

Aprendizaje Automático

Clase 8:

Regresión Logística
Entropía Cruzada
Redes Neuronales
Backpropagation

Repaso

Regresión

Objetivo del aprendizaje supervisado (caso regresión)

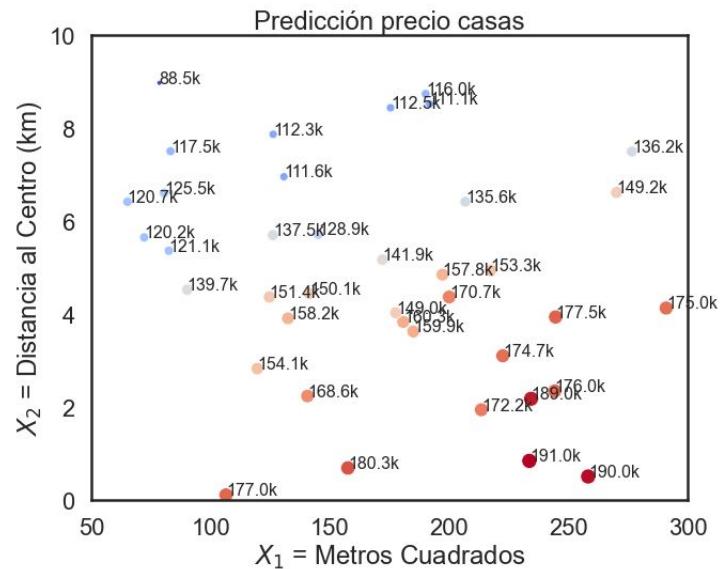
Estimar la **función determinista $f(\mathbf{X})$** que determina la relación $\mathbf{X} \rightarrow \mathbf{Y}$:

Es decir, estimar f tal que $\mathbf{Y} = f(\mathbf{X}) + \varepsilon$ a través de un modelo $\hat{h}_D(\mathbf{X})$. En donde ε es el error irreducible.

En donde $\mathbf{Y} \in \mathbb{R}$.

Ejemplo

Estimar el \mathbf{Y} = **precio de una casa** según
 X_1 = los metros cuadrados cubiertos,
 X_2 = su distancia al centro de la ciudad.



MSE como función de costo

Una métrica muy utilizada en problemas de regresión.

$$\text{MSE}_{X,y} = \frac{1}{n} \sum_{i=1}^n (\hat{h}(x^{(i)}) - y^{(i)})^2$$

Vimos que para regresión lineal suponemos

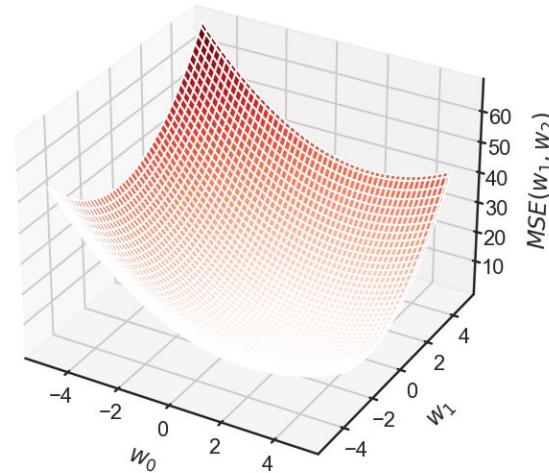
$$Y = w_0 + w_1 X_1 + w_2 X_2 + \dots + w_p X_p + \varepsilon$$

Es decir, \mathbf{h} ya entrenado tendrá la siguiente forma:

$$\hat{h}_{w_0, \dots, w_p}(x^{(i)}) = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_p x_p^{(i)}$$

$$\begin{aligned}\text{MSE}_{X,y} &= \frac{1}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{n} \sum_{i=1}^n (w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_p x_p^{(i)} - y^{(i)})^2\end{aligned}$$

Pensando al MSE como **una función de costo que depende de los** parámetros de un modelo $\text{MSE}_{X,y}(w)$.



“Entrenar el modelo”: encontrar los w que minimicen la siguiente expresión

$$\text{MSE}_{X,y}(w) = \frac{1}{n} \sum_{i=1}^n (\hat{h}_{w_0, w_1, \dots, w_p}(x^{(i)}) - y^{(i)})^2$$

Minimizando el MSE(w)

Para encontrar los pesos óptimos w_0, w_1, \dots, w_p según un dataset, podemos considerar $\text{MSE}(w)$ como la función que deseamos optimizar.

Vimos como utilizar Descenso por el Gradiente, seteando la función de costo J de la siguiente manera:

$$(a) J_{X,y}(w) = \text{MSE}_{X,y}(w) = \frac{1}{n} \sum_{i=1}^n (\hat{h}_w(x^{(i)}) - y^{(i)})^2$$

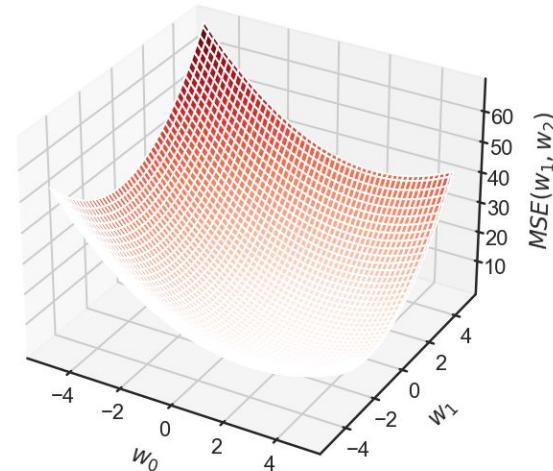
$$(b) \nabla_w J_{X,y}(w) = \nabla_w \text{MSE}_{X,y}(w) = \frac{2}{n} \sum_{i=1}^n (\hat{h}_w(x^{(i)}) - y^{(i)}) * x^{(i)}$$

En donde X, y son $X_{\text{train}}, y_{\text{train}}$ (o un mini batch)

Preguntas para pensar ahora.

- ¿La expresión (a) representa el $\text{MSE}_{X,y}$ para cualquier modelo o vale sólo para regresión lineal?
- ¿La expresión (b) representa el $\nabla \text{MSE}_{X,y}$ siempre? (o es $\nabla \text{MSE}_{X,y}^{\text{reg-lineal}}$)?
- ¿Es $\text{MSE}(w)$ una función convexa, o sólo $\text{MSE}_{X,y}^{\text{reg-lineal}}(w)$?

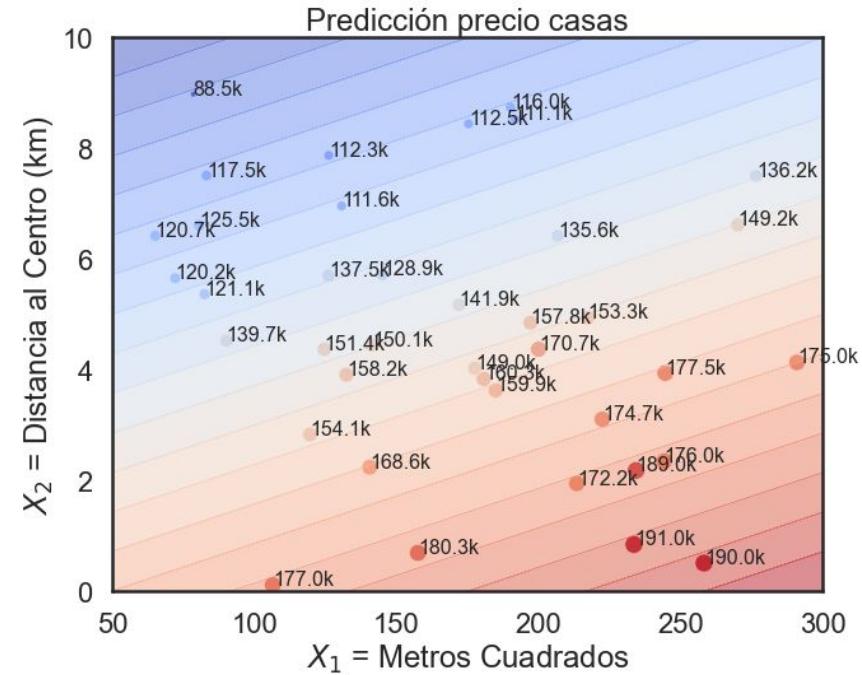
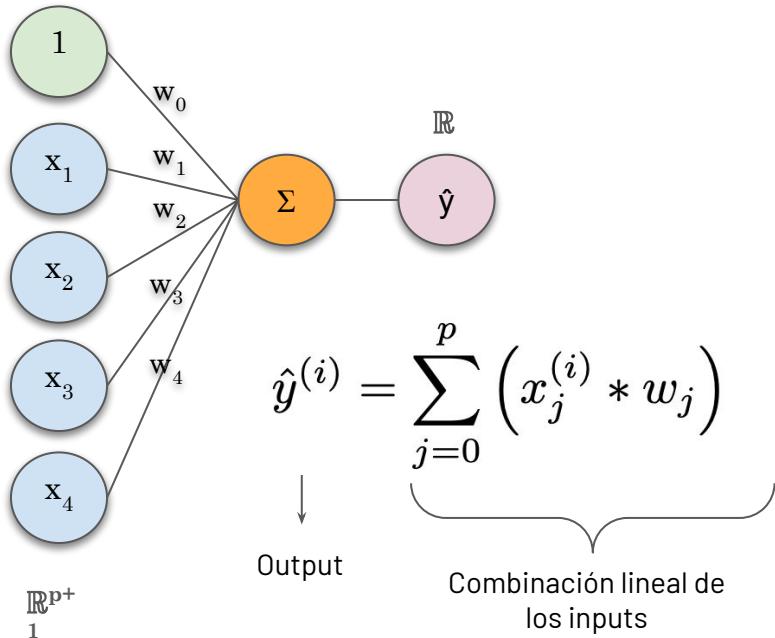
“Entrenar el modelo”: encontrar los w que minimicen la siguiente función



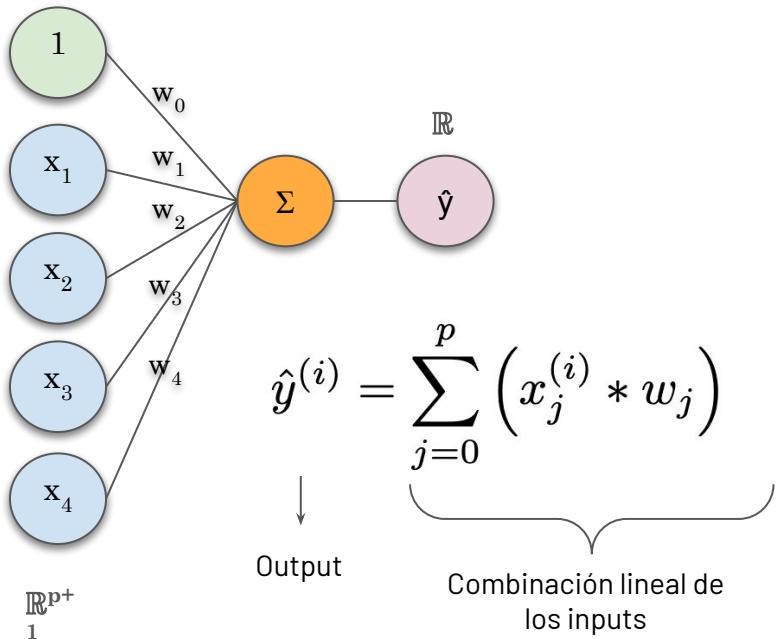
Regresión Logística

Otra visualización para regresión lineal

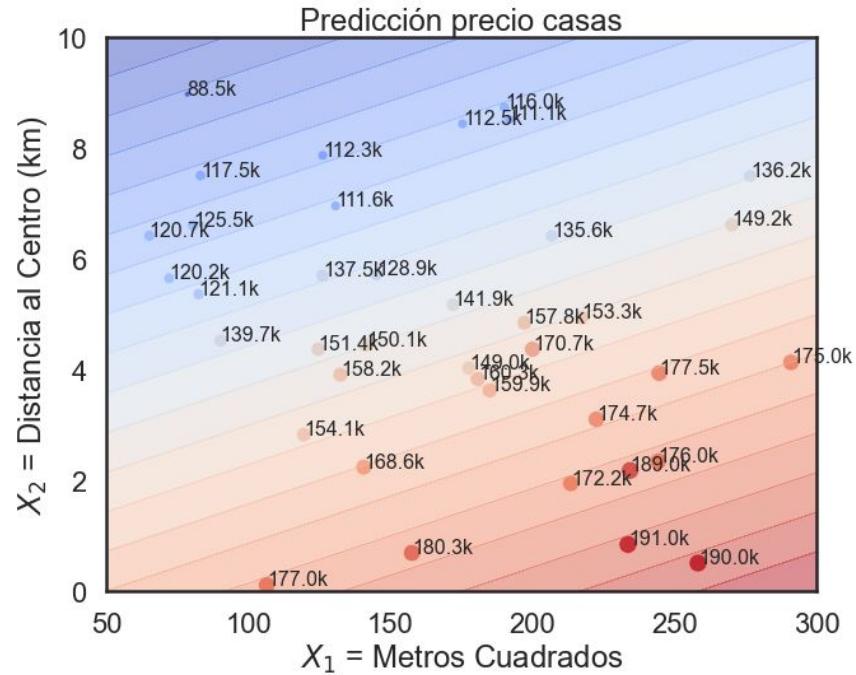
$$\hat{h}_{w_0, \dots, w_p}(x^{(i)}) = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_p x_p^{(i)}$$



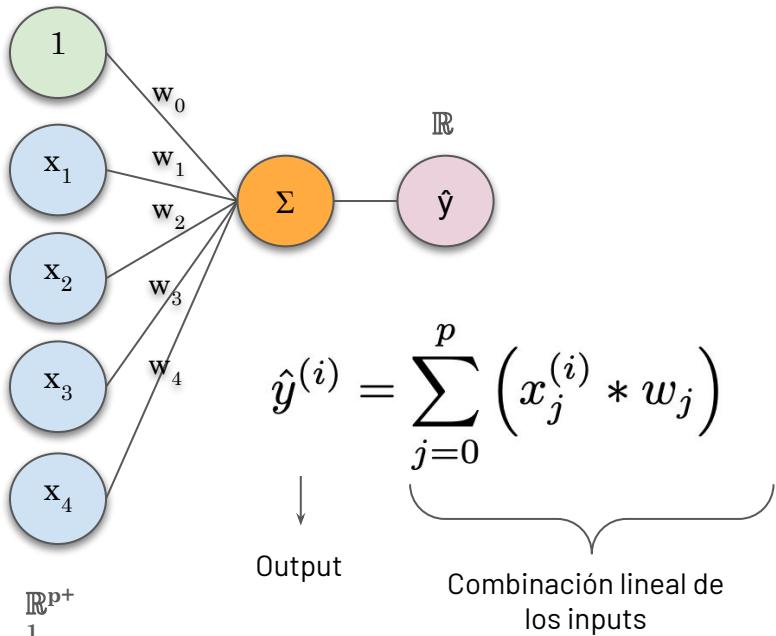
Convirtiendo a clasificación binaria



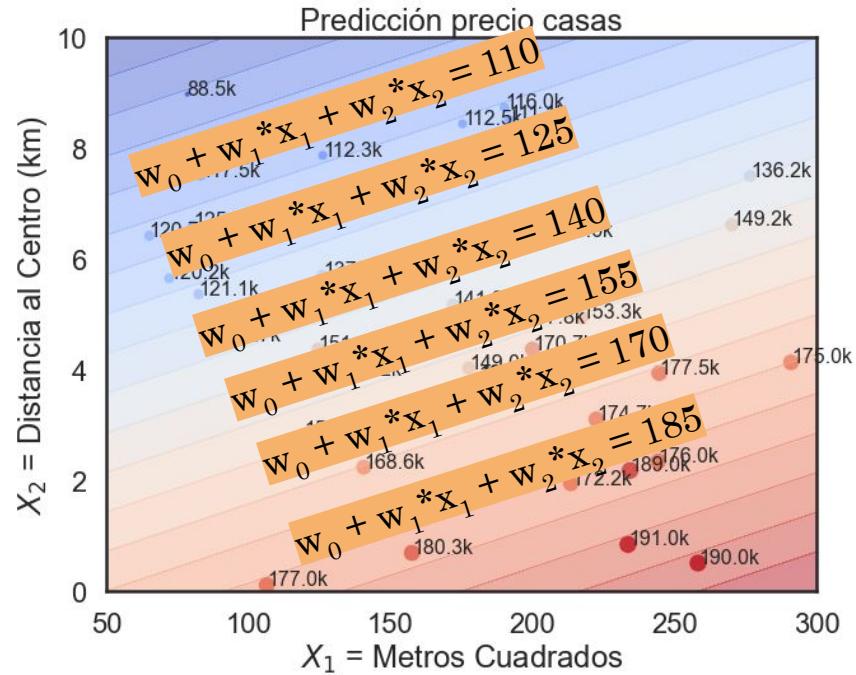
¿Cómo podremos convertir este modelo en un modelo para clasificación? Ej, "CARA" (clase positiva) vs "BARATA"



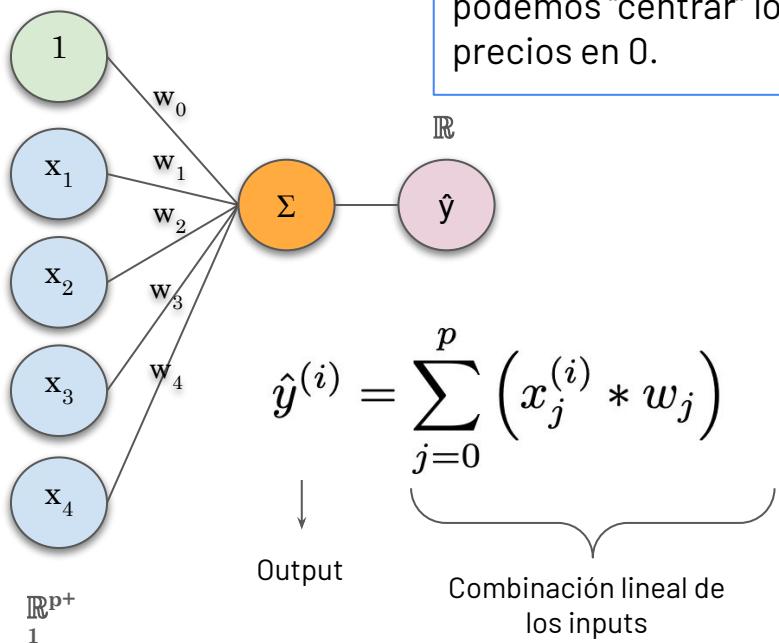
Convirtiendo a clasificación binaria



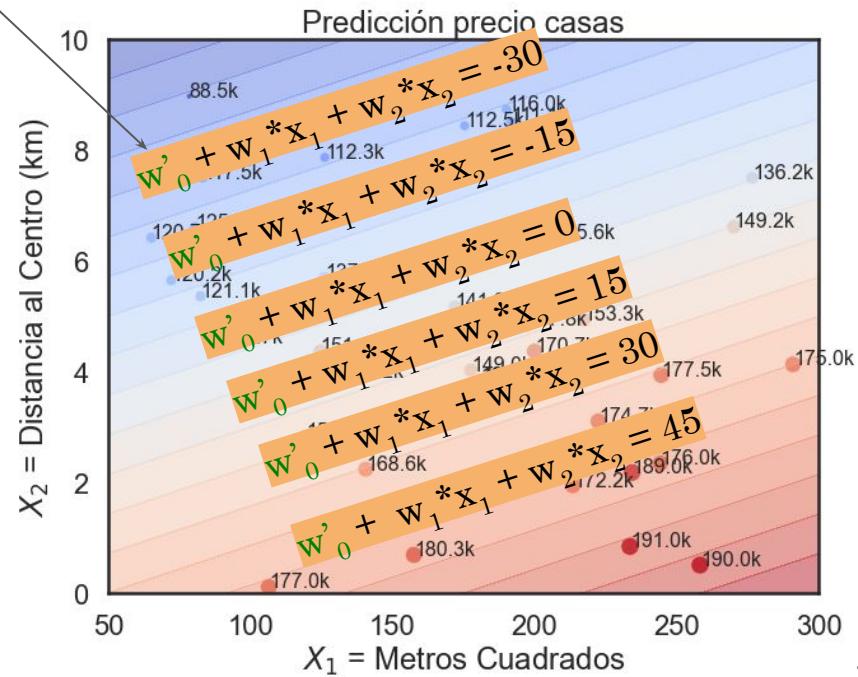
¿Cómo podremos convertir este modelo en un modelo para clasificación? Ej, "CARA" (clase positiva) vs "BARATA"



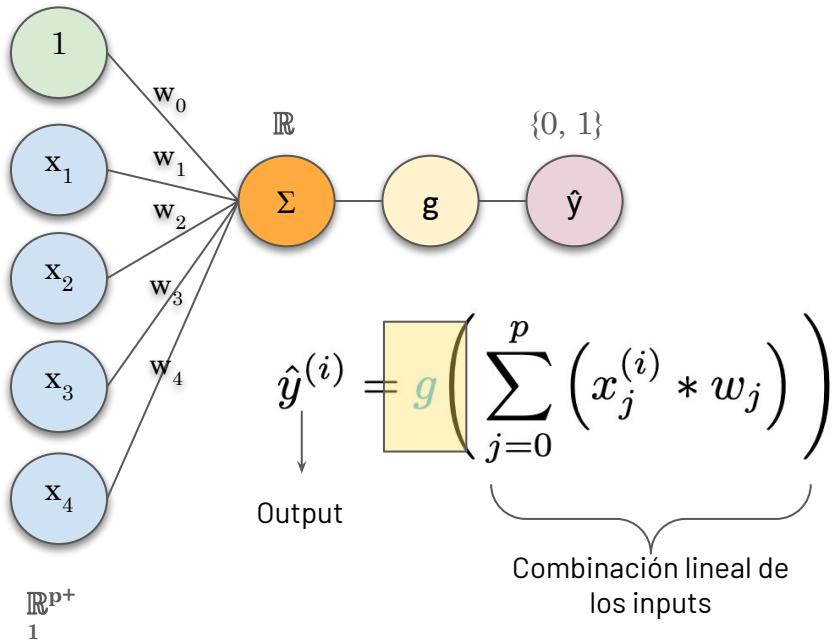
Convirtiendo a clasificación binaria



¿Cómo podremos convertir este modelo en un modelo para clasificación? Ej, "CARA" (clase positiva) vs "BARATA"

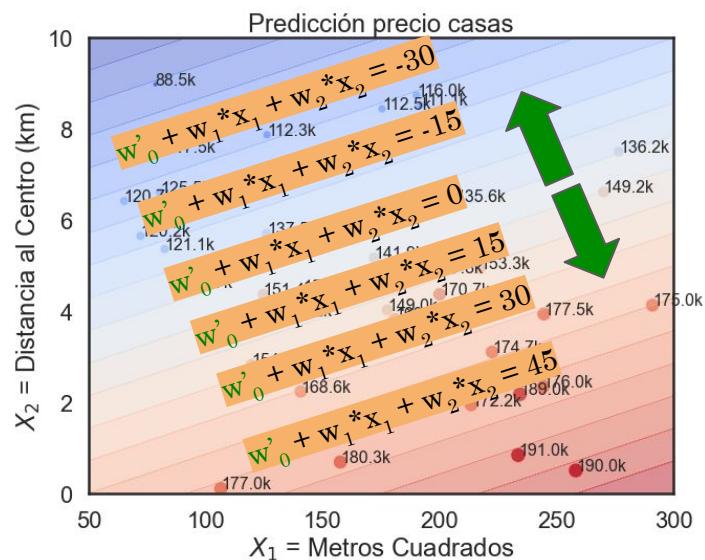
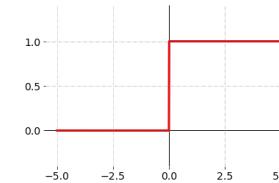


Convirtiendo a clasificación binaria

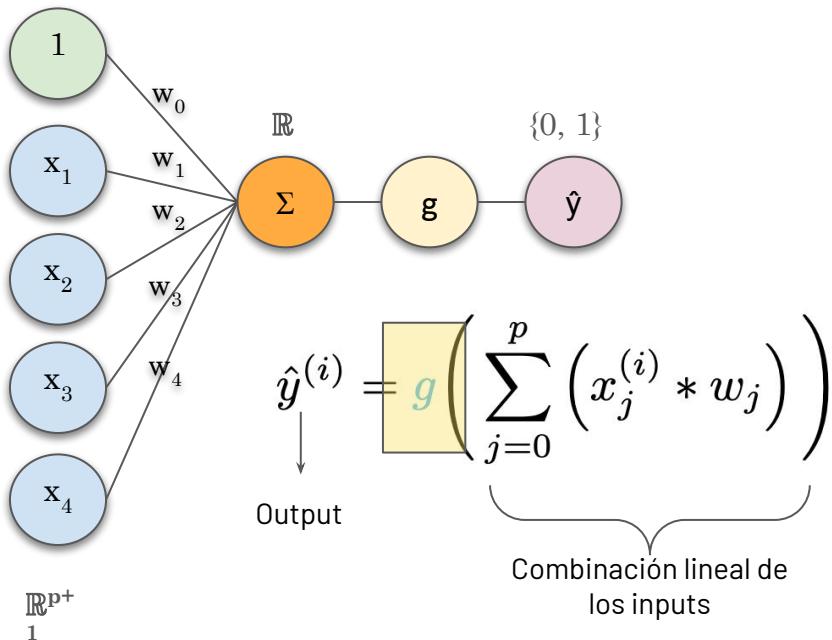


Una idea natural es agregar una función g de la siguiente manera:

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

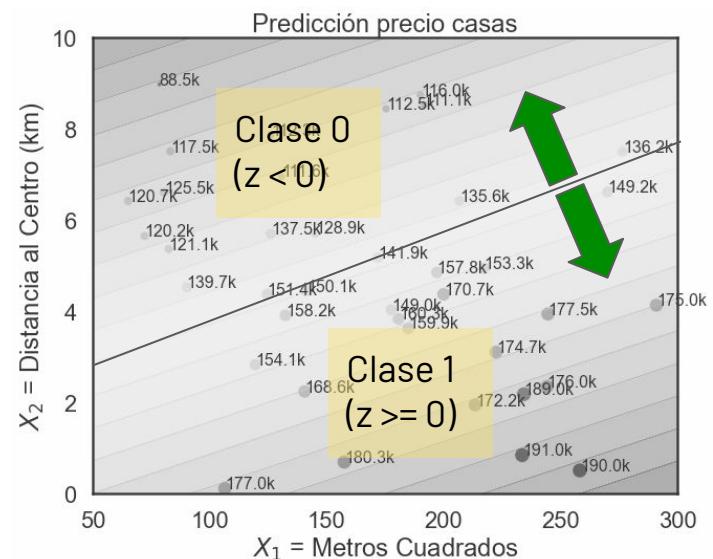
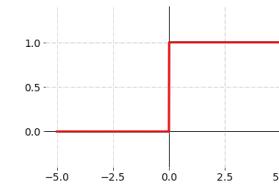


Convirtiendo el problema en un problema de clasificación

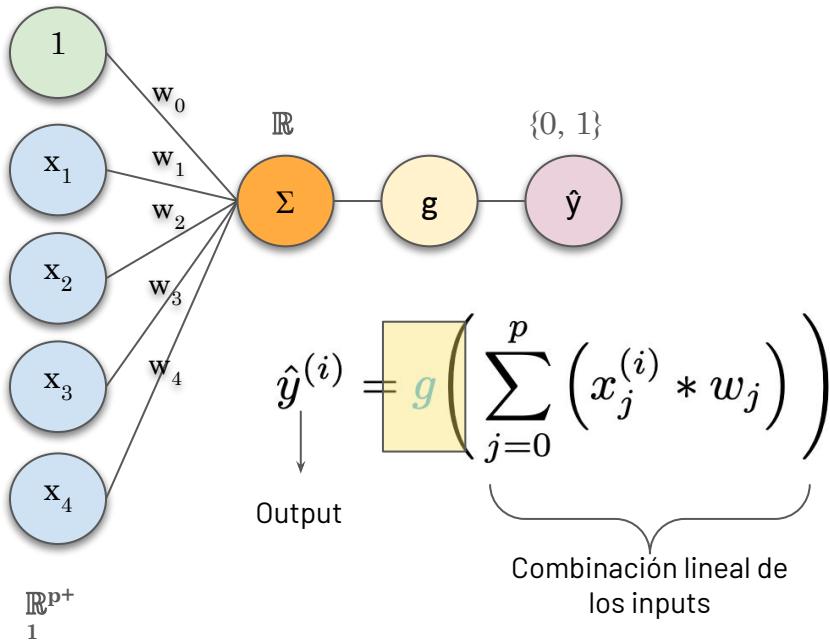


Una idea natural es agregar una función g de la siguiente manera:

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

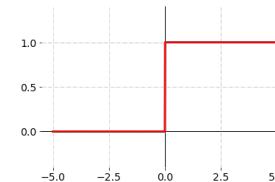


Convirtiendo el problema en un problema de clasificación



Una idea natural es agregar una función g de la siguiente manera:

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



¿Por qué esta función **no es la más** apropiada?

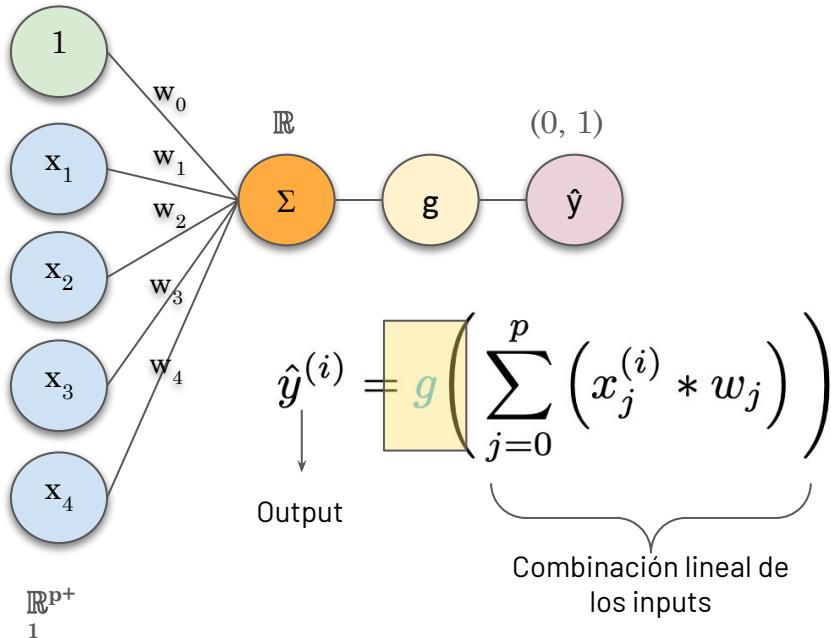
Recordar estuvimos midiendo cómo pequeños cambios aplicados a los pesos w afectan a la predicción final \hat{y} . Es decir, midiendo como:

$w + \Delta w$ se relaciona con $\hat{y} + \Delta \hat{y}$ y cómo esos cambios nos acercan al output esperado, a través de $\nabla_w \text{MSE}(w)$ lo cual debería permitir saber para qué dirección moverse.

El problema ahora es que cambios pequeños en w en general no afectarán a la nueva \hat{y} .

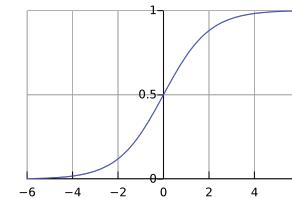
“Regresión” logística

(un modelo para clasificación)



Probamos con otra g de la siguiente pinta:

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



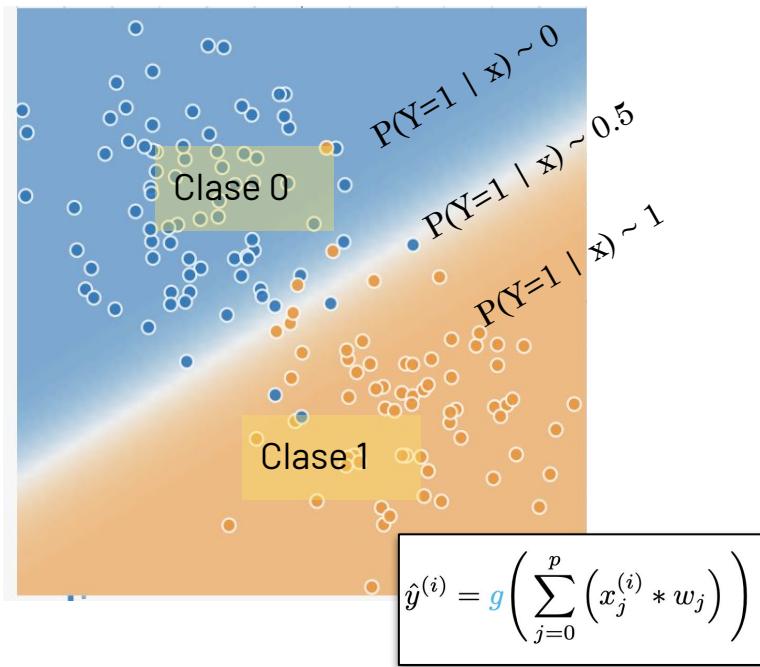
Esta función es conocida como **función sigmoidea devuelve**:

- Cerca de 1 cuando z es muy grande
En nuestro caso cuando $X^t w \gg 0$
- 0.5 cuando $z = 0$
 - En nuestro caso cuando $X^t w = 0$
- Cerca de 0 cuando z es muy chico
○ En nuestro caso cuando $X^t w \ll 0$

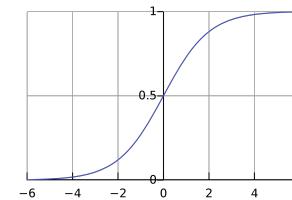
Podemos considerar: $\hat{y}^{(i)} = P(Y=1 | X=x^{(i)})$
(tomamos la salida como la **probabilidad** para la clase positiva)

"Regresión" logística

(un modelo para clasificación)



Probamos con otra g de la siguiente pinta:



Esta función es conocida como **función sigmoidea devuelve**:

- Cerca de 1 cuando z es muy grande
En nuestro caso cuando $X^t w \gg 0$
- 0.5 cuando $z = 0$
 - En nuestro caso cuando $X^t w = 0$
- Cerca de 0 cuando z es muy chico
○ En nuestro caso cuando $X^t w \ll 0$

Podemos considerar: $\hat{y}^{(i)} = P(Y=1 | X=x^{(i)})$
(tomamos la salida como la **probabilidad** para la clase positiva)

El problema ahora es, ¿cómo encontramos los W ?

Nota: Al ser un problema de clasificación, los targets serán valores {0, 1}. Ya no tiene sentido utilizar $MSE_{X,y}$

Abrimos paréntesis:
Entropía Cruzada Binaria

Entropía cruzada

(para caso binario)

Una métrica muy utilizada en problemas de clasificación que permite calcular el error en un dataset para el cual tenemos la **probabilidad estimada** de que cada instancia pertenezca a la **clase positiva**.

Veamos qué pinta tiene esta métrica de error en caso binario, es decir cuando:

$$P(Y=1 | X=x^{(i)}) = 1 - P(Y=0 | X=x^{(i)})$$

$$\text{Costo_BinCE}_{X,y} = \frac{1}{n} \sum_{i=1}^n \text{Binary-CE}^{(i)}$$

$$\text{Binary-CE}^{(i)} = -[y^{(i)} \log(\hat{h}^{\text{bin}}(x^{(i)})) + (1 - y^{(i)}) \log(1 - \hat{h}^{\text{bin}}(x^{(i)}))]$$

equivalentemente

$$\text{Binary-CE}^{(i)} = \begin{cases} -\log(\hat{P}(Y = 1 | X = x^{(i)})) & y^{(i)} = 1 \\ -\log(\hat{P}(Y = 0 | X = x^{(i)})) & y^{(i)} = 0 \end{cases}$$

logaritmo natural (base e)

$$P(Y=1 | X=x^{(i)})$$

$$y^{(i)}$$

$$\text{Binary-CE}^{(i)}$$

$$\text{Costo_BinCE}_{X,y} = <\text{completar}>$$

$$0.775$$

$$1$$

$$<\text{completar}>$$

$$0.116$$

$$0$$

$$<\text{completar}>$$

$$0.884$$

$$1$$

$$<\text{completar}>$$

$$0.744$$

$$0$$

$$<\text{completar}>$$

$$0.320$$

$$0$$

$$<\text{completar}>$$

¿Cuál es el mínimo valor que podemos obtener aquí? ¿Qué debe ocurrir para llegar a ese valor?

Cerramos paréntesis

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

"Regresión" logística

(un modelo para clasificación)

$$\hat{h}_w^{bin}(x^{(i)}) = \hat{P}(Y = 1 | X = x^{(i)}) = g\left(\sum_{j=0}^p (x_j^{(i)} * w_j)\right)$$

Como vimos anteriormente, es común utilizar estas métricas como **funciones de costo** a minimizar:

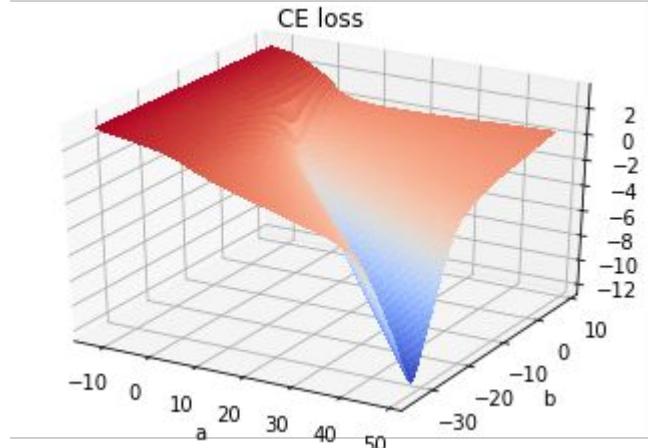
$$J_{X,y}(w) = \frac{1}{n} \sum_{i=1}^n \text{Binary_CE}^{(i)}(w) = \frac{1}{n} \sum_{i=1}^n -[y^{(i)} \log(\hat{h}_w^{bin}(x^{(i)})) + (1-y^{(i)}) \log(1-\hat{h}_w^{bin}(x^{(i)}))]$$

Y conociendo la forma de h , ya que seguimos en el caso regresión logística (binaria).

Podemos calcular analíticamente su gradiente (y por lo tanto utilizar descenso de gradiente):

$$\nabla_w J_{X,y}(w) = \frac{1}{n} \sum_{i=1}^n (\hat{h}_w^{bin}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

(ojo, esta forma es porque \hat{h}_w^{bin} es una reg. logística)



¡Otra función convexa!

"Regresión" logística (un modelo para clasificación)

Ejercicio: Lean y discutan este código en grupos, mientras tomo mate. Discutan cómo se transformaron las fórmulas de la slide anterior.

```
def fit(self, X_train, y_train):
    # Suponemos y_train es un arreglo de 0 y 1, ej: np.array([0,0,1,0,1,...])
    N = len(y_train)
    ones_column = np.ones(X_train.shape[0])
    X_train_ext = np.hstack((ones_column, X_train))

    def cost_binary_ce(w):
        probas = _pred(w) # Para pensar, qué son estas probas, ¿probabilidad de qué?
        return -(1 / N) * np.sum(y_train * log(probas) + (1 - y_train) * log(1 - probas))

    def grad_cost_binary_ce(w):
        probas = _pred(w)
        return 1 / N * (X_train_ext.T @ (probas - y_train))

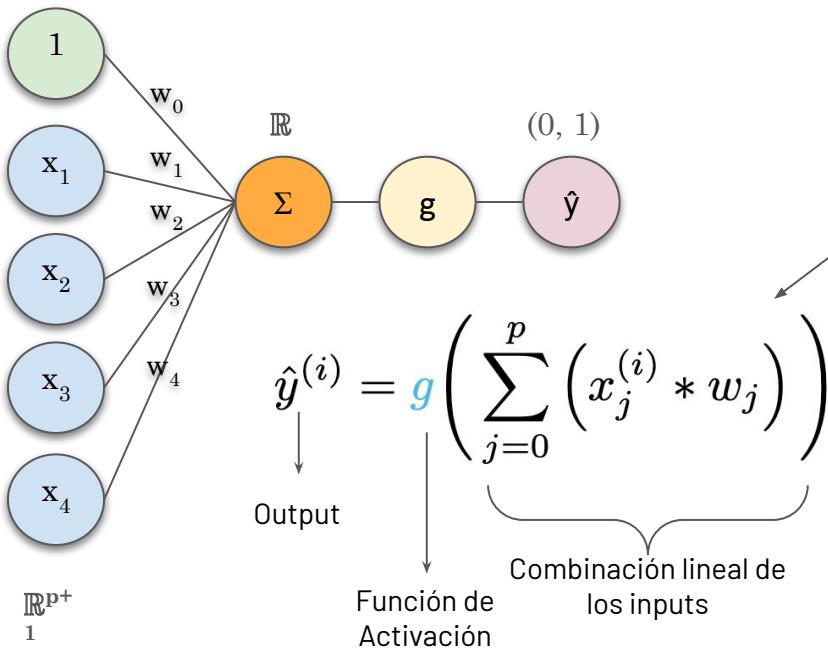
    def _pred(w):
        return _sigmoid(X_train_ext @ w)

    def _sigmoid(z):
        return 1 / (1 + np.exp(-z))

    zeros = np.zeros(X_train_ext.shape[1])
    self.w = descenso_gradiente(cost_binary_ce, grad_cost_binary_ce, z_init=zeros, alpha=0.01)
```

Redes neuronales

Redes neuronales simples

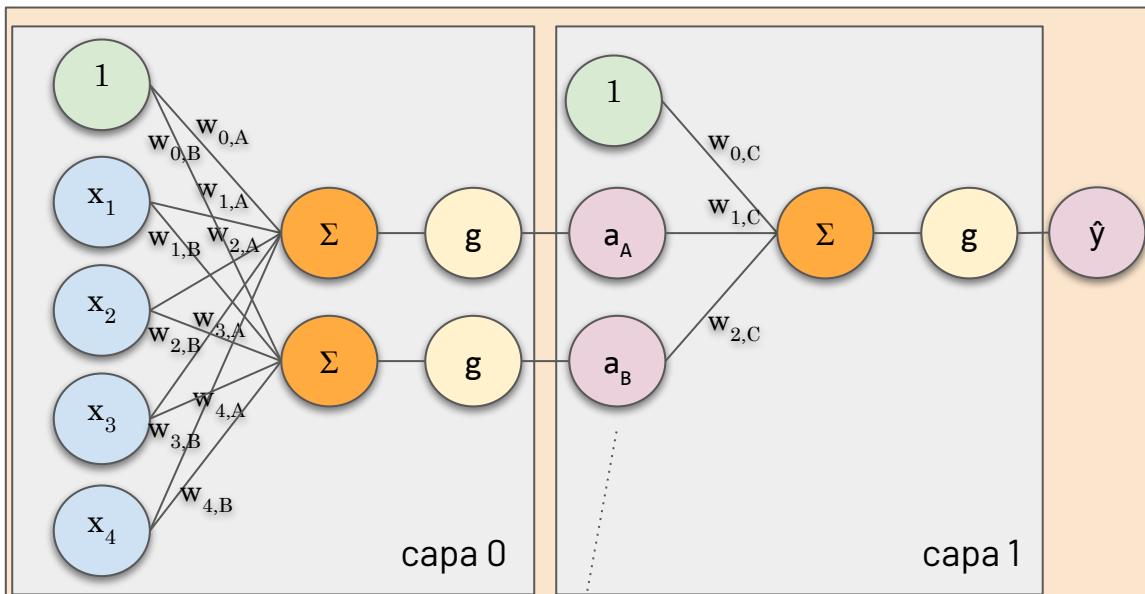


La red neuronal más sencilla es la conocida como "Perceptrón Simple" ([Rosenblatt, 1959](#)).

El Perceptrón Simple tiene la pinta que estuvimos analizando.

Tanto la regresión logística como la regresión lineal son instancias del perceptrón simple en donde **g** es la función **sigmoidea** (caso logística) o la función **identidad** (caso reg. lineal).

Redes neuronales multicapa

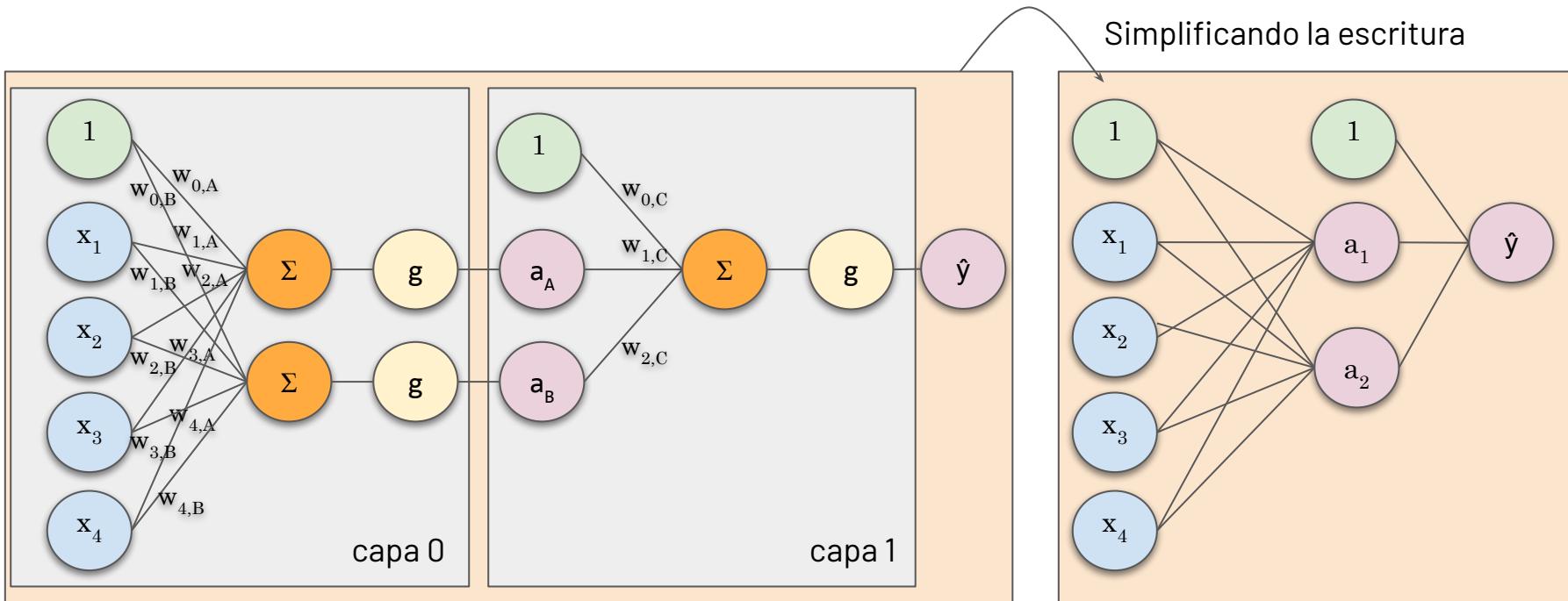


Una **red multicapa** genera **representaciones intermedias** que serán inputs de las capas sucesivas. En este ejemplo, la primera capa tiene dimensión 4 y la segunda capa, dimensión 2.

Ejercicio, ¿Cuánto valdrá \hat{y} si todo vale 0 salvo lo siguiente?:

- $x_4 = 1$
- $g(z) = z$ (*func act. identidad*)
- $w_{0,A} = -3$
- $w_{4,B} = 0.5$
- $w_{0,C} = 1$
- $w_{1,C} = 2$
- $w_{2,C} = 4$

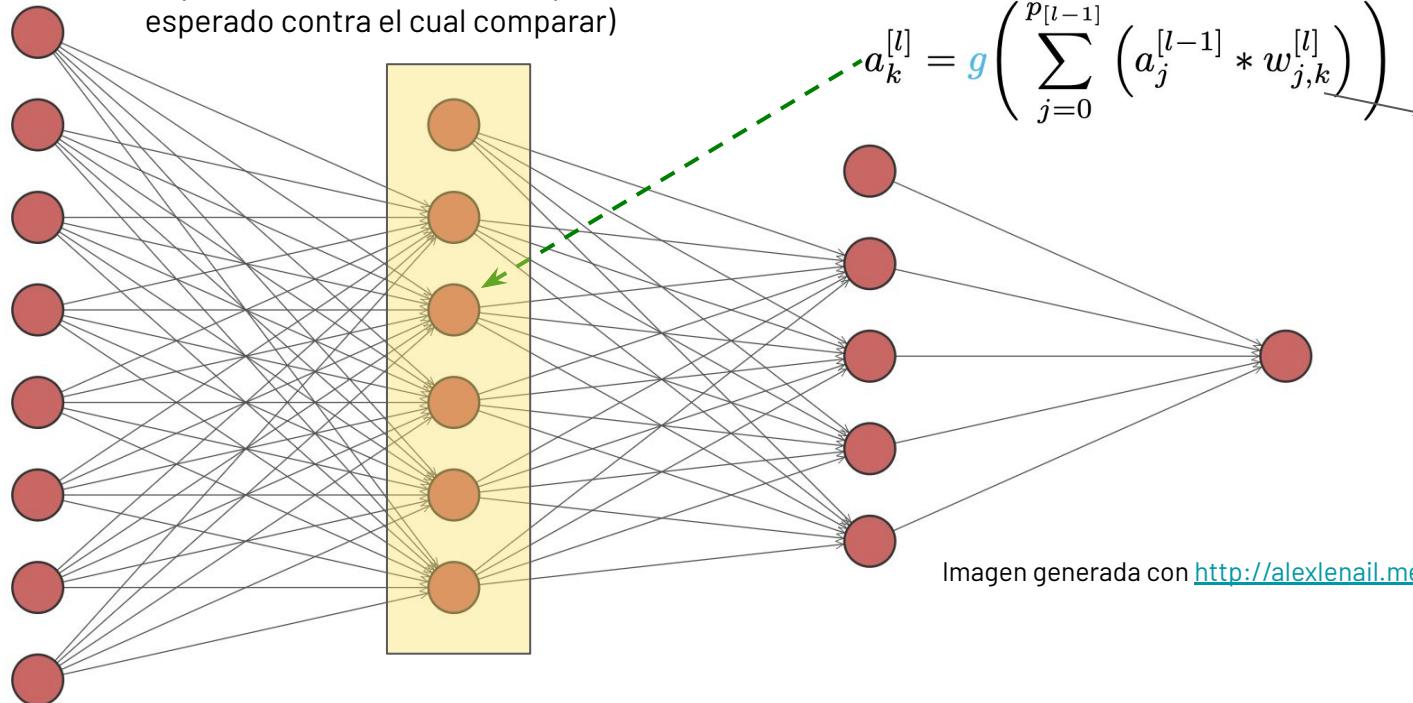
Redes neuronales multicapa



Una **red multicapa** genera **representaciones intermedias** que serán inputs de las capas sucesivas. En este ejemplo, la primera capa tiene dimensión 4 y la segunda capa, dimensión 2.

Redes neuronales multicapa

Capa "oculta" (no tenemos output esperado contra el cual comparar)



Capa de entrada
 $\in \mathbb{R}^8$ (input en \mathbb{R}^7)

Capa oculta
 $\in \mathbb{R}^6$

Capa oculta
 $\in \mathbb{R}^5$

Capa de salida
 $\in \mathbb{R}^1$

Valor de la neurona k en la capa l

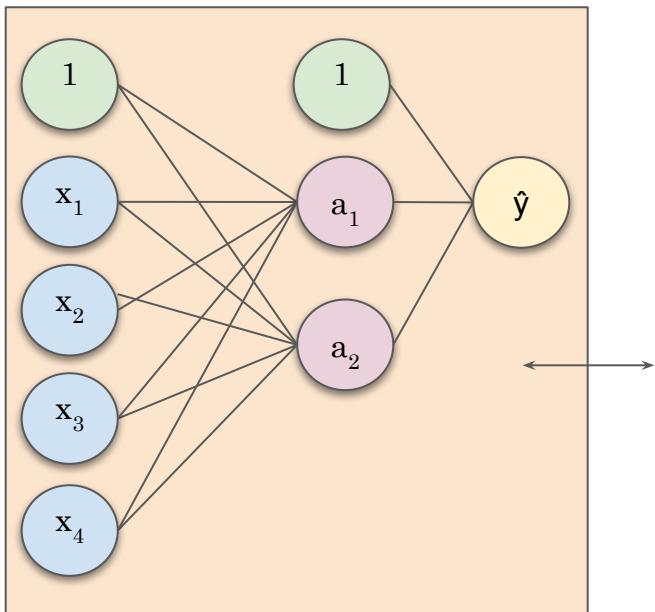
Cantidad de neuronas de la capa anterior

$$a_k^{[l]} = g\left(\sum_{j=0}^{p_{[l-1]}} (a_j^{[l-1]} * w_{j,k}^{[l]})\right)$$

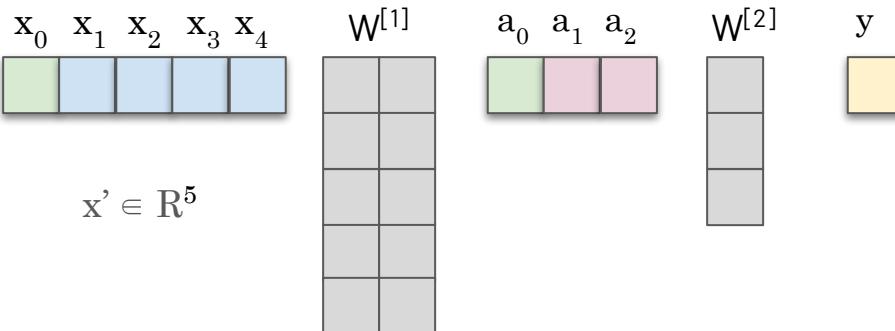
Peso que conecta la neurona j de la capa anterior con la neurona k de la capa actual

Imagen generada con <http://alexlenail.me/NN-SVG/index.html>

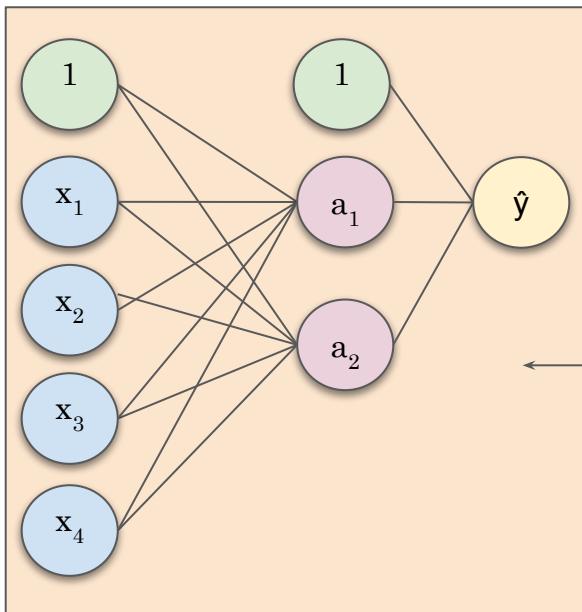
Redes neuronales multicapa



Para cada capa, ahora tenemos una Matriz de pesos $W^{[l]}$.



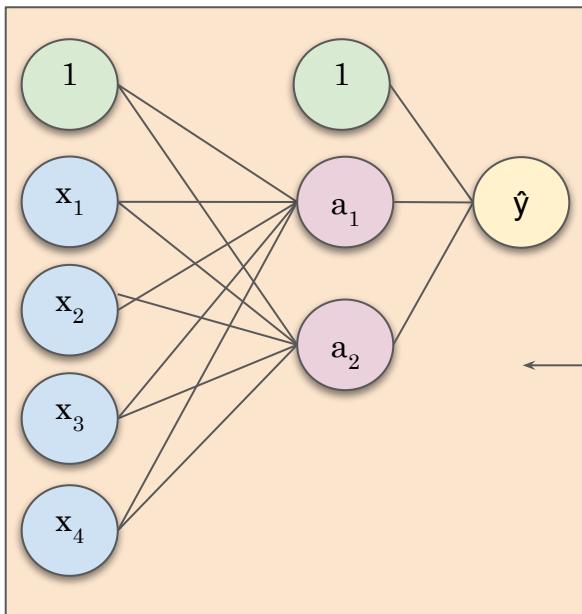
Redes neuronales multicapa



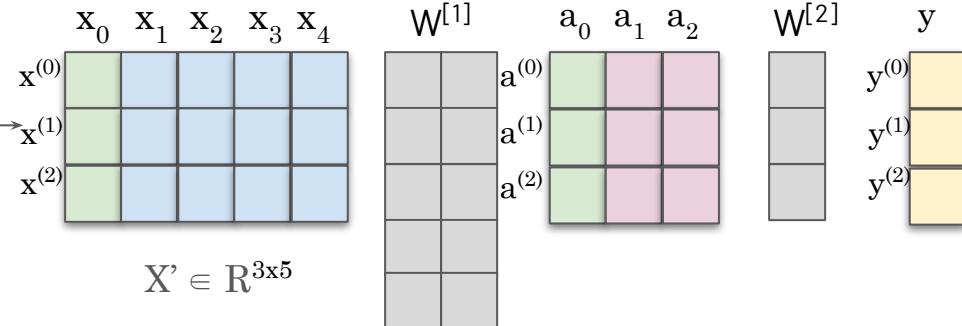
Para cada capa, ahora tenemos una Matriz de pesos $W^{[l]}$. Esta forma matricial también muestra que **agregar instancias** no cambia la dimensión de los pesos

$$\begin{array}{c} \begin{matrix} & x_0 & x_1 & x_2 & x_3 & x_4 \\ x^{(0)} & \text{green} & \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ x^{(1)} & \text{green} & \text{blue} & \text{blue} & \text{blue} & \text{blue} \\ x^{(2)} & \text{green} & \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{matrix} \\ X' \in \mathbb{R}^{3 \times 5} \end{array} \quad \begin{array}{c} W^{[1]} \\ \begin{matrix} & a_0 & a_1 & a_2 \\ a^{(0)} & \text{grey} & \text{grey} & \text{grey} \\ a^{(1)} & \text{grey} & \text{grey} & \text{grey} \\ a^{(2)} & \text{grey} & \text{grey} & \text{grey} \end{matrix} \end{array} \quad \begin{array}{c} W^{[2]} \\ \begin{matrix} & y \\ y^{(0)} & \text{yellow} \\ y^{(1)} & \text{yellow} \\ y^{(2)} & \text{yellow} \end{matrix} \end{array}$$

Redes neuronales multicapa



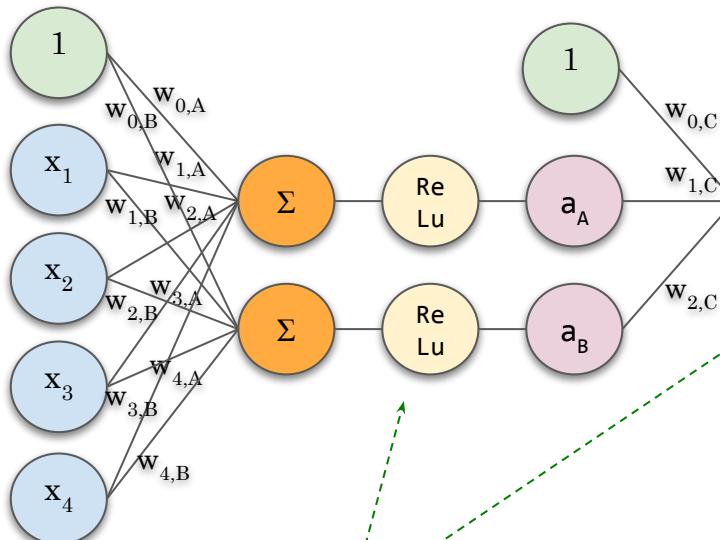
Para cada capa, ahora tenemos una Matriz de pesos $W^{[l]}$. Esta forma matricial también muestra que **agregar instancias** no cambia la dimensión de los pesos



Ejercicio: rehacer estos dos diagramas y completar las dimensiones faltantes para el caso de un dataset X que tiene 4 instancias con 2 atributos (mts^2 y distancia obelisco) para predecir el precio. $X' \in \mathbb{R}^{? \times ?}$, $W^{[1]} \in \mathbb{R}^{? \times 2}$, $A^{[1]} \in \mathbb{R}^{? \times ?}$, $W^{[2]} \in \mathbb{R}^{? \times 1}$, $A^{[2]} \in \mathbb{R}^{? \times ?}$, $W^{[3]} \in \mathbb{R}^{? \times 1}$, $Y \in \mathbb{R}^{? \times 1}$

Redes neuronales multicapa

Función de activación



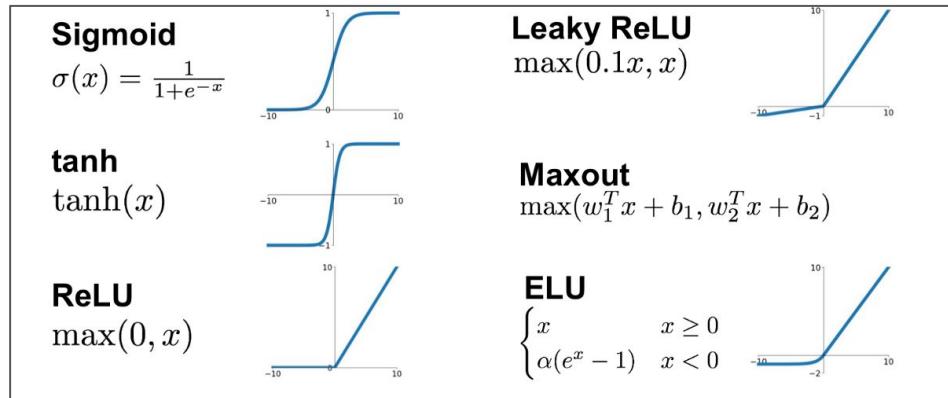
En general, se utiliza la misma función de activación en todas las capas intermedias (en este caso sólo una capa intermedia). Pero la de la última capa depende del tipo de problema.

Además de tener una función no lineal como última operación en el caso de clasificación, es común (necesario) **introducir no linealidades** entre las distintas capas del modelo (tanto para clasificación como para regresión).

Ver

<https://dashee87.github.io/deep%20learning/visuallying-activation-functions-in-neural-networks/>

Funciones de activación



Fuente <https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092>

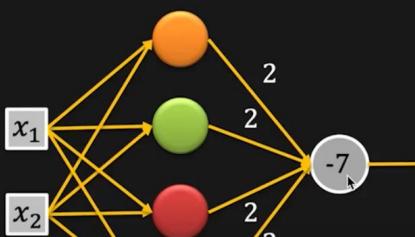
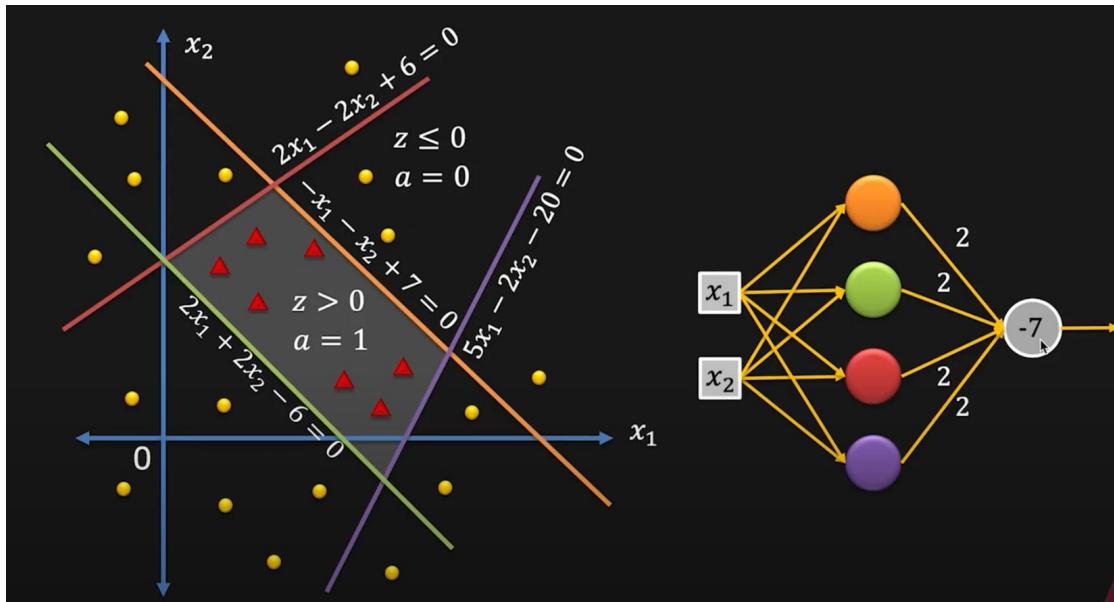
Redes neuronales multicapa

[PDF] Kolmogorov's mapping neural network existence theorem

R Hecht-Nielsen - Proceedings of the international conference on ..., 1987 - cs.uwaterloo.ca

An improved version of Kolmogorov's powerful 1957 theorem concerning the representation of arbitrary continuous functions from the n -dimensional cube to the real numbers in terms of one dimensional continuous functions is reinterpreted to yield an existence theorem for mapping neural networks.

☆ Guardar ⌂ Citar Citado por 2088 Artículos relacionados Las 4 versiones ☰



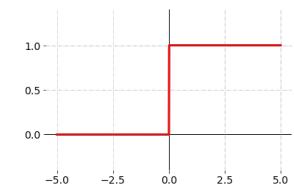
Veamos <https://www.youtube.com/watch?v=torNuKNLwBE>
y <https://playground.tensorflow.org/>

¿Para qué multicapa?

Veamos cómo una red neuronal multicapa con funciones de activación no lineales permiten capturar patrones complejos en el espacio de atributos.

(este ejemplo usa la siguiente g)

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



Redes neuronales profundas

cuando una red tiene muchas capas (> 2 en general) la llamamos "red profunda"

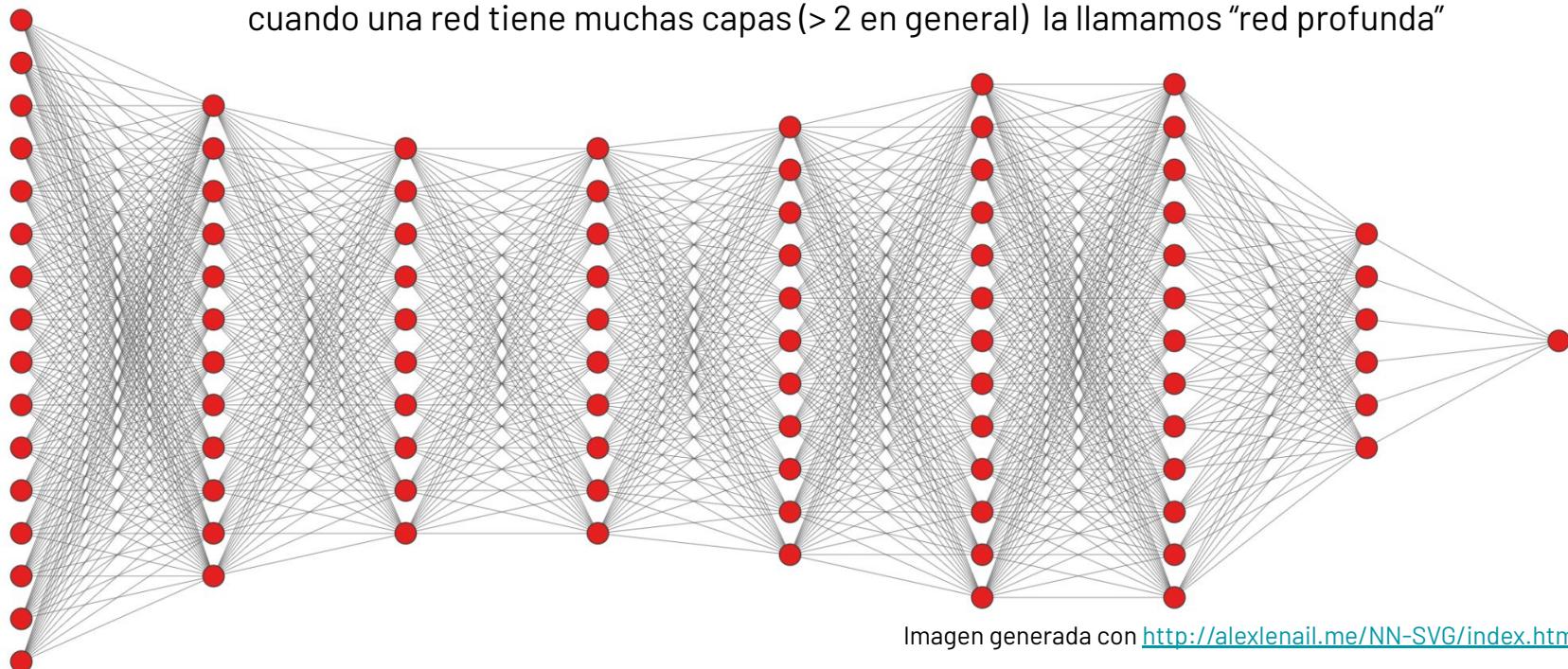


Imagen generada con <http://alexlenail.me/NN-SVG/index.html>

Entrenamiento

$$\nabla_{W_s} J_{X,y}(W_s)$$

Una red neuronal profunda define un modelo de la pinta:

$$h_{W_s}(x^{(i)}) = h^{[L]}(\dots h^{[2]}(h^{[1]}(x^{(i)}))) \text{ La función de costo: } J_{X,y}(W_s) = 1/n * \sum_{X,y} (\text{Loss}(h_{W_s}(x^{(i)}), y^{(i)}))$$

Necesitamos $\nabla_{W_s} J_{X,y}(W_s)$ para poder aplicar descenso por gradiente.

Opciones:

- **Derivada analítica** (derivar a mano, escribir el código) – una pesadilla.
- **Diferenciación numérica** (diferencias finitas, metnum) – puede introducir errores de redondeo en el proceso de discretización y cancelación. Además, lento.
- **Diferenciación simbólica** (obtener la expresión matemática que describe a la derivada automáticamente) – explota rápidamente por el tamaño de la expresión generada (“expression swell”). Además, lento.
- **Diferenciación automática**. Construcción de un grafo computacional + definir reglas para primitivas + regla de la cadena con programación dinámica.

Diferenciación simbólica

$64x(1-x)(1-2x)^2$	$128x(1-x)(-8+16x)(1-2x)^2(1 - (1 - 8x + 8x^2)^2$
$(1-8x+8x^2)^2$	$8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x)^2(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$
$f(x)$	$f'(x)$

Backpropagation

Backpropagation: Proceso automático para
calcular gradientes a partir del concepto de "la
regla de la cadena".

Vamos con un ejemplo. Imaginense que tenemos
la siguiente función.

$$f(w_0, w_1, x_1, w_2, x_2) = \frac{1}{1 + e^{-(w_0 + w_1 * x_1 + w_2 * x_2)}}$$

Y que estamos interesados en conocer su
gradiente, **en un punto**. Ej, en **w0=-3, w1=2, x1=-1,
w2=-3, x2=-2.**

$$\nabla_w(f(w; x)) = \begin{bmatrix} \nabla_{w_0}(f(w; x)) \\ \nabla_{w_1}(f(w; x)) \\ \nabla_{w_2}(f(w; x)) \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial w_0} \\ \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \end{bmatrix}$$

Nota₁: esta función se parece mucho a una sigmoidea
aplicada a un perceptrón simple con una entrada de
dimensión 2.

Nota₂: En general no nos interesa este gradiente, nos
interesa:

$$\nabla_{W_s} J_{X,y}(W_s) = \nabla_{W_s} 1/n * \sum_{X,y} (\text{Loss}(h_{W_s}(x^{(i)}), y^{(i)}))$$

pero esto un ejemplo :).

Primero: Repasemos regla de la cadena

la derivada de una función compuesta $(f \circ g)(p)$ se puede descomponer el “encadenamiento” de multiplicaciones de la pinta:

$$\frac{\partial f(g(p))}{\partial p} = \frac{\partial f(u)}{\partial u} * \frac{\partial g(p)}{\partial p} \quad \text{en donde } g(p) = u$$

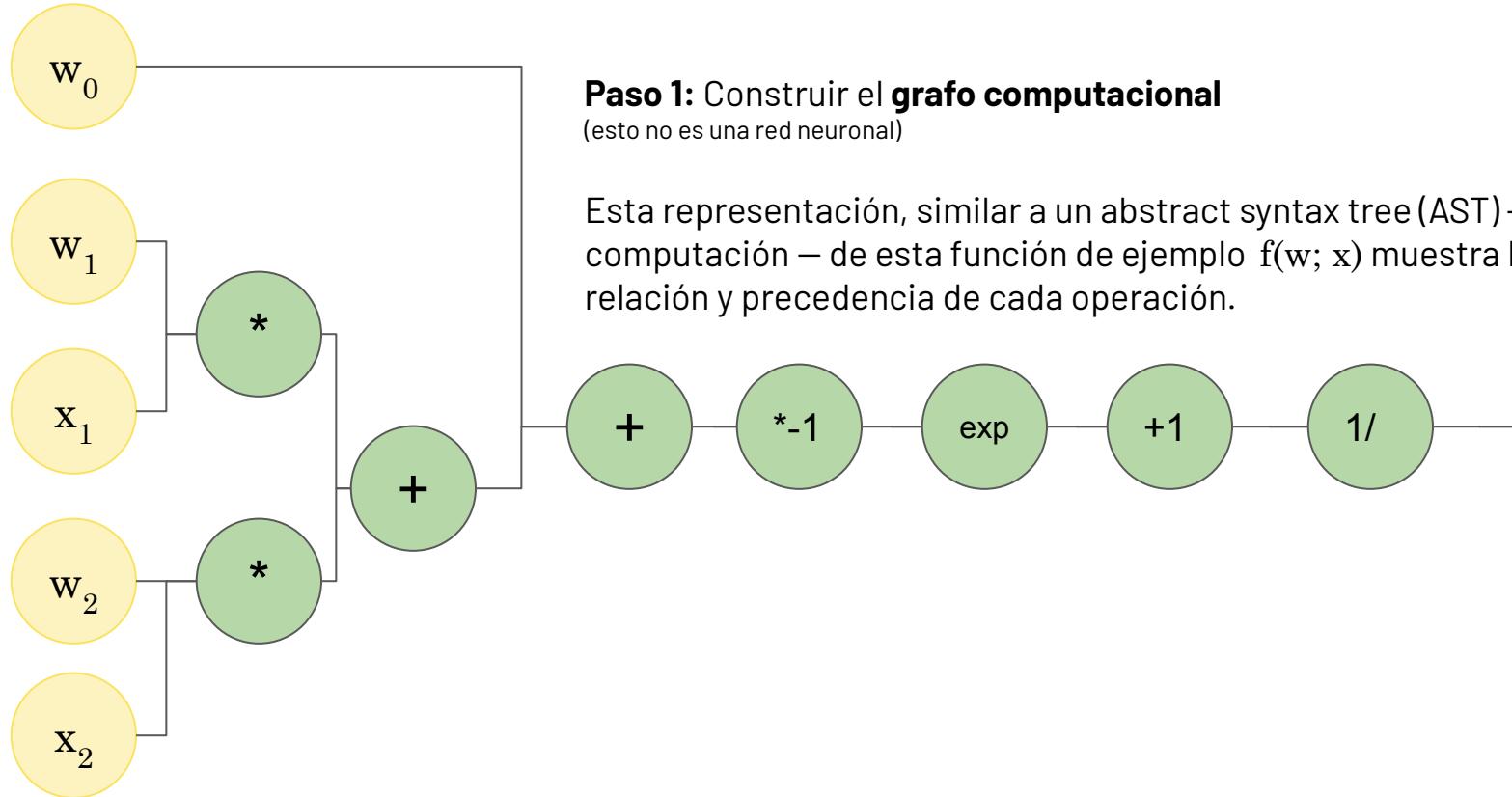
Ejemplo, cuál es la derivada de $e^{(\sin(p))}$ respecto a p ?

$$\frac{\partial e^{\sin(p)}}{\partial p} = \frac{\partial e^u}{\partial u} * \frac{\partial \sin(p)}{\partial p} = (e^u) * \cos(p) = (e^{\sin(p)}) * \cos(p)$$

\nearrow
 $u = \sin(p)$

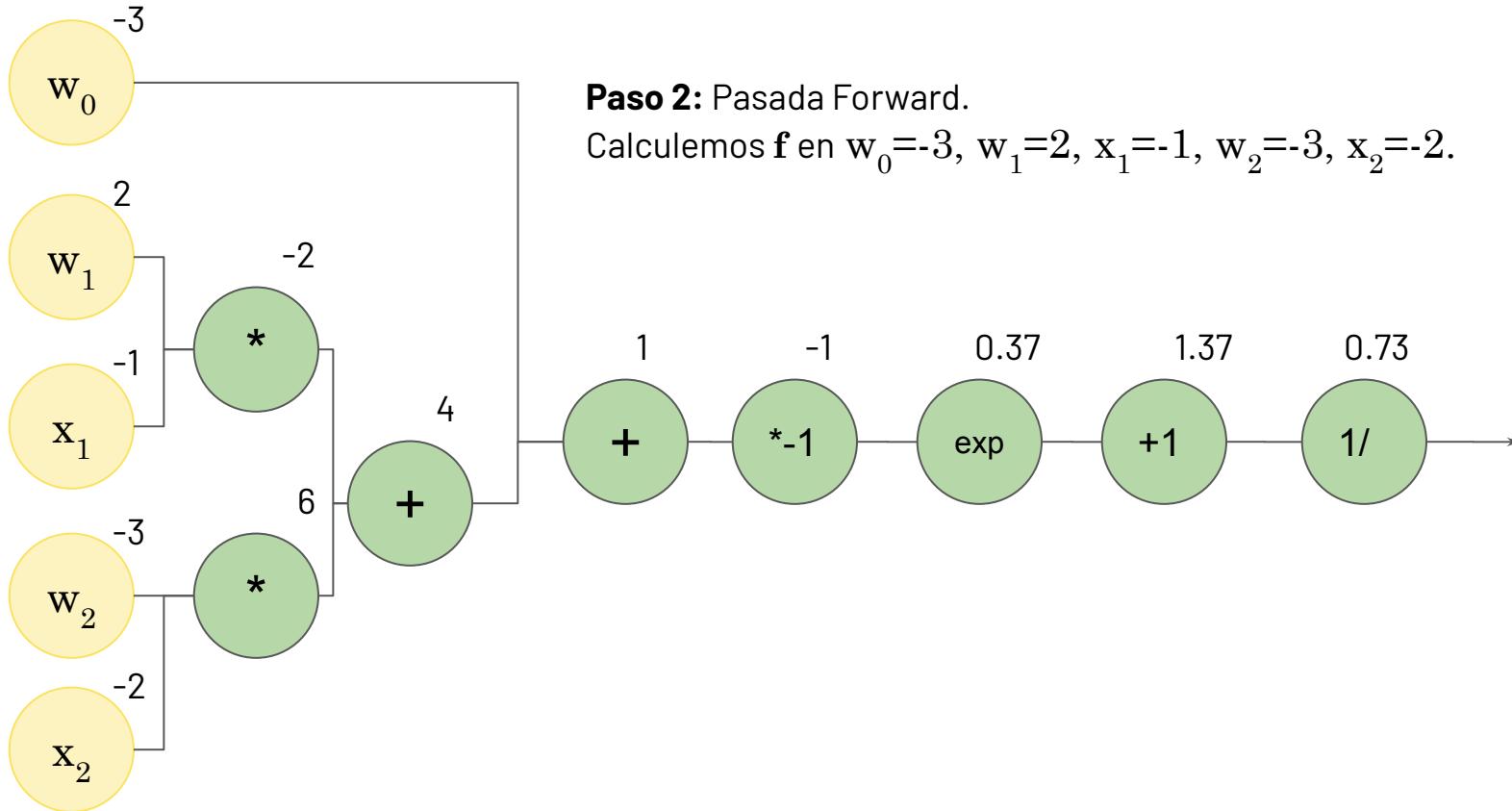
$$f(w_0, w_1, x_1, w_2, x_2) = \frac{1}{1 + e^{-(w_0 + w_1 * x_1 + w_2 * x_2)}}$$

Dif. Automática Reversa



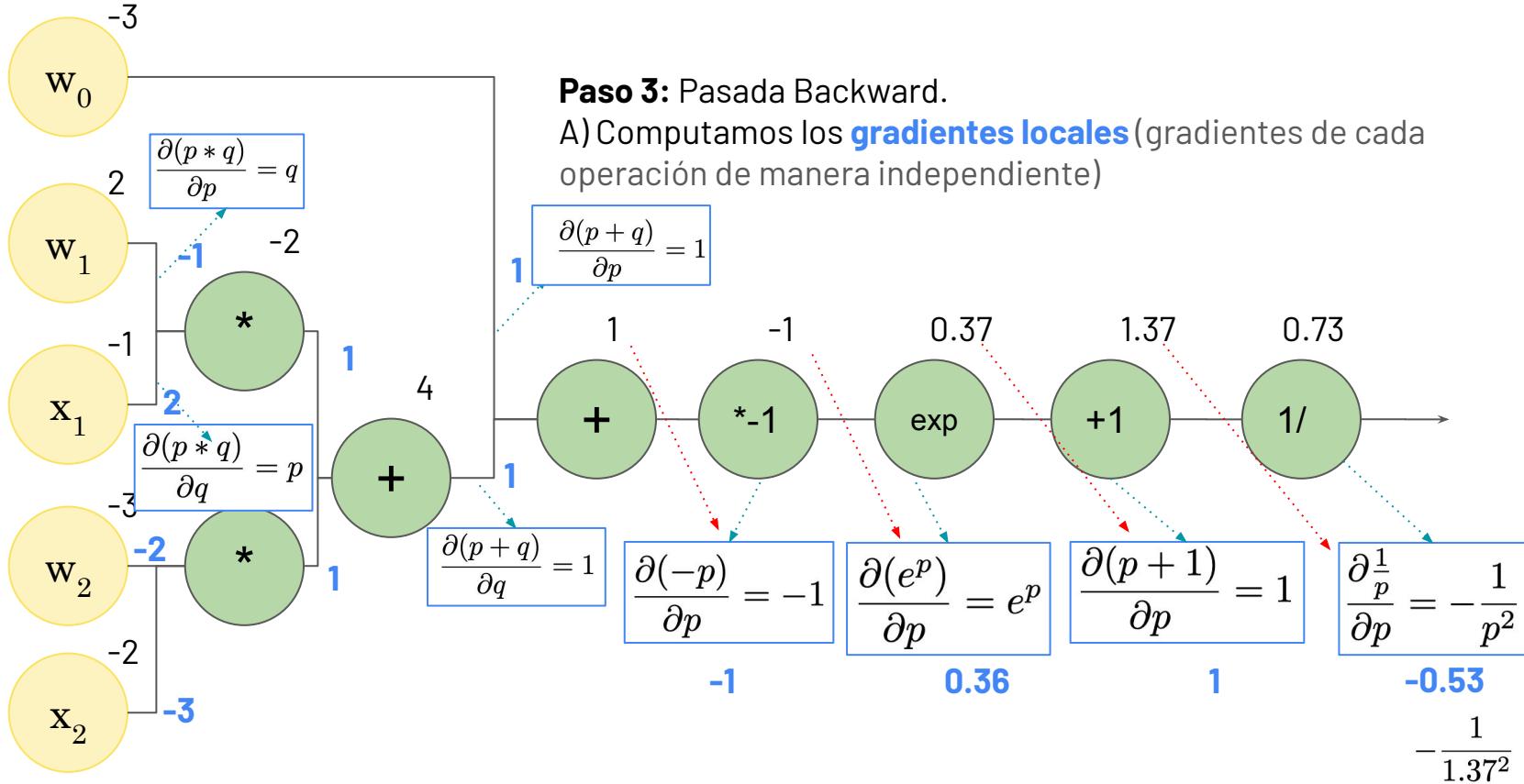
Dif. Automática Reversa

$$f(w_0, w_1, x_1, w_2, x_2) = \frac{1}{1 + e^{-(w_0 + w_1 * x_1 + w_2 * x_2)}}$$



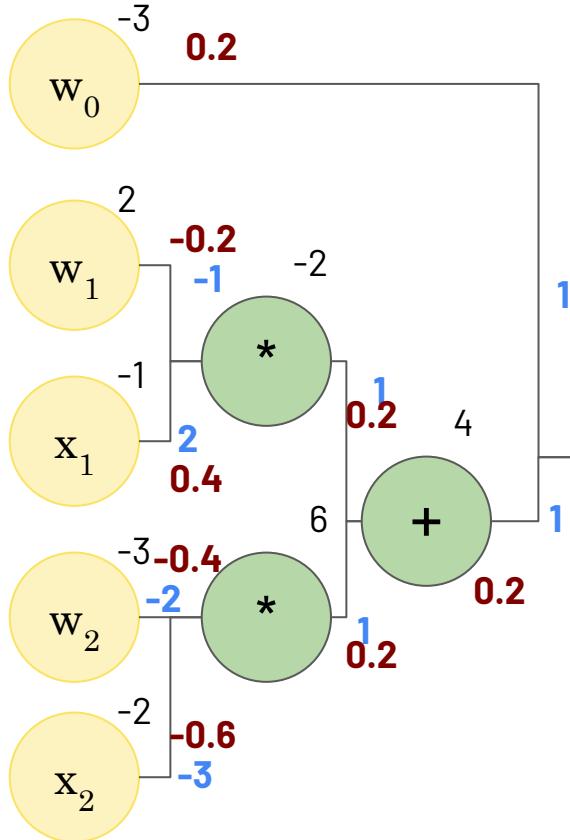
Dif. Automática Reversa

$$f(w_0, w_1, x_1, w_2, x_2) = \frac{1}{1 + e^{-(w_0 + w_1 * x_1 + w_2 * x_2)}}$$



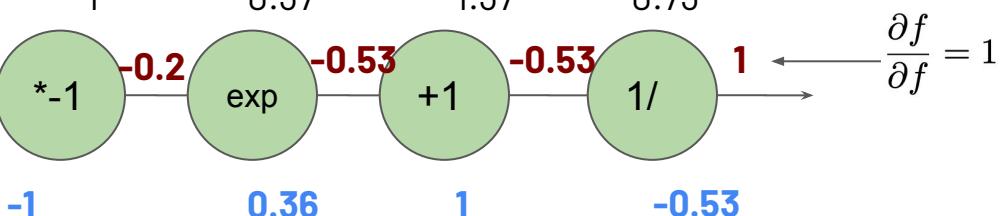
Dif. Automática Reversa

$$f(w_0, w_1, x_1, w_2, x_2) = \frac{1}{1 + e^{-(w_0 + w_1 * x_1 + w_2 * x_2)}}$$



Paso 3: Pasada Backward.

B) Cálculo del **gradiente** usando la regla de la cadena (empezamos desde 1 y vamos multiplicando por los **gradientes locales**)



Resulta entonces
 $\nabla_w f = (0.2, -0.2, -0.4)$

De paso también calculamos
 $\nabla_x f = (0.4, -0.6)$

Por suerte.. los frameworks hacen este trabajo

Pytorch

```
import torch

x = torch.tensor([1., -1., -2.]) # Tensor = vector en este ejemplo
w = torch.tensor([-3., 2., -3.], requires_grad=True) # pesos
z = x @ w
a = 1 / (1 + torch.exp(-z))
# Ultimas dos lineas equivalentes a
# a = torch.sigmoid(x @ w)

a.backward()
print("Gradiente con respecto a w:", w.grad)
print("Gradiente con respecto a x:", x.grad) # "No me lo da, no le
pedí que lo calcule"
```

```
Gradiente con respecto a w: tensor([ 0.1966, -0.1966, -0.3932])
Gradiente con respecto a x: None
```

(da lo mismo, errores de redondeo en el ejemplo)

¿Cómo definir funciones
"base" propias?

```
class Exp(torch.autograd.Function):
    @staticmethod
    def forward(ctx, i):
        result = i.exp()
        ctx.save_for_backward(result)
        return result

    @staticmethod
    def backward(ctx, grad_output):
        result, = ctx.saved_tensors
        return grad_output * result

# Call the function
Exp.apply(torch.tensor(0.5, requires_grad=True))
# Outputs: tensor(1.6487, grad_fn=<ExpBackward>)
```

<https://pytorch.org/blog/computational-graphs-constructed-in-pytorch/>

Gradiente de la función de costo final

Resumiendo.

Tenemos una función de costo:

$$J_{X,y}(W_s) = \frac{1}{n} * \sum_{X,y} (\text{Loss}(h_{W_s}(x^{(i)}), y^{(i)})) \text{ (esto aplica a regresión y clasificación)}$$

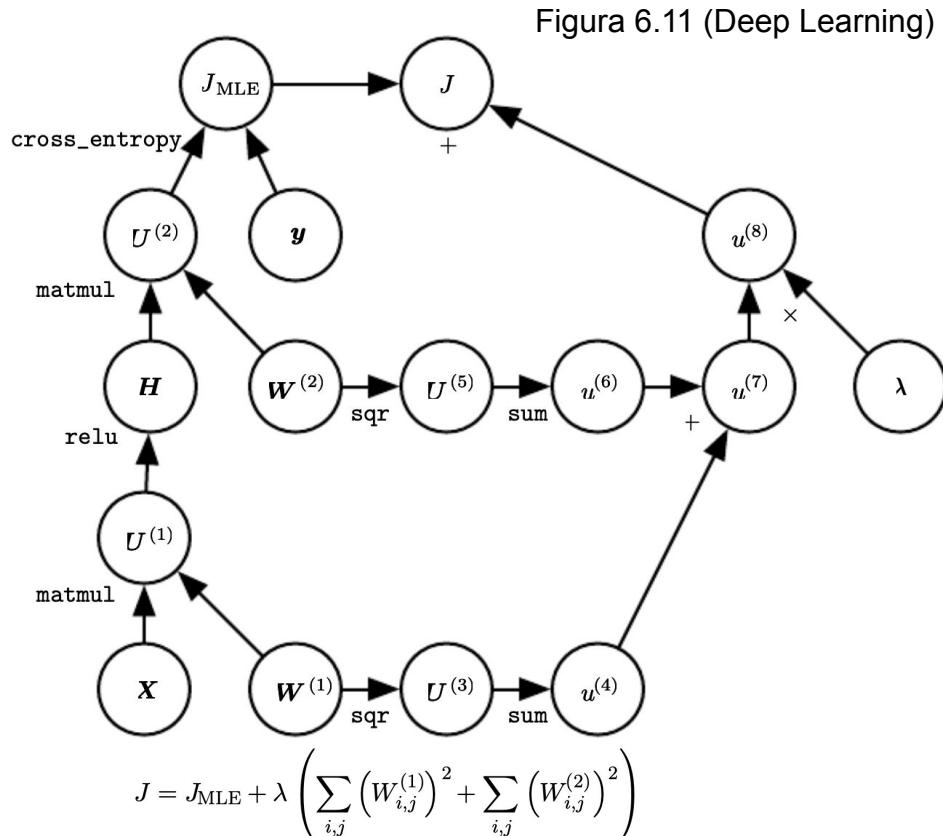
Aplicamos **descenso por el gradiente** para entrenarla.

Pero no conocemos una forma cerrada para su gradiente:

$$\nabla_{W_s} J_{X,y}(W_s) = \nabla_{W_s} \frac{1}{n} * \sum_{X,y} (\text{Loss}(h_{W_s}(x^{(i)}), y^{(i)}))$$

Usamos backpropagation para calcular el gradiente en un punto dado.

- 1) Se crea un grafo computacional que define el cómputo de la función
- 2) Se hace una pasada forward: evaluar el grafo en el punto particular.
- 3) Se hace una pasada backward en donde se aplica la regla de la cadena.



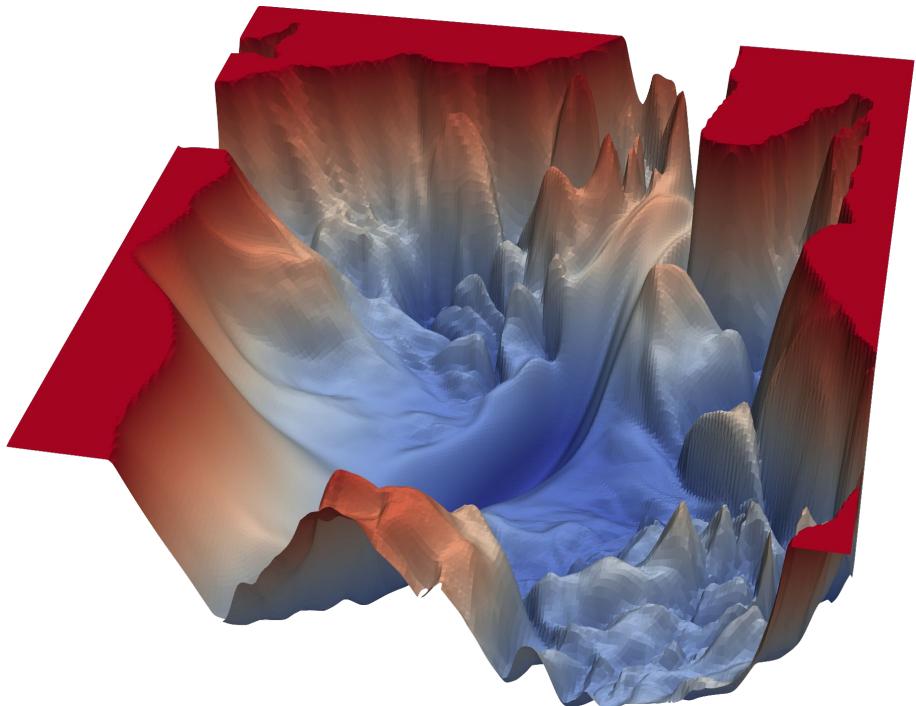
Descenso de Gradiente en NN

Descenso de gradiente **no garantiza encontrar el mínimo global en funciones no convexas.**

Sin embargo, **para redes neuronales “grandes”**, la mayoría de los **mínimos locales son similares** y presentan performance similar en los datasets de tests.

La probabilidad de **encontrar mínimos locales malos decrece con el tamaño de la red.**

Focalizarse demasiado en la búsqueda del mínimo global en el dataset de entrenamiento no es muy útil en la práctica ya que solemos caer en **sobreajuste**.



Fuente: <https://www.cs.umd.edu/~tomq/projects/landscapes/?ref=jeremyjordan.me>

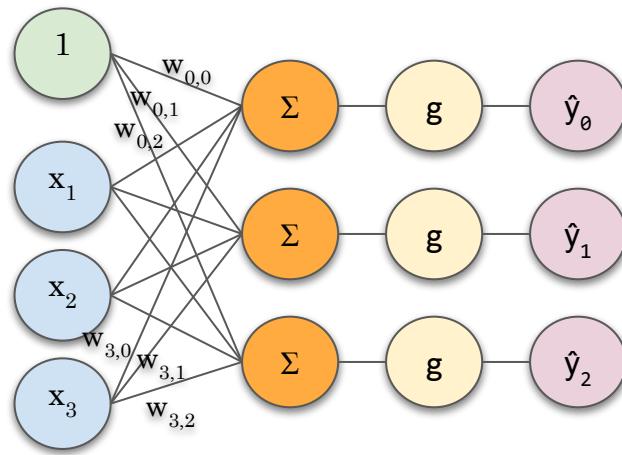
Ver también <https://losslandscape.com/> (arte, tremendo)

Redes Neuronales Multi Output

Redes Neuronales Multi Output

Imaginen que ahora queremos clasificar una instancia entre 3 clases distintas (ej: "perro", "gato", "pato").

Podríamos usar la siguiente arquitectura:

 \mathbb{R}^{p+1} \mathbb{R}^C \mathbb{R}^C

Es decir, podemos utilizarlo para obtener:

$$\hat{y}_0 = P(Y = \text{clase 0} \mid X = x^{(i)})$$

$$\hat{y}_1 = P(Y = \text{clase 1} \mid X = x^{(i)})$$

$$\hat{y}_2 = P(Y = \text{clase 2} \mid X = x^{(i)})$$

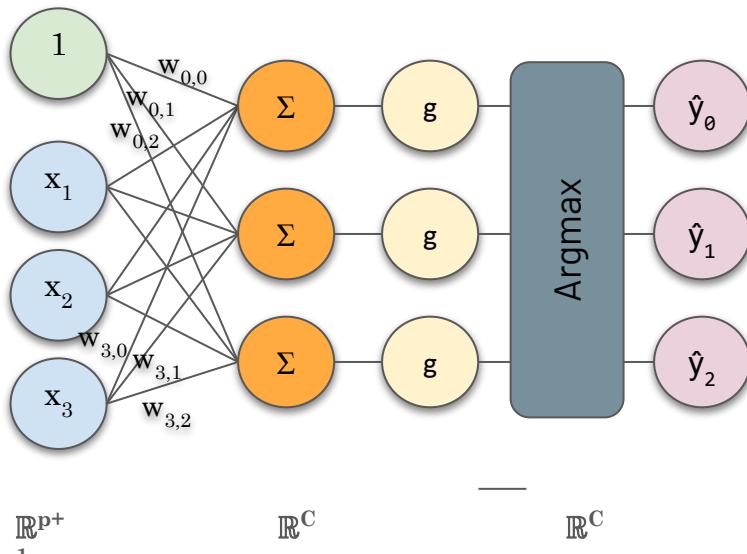
Vimos que si $g(z) = \text{sigmoidea}(z)$ obtenemos scores entre 0 y 1. ¿Estaría bien usarla en este caso?

¿Cómo elegimos la clase final?

Redes Neuronales Multi-Output

Imaginen que ahora queremos clasificar una instancia entre 3 clases distintas (ej: "perro", "gato", "pato").

Podríamos usar la siguiente arquitectura:



Es decir, podemos utilizarlo para obtener:

$$\begin{aligned}\hat{y}_0 &= P(Y = \text{clase 0} \mid X = x^{(i)}) \\ \hat{y}_1 &= P(Y = \text{clase 1} \mid X = x^{(i)}) \\ \hat{y}_2 &= P(Y = \text{clase 2} \mid X = x^{(i)})\end{aligned}$$

Una opción sería usar

- $g(z) = x$
- o quizás $g(z) = \text{sigmoidea}(z)$

y luego usar $\text{arg_max}(\hat{y})$.

Problemas:

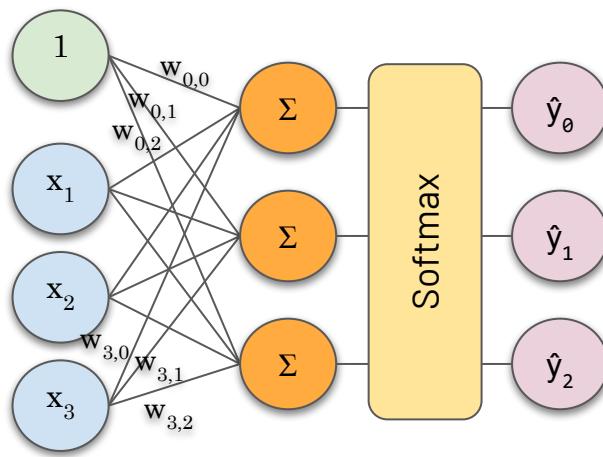
- 1) Perdemos las probabilidades para las distintas clases (no suman 1).
- 2) $\text{arg_max}(\hat{y})$ no es diferenciable, cuando querramos calcular la loss, no se propaga el gradiente!

Redes Neuronales Multi-Output

Softmax

Imaginen que ahora queremos clasificar una instancia entre 3 clases distintas (ej: "perro", "gato", "pato").

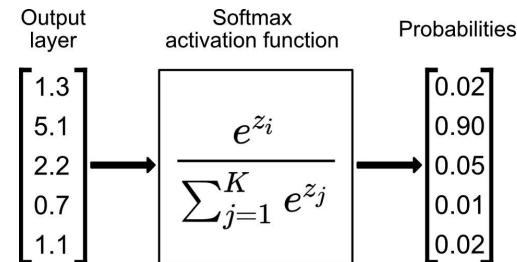
Podríamos usar la siguiente arquitectura:



Es decir, podemos utilizarlo para obtener:

$$\begin{aligned}\hat{y}_0 &= P(Y = \text{clase 0} \mid X = x^{(i)}) \\ \hat{y}_1 &= P(Y = \text{clase 1} \mid X = x^{(i)}) \\ \hat{y}_2 &= P(Y = \text{clase 2} \mid X = x^{(i)})\end{aligned}$$

Usamos **Softmax** una función de activación (que toma en cuenta los valores del resto de las neuronas). Su salida **suma 1 y es diferenciable**.



Redes Neuronales Multi-Output

Softmax



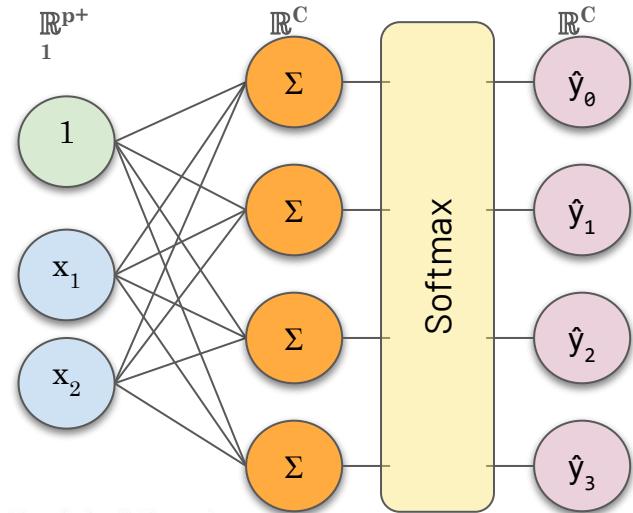
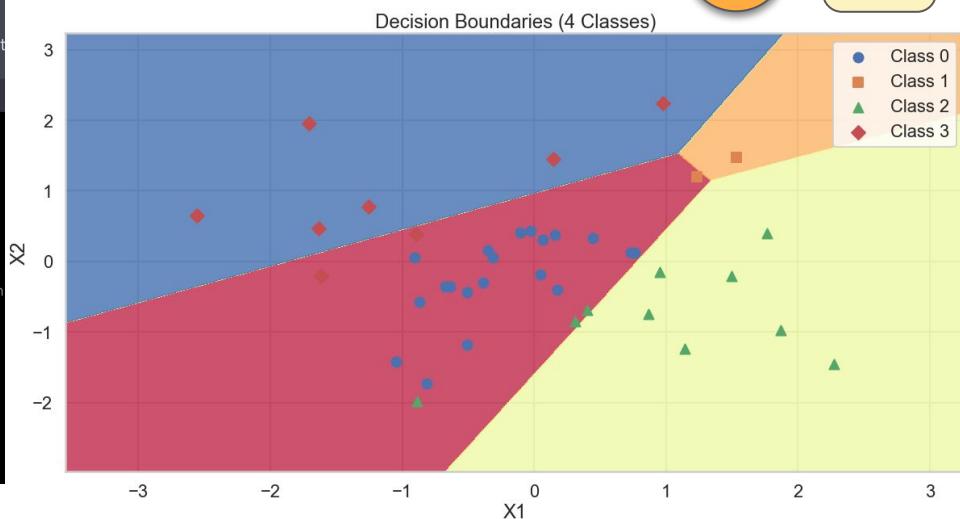
create a code in python that trains a neural network using pytorch, the network is only a linear layer with softmax activation. And there is a plot showing the decision boundaries for the example.

Certainly! Here's a Python code that trains a simple neural network with a linear layer and softmax activation using PyTorch and then plots the decision boundaries for a given dataset

```
python Copy code
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Define a simple neural network with a linear layer and softmax activation
class SimpleNN(nn.Module):
    def __init__(self, input_size, output_size):
        super(SimpleNN, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

    def forward(self, x):
        return self.linear(x)
```



Entropía Cruzada (multiclasificación)

Entropía cruzada (multiclas)

Una métrica muy utilizada en problemas de clasificación que permite calcular el error en un dataset para el cual tenemos la probabilidad estimada **para cada clase**.

$$J_{X,y} = \frac{1}{n} \sum_{i=1}^n \text{CE}^{(i)}$$

$$\hat{h}(x^{(i)}) = \text{softmax}(f^{(L)}(f^{(L-1)} \dots (x^{(i)})))$$

$$\begin{bmatrix} \hat{P}(Y = 0|X = x^{(i)}) \\ \hat{P}(Y = 1|X = x^{(i)}) \\ \vdots \\ \hat{P}(Y = C|X = x^{(i)}) \end{bmatrix}$$

$$\text{CE}^{(i)} = \sum_{c=1}^C -[y_c^{(i)} \log(\hat{h}_c(x^{(i)}))] \quad \text{en donde } \hat{h}_c(x^{(i)}) \text{ representa la probabilidad asignada a la clase c.}$$

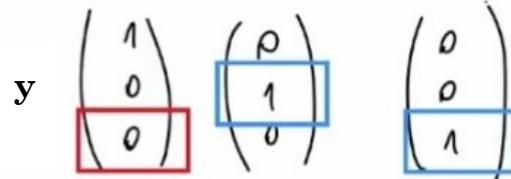
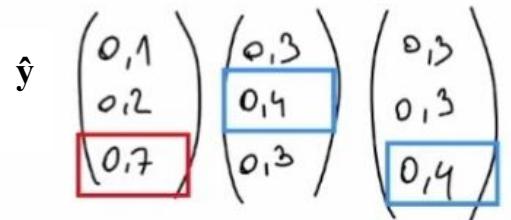
$y^{(i)}$ representa un vector de ceros salvo en la posición de la clase c, en donde vale 1.

equivalentemente

$$\text{CE}^{(i)} = \begin{cases} -\log(\hat{P}(Y = 0|X = x^{(i)})) & y_1^{(i)} = 1 \\ -\log(\hat{P}(Y = 1|X = x^{(i)})) & y_2^{(i)} = 1 \\ \dots & \\ -\log(\hat{P}(Y = C|X = x^{(i)})) & y_C^{(i)} = 1 \end{cases}$$

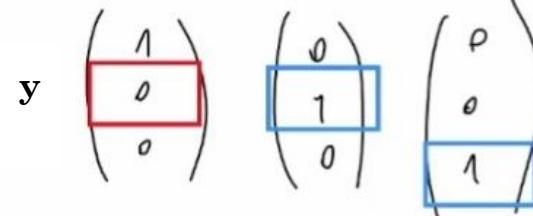
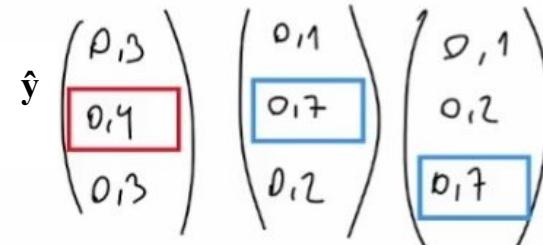
Notar que, como en caso binario, sólo un término sobrevive para cada instancia (todos los términos de la sumatoria son cero salvo uno)

Entropía cruzada vs Accuracy



Accuracy = $\frac{2}{3}$

Entropía cruzada = 4.14



Accuracy = $\frac{2}{3}$

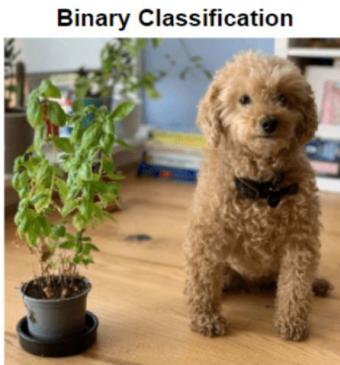
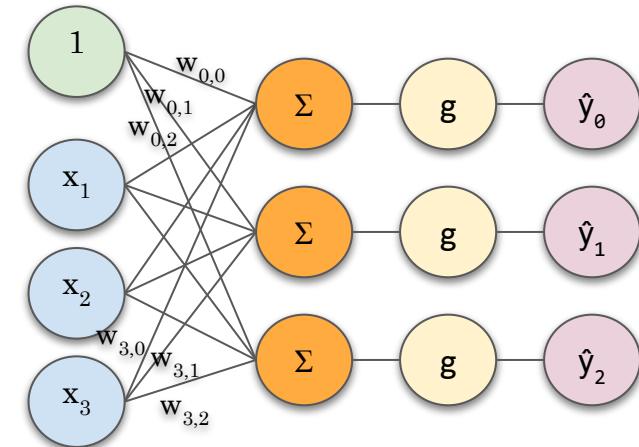
Entropía cruzada = 1.92

Redes Neuronales Multi Label

Redes Neuronales Multi-label

Hay problemas en los cuales tenemos más de una etiqueta por instancia.

- ¿Qué g se les ocurre utilizar en este caso?
- ¿Cómo definirían la función de pérdida?
- ¿Qué métrica se podría usar para medir el error bajo este problema? Buscar por ejemplo: Precision at k (P@k).



Tarea

Completar el notebook y formulario

Lecturas obligatorias:

- ISLR, capítulo 10: hasta la sección **10.2 (inclusive)**
- ISLR, capítulo 10: sección **10.7**
- Deep Learning: **Intro del Capítulo 6 + Sección 6.1 + Sección 6.5 hasta 6.5.2 inclusive.**

Opcional:

- Deep learning: El resto de la **Sección 6.5**.
- ISLR, capítulo 4: sección **4.3** Logistic Regression
- ISLR, capítulo 10: sección **10.8** Interpolation and Double Descent.
- Deep Learning: Resto del **Capítulo 6**.