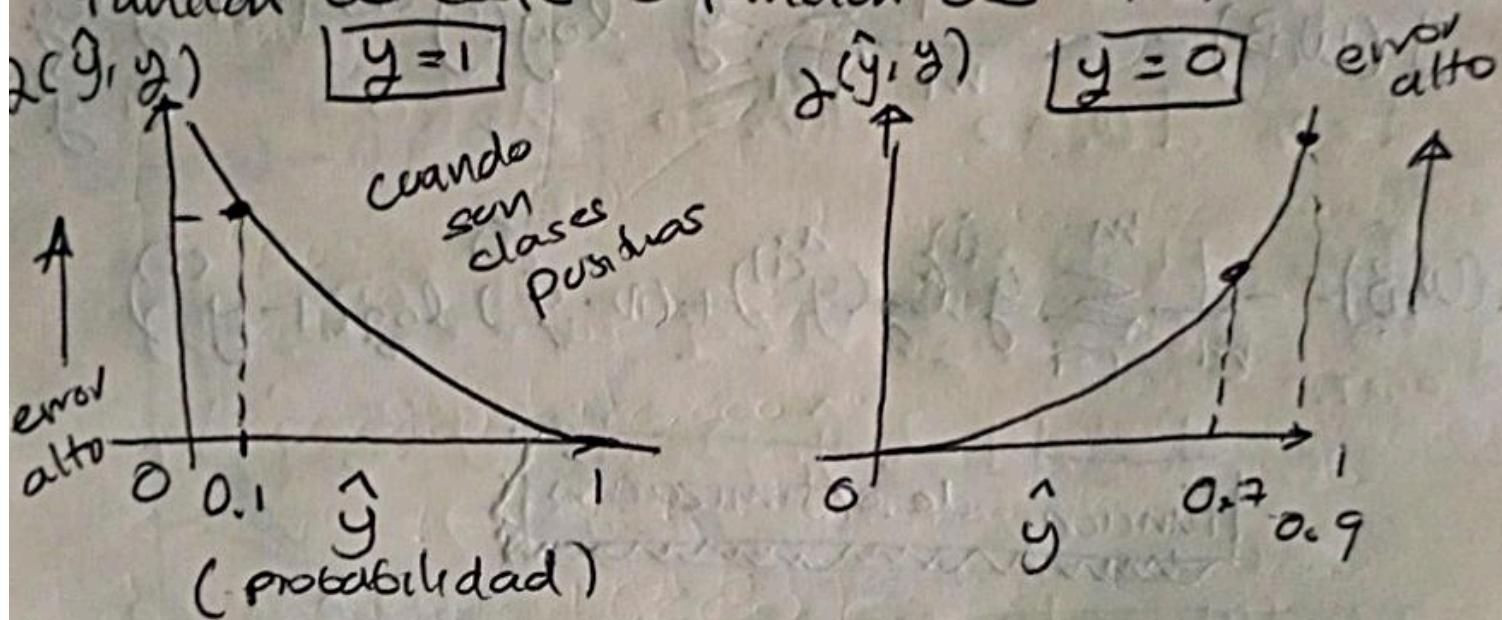
$$\hat{y} = a^{(3)} = \text{sigmoid}(z^{(3)})$$

función de
coste

Función de Coste

Crossentropy

Función de coste o función de error



$$\hat{y} = \text{sigmoid}(x_1 w_1 + \dots + x_n w_n + b)$$

$$\lambda(\hat{y}, y) = -\log(\hat{y}) \quad \text{si } y = 1$$
$$\lambda(\hat{y}, y) = -\log(1-\hat{y}) \quad \text{si } y = 0$$

* es el mismo algoritmo de Regresión Logística
única función de error.

$$\lambda(\hat{y}, y) = -(y \log(\hat{y}) + (1-y) \log(1-\hat{y}))$$

$$J(W, B) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

todos
los
parametros
del
modelo

$$L(\hat{y}^{(i)}, y^{(i)}) = \underbrace{L(\hat{y}^{(i)}, y^{(i)})}_{+} + \underbrace{L(\hat{y}^{(i)}, y^{(i)})}_{\frac{1}{m}} \rightarrow$$

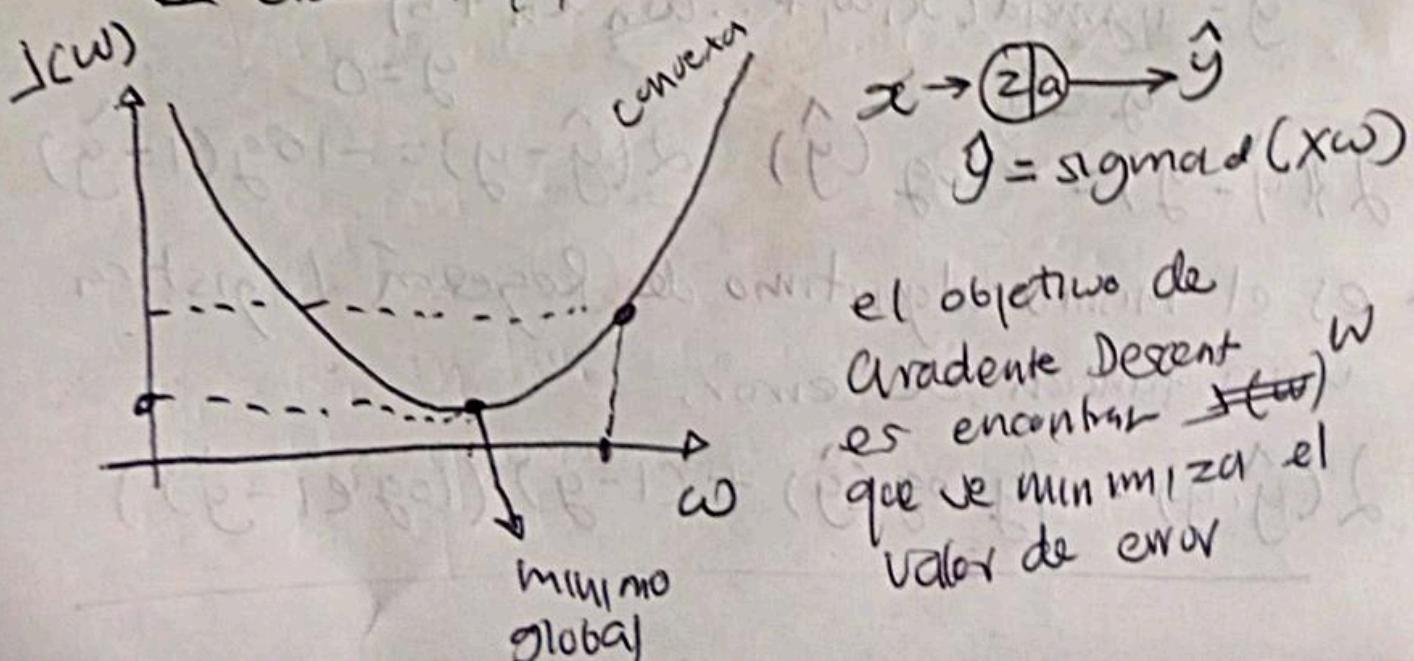
$$J(W, B) = - \frac{1}{m} \sum_{i=1}^m y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

$$J(W, B) = - \frac{1}{m} \sum_{i=1}^m y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

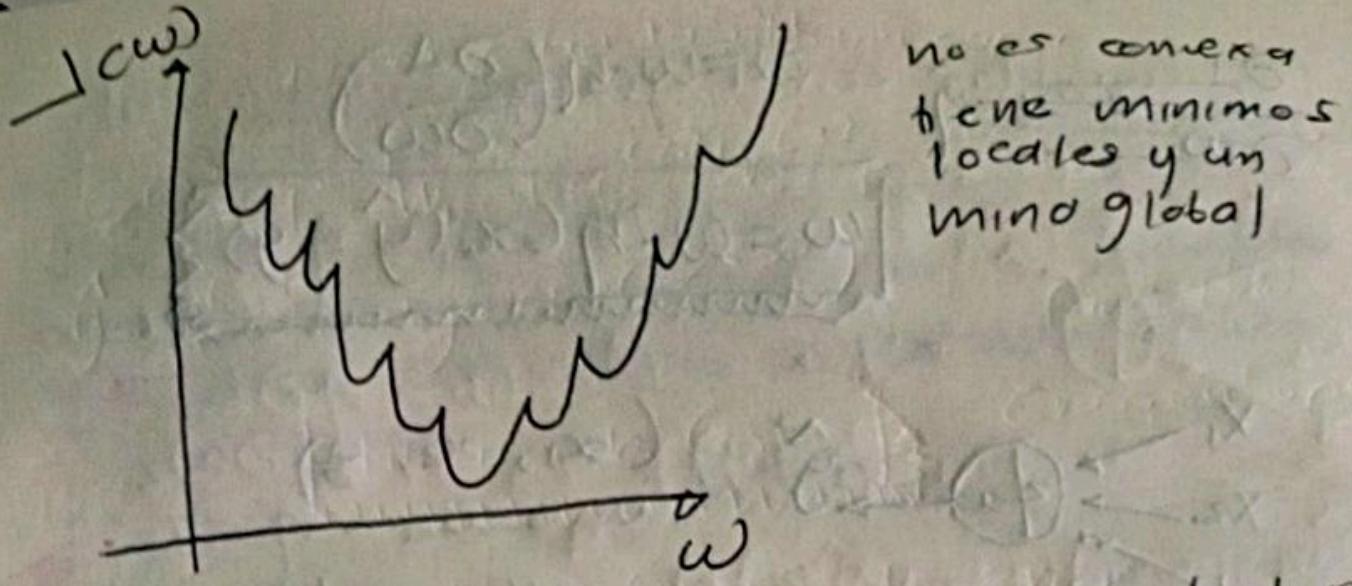
Función de optimización

Gradient Descent

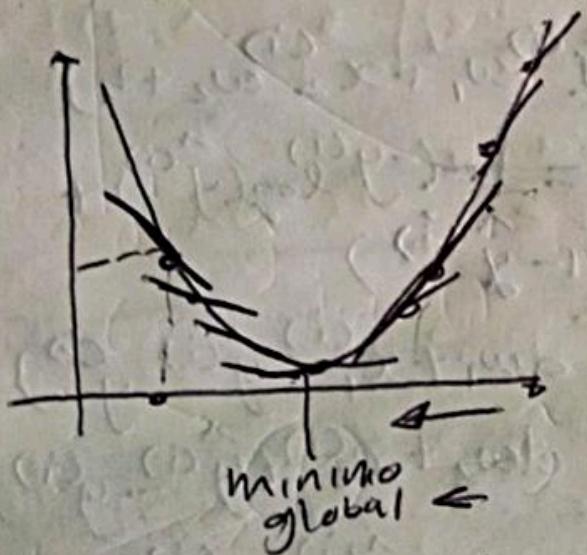
Función de corte: $J(W, B) = - \frac{1}{m} \sum_{i=1}^m y^{(i)} \ln(\hat{y}^{(i)}) + (1-y^{(i)}) \ln(1-\hat{y}^{(i)})$
 ↪ cross entropy loss



el objetivo de
Gradiente Descent
es encontrar ~~w~~ ^w
que minimiza el
valor de error



→ para el primer paso se da un valor aleatorio

$$\omega = \omega - \eta \left(\frac{\partial J}{\partial \omega} \right)$$


$$\omega = \omega - \eta \frac{\partial J}{\partial \omega}$$

$$\omega = \omega - \eta \text{ (número positivo)}$$

modificar los valores
parámetros del modelo
en contra de la pendiente

cuando se llegue al mínimo
global la pendiente es igual a cero

No varia
si varia

$$\frac{\partial J}{\partial \omega} = 0 \quad \text{y por lo tanto } \underline{\omega = \omega},$$

en el caso de pendiente negativa

$$↑ \omega = \omega - \eta \text{ (número negativo)}$$

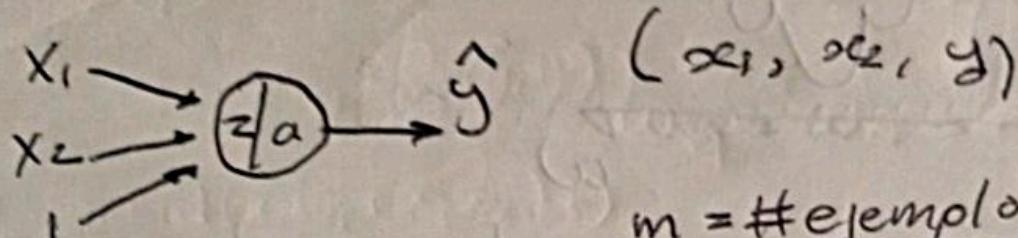
Creece
el algoritmo repite hasta que ω no varie

se encuentra
el
mínimo
global.

$$\frac{\partial L}{\partial \omega} = \omega - \eta$$

$$L = L - \eta \left(\frac{\partial L}{\partial \omega} \right)$$

$$L = L - \eta (x(\hat{y} - y))$$



$$J=0, d\omega_1=0, d\omega_2=0, db=0$$

for $i=1$ to m :

FORWARD PROPAGAT. $\hat{y}^{(i)} = \text{sigmoid}(x_1^{(i)}\omega_1 + x_2^{(i)}\omega_2 + b)$

$$J = J + \mathcal{L}(\hat{y}, y) = J + (-y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)}))$$

$$d\omega_1 = d\omega_1 + \frac{\partial J}{\partial \omega_1} = d\omega_1 + x_1^{(i)} (\hat{y}^{(i)} - y^{(i)})$$

$$d\omega_2 = d\omega_2 + \frac{\partial J}{\partial \omega_2} = d\omega_2 + x_2^{(i)} (\hat{y}^{(i)} - y^{(i)})$$

$$db = db + \frac{\partial J}{\partial b} = db + (\hat{y}^{(i)} - y^{(i)})$$

$$J = J/m \quad \text{calcular valor medio de } J$$

$$d\omega_1 = d\omega_1 / m$$

$$d\omega_2 = d\omega_2 / m$$

$$db = db / m$$

$$\omega_1 = \omega_1 - \eta d\omega_1$$

$$\omega_2 = \omega_2 - \eta d\omega_2$$

$$b = b - \eta db$$

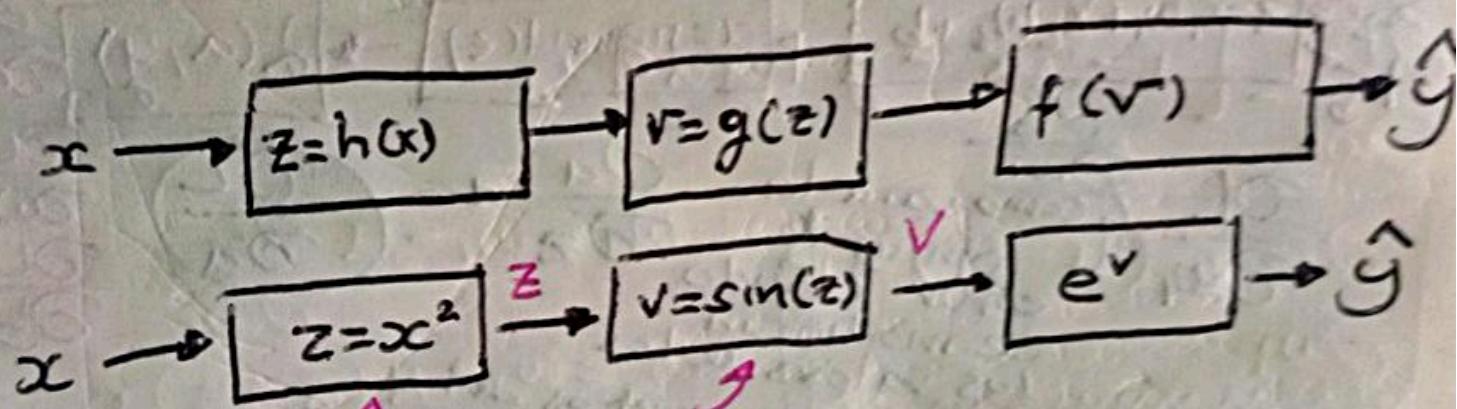
GRAFO COMPUTACIONAL

Son grafos dirigidos cuyos nodos se corresponden con una operación o variable y los arcos con valores de entrada/salida

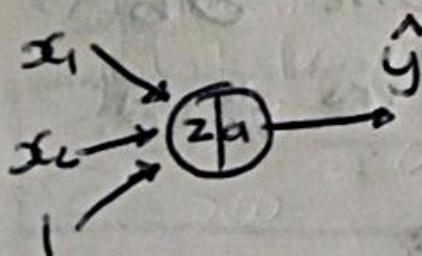
\nearrow compuesta

$$f(g(h(x))) \longrightarrow e^{\sin(x^2)}$$

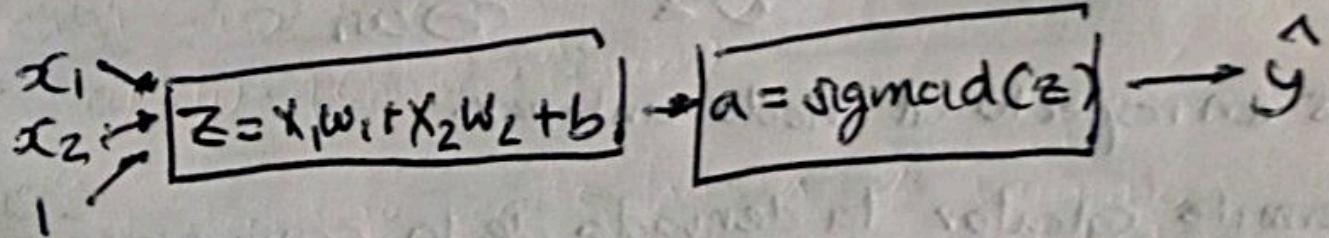
$$\begin{aligned} & \circ f(x) = e^x \\ & \circ g(x) = \sin(x) \\ & \circ h(x) = x^2 \end{aligned} \quad \left\{ \begin{aligned} & g(h(x)) = \sin(x^2) \\ & f(g(h(x))) = e^{\sin(x^2)} \end{aligned} \right.$$



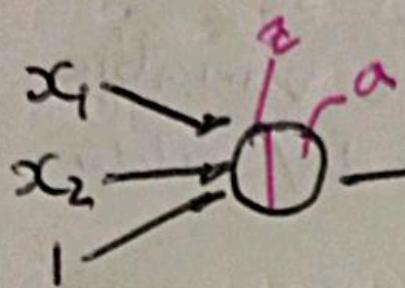
Aplicación del grafo computacional en una RNA sencilla



$$\hat{y} = \text{sigmoid}(x_1 w_1 + x_2 w_2 + b)$$



Dervadas con el gráfico computacional.
Gradient descent



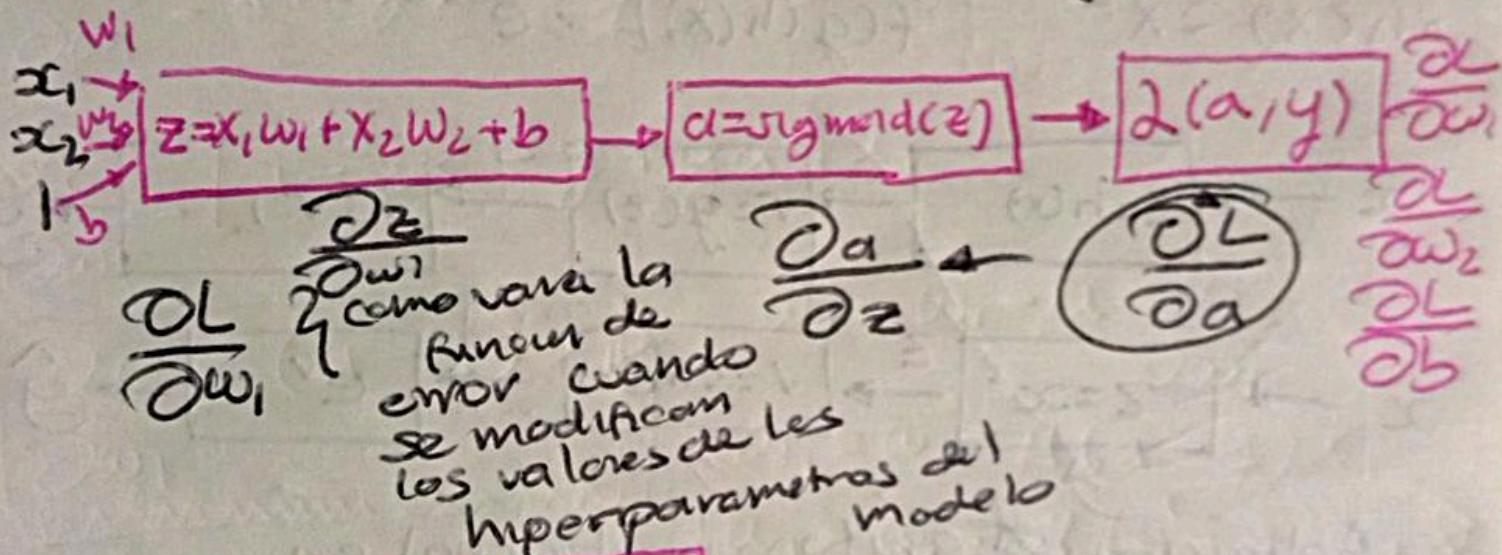
$$w_1 = w_1 - \eta \frac{\partial L}{\partial w_1}$$

$$w_2 = w_2 - \eta \frac{\partial L}{\partial w_2}$$

$$w_n = w_n = \eta \frac{\partial L}{\partial w_n}$$

$$\hat{y} = \text{sigmoid}(x_1 w_1 + x_2 w_2 + b)$$

$$b = b - \eta \frac{\partial L}{\partial b}$$



Regla de la cadena

$$\frac{d}{dw_1}, \frac{d}{dw_2}, \frac{d}{db}$$

primero se calcula con respecto al res de la función de activación.

$$\frac{\partial L}{\partial a} \rightarrow \frac{\partial a}{\partial z} \rightarrow \frac{\partial z}{\partial w_1}$$

es con respecto al nodo anterior.

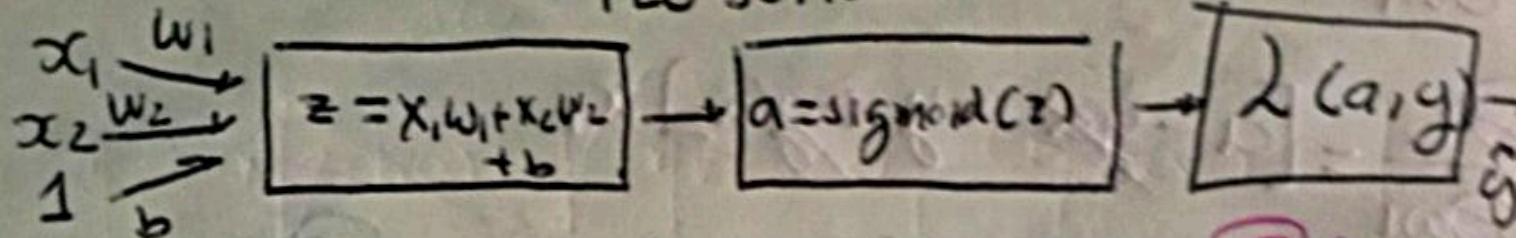
Permite calcular la derivada de la composición de dos o más funciones

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_2}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b}$$

Ahora otro ejemplo, supongamos que $\hat{y} = a$
TLU sencilla.



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

$$\frac{\partial L}{\partial a} = \frac{\partial}{\partial a} [-y \log(a) - (1-y) \log(1-a)]$$

$$\frac{\partial L}{\partial a} = \frac{1-y}{1-a} - \frac{y}{a}$$

cross entropy loss
sigmoid function

$$\frac{\partial a}{\partial z} = \frac{\partial}{\partial z} \left[\frac{1}{1+e^{-z}} \right] = a(1-a)$$

$$\frac{\partial a}{\partial w_1} = \frac{\partial}{\partial w_1} [\omega_1 x_1 + \omega_2 x_2 + b] = x_1$$

regla de la cadena

Así que

$$\frac{\partial L}{\partial w_1} = \left(\frac{1-y}{1-a} \right) \frac{y}{a} (a(1-a)) x_1$$

$$\frac{\partial L}{\partial w_1} = \frac{(1-y)a - y(1-a)}{(1-a)a} (a(1-a)) x_1$$

$$\frac{\partial L}{\partial w_1} = (1-y)a - y(1-a) x_1$$

$$\frac{\partial L}{\partial w_1} = [a - yx - y + ya] x_1$$

$$\frac{\partial L}{\partial w_1} = (a - y) x_1$$

learning
rate

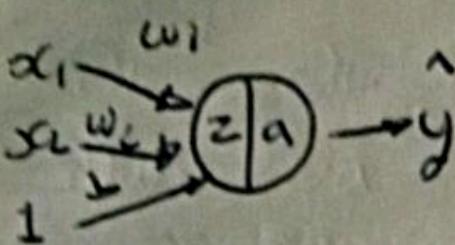
$$w_1 = w_1 - \mu \frac{\partial L}{\partial w_1}$$

$$w_1 = w_1 - \mu (a - y) x_1$$

$$w_1 = w_1 - \mu (\bar{y} - y) x_1$$

BackPropagation \rightarrow es todo esto realizado de obtener las derivadas.

BACK PROPAGATION

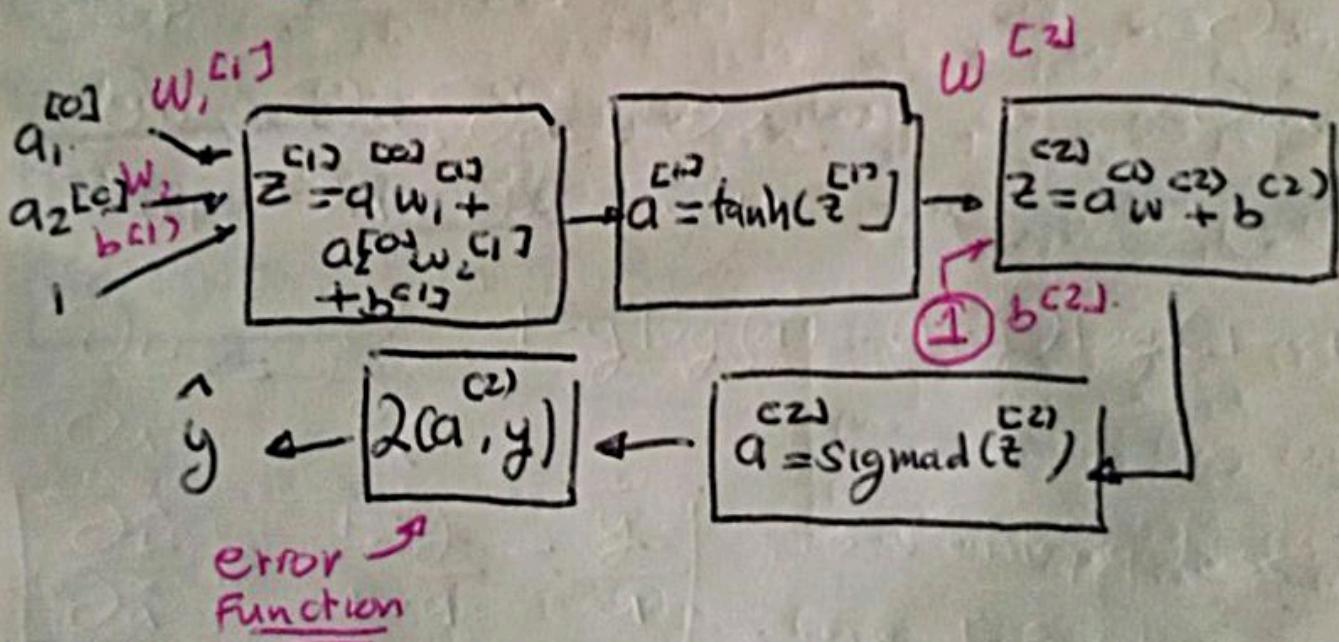
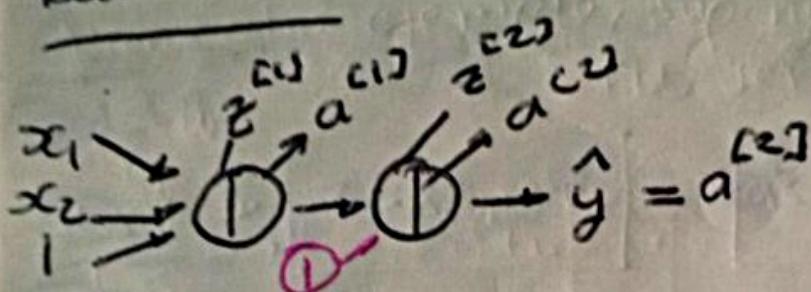


$$w_1 = w_1 - \eta \frac{\partial L}{\partial w_1} = w_1 - \eta (x_1(a-y))$$

$$w_2 = w_2 - \eta \frac{\partial L}{\partial w_2} = w_2 - \eta (x_2(a-y))$$

$$b = b - \eta \frac{\partial L}{\partial b} = w_1 - \eta (a-y) \quad (1)$$

Backward



$$\frac{\partial L}{\partial w_{1,c1}}, \frac{\partial L}{\partial w_{2,c2}}$$

$$\frac{\partial L}{\partial w_{1,c1}} = \frac{\partial L}{\partial a^{c2}} \cdot \frac{\partial a^{c2}}{\partial z^{c2}} \cdot \frac{\partial z^{c2}}{\partial a^{c1}} \cdot \frac{\partial a^{c1}}{\partial z^{c1}} \cdot \frac{\partial z^{c1}}{\partial w_{1,c1}}$$

$$\frac{\partial L}{\partial w_{2,c2}} = \frac{\partial L}{\partial a^{c2}} \cdot \frac{\partial a^{c2}}{\partial z^{c2}} \cdot \frac{\partial z^{c2}}{\partial w_{2,c2}}$$

backward propagation

$$\frac{\partial L}{\partial w^{[2]}} = a^{[2]}(a^{[2]} - y)$$

$$a^{[2]} - y$$

$$\frac{\partial L}{\partial b^{[2]}}$$

$$\frac{\partial L}{\partial w_1^{[1]}} = (a^{[2]} - y)w^{[2]}(1 - a^{[1]2}) \cdot x_1$$

$$\frac{\partial L}{\partial w_2^{[1]}} = [a^{[2]} - y]w^{[2]}(1 - a^{[1]2}) \cdot x_2$$

$$\frac{\partial L}{\partial b} = (a^{[2]} - y)w^{[2]}(1 - a^{[1]2})$$

Parameter update

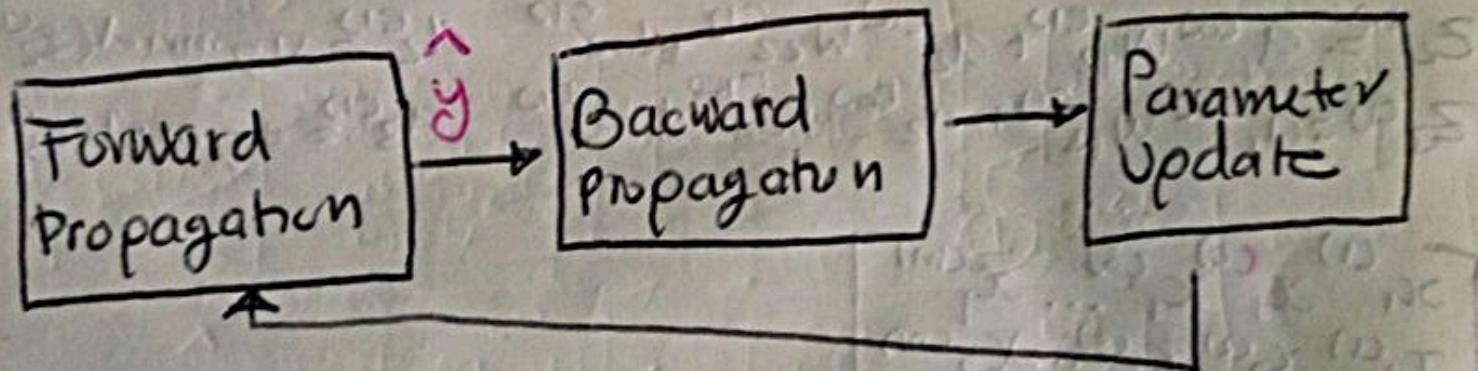
$$w^{[2]} = w^{[2]} - \eta \cdot \frac{\partial L}{\partial w^{[2]}}$$

$$b^{[2]} = b^{[2]} - \eta \cdot \frac{\partial L}{\partial b^{[2]}}$$

$$w_1^{[1]} = w_1^{[1]} - \eta \cdot \frac{\partial L}{\partial w_1^{[1]}}$$

$$w_2^{[1]} = w_2^{[1]} - \eta \cdot \frac{\partial L}{\partial w_2^{[1]}}$$

$$b^{[1]} = b^{[1]} - \eta \cdot \frac{\partial L}{\partial b^{[1]}}$$

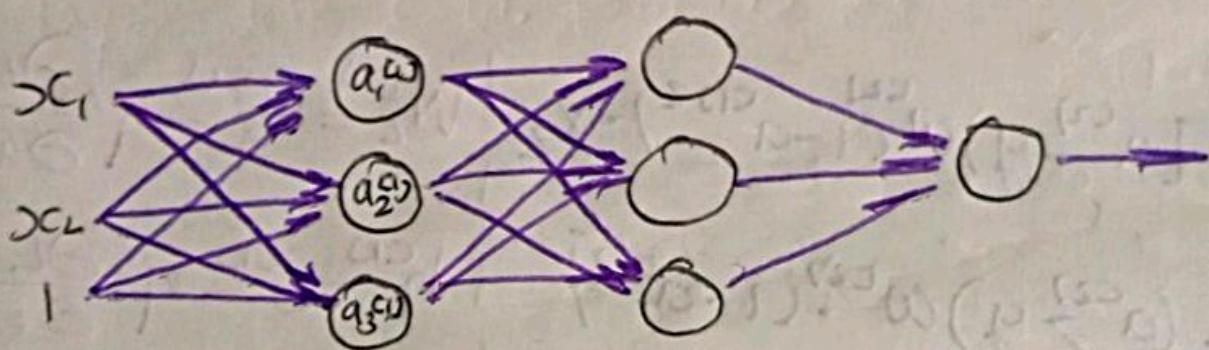


$$\begin{bmatrix} (w^{[2]11}, \dots, w^{[2]1d}, b^{[2]1}) \\ (w^{[2]21}, \dots, w^{[2]2d}, b^{[2]2}) \\ \vdots \\ (w^{[2]L1}, \dots, w^{[2]Ld}, b^{[2]L}) \end{bmatrix} = A$$

Implementación Vectorizada de RNNs

Forward Propagation: Vectorización

Input layer Hidden layer 1 Hidden layer 2 Output layer



$$z_1^{(C0)} = a_1^{(C0)} w_{11}^{(C1)} + a_2^{(C0)} w_{21}^{(C1)} + b_1^{(C1)}$$

$$z_2^{(C0)} = a_1^{(C0)} w_{12}^{(C1)} + a_2^{(C0)} w_{22}^{(C1)} + b_2^{(C1)}$$

$$z_3^{(C0)} = a_1^{(C0)} w_{13}^{(C1)} + a_2^{(C0)} w_{23}^{(C1)} + b_3^{(C1)}$$

$$a_1^{(C1)} = \text{sigmoid}(z_1^{(C1)})$$

$$a_2^{(C1)} = \text{sigmoid}(z_2^{(C1)})$$

$$a_3^{(C1)} = \text{sigmoid}(z_3^{(C1)})$$

$$\begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & \dots & x_2^{(m)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & x_n^{(3)} & \dots & x_n^{(m)} \end{bmatrix}$$

$$A^{(C0)} = \begin{bmatrix} a_1^{(C0)(1)} & a_1^{(C0)(2)} & \dots & a_1^{(C0)(m)} \\ a_2^{(C0)(1)} & a_2^{(C0)(2)} & \dots & a_2^{(C0)(m)} \\ \vdots & \vdots & \ddots & \vdots \\ a_n^{(C0)(1)} & a_n^{(C0)(2)} & \dots & a_n^{(C0)(m)} \end{bmatrix}$$

$$W^{(C1)} = \begin{bmatrix} w_{11}^{(C1)} & w_{12}^{(C1)} & w_{13}^{(C1)} \\ w_{21}^{(C1)} & w_{22}^{(C1)} & w_{23}^{(C1)} \\ \vdots & \vdots & \vdots \\ w_{n1}^{(C1)} & w_{n2}^{(C1)} & w_{n3}^{(C1)} \end{bmatrix}$$

$$b^{(C1)} = \begin{bmatrix} b_1^{(C1)} \\ b_2^{(C1)} \\ b_3^{(C1)} \end{bmatrix}$$

$$Z^{[0]} = W^{[0]T} A^{[0]} + b^{[0]}$$

$$Z^{[0]} = \begin{bmatrix} W_{11}^{[0]} & W_{21}^{[0]} & \dots & W_{n1}^{[0]} \\ W_{12}^{[0]} & W_{22}^{[0]} & \dots & W_{n2}^{[0]} \\ W_{13}^{[0]} & W_{23}^{[0]} & \dots & W_{n3}^{[0]} \end{bmatrix} \begin{bmatrix} a_1^{[0](1)} \\ a_2^{[0](1)} \\ \vdots \\ a_n^{[0](1)} \end{bmatrix} + \begin{bmatrix} a_1^{[0](m)} \\ a_2^{[0](m)} \\ \vdots \\ a_n^{[0](m)} \end{bmatrix}$$

$$\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[1]} a_1^{[0](1)} + w_{21}^{[1]} a_2^{[0](1)} + \dots + w_{n1}^{[1]} a_n^{[0](1)} \\ w_{12}^{[1]} a_1^{[0](1)} + w_{22}^{[1]} a_2^{[0](1)} + \dots + w_{n2}^{[1]} a_n^{[0](1)} \\ w_{13}^{[1]} a_1^{[0](1)} + w_{23}^{[1]} a_2^{[0](1)} + \dots + w_{n3}^{[1]} a_n^{[0](1)} \end{bmatrix}$$

$$+ \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix} = \begin{bmatrix} w_{11}^{[1]} a_1^{[0](1)} + w_{21}^{[1]} a_2^{[0](1)} + \dots + w_{n1}^{[1]} a_n^{[0](1)} + b_1^{[1]} \\ \vdots \\ \vdots \end{bmatrix}$$

$$= \begin{bmatrix} Z_1^{1} & Z_1^{[1](2)} & \dots & Z_1^{[1](m)} \\ Z_2^{1} & Z_2^{[1](2)} & \dots & Z_2^{[1](m)} \\ Z_3^{1} & Z_3^{[1](2)} & \dots & Z_3^{[1](m)} \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} a_1^{1} & a_1^{[1](2)} & \dots & a_1^{[1](m)} \\ a_2^{1} & a_2^{[1](2)} & \dots & a_2^{[1](m)} \\ a_3^{1} & a_3^{[1](2)} & \dots & a_3^{[1](m)} \end{bmatrix}$$

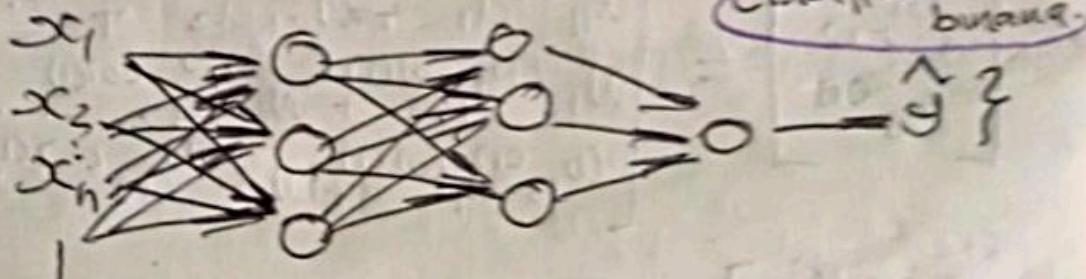
$$W^{[2]} = \begin{bmatrix} W_{11}^{[2]} & W_{12}^{[2]} & W_{13}^{[2]} \\ W_{21}^{[2]} & W_{22}^{[2]} & W_{23}^{[2]} \\ W_{31}^{[2]} & W_{32}^{[2]} & W_{33}^{[2]} \end{bmatrix}$$

$$b^{[2]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix} \quad Z^{[2]} = W^{[2]T} A^{[1]} + b^{[2]}$$

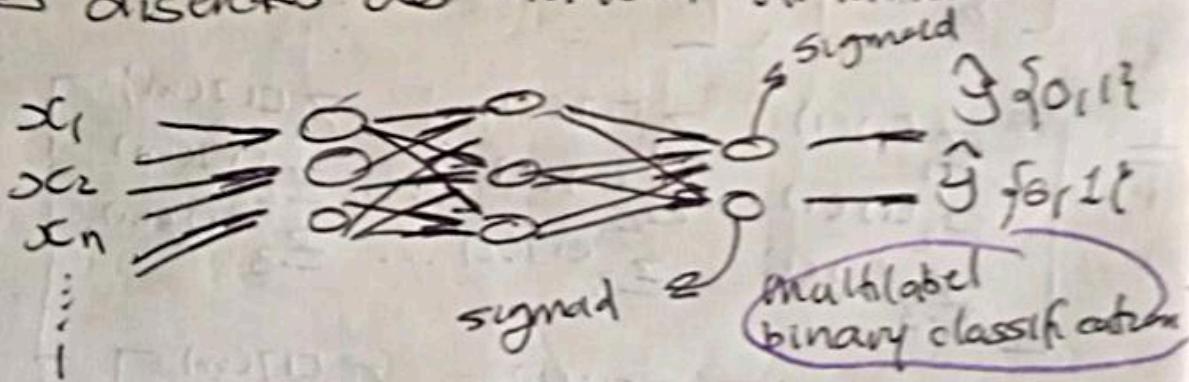
Perceptrón Multicapa: Clasificación

Las redes neuronales profundas pueden utilizarse para predecir un valor discreto. $\{0, 1, 2, 3, 4\}$

Para predecir un único valor discreto, la output layer debe estar formado por una única neurona



También pueden usarse RNNs para predecir varios valores discretos de manera simultánea-



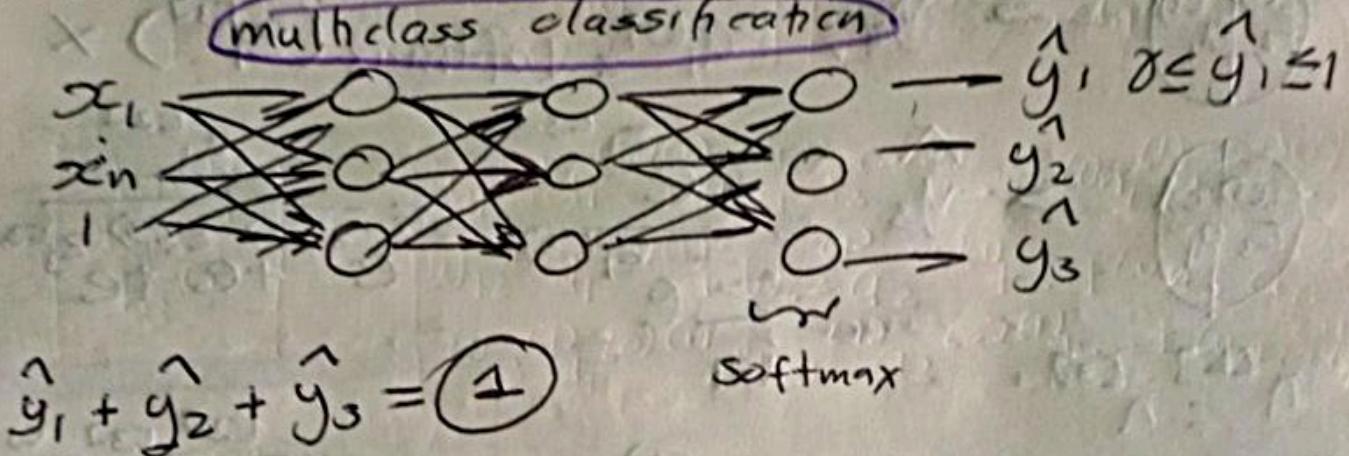
Clasificación binaria va a utilizar la función de activación sigmoid.

Clasificación binaria simultánea, se utilizarán dos neuronas en la output layer con función de activación sigmoid

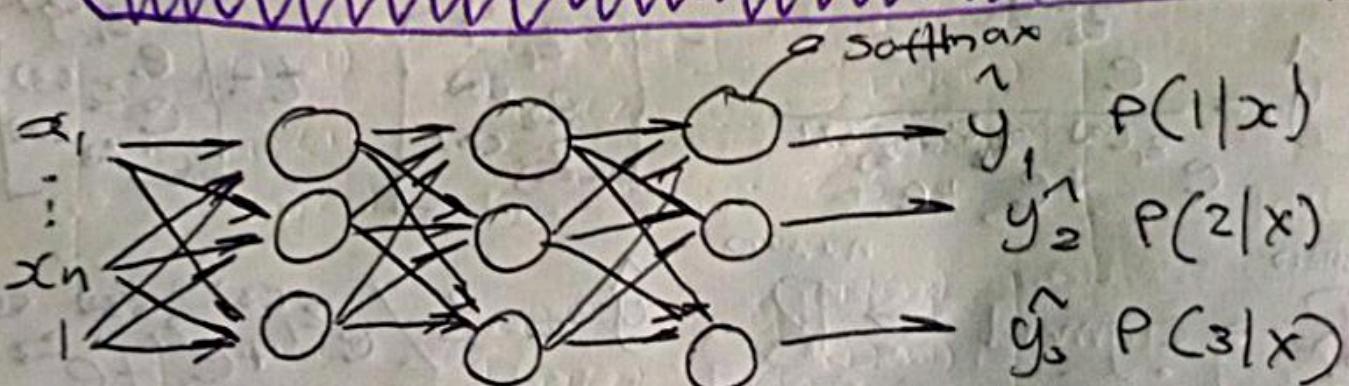
Pueden realizarse clasificación de más de dos clases (multiclas), en este caso se utilizarán tantas neuronas como clases se desean, decir y la función de activación softmax.

- En la clasificación multiclas la salida de las neuronas de output layer debe sumar 1.

multiclass classification



Multiclass classification: Softmax Regression

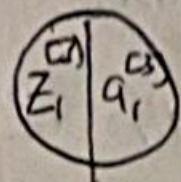


Carro \rightarrow class 1 $\rightarrow y=1$
 motocicleta $\rightarrow y=2$
 bicicleta $\rightarrow y=3$

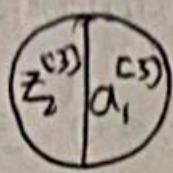
$$p(1|x) + p(2|x) + p(3|x) = \hat{y}_1 + \hat{y}_2 + \hat{y}_3 = 1$$

Softmax layer

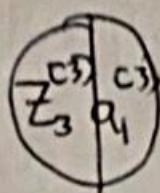
3 classes



$$\rightarrow \hat{y}_1 \rightarrow z_1^{(3)} = a_1 w_1^{(2)} + a_2 w_{21} + a_3 w_{31} + b_1^{(3)}$$



$$\rightarrow \hat{y}_2 \quad a_1^{(3)} = \text{softmax}(z_1^{(3)}) \times$$



$$\rightarrow \hat{y}_3 \quad = \frac{e^{z_1^{(3)}}}{\sum_{j=1}^3 e^{z_j^{(3)}}} = \frac{e^{z_1^{(3)}}}{e^{z_1^{(3)}} + e^{z_2^{(3)}} + e^{z_3^{(3)}}}$$

$$n^{[L]} = n^{[3]}$$

#neuronas capa 3

$$\Rightarrow a_1^{(3)} = \text{softmax}(z_1, z_2, z_3) \\ = \hat{y}_1$$

$$\hat{y}_2 = \frac{e^{z_2^{(3)}}}{e^{z_1^{(3)}} + e^{z_2^{(3)}} + e^{z_3^{(3)}}}$$

$$\begin{bmatrix} z_1^{(3)} \\ z_2^{(3)} \\ z_3^{(3)} \end{bmatrix}$$

$$t = \begin{bmatrix} e^{z_1^{(3)}} \\ e^{z_2^{(3)}} \\ e^{z_3^{(3)}} \end{bmatrix}$$

$\mathbb{R} \xrightarrow{\text{numero}} \text{sigmoid} \xrightarrow{\text{numero}} \mathbb{R}$

$$\hat{y} = t / e^{z_1^{(3)}} + e^{z_2^{(3)}} + e^{z_3^{(3)}}$$

Vector $\xrightarrow{\text{SOFTMAX}}$ vector $\xrightarrow{=}$ \mathbb{R}

$$\hat{y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix} = \begin{bmatrix} 0.87 \\ 0.1 \\ 0.1 \end{bmatrix}$$

Entrenamiento de un clasificador multiclas

$$a^{[L]} = \hat{y} = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix} \quad \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3$$

$L = \# \text{capas}$

$$\hat{y} = z = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\hat{y} = 1 = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

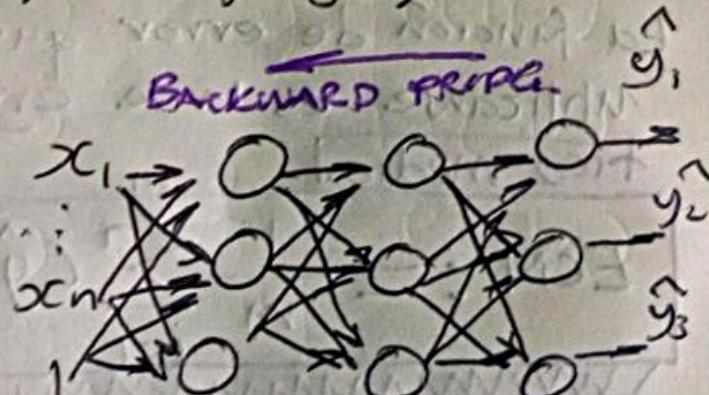
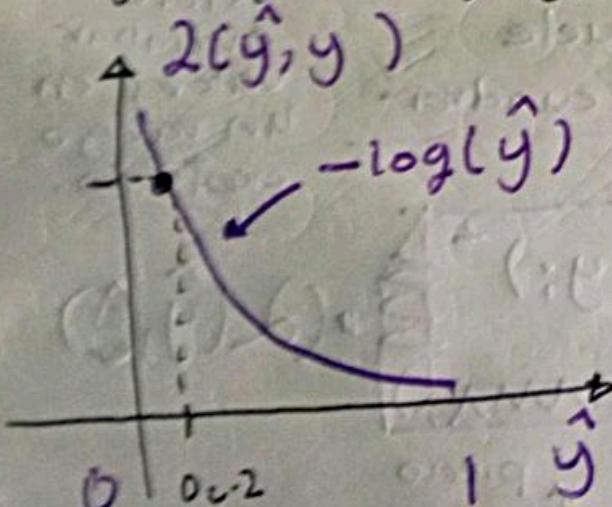
loss function $L(\hat{y}, y)$

$$L(\hat{y}, y) = - \sum_{j=1}^m y_j \log(\hat{y}_j) =$$

$$= -[(y_1 \log(\hat{y}_1) + y_2 \log(\hat{y}_2) + y_3 \log(\hat{y}_3))]$$

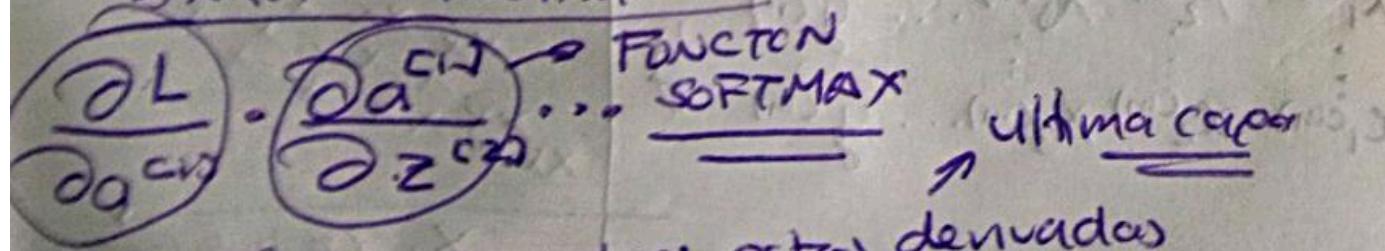
ejemplo : $y = z = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

$$L(\hat{y}, y) = -(0 + \log(\hat{y}_2) + 0) = -\log(\hat{y}_2)$$



FORWARD PROPAGATION

BACKWARD PROPAGATION



solo va a cambiar estas derivadas

Perceptron Multicapa: Regresión

Las redes neuronales profundas pueden utilizarse para predecir un valor continuo.

Para predecir un único valor continuo la output layer debe estar formado por una única neurona.

También las RNNs pueden predecir varios valores continuos de manera simultánea.

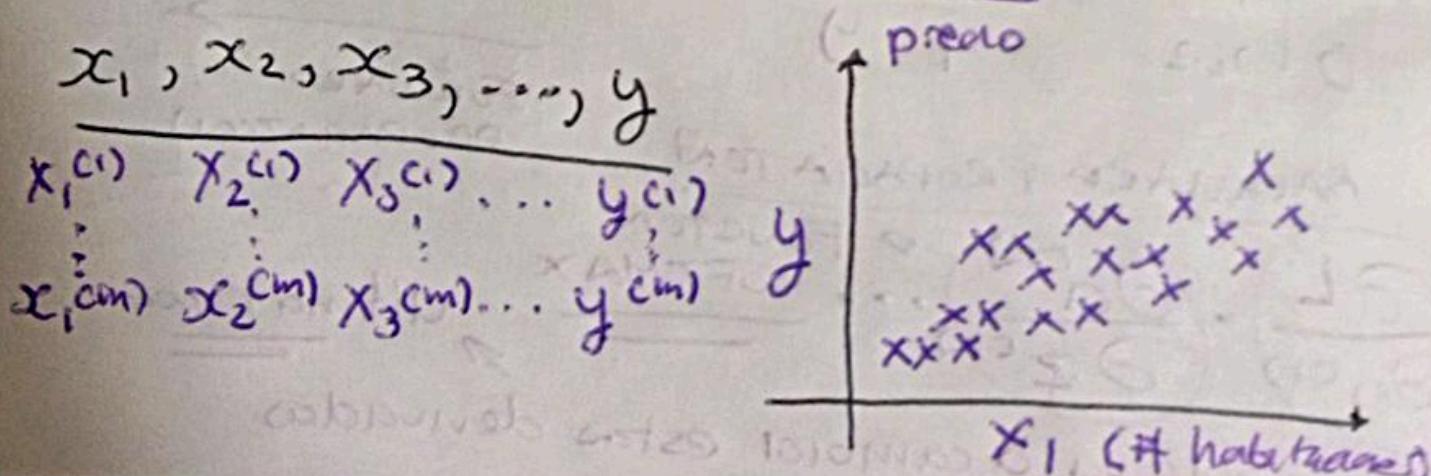
Para predecir varios valores continuos se requiere una neurona por cada valor en la output layer.

- Las RNNs utilizadas para regresión, en general no van a utilizar una función de activación en las neuronas en la output layer. $\leftarrow \text{ReLU}$

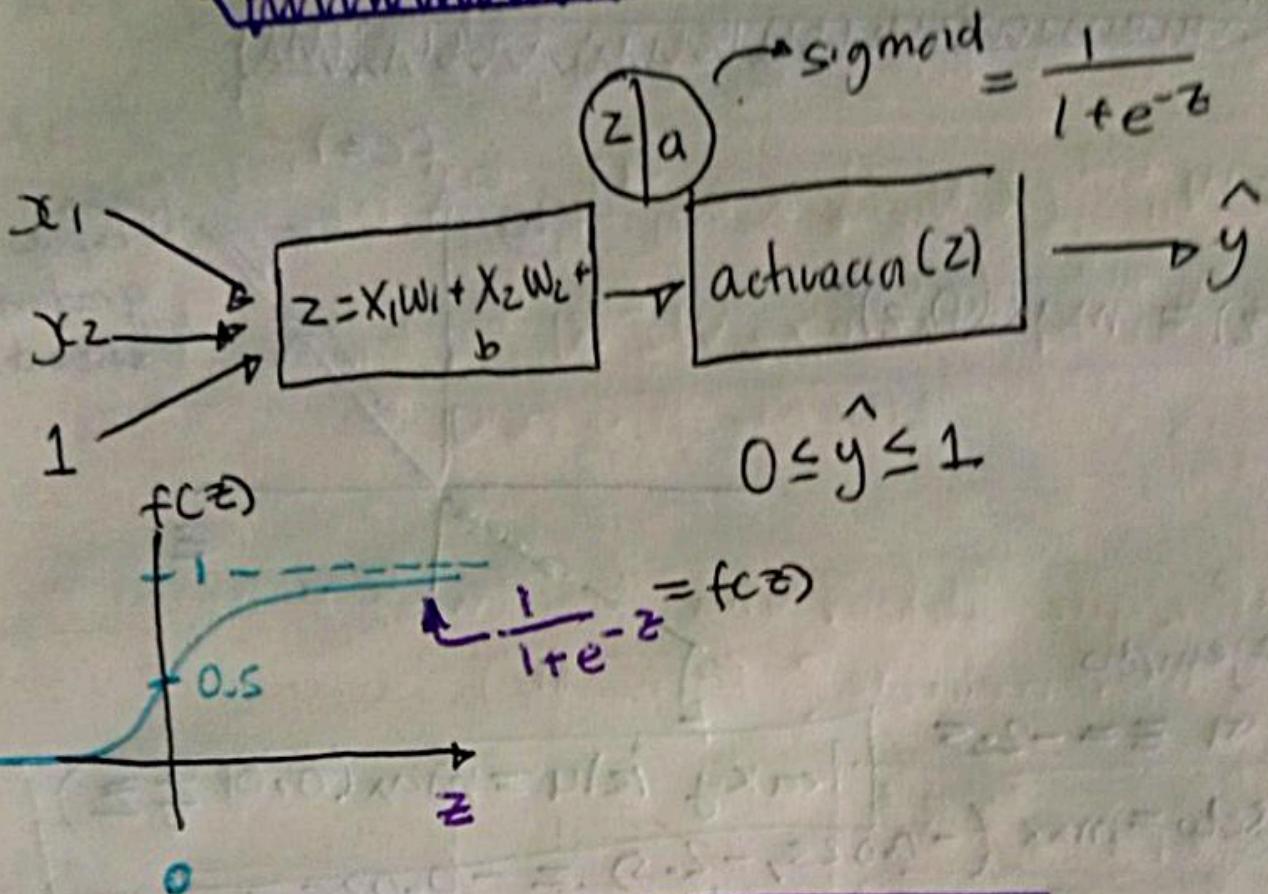
La función de error que suele utilizarse es la de error cuadrático medio.

si ve quiere encontrar valores en un rango acotado.

$$\text{ECM} := \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2 \rightarrow (2(\hat{y}, y))$$



INTRODUCCION FUNCIONES DE ACTIVACION

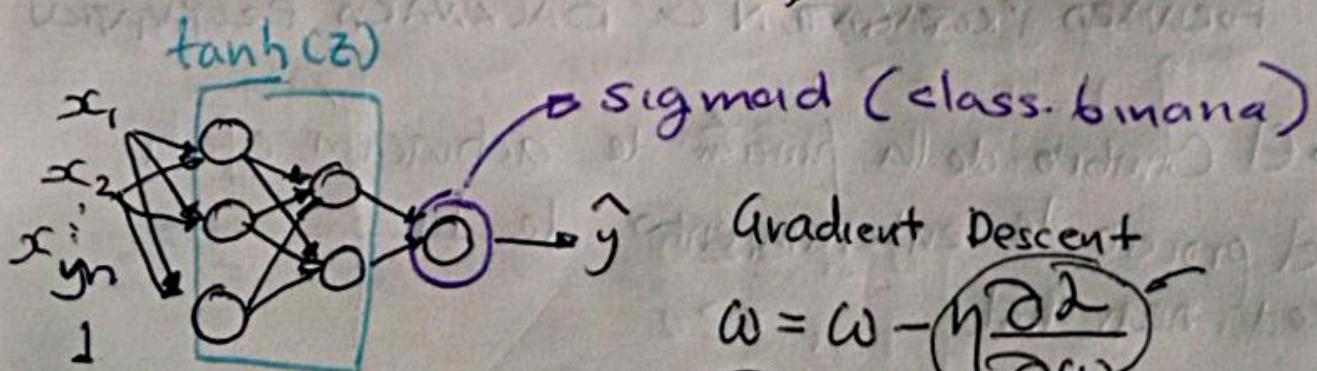
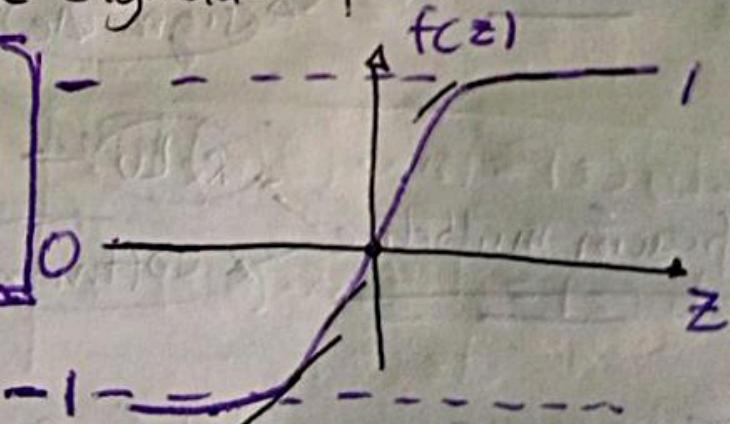


FUNCION ACTIVACION: $\tanh(z)$

funciona mejor que sigmoides para hidden layer

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

$$-1 \leq \tanh(z) \leq 1$$

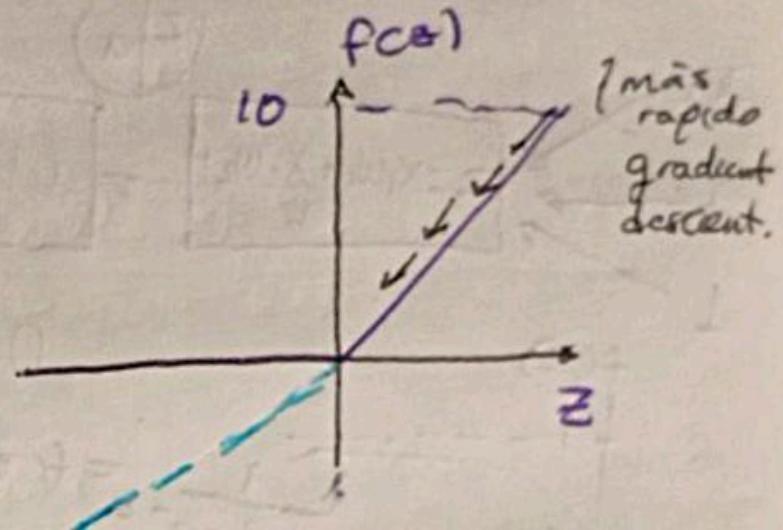


Si z es muy grande
o muy pequeño

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial w} \cdot \frac{\partial a}{\partial z} \dots$$

FUNCIÓN DE ACTIVACIÓN: ReLU(z)

$$\text{ReLU}(z) = \max(0, z)$$



Ejemplo

$$\text{si } z = -2.5$$

$$\text{leaky ReLU} = \max(0.01z, z)$$

$$\text{leaky ReLU} = \max(-0.025, -2.5) = -0.025$$

Clasificación binaria

ReLU → hidden layer

sigmoide → output layer

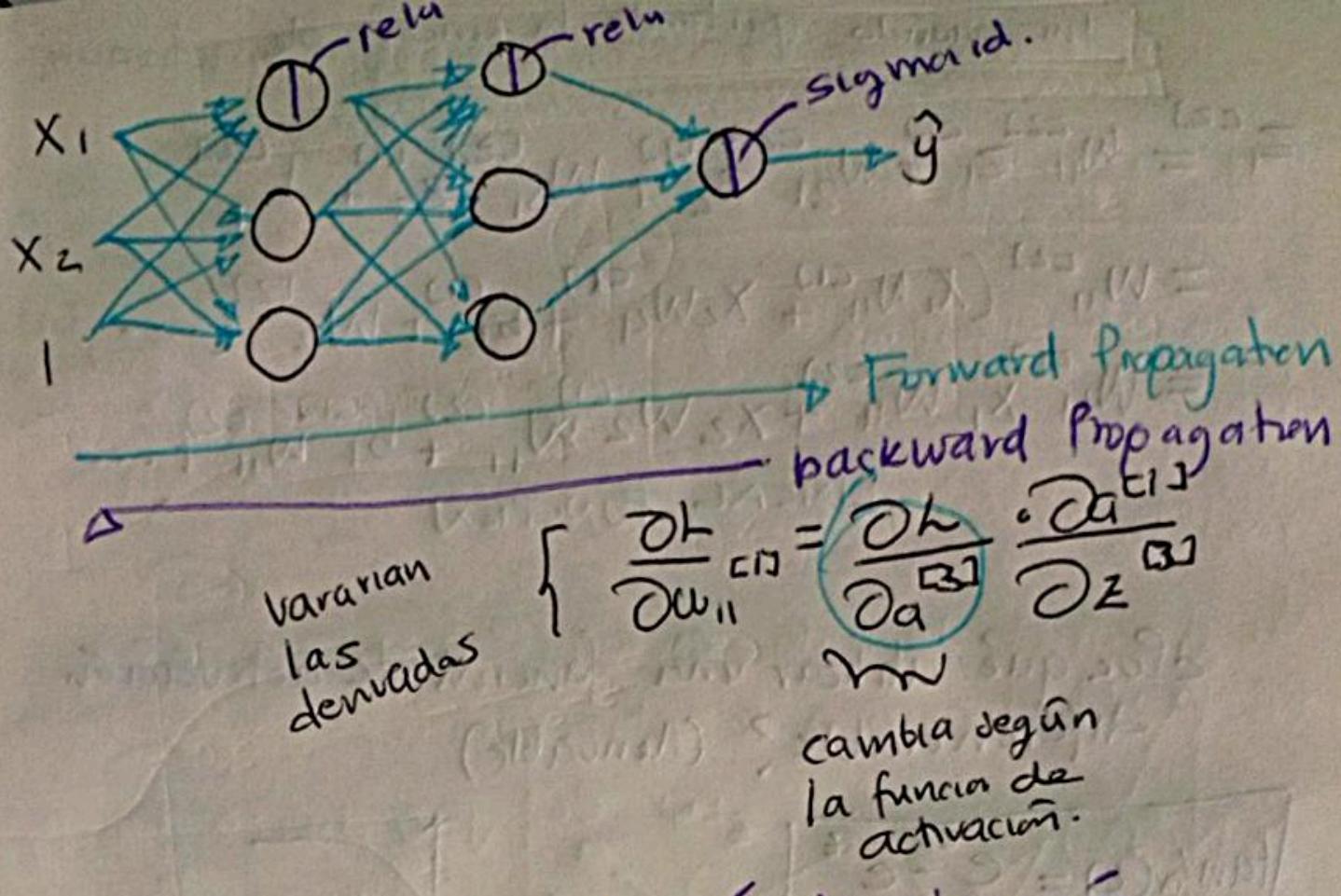
Clasificación multiclase

ReLU → hidden layer

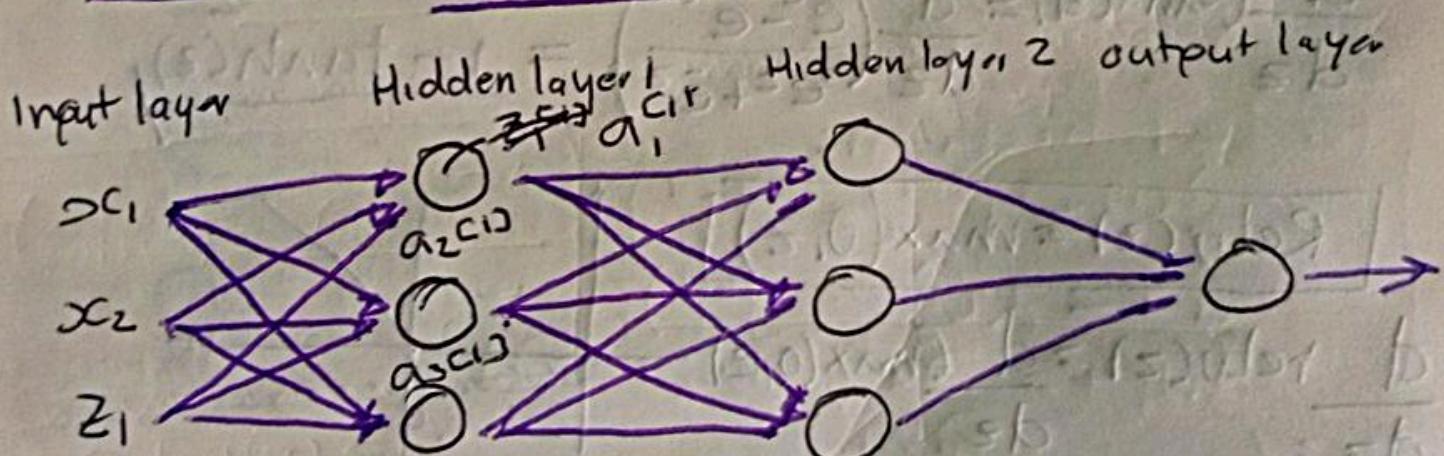
softmax → output layer

FORWARD PROPAGATION & BACKWARD PROPAGATION

- El cambio de la función de activación afecta el proceso de entrenamiento de la red neuronal.



Por qué utilizar una función de activación no lineal?



$$z_1^{(1)} = x_1 w_{11}^{(1)} + x_2 w_{21}^{(1)} + b_1^{(1)} \quad a_1^{(1)} = \text{sigmoid}(z_1^{(1)})$$

$$z_2^{(1)} = x_1 w_{12}^{(1)} + x_2 w_{22}^{(1)} + b_2^{(1)} \quad a_2^{(1)} = \text{sigmoid}(z_2^{(1)})$$

$$z_3^{(1)} = x_1 w_{13}^{(1)} + x_2 w_{23}^{(1)} + b_3^{(1)} \quad a_3^{(1)} = \text{sigmoid}(z_3^{(1)})$$

Si la función de activación es lineal, se obtendrá un límite de decisión lineal (PROBLEMA para problemas que NO SON Linealmente separables)

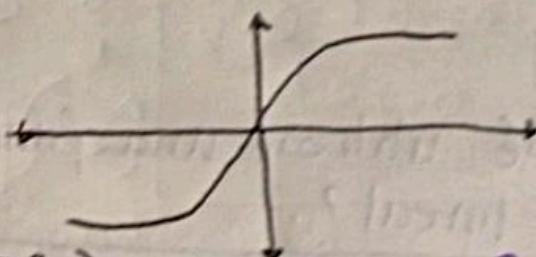
Imaginando quitando la función de activación.

$$\begin{aligned}z_1^{[2]} &= w_{11}^{[2]} z_1^{[1]} + w_{21}^{[2]} z_2^{[1]} + w_{31}^{[2]} z_3^{[1]} + b_1^{[2]} \\&= w_{11}^{[2]} (x_1 w_{11}^{[1]} + x_2 w_{21}^{[1]} + b_1^{[1]}) + w_{21}^{[2]} (\dots) \\&= \underline{w_{11}^{[1]} x_1 w_{11}^{[2]} + x_2 w_{21}^{[1]} w_{21}^{[2]} + b_1^{[1]} w_{11}^{[2]} + \dots}\end{aligned}$$

LINEAL FUNCION

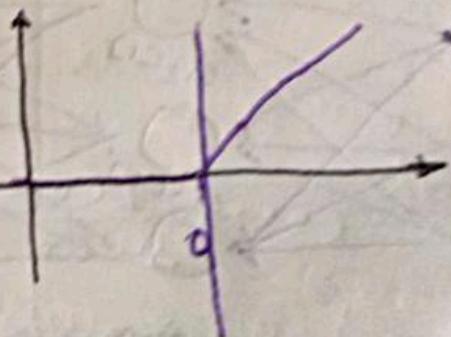
¿Por qué utilizar una función de activación diferenciable? (derivable)

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



$$\frac{d}{dz}(\tanh(z)) = \frac{d}{dz}\left(\frac{e^z - e^{-z}}{e^z + e^{-z}}\right) = \frac{1 - \tanh^2(z)}{1 + \tanh^2(z)}$$

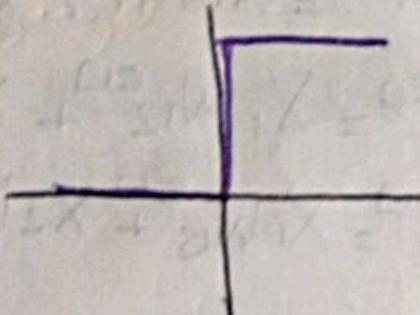
$$\text{ReLU}(z) = \max(0, z)$$



$$\frac{d}{dz} \text{relu}(z) = \frac{d}{dz}(\max(0, z))$$

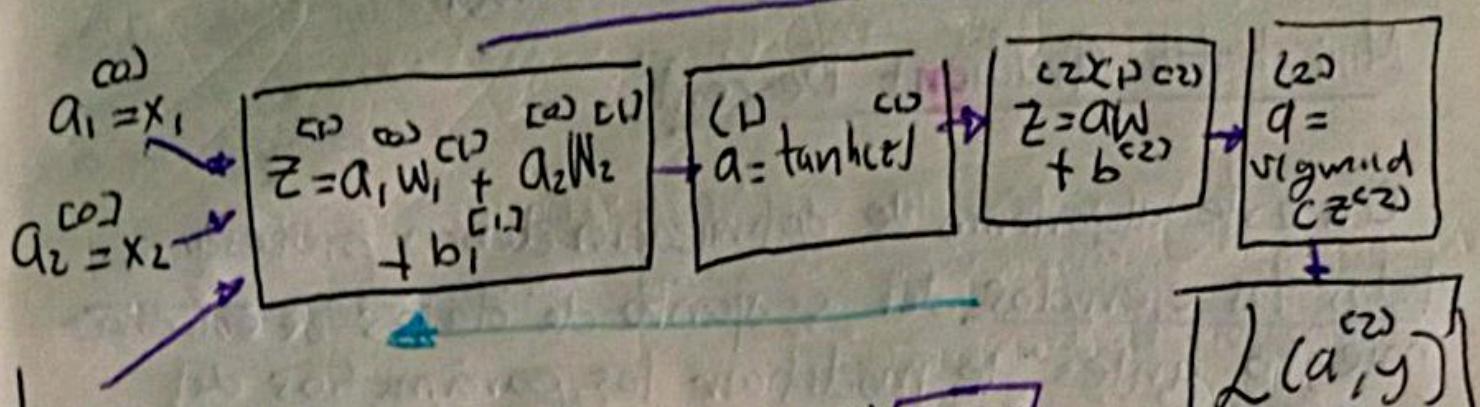
$$\begin{cases} 0 & \text{si } z < 0 \\ 1 & \text{si } z > 0 \\ \text{undefined} & \text{si } z = 0 \end{cases}$$

$$\text{heaviside}(z) = \begin{cases} 0 & \text{si } z < 0 \\ 1 & \text{si } z \geq 0 \end{cases}$$



$$\frac{d}{dz} \text{heaviside}(z) = \begin{cases} 0 & \text{si } z > 0 \\ 1 & \text{si } z < 0 \end{cases}$$

BACKWARD PROPAGATION



$$\frac{\partial L}{\partial w_1^{(0)}} = \frac{\partial L}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial a^{(1)}} \cdot \frac{\partial a^{(1)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial w_1^{(0)}}$$

$$\frac{\partial L}{\partial w_2^{(0)}} = \frac{\partial L}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial a^{(1)}} \cdot \frac{\partial a^{(1)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial w_2^{(0)}}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a^{(2)}} \cdot \frac{da^{(2)}}{dz^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial a^{(1)}} \cdot \frac{\partial a^{(1)}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial b}$$

$$\frac{\partial L}{\partial w_2^{(2)}} = \frac{\partial L}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w_2^{(2)}}$$

$$\frac{\partial L}{\partial b^{(2)}} = \frac{\partial L}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial b^{(2)}}$$

Imaginando que $a = \text{heaviside}(z)$

$$\frac{\partial a^{(1)}}{\partial z^{(1)}} = 0 \text{ entonces } \frac{\partial L}{\partial w_1^{(0)}} = 0$$

$$W_1^{(0)} = W_1^{(0)} - \eta \frac{\partial L}{\partial w_1^{(0)}} = W_1^{(0)} = 0$$

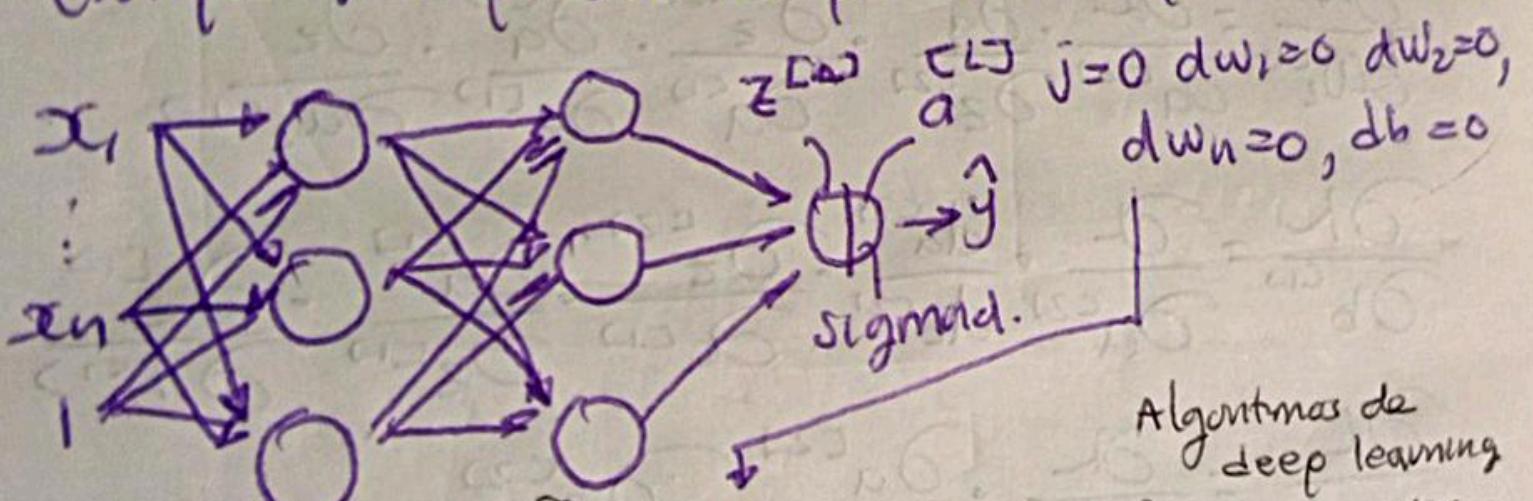
No se podrán actualizar los parámetros si se tiene una función no derivable

FUNCIÓNES DE OPTIMIZACIÓN

Mini-batch Gradient Descent.

Con este algoritmo de optimización deben procesarse todos los ejemplos del conjunto de datos de entrenamiento antes de modificar los parámetros del modelo.

Si el conjunto de datos es muy grande (ej. 2000000 ejemplos) el proceso de optimización puede ser lento.



$z^{[L]} \quad j=0 \quad dw_1=0 \quad dw_2=0,$
 $dwn=0, db=0$

Algoritmos de
deep learning
funcionan mejor
necesitan conjunto de
datos muy grandes

vectorización proceso.
for $i = 1$ hasta m :

$$\hat{y}^{(i)} = a^{[L]}(z^{[L]})^{[i]}$$

$$j \neq \lambda(\hat{y}^{(i)}, y^{(i)})$$

$$dw_2 += \frac{\partial J}{\partial w_2} \rightarrow \dots, db += \frac{\partial J}{\partial b}$$

$$dw_1 += \frac{\partial J}{\partial w_1}, dw_1 = dw_1/m, \dots, dw_n = dwn/m, db = \frac{db}{m}$$

$$j = j/m, dw_1 = dw_1/m, \dots, w_2 = w_2 - \eta dw_2, \dots, \theta$$

$$w_1 = w_1 - \eta dw_1$$

Una de las variaciones de Batch Gradient Descent consiste en dividir el conjunto de datos de entrenamiento en subconjuntos más pequeños llamados mini-batches

$$x_1, x_2, \dots, x_n \\ x_1^{(1)}, x_2^{(1)}, \dots, x_n^{(1)} \Rightarrow X = \begin{bmatrix} x_1^{(1)}, x_2^{(2)}, \dots, x_n^{(m)} \\ x_1^{(2)}, x_2^{(2)}, \dots, x_n^{(m)} \\ \vdots \\ x_1^{(m)}, x_2^{(m)}, \dots, x_n^{(m)} \end{bmatrix}$$

$$= \begin{bmatrix} X^{(1)} & X^{(2)} & \dots & X^{(m)} \end{bmatrix}$$

η = learning rate
 J = función de coste

$$Y = [y^{(1)} \ y^{(2)} \ y^{(3)} \ \dots \ y^{(m)}]$$

$$X^{fit} \xrightarrow{\text{mini batch}} \begin{bmatrix} x_1^{(1)}, x_2^{(2)}, \dots, x_n^{(1000)} \end{bmatrix} \xrightarrow{\text{subdividir en mini batches, ejemplo = 1000}} \text{mini batch}$$

$$X^{f2t} = \begin{bmatrix} x^{(1001)}, x^{(1002)}, \dots, x^{(2000)} \end{bmatrix} \text{mini batch}$$

$$X^{fit}$$

$$t = \# \text{mini batches}$$

$$y^{fit} = [y^{(1)}, y^{(2)}, \dots, y^{(1000)}]$$

$$y^{f2t} = [y^{(1001)}, y^{(1002)}, \dots, y^{(2000)}]$$

$$y^{fit}$$

Pseudo código I

mini batch
Gradient descent

for $j=1$ hasta t :

for $i=1$ hasta m :

$$g^{(i)} = q^{(L)(i)}, g^{(L)(i)}$$

$$j+ = \lambda(y^{(i)}, y^{(i)})$$

$m^{fit} = \# \text{ejemplos}$
del mini batch

1000 ejemplos

back propagation

$$\frac{dW_{1t}}{dw_i} = \frac{\partial L}{\partial w_i}, \frac{dW_{2t}}{dw_i} = \frac{\partial L}{\partial w_i}, \dots, \frac{\partial b}{db} = \frac{\partial L}{\partial b}$$

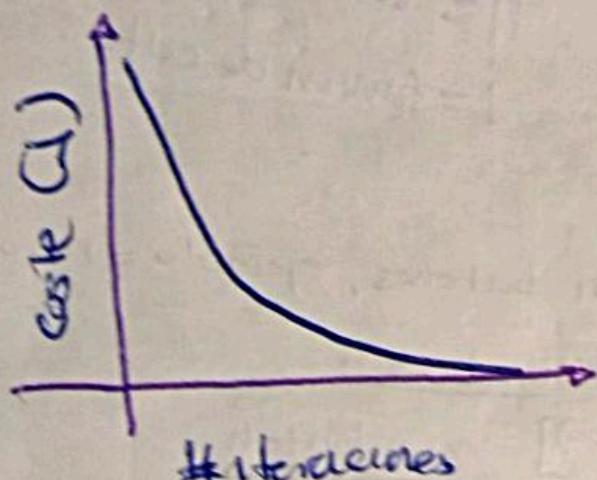
$$J = j/m, \frac{dw_i}{dw_i} = dw_i/m, \dots, \frac{db}{db} = db/m$$

$$w_i = w_i - \eta dw_i, dW_{nt} = W_n - \eta dW_{nt}, db = \eta db$$

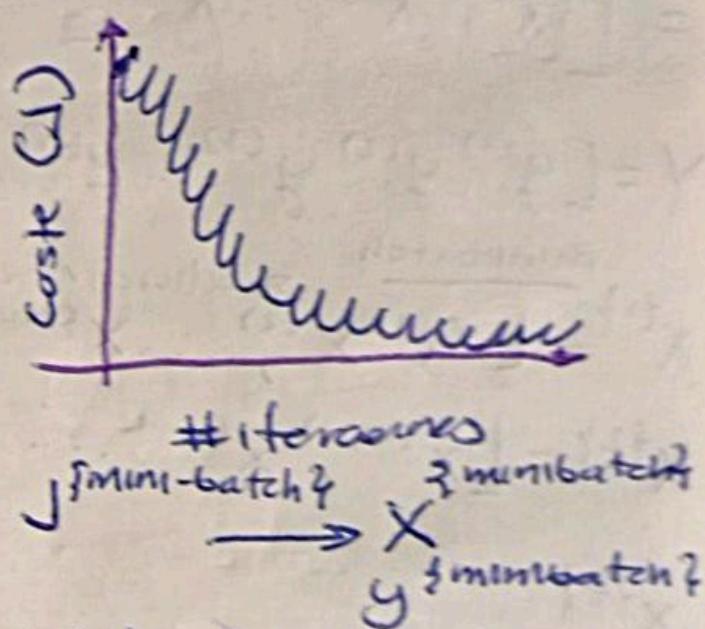
Se recorre el primer minibatch, luego actualizar los parámetros del modelo, no esperamos hasta que se completen todos los minibatches

STOCHASTIC GRADIENT DESCENT

Batch Gradient Descent



Mini Batch Gradient Descent



es normal que en minibatch puede aumentar un poco el error en comparación con batch gradient descent

Tamaño del minibatch

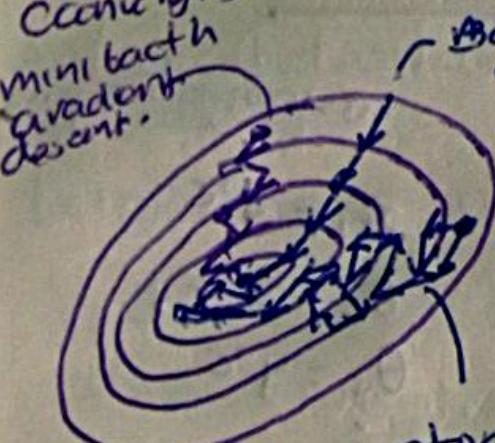
$m = \# \text{ejemplos del conjunto de entrenamiento}$

→ tamaño del minibatch = $m \rightarrow$ Batch Gradient Descent.

→ tamaño del minibatch = $1 < x < m \rightarrow$ mini-batch Gradient Descent.

→ tamaño del minibatch = $1 \rightarrow$ Stochastic Gradient Descent

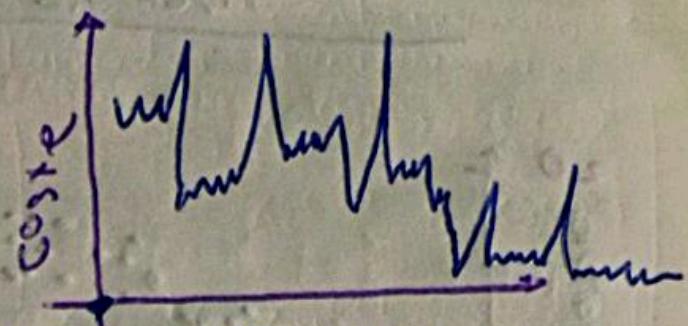
Substochastic Gradient descent
 Converge
 mini batch
 gradient
 descent.



Ideal para
 conjuntos de
 datos
 pequeños

Batch
 gradient
 descent
 (Converge)

stochastic
 gradient
 descent (nunca
 converge)



Ideal
 conjunto
 de datos grandes

TAMAÑO RECOMENDADO
 $\{64, 128, 256, 512\}$

BATCH GRADIENT DESCENT

Procesa el conjunto de datos completo en cada iteración.

Función de optimización mucho mas lenta.

Suele converger y alcanzar el mínimo de la función de error.

MINI BATCH GRADIENT DESC.

Procesa un subconjunto de datos en cada iteración.

Función de optimización más rápida que BGD.

Suele converger y alcanzar el mínimo de la función de error, pero de manera menos directa.

STOCHASTIC

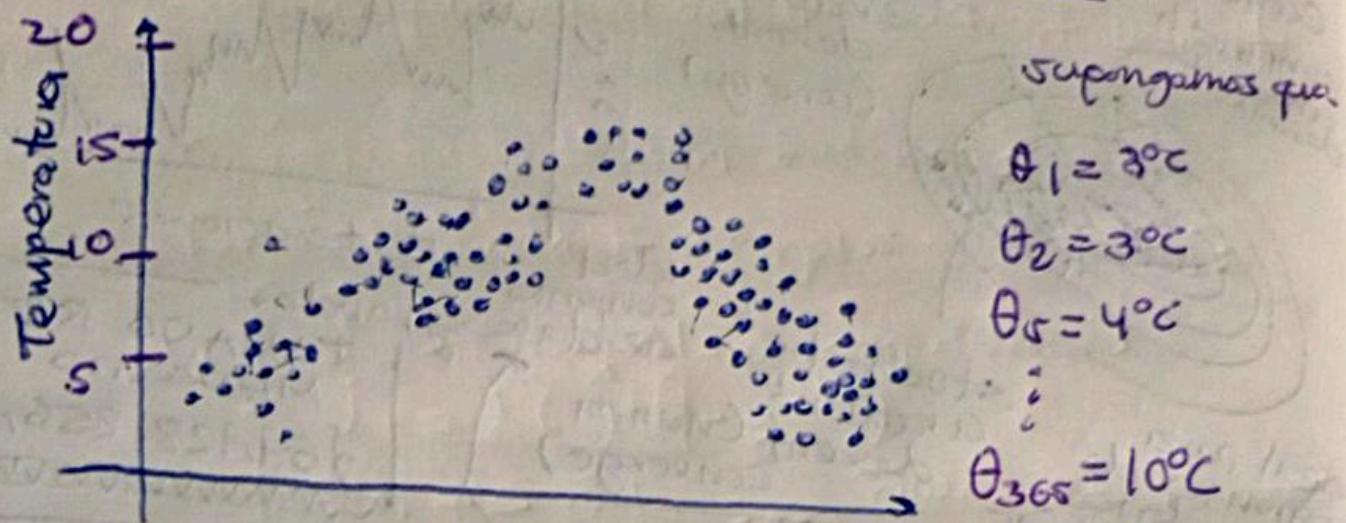
Procesa un ejemplo del conjunto de datos en cada iteración.

Función de optimización más rápida de las tres

No suele converger y alcanzar el mínimo de la función de error, aunque alcanza un valor aceptable

Minibatch GD combina lo mejor de ambos mundos: estabilidad del Batch y rapidez de stochastic, por eso es el más utilizado en entrenamiento de RNNs.

EXPONENTIALLY WEIGHTED MOVING AVERAGE



Así que, se tiene que recalcular los datos.

$$V_t = \beta V_{t-1} + (1-\beta) \theta_t$$

$$V_0 = 0$$

índice para elegir

Supongamos que $\theta_1 = 30^\circ C$

$\beta = 0.9$ $0 < \beta < 1$

$$V_1 = 0.9(0) + (1-0.9)(3) = \underline{0.3^\circ C}$$

Así con esto se va recalcando todas las exemplares del dataset.

La influencia de los puntos anteriores es pioneros.

$V_t \Rightarrow$ temperatura media aproximada de los

últimos $\frac{1}{1-\beta}$ días $\uparrow \beta$ aumenta más días de influencia

mas cercano a

1 más suave cercano a 0 reactivo

EJEMPLO

$$V_{100} = ? \quad \beta = 0.9$$

$$V_{100} = 0.9V_{99} + 0.1\theta_{100} = 0.1\theta_{100} + 0.9V_{99}$$

$$V_{100} = 0.1\theta_{100} + 0.9(0.9V_{98} + 0.1\theta_{99}) =$$

$$= 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9(0.1\theta_{98} + 0.9V_{97}))$$

Observando tiene en cuenta todas las temperaturas anteriores, así que:

$$\begin{aligned} V_{100} &= 0.1\theta_{100} + 0.9V_{99} = 0.1\theta_{100} + 0.09\theta_{99} + 0.81V_{97} \\ &= 0.1\theta_{100} + 0.09\theta_{99} + 0.081\theta_{98} + 0.729V_{97} \end{aligned}$$

$$\frac{1}{1-\beta} = \frac{1}{1-0.9} = 10 \text{ días} \quad \left. \begin{array}{l} \text{a partir de } V_{99} \\ \theta \approx 0 \text{ y número pequeño} \end{array} \right\} \therefore \text{NO TENDRA INFLUENCIA}$$

BIAS CORRECTION

$$\frac{V_t}{1-\beta^t} = \beta V_{t-1} + (1-\beta)\theta_t$$

MOMENTUM GRADIENT DESCENT

Más rápido que mini batch gradient descent

for $j=1$ hasta # mini batches:

for $i=1$ hasta # ejemplos minibatch j :

vectorizada $\left\{ \begin{array}{l} \hat{y}^{(i)} \rightarrow \text{Forward-propagation } (i) \\ j+ = \lambda (\hat{y}^{(i)} - y^{(i)}) \end{array} \right.$

$$\partial w_1 + = \frac{\partial J}{\partial w_1}, \dots, \partial w_n = \frac{\partial J}{\partial w_n}, \partial b = \frac{\partial J}{\partial b}$$

$$J = J/m^{\text{ejos}}, \underline{\partial w_1} = \partial w_1/m^{\text{ejos}}, \dots \underline{dw} = \frac{dw}{m^{\text{ejos}}}, \underline{db} = \frac{db}{m^{\text{ejos}}}$$

aplicar exponential weighted moving

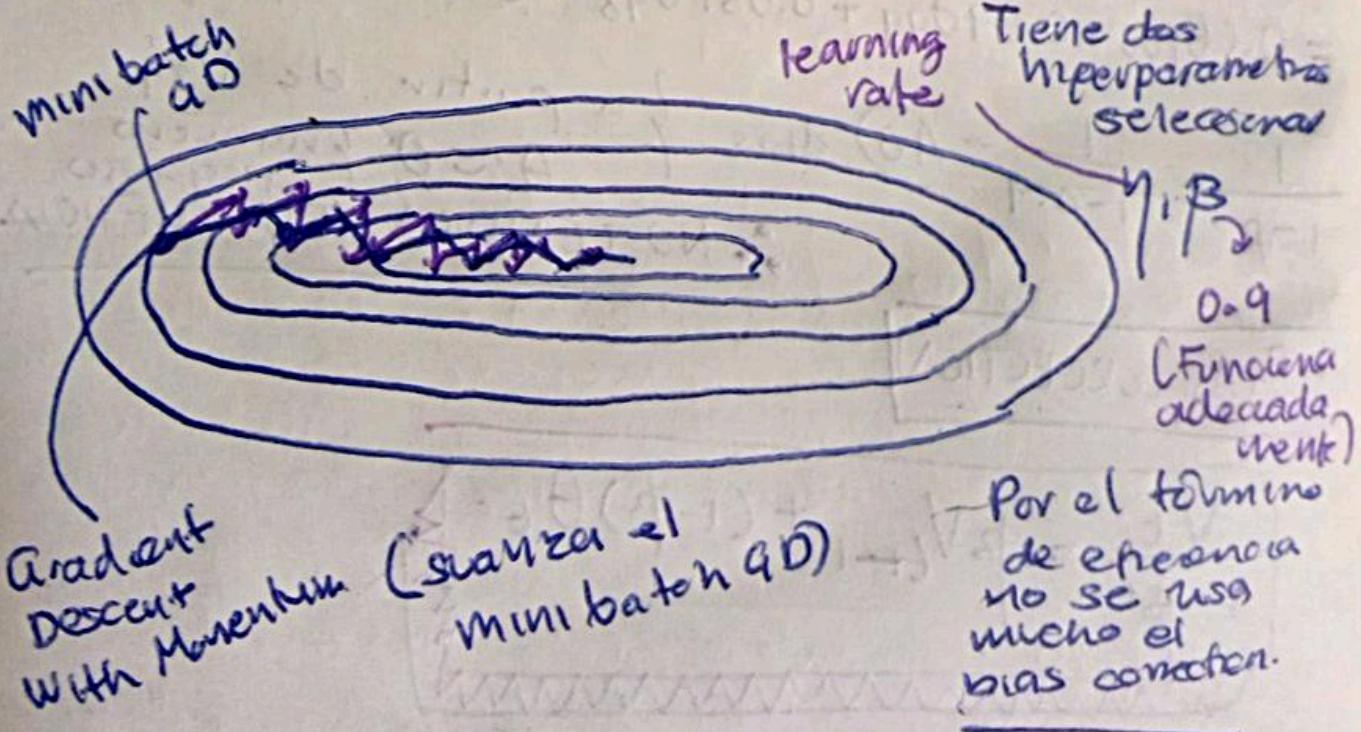
$$Vdw_1 = \beta \cdot Vdw_1 + (1-\beta) \partial w_1, \dots, Vdw_n = \dots$$

$$Vdb = \beta Vdb + (1-\beta) \partial b$$

actualizar los parámetros

$$w_1 = w_1 - \eta Vdw_1, \dots, w_n = w_n - \eta Vdw_n$$

$$b = b - \eta Vdb$$



RMS Prop

VARIABLES TIPOOS

$$\eta = 0.001$$

$$\beta = 0.9$$

$$\epsilon = 10^{-8} = \Sigma$$

for $j=1$ hasta # mini batches:

for $i=1$ hasta # ejemplo minibatch:

$\hat{y}^{(i)}$ = forward-propagation().

$$J = J(\hat{y}^{(i)}, y^{(i)})$$

$$dw_1 = \frac{\partial J}{\partial w_1}, \dots, dw_n = \frac{\partial J}{\partial w_n}, db = \frac{\partial J}{\partial b}$$

$$J = J/m^{\frac{1}{2}}, dw_1 = dw_1/m^{\frac{1}{2}}, \dots, dw_n = dw_n/m^{\frac{1}{2}}$$

$$db = db/m^{\frac{1}{2}}$$

$$sdw_1 = \beta sdw_1 + (1-\beta) dw_1^2, \dots,$$

$$sdw_n = \beta sdw_n + (1-\beta) dw_n^2,$$

$$sd_b = \beta sd_b + (1-\beta) dw_b^2$$

actualizar de la sig manera:

$$w_1 = w_1 - \eta \frac{dw_1}{\sqrt{sdw_1 + \epsilon}}, \dots, w_n = w_n - \eta \frac{dw_n}{\sqrt{sdw_n + \epsilon}}$$

$$b = b - \eta \frac{db}{\sqrt{sd_b + \epsilon}}$$

$\epsilon \rightarrow$ constante epsilon 10^{-8}

RMSProp surge para resolver los problemas de oscilación y lentitud en direcciones escala de gradiente (por ejemplo, cuando una dimensión tiene gradientes muy grandes y otros muy pequeños)

Acumulado de gradiente al cuadrado (suavizado exponencial)

$$sdw = \beta sdw + (1-\beta)(dw)^2$$

$\beta \approx 0.9$ hiperparámetro de suavizado

RMS adopta el paso de \rightarrow
aprendizaje a cada
paso

J, un peso tiene gradientes
grandes, su
sdw crece, su actualización
se reduce

Si un peso tiene gradientes pequeños, su δdW es pequeño, su actualización aumenta.

Resultado: El aprendizaje se estabiliza y converge más rápido que con SGD puro o Momentum.

ADAM

$VdW_1 = 0, \dots, VdW_n = 0, Vdb = 0, \delta dW_1 = 0, \dots, \delta dW_n = 0$

for $j=1$ hasta #mini-batches:

for $i=1$ hasta #ejemplo mini-batches:

$\hat{y}^{(i)} = \text{Forward-propagation}$

$J_t = \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

$dW_1 = \frac{\partial J}{\partial W_1}, \dots, dW_n = \frac{\partial J}{\partial W_n}, db = \frac{\partial J}{\partial b}$

$J = J/m^{fit}, dW_1 = dW_1/m^{fit}, \dots, dW_n = dW_n/m^{fit}$

$b = b/m^{fit}$

Momentum
GD

$$\begin{cases} VdW_1 = \beta_1 VdW_1 + (1 - \beta_1) dW_1 \\ VdW_n = \beta_1 VdW_n + (1 - \beta_1) dW_n \\ Vdb = \beta_1 Vdb + (1 - \beta_1) db \end{cases}$$

RMSProp

$$\begin{cases} \delta dW_1 = \beta_2 \delta dW_1 + (1 - \beta_2) dW_1^2 \\ \delta dW_n = \beta_2 \delta dW_n + (1 - \beta_2) dW_n^2 \\ \delta db = \beta_2 \delta db + (1 - \beta_2) db^2 \end{cases}$$

Adam aplca bias correction.

$$V_{dw_1}^{corr} = VdW_1 / (1 - \beta_1^j) \dots$$

$$V_{dw_n}^{corr} = VdW_n / (1 - \beta_1^j)$$

$$Vdb = Vdb / (1 - \beta_1^j)$$

$$\delta_{dw_1}^{corr} = \delta dW_1 / (1 - \beta_2^j), \dots, \delta_{dw_n}^{corr} = \delta dW_n / (1 - \beta_2^j)$$

$$w_1 = w_1 - \eta \frac{V_{dw_1}^{corr}}{\sqrt{\delta_{dw_1}^{corr} + \epsilon}}, \dots, b = b - \eta \frac{V_{db}^{corr}}{\sqrt{\delta_{db}^{corr} + \epsilon}}$$

Hiperparámetros

- $\eta = \text{learning rate}$ (a probar) $\sum \text{o } \epsilon \rightarrow 10^{-9}$
 $\beta_1 \rightarrow 0.9$ (mejor función)
 $\beta_2 \rightarrow 0.999$ (mejor función)

Adam (Adaptive Momentum Estimation)

Combinación lo mejor de Momentum & RMSprop

- Usa el promedio exponencial de los gradientes (como Momentum)
- Usa el promedio exponencial de los gradientes al cuadrado (como RMSprop)

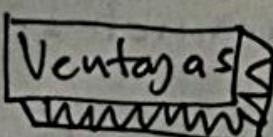
Esto permite que Adam adapte la tasa de aprendizaje para cada parámetro y suavice la dirección del gradiente

Gradient Descent \rightarrow lento, paso fijo

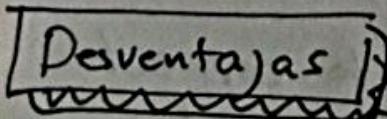
Momentum \rightarrow suave dirección

RMSprop \rightarrow Adapta paso

ADAM = Momentum + RMSprop.



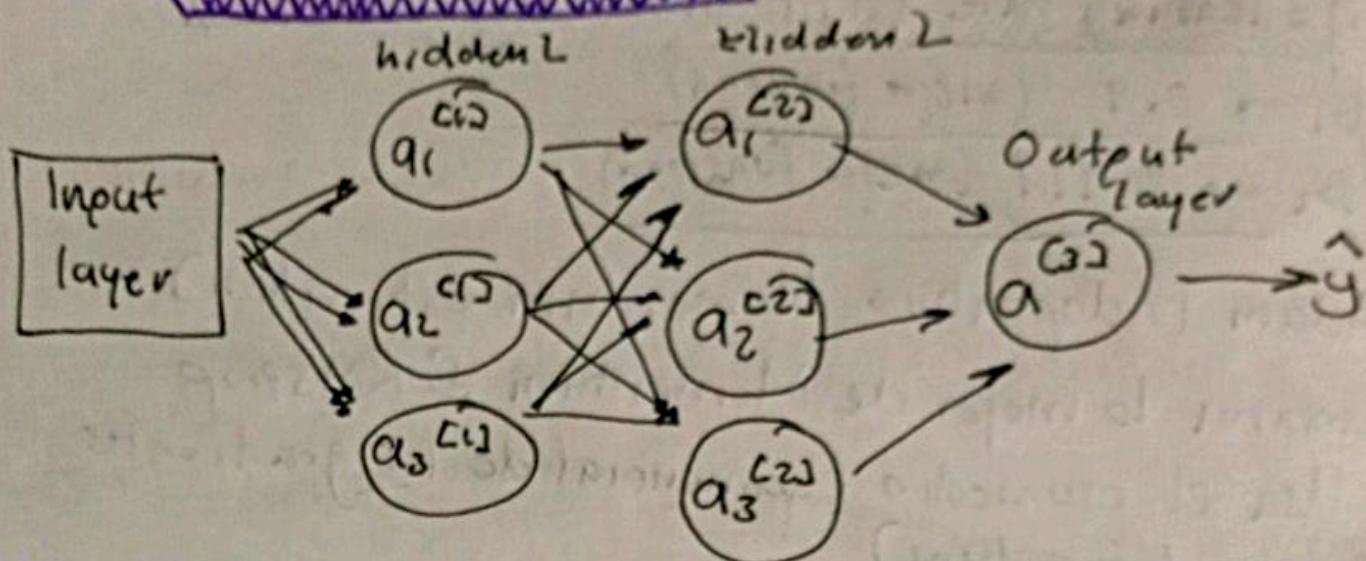
- Adapta el η (learning rate) para cada parámetro
- Combina estabilidad y rapidez
- Excelente rendimiento en la mayoría de problemas de Deep learning



- Puede causar overfitting
- No siempre alcanza el mínimo global, aunque converge rápido a un local estable.

HIPERPARÁMETRO

X_1
 X_2
 X_3
:
 X_n
1



Seleccionar

(Decisiones)

- # hidden layer
- # Neuronas por hidden layer
- Función de activación de las neuronas hidden layer
- Función de activación de las neuronas de la output layer.
- learning rate (η) algoritmo de optimización
- Algoritmo de optimización.

¿Cómo seleccionar todos estos hiperparámetros?

Keras Tuner

Es una biblioteca que ayuda a elegir el conjunto óptimo de hiperparámetros para el modelo.

El proceso de seleccionar el conjunto correcto de hiperparámetros para su aplicación de aprendizaje automático (ML) se le llama tuning de hiperparámetros o hypertuning.

¿Qué son los hiperparámetros?

Son valores que no aprende el modelo, sino que se establecen antes del entrenamiento. Ajustarlos bien es crucial, porque determinan que tan rápido y que tan bien aprende la red.

El objetivo es encontrar la combinación óptima de hiperparámetros que maximice el rendimiento del modelo.

Métodos Principales:

Método	Descripción	Ventajas	Desventajas
Manual Search	Ajustar a mano según experiencia	Simple	Ineficiente
Grid Search	Prueba todas las combinaciones posibles	Completo	Alto Costo Computacional.
Random Search	Toma combinaciones aleatorias	Rápido/efectivo	No garantiza que sea óptimo.
Bayesian Optimization	Aprende qué cambios probar después	Muy eficiente	Más complejo
Keras tuner	Implementa varias de estas métodos en Keras.	Fácil de usar	Puede ser incómodo con muchos parámetros

Hiperparámetros relevantes

Número de Capas Óptimo

Las redes neuronales artificiales con un número elevado de hidden layers tiene una eficiencia de los parámetros más altas, permiten alcanzar un rendimiento más elevado con el mismo conjunto de datos de entrenamiento.

El número elevado de hidden layers facilitan la generalización para nuevos conjuntos de datos: transfer learning?

¿Cómo seleccionar el # óptimo de capas?

Número de neuronas óptimo

El número de neuronas de la input layer y de la output layer está determinado por el tipo de entrada y salida que requiere nuestra tarea.

Lo más común es que el # de neuronas por cada hidden layer formen una pirámide, con más neuronas en las primeras capas que en las últimas.

sin embargo, con el paso del tiempo se ha llegado a la conclusión que utilizar el mismo número de neuronas por capa arroja resultados muy similares y reduce el espacio de hiperparámetros a evaluar.

Un número muy elevado de neuronas normalmente produce Overfitting.

OTROS HIPERPARÁMETROS RELEVANTES

→ 0.001, 0.0001, 0.01,

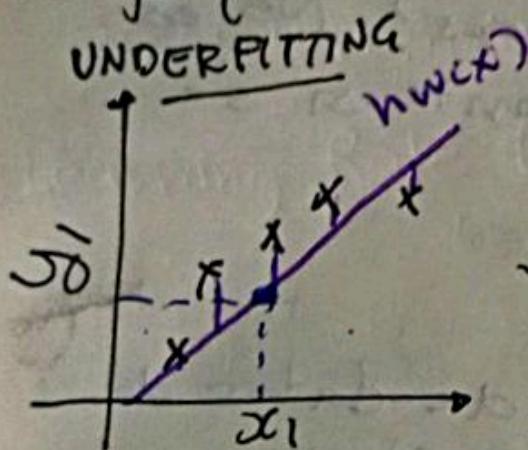
► Learning Rate (η): Es probablemente el hiperparámetro más importante. La estrategia de selección de η más utilizada consiste en comenzar con un valor más pequeño (10^{-5}) e incrementarlo hasta un valor grande (10^1). → ADAM

► Función de optimización: La más utilizada es ADAM, sin embargo, puede variarse con RMSProp, Momentum, etc. dependiendo del problema. → 2, 16, 32 → 512 Mejor rendimiento.

► Batch Size: Un tamaño de batch size elevado permite que componentes hardware como la GPU sean más eficiente procesando los ejemplos de entrenamiento. Sin embargo, en la práctica entrenar con un batch size elevado puede conducir a instabilidades en el entrenamiento que generen un modelo que no sea capaz de generar bien. Como norma utiliza batches 12 y 32 selección

OVERFITTING & UNDERFITTING

Si nuestra red neuronal tiene muchas capas y neuronas, es posible que la función hipótesis generada por el algoritmo que se adapte muy bien al conjunto de entrenamiento $J(w) = 0$ pero falle al generalizar con nuevos ejemplos a esto se le llama overfitting.

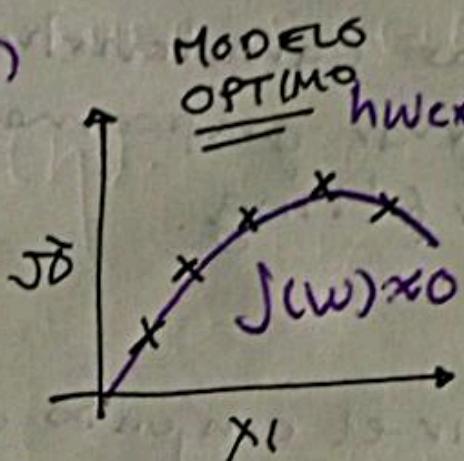


$$h_w(x) := w_0 + w_1 x_1$$

$$x_1 > 0 \rightarrow \hat{y}(h_w(x))$$

$$z = y_i w_i + w_0$$

UNDERFITTING



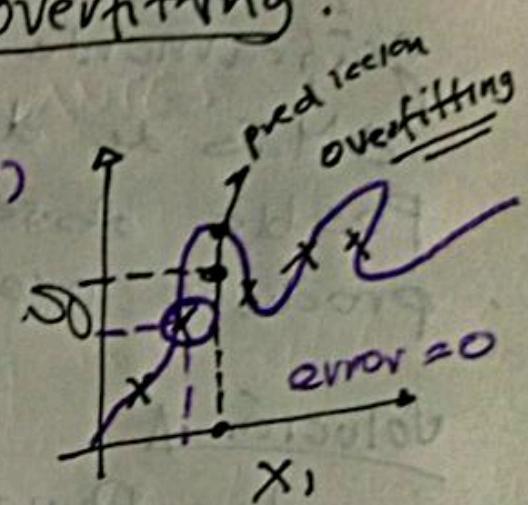
$$h_w(x) = w_0 + w_1 x_1 + w_2 x_1^2$$

$$\downarrow$$

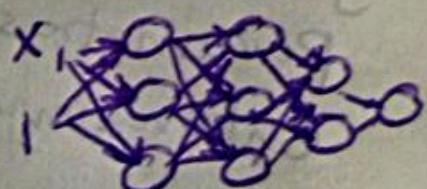
$$x_1 \rightarrow O \rightarrow \hat{y}(h_w(x))$$

Genera un pequeño, pero es muy cercano a cero.

MODELO OPTIMO



$$h_w(x) = w_0 + w_1 x_1 + w_2 x_1^2 + w_3 x_1^3 + \dots + w_n x_1^n$$



de ajuste tan bien

OVERFITTING

SOLUCIONES DE OVERFITTING



- Aumentar el conjunto de datos
- Reducción de numero de capas.
- Reducción de numero de neuronas
- Regularización • Dropout • Data Argumentation

DIVISIÓN DEL CONJUNTO DE DATOS

Evaluación de la hipótesis

- Un número elevado de capas y neuronas provoca overfitting.

Una/dos características se puede representar gráficamente la función hipótesis.
Más de dos características? No es posible observar gráficamente si se produce overfitting.

Solución

↳ Dividir el conjunto de datos de entrenamiento y prueba

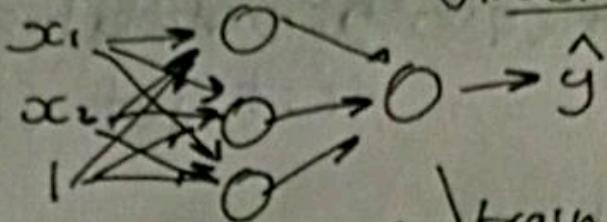
Se calcula la función hipótesis con el subconjunto de entrenamiento (se calculan los parámetros W minimizando el error del entrenamiento $J(W)$).

$2\% \rightarrow \text{test set}$

$98\% \rightarrow \text{train set}$

Selección del modelo

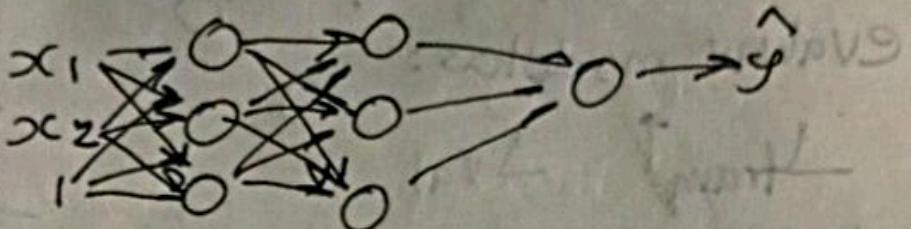
1 hidden layer



UNDERFITTING

el modelo no es suficiente, aumentar otra capa si el error de entrenamiento es alto.

2 hidden layers



$J_{train} \approx 0.2$

$J_{train} \approx 0.25$ en este caso
no se produce
overfitting, modelo óptimo.

n hidden layers



$J_{train} \approx 0$ $J_{test} \approx \uparrow$ es alto

Produce overfitting (NO ÓPTIMO)

Subconjunto train (98%)
Subconjunto test (2%)

Puede que el modelo "óptimo" sin embargo no puede generalizar tan bien

Agregar un nuevo subconjunto (Validación)

Finalmente, la división quedaria

train set (98%)

evaluar el valor

Validation set (1%)

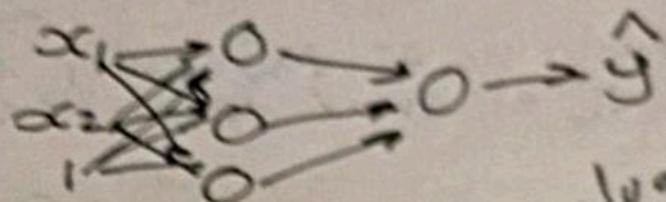
óptimo de los hiperparámetros

test set (1%)

evaluar ultima vez para analizar el error bajo

train set (78%); validation set (11%);
test set (1%)

1 hidden layer



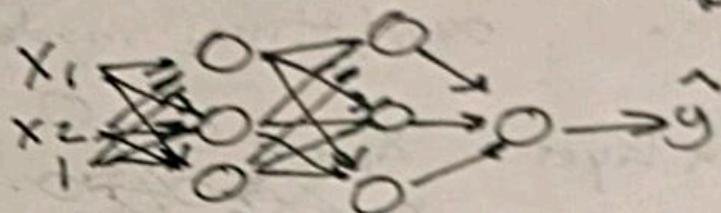
Evaluar modelos:

↓ train ↑ val

Underfitting

evaluar
↓ test →
probar al
final de
todo!

2 hidden layer

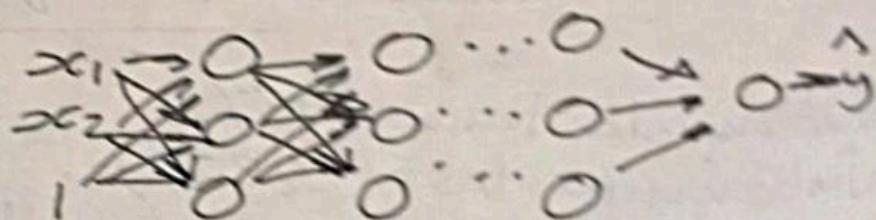


Evaluar modelo:

↑ train ↓ val

Underfitting

n hidden layer



↓ train ↓ val ↓

Ji ↓ train ∧ ↓ val son bajas UNDERFITTING

entradas ↓ test

Ji ↓ test es bajo → Generaliza bien

Ji ↓ test es alto → OVERFITTING

hipótesis

OVERFITTING

REGULARIZACIÓN

La regularización agrega una penalización en los diferentes parámetros del modelo para reducir la libertad del modelo. Por lo tanto es menos probable que el modelo se ajuste al ruido de los datos de entrenamiento y mejorará las capacidades de generalización del mismo.

- Mantiene todas las características, pero reduce la magnitud de los parámetros w .
- La regularización funciona bien cuando tenemos características ligeramente útiles

$\rightarrow J + \text{penalización}$

tensorflow \rightarrow ^{lo}
aplica por defecto

$$x_1 \rightarrow z/a \rightarrow \hat{y} = \text{sigmoid}(z) \quad \hat{y} = \text{sigmoid}(z)$$

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m 2(\hat{y}_i, y^{(i)}) + \frac{\lambda}{2m} \|w\|_2^2$$

$$\|w\|_2^2 = \sum_{j=1}^m w_j^2 \quad \text{error conjunto de datos}$$

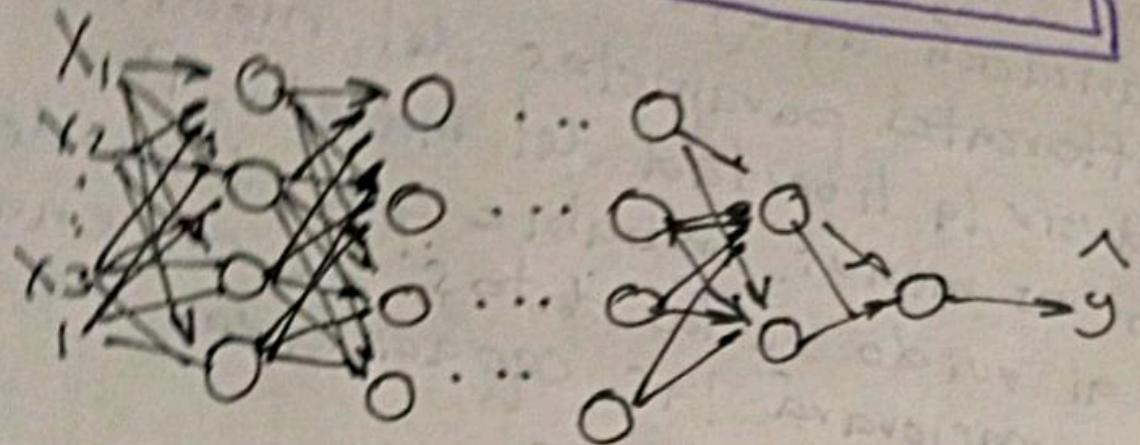
λ = hiperparámetro
ajustado manual

L2 regularización más utilizado

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m 2(\hat{y}_i, y^{(i)}) + \frac{\lambda}{m} \sum_{j=1}^m \|w_j\|_1$$

$\|w_j\|_1 \leftarrow L_1$ regularization menos utilizado

$\|b\| \rightarrow$ no se suele añadir



$$J(w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}, \dots, w^{(L)}, b^{(L)}) = \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}^{(i)}, y^{(i)})$$

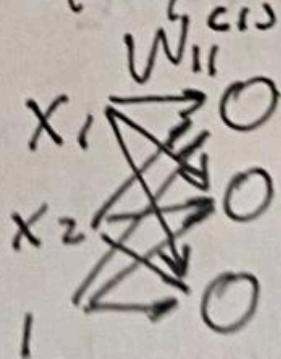
O bien

$$J(w, b) := \frac{1}{m} \sum_{i=1}^m \ell(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{i=1}^L \|w^{(i)}\|_F^2$$

entonces:

$$\|w^{(i)}\|_F^2 = \sum_{j=1}^{n^{(i-1)}} \sum_{j=1}^{n^{(i)}} (w_{ij}^{(i)})^2 \quad \begin{matrix} \text{Frobenios} \\ \text{norm} \end{matrix}$$

Ejemplo:



$$\begin{aligned} \|w^{(1)}\|_F^2 &= \sum_{i=1}^n \sum_{j=1}^{n'} (w_{ij}^{(1)})^2 \\ &= W_{11}^{(1)} + (W_{12}^{(1)})^2 + (W_{21}^{(1)})^2 + (W_{22}^{(1)})^2 \end{aligned}$$

$\lambda \rightarrow$ hiperparámetro de regularización
asignarlo de manera manual

Gradient Descent y regularización

for j en # mini batches:

for i en # ejemplos mini batches:

$$\hat{y}^{(i)} = \text{forward_propagation}()$$

$$J_+ = \lambda(\hat{y}^{(i)}, y^{(i)})$$

$$dW_1 = \frac{\partial J}{\partial W_{11}^{(i)}} ; \text{ donde}$$

$$\frac{\partial J}{\partial W_{11}^{(i)}} = \frac{\partial}{\partial W_{11}^{(i)}} \left(\frac{1}{m} \sum_{i=1}^m \lambda(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{i=1}^L \|W^{(i)}\|_F^2 \right)$$

$$\|W^{(i)}\|_F^2$$

Por lo tanto:

$$dW_1^{(i)} = \frac{\partial J}{\partial W_{11}^{(i)}} + \frac{\lambda}{m} W_{11}^{(i)}$$

actualización de los parámetros

en otras palabras, se hace:

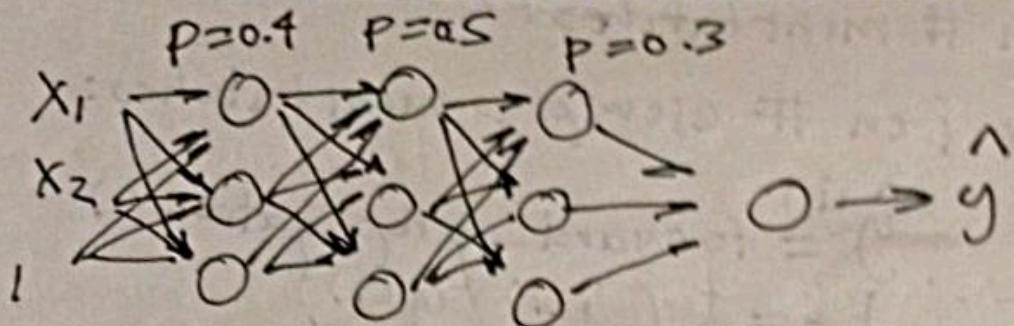
exactamente lo mismo solo que se denota
ahora con la suma de regularización en
backward propagation

$$\frac{\partial J}{\partial W_{11}^{(i)}} = \frac{\partial}{\partial W_{11}^{(i)}} \left(\underbrace{\frac{1}{m} \sum_{i=1}^m \lambda(\hat{y}^{(i)}, y^{(i)})}_{\text{backward propagation}} + \underbrace{\frac{\lambda}{2m} \sum_{i=1}^L \|W^{(i)}\|_F^2}_{\text{regularización}} \right)$$

backward
propagation

regulariza-
ción

Dropout Regularization



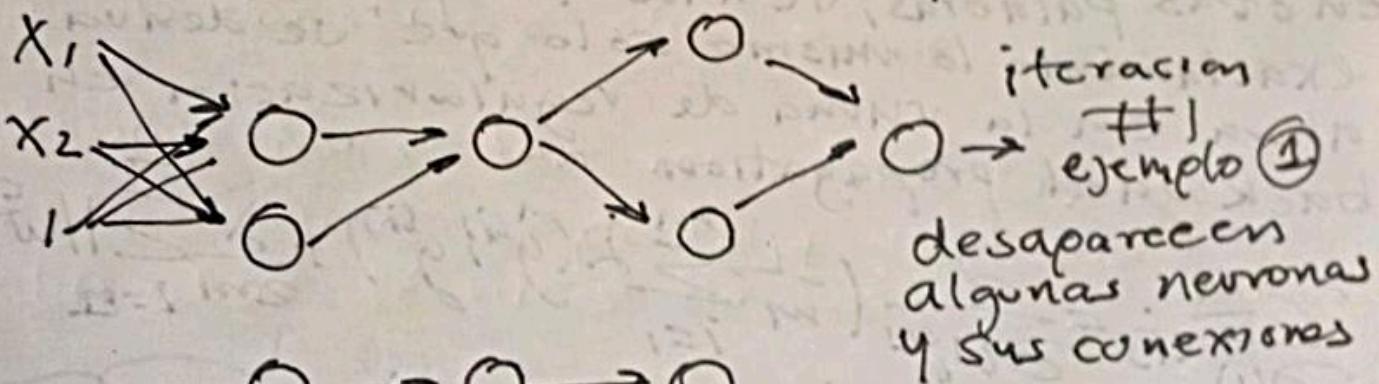
asignar probabilidades a las hidden layers.

for _____

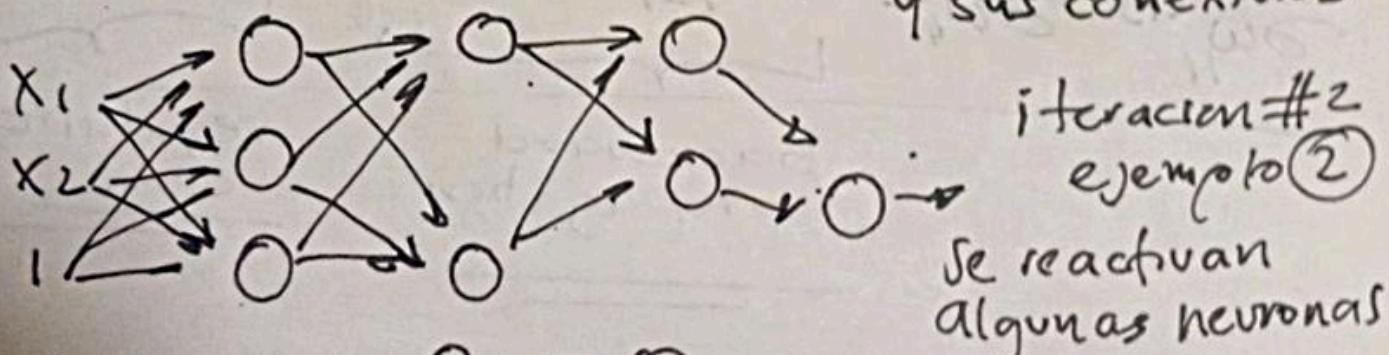
for ^{forward} _{backward} } ir eliminando
neuronas con base
a la probabilidad
asignada.

$P=0.5 \rightarrow$ probabilidad de desaparecer
de las hidden layers

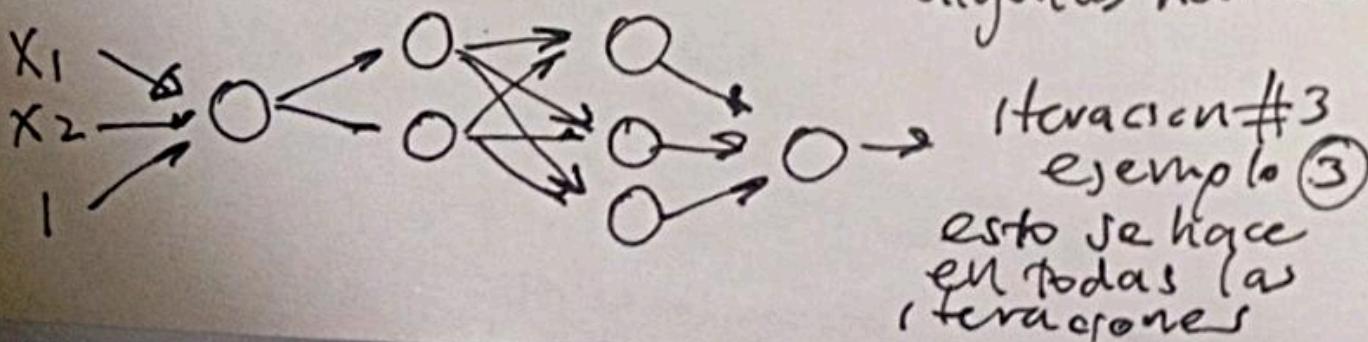
①



②



③



EXPLICACION

Cada una de las neuronas se encargan en identificar una característica del problema que se desea resolver al ajustar los pesos y los bias en una red conectada.

Con Dropout regularización estamos haciendo que las neuronas de las diferentes capas no dependan de las mismas neuronas de la capa anterior (mismas features) cuando procesan sus características, ya que al pasar la siguiente iteración (sig. ejemplo de entrenamiento), esa neurona que dependía de la capa anterior ha desaparecido. Esto hace que las neuronas se "especialicen" menos al la hora de identificar determinadas características generalizan mejor y veinte el overfitting. Esta técnica funciona muy bien, las CNN (redes neuronales convolucionales) las aplican en gran porcentaje, e incluso las redes neuronales recurrentes (20%).

Dropout actúa como si entrenáramos muchas redes neuronales distintas y luego combináramos sus predicciones

Dropout \approx train many small networks + average them at inference

OTROS MECHANISMOS DE REGULARIZACION

Data augmentation

Se aumenta el conjunto de datos de entrenamiento mediante la modificación de los ejemplos existentes

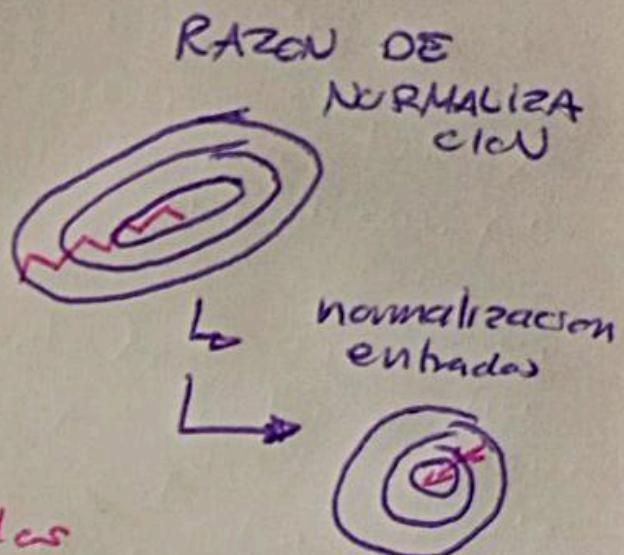
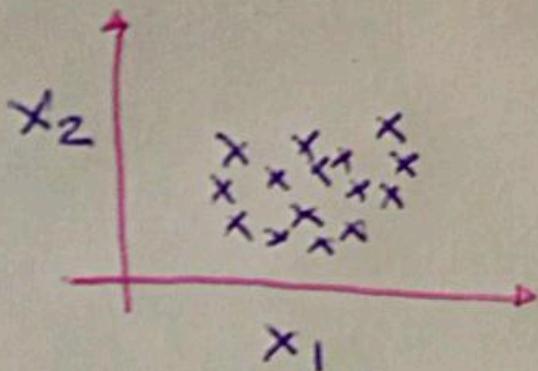
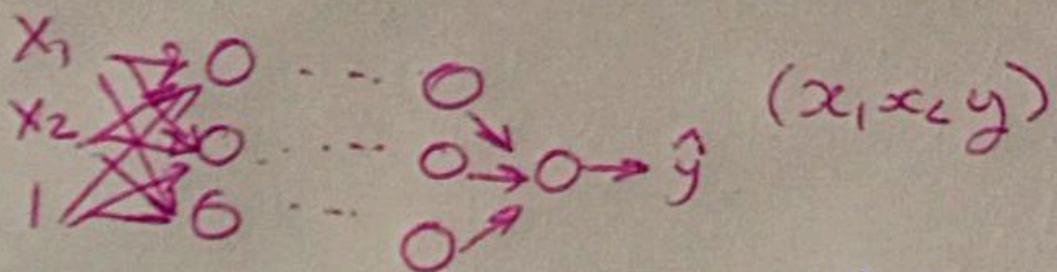
Train set → aumentarlo

Early Stopping

Detener el entrenamiento de la red neuronal antes de que minimice el error con el conjunto de datos de entrenamiento

#NOTA: No es tan recomendada, pero esta técnica se podría utilizar si las otras técnicas no funcionan

Normalización de entradas



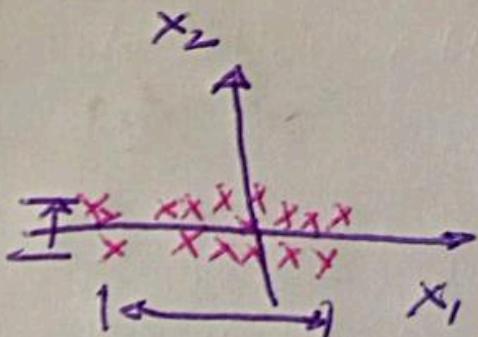
① Normalizar las entradas

1 → restar la media:

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

Datos en una escala similar

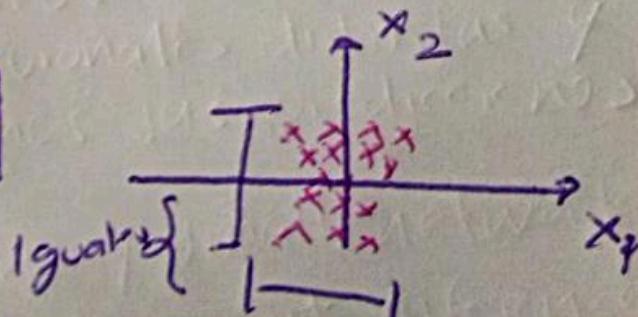
$$x = x - \mu$$



2 → normalizar la varianza σ^2

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m x^{(i)} - \mu^2 \quad \left\{ \begin{array}{l} \text{elevar al cuadrado} \\ \text{restar la media} \end{array} \right.$$

$$x = \frac{x - \mu}{\sigma^2}$$



utilizar el mismo σ^2

para nuestro conjunto de val, test y nuevos ejemplos.