



# DIPLOMADO Inteligencia Artificial Aplicada

3<sup>a</sup>  
Emisión

## Módulo 11: Introducción a las redes neuronales

### 1. Redes densas

Instructor: Blanca Vázquez



# Objetivo de la sesión

- Aprender las características de las redes densas, estudiar a profundidad la propagación hacia adelante y el cálculo de los gradientes, así como analizar técnicas para evitar la explosión y desvanecimiento de gradiente.



# Contenido

- 2.1. De neuronas individuales a redes de neuronas
- 2.2. Cálculo de los gradientes en redes neuronales con múltiples capas
- 2.3. Algoritmo de retropropagación de errores
- 2.4. Sobreajuste y regularización
- 2.5. El problema de la explosión y desvanecimiento del gradiente
- 2.6. Capas de normalización
- 2.7. Variantes del descenso por gradiente: RMSProp y ADAM.

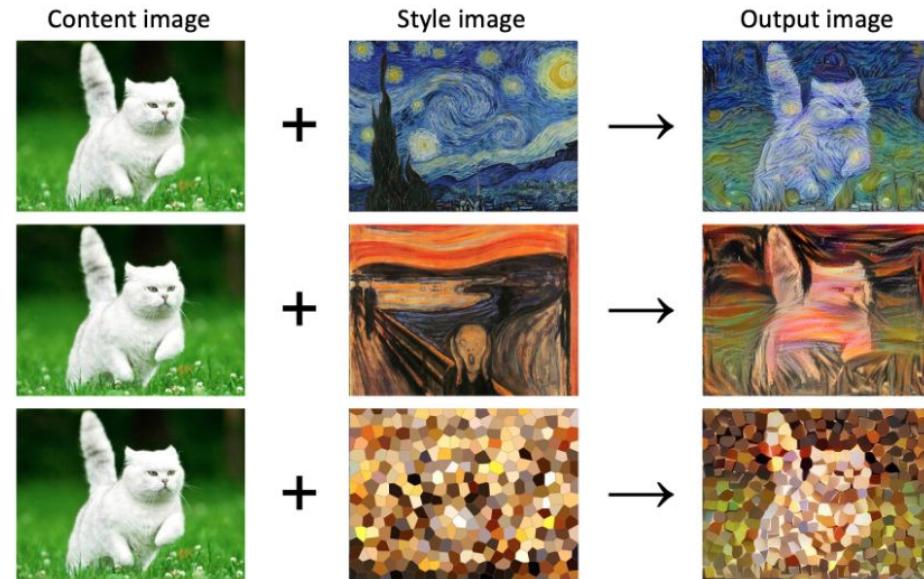


# Contexto

# Generating musical compositions



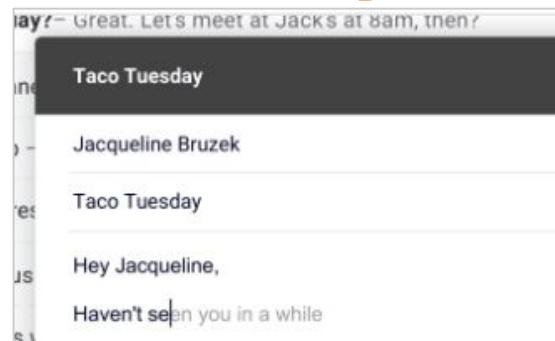
# Transfer style



Gatys, Ecker, and Bethge, A Neural Algorithm of Artistic Style, 2015.

Imagen tomada de Bertens, 2019.

# Smart compose

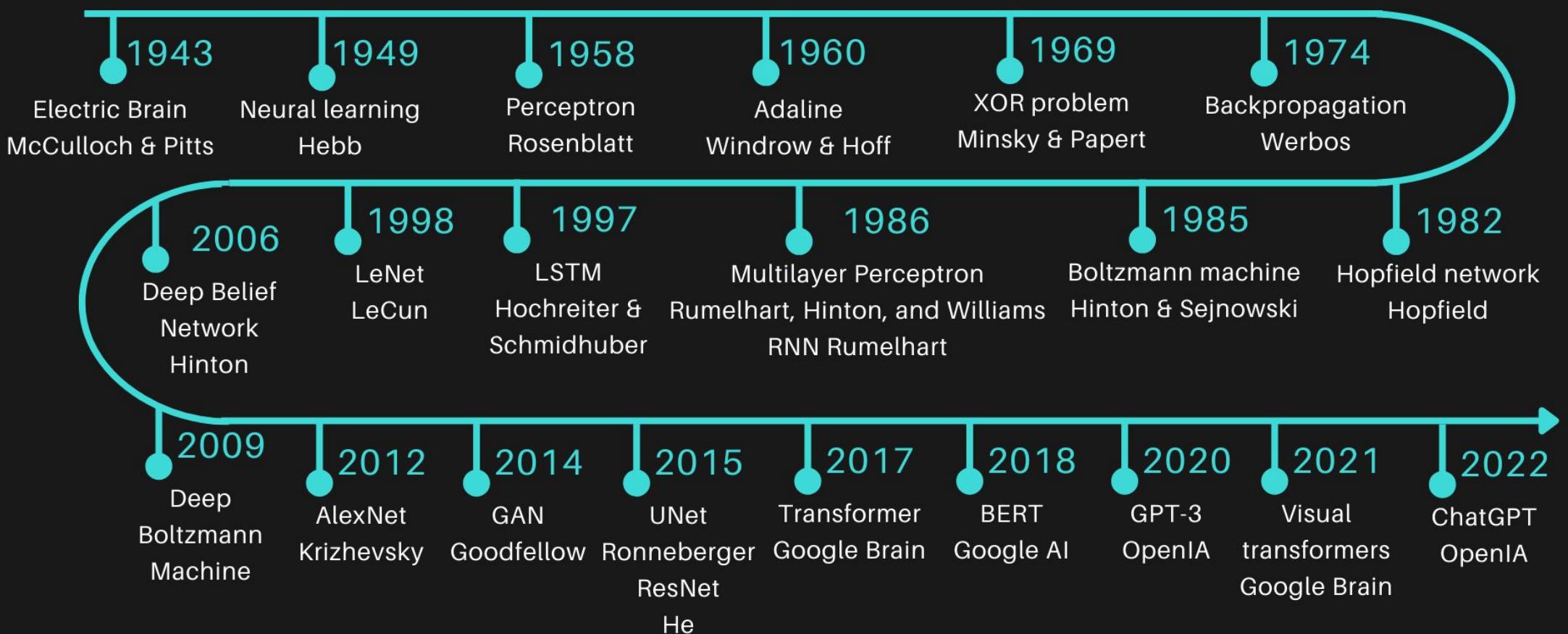


# Creating video from text



Prompt: A stylish woman walks down a Tokyo street filled with warm glowing neon and animated city signage. She wears a black leather jacket, a long red dress, and black boots, and carries a black purse. She wears sunglasses and red lipstick. She walks confidently and casually. The street is damp and reflective, creating a mirror effect of the colorful lights. Many pedestrians walk about.

## Evolución de la IA



Fuente: elaboración propia



# Red neuronal densa

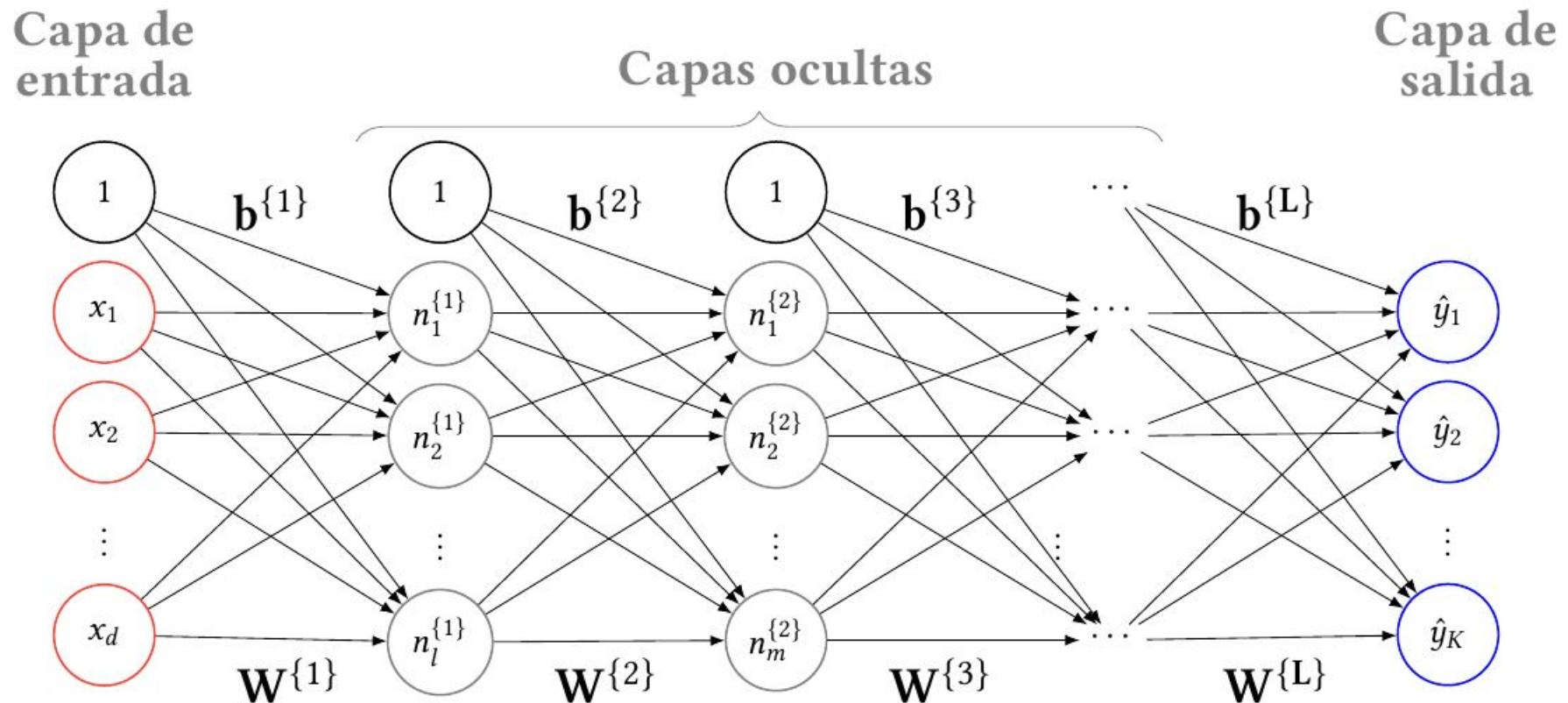


Imagen tomada de Fuentes, 2023.

# Características de las redes densas

- Son aproximadores universales<sup>1,2,3</sup> (con 1 sola capa oculta con un número finito de neuronas).
  - Pueden modelar cualquier función.
- Frecuentemente sobreparametrizados<sup>4</sup>
- Usualmente empleados como bloques de clasificación (no tan profundos) en conjunto con otros tipos de capas

<sup>1</sup> Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into Deep Learning, 2020.

<sup>2</sup> Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4), 303–314.

<sup>3</sup> Hornik et al. Multilayer Feedforward Networks are Universal Approximators, 1989.

<sup>4</sup> Allen-Zhu et al. Learning and Generalization in Overparameterized Neural Networks, Going Beyond Two Layers, 2020.



# Propagación hacia adelante

Cada neurona lleva a cabo dos operaciones:

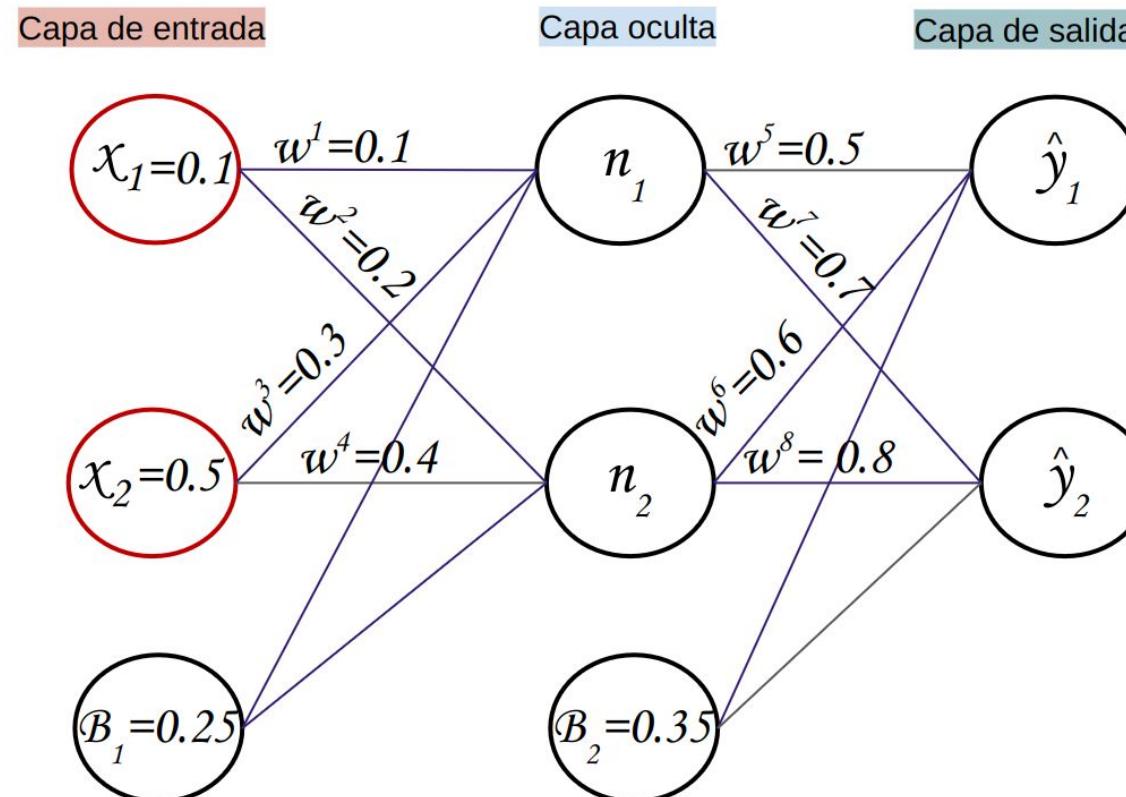
1. Realiza una suma pesada
2. Proceso la suma a través de una función de activación

En inglés, se conoce como: *forward Pass*



# Propagación hacia adelante (red densa)

Considera una red densa con 1 capa de entrada, 1 capa oculta con 2 neuronas con función de activación sigmoide y 2 neuronas de salida con función de activación lineal.



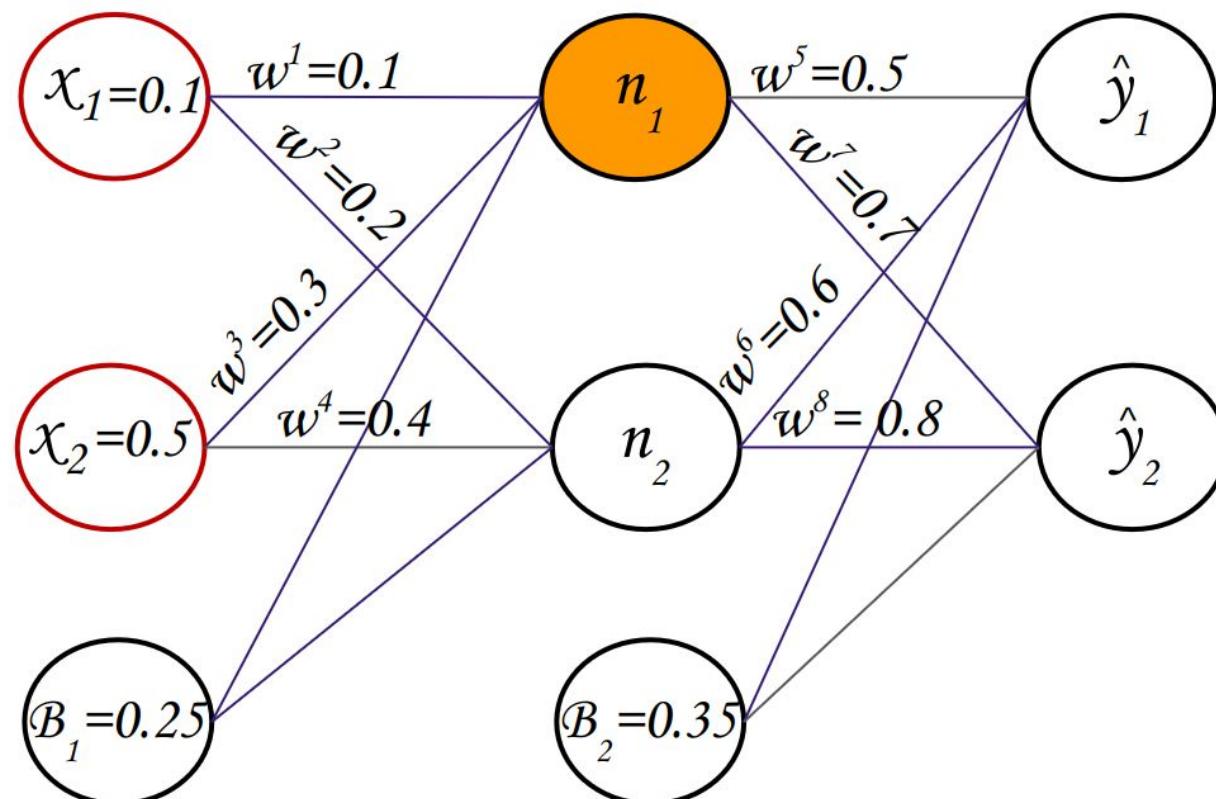
# Propagación hacia adelante

Calculando  $n_1$

Capa de entrada

Capa oculta

Capa de salida



$$sum_{n_1} = (x_1 * w^1) + (x_2 * w^3) + B_1$$

$$sum_{n_1} = (0.1 * 0.1) + (0.5 * 0.3) + 0.25 = 0.41$$

Ahora pasamos la suma pesada a la función logística:

$$output_{n_1} = \frac{1}{1 + e^{-sum_{n_1}}}$$

$$output_{n_1} = \frac{1}{1 + e^{-0.41}}$$

$$output_{n_1} = 0.60108$$



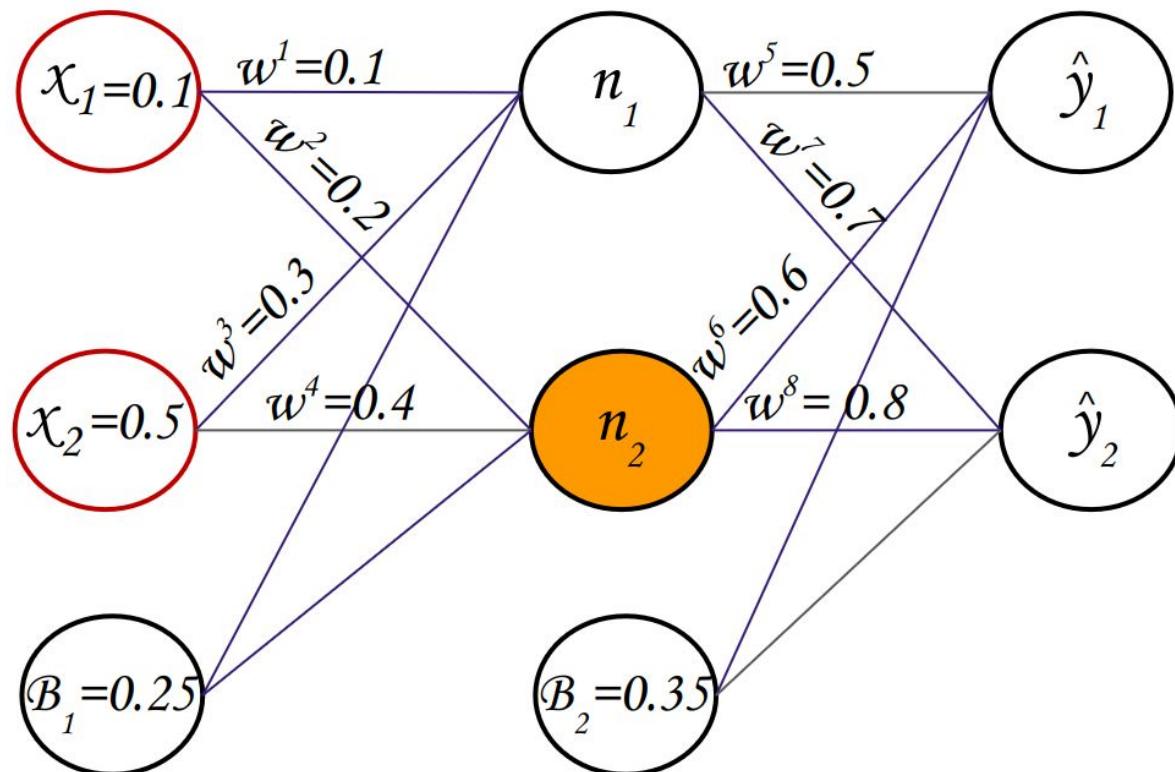
# Propagación hacia adelante

Calculando  $n_2$

Capa de entrada

Capa oculta

Capa de salida



$$sum_{n_2} =$$

$$sum_{n_2} =$$

Ahora pasamos la suma pesada a la función logística:

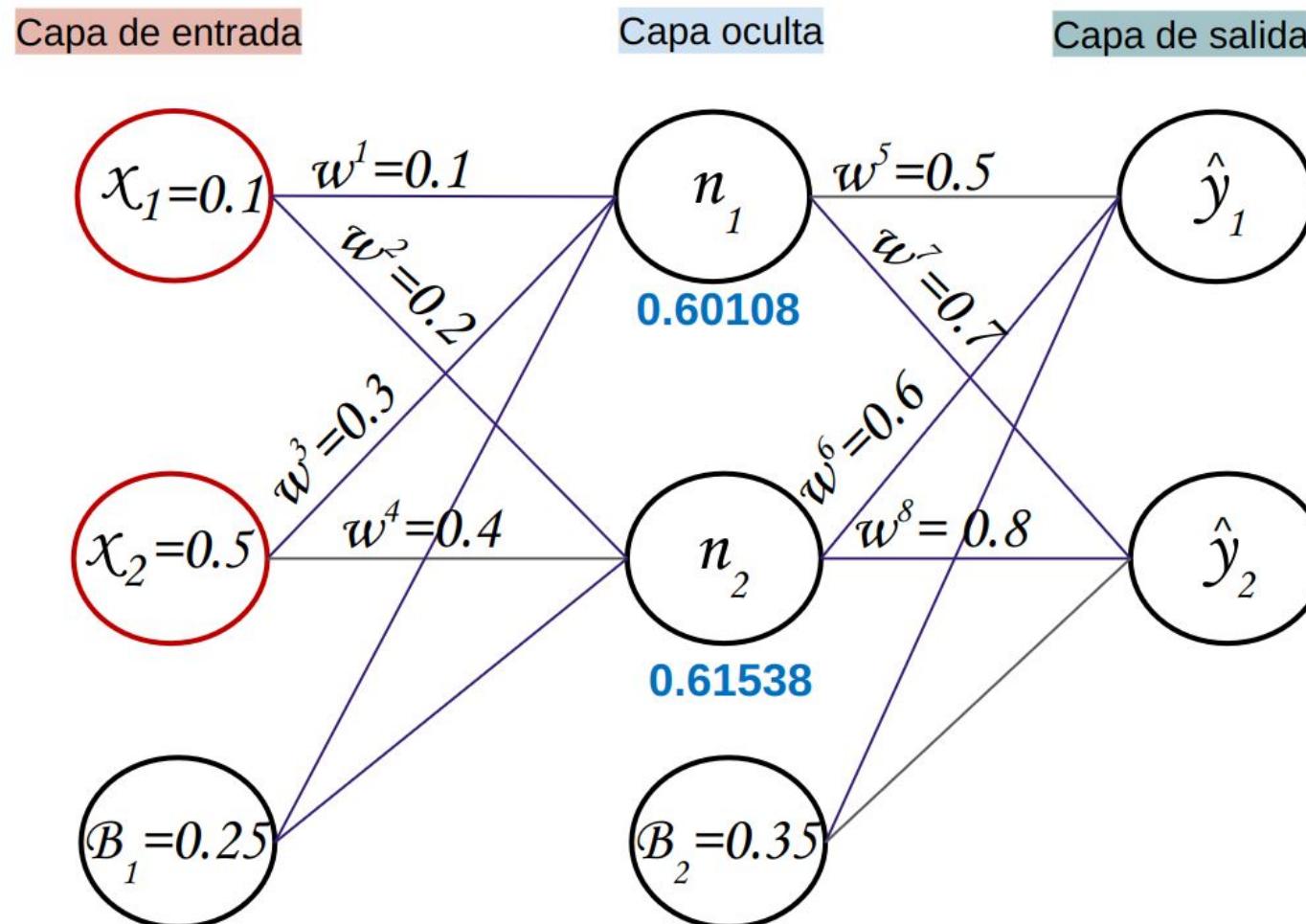
$$output_{n_2} = \frac{1}{1 + e^{-sum_{n_2}}}$$

$$output_{n_2} =$$

$$output_{n_2} =$$



# Propagación hacia adelante



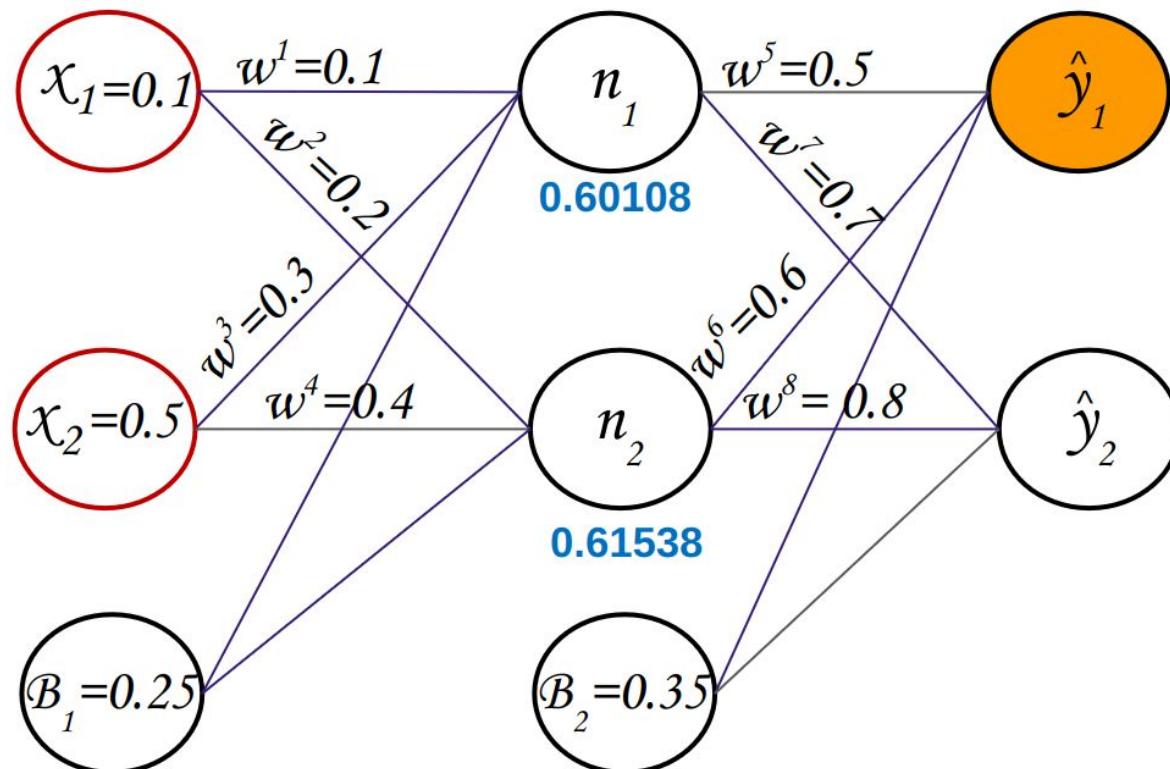
# Propagación hacia adelante

Calculando  $\hat{y}_1$

Capa de entrada

Capa oculta

Capa de salida



$$sum_{\hat{y}_1} = (n_1 * w^5) + (n_2 * w^6) + B_2$$

$$sum_{\hat{y}_1} = (0.60108 * 0.5) + (0.61538 * 0.6) + 0.35 = 1.01977$$

Ahora pasamos la suma pesada a la función logística:

$$\hat{y}_1 = \frac{1}{1 + e^{-sum_{\hat{y}_1}}}$$

$$\hat{y}_1 = \frac{1}{1 + e^{-1.01977}}$$

$$\hat{y}_1 = 0.73492$$



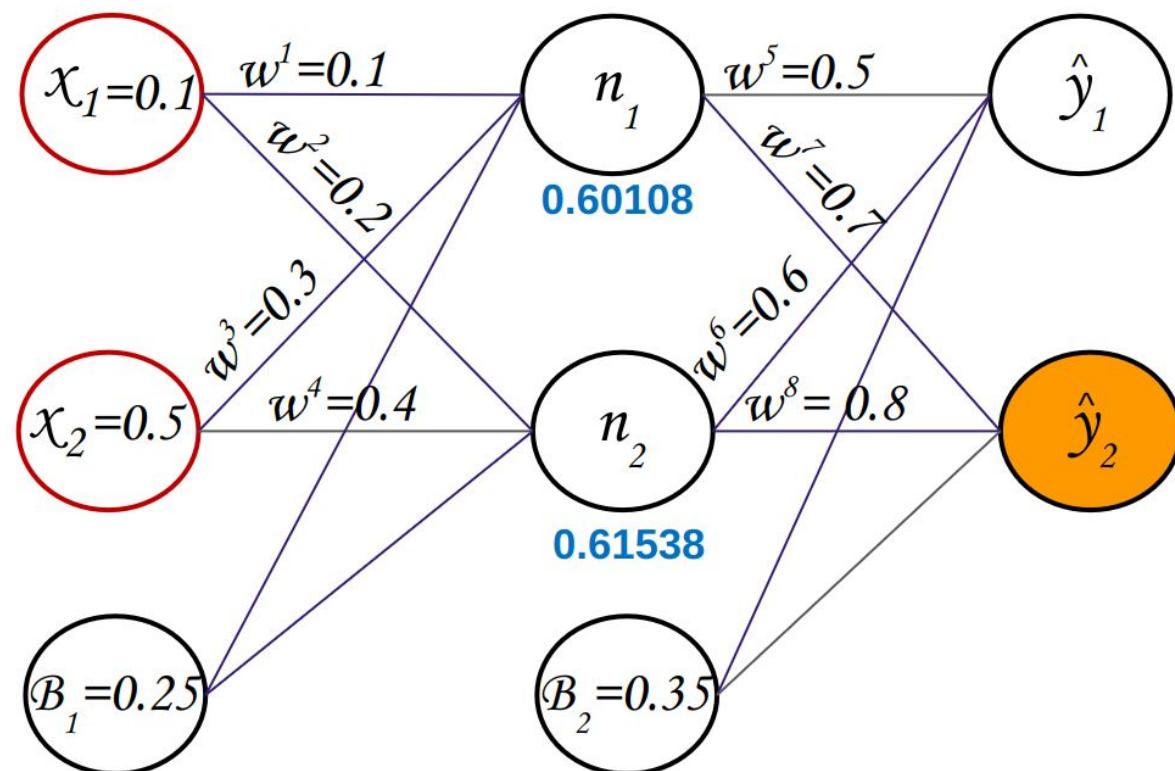
# Propagación hacia adelante

Calculando  $\hat{y}_2$

Capa de entrada

Capa oculta

Capa de salida



$$sum_{\hat{y}_2} =$$

$$sum_{\hat{y}_2} =$$

Ahora pasamos la suma pesada a la función logística:

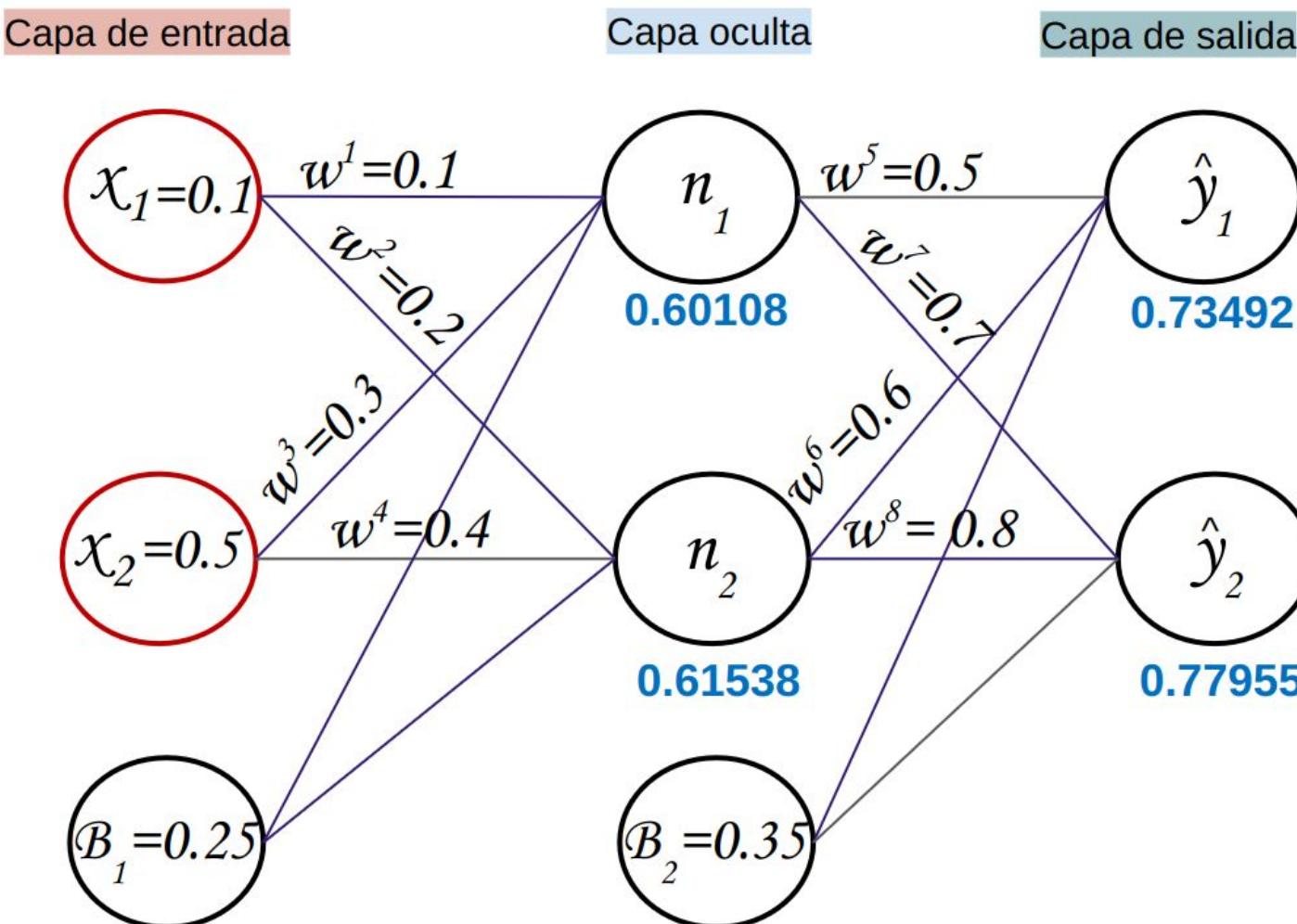
$$\hat{y}_2 = \frac{1}{1 + e^{-sum_{\hat{y}_2}}}$$

$$\hat{y}_2 =$$

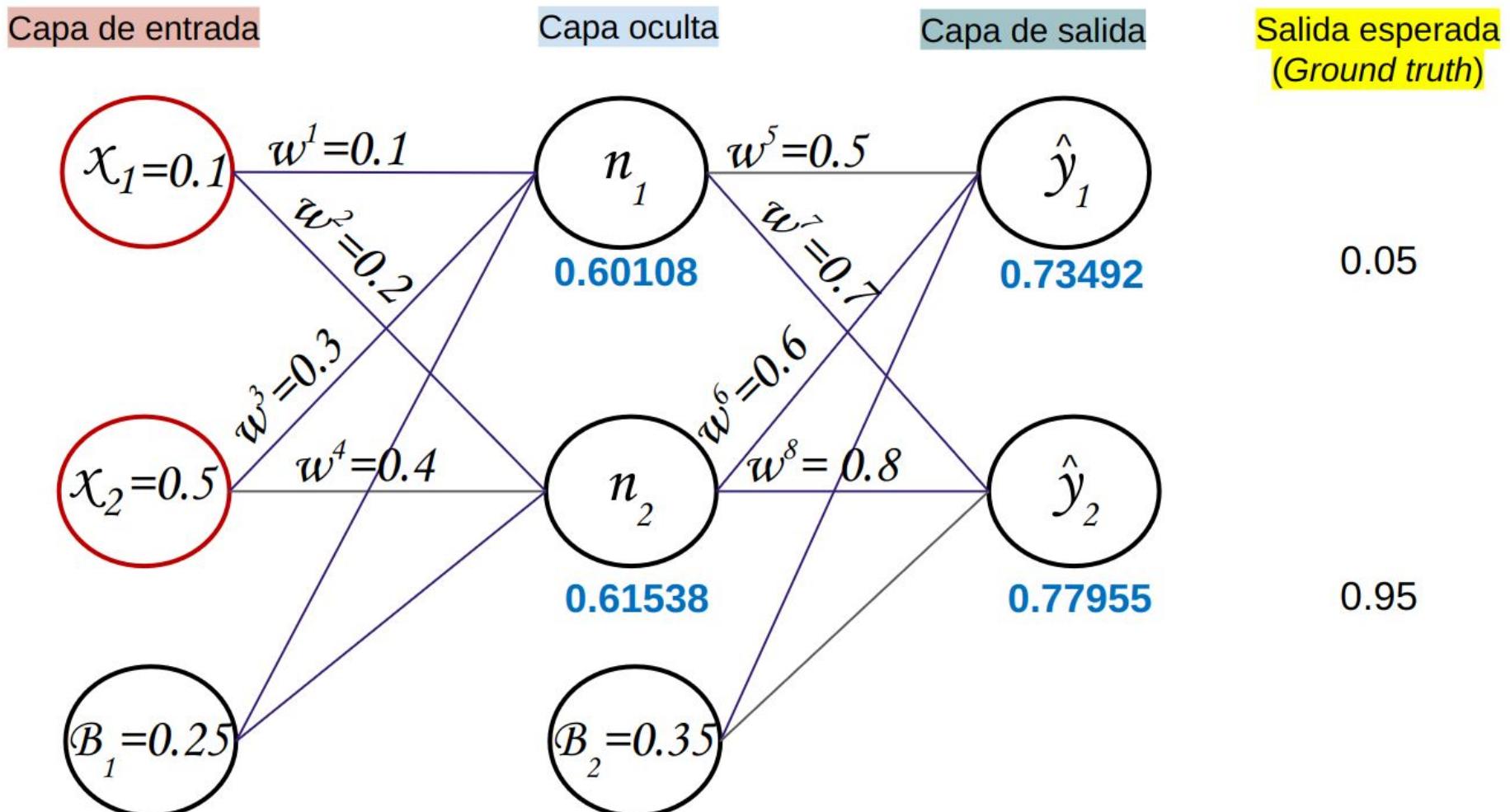
$$\hat{y}_2 =$$



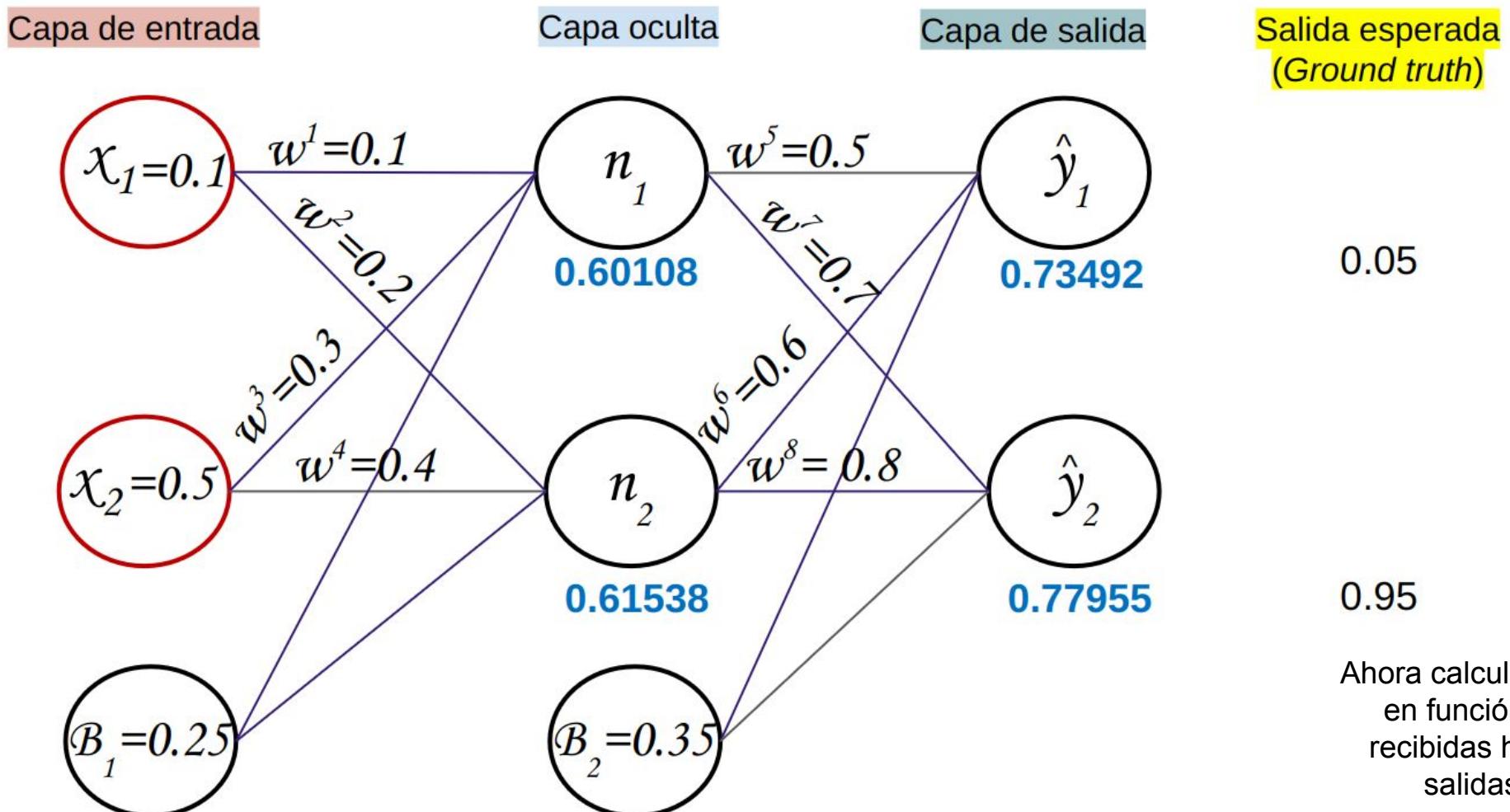
# Propagación hacia adelante



# Propagación hacia adelante



# Propagación hacia adelante



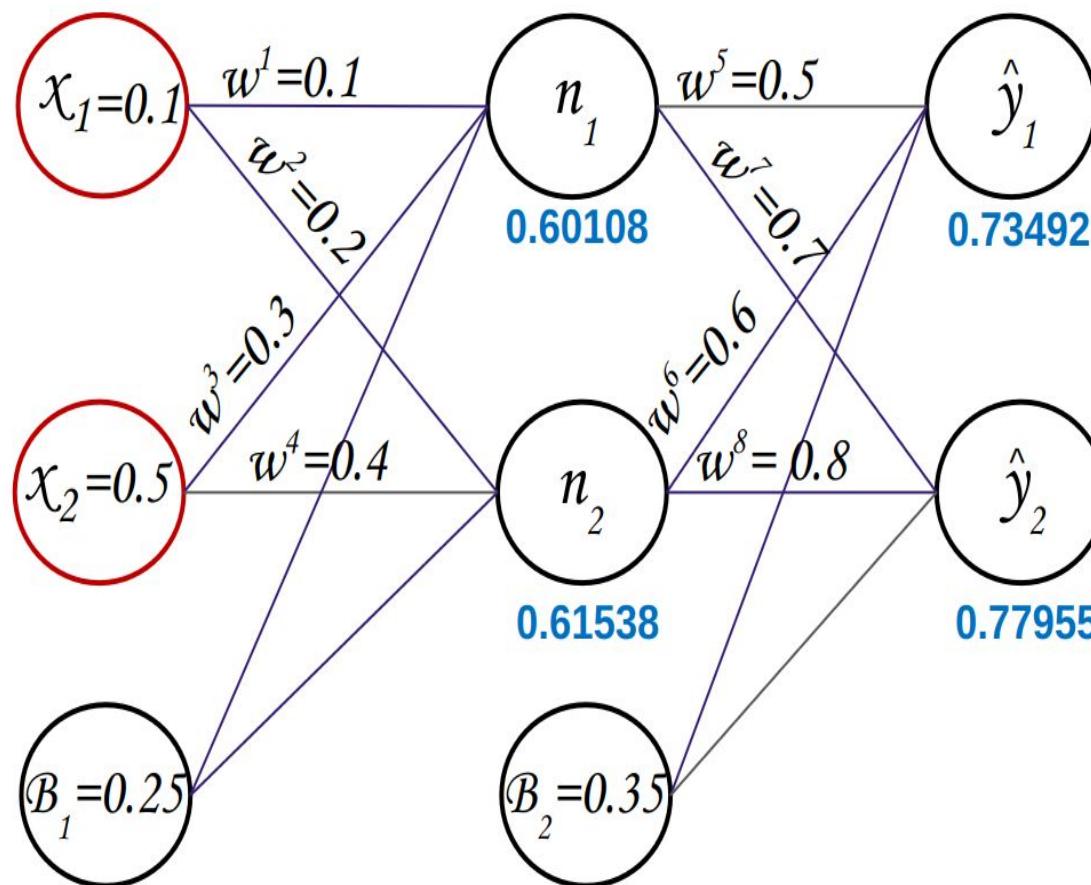
# Propagación hacia adelante

Capa de entrada

Capa oculta

Capa de salida

Salida esperada  
(*Ground truth*)



## Cálcular el error

$$E_{total} = \sum \frac{1}{2}(y_{true} - y_{pred})^2$$

Para calcular el error total, primero se calcula en  $\hat{y}_1$ :

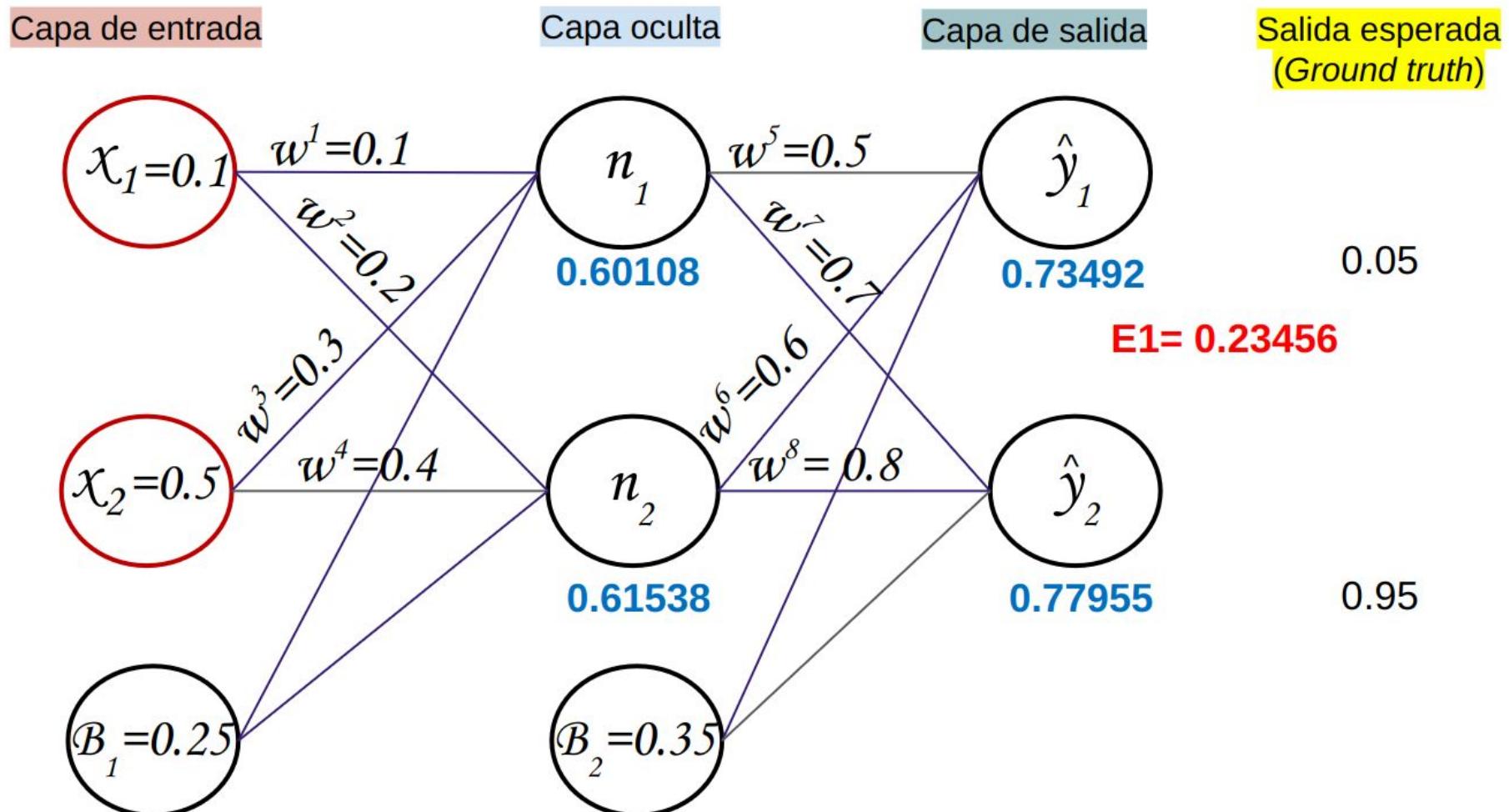
$$E_1 = \frac{1}{2}(y_{true1} - \hat{y}_1)^2$$

$$E_1 = \frac{1}{2}(0.05 - 0.73492)^2$$

$$E_1 = 0.23456$$



# Propagación hacia adelante



# Propagación hacia adelante

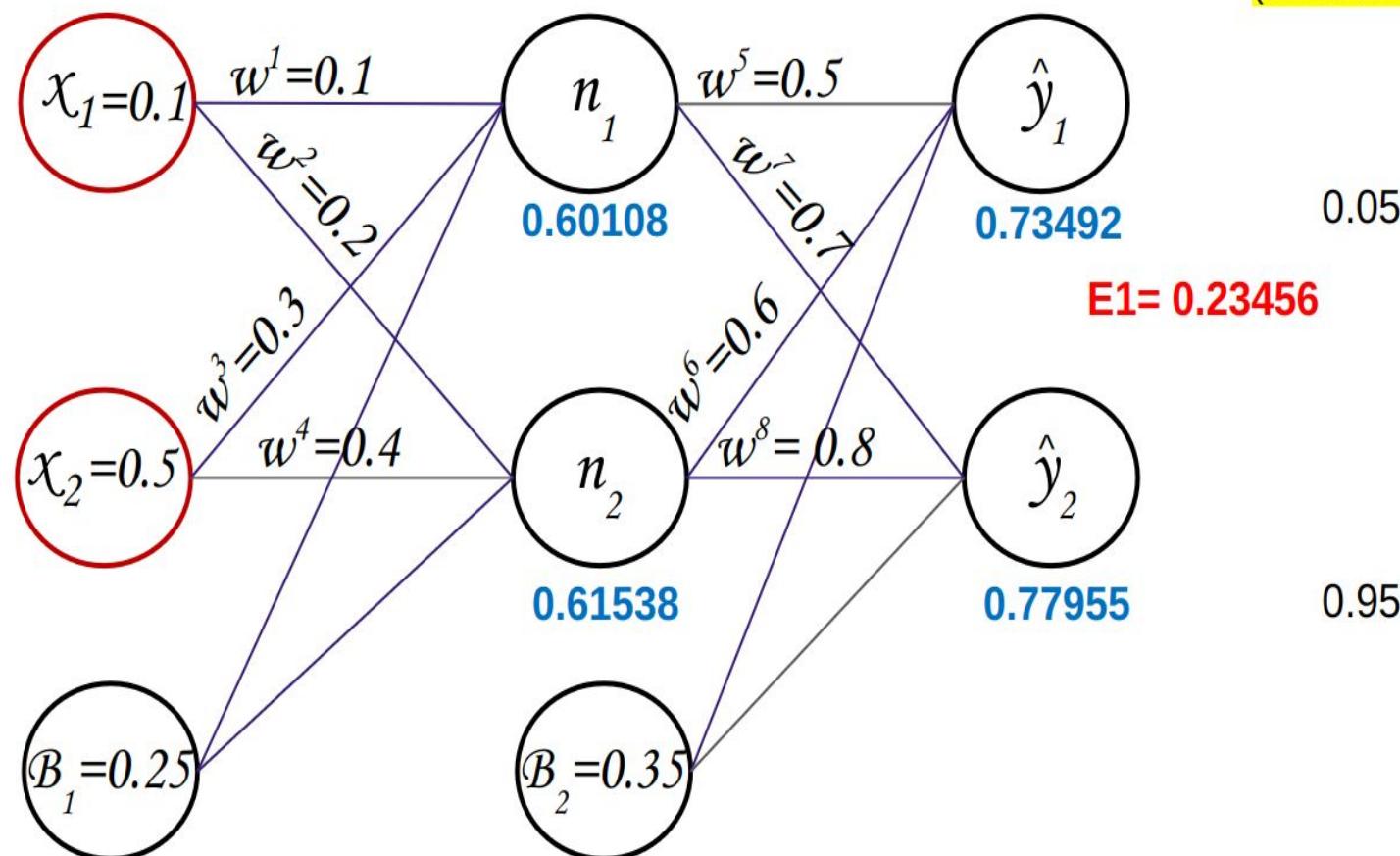
Capa de entrada

Capa oculta

Capa de salida

Salida esperada  
(Ground truth)

Cálcular el error



$$E_{total} = \sum \frac{1}{2}(y_{true} - y_{pred})^2$$

Ahora se calcula en  $\hat{y}_2$ :

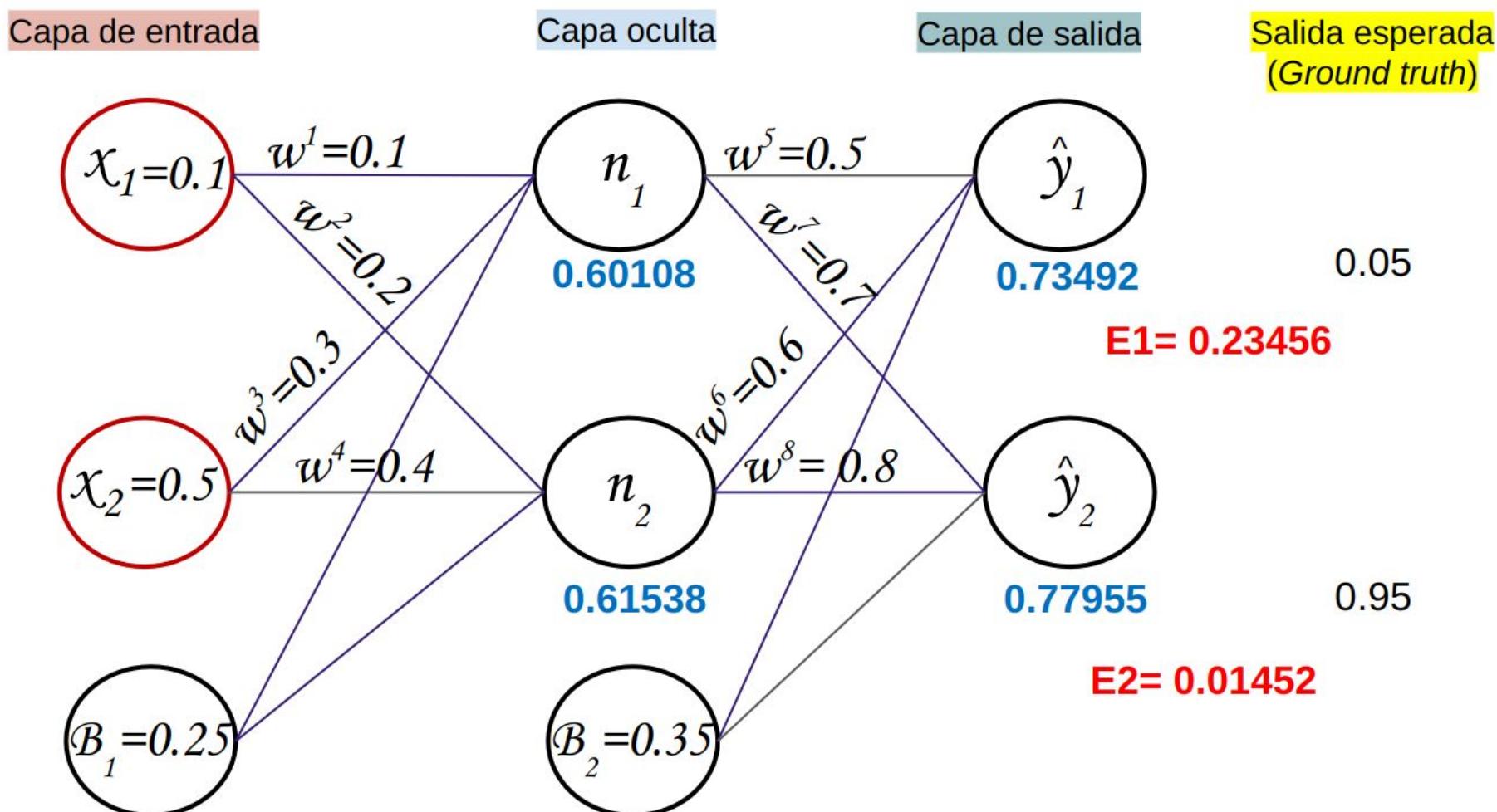
$$E_2 = \frac{1}{2}(y_{true}_2 - \hat{y}_2)^2$$

$$E_2 =$$

$$E_2 =$$



# Propagación hacia adelante



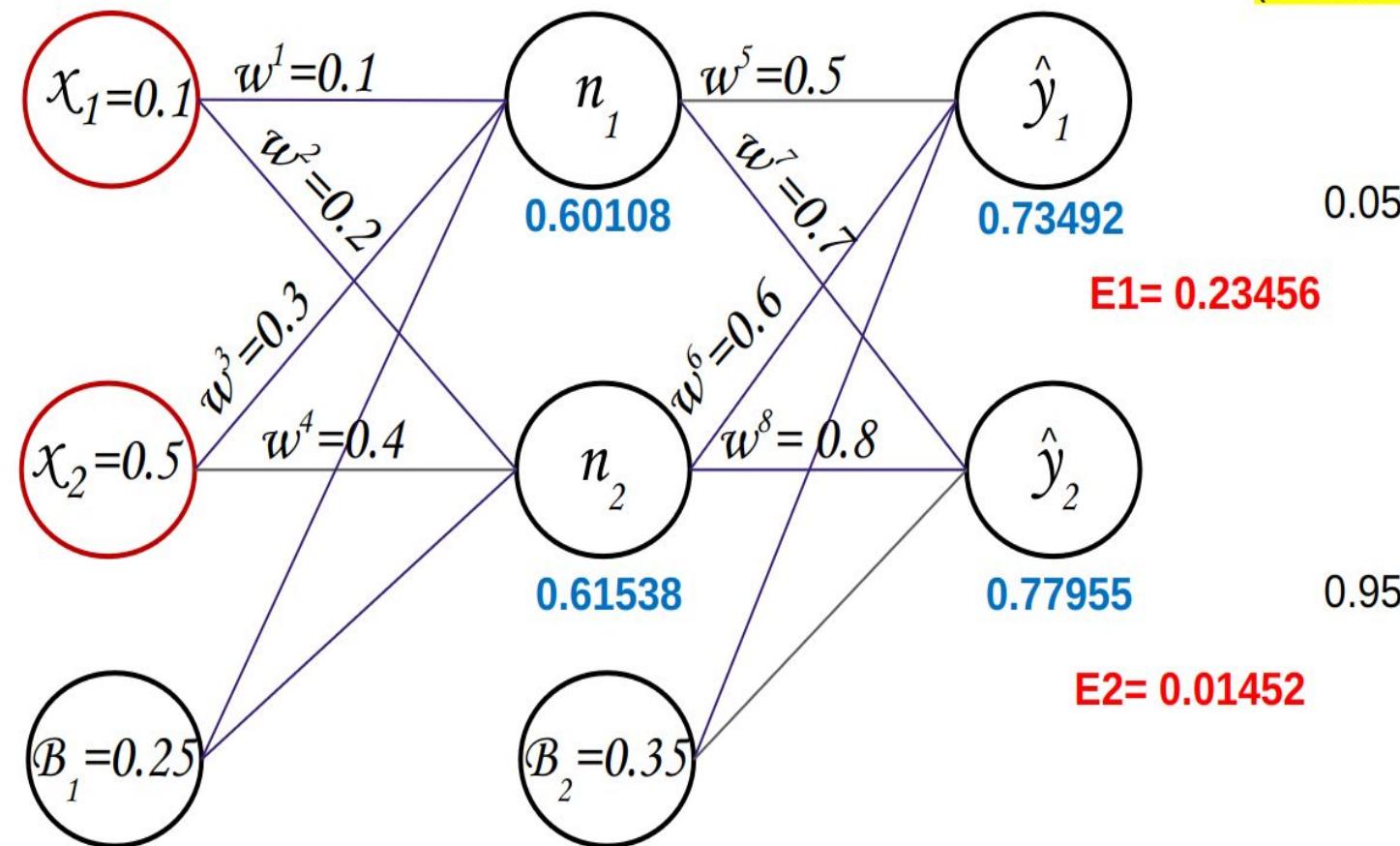
# Propagación hacia adelante

Capa de entrada

Capa oculta

Capa de salida

Salida esperada  
(Ground truth)



Calcular el error total

$$E_{total} = \sum E_1 + E_2$$

$$E_{total} = 0.23456 + 0.01452$$

$$E_{total} = 0.24908$$



# Retropropagación de errores

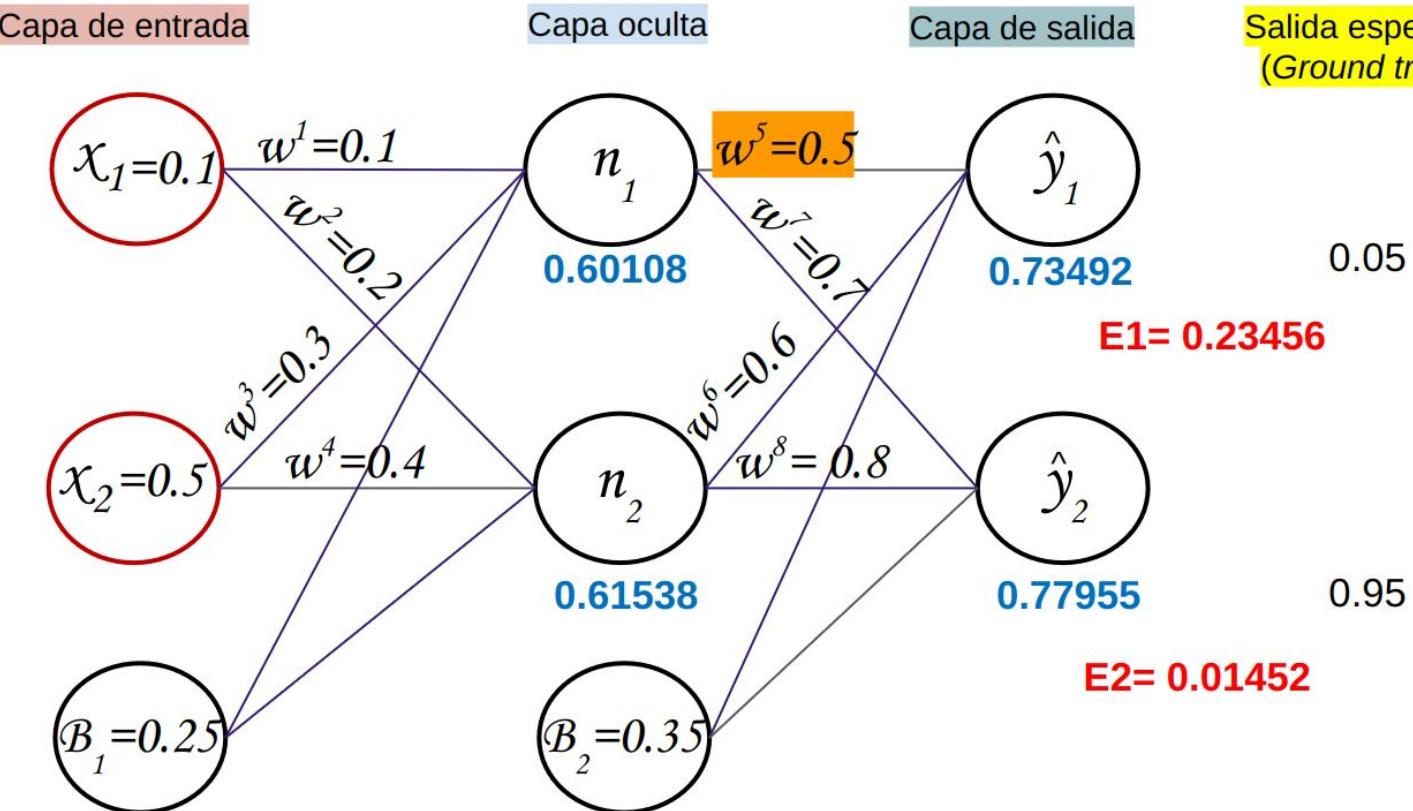
El objetivo de la retropropagación es **distribuir** el error total a la red para actualizar los pesos y minimizar la función de costo (pérdida).

- Los pesos se actualizan de tal manera que, cuando la siguiente propagación hacia adelante suceda, utilice los pesos actualizados.
- De esta forma, el error total se reducirá en un cierto margen (hasta que se alcance el mínimo).
- En inglés, se conoce como: *backward Pass*



# Propagación hacia atrás

Calcular cuánto contribuye  $w^5$  en  $E_1$



Observemos:

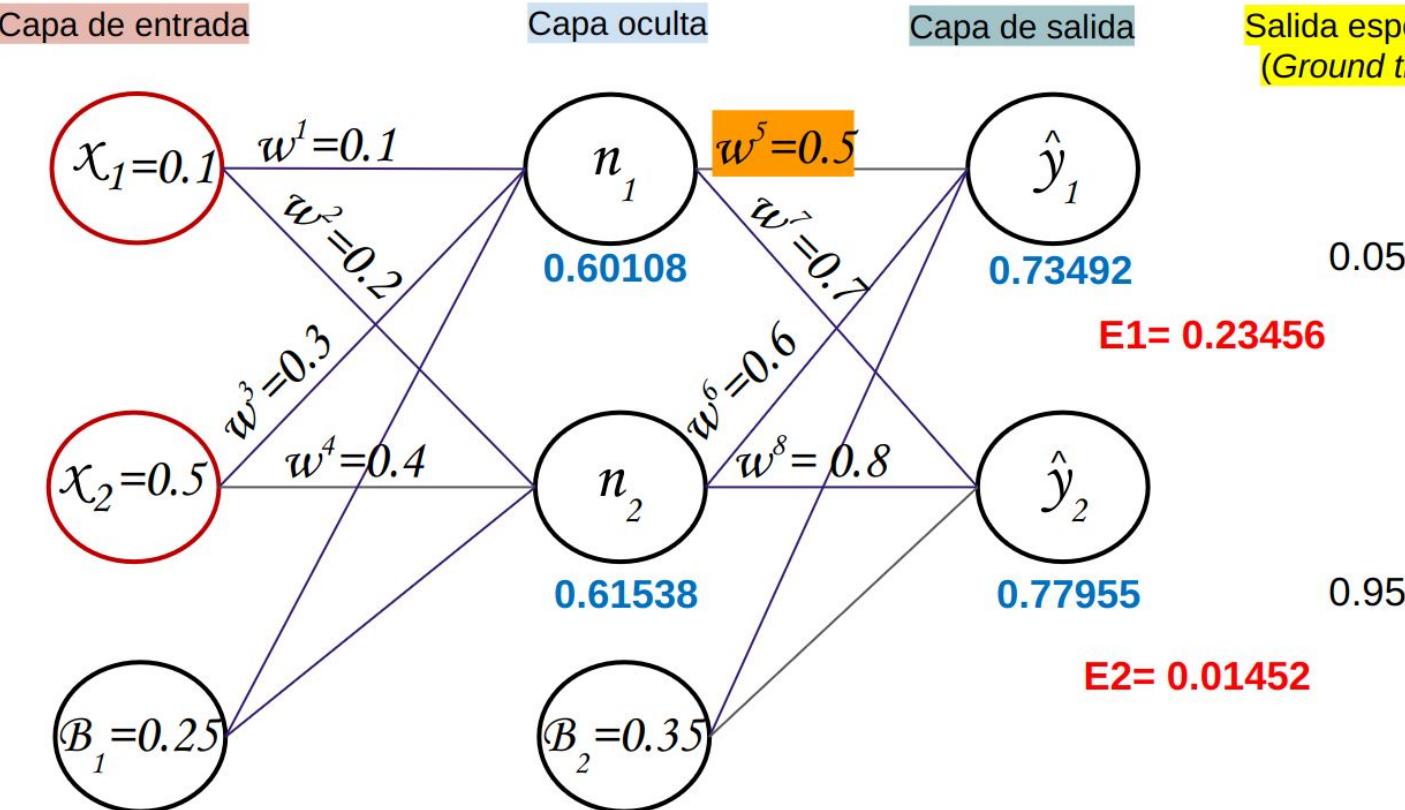
1. El error  $E_1$  está siendo afectado por  $\hat{y}_1$ .
2. Ahora,  $\hat{y}_1$  se ve afectado por  $sum_{\hat{y}_1}$ .
3. Finalmente,  $sum_{\hat{y}_1}$  se ve afectada por  $w_5$ .

A esto se le conoce como la regla de la cadena.



# Propagación hacia atrás

Calcular cuánto contribuye  $w^5$  en  $E_1$



Observemos:

1. El error  $E_1$  está siendo afectado por  $\hat{y}_1$ .
2. Ahora,  $\hat{y}_1$  se ve afectado por  $sum_{\hat{y}_1}$ .
3. Finalmente,  $sum_{\hat{y}_1}$  se ve afectada por  $w_5$ .

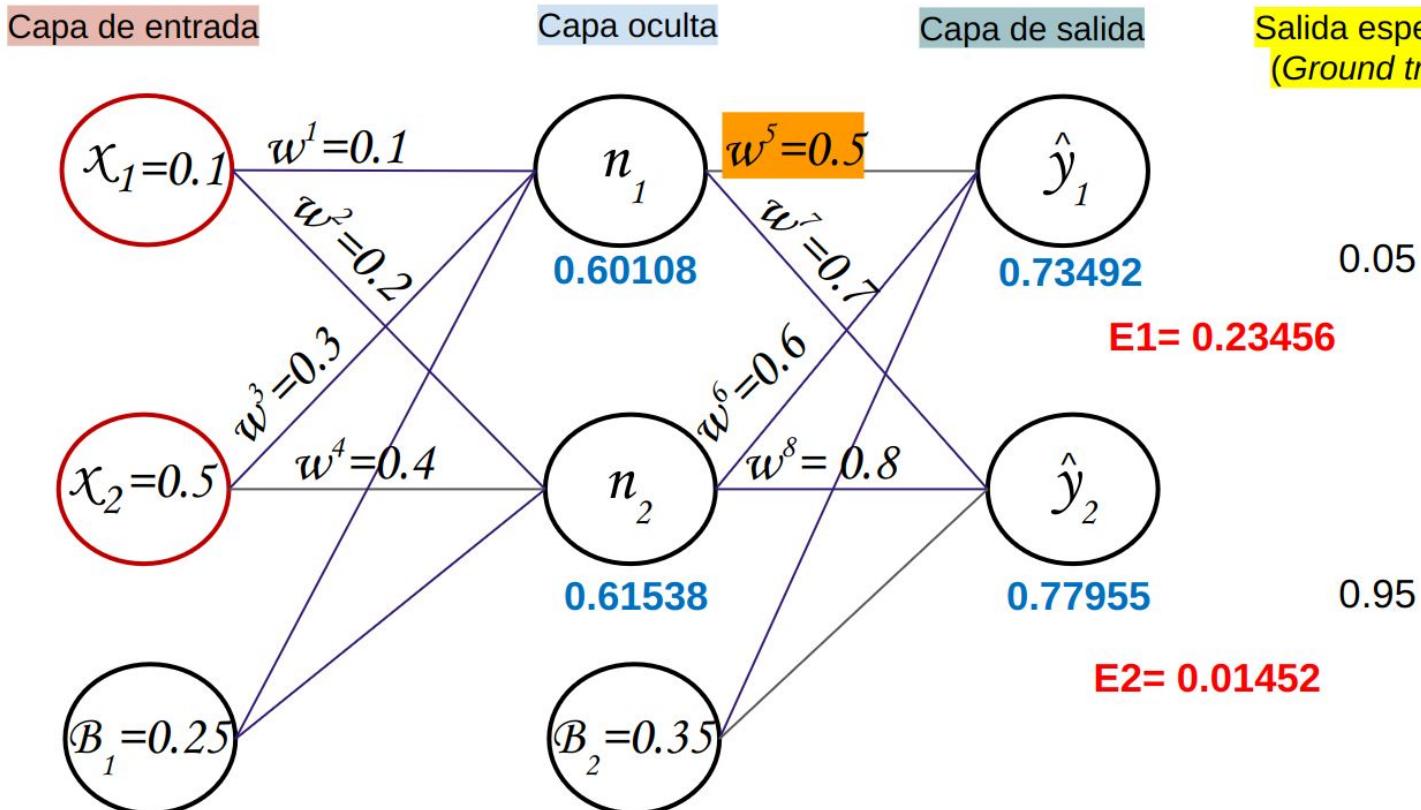
A esto se le conoce como la regla de la cadena.

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial w_5}$$



# Propagación hacia atrás

Calcular el 1er componente



$$\frac{\partial E_{total}}{\partial w_5} = \boxed{\frac{\partial E_{total}}{\partial \hat{y}_1}} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial w_5}$$

Recordemos:

$$E_{total} = \sum \frac{1}{2} (y_{true} - y_{pred})^2$$

$$E_{total} = \sum \frac{1}{2} (y_{true1} - \hat{y}_1)^2 + \frac{1}{2} (y_{true2} - \hat{y}_2)^2$$

Por lo tanto:

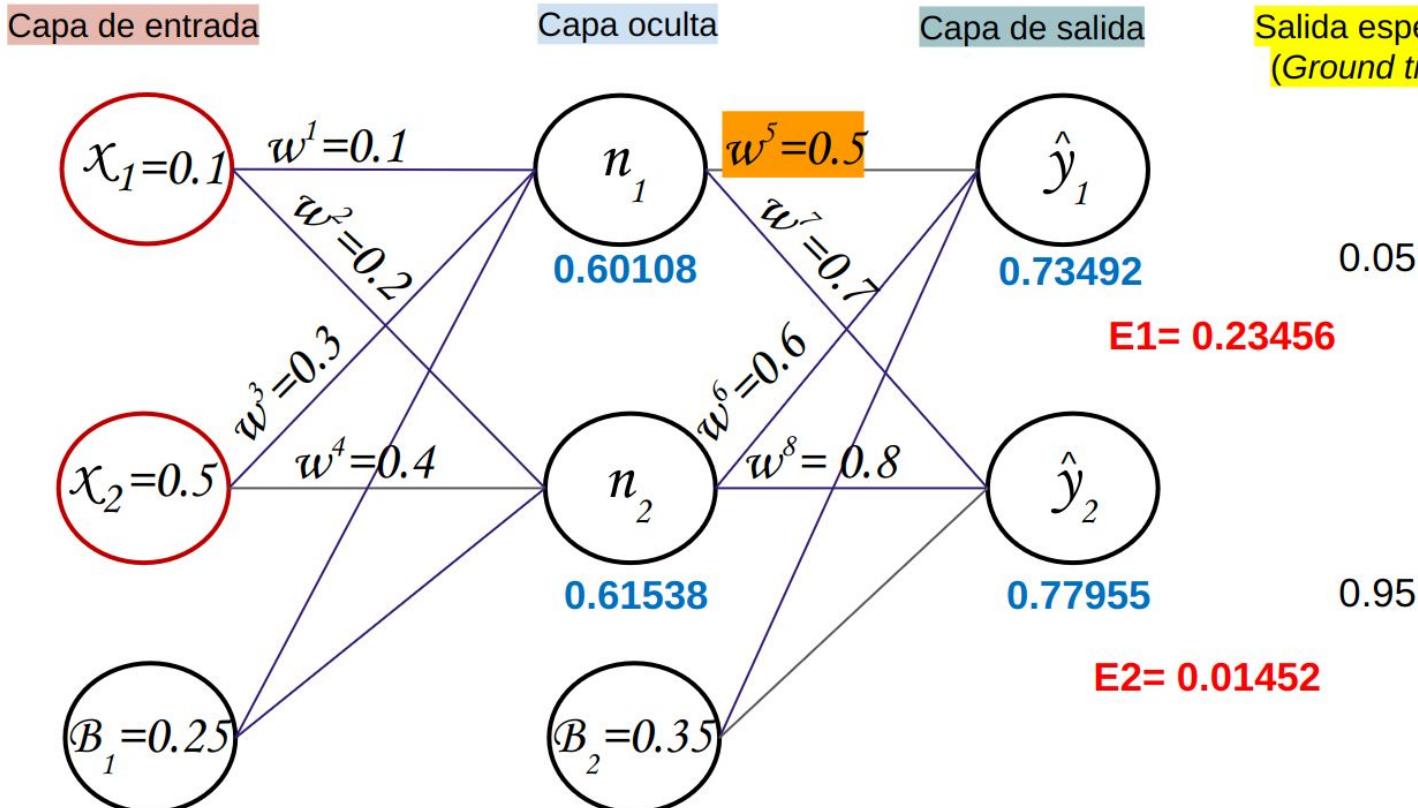
$$\frac{\partial E_{total}}{\partial \hat{y}_1} = 2 * \frac{1}{2} * (y_{true1} - \hat{y}_1) * -1$$

$$\frac{\partial E_{total}}{\partial \hat{y}_1} = \hat{y}_1 - y_{true1}$$



# Propagación hacia atrás

Calcular el 1er componente



$$\frac{\partial E_{total}}{\partial w_5} = \boxed{\frac{\partial E_{total}}{\partial \hat{y}_1}} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial w_5}$$

Recordemos:

$$E_{total} = \sum \frac{1}{2} (y_{true} - y_{pred})^2$$

$$E_{total} = \sum \frac{1}{2} (y_{true1} - \hat{y}_1)^2 + \frac{1}{2} (y_{true2} - \hat{y}_2)^2$$

Por lo tanto:

$$\frac{\partial E_{total}}{\partial \hat{y}_1} = 2 * \frac{1}{2} * (y_{true1} - \hat{y}_1) * -1$$

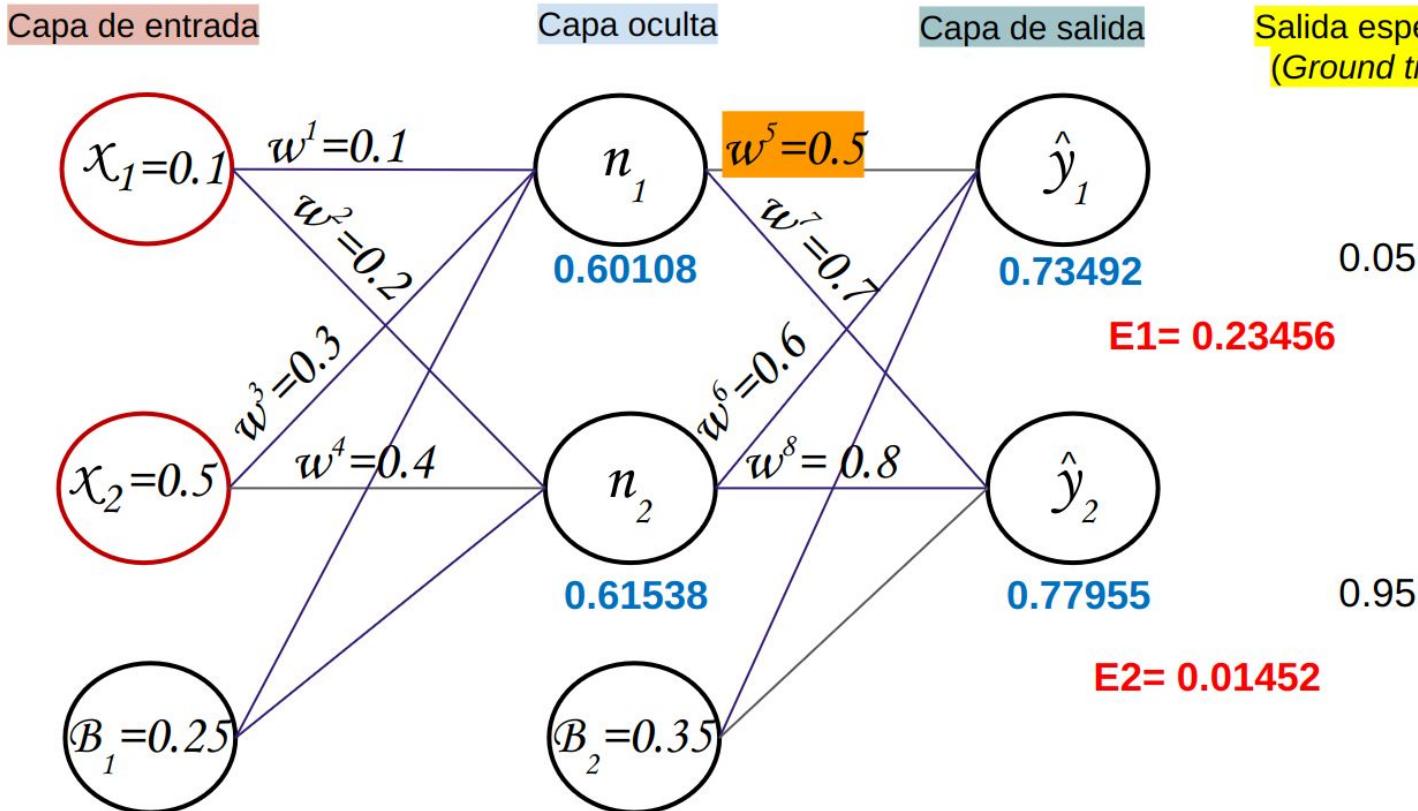
$$\frac{\partial E_{total}}{\partial \hat{y}_1} = \hat{y}_1 - y_{true1}$$

$$\frac{\partial E_{total}}{\partial \hat{y}_1} = \hat{y}_1 - y_{true1} = 0.73492 - 0.05 = 0.68492$$



# Propagación hacia atrás

Calcular el 2do componente



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial \hat{y}_1} * \boxed{\frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}}} * \frac{\partial sum_{\hat{y}_1}}{\partial w_5}$$

Recordemos: La salida de  $\hat{y}_1$  usa una función de activación sigmoide de salida. Entonces, calculamos la derivada de la función.

$$\sigma(x) = \frac{1}{1 + e^{-sum_{n_1}}} \quad (20)$$

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

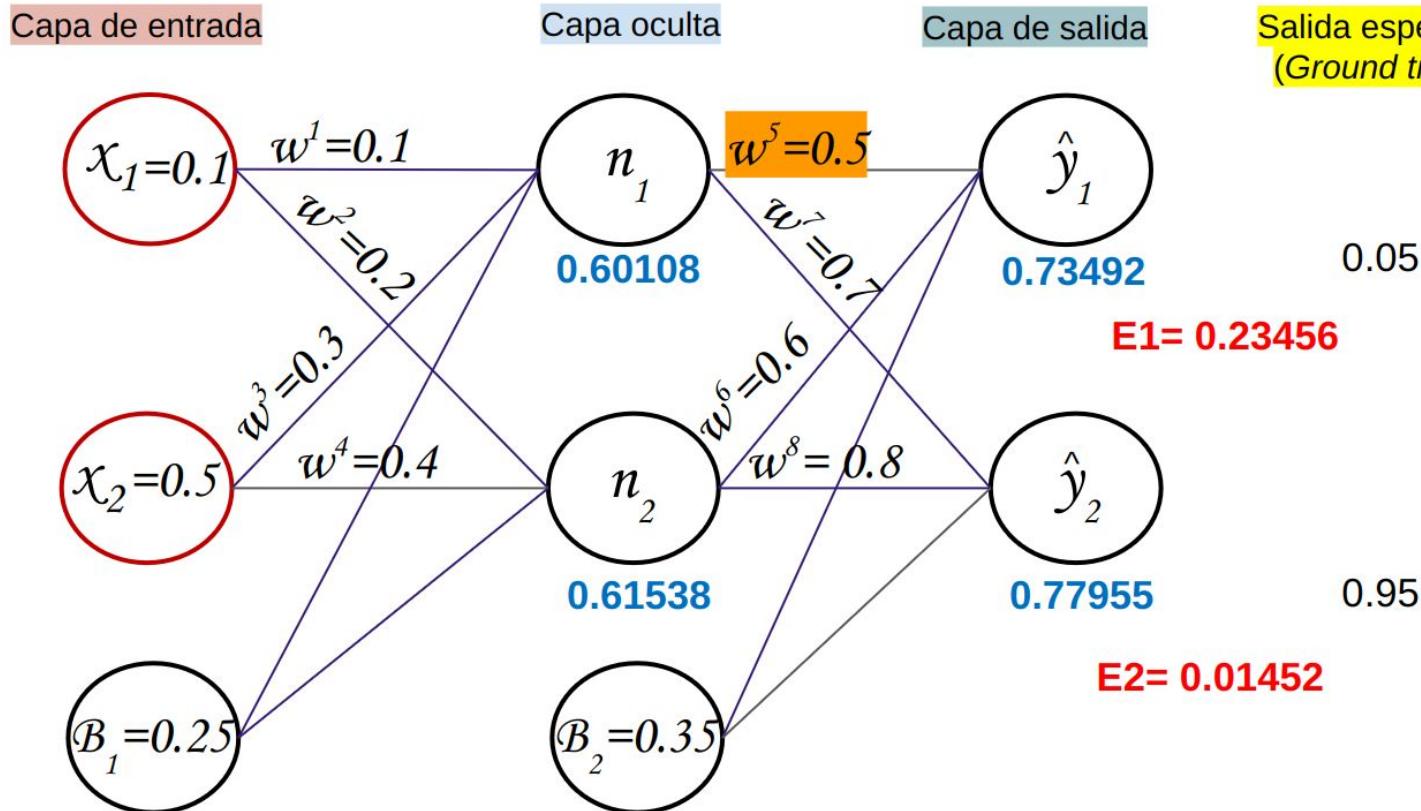
Por lo tanto, la derivada de la función sigmoide es:

$$\frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} = \hat{y}_1(1 - \hat{y}_1) \quad (21)$$



# Propagación hacia atrás

Calcular el 2do componente



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial \hat{y}_1} * \boxed{\frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}}} * \frac{\partial sum_{\hat{y}_1}}{\partial w_5}$$

Recordemos: La salida de  $\hat{y}_1$  usa una función de activación sigmoide de salida. Entonces, calculamos la derivada de la función.

$$\sigma(x) = \frac{1}{1 + e^{-sum_{n_1}}} \quad (20)$$

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

Por lo tanto, la derivada de la función sigmoide es:

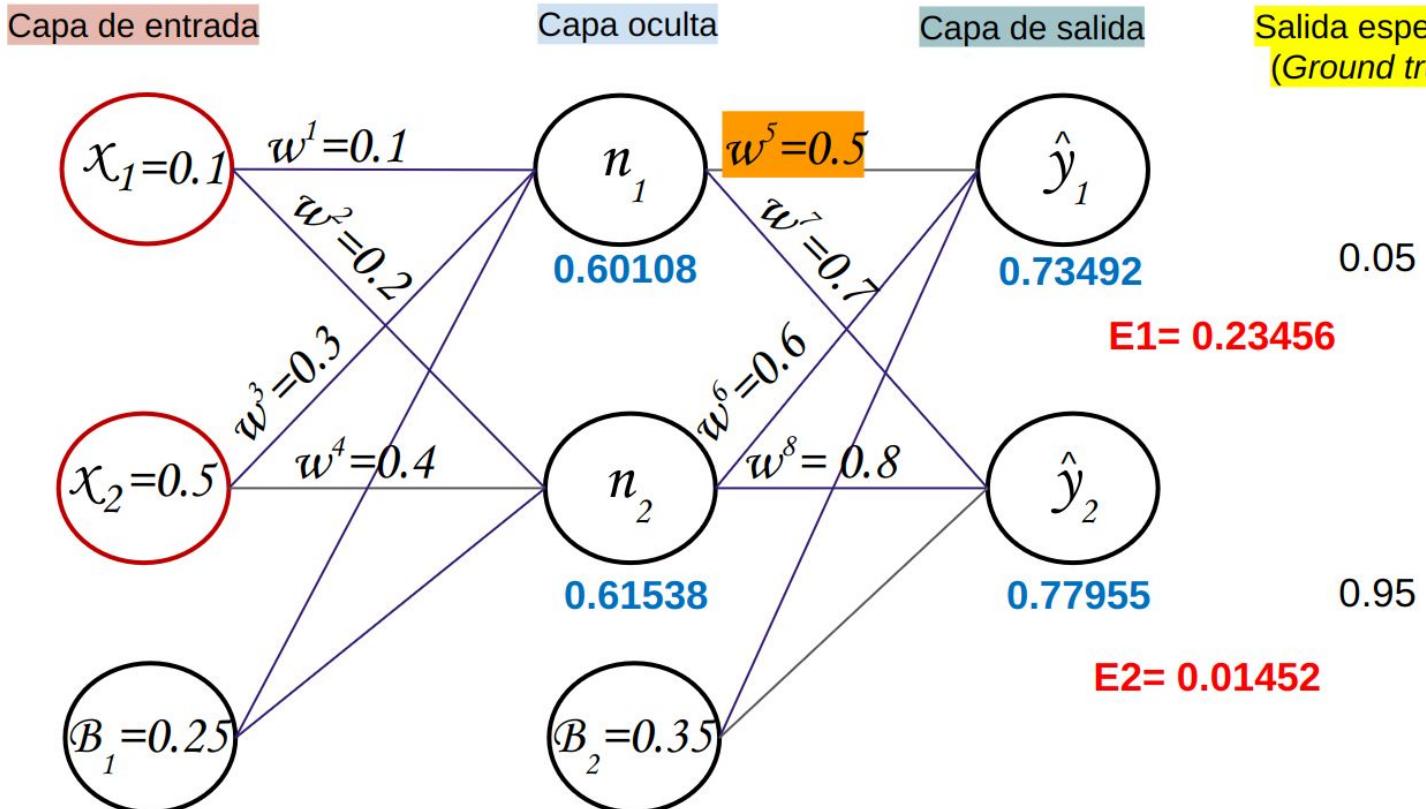
$$\frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} = \hat{y}_1(1 - \hat{y}_1) \quad (21)$$

$$\frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} = 0.73492(1 - 0.73492) = 0.19480$$



# Propagación hacia atrás

Calcular el 3er componente



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \boxed{\frac{\partial sum_{\hat{y}_1}}{\partial w_5}}$$

Recordemos:

$$sum_{\hat{y}_1} = (n_1 * w^5) + (n_2 * w^6) + B_2$$

Por lo tanto,

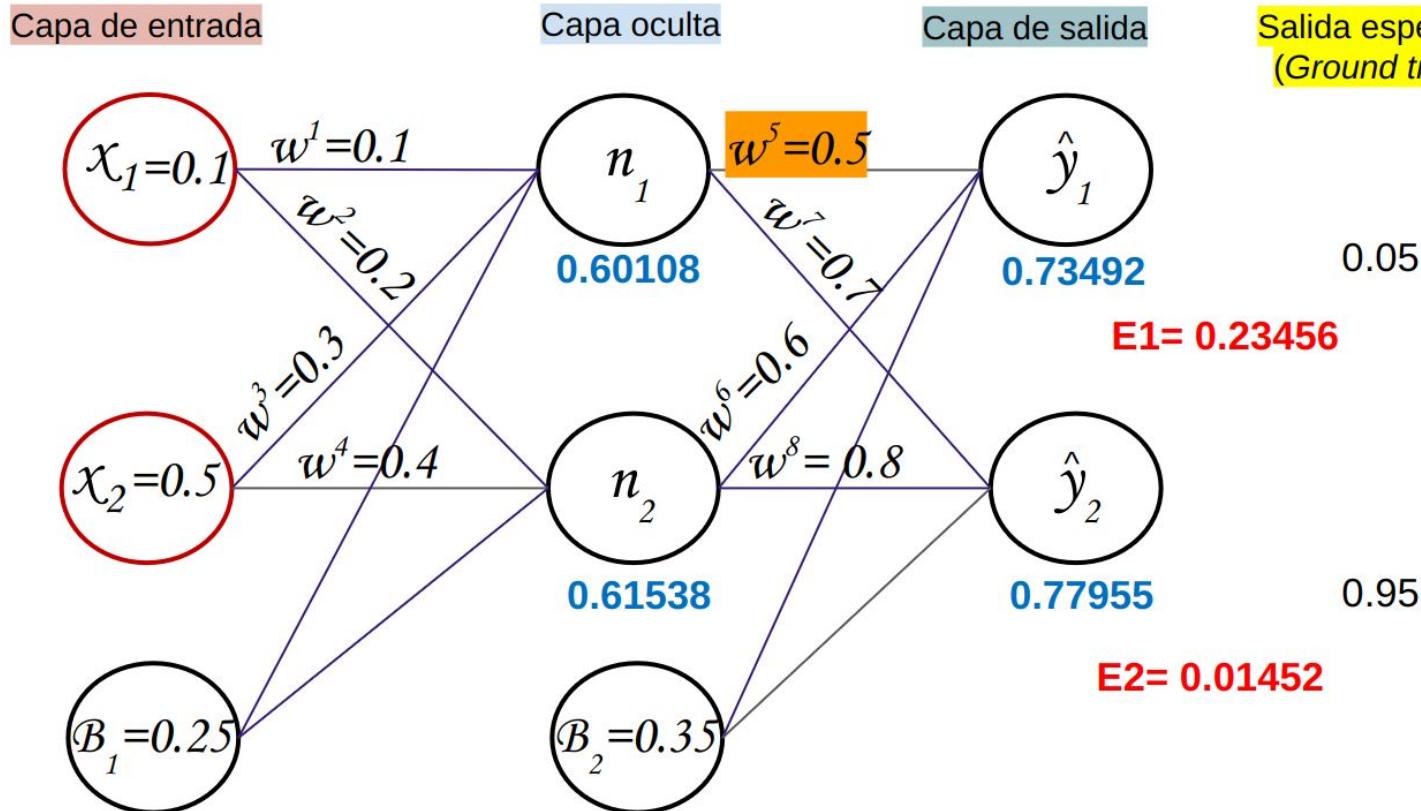
$$\frac{\partial sum_{\hat{y}_1}}{\partial w_5} = n_1$$

$$\frac{\partial sum_{\hat{y}_1}}{\partial w_5} = 0.60108$$



# Propagación hacia atrás

Pongamos todo junto



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial w_5}$$

$$\frac{\partial E_{total}}{\partial w_5} = [\hat{y}_1 - y_{true1}] * [\hat{y}_1(1 - \hat{y}_1)] * [n_1]$$

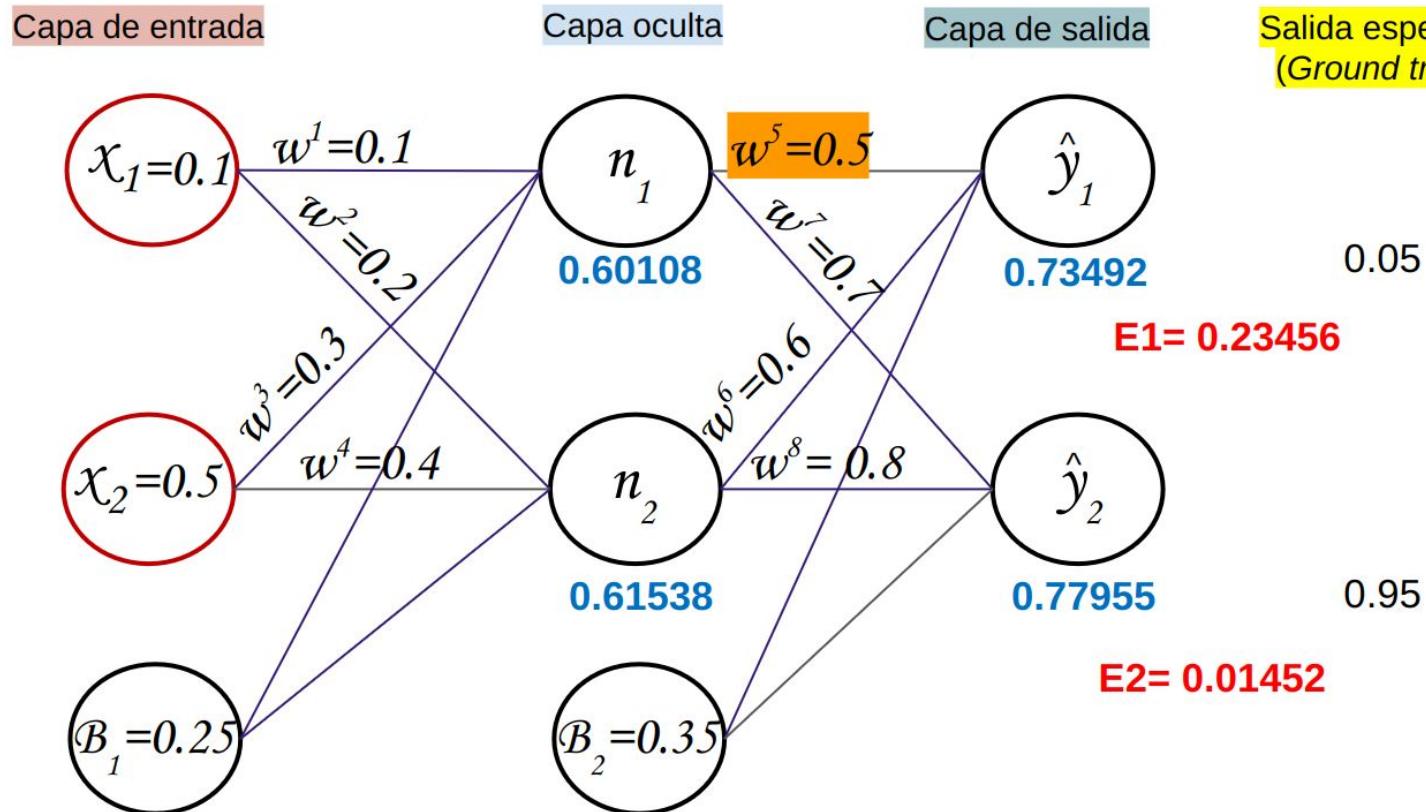
$$\frac{\partial E_{total}}{\partial w_5} = 0.68492 * 0.19480 * 0.60108$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.0802$$



# Propagación hacia atrás

Pongamos todo junto



$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial w_5}$$

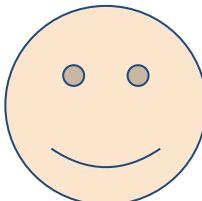
El *nuevo\_w5* es:

$$nuevo\_w5 = actual\_w5 - n * \frac{\partial E_{total}}{\partial w_5}$$

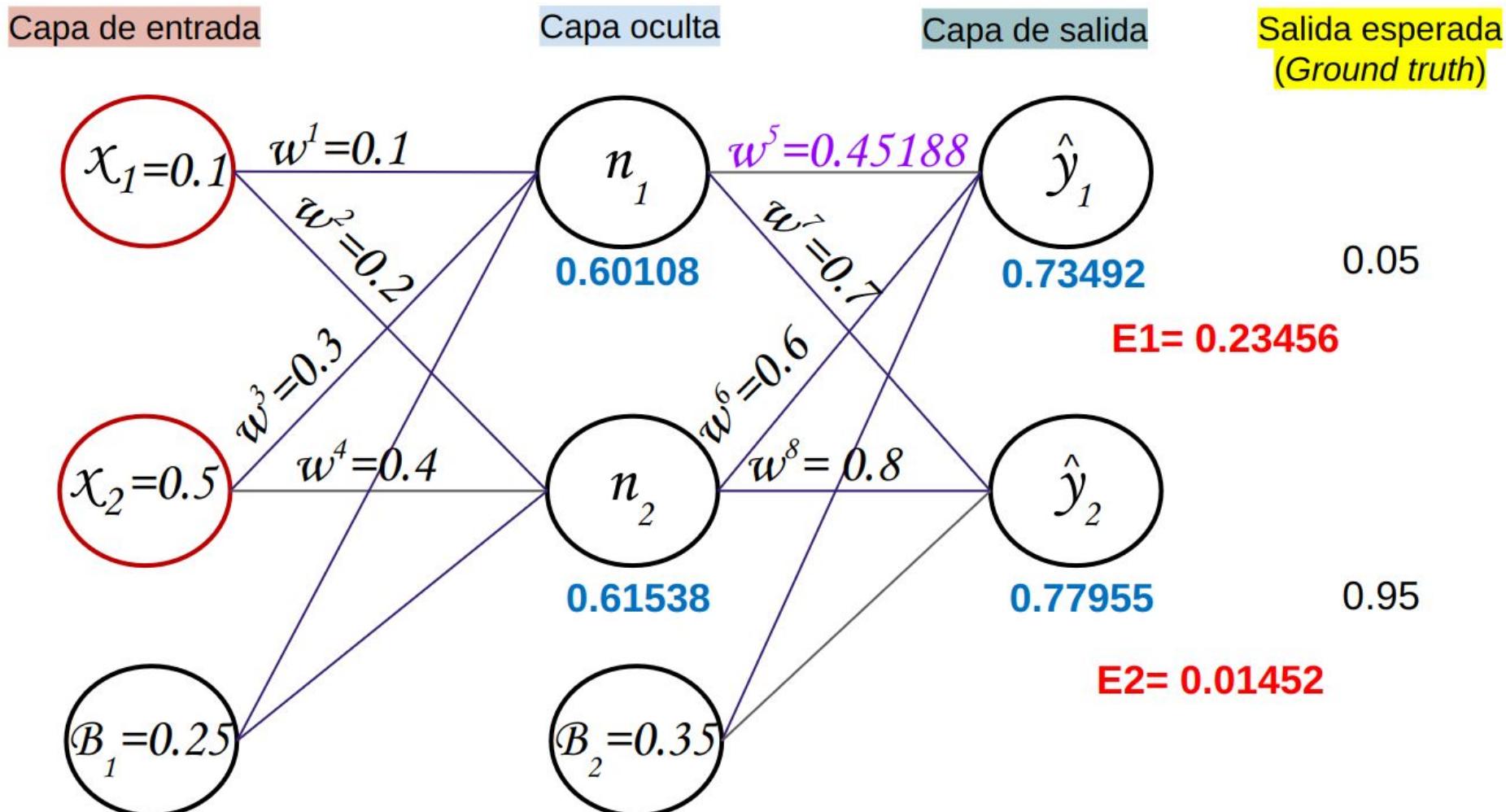
Donde *n* es la tasa de aprendizaje:

$$nuevo\_w5 = 0.5 - 0.6 * 0.0802$$

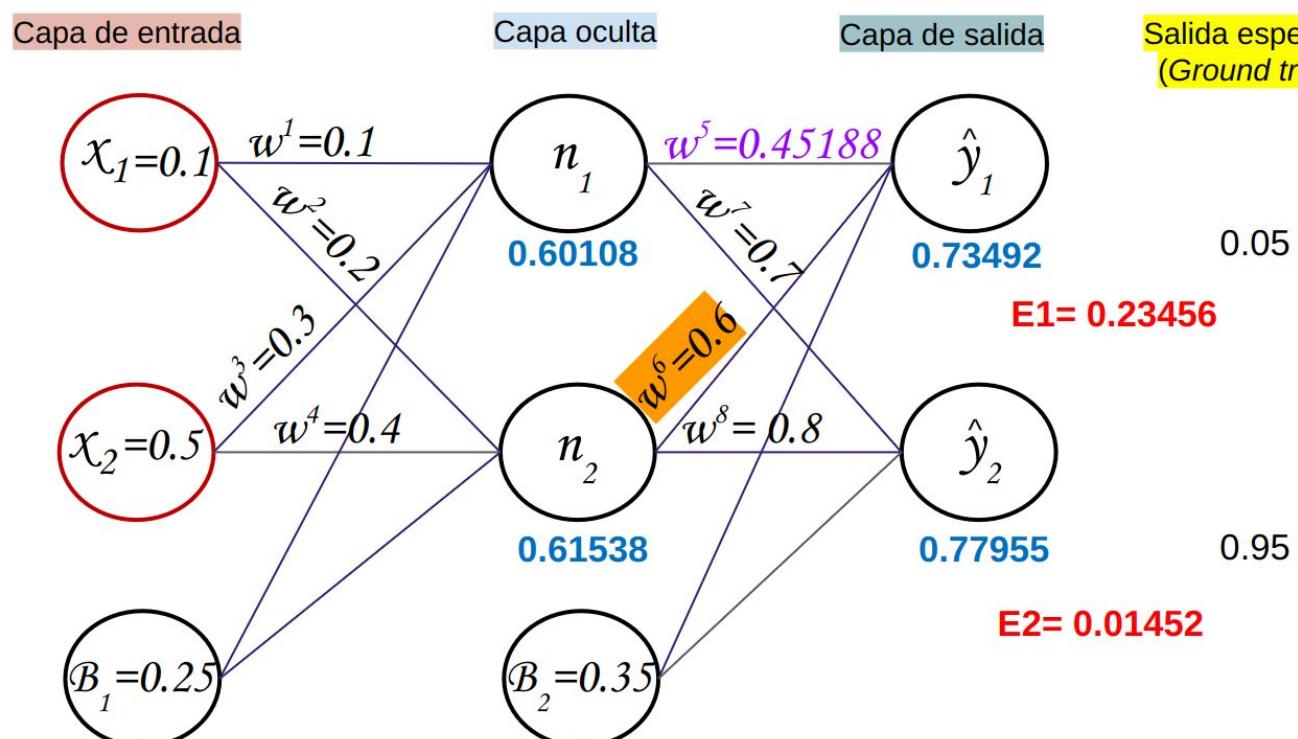
$nuevo\_w5 = 0.45188$



# Añadiendo el nuevo peso ( $w^5$ )



# Propagación hacia atrás



Calculando el nuevo peso de  $w_6$

$$\frac{\partial E_{total}}{\partial w_6} = \frac{\partial E_{total}}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial w_6}$$

Los dos primeros componentes ya se calcularon previamente.  
Para el 3er componente, queda de la siguiente manera:

$$\frac{\partial sum_{\hat{y}_1}}{\partial w_6} = n_2$$

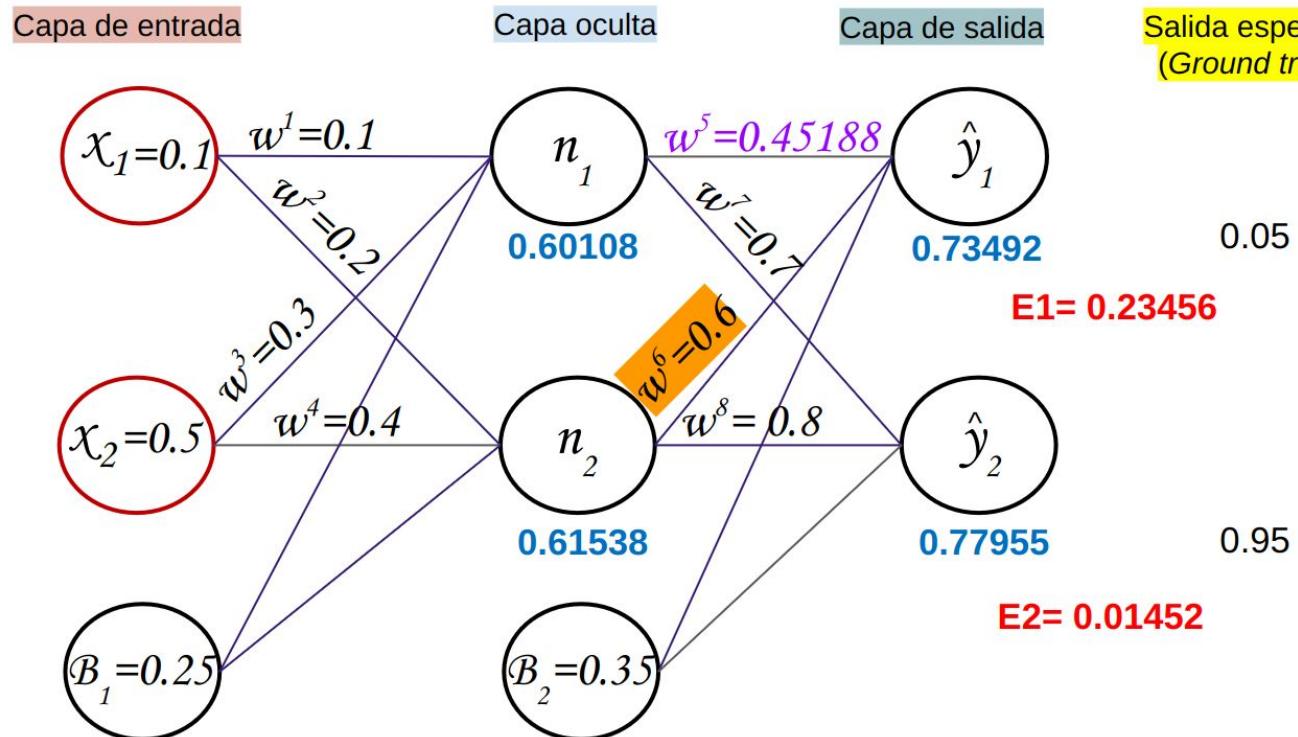
Por lo tanto,

$$\frac{\partial E_{total}}{\partial w_6} = 0.68492 * 0.19480 * 0.61538$$

$$\frac{\partial E_{total}}{\partial w_6} = 0.08211$$



# Propagación hacia atrás



Calculando el nuevo peso de  $w_6$

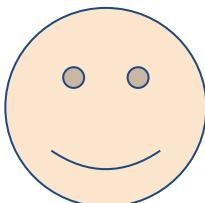
El  $new\_w6$  es:

$$new\_w6 = actual\_w6 - n * \frac{\partial E_{total}}{\partial w_6}$$

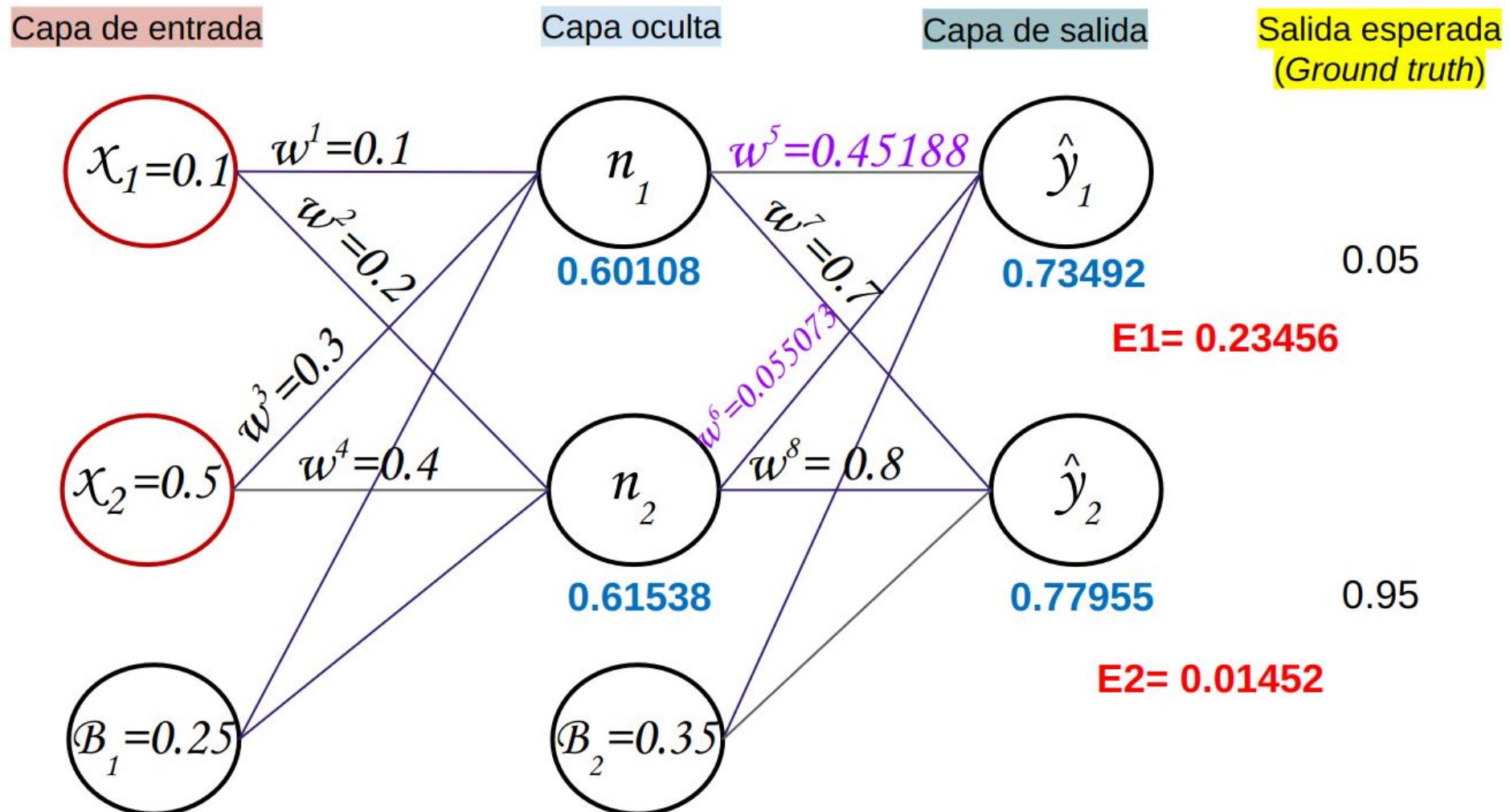
Donde  $n$  es la tasa de aprendizaje:

$$new\_w6 = 0.6 - 0.6 * 0.08211$$

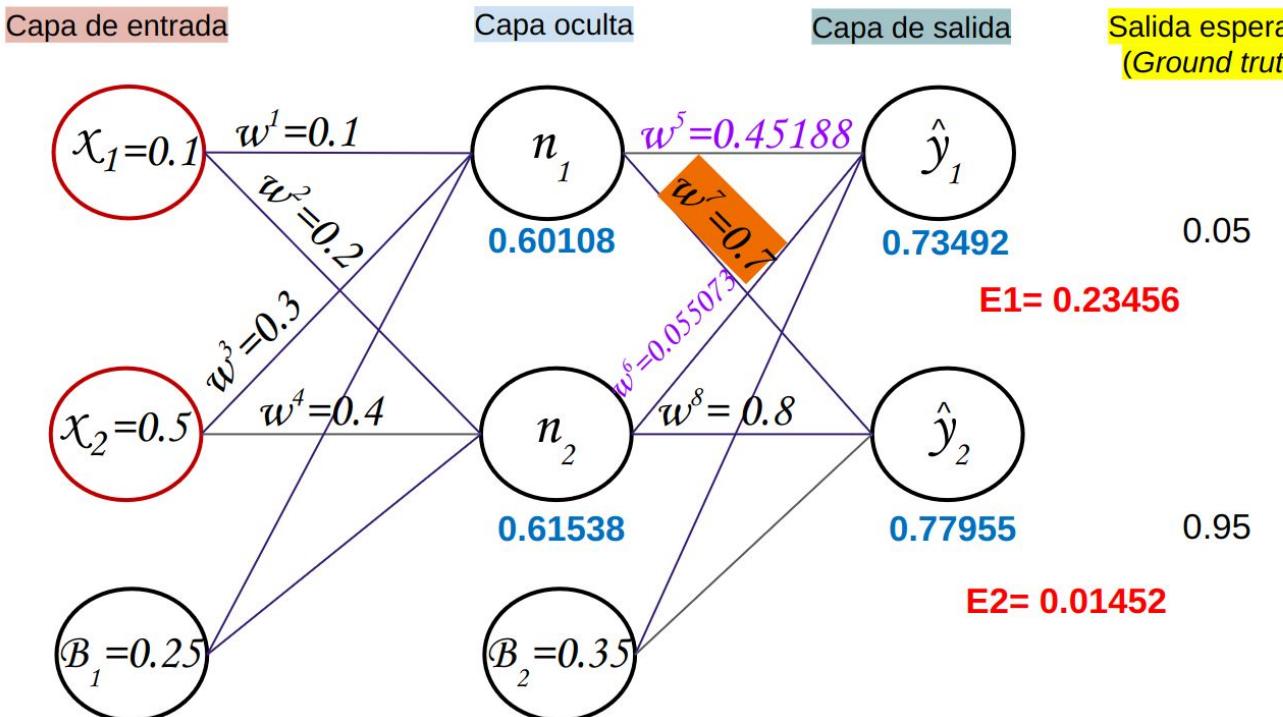
$$new\_w6 = 0.55073$$



# Añadiendo el nuevo peso ( $w^6$ )



# Propagación hacia atrás



Calculando el nuevo peso de  $w_7$

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial sum_{\hat{y}_2}} * \frac{\partial sum_{\hat{y}_2}}{\partial w_7}$$

1er componente:

$$\frac{\partial E_{total}}{\partial \hat{y}_2} = \hat{y}_2 - y_{true}_2$$

2do componente:

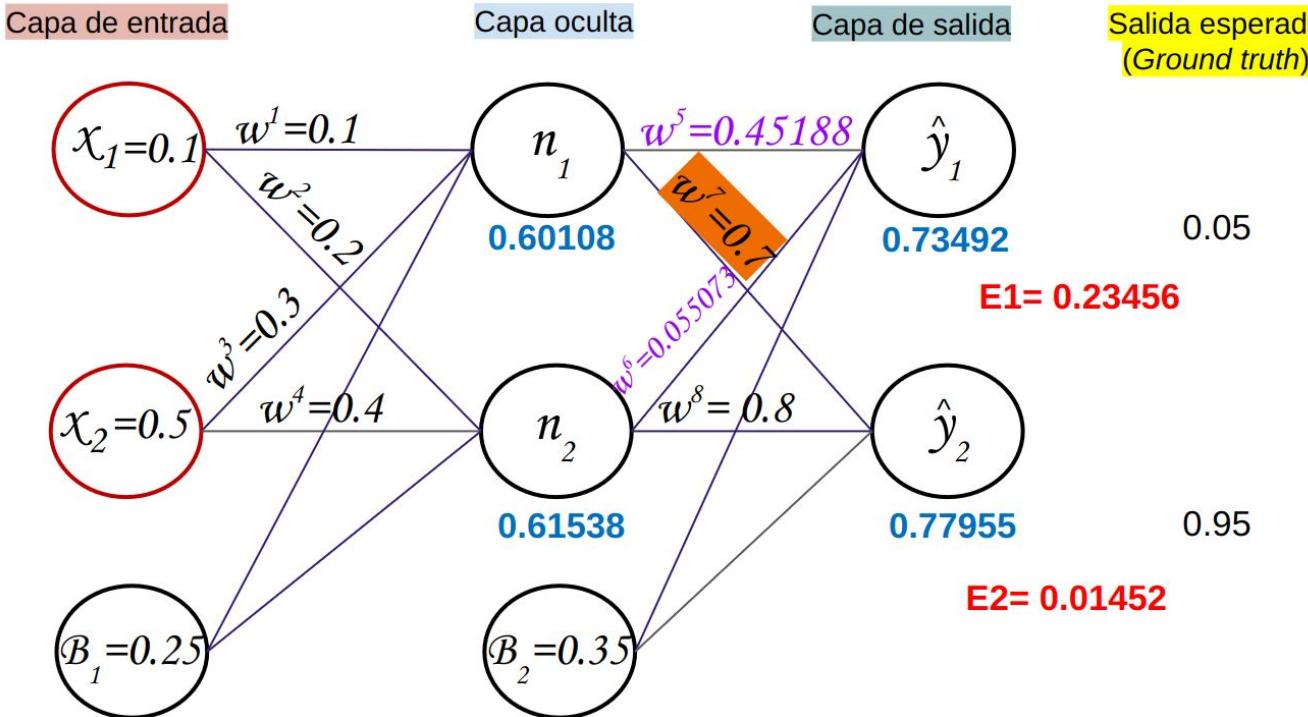
$$\frac{\partial \hat{y}_2}{\partial sum_{\hat{y}_2}} = \hat{y}_2(1 - \hat{y}_2)$$

3er componente:

$$\frac{\partial sum_{\hat{y}_2}}{\partial w_7} = n_1$$



# Propagación hacia atrás



Calculando el nuevo peso de  $w_7$

Poniendo todo junto:

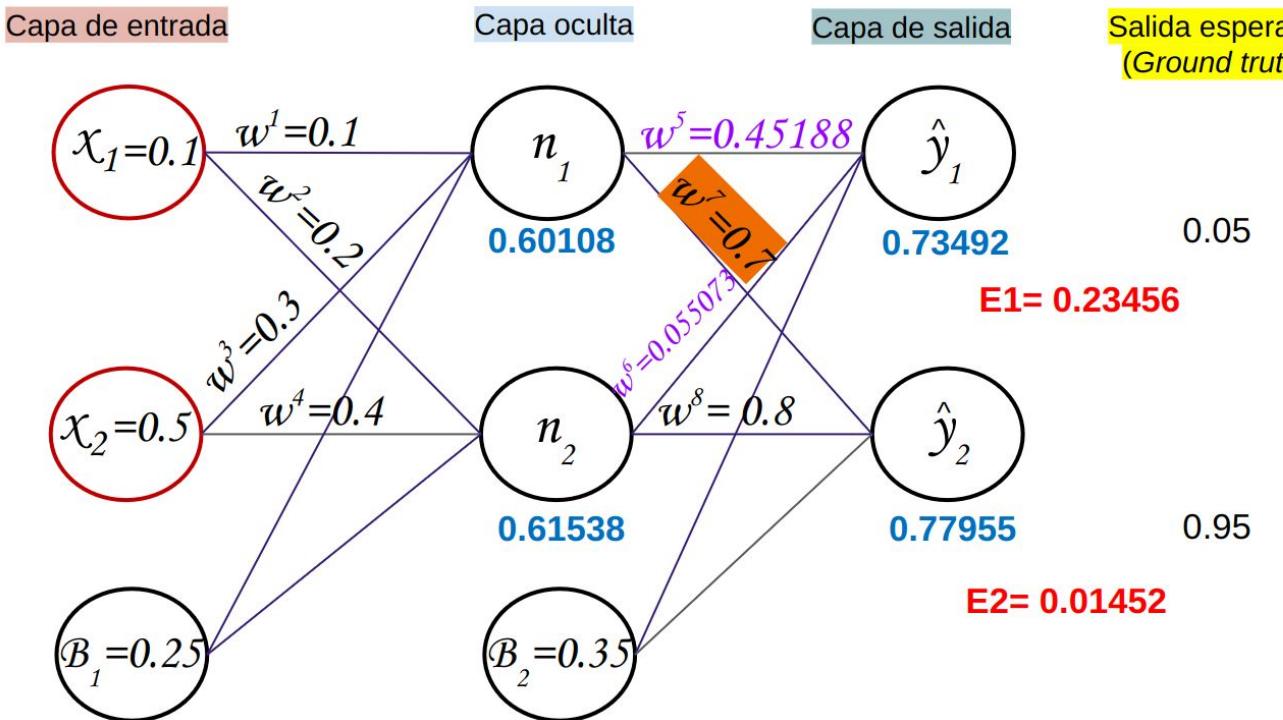
$$\frac{\partial E_{total}}{\partial w_7} = [\hat{y}_2 - y_{true}_2] * [\hat{y}_2(1 - \hat{y}_2)] * [n_1]$$

$$\frac{\partial E_{total}}{\partial w_7} = -0.17044 * 0.17184 * 0.60108$$

$$\frac{\partial E_{total}}{\partial w_7} = -0.01760$$



# Propagación hacia atrás



Calculando el nuevo peso de  $w_7$

El  $nuevo\_w7$  es:

$$nuevo\_w7 = actual\_w7 - n * \frac{\partial E_{total}}{\partial w7}$$

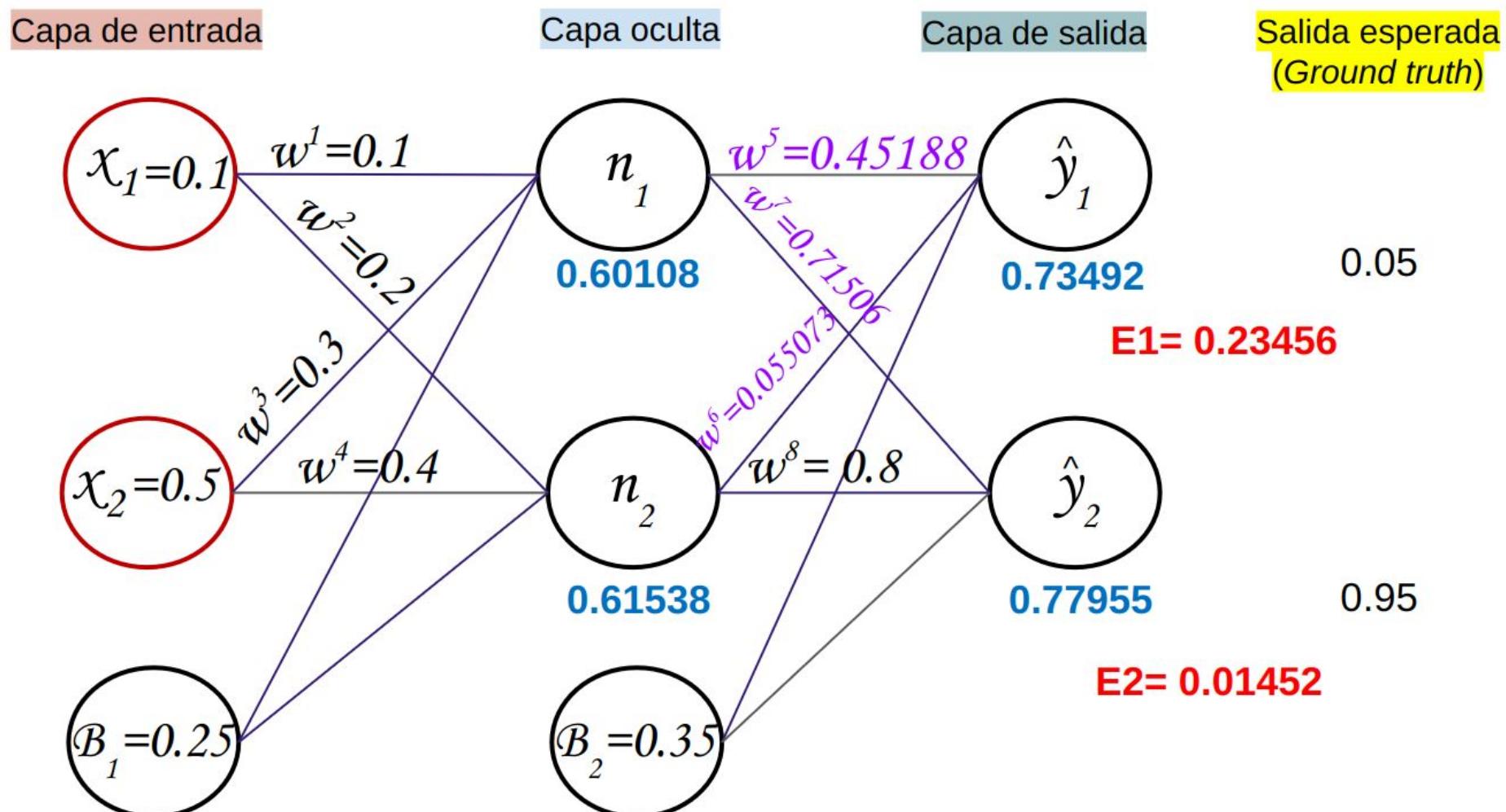
Donde  $n$  es la tasa de aprendizaje:

$$nuevo\_w7 = 0.7 - 0.6 * -0.01760$$

$$nuevo\_w7 = 0.71056$$

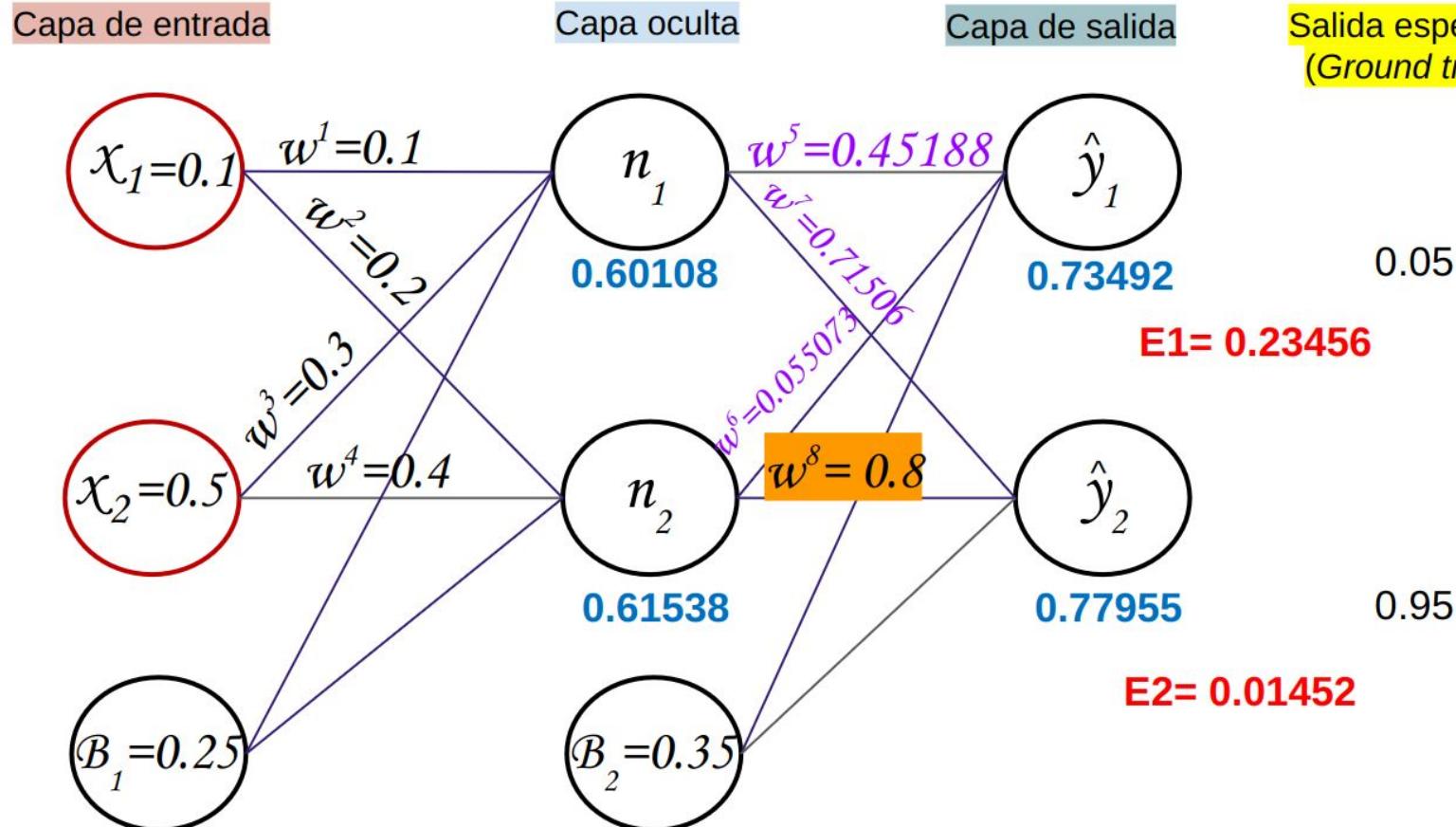


# Añadiendo el nuevo peso ( $w^7$ )



# Propagación hacia atrás

Calculando el nuevo peso de  $w_8$

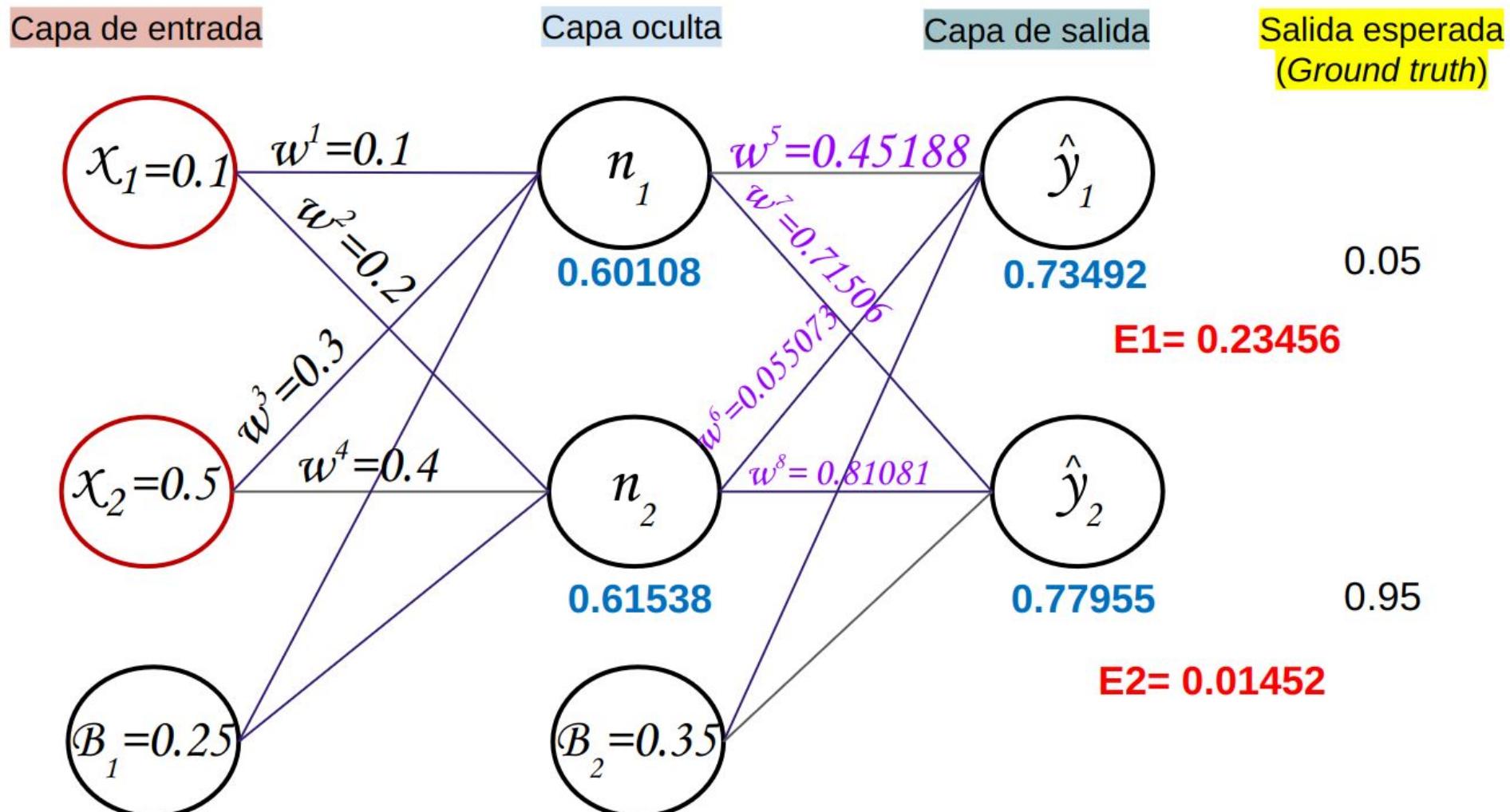


$$\frac{\partial E_{total}}{\partial w_8} = -0.01802$$

$$nuevo\_w8 = 0.81081$$



# Añadiendo el nuevo peso ( $w^8$ )



# ¿Cómo calcular los nuevos pesos de $W^1$ , $W^2$ , $W^3$ y $W^4$ ?

- Siempre buscamos encontrar la derivada parcial del error con respecto a un peso particular.
- Los pasos son similares a los realizados previamente.
- Aunque la cadena, llega a ser un poco larga en el número de componentes.

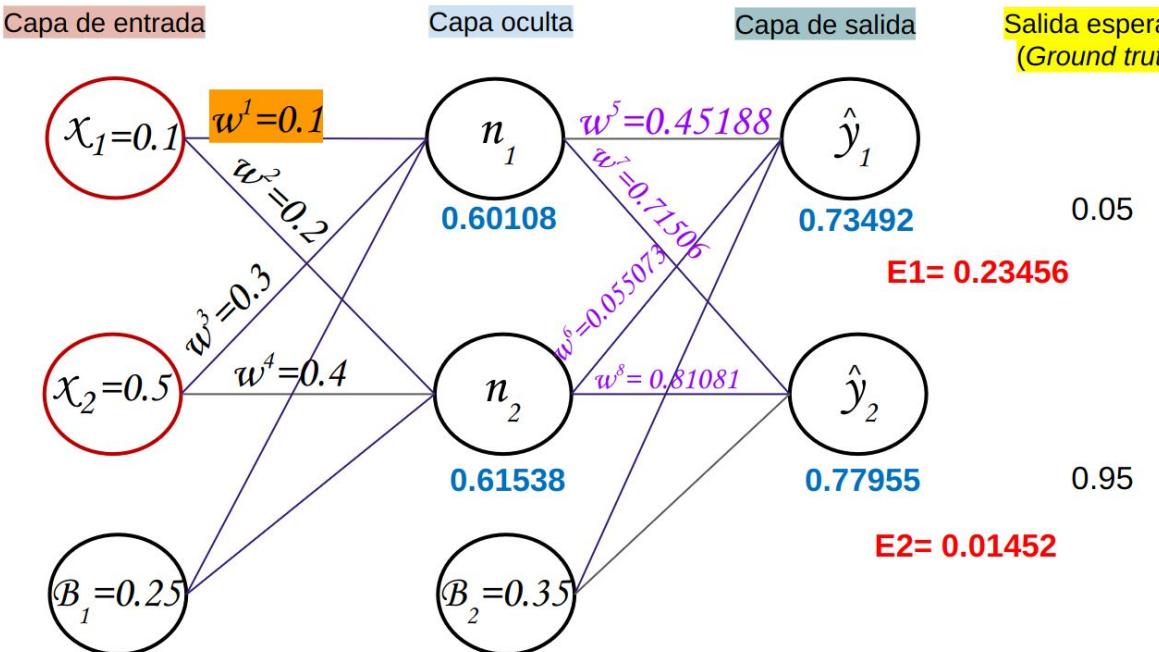
## Recuerda:

- ❖ No importa qué tan profunda sea la red neuronal, todo lo que necesitamos saber es cuánto error se propaga (contribuye) por un peso particular al error total de la red.



# Propagación hacia atrás

Calcular cuánto contribuye  $w^1$  en la  $E_1$



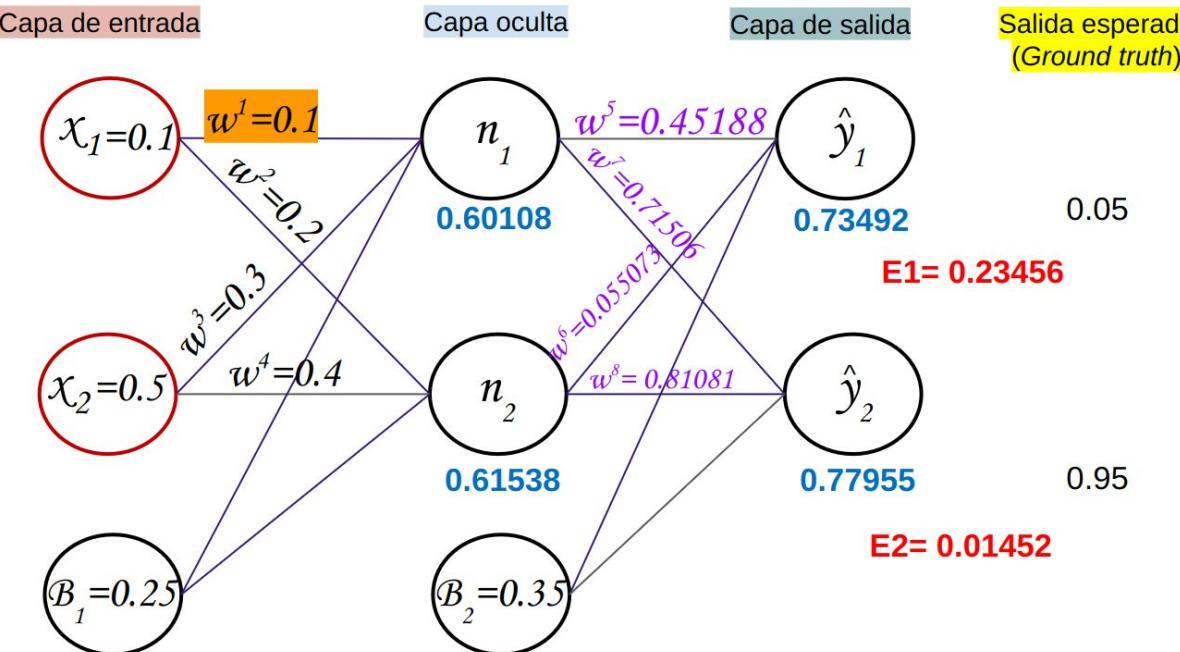
Por simplicidad, vamos a calcular de forma separada  $\frac{\partial E_1}{\partial w_1}$  y  $\frac{\partial E_2}{\partial w_1}$ . Una vez calculados, obtendremos:  $\frac{\partial E_{total}}{\partial w_1}$ .

$$\frac{\partial E_1}{\partial w_1} = \frac{\partial E_1}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial n_1} * \frac{\partial n_1}{\partial sum_{n_1}} * \frac{\partial sum_{n_1}}{\partial w_1} \quad (4)$$



# Propagación hacia atrás

Calcular el 1er componente



$$\frac{\partial E_1}{\partial w_1} = \boxed{\frac{\partial E_1}{\partial \hat{y}_1}} * \frac{\partial \hat{y}_1}{\partial \text{sum}_{\hat{y}_1}} * \frac{\partial \text{sum}_{\hat{y}_1}}{\partial n_1} * \frac{\partial n_1}{\partial \text{sum}_{n_1}} * \frac{\partial \text{sum}_{n_1}}{\partial w_1}$$

1er componente:

$$\frac{\partial E_1}{\partial \hat{y}_1} = \hat{y}_1 - \text{target}_1$$

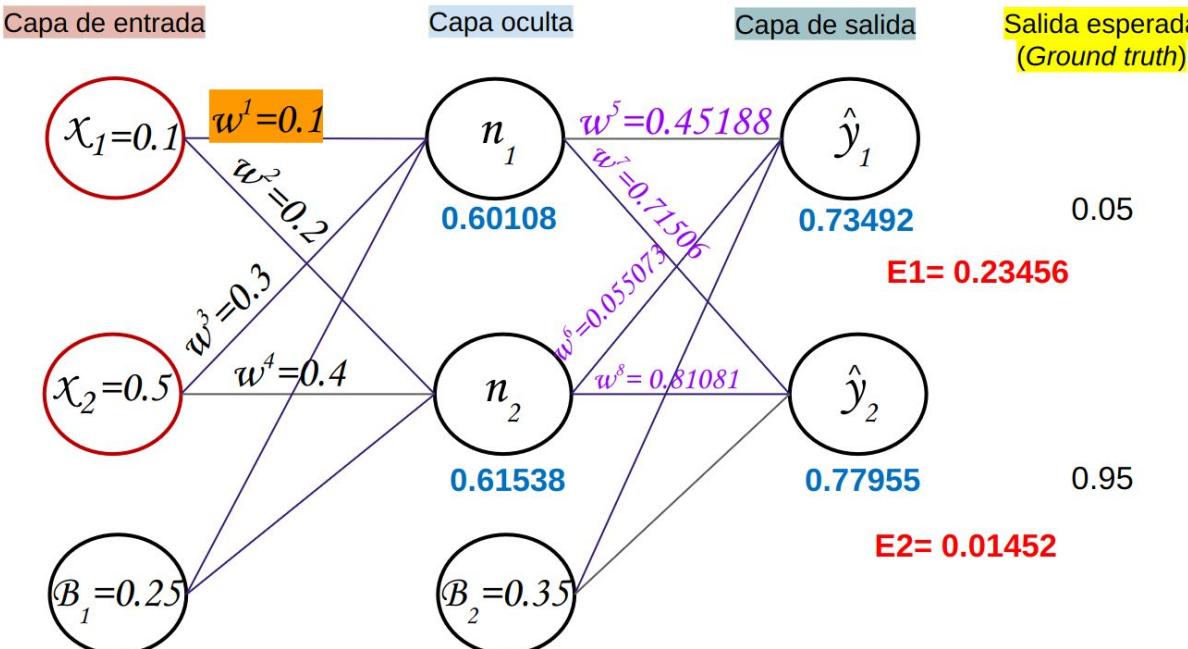
$$\frac{\partial E_1}{\partial \hat{y}_1} = 0.73492 - 0.05$$

$$\frac{\partial E_1}{\partial \hat{y}_1} = 0.68492$$



# Propagación hacia atrás

Calcular el 2do componente



$$\frac{\partial E_1}{\partial w_1} = \frac{\partial E_1}{\partial \hat{y}_1} * \boxed{\frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}}} * \frac{\partial sum_{\hat{y}_1}}{\partial n_1} * \frac{\partial n_1}{\partial sum_{n_1}} * \frac{\partial sum_{n_1}}{\partial w_1}$$

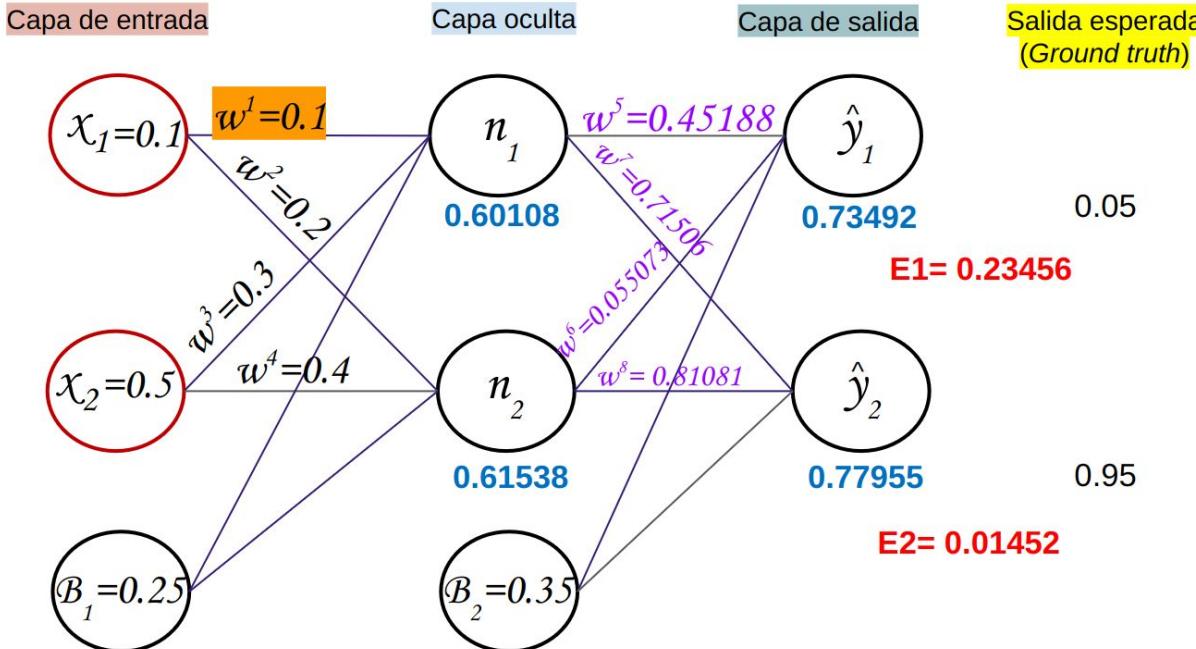
2do componente: ¡previamente ya se había calculado!

$$\frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} = 0.19480$$

Uno de los beneficios de la regla de la cadena, es que podemos reusar cálculos previos.



# Propagación hacia atrás



Calcular el 3er componente

$$\frac{\partial E_1}{\partial w_1} = \frac{\partial E_1}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \boxed{\frac{\partial sum_{\hat{y}_1}}{\partial n_1}} * \frac{\partial n_1}{\partial sum_{n_1}} * \frac{\partial sum_{n_1}}{\partial w_1}$$

3er componente:

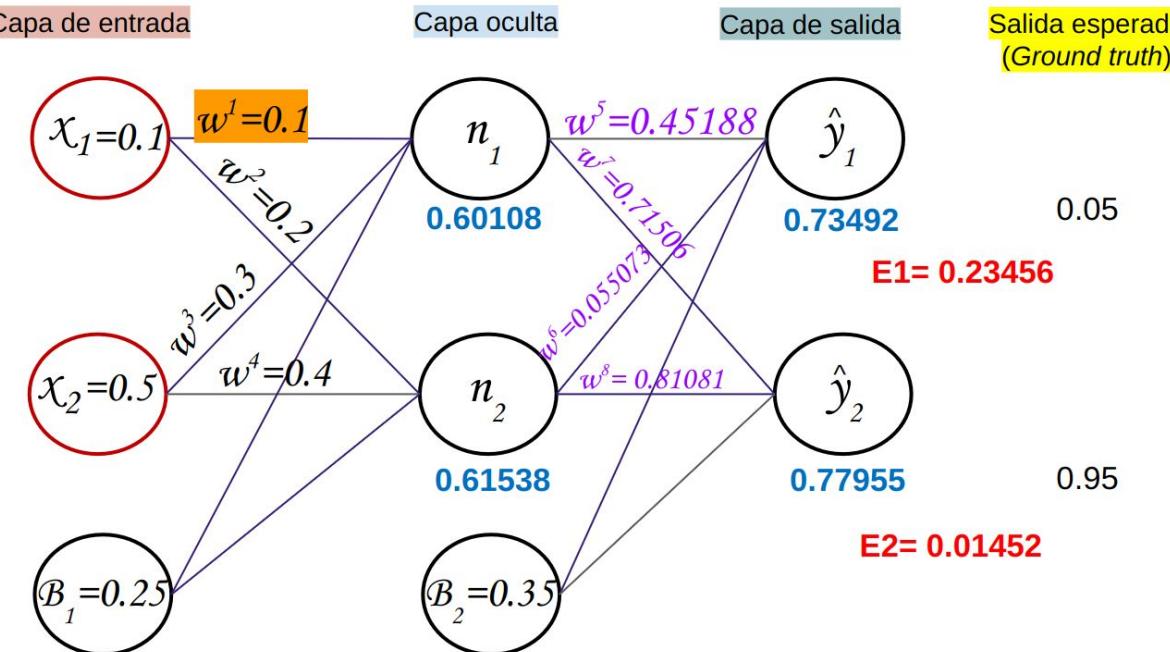
$$sum_{\hat{y}_1} = \hat{y}_1 * w5 + \hat{y}_2 * w6 + B_2$$

$$\frac{\partial sum_{\hat{y}_1}}{\partial n_1} = w5$$



# Propagación hacia atrás

Calcular el 4to componente



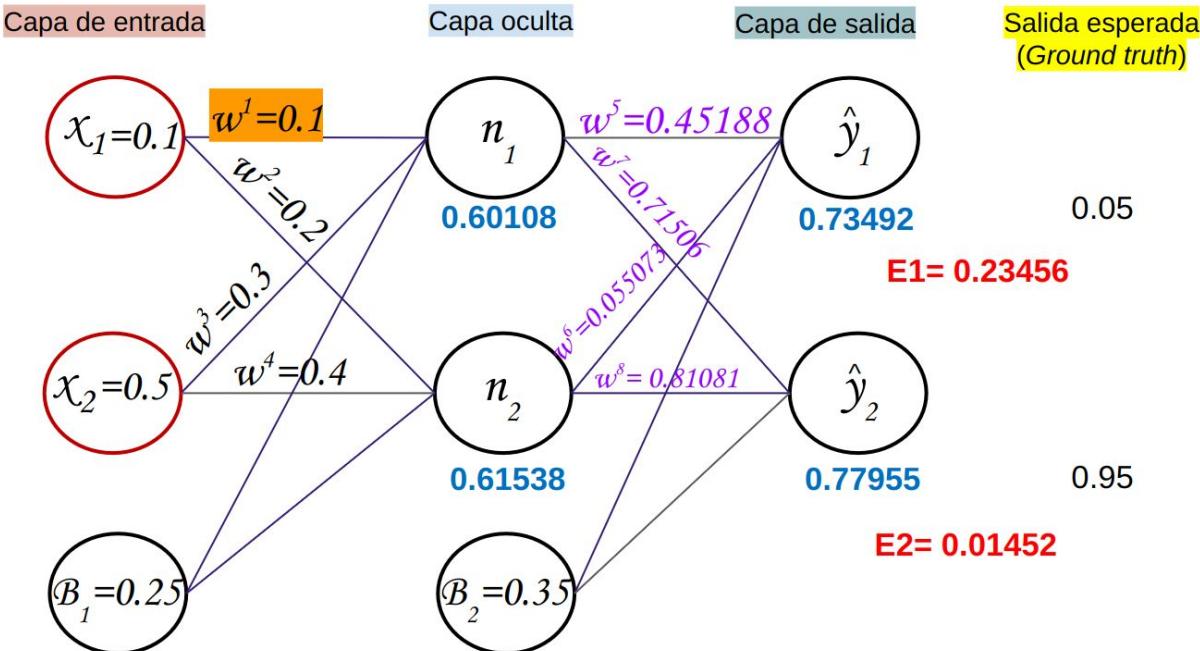
$$\frac{\partial E_1}{\partial w_1} = \frac{\partial E_1}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial n_1} * \frac{\partial n_1}{\partial sum_{n_1}} * \frac{\partial sum_{n_1}}{\partial w_1}$$

4to componente:

$$\frac{\partial n_1}{\partial sum_{n_1}} = n_1 * (1 - n_1)$$



# Propagación hacia atrás



Calcular el 5to componente

$$\frac{\partial E_1}{\partial w_1} = \frac{\partial E_1}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial n_1} * \frac{\partial n_1}{\partial sum_{n_1}} * \boxed{\frac{\partial sum_{n_1}}{\partial w_1}}$$

5to componente:

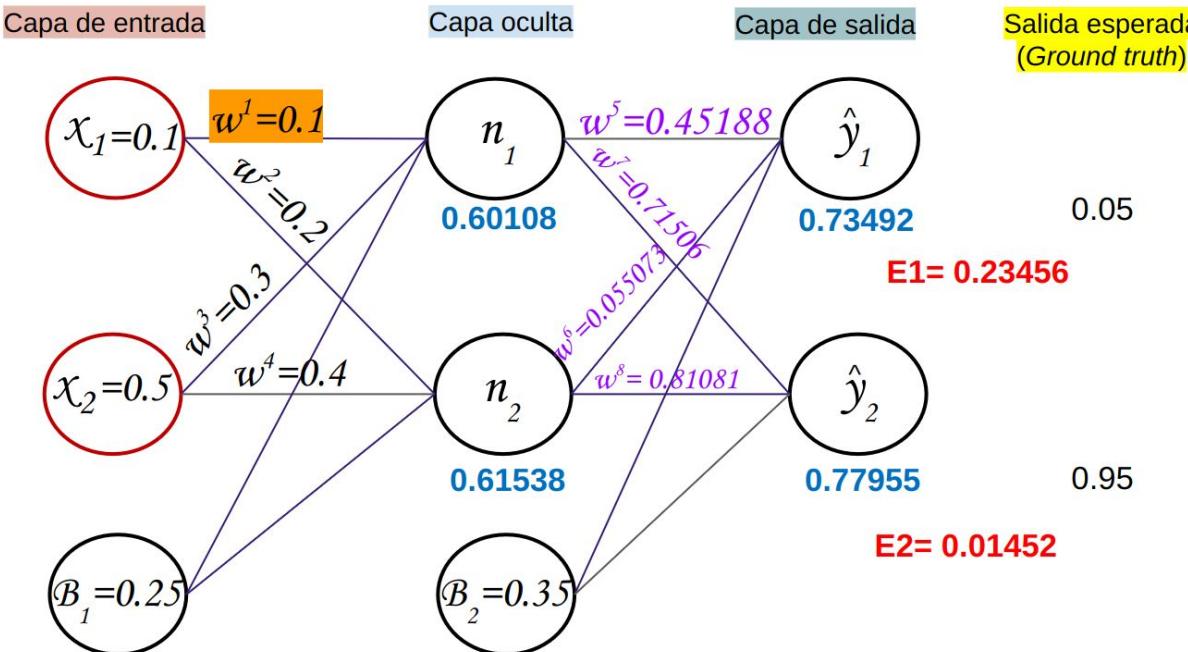
$$sum_{n_1} = (x_1 * w_1) + (x_2 * w_3) + B_1$$

$$\frac{\partial sum_{n_1}}{\partial w_1} = x_1$$



# Propagación hacia atrás

Escribiendo todo junto



$$\frac{\partial E_1}{\partial w_1} = \frac{\partial E_1}{\partial \hat{y}_1} * \frac{\partial \hat{y}_1}{\partial sum_{\hat{y}_1}} * \frac{\partial sum_{\hat{y}_1}}{\partial n_1} * \frac{\partial n_1}{\partial sum_{n_1}} * \frac{\partial sum_{n_1}}{\partial w_1}$$

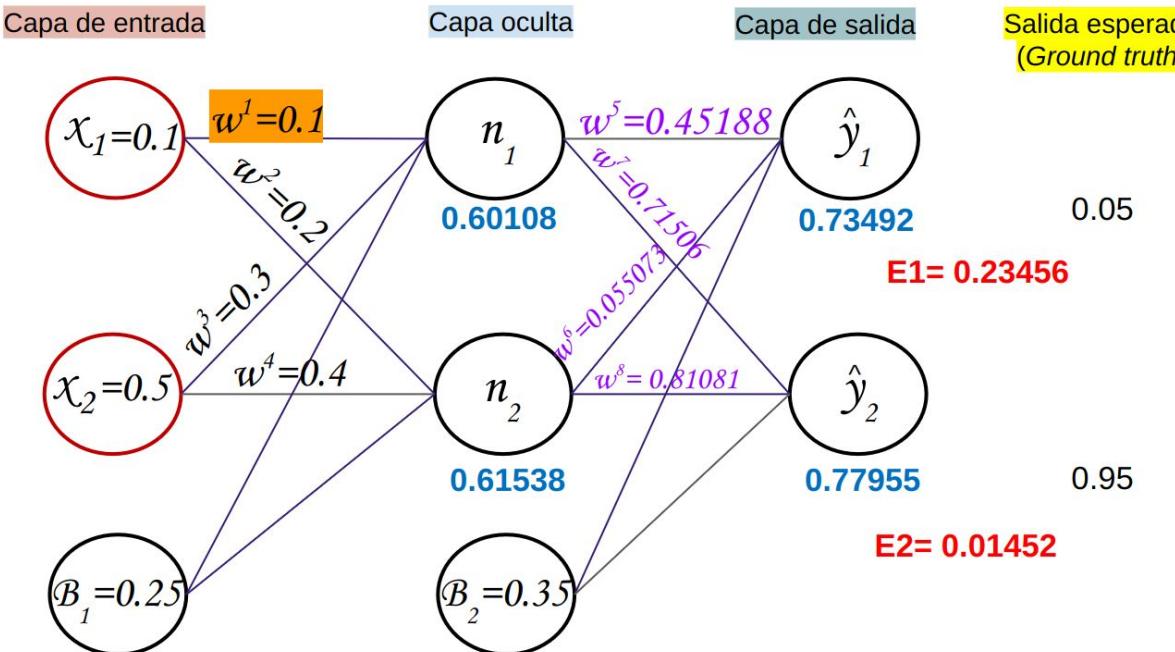
$$\frac{\partial E_1}{\partial w_1} = 0.68492 * 0.19480 * 0.5 * 0.23978 * 0.1$$

$$\frac{\partial E_1}{\partial w_1} = 0.00159$$



# Propagación hacia atrás

Calcular cuánto contribuye  $w^1$  en la  $E_2$



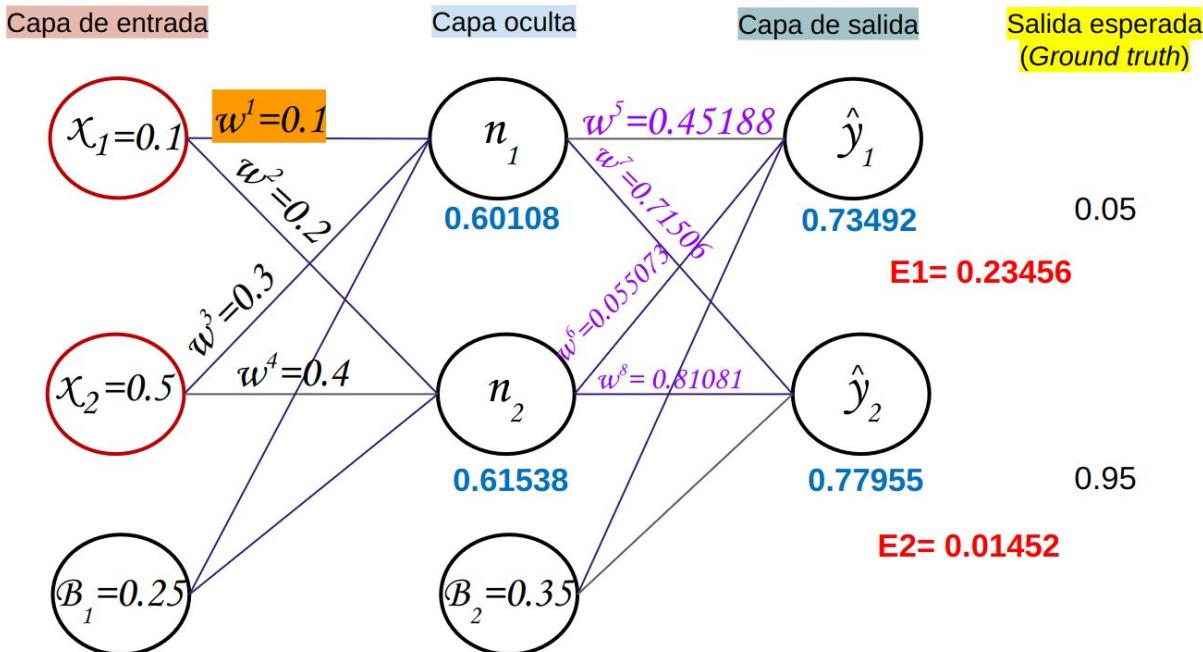
Similar a  $W_1$  con respecto a  $E_1$ , ahora lo calculamos para  $E_2$

$$\frac{\partial E_2}{\partial w_1} = \frac{\partial E_2}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial \text{sum}_{\hat{y}_2}} * \frac{\partial \text{sum}_{\hat{y}_2}}{\partial n_1} * \frac{\partial n_1}{\partial \text{sum}_{n_1}} * \frac{\partial \text{sum}_{n_1}}{\partial w_1}$$



# Propagación hacia atrás

Calcular el 1er componente



$$\frac{\partial E_2}{\partial w_1} = \boxed{\frac{\partial E_2}{\partial \hat{y}_2}} * \frac{\partial \hat{y}_2}{\partial \text{sum}_{\hat{y}_2}} * \frac{\partial \text{sum}_{\hat{y}_2}}{\partial n_1} * \frac{\partial n_1}{\partial \text{sum}_{n_1}} * \frac{\partial \text{sum}_{n_1}}{\partial w_1}$$

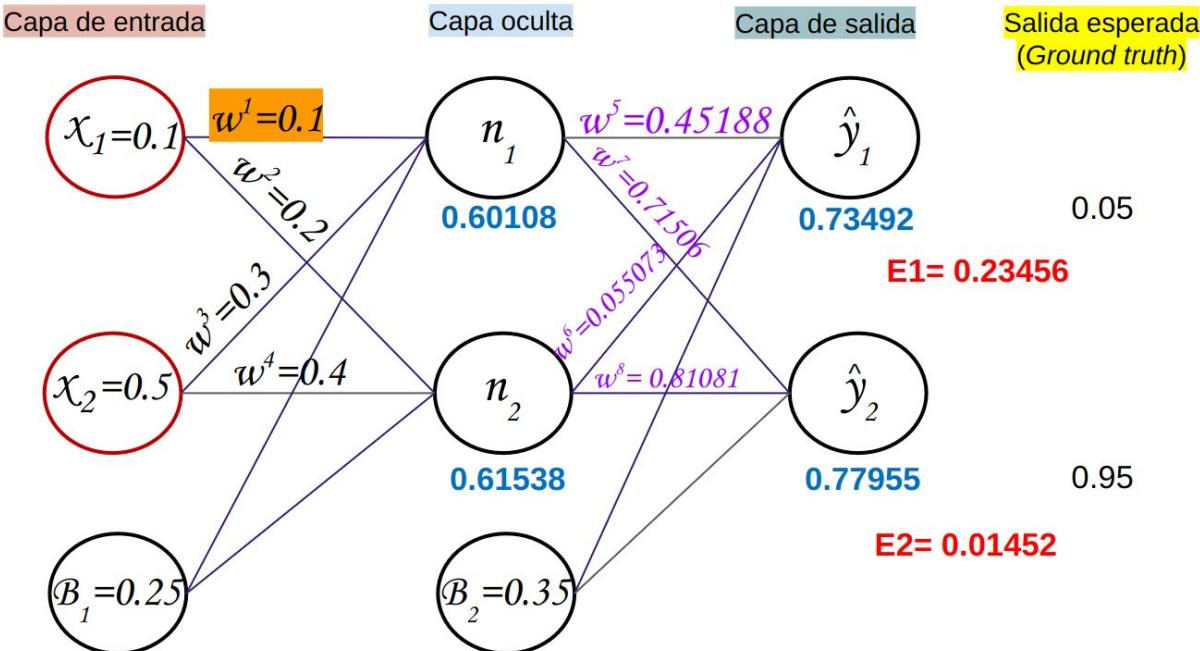
$$\frac{\partial E_2}{\partial \hat{y}_2} = \hat{y}_2 - \text{target}_2$$

$$\frac{\partial E_2}{\partial \hat{y}_2} = 0.77955 - 0.95$$

$$\frac{\partial E_2}{\partial \hat{y}_2} = -0.17044$$



# Propagación hacia atrás



Calcular el 2do componente

$$\frac{\partial E_2}{\partial w_1} = \frac{\partial E_2}{\partial \hat{y}_2} * \boxed{\frac{\partial \hat{y}_2}{\partial \text{sum}_{\hat{y}_2}}} * \frac{\partial \text{sum}_{\hat{y}_2}}{\partial n_1} * \frac{\partial n_1}{\partial \text{sum}_{n_1}} * \frac{\partial \text{sum}_{n_1}}{\partial w_1}$$

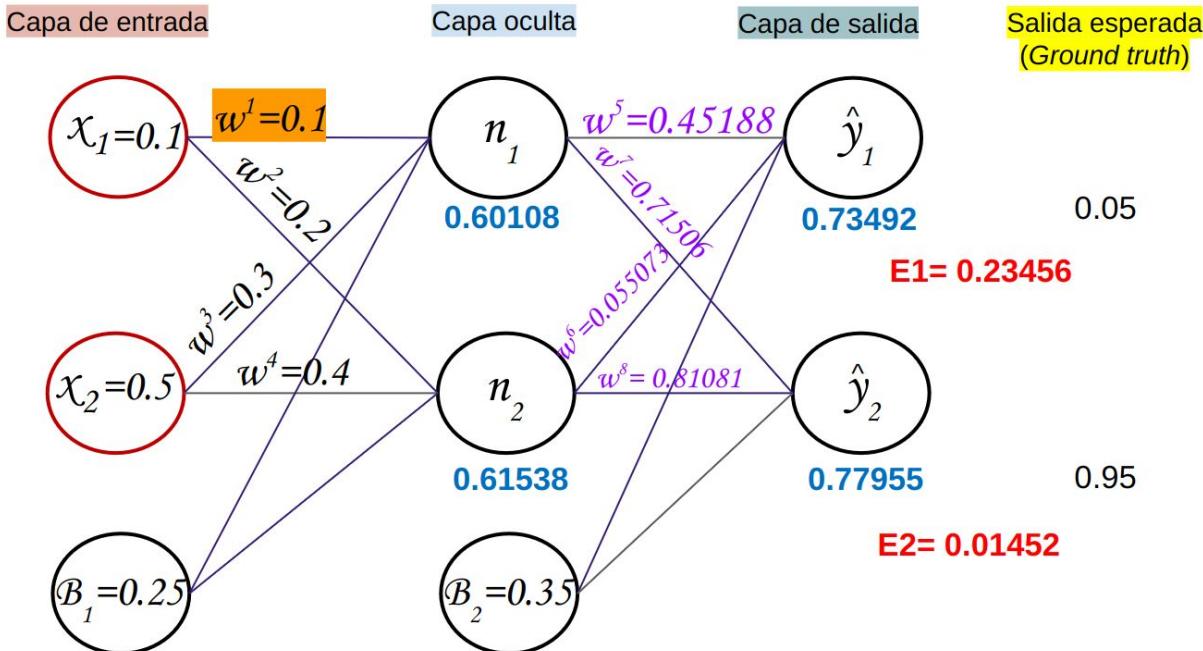
Previamente, ya lo habíamos calculado:

$$\frac{\partial \hat{y}_2}{\partial \text{sum}_{\hat{y}_2}} = \hat{y}_2(1 - \hat{y}_2)$$

$$\frac{\partial \hat{y}_2}{\partial \text{sum}_{\hat{y}_2}} = 0.17184$$



# Propagación hacia atrás



Calcular el 3er componente

$$\frac{\partial E_2}{\partial w_1} = \frac{\partial E_2}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial \text{sum}_{\hat{y}_2}} * \boxed{\frac{\partial \text{sum}_{\hat{y}_2}}{\partial n_1}} * \frac{\partial n_1}{\partial \text{sum}_{n_1}} * \frac{\partial \text{sum}_{n_1}}{\partial w_1}$$

Recordemos

$$\text{sum}_{\hat{y}_2} = \hat{y}_2 * w_7 + \hat{y}_2 * w_8 + B_2$$

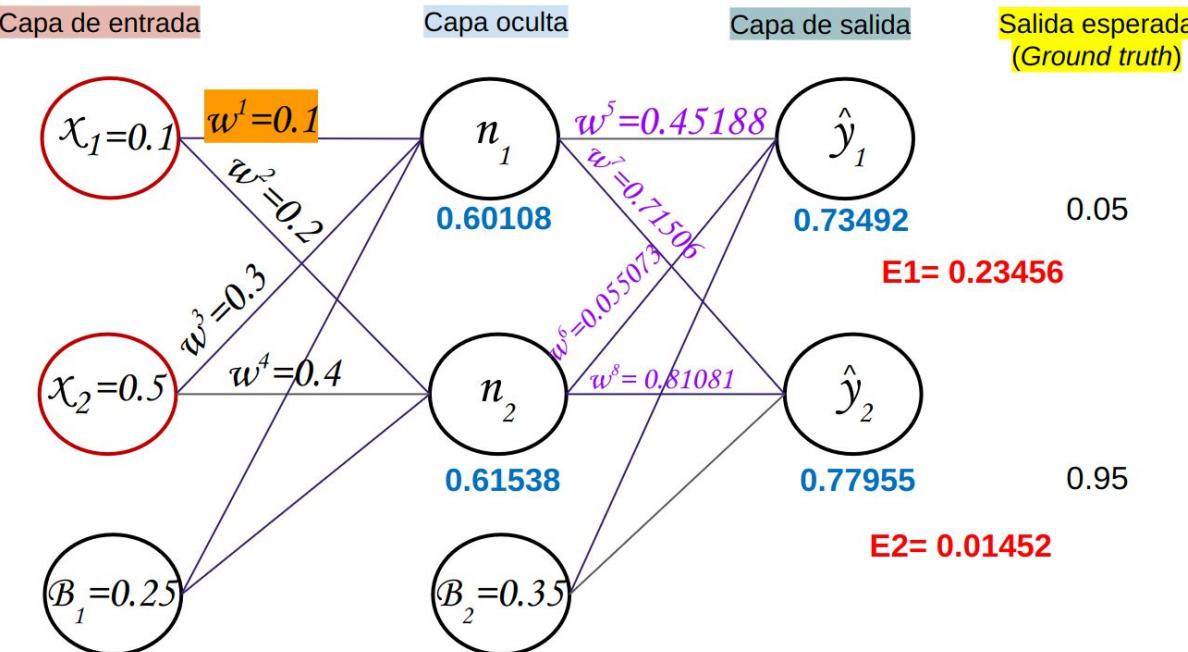
Entonces:

$$\frac{\partial \text{sum}_{\hat{y}_2}}{\partial n_1} = w_7$$



# Propagación hacia atrás

Calcular el 4to componente



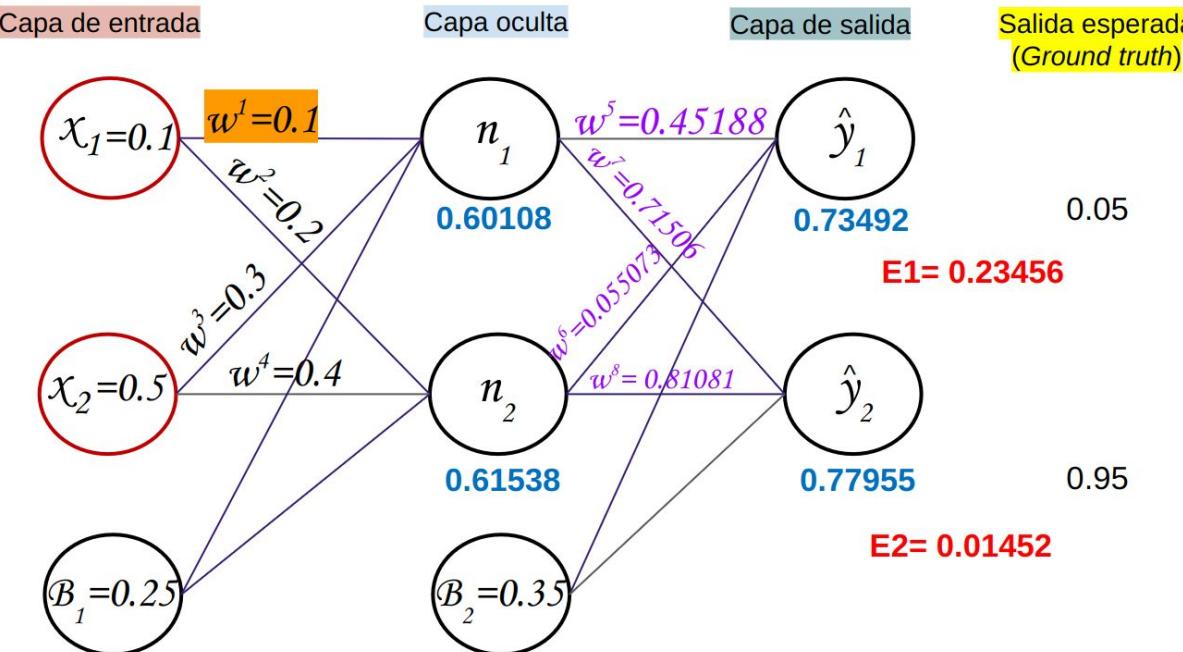
$$\frac{\partial E_2}{\partial w_1} = \frac{\partial E_2}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial sum_{\hat{y}_2}} * \frac{\partial sum_{\hat{y}_2}}{\partial n_1} * \frac{\partial n_1}{\partial sum_{n_1}} * \frac{\partial sum_{n_1}}{\partial w_1}$$

$$\frac{\partial n_1}{\partial sum_{n_1}} = n_1 * (1 - n_1)$$



# Propagación hacia atrás

Calcular el 5to componente



$$\frac{\partial E_2}{\partial w_1} = \frac{\partial E_2}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial sum_{\hat{y}_2}} * \frac{\partial sum_{\hat{y}_2}}{\partial n_1} * \frac{\partial n_1}{\partial sum_{n_1}} * \boxed{\frac{\partial sum_{n_1}}{\partial w_1}}$$

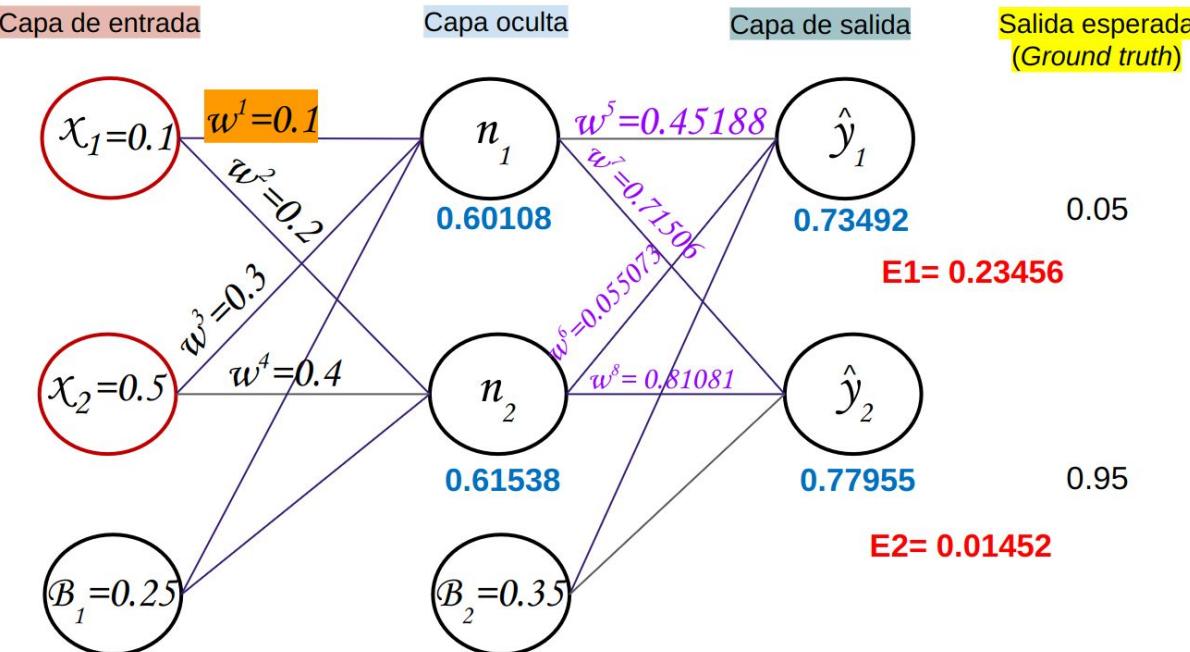
$$sum_{n_1} = (x_1 * w_1) + (x_2 * w_3) + B_1$$

$$\frac{\partial sum_{n_1}}{\partial w_1} = x_1$$



# Propagación hacia atrás

Escribiendo todo junto



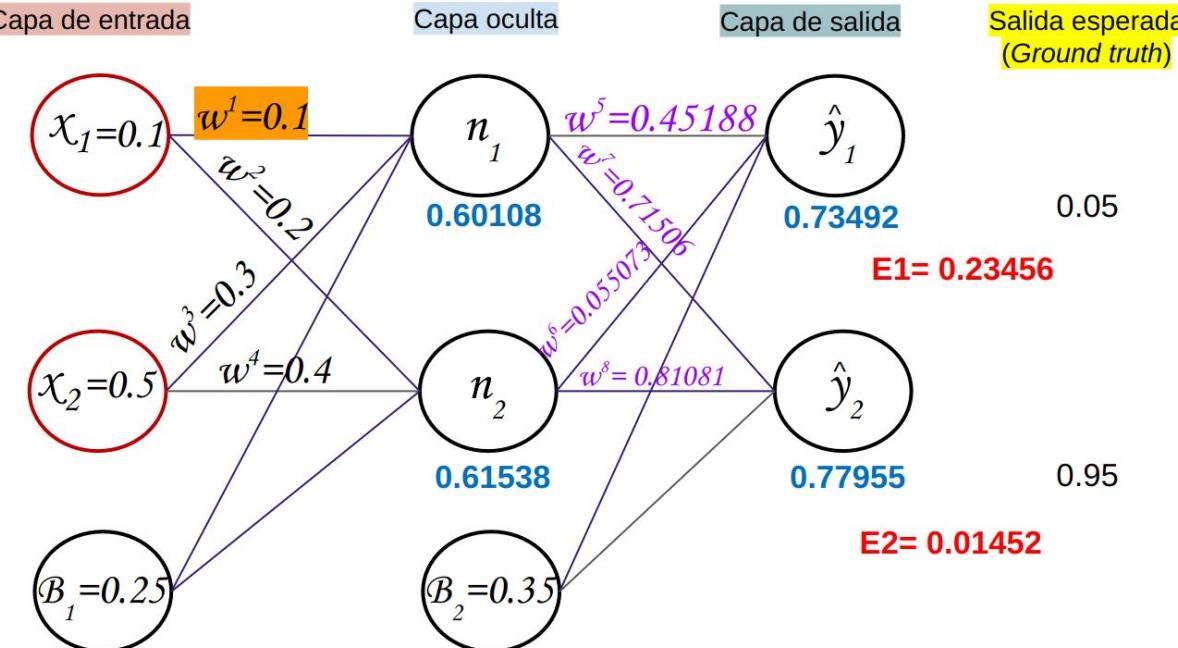
$$\frac{\partial E_2}{\partial w_1} = \frac{\partial E_2}{\partial \hat{y}_2} * \frac{\partial \hat{y}_2}{\partial sum_{\hat{y}_2}} * \frac{\partial sum_{\hat{y}_2}}{\partial n_1} * \frac{\partial n_1}{\partial sum_{n_1}} * \frac{\partial sum_{n_1}}{\partial w_1}$$

$$\frac{\partial E_2}{\partial w_1} = -0.17044 * 0.17184 * 0.7 * 0.23978 * 0.1$$

$$\frac{\partial E_2}{\partial w_1} = -0.00049$$



# Propagación hacia atrás



## Calculando el error

Ahora calculamos:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_1}{\partial w_1} + \frac{\partial E_2}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.00159 + (-0.00049)$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.0011$$

Entonces, el nuevo  $w_1$  se obtiene:

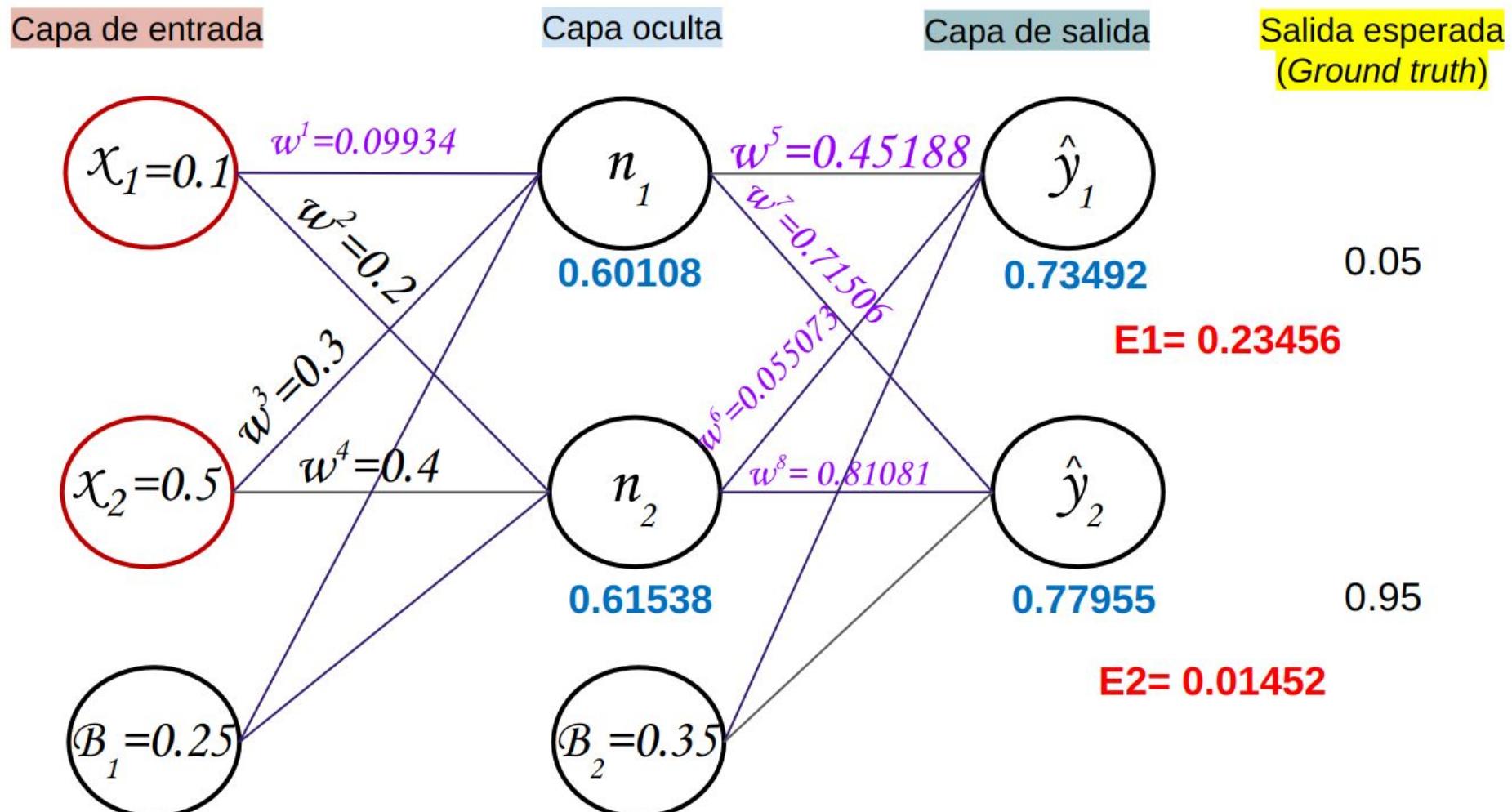
$$\text{nuevo\_w1} = w_1 - n * \frac{\partial E_{total}}{\partial w_1}$$

$$\text{nuevo\_w1} = 0.1 - 0.6 * 0.0011$$

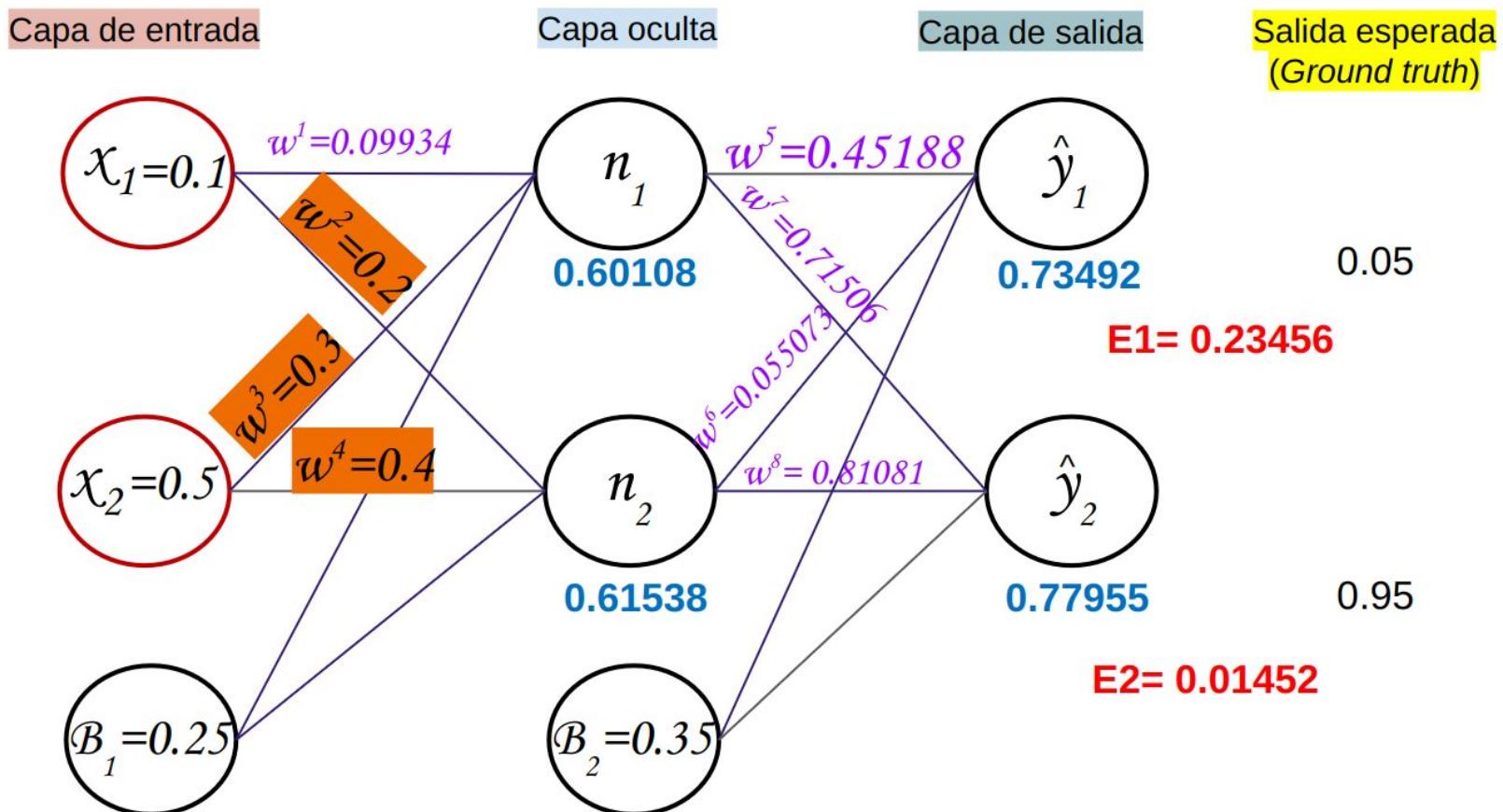
$$\text{nuevo\_w1} = 0.09934$$



# Añadiendo el nuevo peso ( $w^1$ )



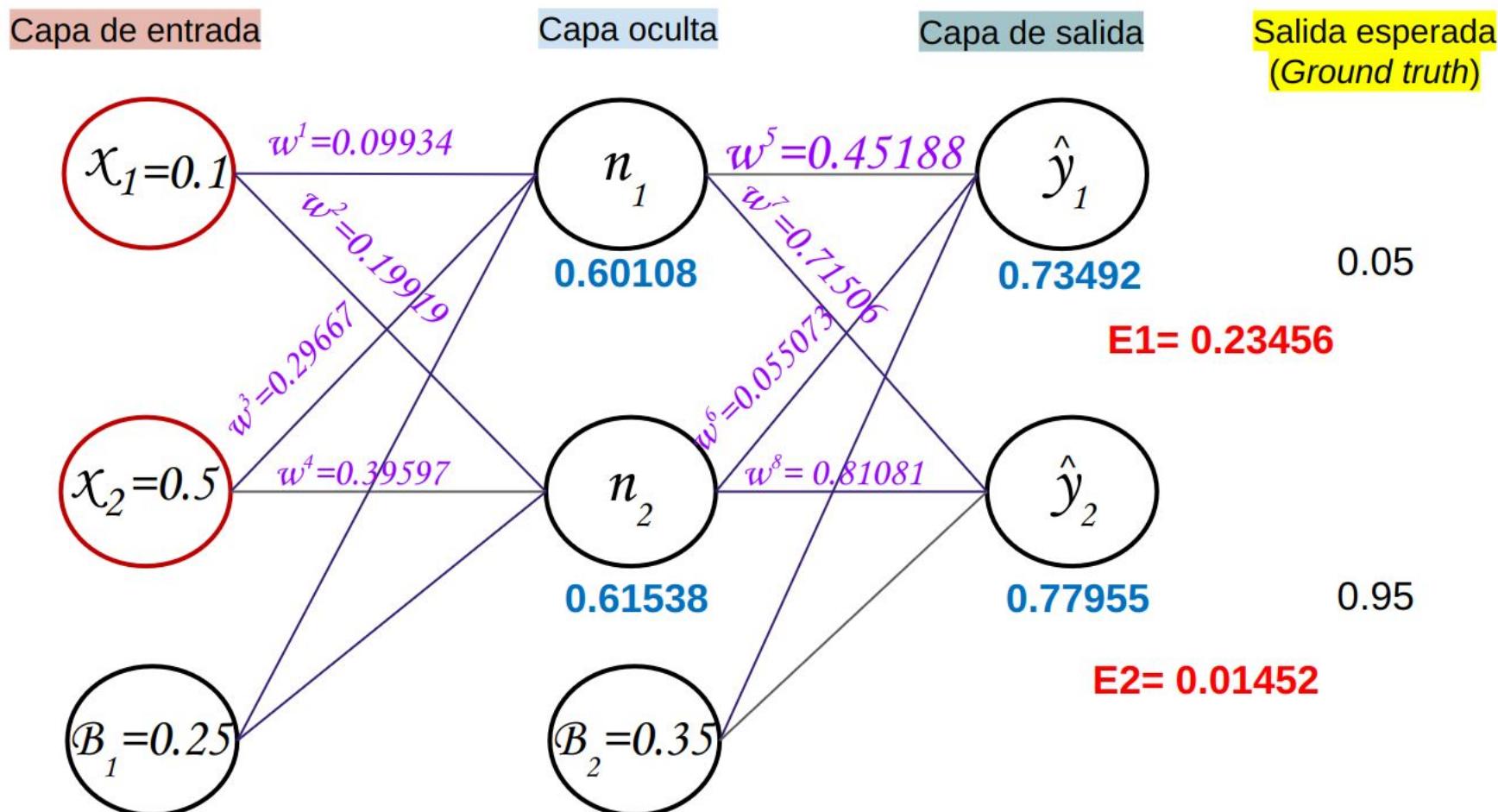
# Propagación hacia atrás



Calcula los pesos para  $w^2$ ,  $w^3$ , y  $w^4$ .



# Propagación hacia atrás: iteración 1 finalizada



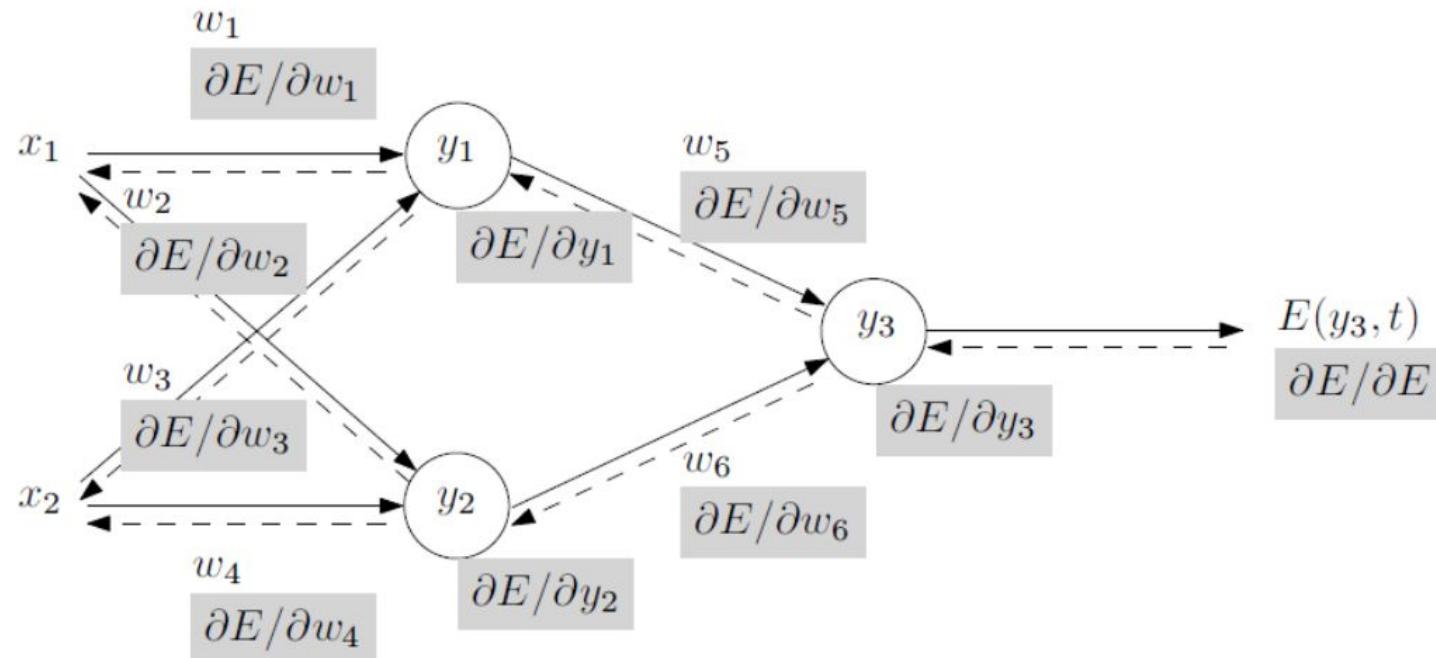
# Recuerda:

- Al concluir la propagación hacia atrás, todos los viejos pesos se sustituyen por los nuevos pesos.
- A continuación, inicia nuevamente la propagación hacia adelante y se calcula el nuevo error total.
- A partir de este nuevo error, inicia nuevamente la propagación hacia atrás, y se actualizan los pesos.
- Esto continúa hasta que el valor de la pérdida converge al mínimo.



# Esquema de la propagación hacia adelante y hacia atrás

(a) Forward pass



(b) Backward pass

Imagen tomada de Baydin et. al. Automatic Differentiation in Machine Learning: a Survey, 2018.



# Time to Code

Retropropagacion.ipynb





# Time to Code

playground.tensorflow



# Sobreajuste y regularización

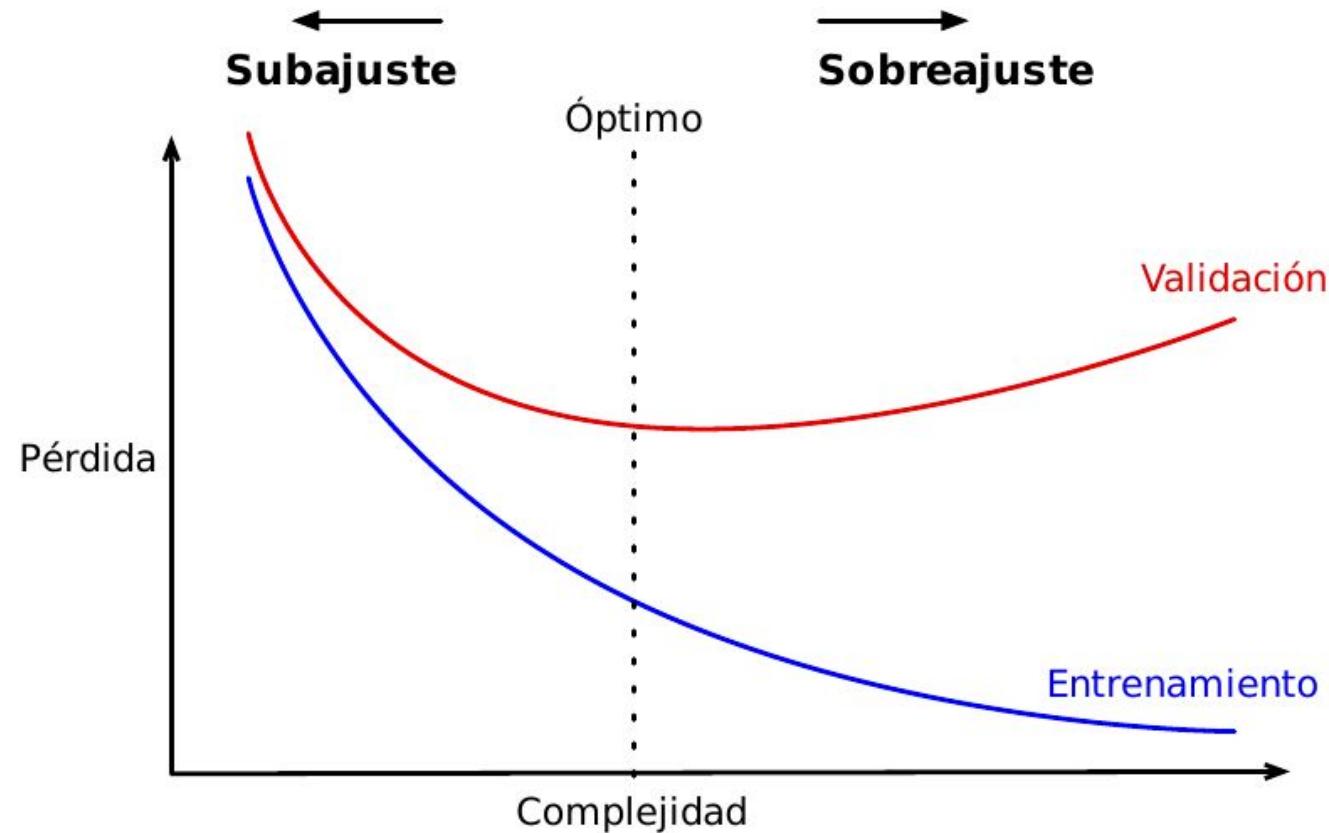


Imagen tomada de Fuentes, 2023.

# Ajuste normal

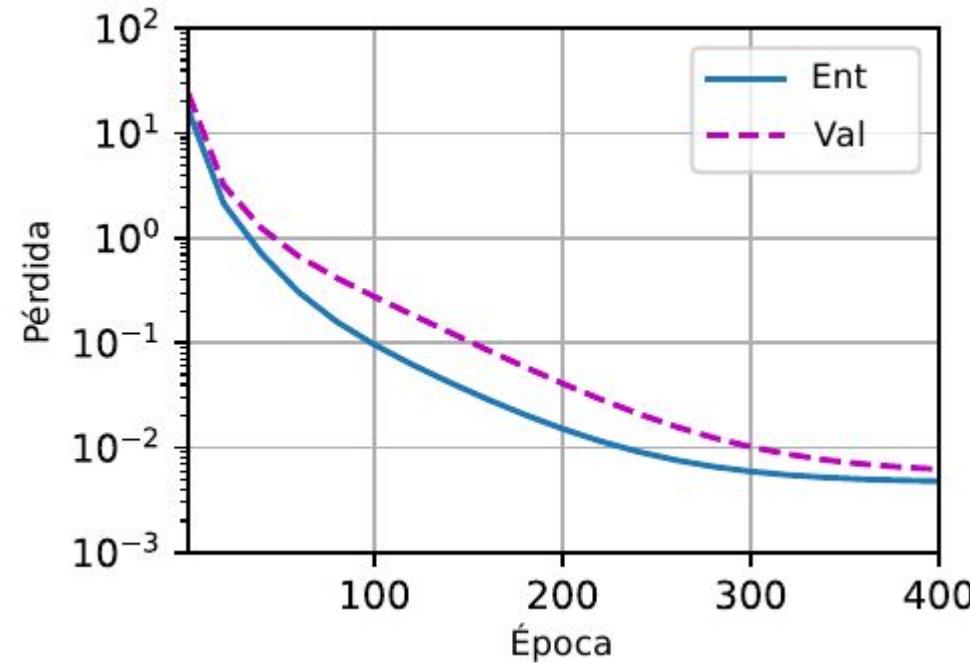


Imagen tomada de Fuentes, 2023.

# Sobreajuste: estrategias de regularización

- Penalización de la función de error
- Paro temprano
- Dropout
- Adición de ruido.
- Aumentado de datos (data augmentation)
- Normalización por lotes
- Versiones estocásticas del descenso del gradiente y variantes.



# Sobreajuste: penalización por norma

Esta regularización agrega un término de penalización en la función de costo del modelo, desalentando los modelos demasiado complejos

- Norma  $\ell_1$  (Regresión Lasso)

$$\tilde{E}(\boldsymbol{\theta}) = - \sum_{i=1}^n \{y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\} + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_1$$

La norma L1 se calcula simplemente como la suma del valor absoluto de los elementos del vector.

- Norma  $\ell_2$  (Regresión Ridge)

$$\tilde{E}(\boldsymbol{\theta}) = - \sum_{i=1}^n \{y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)\} + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2$$

La norma L2 se calcula como raíz cuadrada de la suma de los cuadrados de los elementos del vector y también se conoce como norma euclídea.



# Sobreajuste: paro temprano

Esta regularización detiene el entrenamiento cuando se cumple un determinado criterio. Por ejemplo: que la pérdida de la validación deje de disminuir, o que la precisión deje de aumentar.

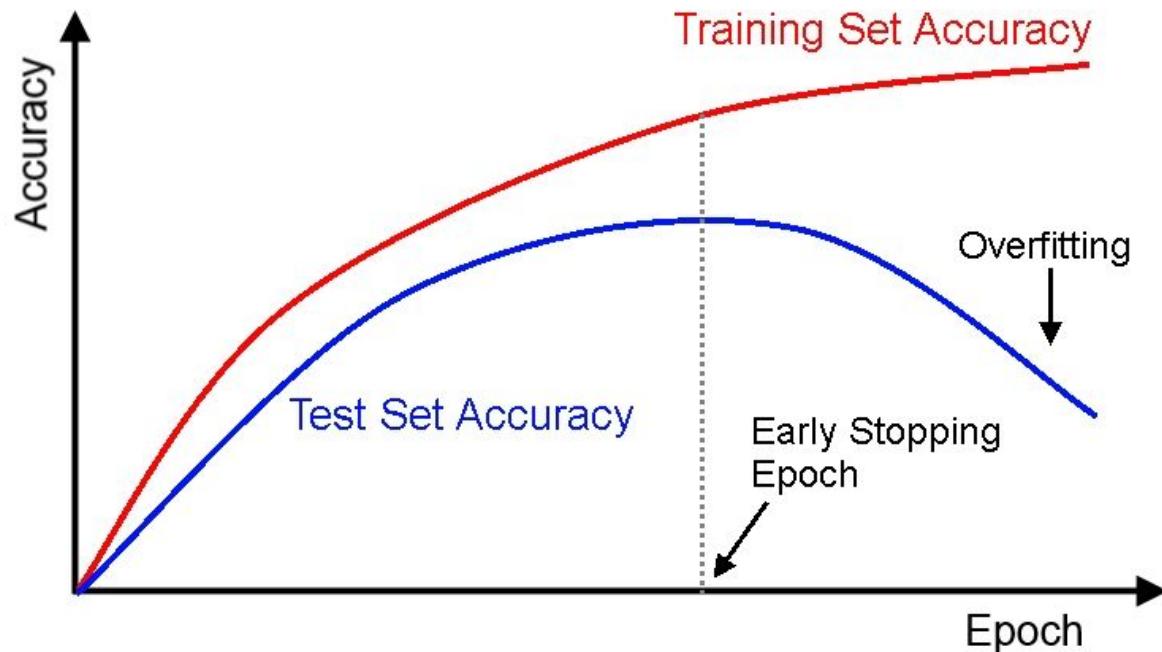


Imagen tomada de <https://deeppai.org/>

# Sobreajuste: dropout

El objetivo de esta estrategia de regularización es desactivar neuronas de forma aleatoria para evitar co-adaptación.

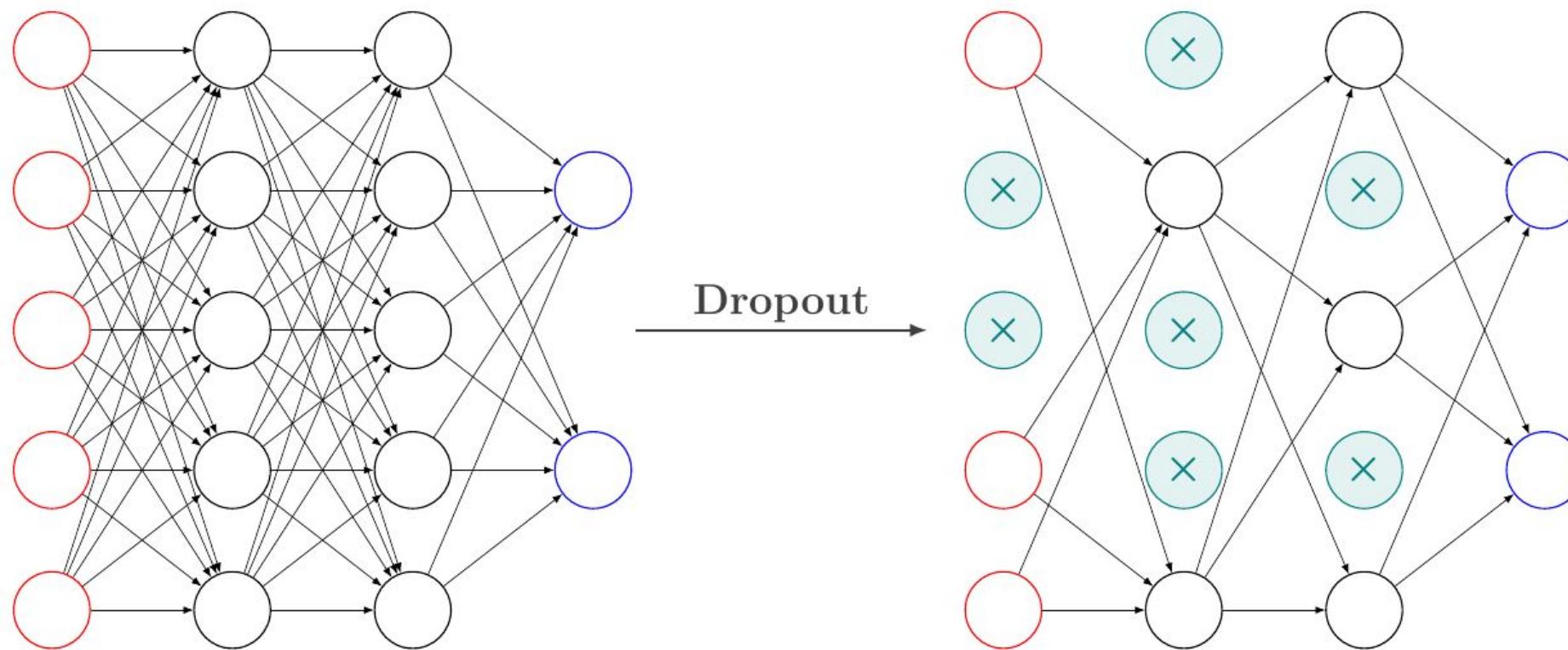


Imagen tomada de Fuentes, 2023.

# Sobreajuste: suavizado de etiquetas

- El objetivo de esta estrategia es agregar ruido a la representación 1 de  $k$  de las etiquetas.

De tal forma que, se reemplazan los ceros con

$$\frac{\epsilon}{k-1}$$

y los unos con

$$1 - \epsilon$$

- Por ejemplo, para  $K = 5$ , la representación de la etiqueta para la clase 3 sería:

$$[0, 0, 1, 0, 0]$$

- La etiqueta suavizada con  $\epsilon = 0.01$  es

$$[0.0025, 0.0025, 0.99, 0.0025, 0.0025]$$





# Time to Code

Dropout



# Explosión y desvanecimiento del gradiente

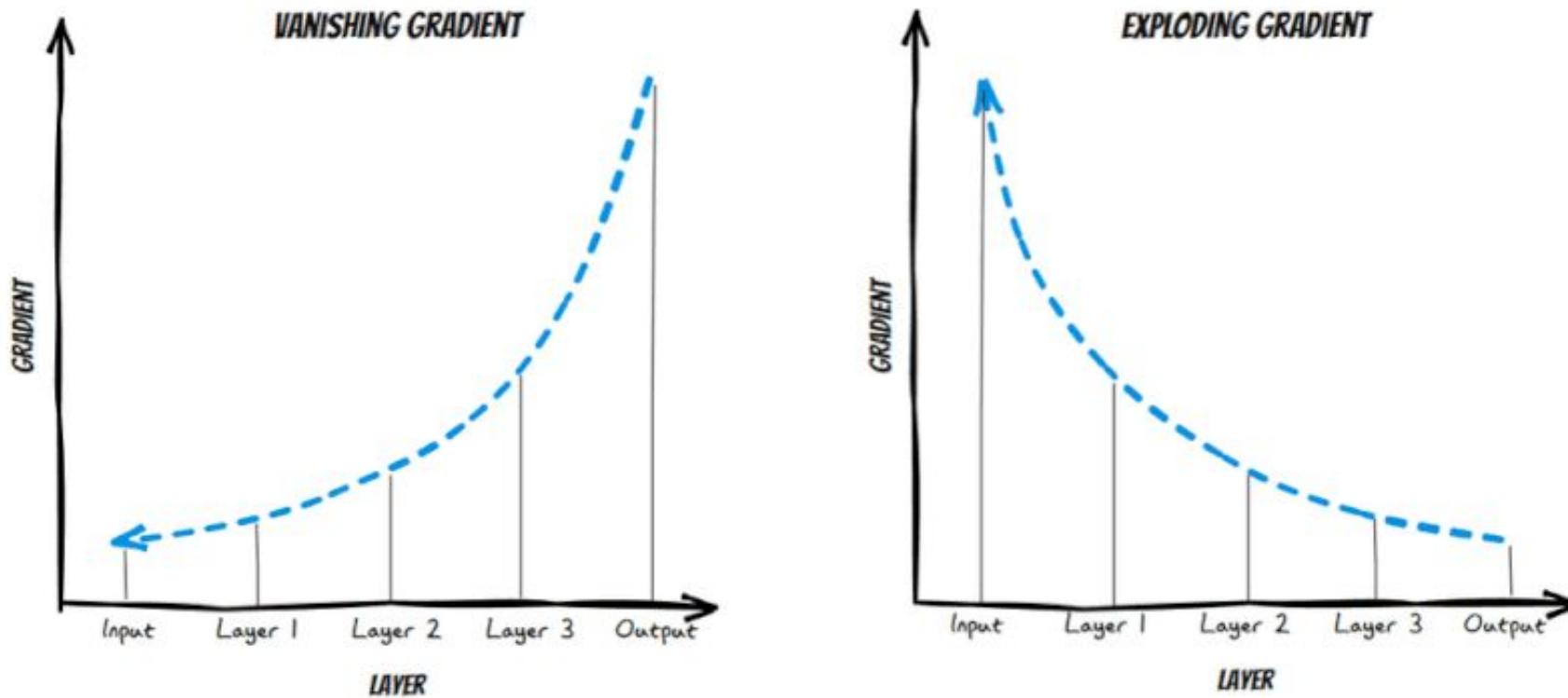


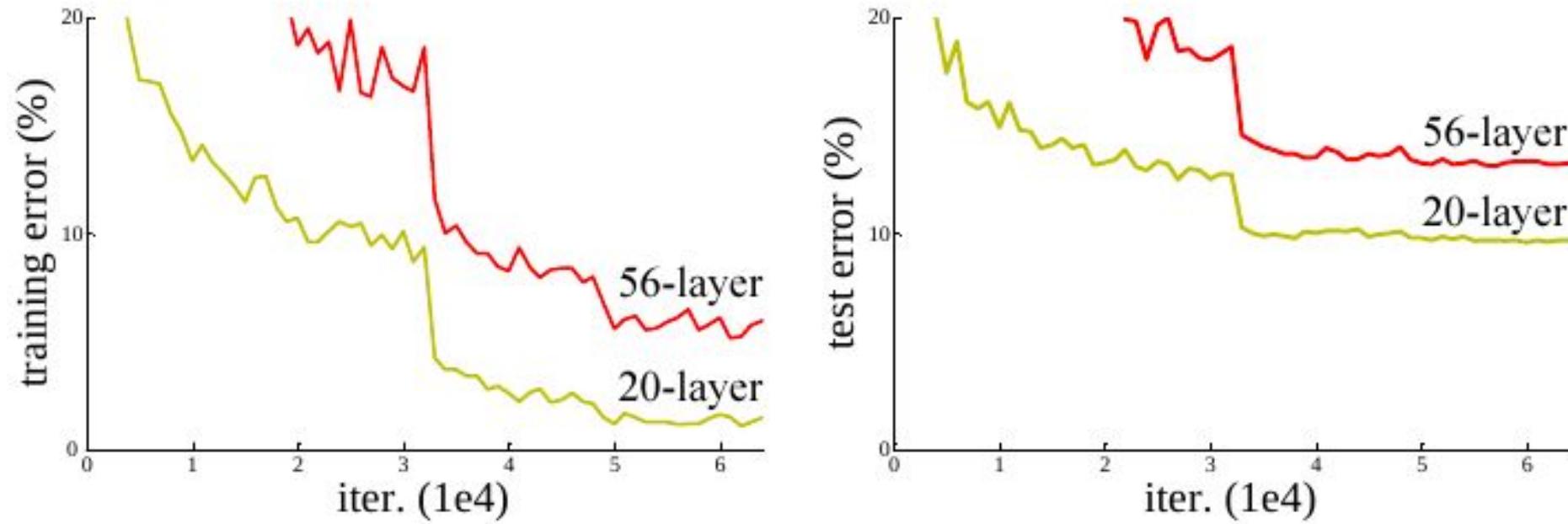
Imagen tomada de Bhoomkar, 2023.

# Desvanecimiento del gradiente

- Durante el entrenamiento de una red ocurre: la propagación hacia adelante y la propagación hacia atrás.
- Durante la propagación hacia atrás, se calcula el gradiente (derivada del error).
- Este error se propaga de la última capa hasta la primera.
- En redes muy profundas, el gradiente se desvanece (se vuelve 0).
- En inglés a este problema se le conoce como: *the vanishing gradient problem*.



# Graficando el problema del desvanecimiento del gradiente

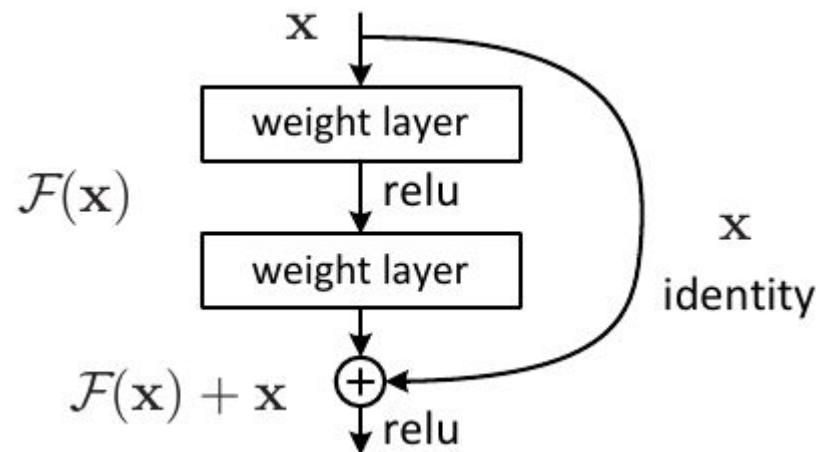


Tasa de error en el conjunto de entrenamiento (izquierda) y en el conjunto de prueba (derecha) en el conjunto de CIFAR-10.

Imagen tomada de Kaiming, 2015.

# Conexiones residuales (residual conexions)

Es una estrategia para mitigar el problema del desvanecimiento del gradiente. Surgen de la analogía de que las células piramidales del cerebro se comunican con áreas profundas del encéfalo.



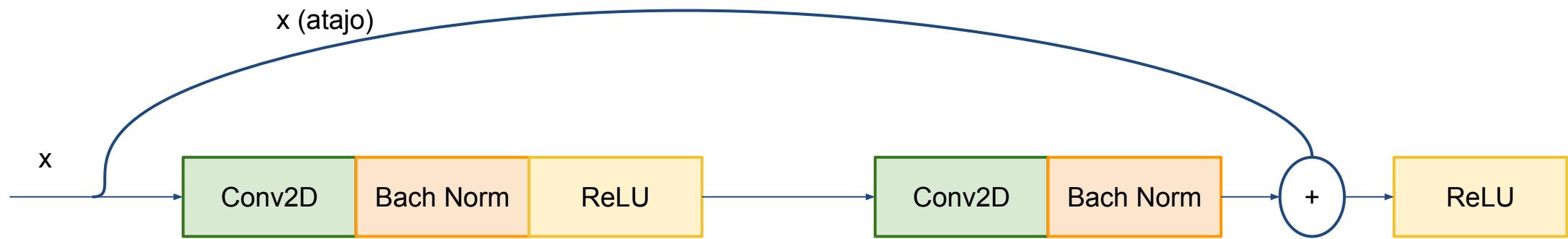
Un bloque residual se expresa como:  
 $H(x) = F(x) + x$

Se compone de i) una ruta residual  $F(x)$  (izquierda) y ii)  
una conexión atajo  $x$  (derecha).

Imagen tomada de Kaiming, 2015.

# Bloque identidad

Durante el entrenamiento, aprendemos 2 funciones:  $F(x)$  y la función identidad  $I(x) = x$  y a través de una suma se combinan ambos resultados.

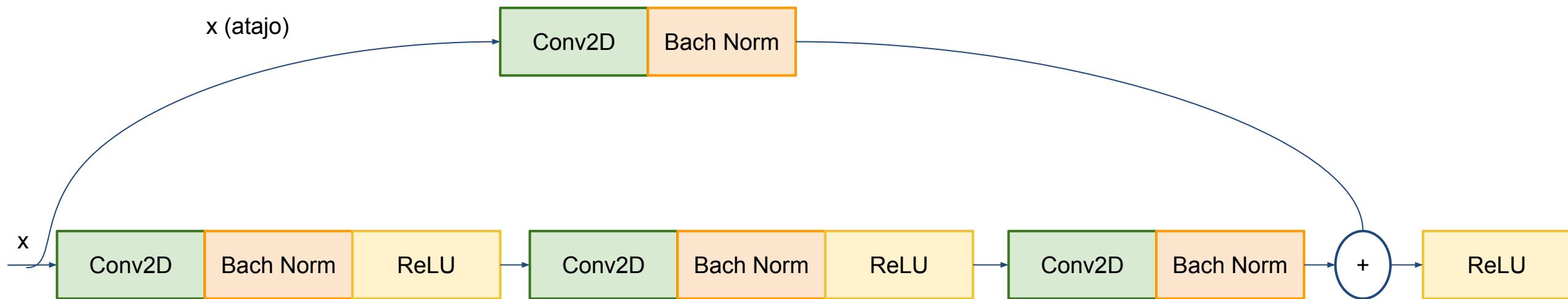


Dónde  $x$  y  $F(x)$  son de la misma dimensión.



# Bloque convolucional

Son bloques donde las dimensiones de  $F(x)$  no es el mismo que  $x$ .



Se añade una convolucional con el objetivo de  
reducir la dimensión de  $x$



# Conexiones residuales (residual conexions)

- ResNet: Fue la red ganadora en el concurso *ImageNet Large Scale Visual Recognition Challenge 2015* (*ILSVRC2015*).
- Por primera vez se permitió entrenar arquitecturas profundas (más de 100 capas)

**Deep Residual Learning for Image Recognition**

Kaiming He      Xiangyu Zhang      Shaoqing Ren      Jian Sun  
Microsoft Research

{kahe, v-xiangz, v-shren, jiansun}@microsoft.com



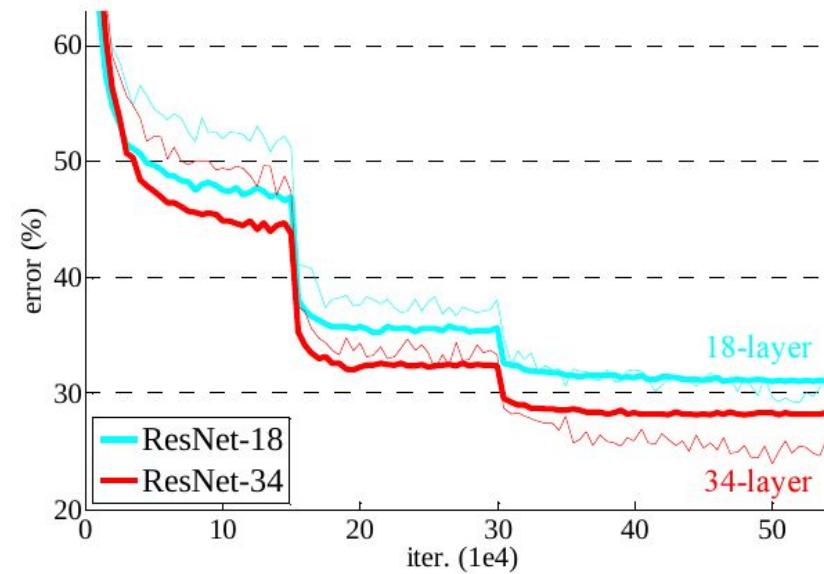
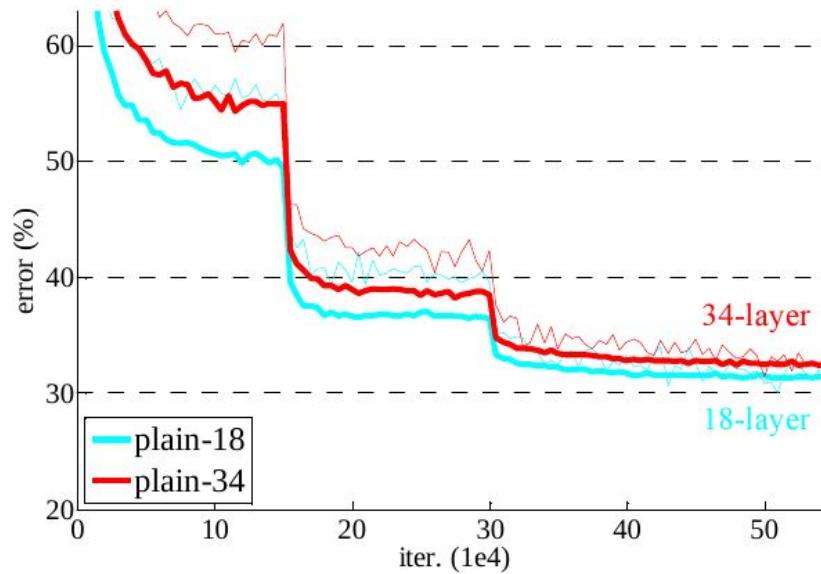
# Resultados en el reto del 2015: ResNet

- 1er lugar en tarea de clasificación.
- 1er lugar en detección (ImageNet)
- 1er lugar en localización (ImageNet)
- 1er lugar en detección (COCO)
- 1er lugar en segmentación (COCO)

Red	Año	Capas	Error %
AlexNet	2013	8	11.7
VGG	2014	19	7.3
Inception	2014	22	6.7
ResNet	2015	152	3.6



# Comparativa de rendimiento sobre ImageNet

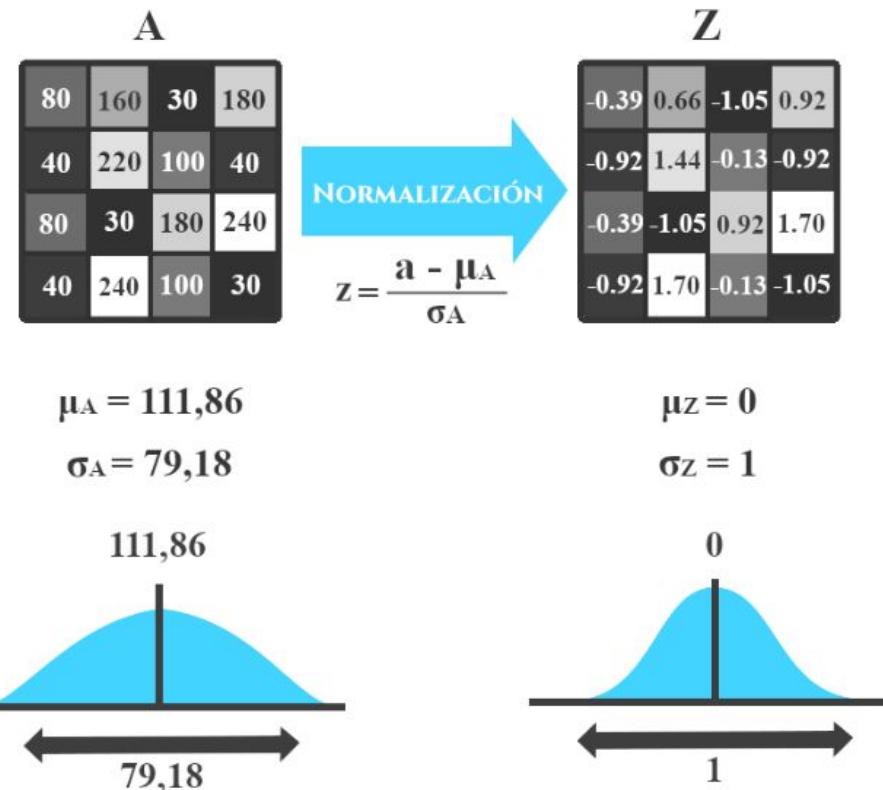


Red convolucional (izquierda) y ResNet (derecha). Se observa la tasa de error durante el entrenamiento (color verde) y de validación (color rojo).

Imagen tomada de Kaiming, 2015.

# Capas de normalización: normalización por lotes

Consiste en normalizar la entrada de cada capa restando la media del minilote y dividiéndola por la desviación estándar del minilote.



Una matriz de datos  $A$ , con media  $\mu_A = 111.86$  y desviación típica  $\delta_A = 79.18$  puede ser convertida en una nueva distribución  $Z$  con media  $\mu_Z = 0$  y desviación  $\delta_Z = 1$ , es decir, una distribución normal. Para ello solo es necesario aplicar la fórmula a cada valor  $a$  de la primera distribución.

Imagen tomada de Rodríguez Abril, 2021.

# Capas de normalización: normalización por lotes

La normalización por lotes (BN) consta de dos algoritmos.

- El algoritmo 1 es la transformación de la entrada original de una capa  $x$  al valor desplazado y normalizado  $y$ .
- El algoritmo 2 es el entrenamiento general de una red normalizada por lotes.

En ambos algoritmos, el valor  $\epsilon$  se inserta para evitar dividir por 0 y se selecciona a propósito para que sea insignificantemente pequeño (por ejemplo,  $\sim 10(-8)$ ).



# Capas de normalización: normalización por lotes

**Entrada:** Valores de  $x$  sobre un minilote:  $B = x_1, \dots, m$ ;

Parámetros:  $\sigma, \beta$

Media y varianza del lote:

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

$$\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu_B)^2$$

Normalización:

$$\hat{x}^{(i)} \leftarrow \frac{x^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Escalado y desplazamiento:

$$y^{(i)} \leftarrow \gamma \odot \hat{x}^{(i)} + \beta$$

donde  $\odot$  es el producto de Hadamard (elemento a elemento),  $\epsilon$  es un valor pequeño y  $m$  es el tamaño del lote.



# Capas de normalización: normalización por lotes

## Algoritmo 2:

1. Normaliza la red por minilotes
2. Entrena la red con propagación hacia atrás.
3. Transforma estadísticos del lote a estadísticos de población

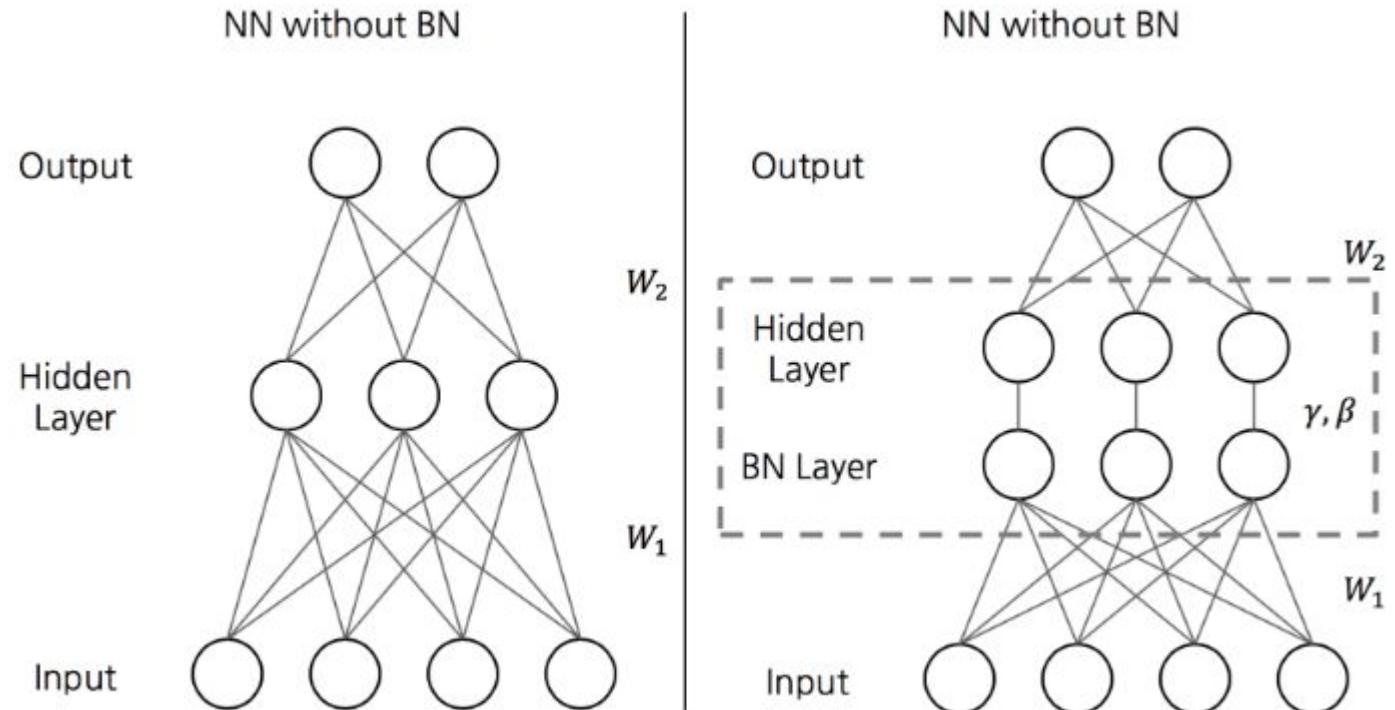


Imagen tomada de <https://collab.dvb.bayern/display/TUMIfdv/Batch+Normalization>

# Capas de normalización: normalización por lotes

## Beneficios

- Acelera el entrenamiento
- Permite tasas de aprendizaje más grandes
- Actúa como un tipo de regularizador
- Facilita la creación de redes profundas



# Problemas con el descenso del gradiente

- Desvanecimiento del gradiente
- Explosión del gradiente
- Mínimos locales y puntos de silla



# Mínimos locales y puntos de silla

- Los mínimos locales imitan un mínimo global.
- La pendiente de la función de costos aumenta a ambos lados del punto actual.

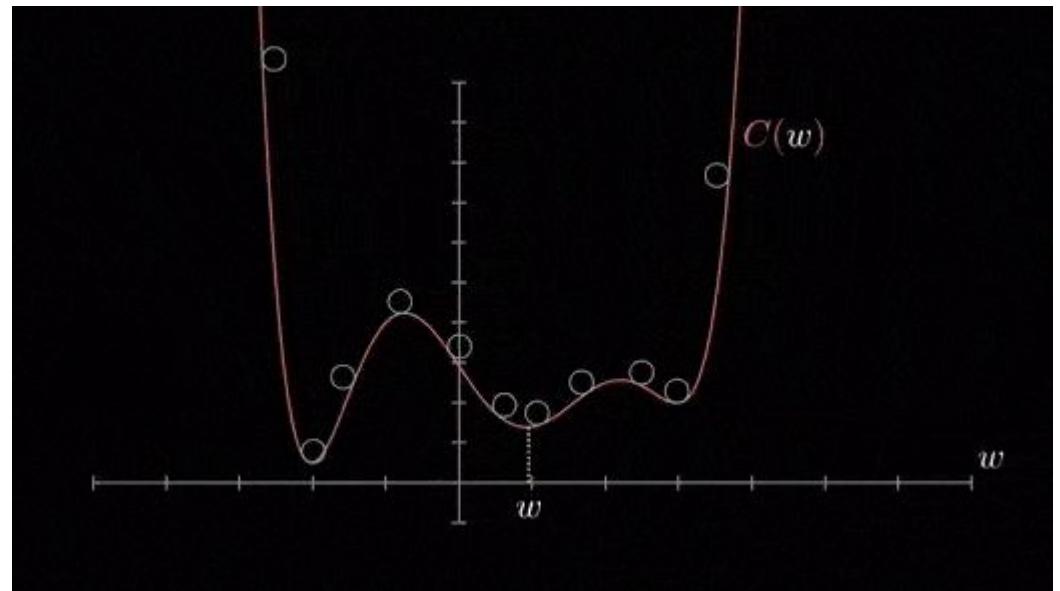


Imagen tomada de Venugopal, 2020.

# Variantes del descenso del gradiente

## Descenso del gradiente estocástico (SGD)

Versión optimizada que realiza menos comparaciones por iteración.

Se escoge de manera aleatoria los puntos con los que se calculan las derivadas. De tal forma que se selecciona un **subconjunto** de datos (mini-lote) para calcular la función de costo.

Ventaja: reduce la elevada carga computacional.

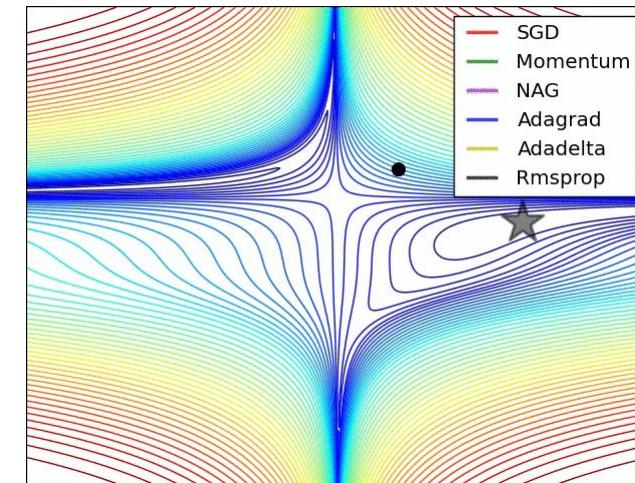
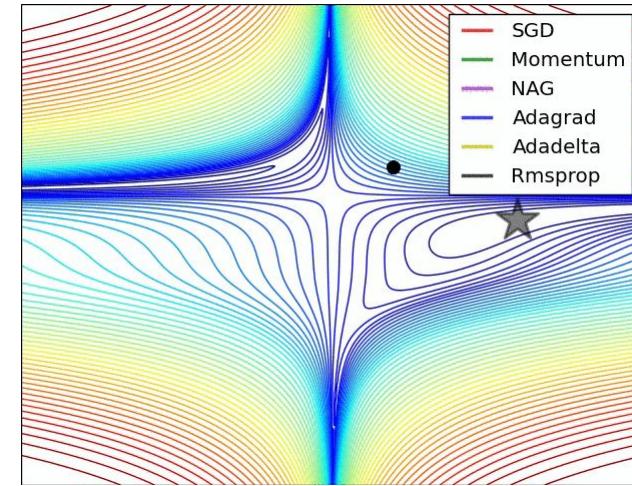


Imagen tomada de Asha Ponraj, 2020.

# Variantes del descenso del gradiente

## Momento

- Es una extensión del algoritmo del descenso de gradiente, a menudo denominado descenso de gradiente con momento.
- Está diseñado para **acelerar** el proceso de optimización, es decir, busca **disminuir** el número de evaluaciones de funciones necesarias para alcanzar el óptimo.
- Se basa en la metáfora del impulso de la física donde la aceleración en una dirección se puede acumular a partir de actualizaciones pasadas.



El algoritmo de impulso acumula una media móvil que decae exponencialmente de gradientes pasados y continúa moviéndose en su dirección.

Imagen tomada de Asha Ponraj, 2020.

# Variantes del descenso del gradiente

## RMSProp

- RMSprop significa Root Mean Square Propagation y fue propuesto por Geoffrey Hinton.
- Actualiza los parámetros a partir de los promedios móviles ponderados de los gradientes al cuadrado (2do momento de los gradientes)



# Variantes del descenso del gradiente

## Algoritmo RMSProp

1. Inicializa los parámetros  $\theta$  y la variable  $v$  a cero.
2. Por cada iteración, calcula los gradientes  $g$  con respecto a los parámetros  $\theta$  utilizando el minilote de datos actual.
3. Actualiza  $v$  aplicando una caída exponencial:  
 $v = \beta v + (1 - \beta)g^2$ , donde  $\beta$  es un hiperparámetro que controla la tasa de caída y  $g^2$  es el cuadrado de los gradientes por elementos.
4. Calcula la raíz cuadrada de los elementos de  $v$  y agrega una pequeña constante  $\epsilon$  para evitar la división por cero:  $v = \sqrt{v} + \epsilon$ .
5. Actualiza los parámetros  $\theta$  aplicando una tasa de aprendizaje escalada:  $\theta = \theta - \alpha \frac{g}{\sqrt{v}}$ , donde  $\alpha$  es la tasa de aprendizaje inicial y  $\frac{g}{\sqrt{v}}$  es la división por elementos de los gradientes y la variable  $v$ .
6. Se repiten los pasos del 2 al 5 hasta alcanzar la convergencia o un número máximo de iteraciones.



# Variantes del descenso del gradiente

## Adam

- Adam, o **Adaptive Moment Estimation**, es un algoritmo de optimización que combina las ventajas de los algoritmos RMSprop y Momentum para mejorar el proceso de aprendizaje de un modelo.
- Al igual que Momentum, Adam utiliza una estimación del **momento y de la magnitud** de los gradientes anteriores para actualizar los parámetros del modelo en cada iteración.
- Sin embargo, en lugar de utilizar una tasa de aprendizaje constante para todos los parámetros, Adam **adapta la tasa de aprendizaje de cada parámetro** individualmente en función de su estimación del momento y de la magnitud del gradiente.



# Variantes del descenso del gradiente

## Adam

- Se estima el primer (la media) y segundo (la varianza no centrada) momentos de los gradientes

$$\mathbf{m}^{[t+1]} = \beta_1 \cdot \mathbf{m}^{[t]} + (1 - \beta_1) \cdot \nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})$$

$$\mathbf{v}^{[t+1]} = \beta_2 \cdot \mathbf{v}^{[t]} + (1 - \beta_2) \cdot [\nabla \mathcal{L}(\boldsymbol{\theta}^{[t]})]^2$$

- Debido a que estas estimaciones están sesgadas hacia 0, se realiza una corrección

$$\hat{\mathbf{m}}^{[t+1]} = \frac{\mathbf{m}^{[t+1]}}{1 - \beta_1^{t+1}}$$

$$\hat{\mathbf{v}}^{[t+1]} = \frac{\mathbf{v}^{[t+1]}}{1 - \beta_2^{t+1}}$$

donde  $\beta_1^{t+1}$  y  $\beta_2^{t+1}$  son los factores de ponderación  $\beta_1, \beta_2 \in [0, 1)$  elevados a la potencia  $t + 1$

- Para actualizar los parámetros se usan las estimaciones de los momentos de los gradientes en el tiempo  $t$

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}^{[t+1]}} + \epsilon} \cdot \hat{\mathbf{m}}^{[t+1]}$$





# Time to Code

losslandscape



# Repaso

- Aprendimos las diferencias entre la propagación hacia adelante y la propagación hacia atrás.
- Analizamos a detalle el algoritmo del cálculo de los gradientes en las redes neuronales.
- Estudiamos estrategias para evitar problemas de la explosión y desvanecimiento del gradiente
- Analizamos variantes del algoritmo del descenso del gradiente



# Referencias

- Zhang A, Lipton Z, Li M, and Smola J. Dive into Deep Learning. 2020. Disponible en <https://d2l.ai/>
- Murphy, K. P. (2022). Probabilistic Machine Learning: An introduction. MIT Press. Capítulo 8, 10 y 11. Disponible en <https://probml.github.io/pml-book/book1.html>
- Nielsen, M. (2019). Neural Networks and Deep Learning. Capítulo 1. Disponible en <http://neuralnetworksanddeeplearning.com/index.html>
- Rafael C. Gonzalez, Richard Eugene Woods (2018). Digital Image Processing. Capítulo 12. Disponible en <https://dl.icdst.org/pdfs/files4/01c56e081202b62bd7d3b4f8545775fb.pdf>



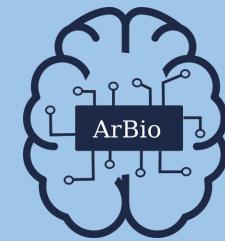
# Contacto

## Dra. Blanca Vázquez

Investigadora Postdoctoral  
Unidad Académica del IIMAS  
en el estado de Yucatán, UNAM.

**Correo:** [blanca.vazquez@iimas.unam.mx](mailto:blanca.vazquez@iimas.unam.mx)

**Github:** <https://github.com/blancavazquez>



Artificial Intelligence in  
Biomedicine Group (ArBio)

<https://iimas.unam.mx/arbio>

