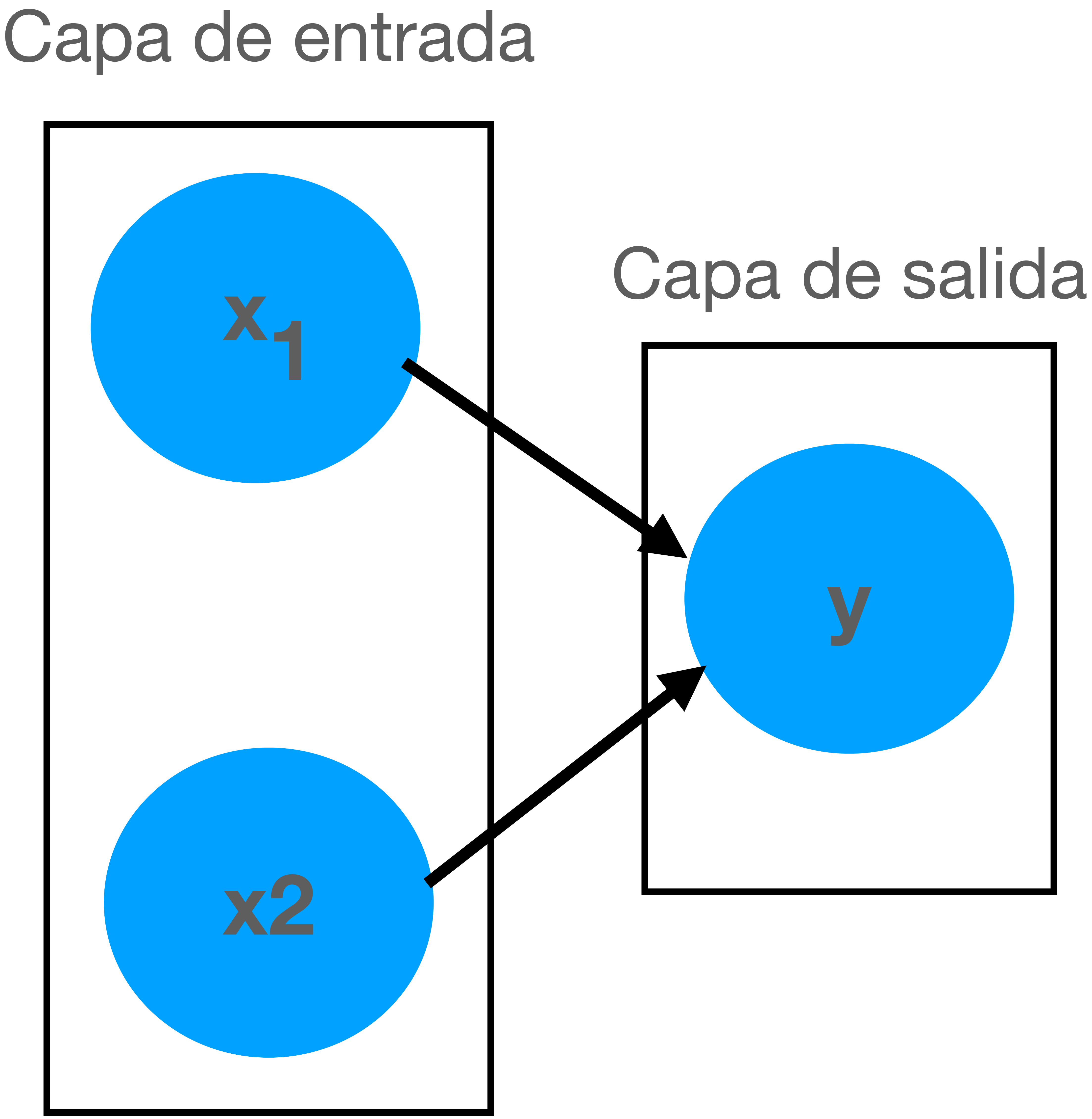
Profundizando en las Redes Neuronales

Luis R. Furlán Agosto de 2021

Capas

Exemplo minimalista

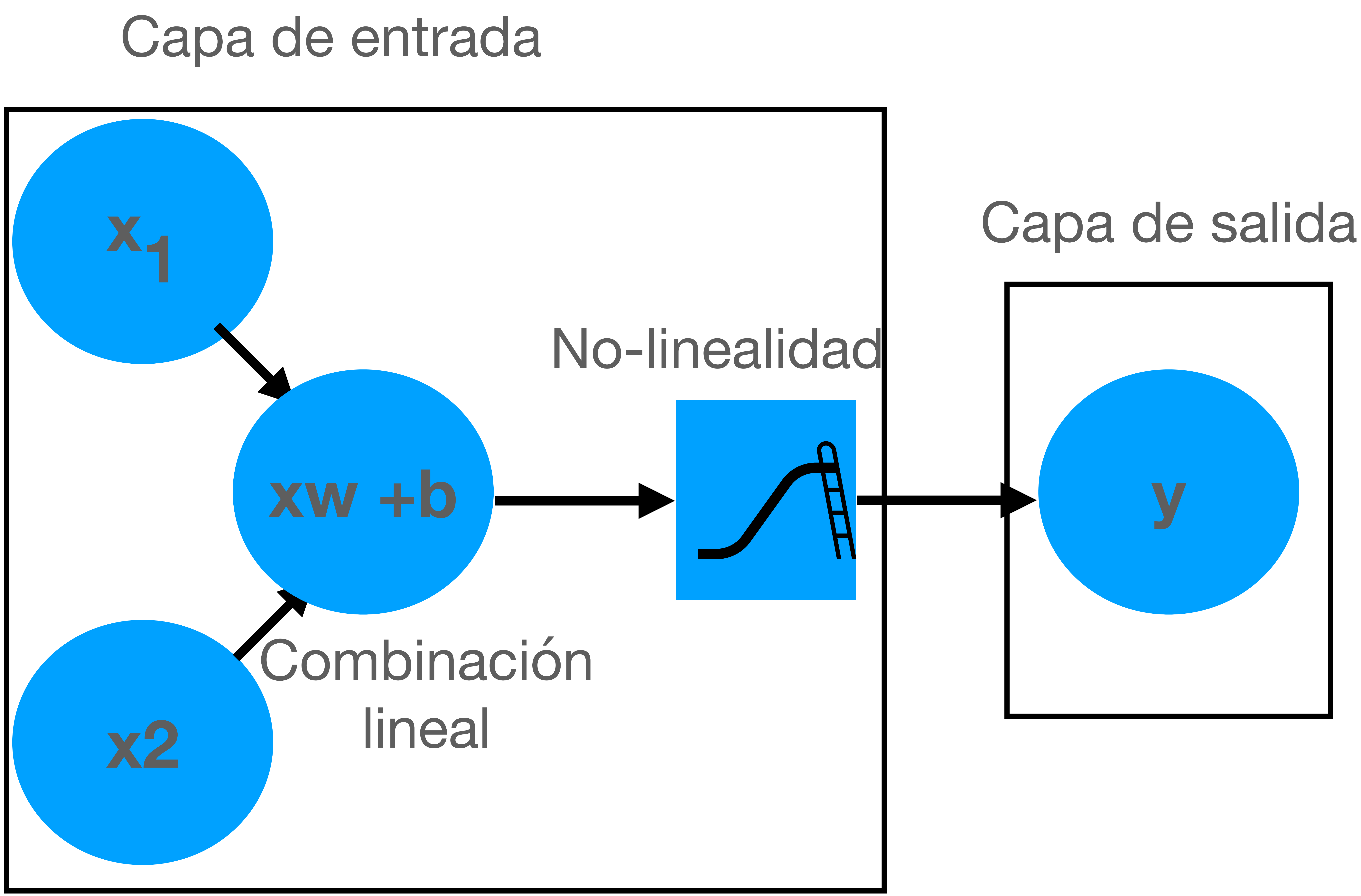
- Hasta ahora hemos trabajado con un ejemplo muy simple
- Una Red Neuronal sin profundidad
- La salida es simplemente una combinación lineal de las entradas



Capas

Red Neuronal

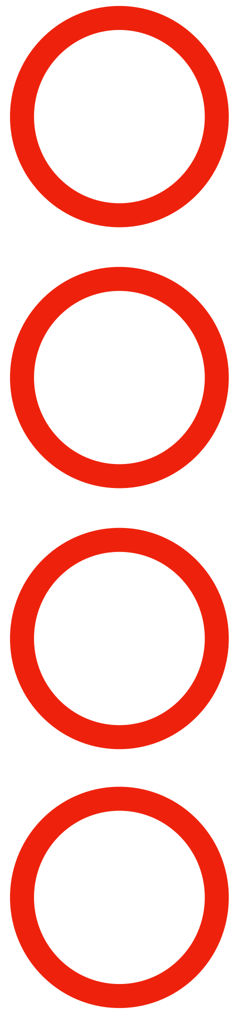
- Se continua con una combinación lineal
- Se agrega una no-linealidad
- Esto permite modelar funciones arbitrarias



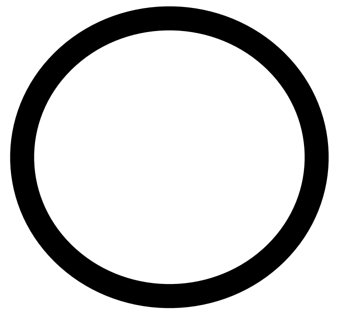
Una Red Neuronal profunda

Esta tiene 5 capas

Cómo leer este diagrama

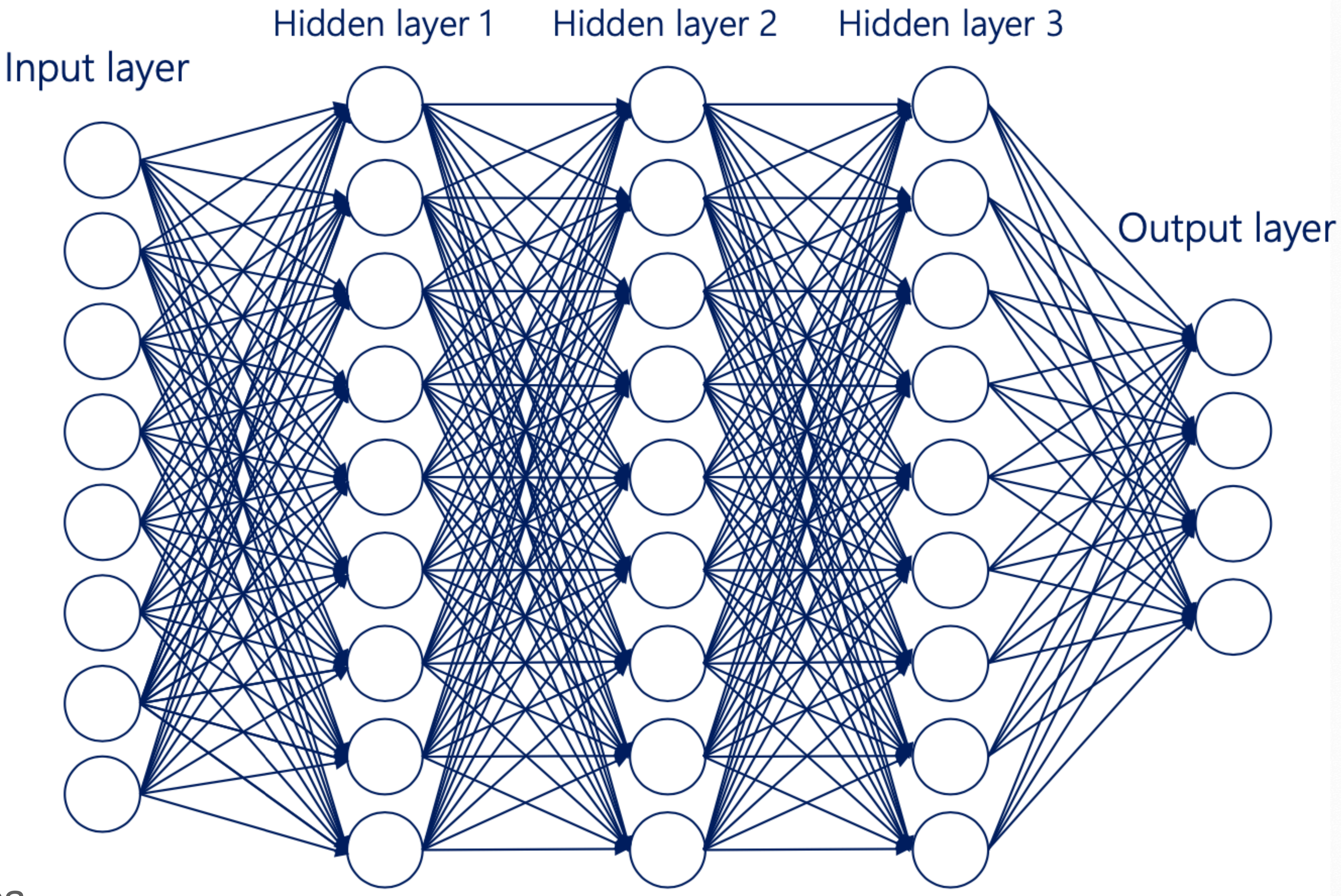


Una capa



Una unidad - perceptron

Flechas representan una transformación matemática



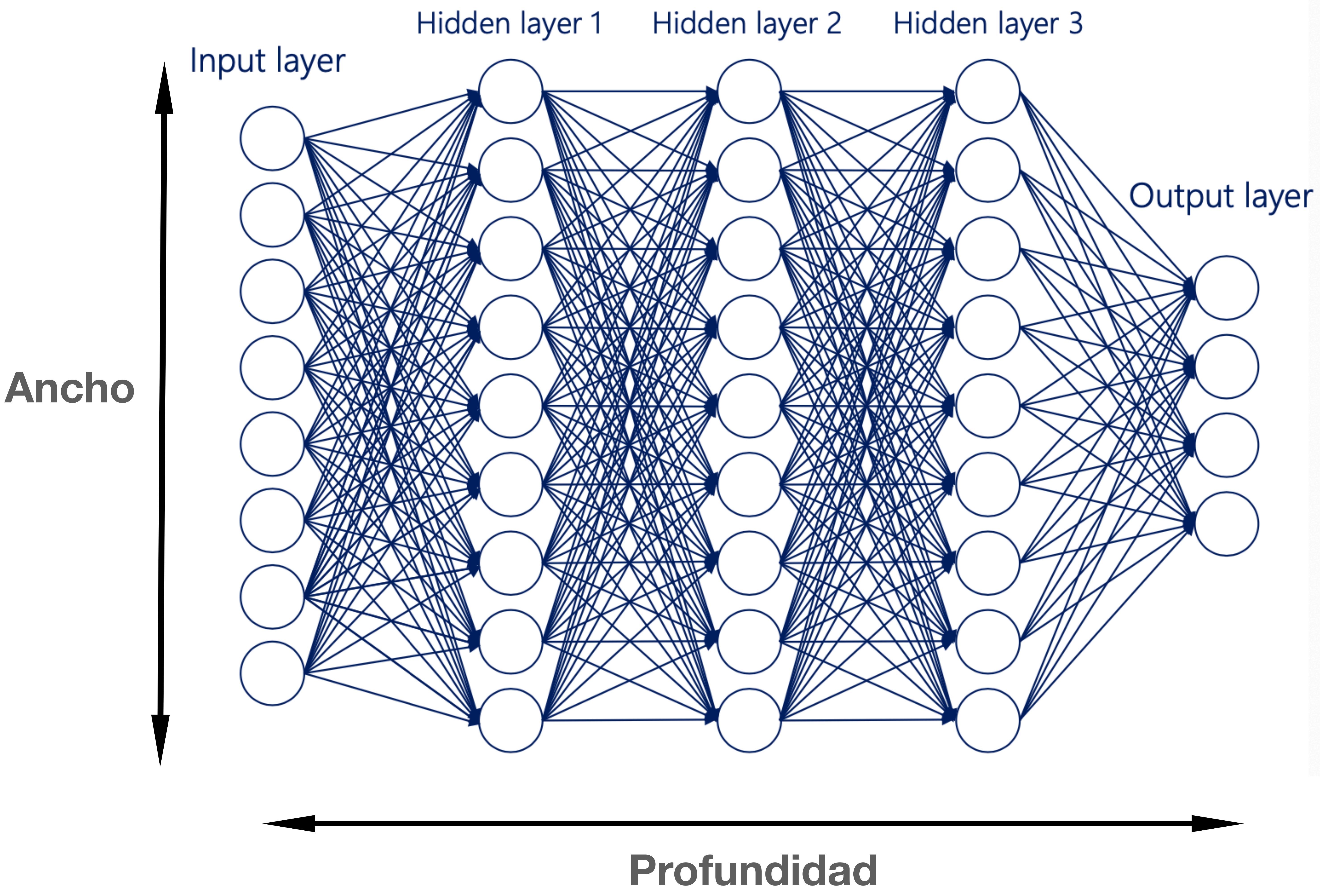
Una Red Neuronal profunda

El **ancho** de una capa es el número de unidades en esa capa.

El **ancho** de la red es el número de unidades de la capa más grande

La **profundidad** de la red generalmente se toma como el número de capas escondidas

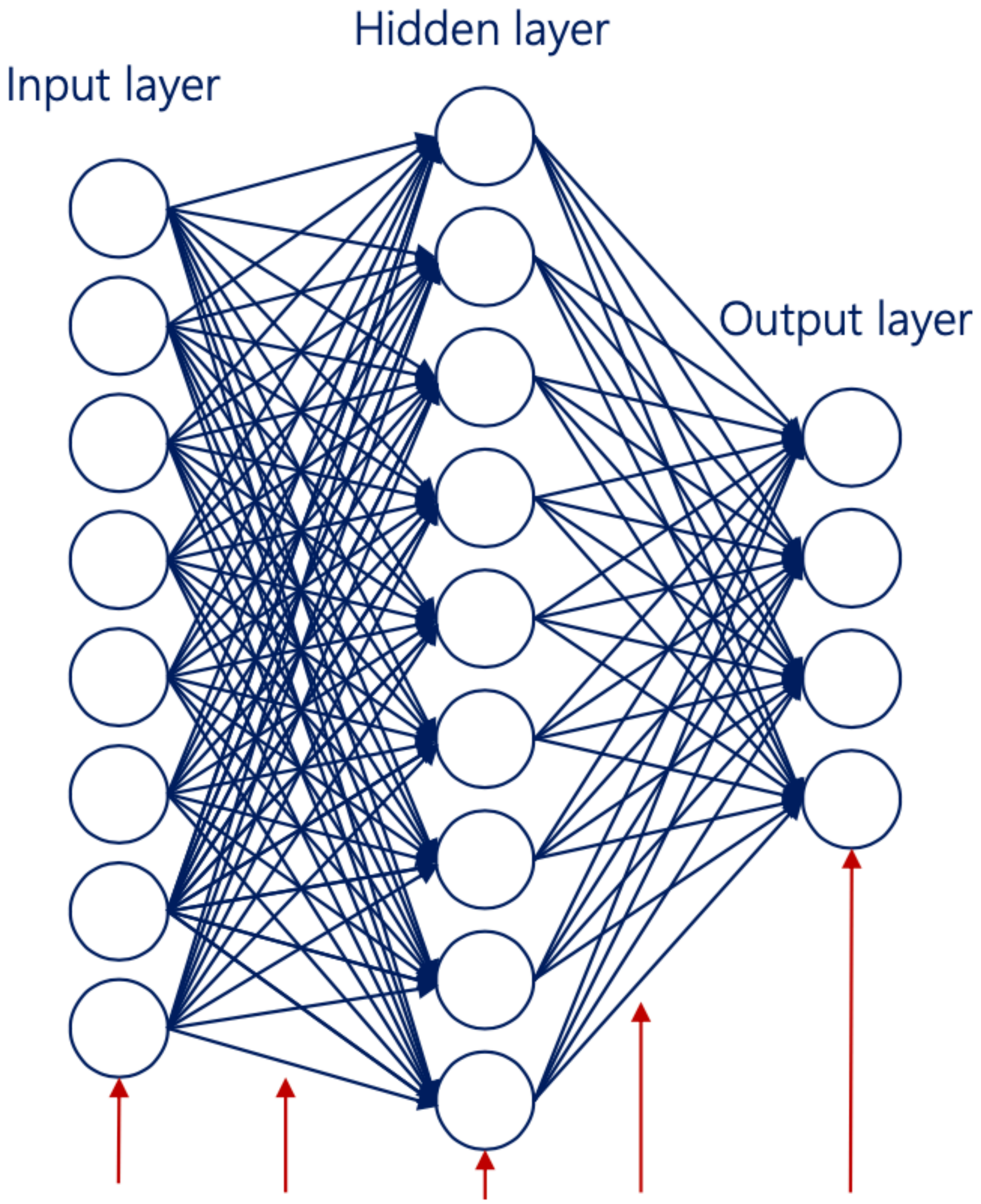
El **ancho** y la **profundidad** de la red se denominan **hiperparámetros.** Se seleccionan manualmente



¿Porqué se necesita la no-linealidad?

Una red sin no-linealidades: son solo combinaciones lineales

Dos transformaciones lineales consecutivas, son equivalentes a una sola



$$\underset{1x8}{x} * \underset{8x9}{w_1} = \underset{1x9}{h} * \underset{9x4}{w_2} = \underset{1x4}{y}$$

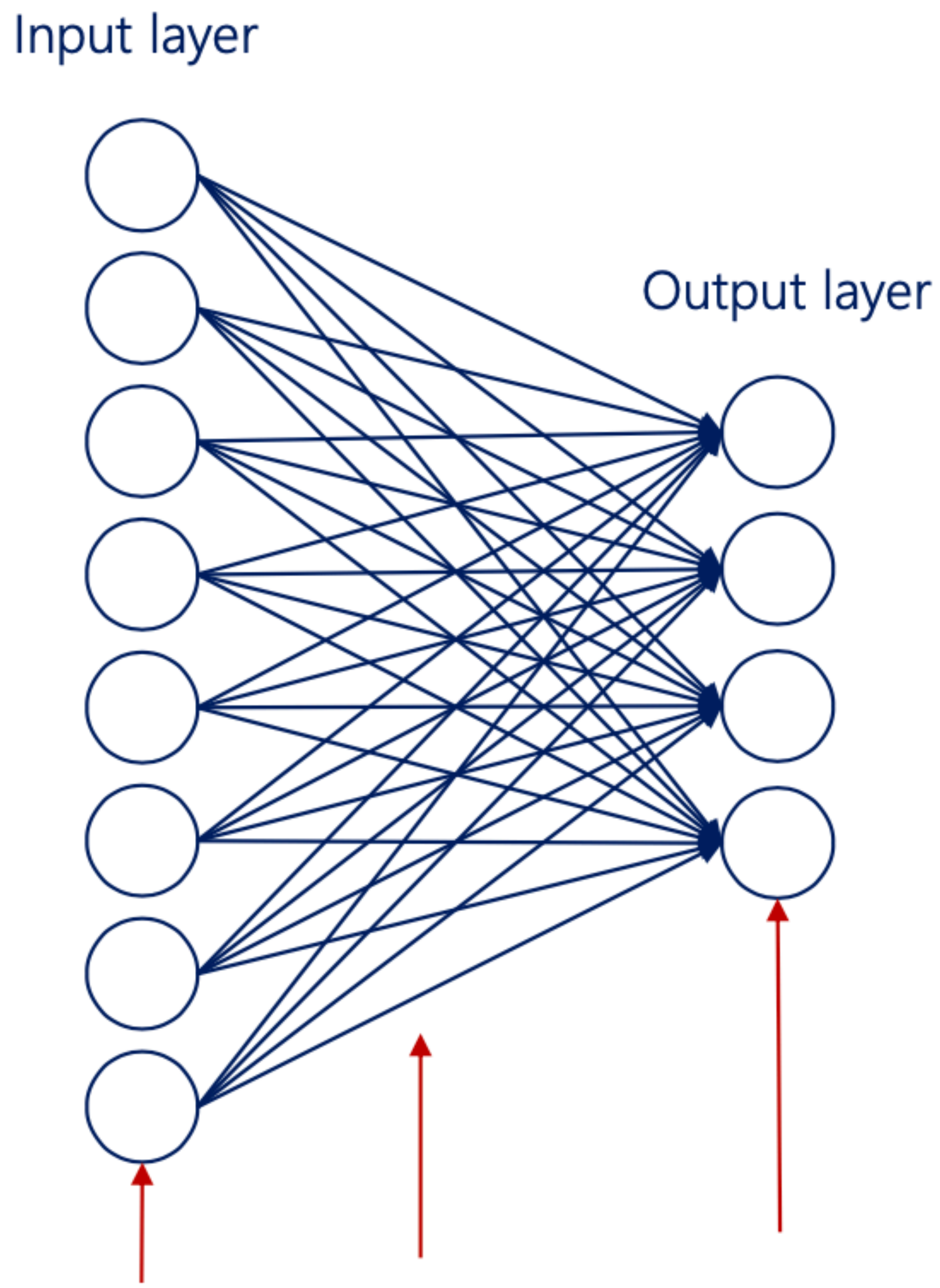
$$h = x * w_1$$

$$y = h * w_2$$

$$y = x * \boxed{\underset{8x9}{w_1} * \underset{9x4}{w_2}}$$

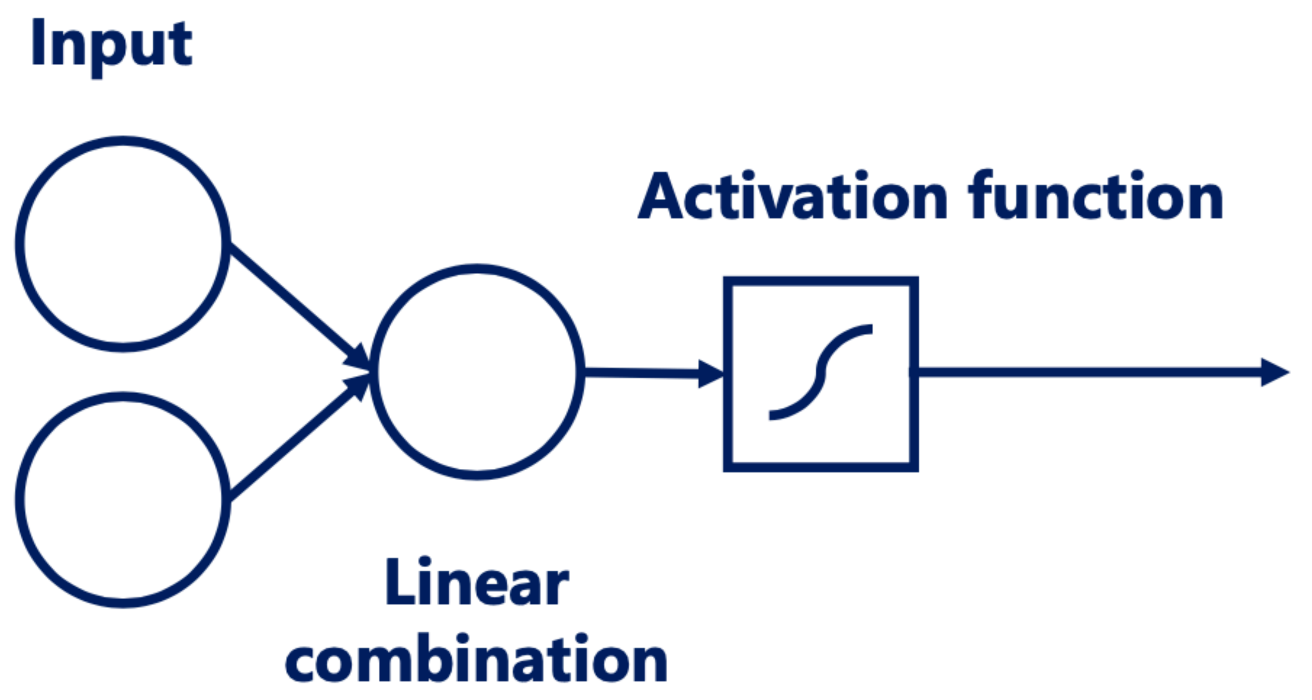
$$y = x * \underset{8x4}{w^\star}$$

Dos transformaciones lineales consecutivas, son equivalentes a una sola

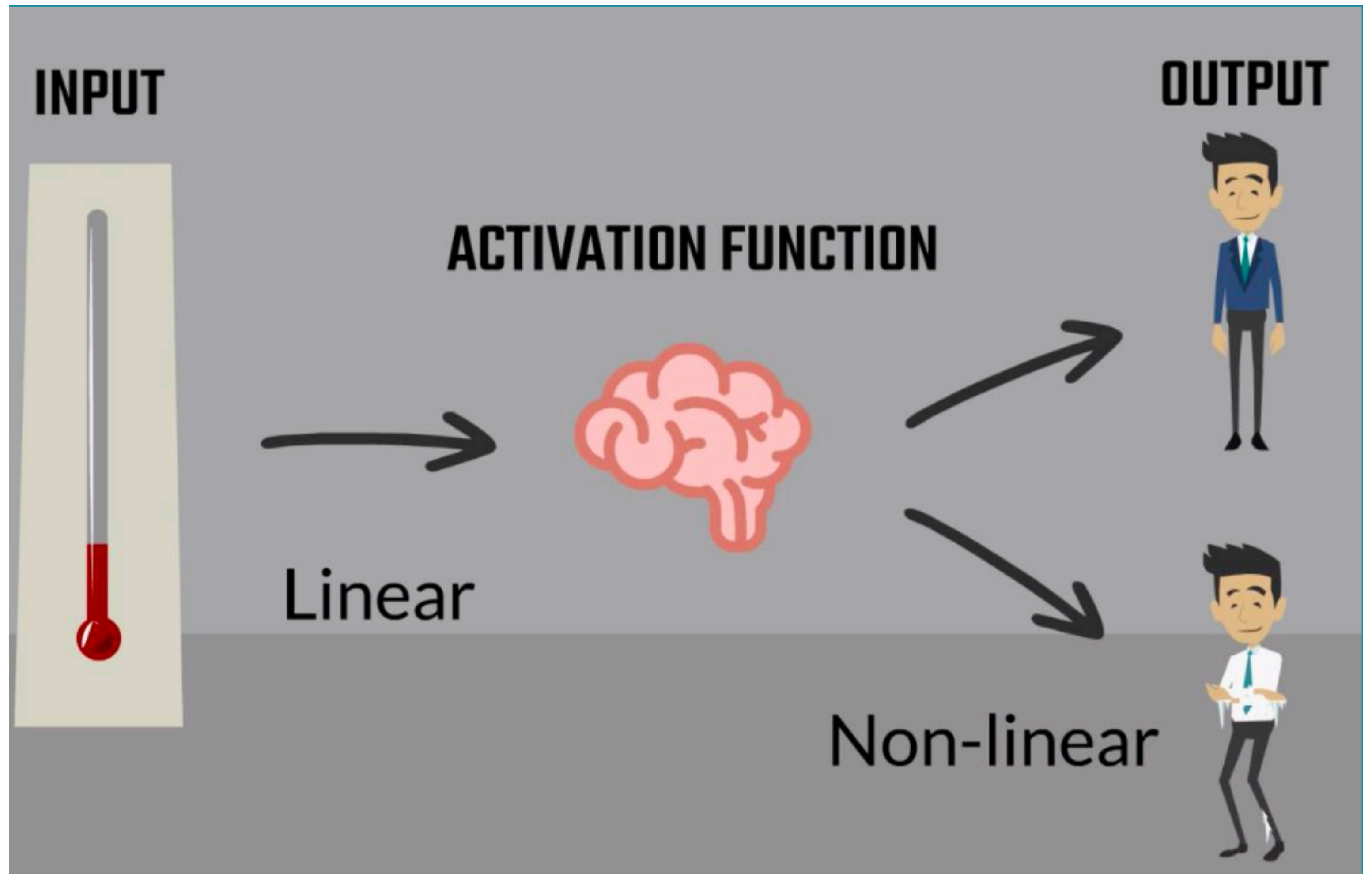


$$\underset{1x8}{x} * \underset{8x4}{w^\star} = \underset{1x4}{y}$$

Funciones de activación



Nuestro cerebro es como una función de activación.
Supongamos que T va bajando (un cambio numérico)



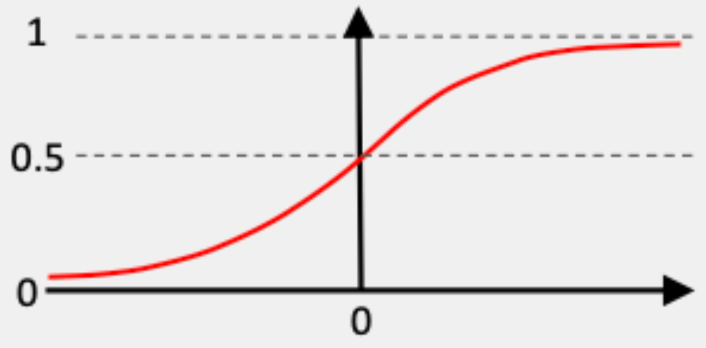
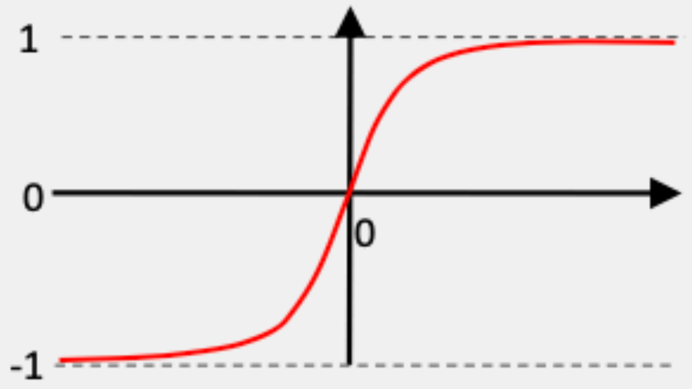
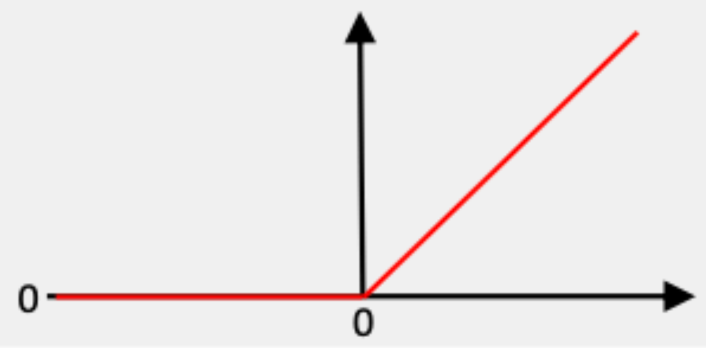
Ponerse la chumpa es una acción binaria :
0 (No me la pongo)   1 (Sí me la pongo)

Se necesitan las funciones de activación (no-linealidades) para romper con la linealidad y representar relaciones más complicadas

Es más, se necesitan las funciones de activación para **apilar capas (stack layers).**

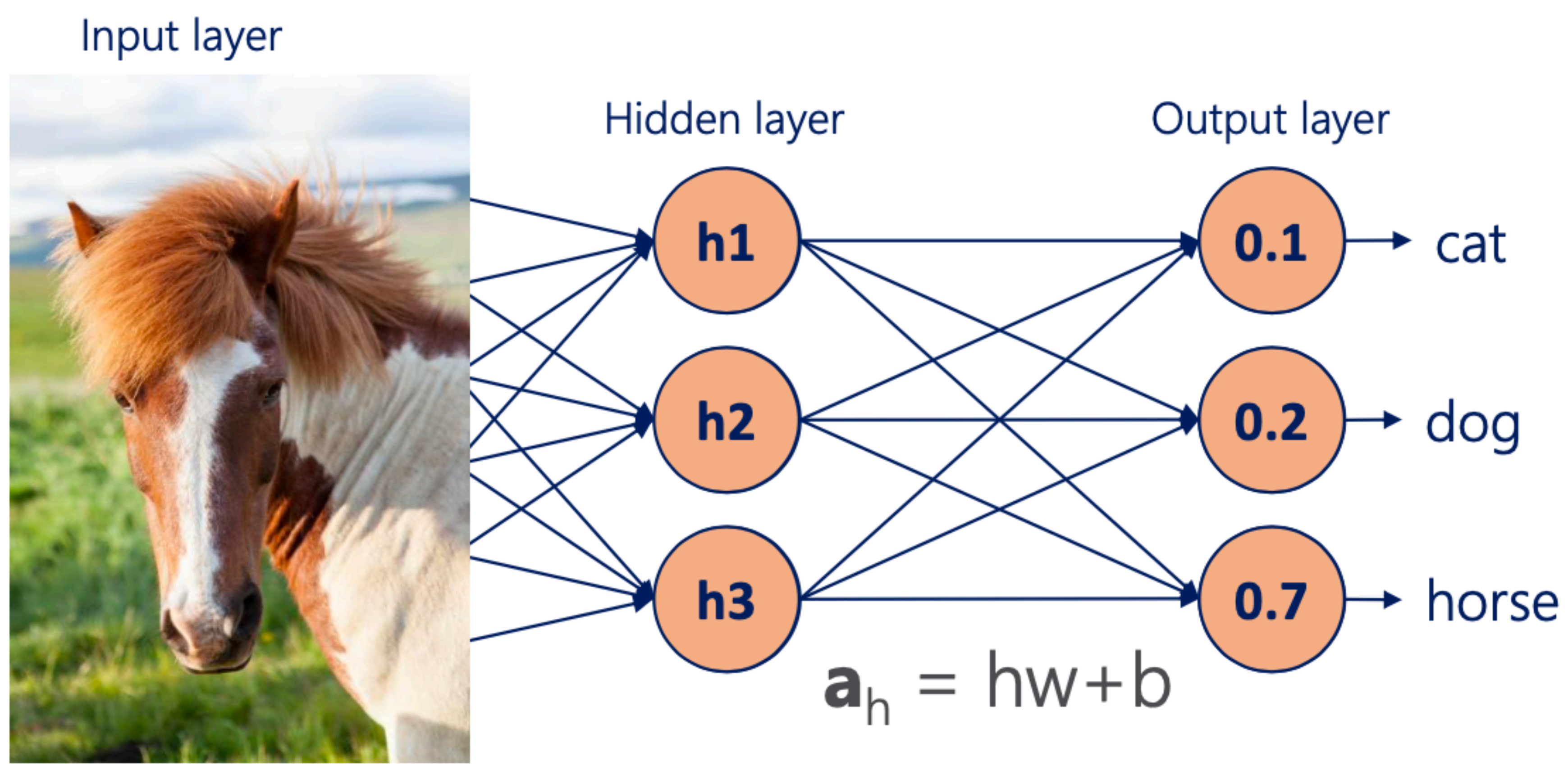
Las funciones de activación transforman las entradas a salidas de diferente tipo.

Funciones comunes de activación

Nombre	Fórmula	Derivada	Gráfica	Rango
sigmoid (logistic function)	$\sigma(a) = \frac{1}{1+e^{-a}}$	$\frac{\partial\sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a))$		(0,1)
TanH (hyperbolic tangent)	$\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$	$\frac{\partial\tanh(a)}{\partial a} = \frac{4}{(e^{a}+e^{-a})^2}$		(-1,1)
ReLu (rectified linear unit)	relu(a) = max(0,a)	$\frac{\partial\,\mathrm{relu}(a)}{\partial a} = \begin{cases} 0, if\ a \le 0 \\ 1, if\ a > 0 \end{cases}$		(0,∞)
softmax	$\sigma_{\mathrm{i}}(\boldsymbol{a}) = \frac{e^{a_i}}{\sum_j e^{a_j}}$	$\frac{\partial\sigma_{\mathrm{i}}(\boldsymbol{a})}{\partial a_j} = \sigma_i(\boldsymbol{a})\left(\delta_{ij} - \sigma_j(\boldsymbol{a})\right)$ Where δ_{ij} is 1 if i=j, 0 otherwise		(0,1)

Todas estas funciones son **monotónicas, continuas, y diferenciables**. Estas son propiedades importantes para la optimización.

Activación Softmax



Ejemplo:

a = [-0.21,0.47,1.72]

softmax (**a**) = $\frac{e^{a_i}}{\sum_j e^{a_j}}$

$\sum_j e^{a_j} = e^{-0.21} + e^{0.47} + e^{1.72} = 8$

softmax (**a**) = $[\frac{e^{-0.21}}{8}, \frac{e^{0.47}}{8}, \frac{e^{1.72}}{8}]$

y = [0.1,0.2,0.7] → distribución de probabilidades

La activación Softmax transforma cualquier tamaño de números arbitrarios en una distribución de probabilidades.

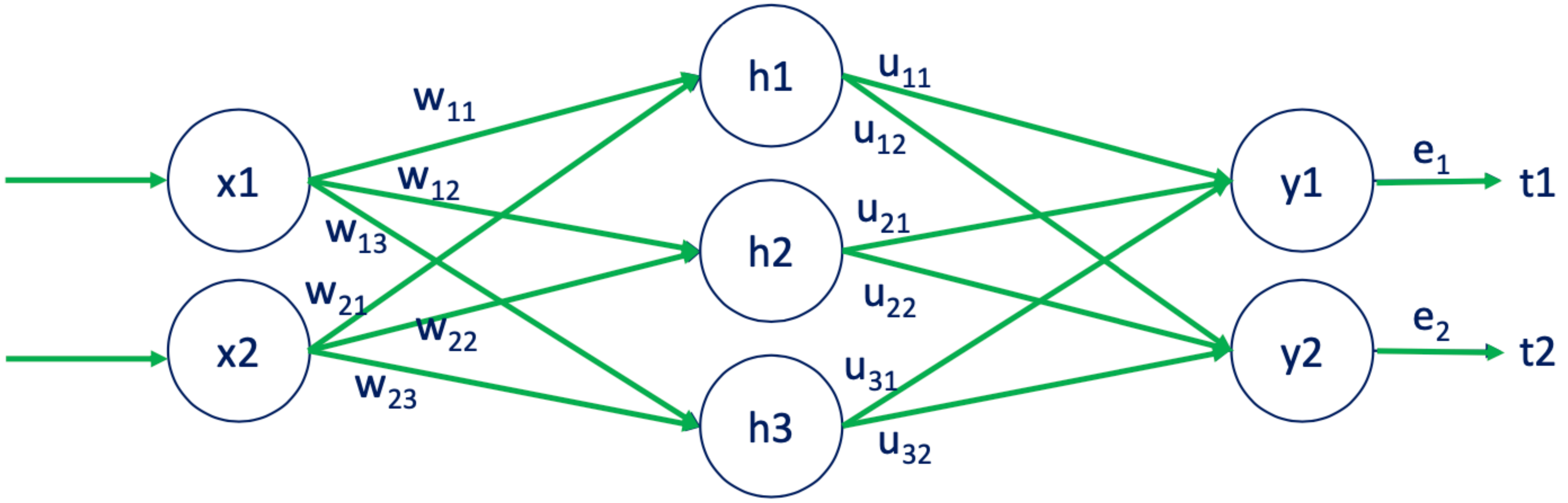
Mientras otras funciones de activación reciben un valor de entrada y lo transforman, sin consideración de los otros elementos, Softmax considera la información de la **totalidad del conjunto de números que tenemos**.

Los valores de salida de Softmax están en el rango de 0 a 1 y la suma es exactamente 1 (como probabilidades).

La propiedad de Softmax, de producir probabilidades, es tan útil e intuitivo que a menudo se utiliza como la función de activación de la **capa final (salida).**

Sin embargo, el Softmax no es satisfactorio cuando se utiliza con capas escondidas, debido a la pérdida de mucha de la información sobre la variabilidad de los datos.

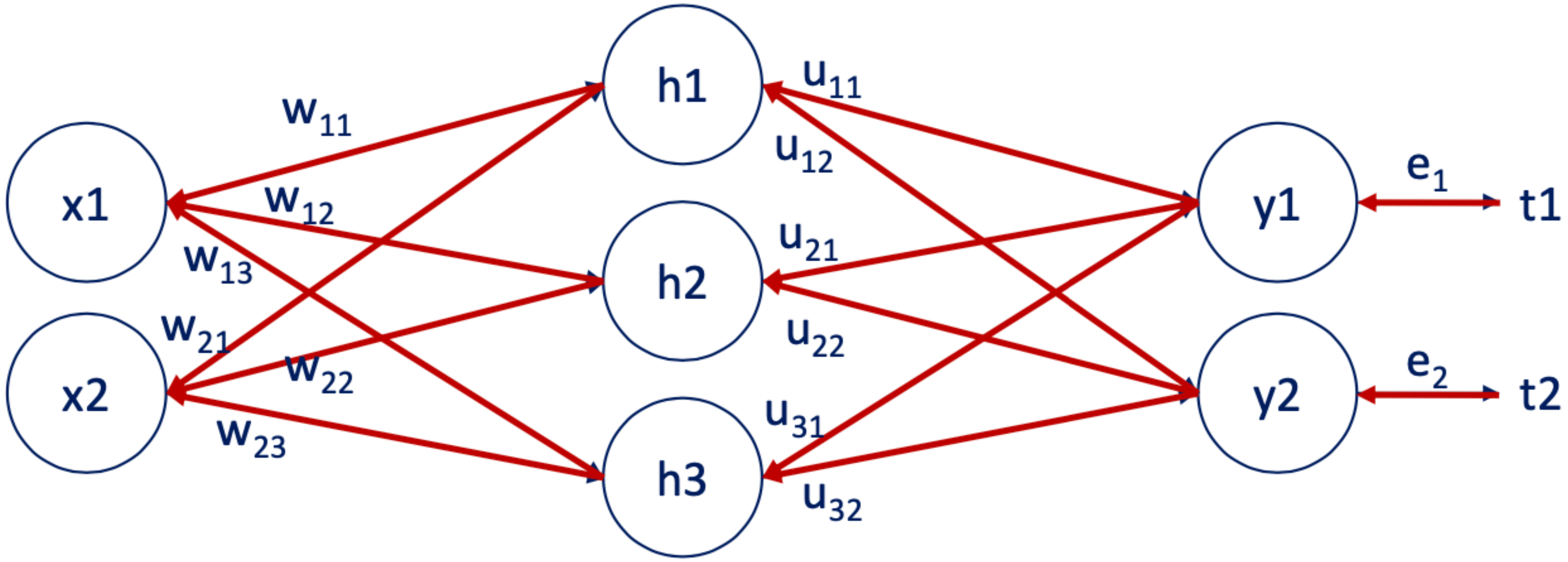
Propagacion hacia adelante



La **propagación hacia adelante** es el proceso de pasar las entradas por la red.

Al final de cada **época,** se comparan las salidas y se comparan con las metas, para obtener los errores

Propagacion hacia atras

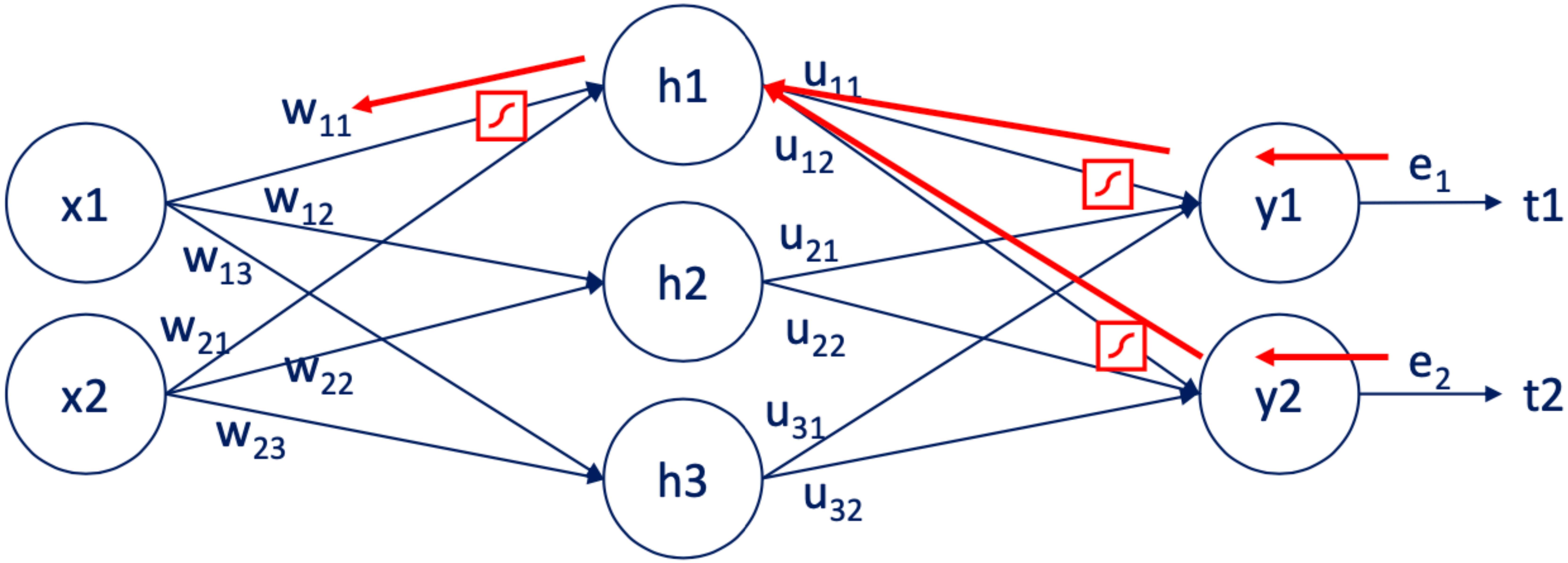


La **propagación hacia atras** es un algoritmo para redes neuronales utilizando el descenso por gradientes.

Se calcula la contribución de cada parámetro a los errores.

Los errores se propagan hacia atrás por la red y se actualizan los parámetros (pesos y sesgos) correspondientemente.

Fórmula de la propagacion hacia atras



$$\frac{\partial L}{\partial w_{ij}} = \delta_j x_i \text{ , donde } \delta_j = \sum_k \delta_k w_{jk}\, y_j(1 - y_j)$$

Si desean explorar la derivación completa, se ha incluido un PDF, en el CANVAS, que usa la misma nomenclatura que hemos usado en clase: **Backpropagation. A peek into the Mathematics of Optimization.**