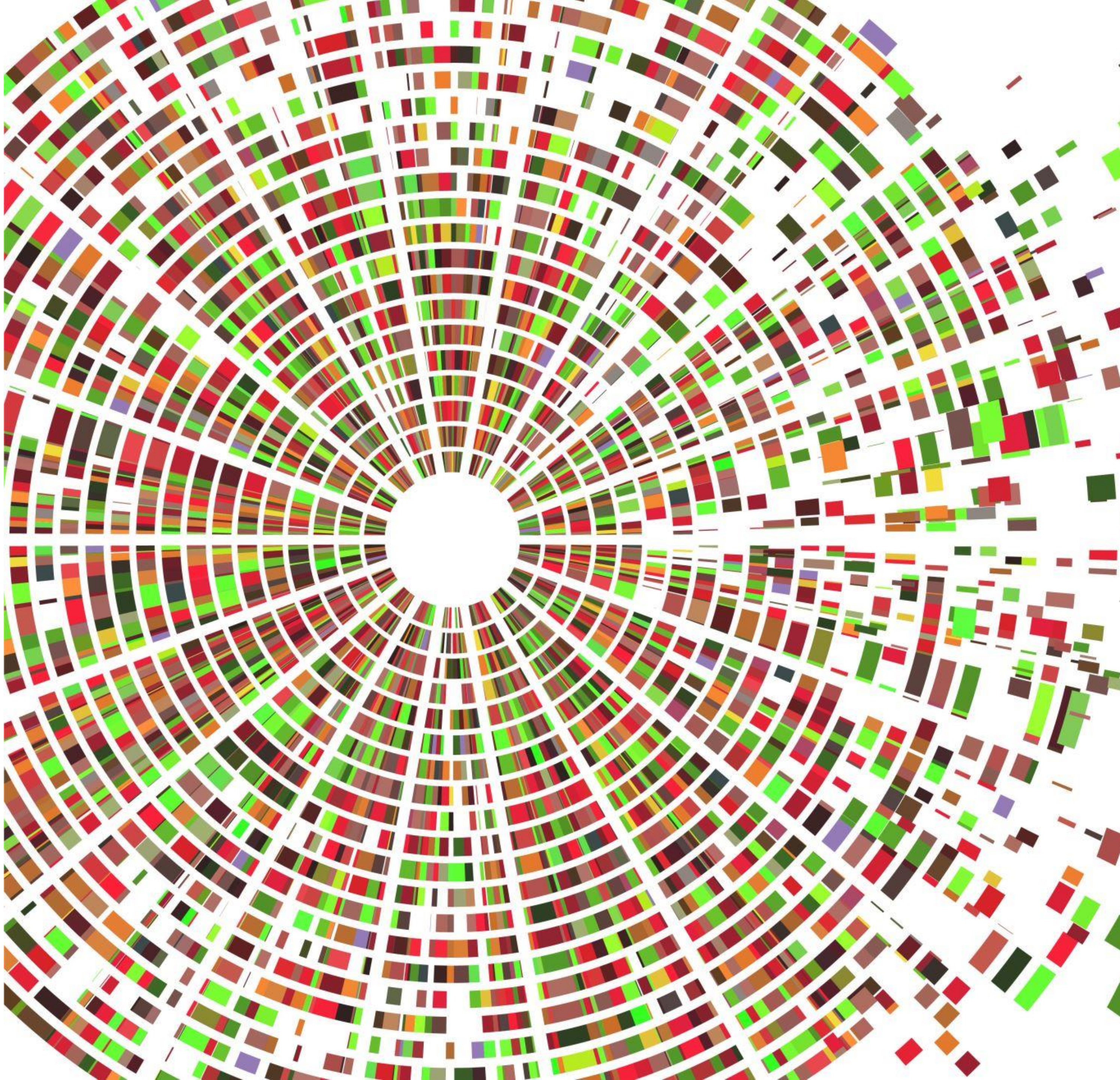
Métodos avanzados de ciencia de datos

Prof. Emily Díaz



Contenido

Qué son las redes neuronales

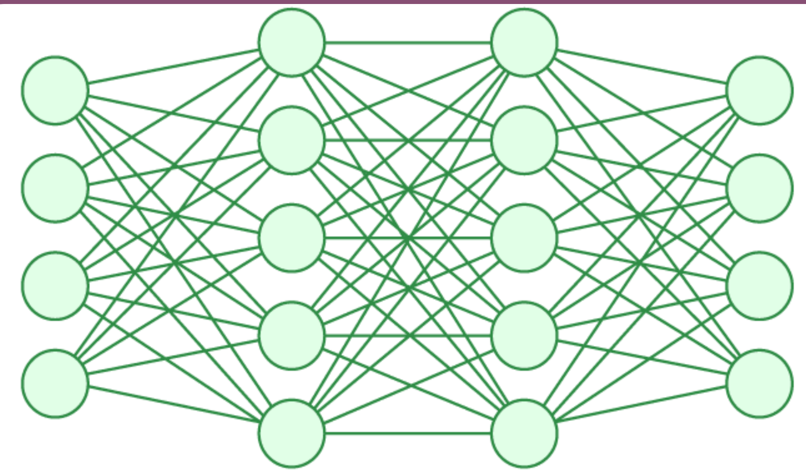
Estructura básica y partes

Propagación hacia delante

Entrenamiento: optimización y propagación hacia atrás

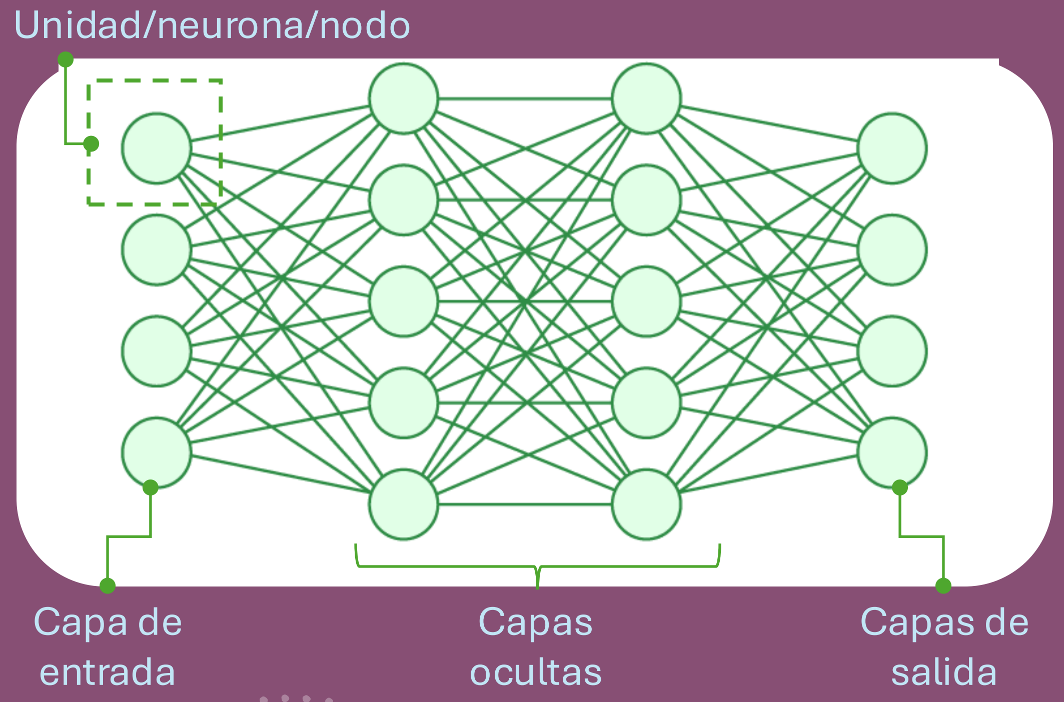
¿Qué son las redes neuronales?

Redes neuronales:



- Consisten en **unidades** o nodos conectados, que modelan vagamente las neuronas del cerebro.
- Las neuronas están conectados por **bordes**, que modelan las sinapsis en el cerebro.
- Cada neurona **recibe señales** de las neuronas conectadas, luego las procesa y **envía una señal** a otras neuronas conectadas.
- La "señal" es un número real y la salida de cada neurona se calcula mediante **una función no lineal** de la suma de sus entradas, llamada **función de activación**.
- La fuerza de la señal en cada conexión está determinada por un **peso**, ajustado durante el **proceso de aprendizaje**.

Redes neuronales:

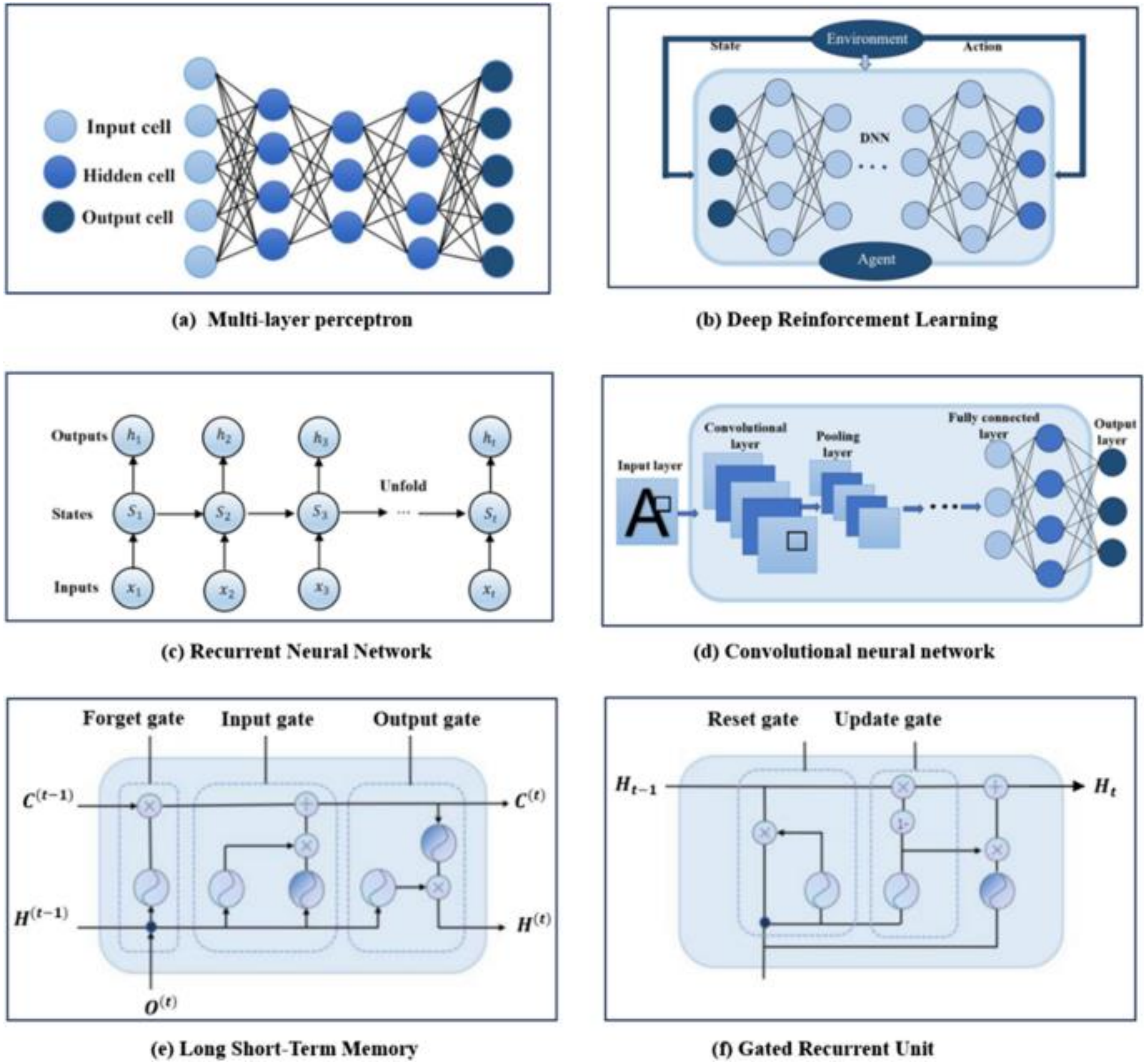


- Se estructuran por capas, teniendo capa de **entrada, ocultas y de salida.** Las señales viajan desde la primera capa (entrada) a la última capa (salida), posiblemente pasando por múltiples capas intermedias (ocultas).
- **Conexiones:** Entre dos capas, son posibles múltiples patrones de conexión. Pueden estar **"completamente conectados"**, con cada neurona en una capa conectada a cada neurona en la siguiente capa. También existen redes que permiten conexiones entre neuronas en la misma capa o en capas anteriores se conocen como redes recurrentes, entre otras.
- **Entrenamiento: Optimizar** los parámetros de la red para minimizar la diferencia/riesgo (**función de pérdida**). Los métodos basados en gradientes, como la **retropropagación**, se utilizan para **iterativamente** estimar los parámetros de la red en pos de la minimización de la función de pérdida.

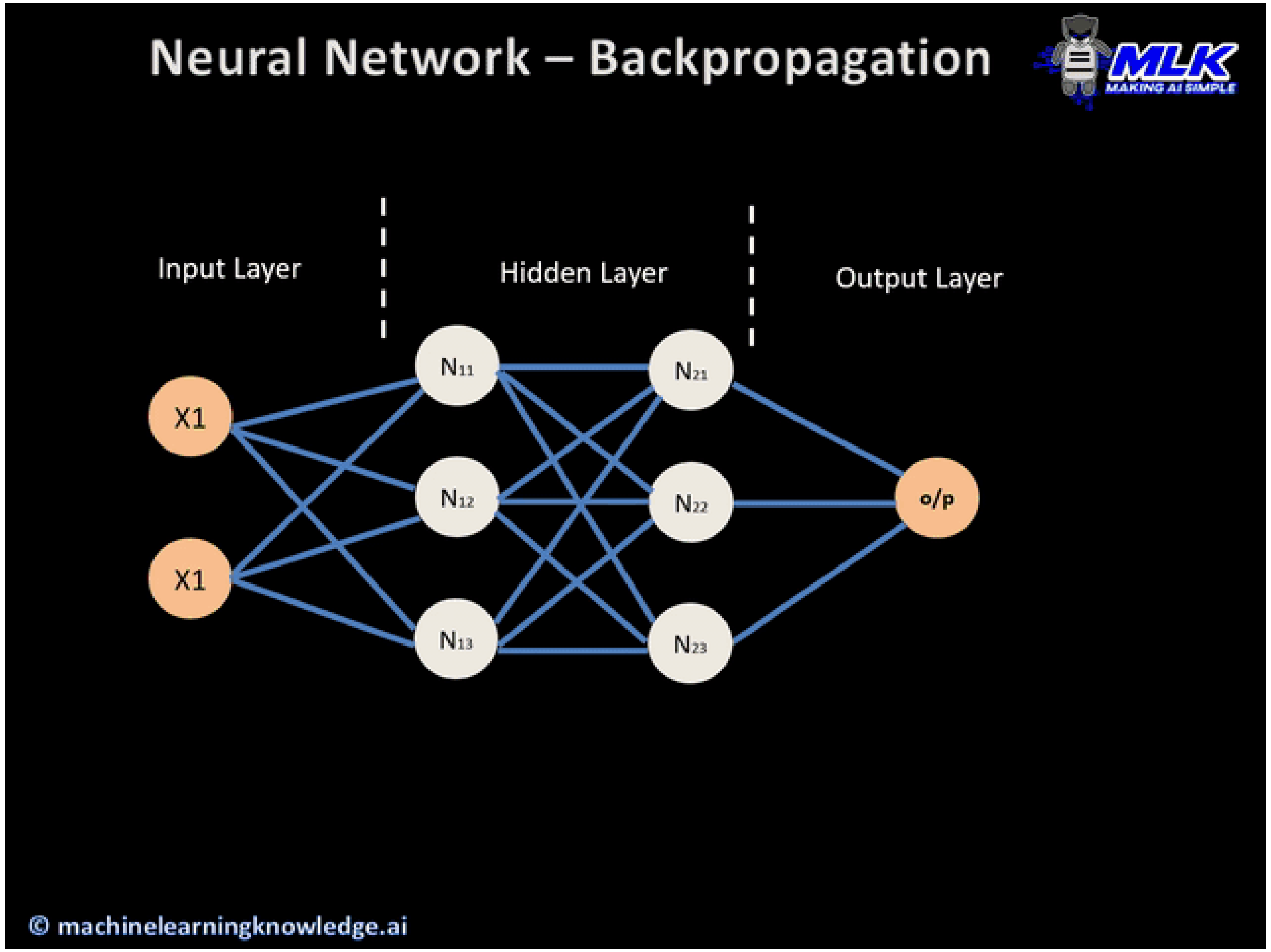
Estructuras básicas

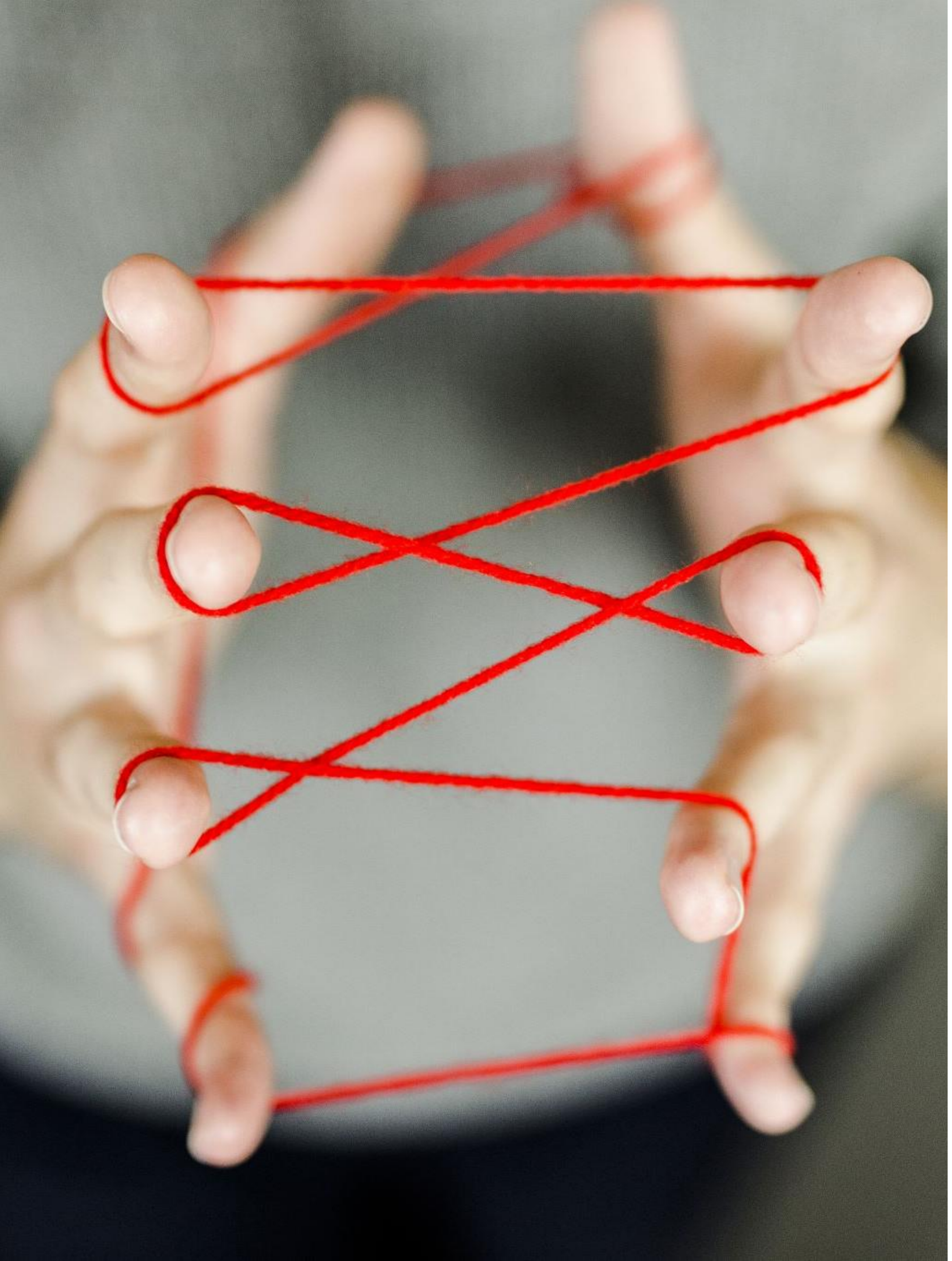
Redes neuronales:

- Arquitectura más sencilla: **Multi-Layer Perceptron (MLP)**
- Otras: **CNNs**, **RNNs**, etc con problemas particulares para los que fueron desarrolladas
- **Hiperparámetros:** Parámetro constante cuyo valor se establece antes de que comience el proceso de aprendizaje. Algunos ejemplos de hiperparámetros son la **tasa de aprendizaje (learning rate), la cantidad de capas ocultas y el tamaño del *batch*.**



Ejemplo de funcionamiento





Pasos de construcción de una red neuronal

1. **Propagación hacia delante (Forward pass):** En esta fase, los datos de entrada se pasan a través de la red neuronal, capa por capa, hasta obtener la salida final.
2. **Cálculo de pérdida:** La pérdida o error se calcula comparando la salida de la red neuronal con el valor real (target). Esta pérdida cuantifica cuán lejos está la predicción de la red del valor esperado.
3. **Propagación hacia atrás (Backward propagation)**: Se calcula el gradiente de la pérdida respecto a cada peso en la red neuronal, utilizando el algoritmo de retropropagación. Este gradiente se usa para actualizar los pesos.
4. **Optimización:** Los pesos de la red se ajustan en función de los gradientes calculados. Este paso generalmente implica el uso de un optimizador, como el descenso de gradiente, que ajusta los pesos para minimizar la pérdida.

Cada iteración de estos pasos se llama un epoch, y el proceso se repite muchas veces hasta que la red converja a una solución óptima (o suficientemente buena).

Propagación hacia delante

Ejemplo: Gato o Perro?

- Queremos diferenciar perros y gatos con dos características físicas: peso del animal y largo de las orejas
- En general, los perros tienden a pesar más que los gatos y a tener orejas de mayor longitud

- Tenemos un ejemplo de perro: 12kg de peso y 8cm de largo de orejas ($x = [8,12]$)

- Usamos una red neuronal con una capa oculta de 3 neuronas

Cuántas neuronas de entrada tenemos y cuantas de salida?

Ejemplo: Gato o Perro?

- Queremos diferenciar perros y gatos con dos características físicas: peso del animal y largo de las orejas
- En general, los perros tienden a pesar más que los gatos y a tener orejas de mayor longitud

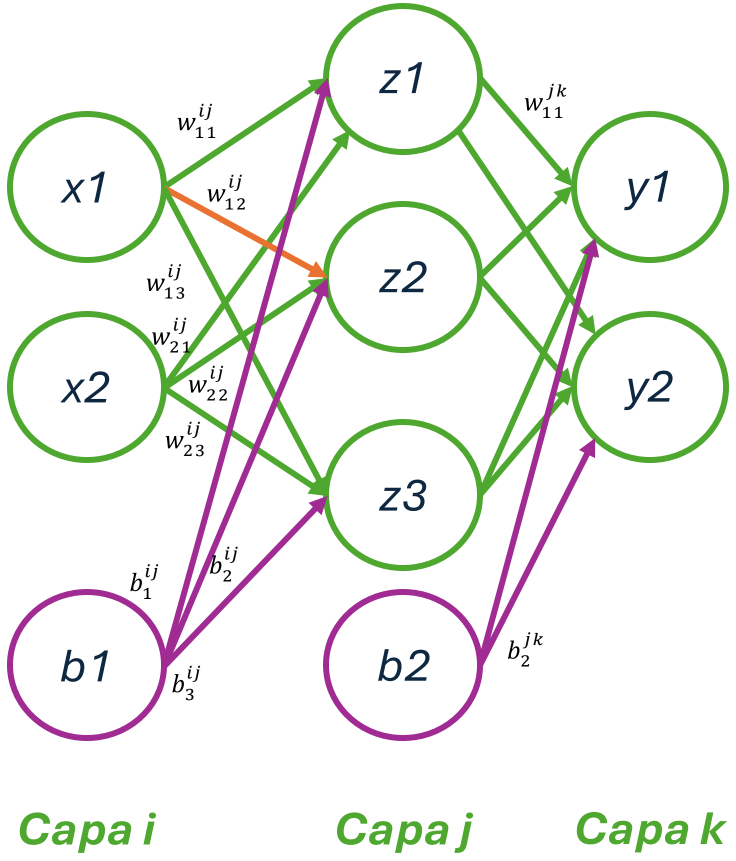
- Tenemos un ejemplo de perro: 12kg de peso y 8cm de largo de orejas (x = [8,12])

- Usamos una red neuronal con una capa oculta de 3 neuronas

Cuántas neuronas de entrada tenemos y cuantas de salida?
2 de entrada (número de variables) y
1 o 2 de salida (número clases a predecir)

Ejemplo: Gato o Perro?

- Por simplicidad, asumamos que los pesos de la red son los siguientes:



$$W_{ij}^T = \begin{array}{cc} & \begin{array}{cc} X_1 & X_2 \end{array} \\ \begin{array}{c} Z_1 \\ Z_2 \\ Z_3 \end{array} & \begin{bmatrix} 0.1 & 0.2 \\ \boxed{0.3} & 0.4 \\ 0.5 & 0.6 \end{bmatrix} \end{array} \qquad b_{ij} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix}$$

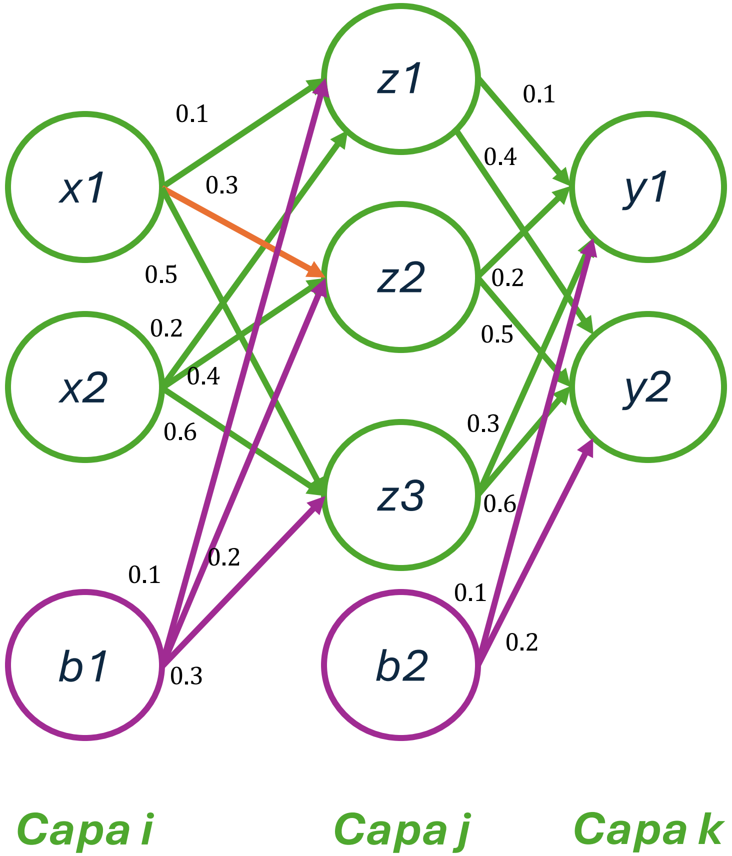
Peso de neurona 1 de capa i a neurona 2 de capa j (anaranjada en el diagrama)

$$W_{jk}^T = \begin{array}{cc} & \begin{array}{ccc} Z_1 & Z_2 & Z_3 \end{array} \\ \begin{array}{c} Y_1 \\ Y_2 \end{array} & \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{bmatrix} \end{array} \qquad b_{jk} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

*En capa de salida, **el número de neuronas responde al tipo de problema**: en caso de regresión que necesitamos un valor numérico predicho usamos una neurona, en el caso de clasificación, depende del número de clases*

Ejemplo: Gato o Perro?

- Por simplicidad, asumamos que los pesos de la red son los siguientes:



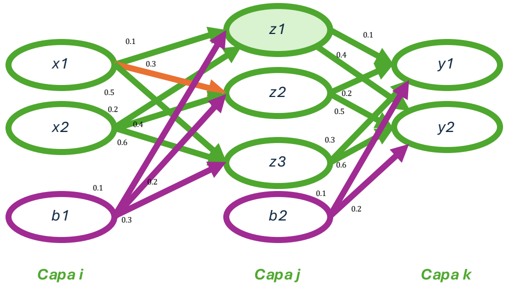
$$W_{ij}^T = \begin{array}{c} \\ Z_1 \\ Z_2 \\ Z_3 \end{array} \begin{array}{c} \begin{array}{cc} X_1 & X_2 \end{array} \\ \begin{bmatrix} 0.1 & 0.2 \\ \boxed{0.3} & 0.4 \\ 0.5 & 0.6 \end{bmatrix} \end{array} \qquad b_{ij} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix}$$

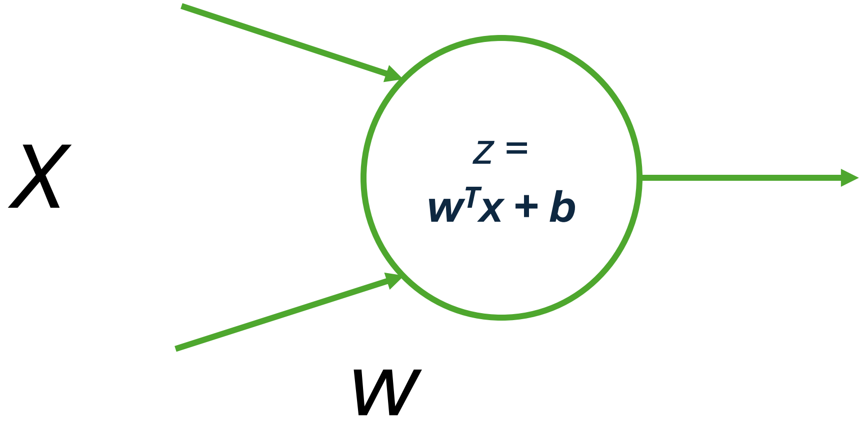
Peso de neurona 1 de capa i a neurona 2 de capa j (anaranjada en el diagrama)

$$W_{jk}^T = \begin{array}{c} \\ Y_1 \\ Y_2 \end{array} \begin{array}{c} \begin{array}{ccc} Z_1 & Z_2 & Z_3 \end{array} \\ \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{bmatrix} \end{array} \qquad b_{jk} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

*En capa de salida, **el número de neuronas responde al tipo de problema**: en caso de regresión que necesitamos un valor numérico predicho usamos una neurona, en el caso de clasificación, depende del número de clases*

Descomposición de red neuronal: 1) Parte lineal





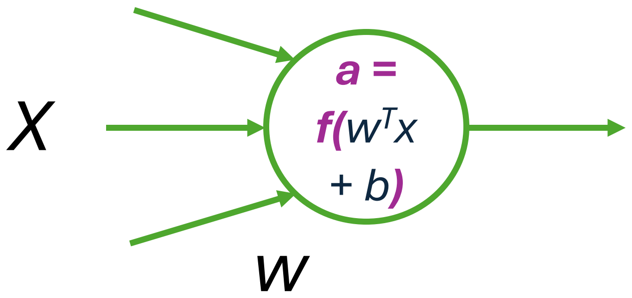
W son los pesos, X es la información de entrada y para obtener el valor de esta neurona realizamos la operación ***$W^T X + b$*** *(matricial)*

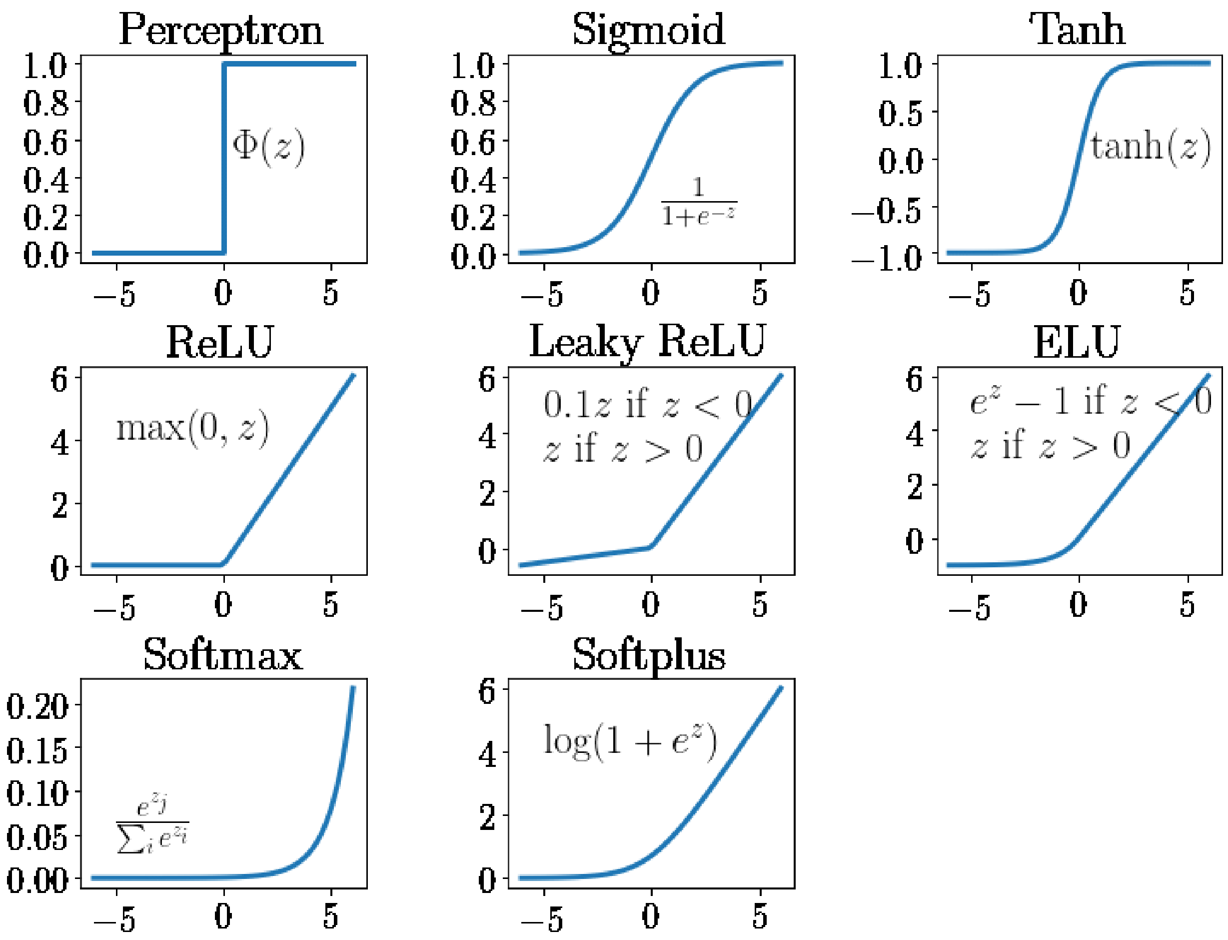
$$x = \begin{bmatrix} 8 \\ 12 \end{bmatrix} \quad b_j = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix} \quad W_{ij}^T = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}$$

Ejemplo de neurona z_1 en capa oculta del clasificador de perro o gato:
b1 + w11 * x1 + w21 *x2 = 0.1 + 0.1*8 + 0.2 * 12 = **3.3**

** Separador decimal como punto en estas diapositivas*

Descomposición de red neuronal: 2) Funciones de activacion





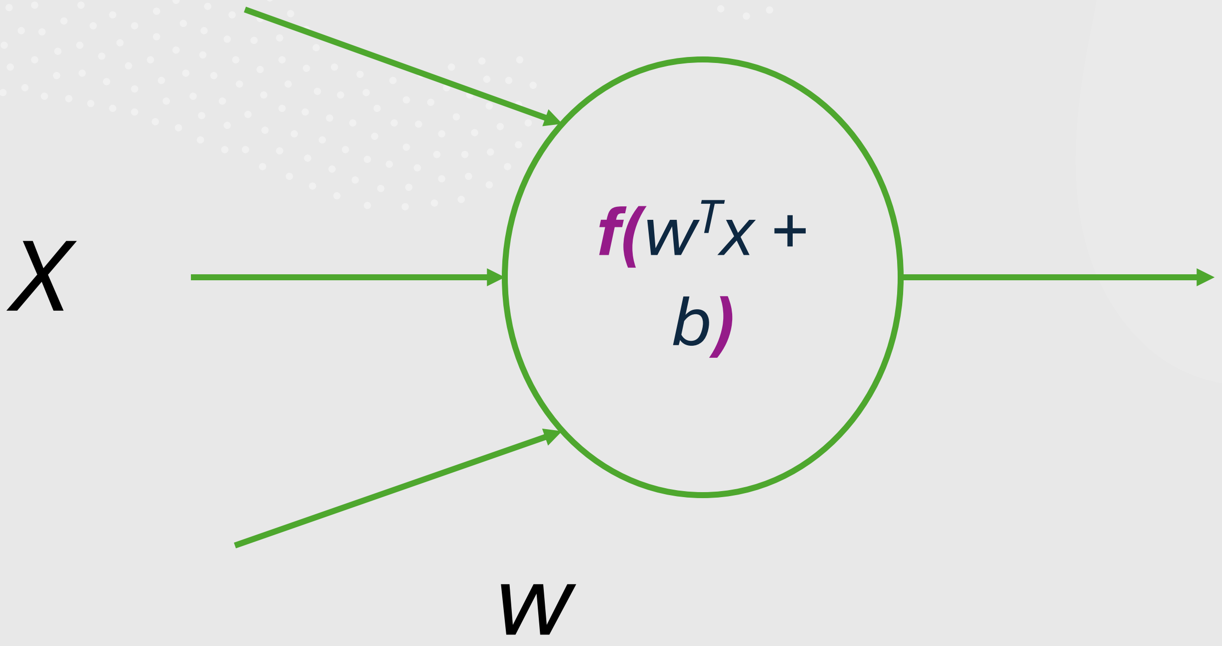
Que es una function de activacion?

- Función matemática aplicada a la salida de una neurona.
- Las funciones de activación son útiles porque añaden no linealidades a la red, lo que les permite aprender operaciones poderosas y patrones complejos.

Cuáles son las más populares?

- Rectified Linear Unit (ReLU), sigmoid, softmax, tanh
- ReLU es el más popular, además de añadir no-linealidad, Evita problemas de saturación y mitiga el problema del gradiente de desaparición (*vanishing gradient* lo estudiaremos más adelante)

Descomposición de red neuronal: 2) Funciones de activacion

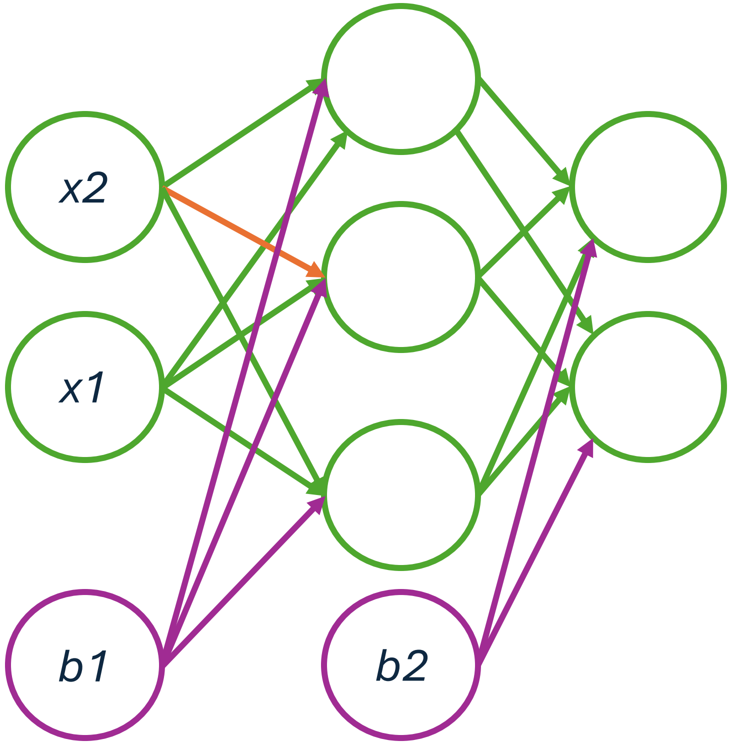


Ejemplo de neurona z_1 en capa oculta del clasificador de perro o gato:

b1 + w11 * x1 + w21 *x2 = 0.1 + 0.1*8 + 0.2 * 12 = **3.3**

Activación ReLU: max(0,3.3) = 3.3

Descomposición de red neuronal: 2) Una capa



Capa i *Capa j* *Capa k*

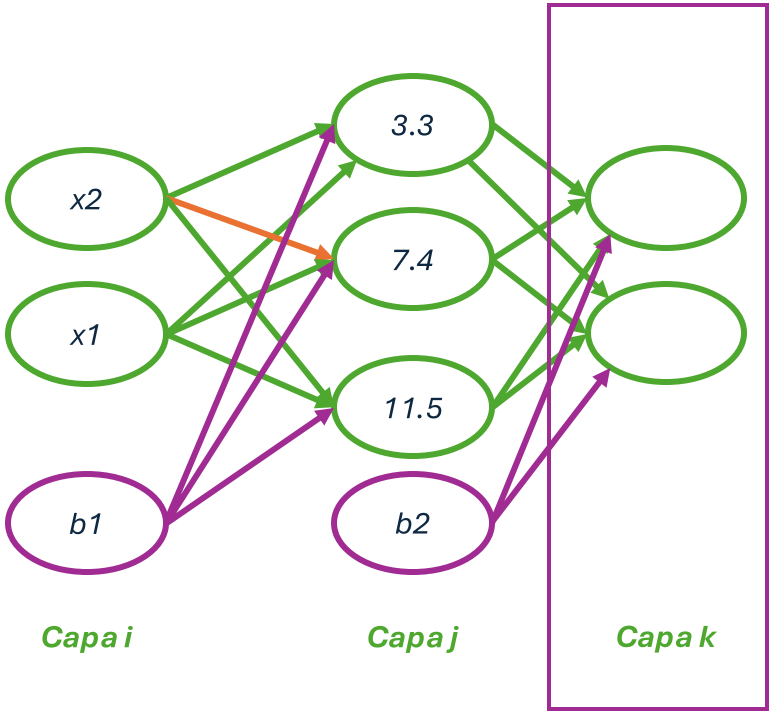
$$W_{ij}^T = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix}_{3x2} \qquad b_{ij} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \end{bmatrix}_{3x1}$$

$$x = \begin{bmatrix} 8 \\ 12 \end{bmatrix}_{2x1}$$

$$W^T X + b = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{bmatrix} \times \begin{matrix} 8 \\ 12 \end{matrix} + \begin{matrix} 0.1 \\ 0.2 \\ 0.3 \end{matrix} = \begin{matrix} 3.3 \\ 7.4 \\ 11.5 \end{matrix}$$

Con ReLU quedan en los mismos valores ya que todos son mayores que cero

Con más capas



$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Capa k es de salida por lo que aplicamos softmax (también podría ser sigmoid por ser 2 clases) – Por qué?

$$W_{jk}^{T} = \begin{bmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{bmatrix} \qquad b_{jk} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

$$x = \begin{bmatrix} 3.3 \\ 7.4 \\ 11.5 \end{bmatrix}$$

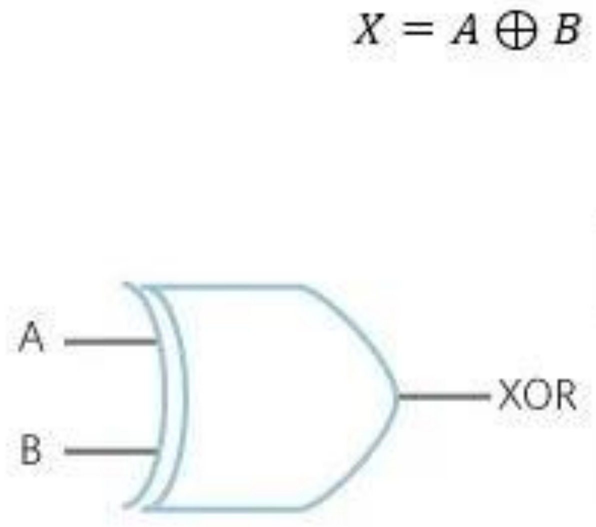
$$W^{T}X + b = \begin{matrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{matrix} \times \begin{matrix} 3.3 \\ 7.4 \\ 11.5 \end{matrix} + \begin{matrix} 0.1 \\ 0.2 \end{matrix} = \begin{matrix} 5.36 \\ 11.92 \end{matrix}$$

Softmax(z)= e(5.36)/(e(5.36) + e(11.92) = 0.0014

$$= \begin{matrix} 0.0014 \\ 0.9986 \end{matrix}$$

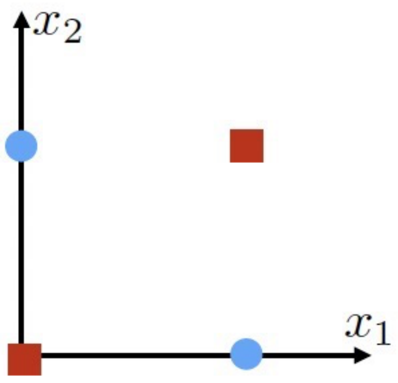
Problema XOR

- XOR significa "OR exclusivo". Es una operación lógica que da como resultado verdadero (1) solo cuando las entradas son diferentes.
- El problema XOR es un ejemplo fundamental de las limitaciones de los perceptrones de una sola capa y de la necesidad de redes neuronales más complejas.
- Al introducir perceptrones multicapa, el algoritmo de retropropagación y las funciones de activación adecuadas, podemos resolver con éxito el problema XOR.



A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

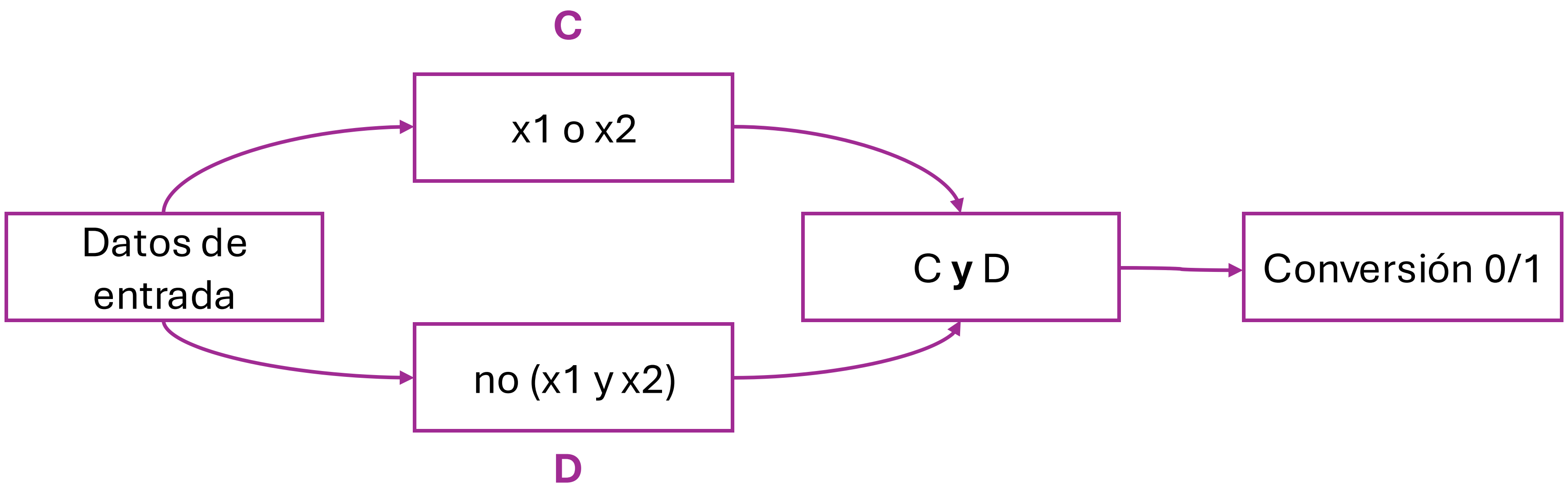
Problema XOR



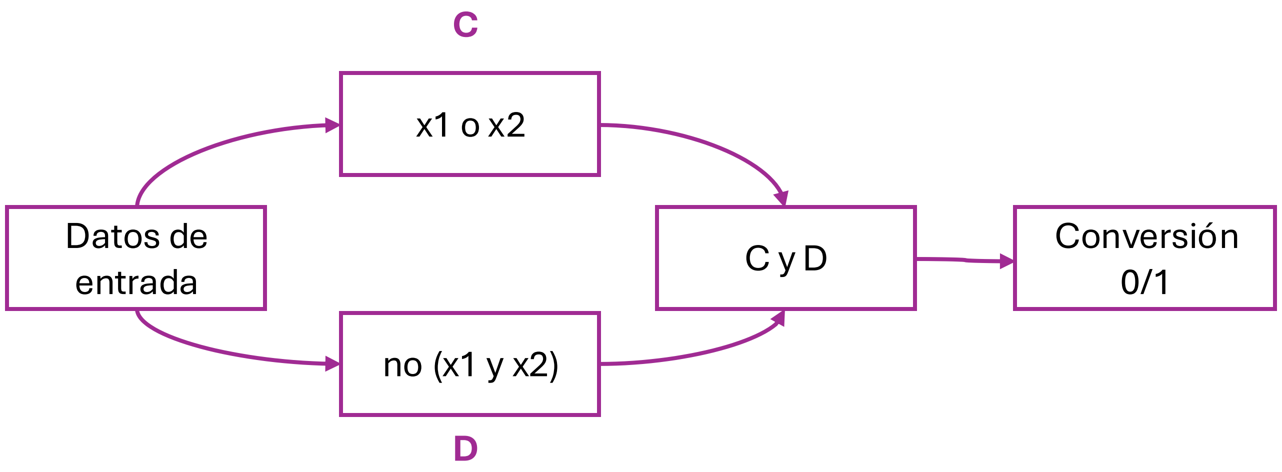
- No se puede tener un límite de decision lineal
- Se puede dividir en distintos cálculos:

(x1 **o** x2) **y** NO (x1 **y** x2)

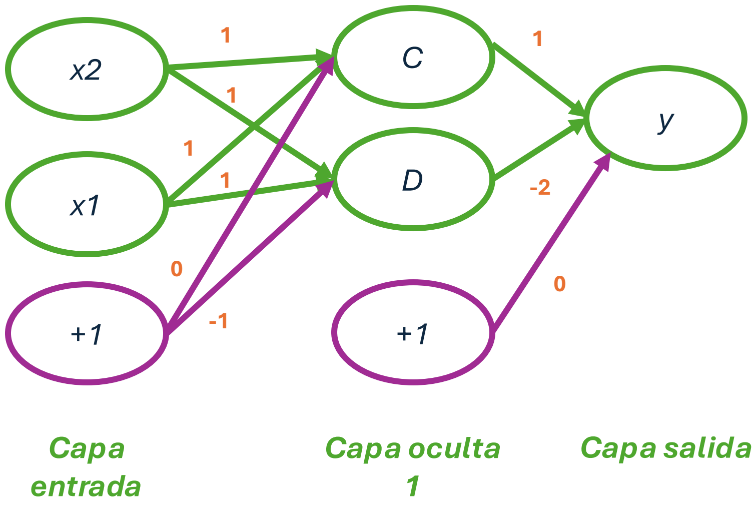
C D



Problema XOR



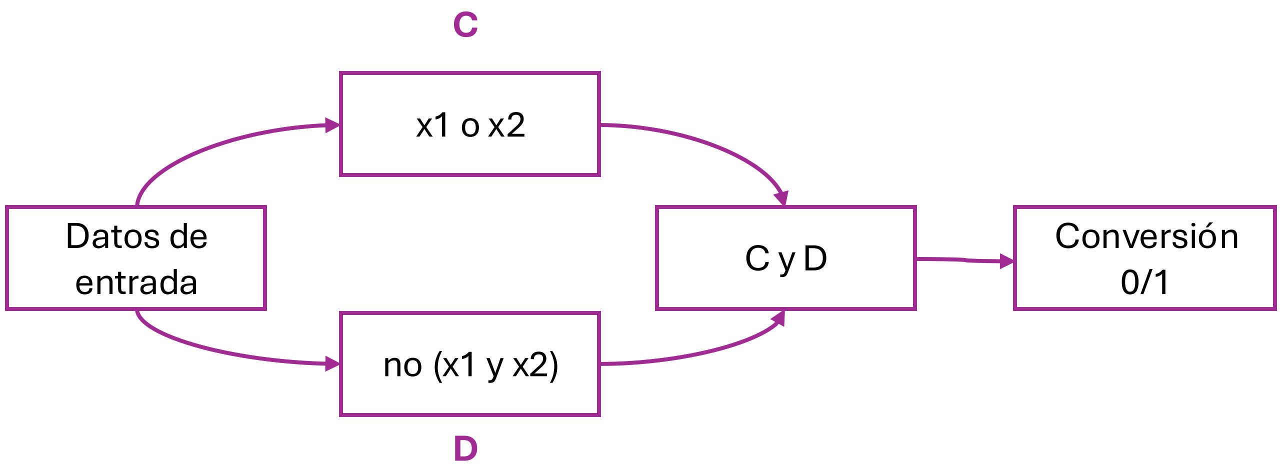
Red neuronal



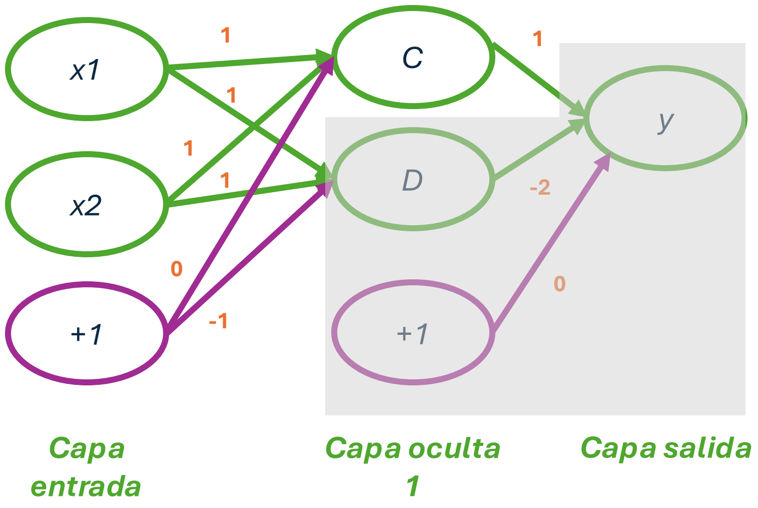
Resultado deseado

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Problema XOR



Red neuronal

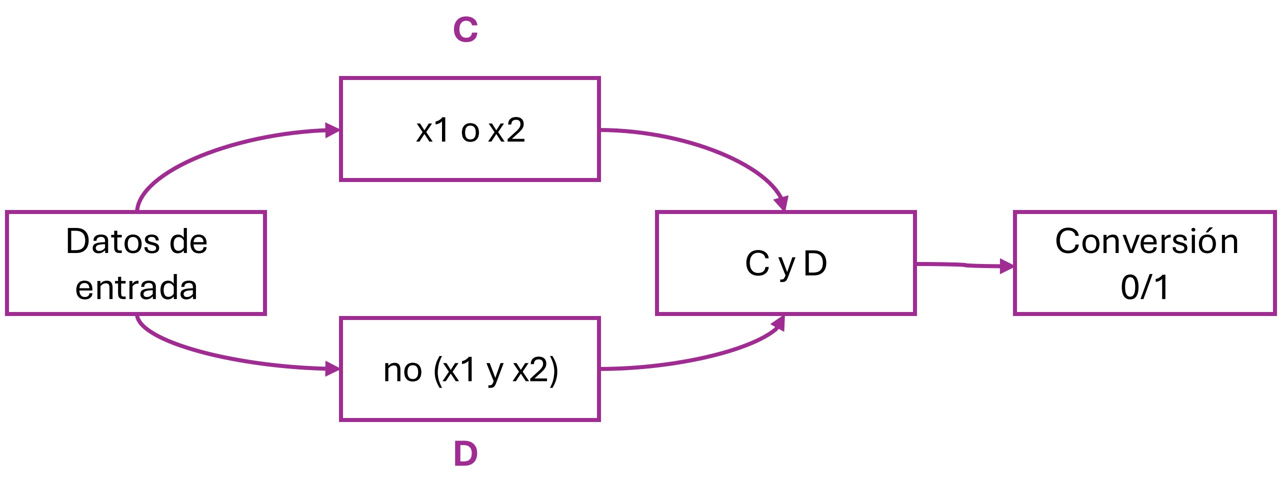


Asumir activación ReLU

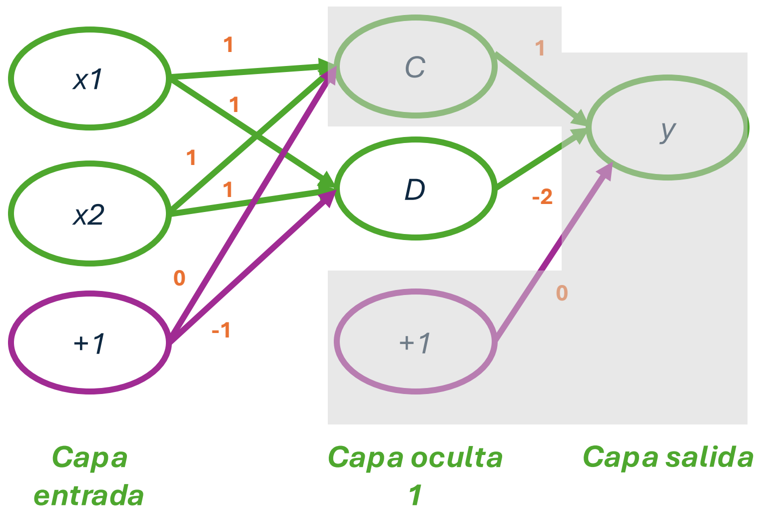
Resultado de C

X1	X2	C
0	0	Max(0, 0*1 + 0*1 + 0) = 0
0	1	Max(0, 0*1 + 1*1+ 0) = 1
1	0	Max(0, 1*1 + 0*1 + 0) = 1
1	1	Max(0, 1*1 + 1*1 + 0) = 2

Problema XOR



Red neuronal



Asumir activación ReLU

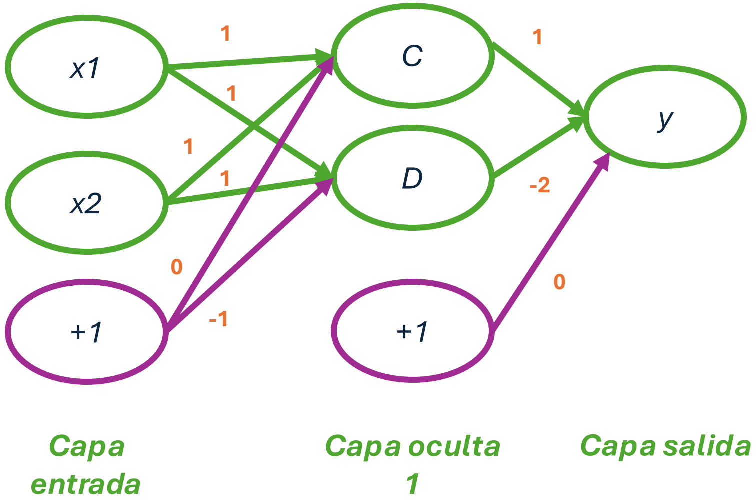
Resultado de D – solo parte (x1 y x2) – la negación se resuelve con los pesos en el siguiente paso

X1	X2	D
0	0	Max(0, 0*1 + 0*1 - 1) = 0
0	1	Max(0, 0*1 + 1*1- 1) =0
1	0	Max(0, 1*1 + 0*1 - 1) = 0
1	1	Max(0, 1*1 + 1*1 - 1) = 1

Problema XOR

Resultado de Y

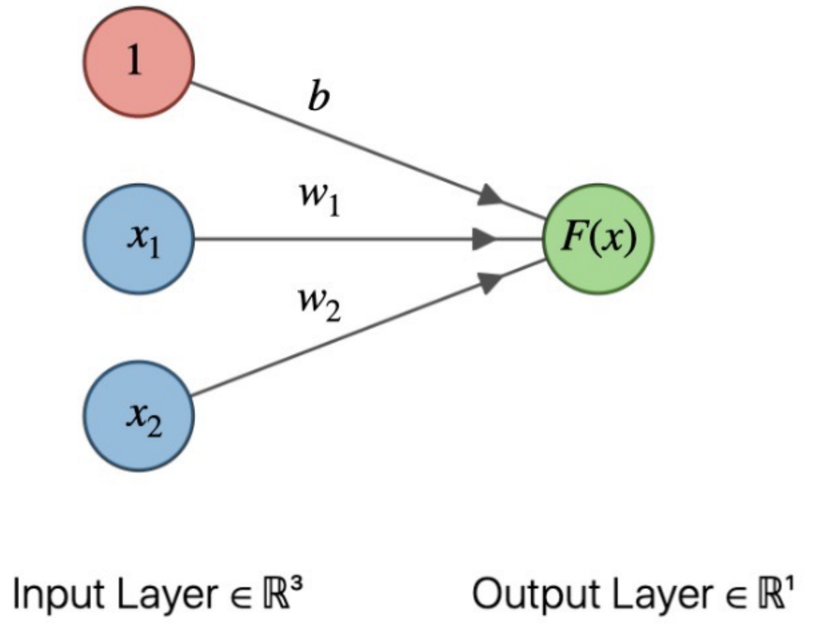
Red neuronal



X1	X2	C	D	Y
0	0	0	0	Max(0, 0*1 + 0*-2 – 0) = 0
0	1	1	0	Max(0, 1*1 + 0*-2 – 0) = 1
1	0	1	0	Max(0, 1*1 + 0*-2 – 0) = 1
1	1	2	1	Max(0, 2*1 + 1*-2 – 0) = 0

Se logra el resultado no lineal deseado por medio de trabajar con varias neuronas y funciones de activación no lineales

Redes neuronales - casos particulares

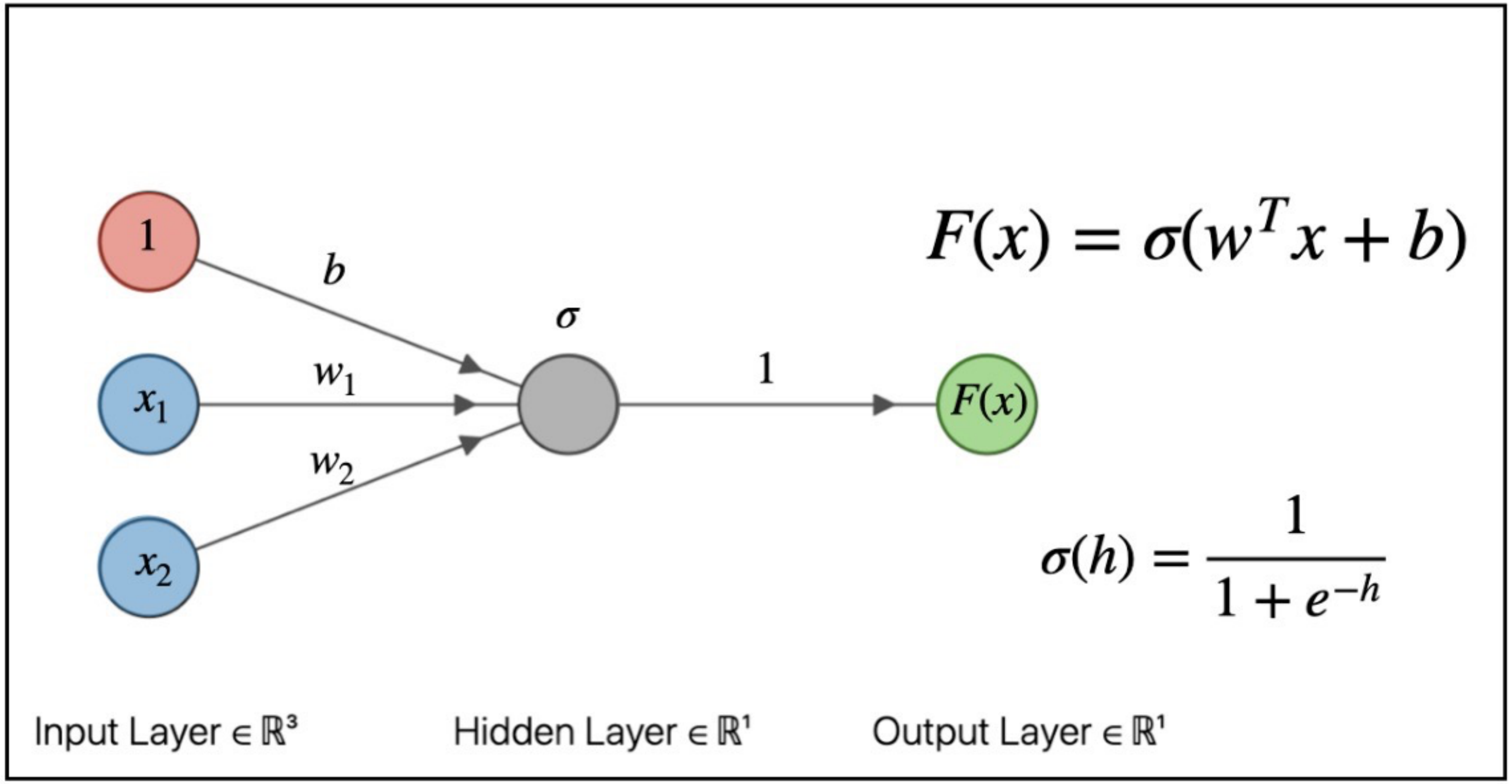


$$F(x) = w^T x + b$$

$$F(x) = w_1 \cdot x_1 + w_2 \cdot x_2 + 1 \cdot b$$

¿Cómo se le conoce a este modelo?

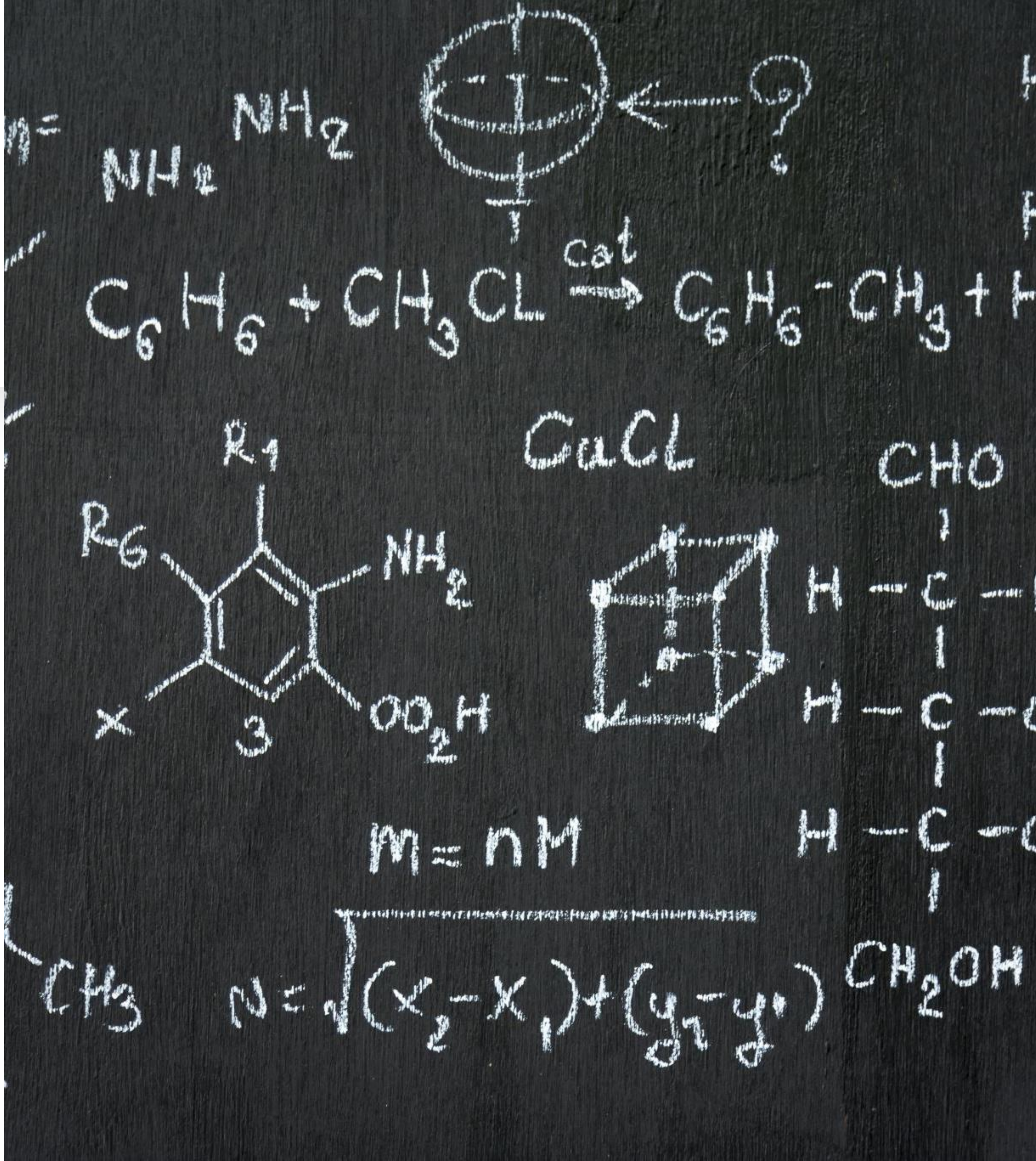
Redes neuronales - casos particulares



¿Cómo se le conoce a este modelo?

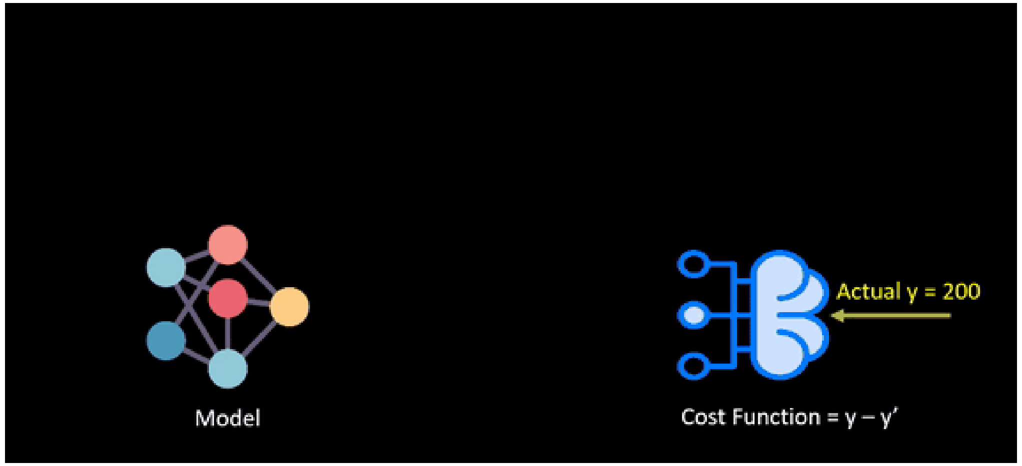
Volvamos a los pasos de construir una red neuronal

1. Forward pass
2. **Cálculo de pérdida**
3. Propagación hacia atrás
4. Optimización



¿Qué es una función de pérdida?

- Una función de pérdida mide la **eficacia de un modelo** para realizar una determinada tarea, que en la mayoría de los casos es regresión o clasificación.
- En el caso de redes neuronales, debemos **minimizar el valor de la función de pérdida** durante el paso de retropropagación/propagación hacia atrás para mejorar la red neuronal.
- Matemáticamente, podemos medir la diferencia (o error) entre el **vector de predicción** y y la **etiqueta/valor real** definiendo una función de pérdida cuyo **valor depende de esta diferencia**.



Ejemplos de funciones de pérdida de otros modelos clásicos de Machine Learning

Error cuadrático medio

$$L(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \longrightarrow \hat{\beta} = argmin_{\beta}\, \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i)$$

En esencia, buscamos el conjunto de coeficientes de la ecuación que minimiza el error

? ¿Cuál modelo han visto con este error?

Ejemplos de funciones de pérdid de otros modelos clásicos de Machine Learning

¿Cuál modelo han visto con este error?

Entropía Cruzada Binaria (Binary cross-entropy)

$$L(y,\hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log(\hat{y}_i) + (1-y_i)\log(1 - \hat{y}_i)) \longrightarrow \hat{\beta} = argmin_{\beta}\frac{1}{n}\sum_{i=1}^{n}L(y_i,\hat{y}_i)$$

Ejemplo de por qué esta función es ideal:

Caso en que realmente es clase 0 y probabilidad de ser 1 es 0.8

- - (0*log(0.8) + (1-0)*log(1-0.8)) = 0.698 (dado el negativo en L(), entre más bajo, más alto el castigo)

Caso en que realmente es clase 0 y probabilidad de ser 1 es 0.2

- - (0*log(0.2) + (1-0)*log(1-0.2)) = 0.096

Caso en que realmente es clase 1 y probabilidad de ser 1 es 0.8

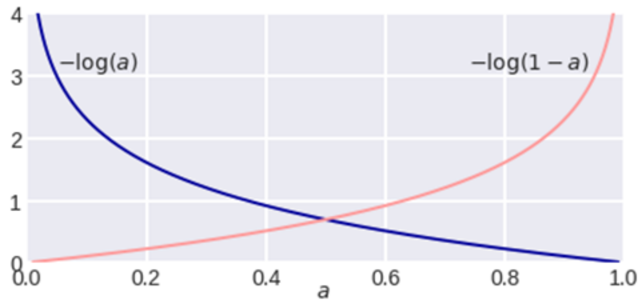
- - (1*log(0.8) + (1-1)*log(1-0.8)) = 0.096 (dado el negativo en L(), entre más bajo, más alto el castigo)

Caso en que realmente es clase 1 y probabilidad de ser 1 es 0.2

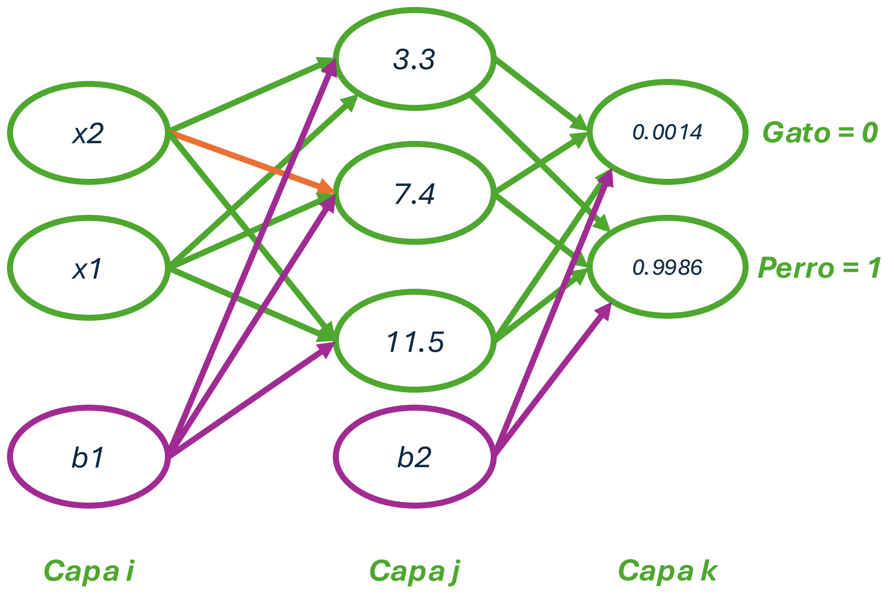
- - (1*log(0.2) + (1-1)*log(1-0.2)) = 0.698

Vemos que cuando la probabilidad es baja de ser 1 y es 0 realmente entonces la penalización/error es muy baja.

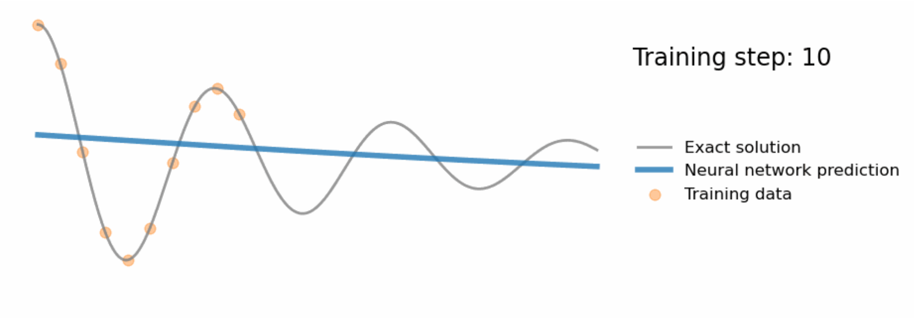
$$-\begin{cases} \log a_i, & y_i = 1, \\ \log(1-a_i), & y_i = 0. \end{cases}$$



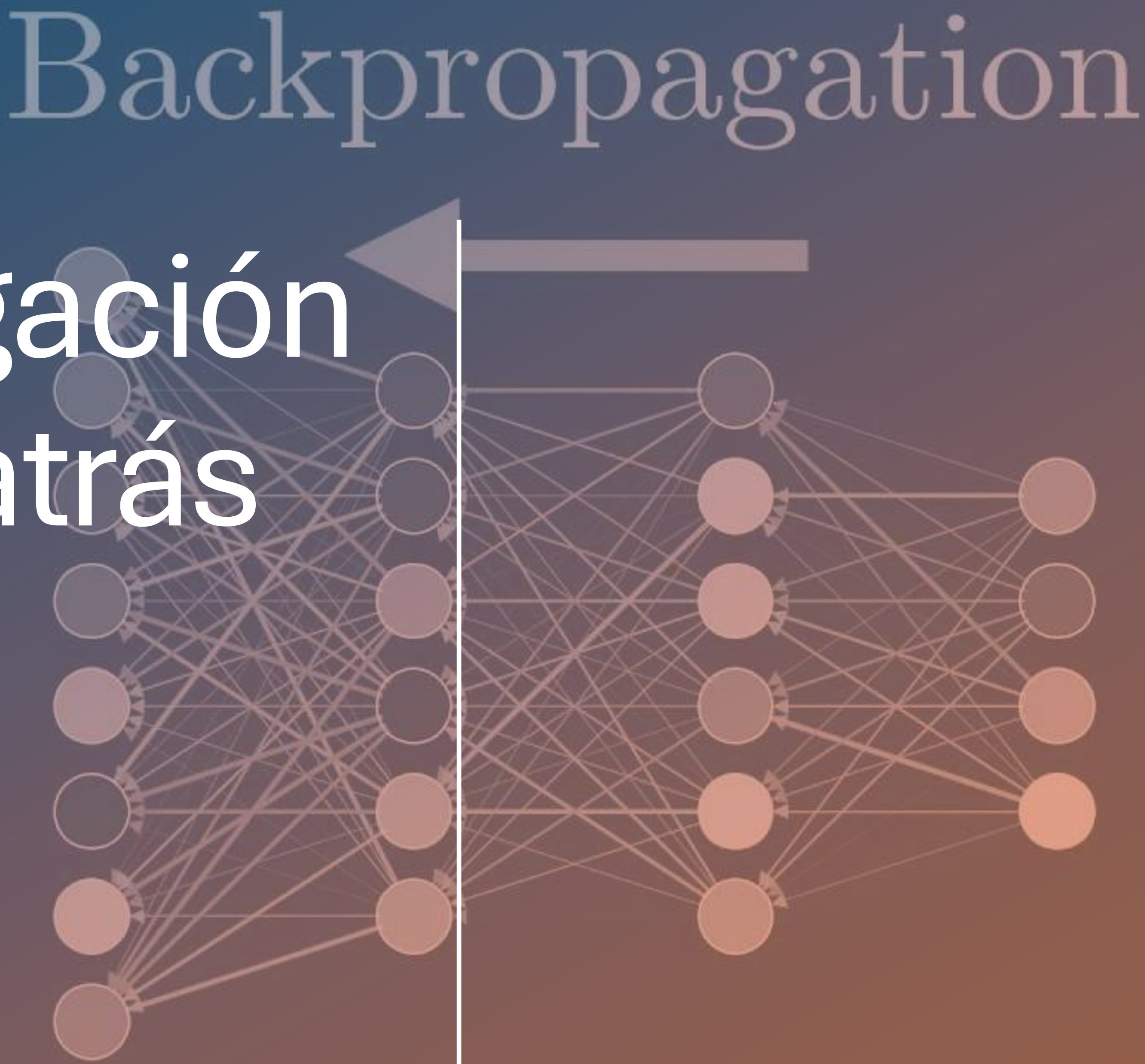
Ejemplo de cálculo para una red neuronal



- Si el ejemplo era **perro**, entonces el cálculo de error para este caso es:
  - (1*log(0.9986) + (1-1)*log(1-0.9986)) = log(0.9986) = ***0.00061***
- Una vez que tenemos las probabilidades predichas para todas las observaciones, **sumamos los errors y dividimos entre N**

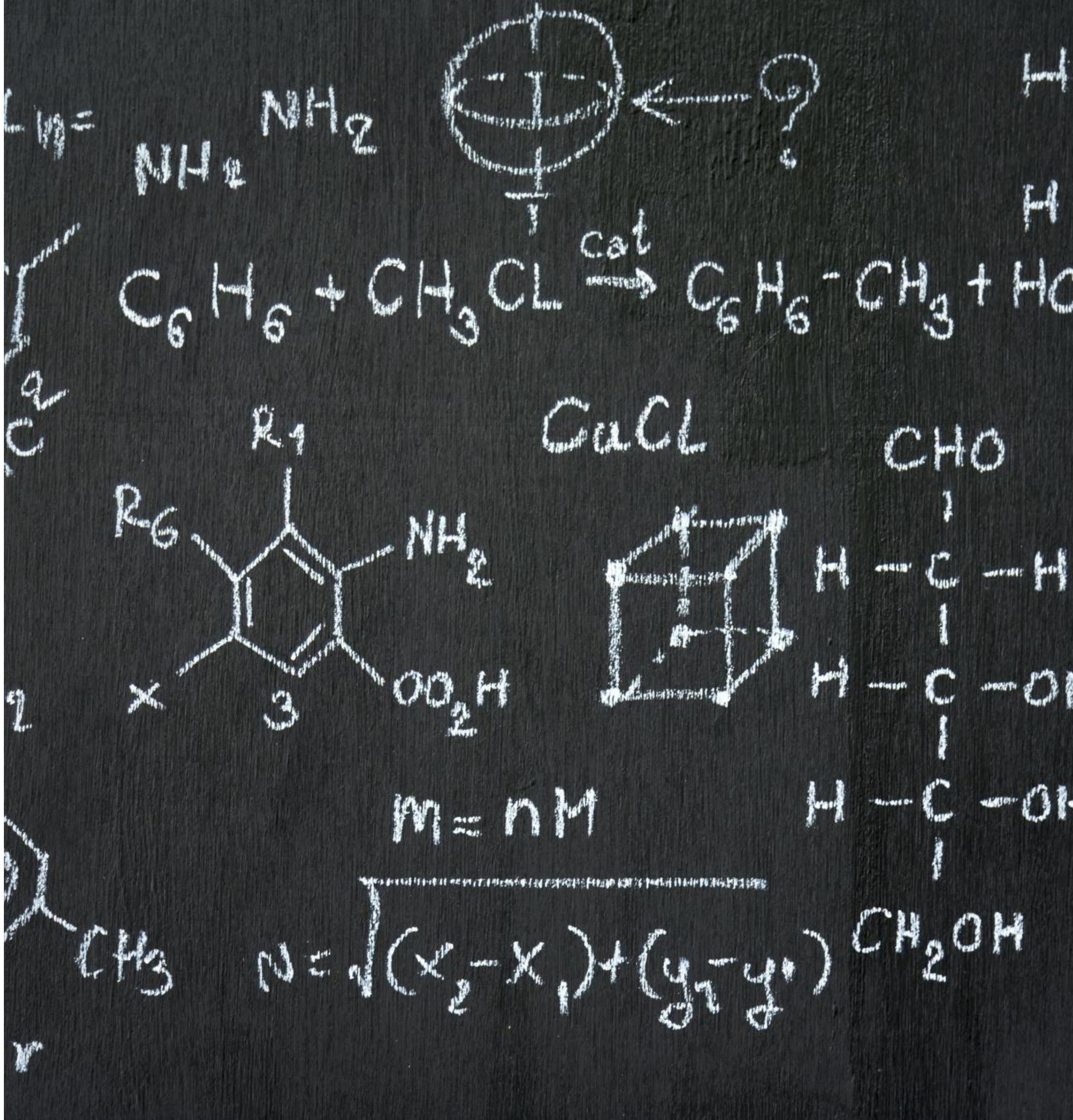


Propagación hacia atrás



Volvamos a los pasos de construir una red neuronal

1. Forward pass
2. Cálculo de pérdida
3. **Propagación hacia atrás**
4. Optimización



¿Qué es la retropropagación o propagación hacia atrás?

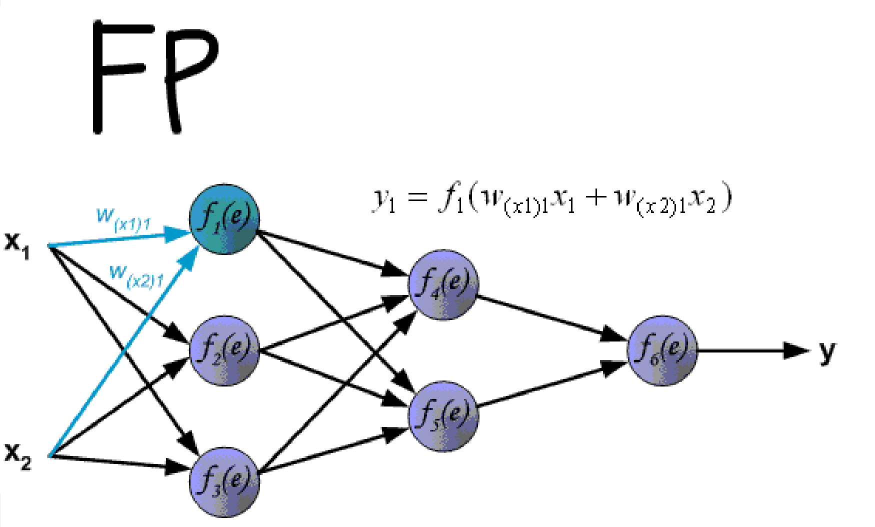
- La retropropagación es una parte esencial del entrenamiento de redes neuronales, que permite que aprendan de los conjuntos de datos de entrenamiento y **mejoren con el tiempo**.
- La retropropagación y el **descenso de gradiente** describen el proceso de mejorar los pesos y sesgos de la red para realizar mejores predicciones.
- Se trata de un sistema de **retroalimentación** en el que, después de cada ronda de entrenamiento o *epoch*, la red calcula la diferencia entre su resultado y la respuesta correcta, conocida como **error**. Luego, ajusta sus parámetros internos, o pesos, para reducir este error en el próximo *epoch*.

Conceptos importantes: Epoch y batch

- **Epoch/periodo:** es un **único** paso por **todo** el conjunto de datos de entrenamiento durante el proceso de entrenamiento.
- **Batch/lote:** En lugar de procesar todo el conjunto de datos a la vez, los datos suelen dividirse en conjuntos más pequeños denominados lotes. El modelo actualiza sus parámetros después de procesar cada lote.
- *Ejemplo:* Si tiene 1000 observaciones de entrenamiento y un tamaño de lote de 100, cada época constaría de 10 iteraciones (1000 muestras/tamaño de lote de 100). Si se entrena el modelo 5 periodos/epochs, el modelo habrá visto cada muestra en el conjunto de datos 5 veces.
- Ya veremos por qué se divide en batches los datos...

Pasos generales de la propagación hacia atrás

1. El valor de error obtenido previamente se utiliza para **calcular el gradiente** de la función de pérdida.
2. El gradiente del error se **propaga de vuelta** a través de la red, comenzando **desde la capa de salida hasta las capas ocultas**.
3. A medida que el gradiente del error se propaga de vuelta, **los pesos se actualizan de acuerdo con su contribución al error**. Esto implica tomar la derivada del error con respecto a cada peso, lo que indica cuánto un cambio en el peso afectaría al error.



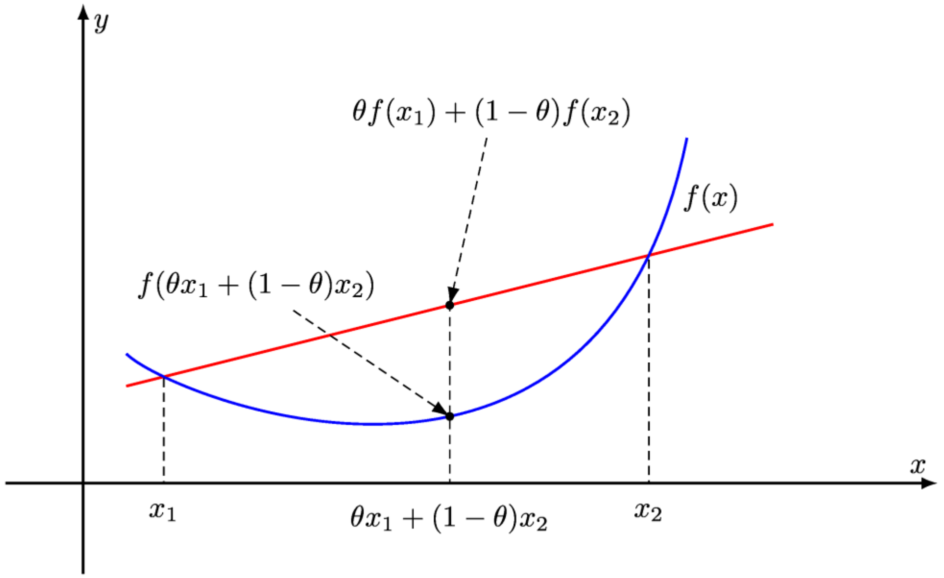
Función de pérdida convexa vs no convexa

- Una función de pérdida convexa tiene solo un mínimo global y ningún mínimo local, lo que facilita su resolución con un algoritmo de optimización más simple. Sin embargo, una función de pérdida no convexa tiene mínimos tanto locales como globales y requiere un algoritmo de optimización avanzado para encontrar el mínimo global.
- Una función f(x) es convexa si para dos puntos cualesquiera x1 y x2 en el dominio de f(x), y para cualquier “t” en el rango [0,1], se cumple la siguiente condición:

$$f(tx1 + (1\text{-}t)x2) \leq tf(x1) + (1\text{-}t)f(x2)$$

Donde $t \in [0,1]$

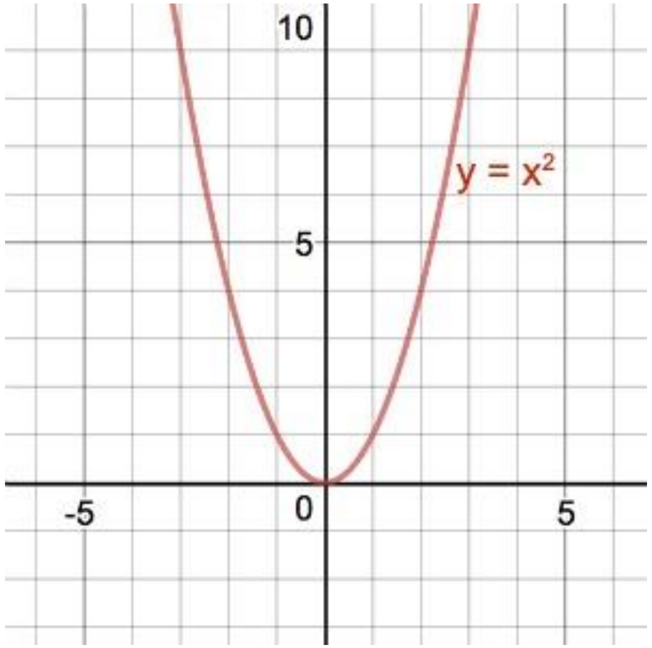
- En términos más simples, el segmento de línea entre dos puntos cualesquiera en el gráfico de la función se encuentra por encima o sobre el gráfico de la función, y no por debajo de él.



Función de pérdida convexa vs no convexa

- Ejemplo x^2

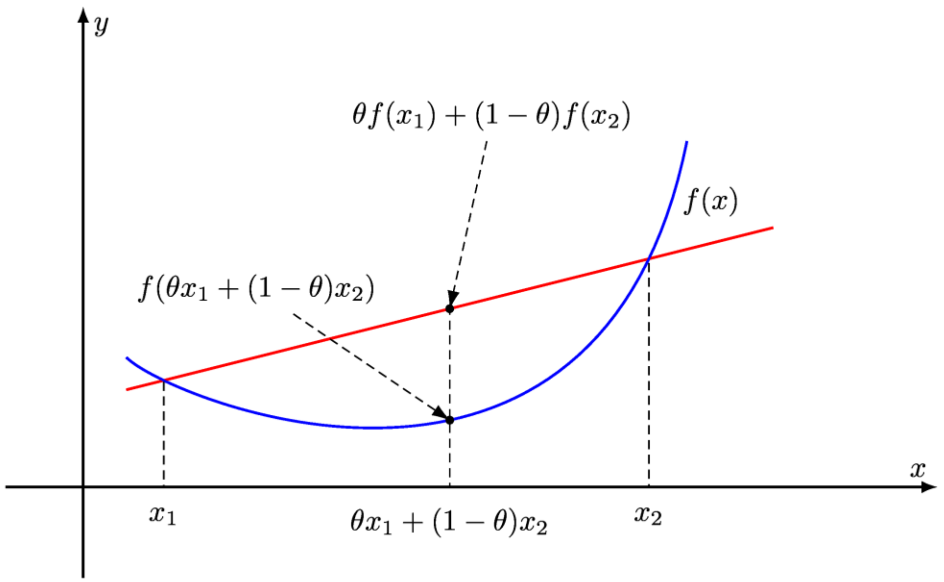
$$f(tx1 + (1\text{-}t)x2) \leq tf(x1) + (1\text{-}t)f(x2)$$



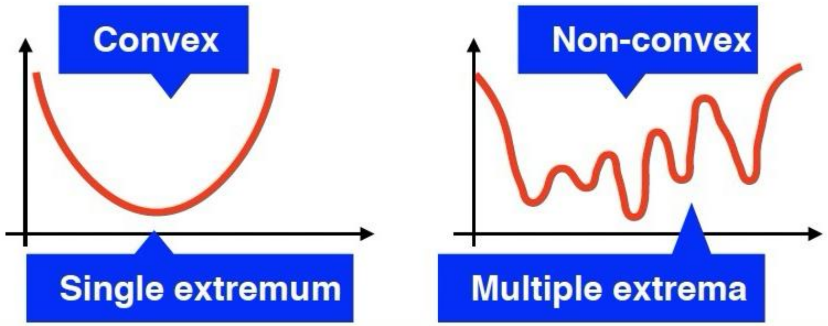
t = 0.5
$(0.5*x1 + 0.5*x2)^2$ <= $0.5*x1^2 + 0.5*x2^2$
$0.25\,(x1+x2)^2$ <= $0.5*(x1^2 + x2^2)$

x1=1 y x2 = 2

$0.25\,(3)^2$ <= 0.5*(1 + 4)
2.25 <= 2.5



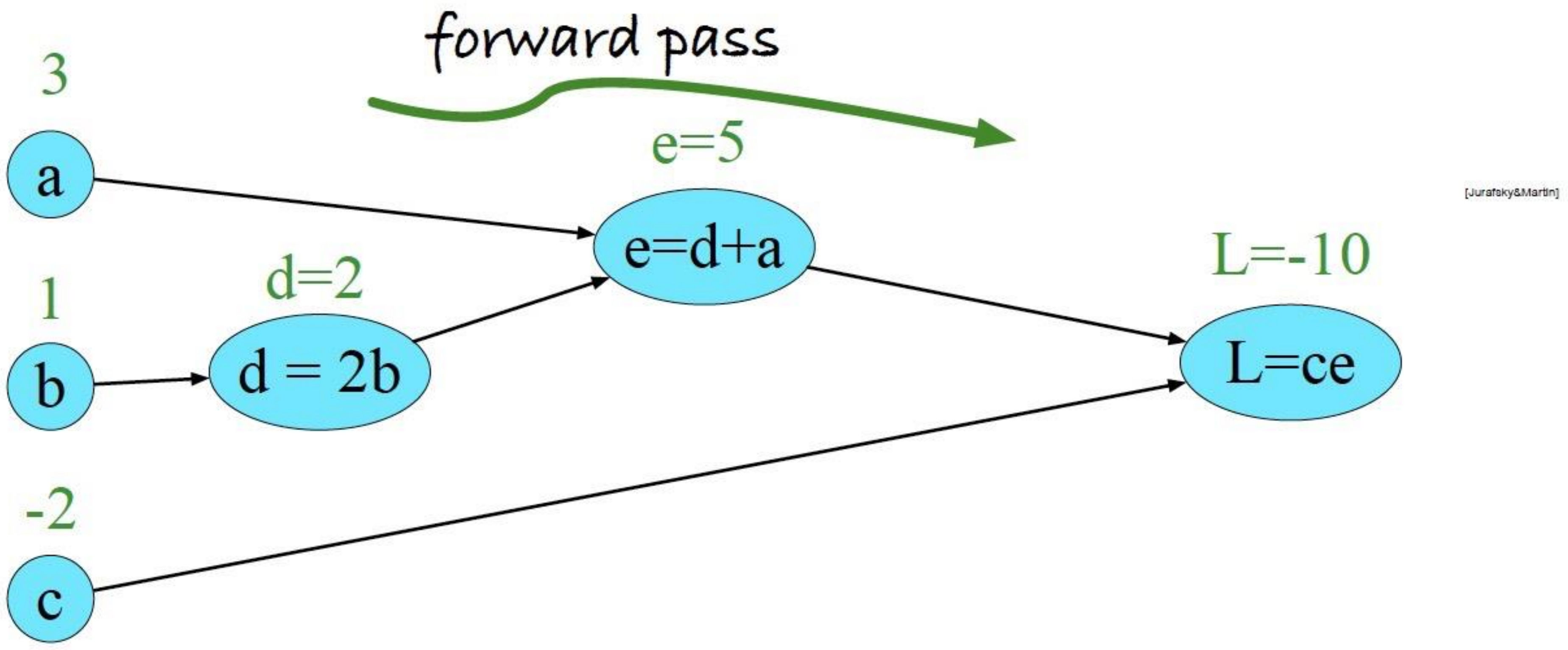
Para redes neuronales, las funciones son no convexas



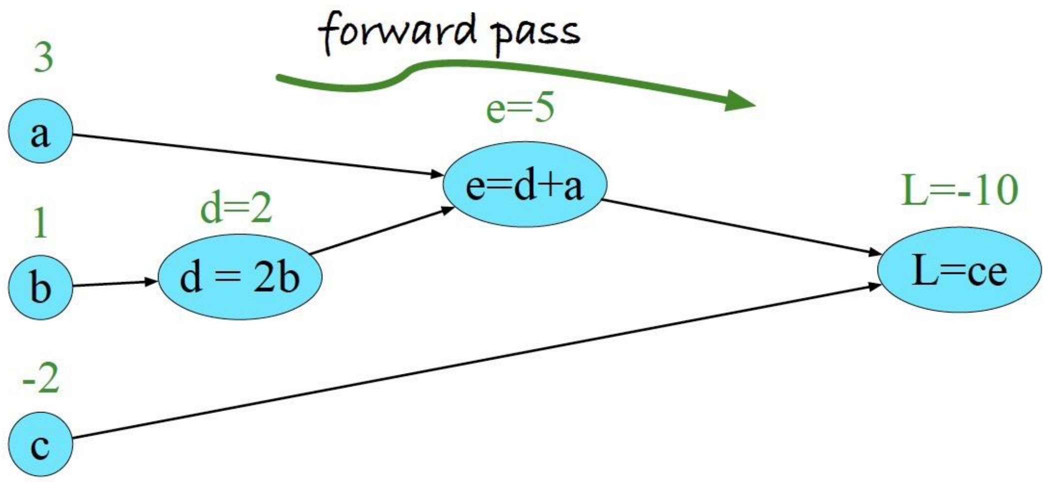
$$E(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \ln(h_\Theta(x^{(i)}))_k + \left(1-y_k^{(i)}\right)\ln\left(1-(h_\Theta(x^{(i)}))_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

- Cuando las funciones son **convexas**, se **garantiza** que los algoritmos de optimización como el descenso de gradiente encontrarán la **mejor solución posible**, en lugar de quedarse estancados en puntos subóptimos.
- Las redes neuronales, son inherentemente no convexas. El panorama de pérdidas en las redes neuronales es sumamente complejo, con muchos **mínimos locales, puntos de silla y regiones planas**.
- **En la práctica**, aunque no son convexas, **técnicas** como el descenso de gradiente estocástico **(SGD)** con momento, tasas de aprendizaje adaptativo **(Adam, RMSprop)** y la **normalización** por lotes ayudan a navegar por el complejo panorama de pérdidas **de manera efectiva**.

Ejemplo de propagación hacia atrás con caso simple



Ejemplo de propagación hacia atrás con caso simple



[Jurafsky&Martin]

Regla de la cadena para derivadas

$y = f(g(x))$

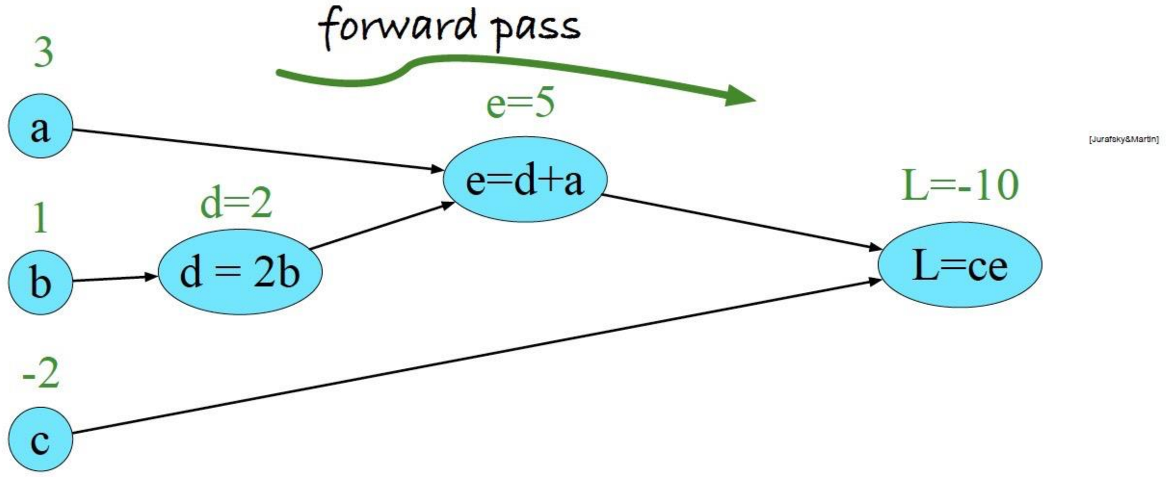
$\frac{\partial y}{\partial x} = f'(g(x)) * g'(x)$

$\frac{\partial L}{\partial a} = ?$

$\frac{\partial L}{\partial b} = ?$

$\frac{\partial L}{\partial c} = ?$

Ejemplo de propagación hacia atrás con caso simple

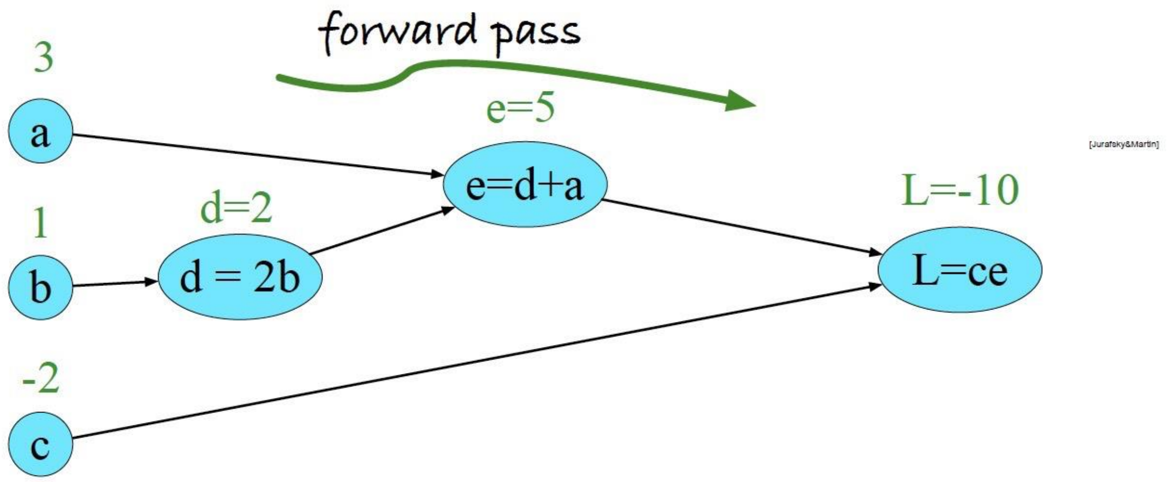


$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial c}$$

Ejemplo de propagación hacia atrás con caso simple



$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} = c$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = 2c$$

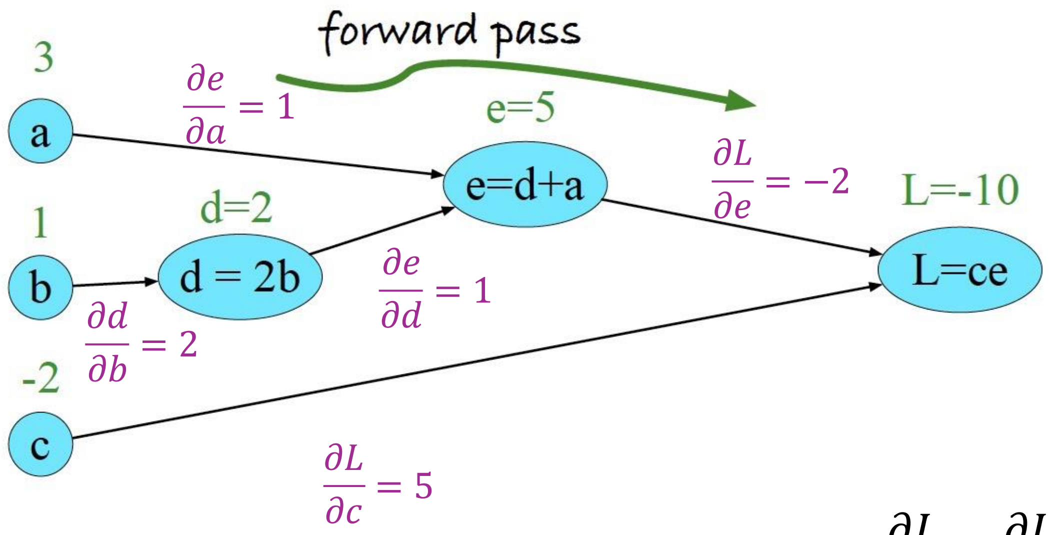
$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial c} = e$$

Ejemplo de propagación hacia atrás con caso simple

$$\frac{\partial L}{\partial a} = -2 * 1 = -2$$

$$\frac{\partial L}{\partial b} = -2 * 1 * 2 = -4$$

$$\frac{\partial L}{\partial c} = 5$$



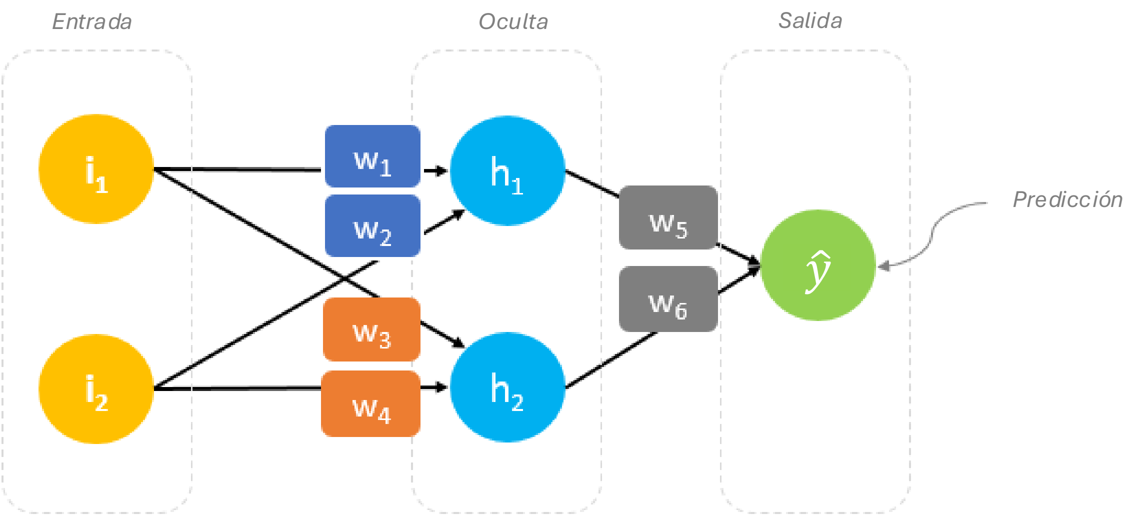
[Jurafsky&Martin]

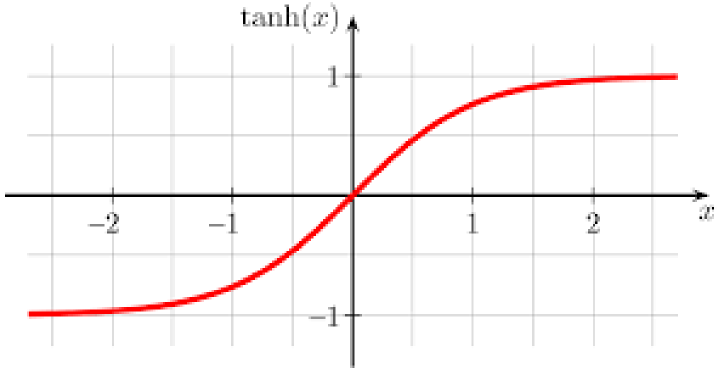
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} = c$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b} = 2c$$

$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial c} = e$$

Ejemplo con red neuronal

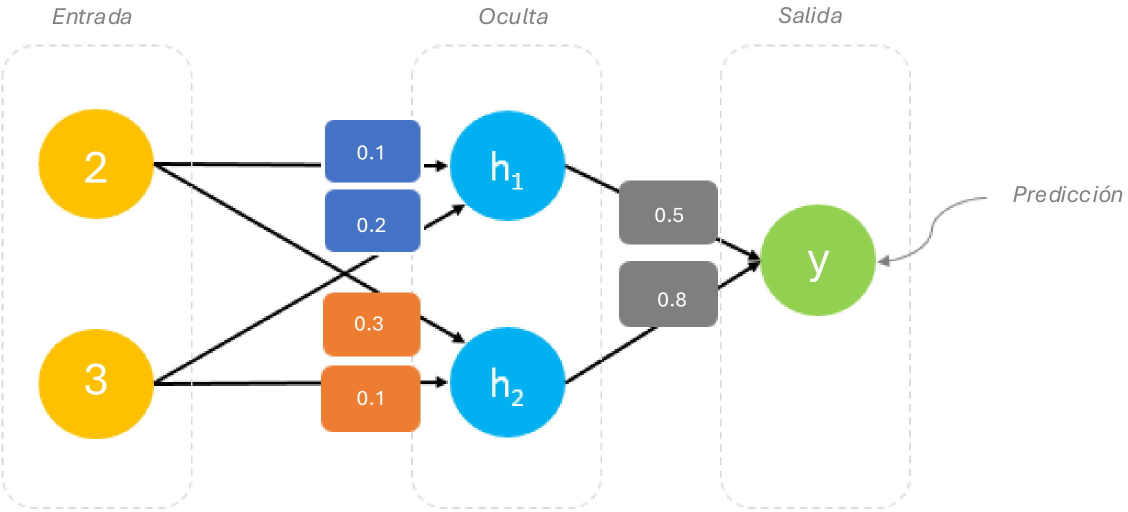




$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

- Asumamos un caso de regresión
- Con una función de pérdida para observación n: $E = \frac{1}{2}(\hat{y} - y)^2$ que la función total sería $E(w) = \sum_{i=1}^{N} E_i$
- Donde y es el valor real y $\hat{y} = \sum_i w_{ki} x_i$ el predicho
- La función de activación de la capa oculta es **tanh** (función tangente hiperbólica)
- En la última capa, al ser regresión, no aplicamos ninguna activación

Ejemplo con red neuronal



- Tenemos una observación con valores de entrada 2 y 3 y un valor de salida de 1.
- Y tenemos los pesos iniciales indicados en el gráfico
- Asumamos que no hay interceptos

Propagación hacia delante:

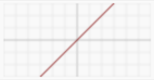
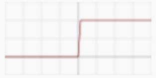
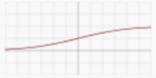
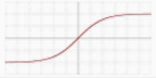
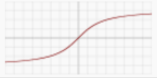
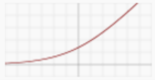
- h_1 = tanh(2*0.1 + 3*0.2) = tanh(0.8) = 0.66
- h_2 = tanh(2*0.3 + 3*0.1) = tanh(0.9) = 0.72
- y = 0.5*0.66 + 0.4 * 0.72 = 0.91

Cálculo de error:

- $\frac{1}{2}(0.91 - 1)^2 = 0.00045$

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Derivadas de funciones de activación comunes

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

$$p(y = j) = \frac{e^{x_j}}{\sum_k e^{x_k}}$$

$$\frac{\partial L}{\partial x_j} = \sum_k \frac{\partial L}{\partial p_k} \cdot \frac{\partial p_k}{\partial x_j}$$

$$\frac{\partial p_k}{\partial x_j} = \begin{cases} -p_k p_j, & \text{if k} \neq \text{j} \\ p_j - p_j^2, & \text{if k=j} \end{cases}$$

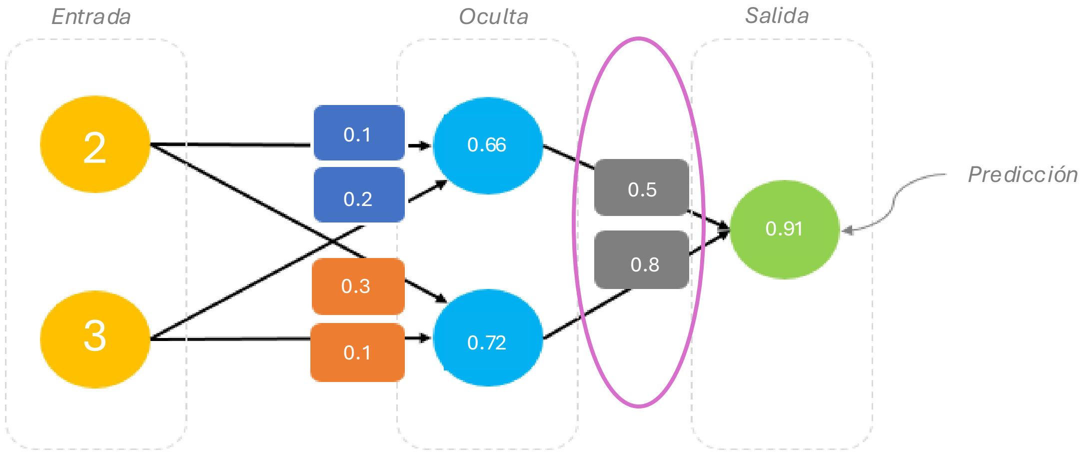
SOFTMAX

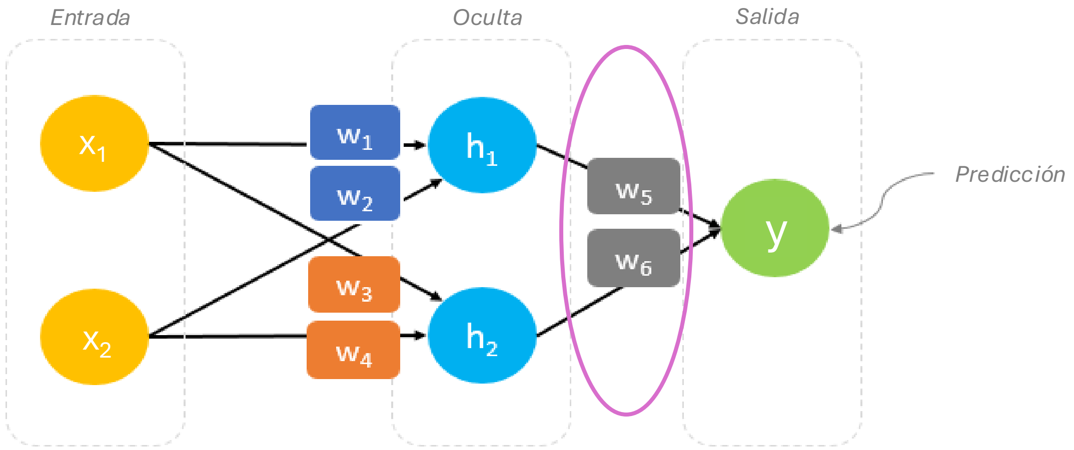
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\sigma'(x) = \frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$$

SIGMOID

Ejemplo con red neuronal - retropropagación



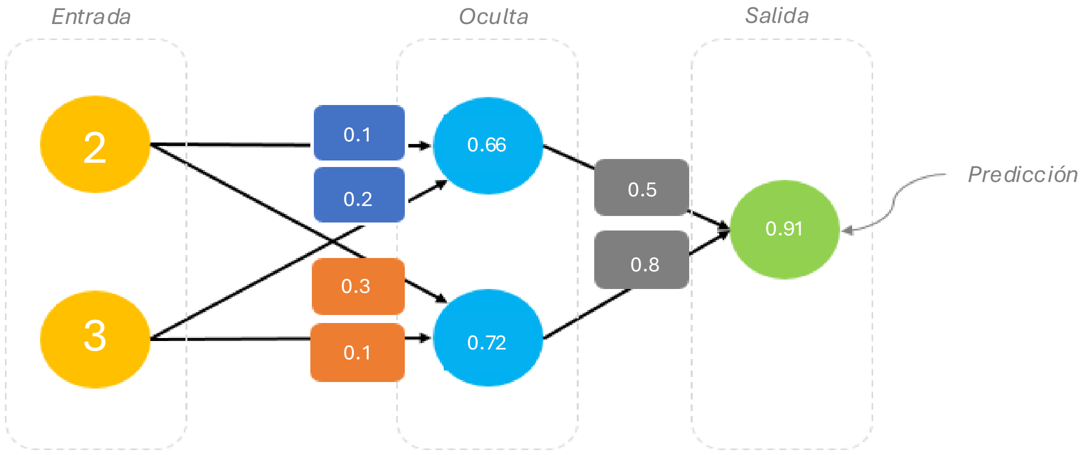


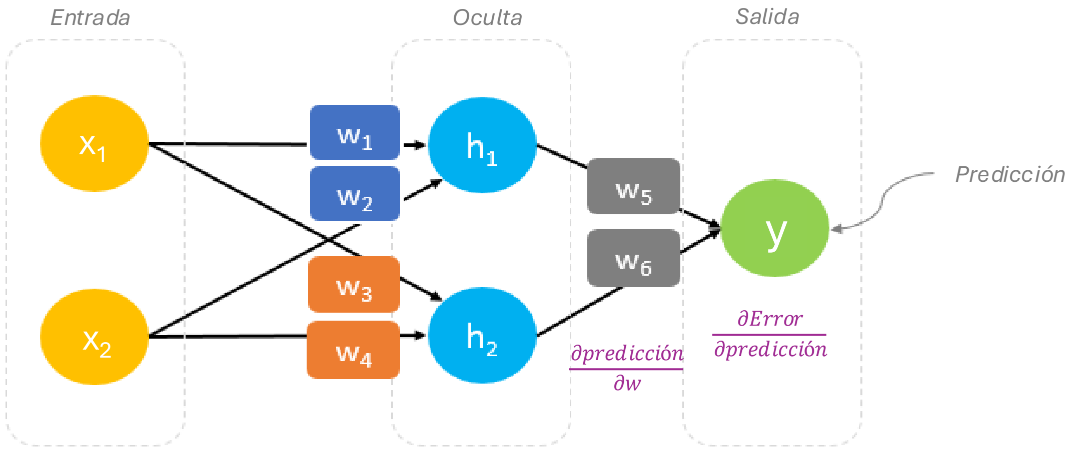
Gradientes de pesos entre capa de salida y oculta:

- $\dfrac{\partial Error}{\partial w_6} = ?$
- $\dfrac{\partial Error}{\partial w_5} = ?$

$$E = \frac{1}{2}(\hat{y} - y)^2$$
$$= \frac{1}{2}((w_5 * h_1 + w_6 * h_2) - y)^2$$

Ejemplo con red neuronal - retropropagación



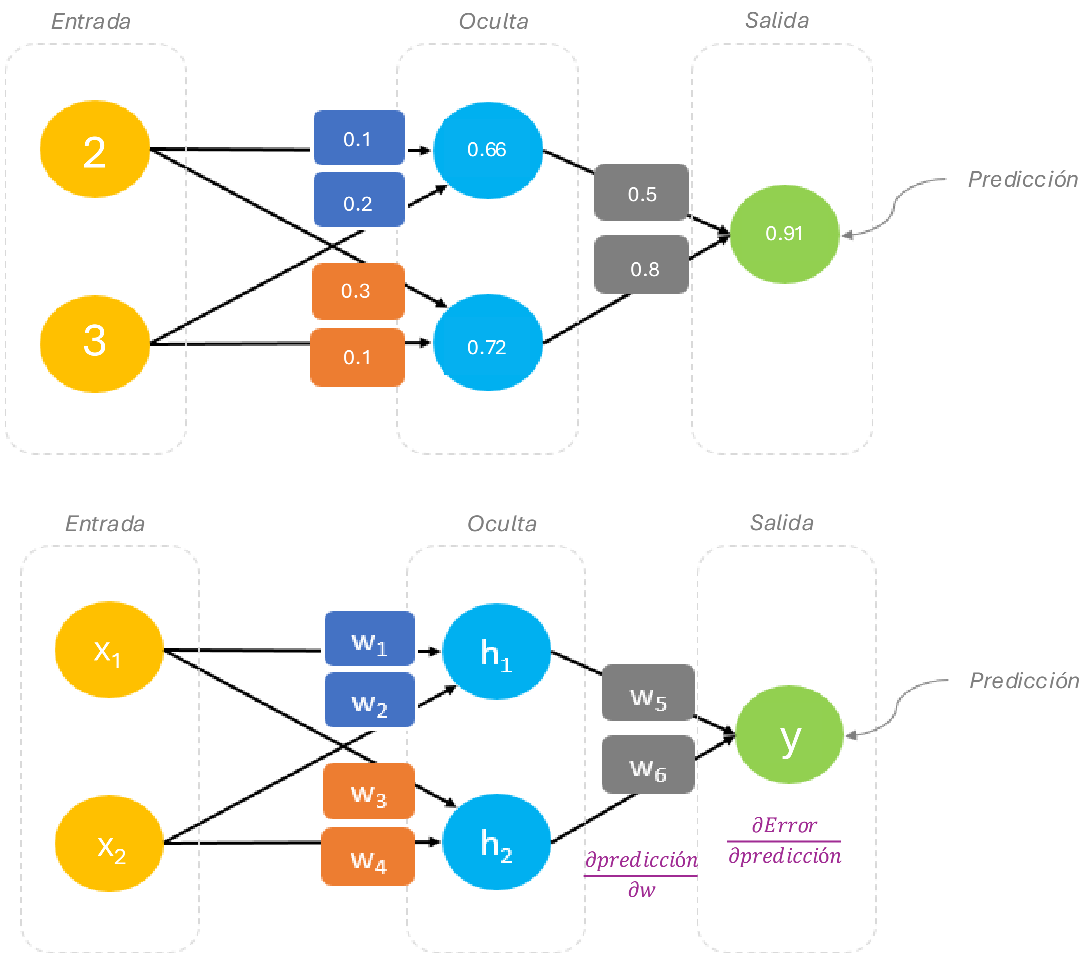


$$E = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}((w_5 * h_1 + w_6 * h_2) - y)^2$$

Gradientes de pesos entre capa de salida y oculta:

- $\frac{\partial Error}{\partial w_6} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_6}$
- $\frac{\partial Error}{\partial w_5} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_5}$

Ejemplo con red neuronal - retropropagación

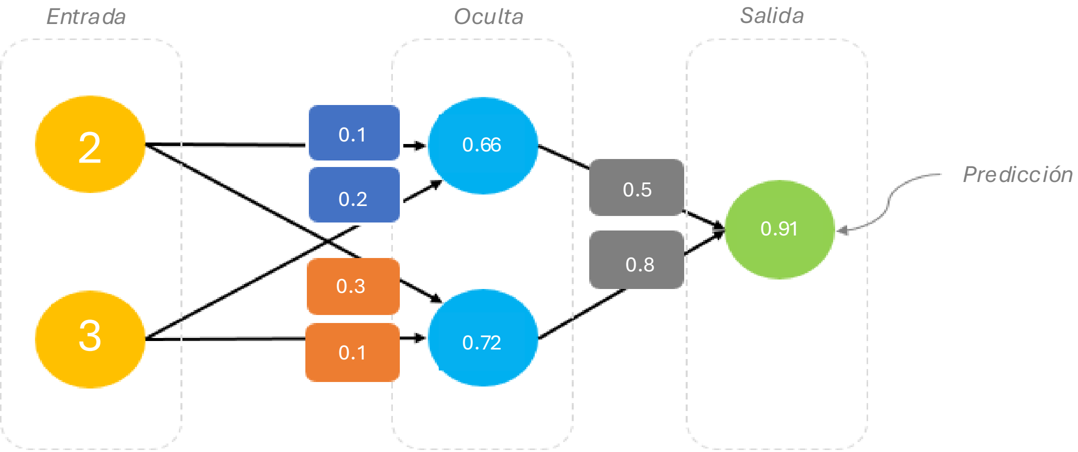


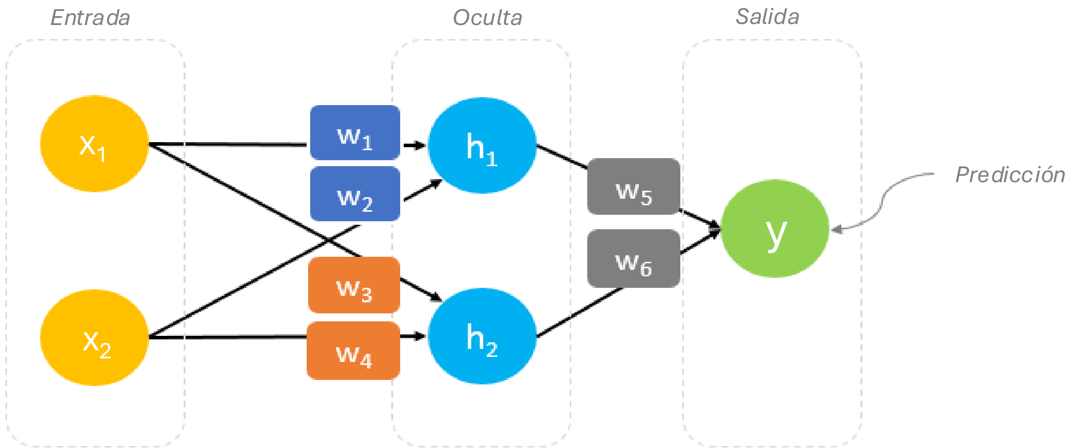
$$\boldsymbol{E = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}((w_5 * h_1 + w_6 * h_2) - y)^2}$$

Gradientes de pesos entre capa de salida y oculta:

- $\frac{\partial Error}{\partial w_6} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_6} = (\hat{y} - y) * h_2 = -0.09 * 0.72 = -0.0648$
- $\frac{\partial Error}{\partial w_5} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_5} = (\hat{y} - y) * h_1 = -0.09 * 0.66 = -0.0594$

Ejemplo con red neuronal - retropropagación





$$E = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}((w_5 * h_1 + w_6 * h_2) - y)^2$$

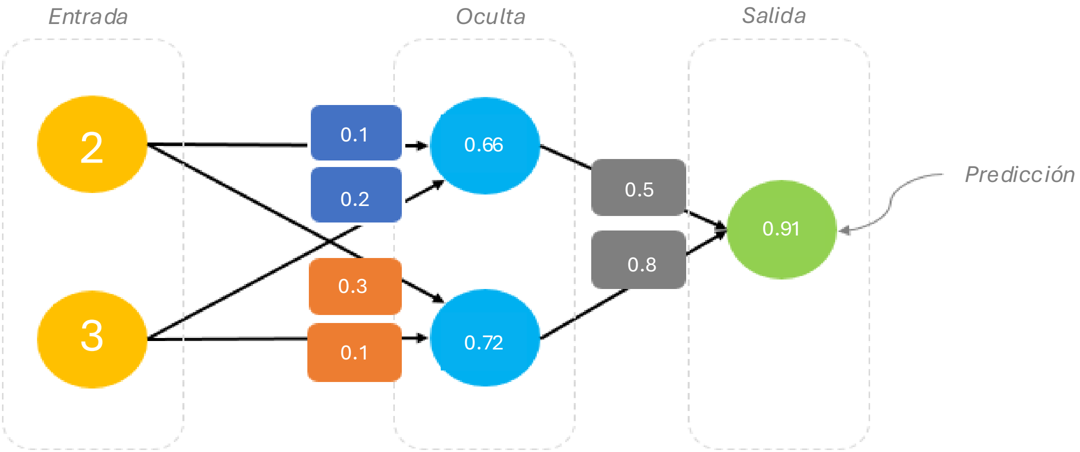
$$h_1 = tanh(a_1)$$

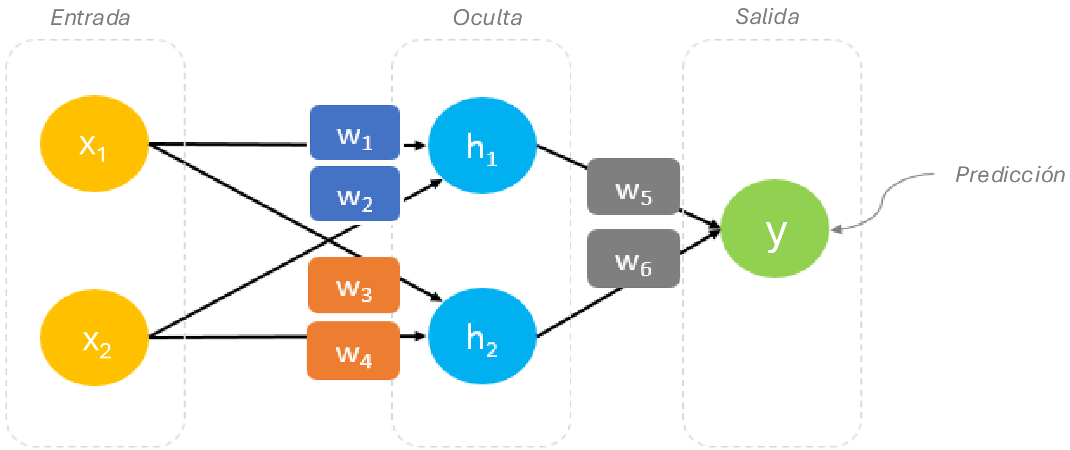
$$a_1 = \sum_{k=1}^{2} w_{jk} * x_k$$

Gradientes de pesos entre capa de salida y oculta:

- $\frac{\partial Error}{\partial w_1} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial w_1} = ?$
- $\frac{\partial Error}{\partial w_2} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial w_2} = ?$
- $\frac{\partial Error}{\partial w_3} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial w_3} = ?$
- $\frac{\partial Error}{\partial w_4} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial w_4} = ?$

Ejemplo con red neuronal - retropropagación





$$\boldsymbol{E = \frac{1}{2}(\hat{y}-y)^2 = \frac{1}{2}((w_5 * h_1 + w_6 * h_2) - y)^2}$$

$$\boldsymbol{h_1 = tanh(a_1)}$$

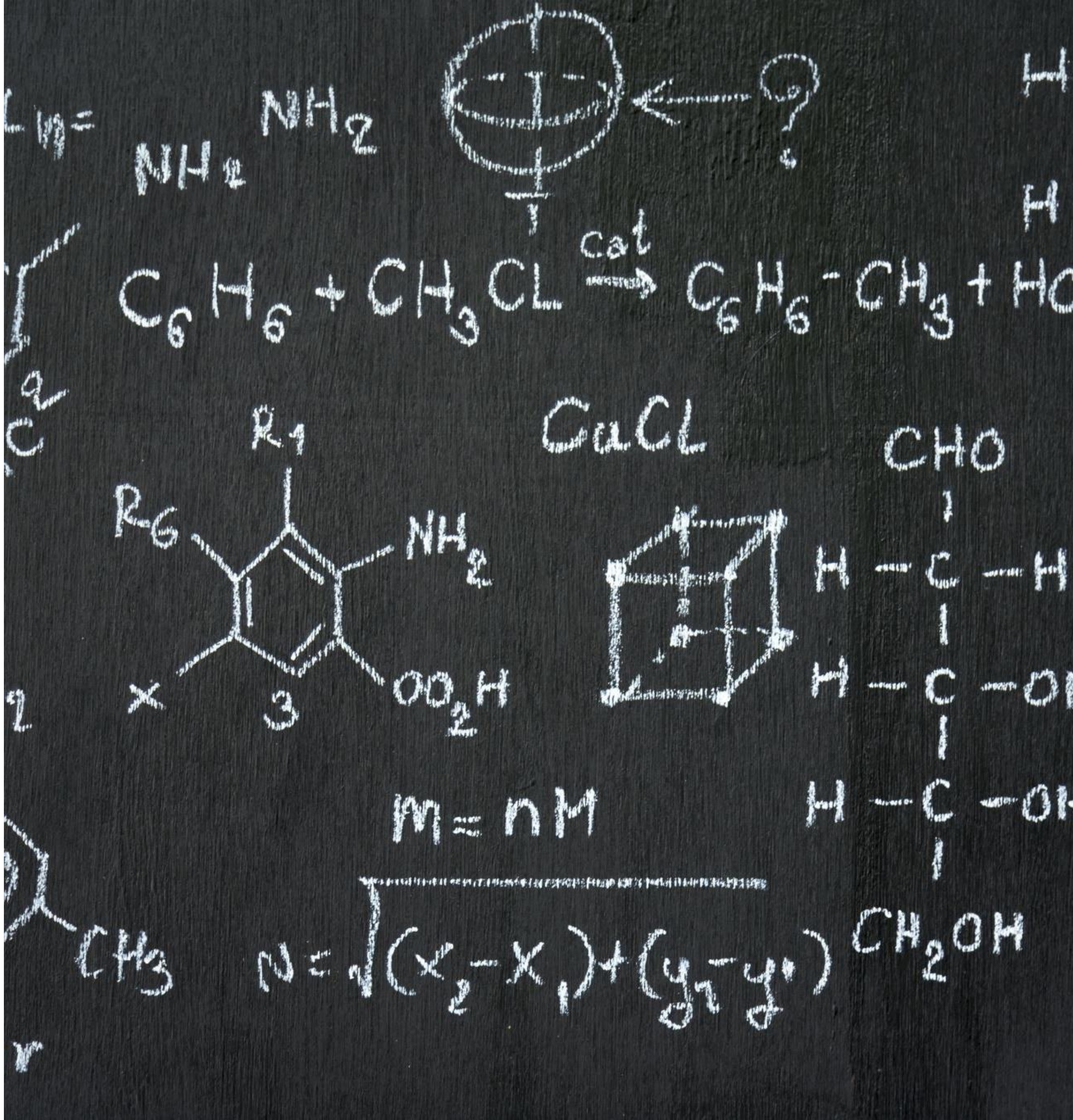
$$\boldsymbol{a_1 = \sum_{k=1}^{2} w_{jk} * x_k}$$

Gradientes de pesos entre capa de salida y oculta:

- $\frac{\partial Error}{\partial w_1} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial w_1} = \underbrace{\boxed{(\hat{y}-y) * w_5 * (1-\tanh(a_1)^2)}}_{\delta h_1} x_1 = -0.09 * 0.5 * (1-0.66^2) * 2 = \mathbf{-0.0508}$
- $\frac{\partial Error}{\partial w_2} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial w_2} = (\hat{y}-y) * w_5 * (1-\tanh(a_1)^2) x_2 = -0.09 * 0.5 * (1-0.66^2) * 3 = \mathbf{-0.0762}$
- $\frac{\partial Error}{\partial w_3} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial w_3} = (\hat{y}-y) * w_6 * (1-\tanh(a_2)^2) x_1 = -0.09 * 0.8 * (1-0.72^2) * 2 = \mathbf{-0.0693}$
- $\frac{\partial Error}{\partial w_4} = \frac{\partial Error}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_2} \frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial w_4} = (\hat{y}-y) * w_6 * (1-\tanh(a_2)^2) x_2 = -0.09 * 0.8 * (1-0.72^2) * 3 = \mathbf{-0.1040}$

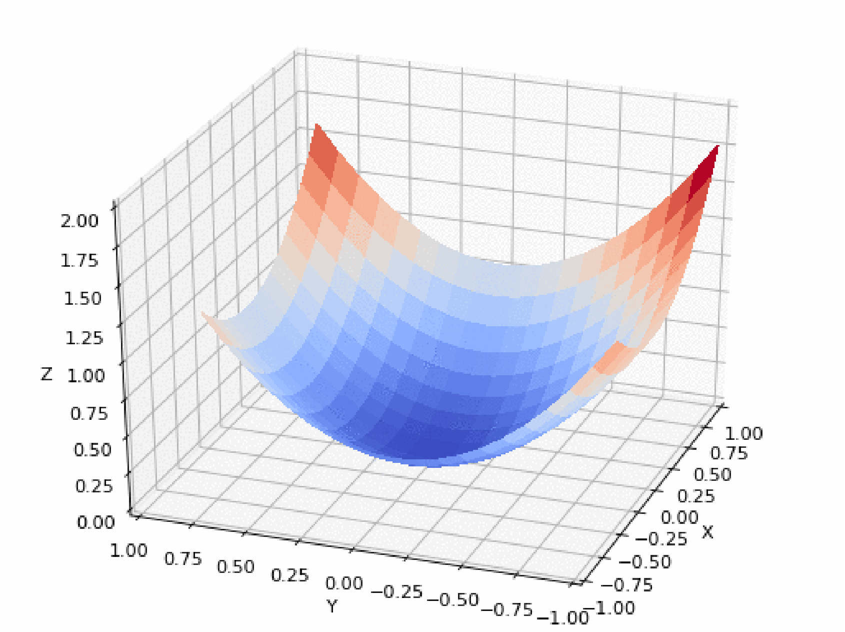
Volvamos a los pasos de construir una red neuronal

1. Forward pass
2. Cálculo de pérdida
3. Propagación hacia atrás
4. **Optimización**



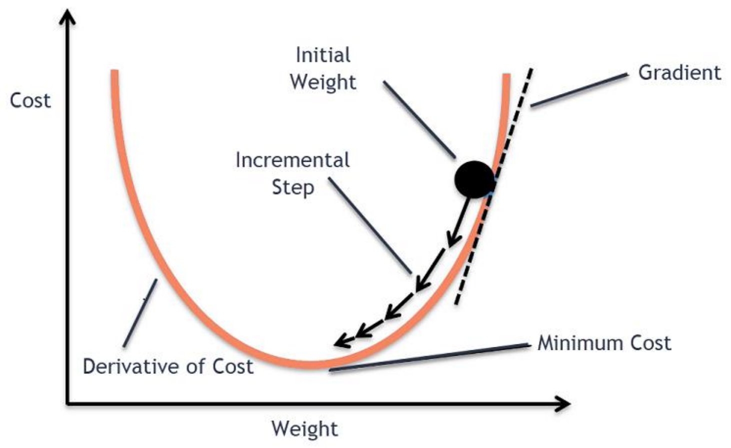
Optimización y actualización de pesos

- Una vez tenemos los gradientes de los pesos con respecto a la función de pérdida, podemos utilizarlos para actualizar los pesos de la red



Descenso de gradiente:

- El gradiente es un **vector de derivadas parciales de la función de pérdida con respecto a cada uno de los parámetros del modelo**. Apunta en la dirección del aumento más pronunciado de la función de pérdida
- Al moverse en la dirección opuesta del gradiente, los parámetros del modelo se pueden ajustar para **reducir la pérdida**. Esto da lugar al nombre de "descenso de gradiente".
- La red comienza con estimaciones iniciales de los parámetros del modelo, a menudo elegidos al azar

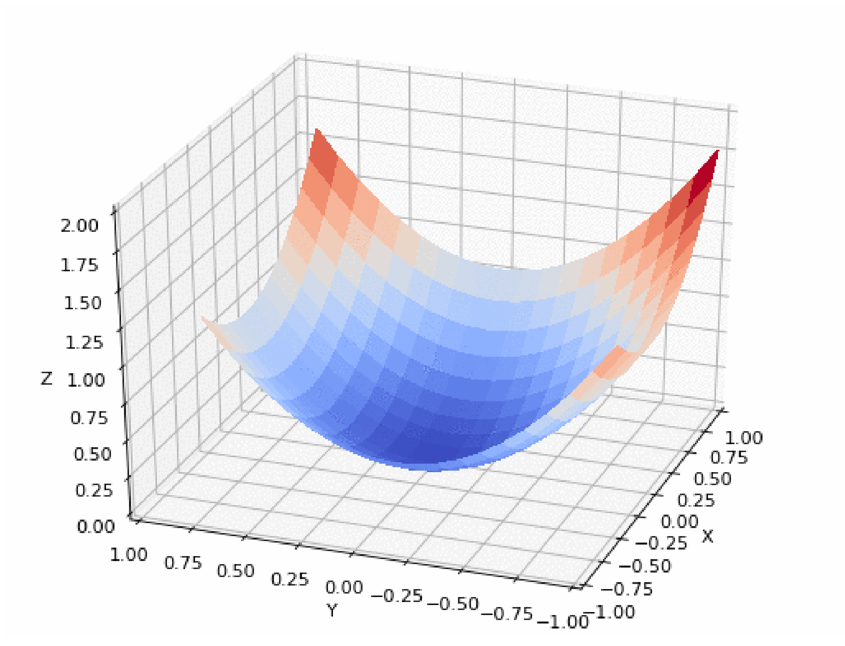


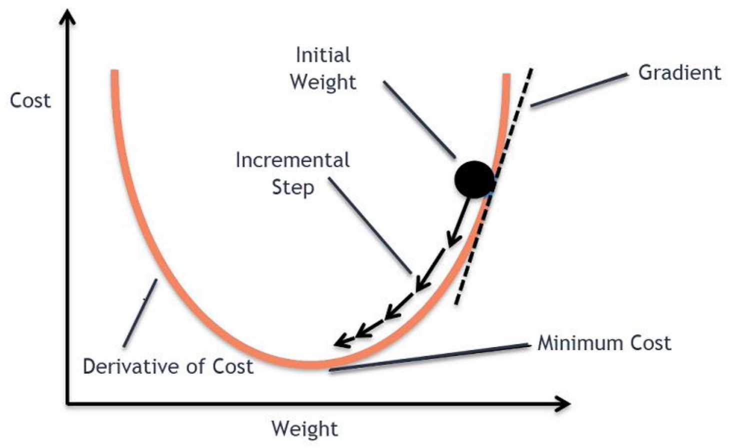
Optimización y actualización de pesos

Descenso de gradiente:

- **Tasa de aprendizaje (learning rate):** es un **hiperparámetro** que controla el **tamaño de los pasos** que se dan en la dirección del gradiente negativo.
- Una tasa de aprendizaje **pequeña** hace que el modelo converja de **forma lenta y constante**, mientras que una tasa de aprendizaje **grande** puede hacer que el modelo converja **más rápido**, pero corre el riesgo de **sobrepasar la solución óptima**
- Este proceso de paso hacia adelante, cálculo de error, propagación hacia atrás y actualización de pesos continúa durante varios periodos hasta que el rendimiento de la red alcanza un nivel satisfactorio o deja de mejorar significativamente

$$w(\tau + 1) = w(\tau) - \eta \nabla_w E$$



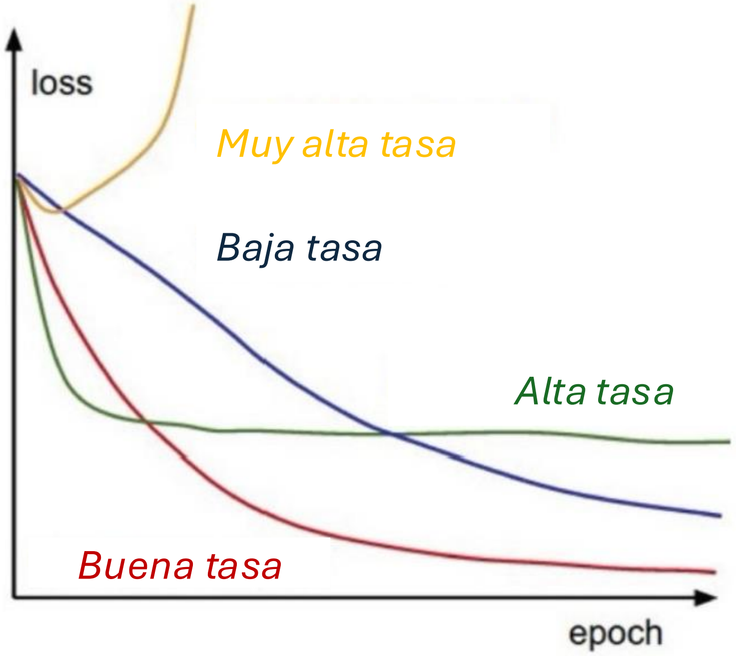


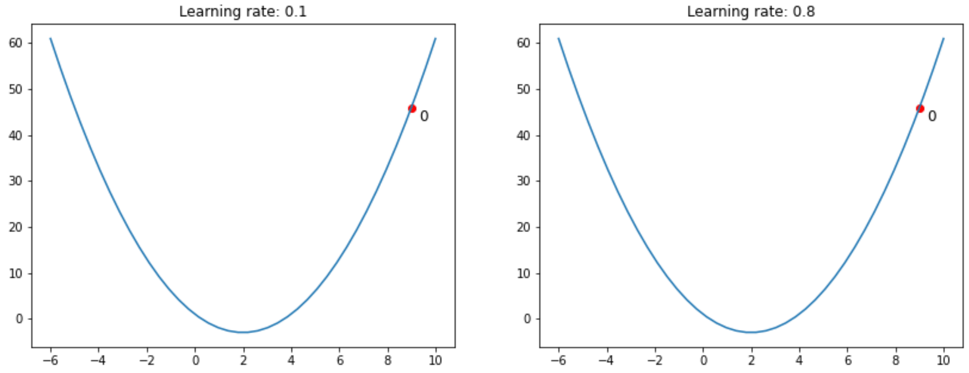
Tasa de aprendizaje

- El dilema de una buena tasa de aprendizaje:
  - Si es muy alta, el error aumenta o no mejora muy rápido
  - Si es muy bajo, el error decrece muy lentamente

$$w(\tau + 1) = w(\tau) \; - \; \eta \, \nabla_w E$$

τ es el periodo, η es la taza de aprendizaje y $\nabla_w E$ es la derivada del error con respecto al peso (w)



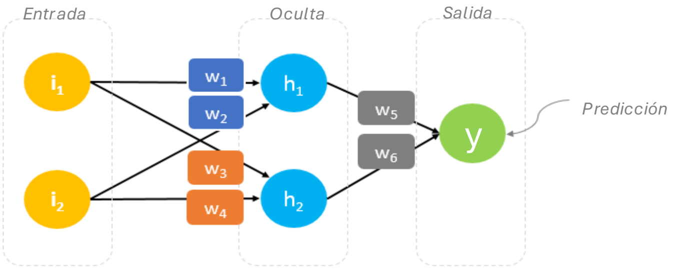


Ejemplo de cálculo

$$w(\tau + 1) = w(\tau) - \eta \nabla_w E$$

- Ejemplo anterior, con una tasa de aprendizaje de 0.5

w	w(t)	$\nabla_w E$	w(t+1)
w_1	0.1	-0.0508	0.1 + 0.5*0.0508 = 0.1254
w_2	0.2	-0.0762	0.2 + 0.5*0.0762 = 0.2381
w_3	0.3	-0.0693	0.3 + 0.5*0.0693 = 0.3346
w_4	0.1	-0.1040	0.1 + 0.5*0.1040 = 0.1520
w_5	0.5	-0.0594	0.5 + 0.5*0.0594 = 0.5297
w_6	0.8	-0.0648	0.8 + 0.5*0.0648 = 0.8324



Esto se repite por muchos epochs/periodos hasta conseguir un resultado satisfactorio en términos de error

Descenso del gradiente vs descenso del gradiente estocástico (SGD)

- En el método de descenso de gradiente estándar, la actualización de los pesos se basa en el gradiente de la función de pérdida con **respecto a todo el conjunto de datos**.
- Ventajas:
  - Proporciona una estimación más precisa del gradiente, lo que genera una convergencia más estable
  - Generalmente requiere menos iteraciones para converger
- Desventajas:
  - Puede ser muy lento, especialmente para conjuntos de datos grandes, porque requiere procesar todo el conjunto de datos para cada actualización
- Alternativa: **Descenso del Gradiente Estocástico (SGD):**
  - La actualización de los pesos se basa en el gradiente de la función de pérdida con **respecto a un solo ejemplo de entrenamiento (o, a veces, un pequeño lote/batch) al azar.**
  - **Ventajas:** Actualizaciones más rápidas, lo que puede generar una convergencia más rápida, especialmente en conjuntos de datos grandes. Pueden escapar de los mínimos locales y los puntos de silla debido al ruido de las estimaciones de gradiente, lo que a veces puede ser beneficioso.
  - **Desventajas:** El ruido de las actualizaciones puede generar fluctuaciones en torno al mínimo, lo que dificulta la convergencia exacta. Por lo general, se requieren más iteraciones para converger y la tasa de aprendizaje debe administrarse con más cuidado.

En otra clase veremos otros algoritmos y mejoras al descenso de gradiente...

Ejemplo de entrenamiento

- Tarea **moral** de práctica de los conceptos:

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/