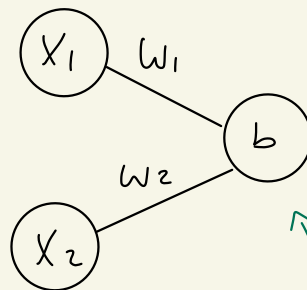
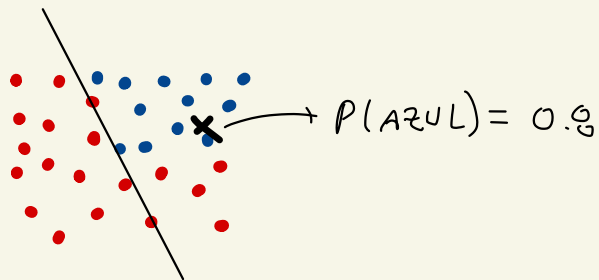
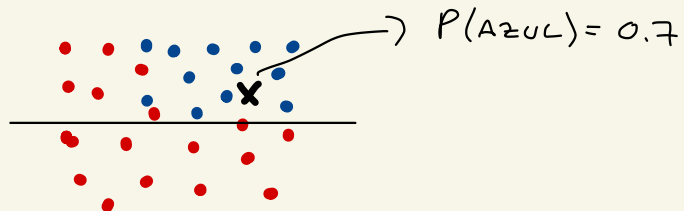


En ocasiones
se denota

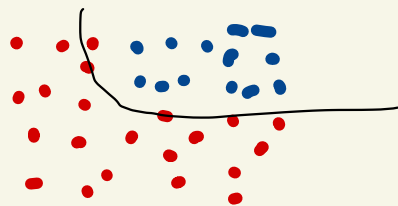


El sesgo
en el nodo

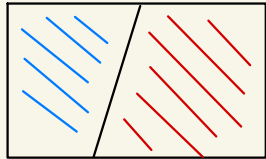
Cuando una recta no es suficiente



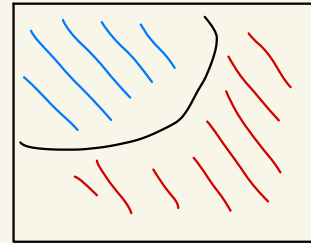
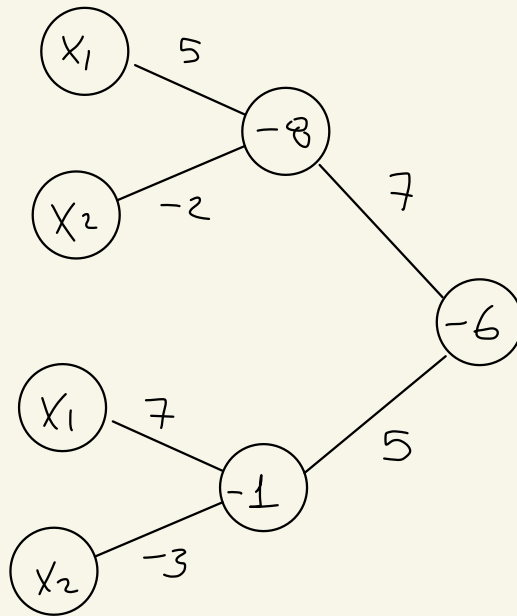
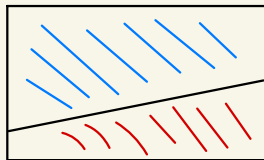
$$0.7 + 0.8 = 1.5 \xrightarrow{\sigma} 0.82$$



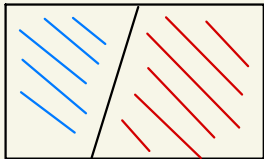
$$5x_1 - 2x_2 - 8$$



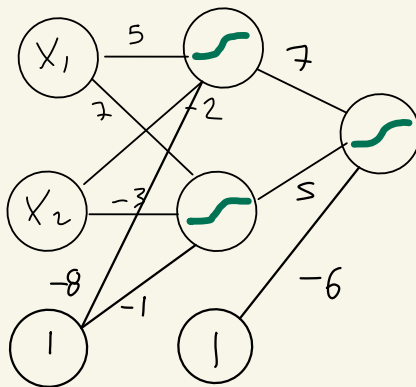
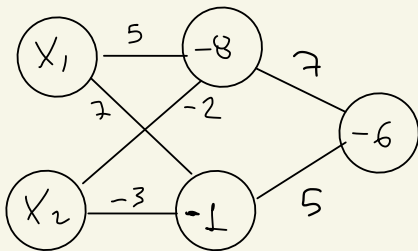
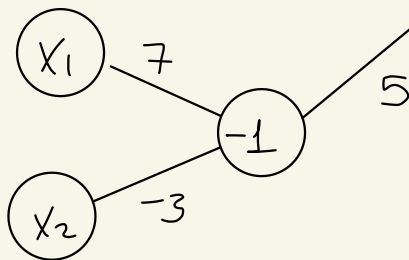
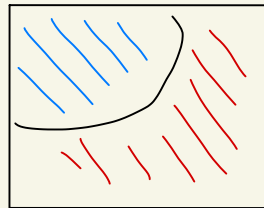
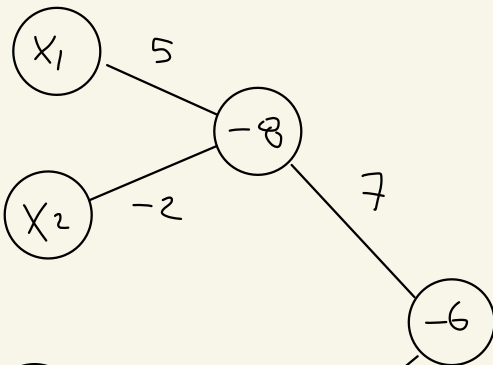
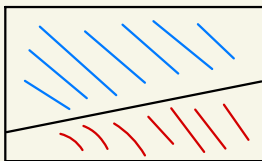
$$7x_1 - 3x_2 - 1$$

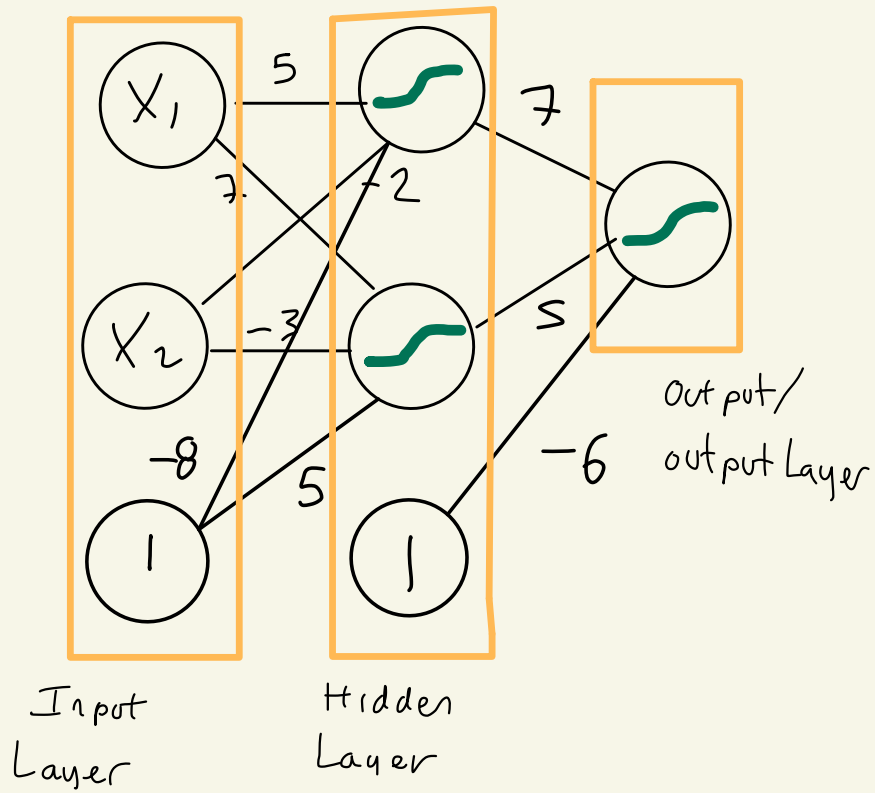


$$5x_1 - 2x_2 - 8$$

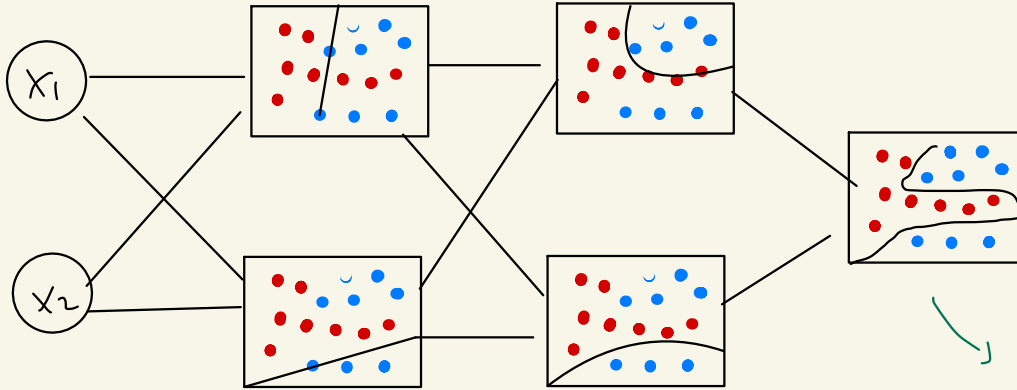


$$7x_1 - 3x_2 - 1$$





Más capas?

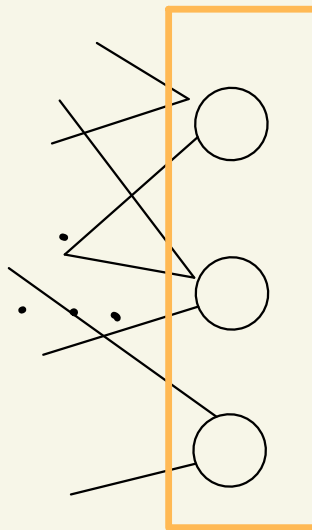
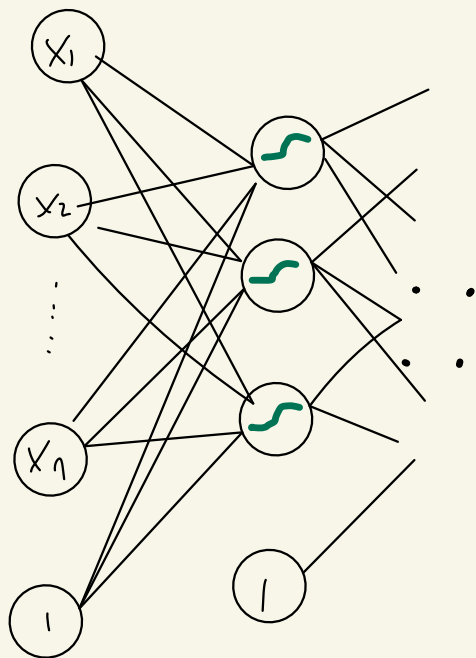



Red neuronal profunda


Curvas más complicadas!


Más nodos en la salida?

3 Clases Por ejemplo,
Gato, Perro, León



→ 0.09 

→ 0.67 

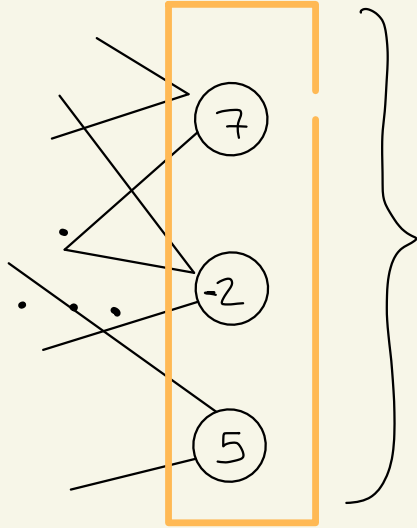
→ 0.24 

Output Layer

Se clasificaría
como **perro**

Soft max function

Salida



Cómo
expresarlos
como probabilidades?

$$\frac{e^7}{e^7 + e^{-2} + e^5} =$$

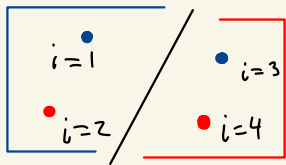
$$\frac{e^{-2}}{e^7 + e^{-2} + e^5} =$$

$$\frac{e^5}{e^7 + e^{-2} + e^5} =$$

Recordemos...

$$\text{Cross-entropy} = - \sum_{i=1}^M y_i \ln(p_i) + (1-y_i) \ln(1-p_i)$$

$\rightarrow P(\text{punto } i^{\text{va}} \text{ es rojo})$

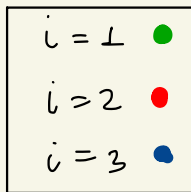


$$\begin{cases} 1 & \text{AZUL} \\ 0 & \text{ROJO} \end{cases} \rightarrow P(\text{punto } i^{\text{va}} \text{ es azul})$$

Multiclas es ?

Punto $j=1$

Punto $j=2$



$y_{1j} = 1$ si el punto j es verde

$y_{2j} = 1$ si el punto j es rojo

$y_{3j} = 1$ si el punto j es azul

Punto $j=3$

Punto $j=4$

$$p_{ij} = P(\text{El punto } j \text{ es de color } i)$$

$$\text{Cross-entropy} = - \sum_{i=1}^n \sum_{j=1}^m y_{ij} \ln(p_{ij})$$

Cross-entropy error function (para redes neuronales)

Predicción $\hat{y} = \sigma(\underbrace{Wx + b}_{\text{sigmoide}})$ → Comb. lineal de nodos y vértices

Función de error

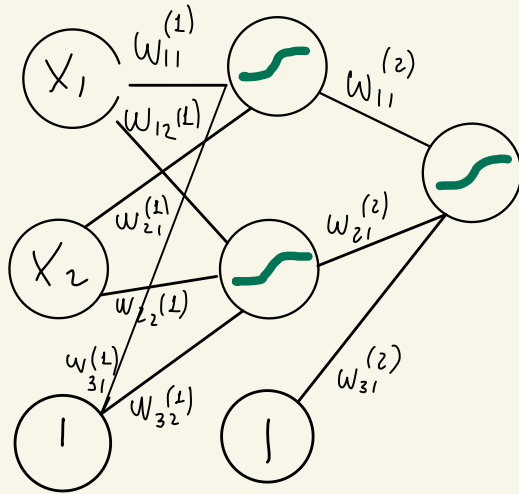
$$E(w) = -\frac{1}{m} \sum_{i=1}^m y_i \ln(\hat{y}_i) + (1-y_i) \ln(1-\hat{y}_i)$$

→ predicción para el vector (\vec{x}_i) (probabilidad)

→ Se toma el promedio en lugar de la suma

Feed Forward

Es el proceso a través del cual la red neuronal calcula una salida a partir de un vector de entrada

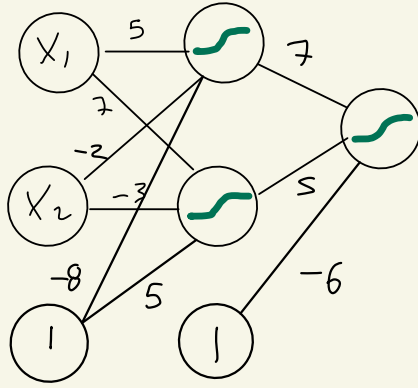


Feed forward

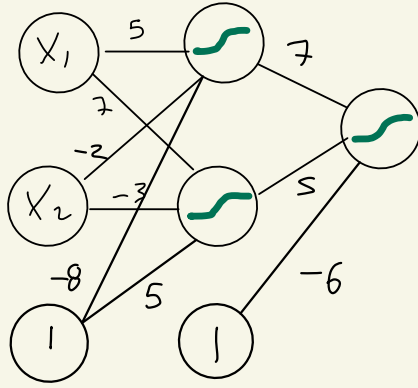
$$\sigma = \text{[green sigmoid curve]}$$

$$\hat{y} = \sigma \begin{pmatrix} w_{11}^{(2)} \\ w_{21}^{(2)} \\ w_{31}^{(2)} \end{pmatrix}^T \sigma \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} \end{pmatrix}^T \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

Predicción para el punto (x_1, x_2)



Ejemplo $\vec{x} = (1, -1)$ $\hat{y} = ?$



Ejemplo $\vec{x} = (1, -1)$

$$\sigma \begin{pmatrix} 7 \\ 5 \\ -6 \end{pmatrix}^T \sigma \begin{pmatrix} 5 & 7 \\ -2 & -3 \\ -8 & 5 \end{pmatrix}^T \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

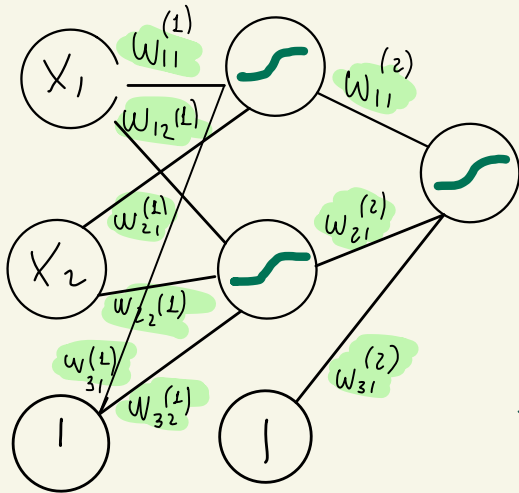
$$\sigma \begin{pmatrix} 7 \\ 5 \\ -6 \end{pmatrix}^T \sigma \begin{pmatrix} 5 & -2 & -8 \\ 7 & -3 & 5 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$$

$$\sigma = \frac{1}{1 + e^{-x}}$$

Entrenamiento de la red neuronal

Queremos encontrar los pesos que mejor modelen los datos

→ serán nuestras variables (no las x_i !!)



- 0 Inicializamos los pesos W aleatoriamente
- 1 Obtenemos un lote de datos,
batch "puntos"
- 2 Calculamos predicciones para este lote
usando Feedforward
- 3 Calculamos la función de error
- ? 4 Calculamos los gradientes de la función de
error para cada parámetro
- 5 Actualizamos los parámetros
- 6 Repetimos hasta tener un buen modelo!

¿Cómo calcular los gradientes?

Back propagation

Llamaremos a_i^k a la salida a de la función de activación (σ) en el nodo i de la capa k .

$$a_i^k = \sigma(z_i^k)$$

$$a_i^0 := x_i$$

$$z_i^k = \vec{a}^{k-1} \cdot \vec{w}_{-i}^k$$

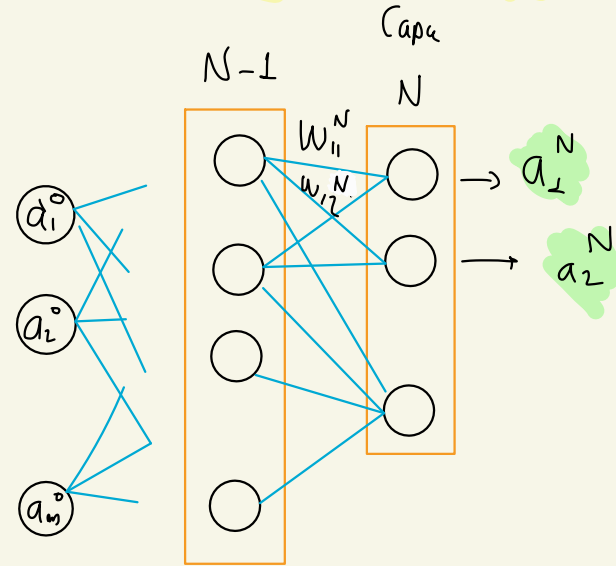
$$z_i^0 = Wx + b$$

$$E = E(\hat{y}) = E(\vec{a}^N)$$

Función de error

$$\frac{\partial E}{\partial w_{ji}^N} = \frac{\partial E}{\partial a_i^N} \cdot \underbrace{\frac{\partial a_i^N}{\partial z_i^N}}_{\sigma(1-\sigma)} \cdot \underbrace{\frac{\partial z_i^N}{\partial w_{ji}^N}}_{a_j^{N-1}}$$

$$\frac{\partial E}{\partial w_{li}^{N-1}} = \frac{\partial E}{\partial \vec{a}^N} \cdot \frac{\partial \vec{a}^N}{\partial \vec{z}^N} \cdot \underbrace{\frac{\partial \vec{z}^N}{\partial a_i^{N-1}}}_{w_{-i}^{N-1}} \cdot \underbrace{\frac{\partial a_i^{N-1}}{\partial z_i^{N-1}}}_{\sigma(1-\sigma)} \cdot \underbrace{\frac{\partial z_i^{N-1}}{\partial w_{li}^{N-1}}}_{a_l^{N-2}} a^{N-2}$$



- ✓ 0 Inicializamos los pesos W aleatoriamente
- ✓ 1 Obtenemos un lote de datos,
batch "puntos"
- ✓ 2 Calculamos predicciones para este lote
usando Feedforward
- ✓ 3 Calculamos la función de error
- ✓ 4 Calculamos los gradientes de la función de
error para cada parámetro
- ? 5 Actualizamos los parámetros
- 6 Repetimos hasta tener un buen modelo!

usamos las derivadas para actualizar los pesos

$$w_{ji} = w_{ji} - \boxed{\frac{\partial E}{\partial w_{ji}}} w_{ji}$$

↪ Back propagation

- ✓ 0 Inicializamos los pesos W aleatoriamente
- ✓ 1 Obtenemos un lote de datos,
batch "puntos"
- ✓ 2 Calculamos predicciones para este lote
usando Feedforward
- ✓ 3 Calculamos la función de error
- ✓ 4 Calculamos los gradientes de la función de
error para cada parámetro
- ✓ 5 Actualizamos los parámetros
- 6 Repetimos hasta tener un buen modelo!