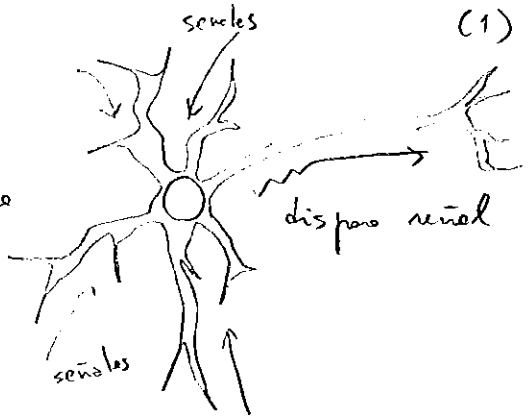
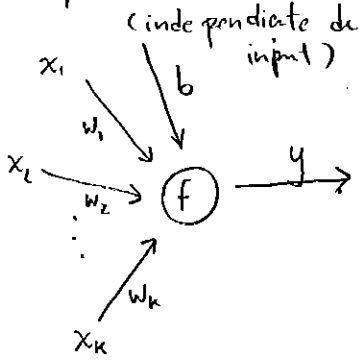
1) Intro a redes neuronales

Perceptrón (~50) motivado por neurona





$$u = x_1 w_1 + x_2 w_2 + \cdots + x_k w_k + b$$

$y = f(u)$ ← f se llama función de activación

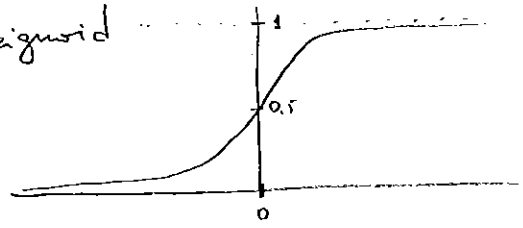
$w_1, w_2, \ldots, w_k$: parámetros

b : sesgo (bias)

$x_1, x_2, \ldots, x_k$ valores de entrada

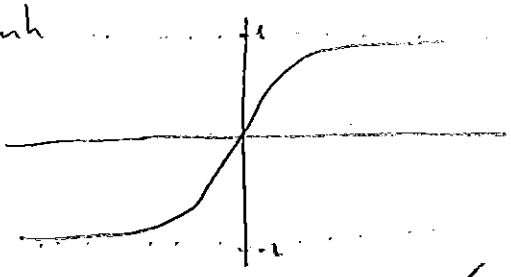
Función de activación

sigmoid



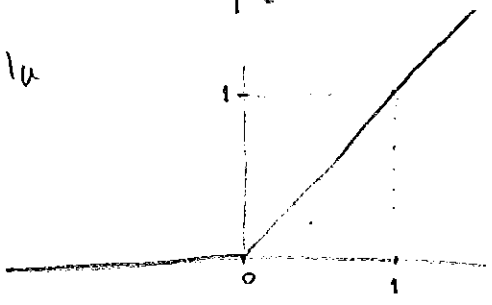
$$sig(u) = \frac{1}{1 + e^{-u}}$$

tanh



$$\tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$$

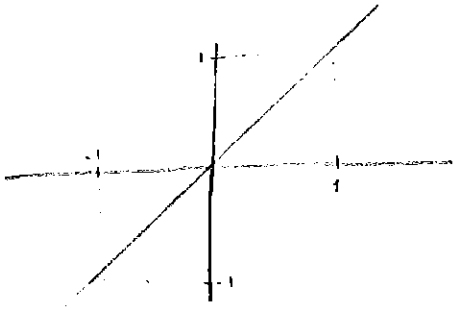
relu



$$relu(u) = \begin{cases} u & \text{si } u > 0 \\ 0 & \text{si } u \leq 0 \end{cases}$$

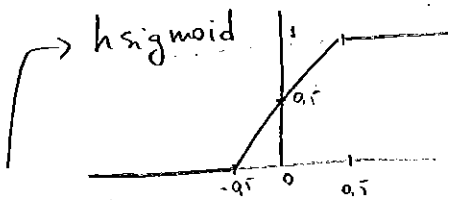
Lo más importante / útil !!!

lin



$$lin(u) = u$$

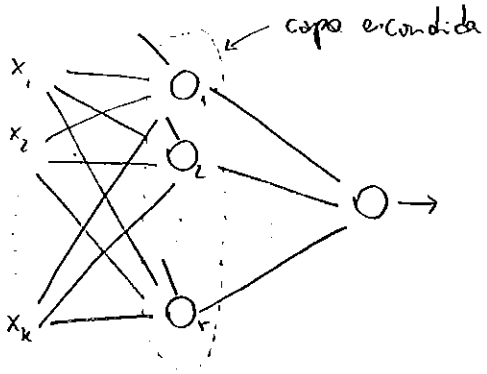
hsigmoid

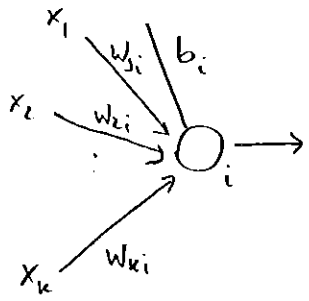


Hay otras funciones de activación

(Tendrán que ver algunas en la tarea)

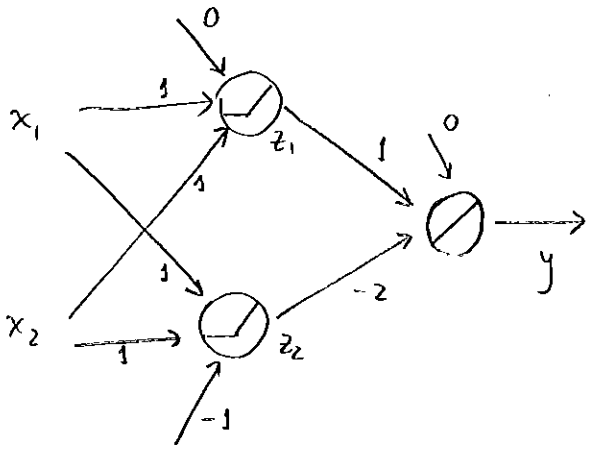
varios perceptrones:





Ejemplo: XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



$h_1 = \text{relu}(x_1 \cdot 1 + x_2 \cdot 1 + 0)$ $\quad y = h_1 - 2h_2$

$h_2 = \text{relu}(x_2 \cdot 1 + x_2 \cdot 1 - 1)$

$x_1 = 0$, $x_2 = 0$ $\Rightarrow$ $h_1 = \text{relu}(0+0) = 0$, $h_2 = \text{relu}(0+0-1) = \text{relu}(-1) = 0$ $\quad y = 0 - 2\cdot 0 = 0$ ✓

$x_1 = 0$, $x_2 = 1$ $\Rightarrow$ $h_1 = \text{relu}(0+1) = 1$, $h_2 = \text{relu}(0+1\cdot 1) = 0$ $\quad y = 1 - 2\cdot 0 = 1$ ✓

$x_1 = 1$, $x_2 = 0$ $\Rightarrow$ $h_1 = \text{relu}(1+0) = 1$, $h_2 = \text{relu}(1+0-1) = 0$ $\quad y = 1 - 2\cdot 0 = 1$ ✓

$x_1 = 1$, $x_2 = 1$ $\Rightarrow$ $h_1 = \text{relu}(1+1) = \text{relu}(2) = 2$, $h_2 = \text{relu}(1+1-1) = \text{relu}(1) = 1$ $\quad y = 2 - 2\cdot 1 = 0$ ✓

↑ La red de perceptrones computa de una forma distinta a como sistemas acostumbrados en computación, no "manipula símbolos", si no opera valores

puede escribirse así

$$(h_1\ h_2) = \text{relu}\left((x_1\ x_2)\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix} + (0\ -1)\right)$$

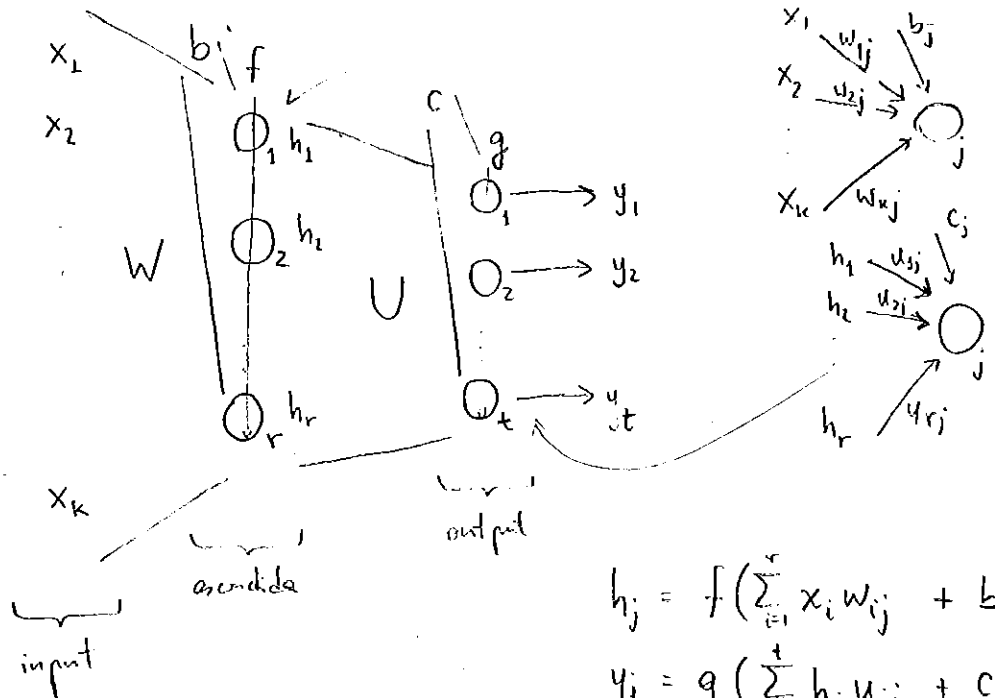
$$(y) = (h_1\ h_2)\begin{pmatrix}1\\ -2\end{pmatrix}$$

$$x = (x_1, x_2) \quad W = \begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix} \quad b = (0\ -1)$$

$$U = \begin{pmatrix}1\\ -2\end{pmatrix}$$

$$u = xW + b$$
$$h = \text{relu}(u)$$
$$y = hU$$

Esto fácil lo podemos generalizar

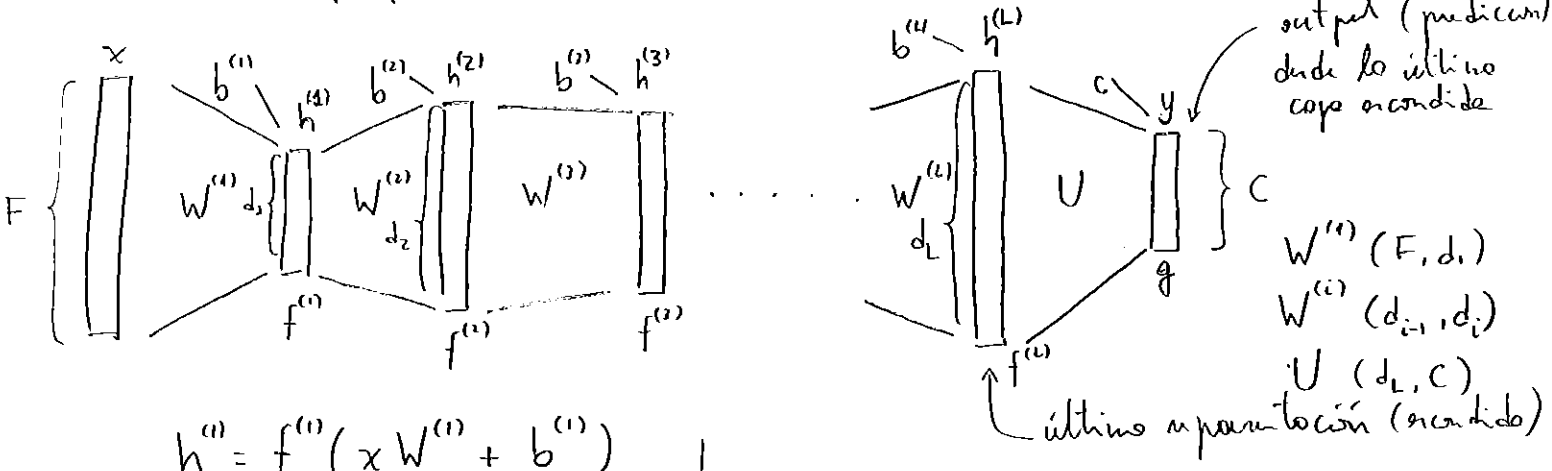


$$h_j = f\left(\sum_{i=1}^{r} x_i w_{ij} + b_j\right)$$
$$y_i = g\left(\sum_{i=1}^{t} h_i u_{ij} + c_j\right)$$

$$h = f(xW + b) \qquad W_{(k,r)} \quad b_{(r)}$$
$$y = g(hU + c) \qquad U_{(r,t)} \quad c_{(t)}$$

[Interludio : Universal approximation Theorem] ⟶ (3b)

Red neuronal Feed Forward o Perceptrón Multicapa, o simplemente Red neuronal profunda:



$$h^{(1)} = f^{(1)}(x W^{(1)} + b^{(1)})$$

$$h^{(i)} = f^{(i)}(h^{(i-1)} W^{(i)} + b^{(i)})$$

$$y = g(h^{(L)} U + c)$$

pasada hacia adelante

$y = \text{forward}(x)$ $\quad (\hat{y} = \text{forward}(x))$

[Nota: en general usaremos la red para computar una predicción y compararla con el valor real que ya conocemos. En este caso generalmente denotaremos por $\hat{y} = \text{forward}(x)$ para diferenciar "$\hat{y}$" de "y" que usamos para denotar el valor "real".]

¿Qué usaremos como función de salida? dependerá mucho de la tarea (y función de pérdida que veremos más adelante)

En general queremos probabilidades

1) clasificación binaria:
- una única neurona
- valor de salida entre 0 y 1
- Interpretación:
  prob $\geq 0.5 \Rightarrow$ respuesta 1
  prob $< 0.5 \Rightarrow$ respuesta 0

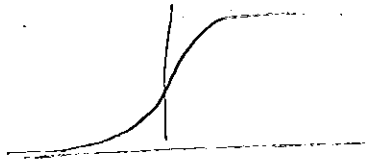
$\Longrightarrow$ podemos usar sigmoid como salida

2) clasificación en varias clases

$\hat{y} = \text{forward}(x)$

$P(\text{clase de } x \text{ es } 1) = \hat{y}$

$P(\text{clase de } x \text{ es } 0) = 1 - \hat{y}$



(continúa)

2) clasificación en varias clases:

- si tenemos C clases usamos C neuronas en la capa de salida
- necesitamos una función que tome valores

$$(u_1\ u_2\ u_3\ \dots\ u_C)$$

y nos permita interpretarlos como probabilidades

- lo más usado (más adelante veremos por qué) es "softmax"

$$\text{softmax}(u_1\ u_2\ u_3\ \dots\ u_C) = (s_1\ s_2\ \dots\ s_C)$$

$$s_i = \frac{e^{u_i}}{\sum_{j=1}^{C} e^{u_j}} \qquad \text{tenemos que} \quad \sum s_i = 1, \quad 0 \le s_i \le 1$$

softmax amplifica las diferencias en los valores de input

$\text{softmax}(0\ 2) = (0.12\ \ 0.88)$ $\qquad$ $\text{softmax}(0\ 10) = (0.00005\ \ 0.99995)$

$\text{softmax}(0\ 0\ 10) = (0.00005\ \ 0.00005\ \ 0.9999)$

Interpretación: $\hat{y} = \text{softmax}(u_1\ u_2\ \dots\ u_C) = (\hat{y}_1\ \hat{y}_2\ \dots\ \hat{y}_C)$

$P(x \text{ pertenece a la clase } i) = \hat{y}_i$ según la red neuronal

En ambos casos las vemos como modelos probabilísticos (parametrizados)

Entrenamiento: Supongamos ejemplos $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(N)}, y^{(N)})\}$

¿Qué tan bien (o mal) lo está haciendo la red?

lo medimos individualmente

$$\text{error}(\hat{y}, y^{(i)})$$

$$\hat{y}^{(i)} = \text{forward}(x^{(i)}) \qquad y^{(i)} \Longrightarrow \qquad \text{error}(x^{(i)}, y^{(i)})$$

↑ lo que la red dice — ↑ lo que esperamos — ↑ cuantifico "qué tan lejos" están $y^{(i)}$ e $\hat{y}^{(i)}$

$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \text{error}(\hat{y}^{(i)}, y^{(i)})$ ← promedio de los errores que cometimos sobre todos los ejemplos.

¿de qué depende $\mathcal{L}$? depende de la red y los que definen a la red: $\underbrace{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \ldots, W^{(L)}, b^{(L)}, U, c}_{\theta}$ ← parámetros

$\mathcal{L}(\theta)$ $\mathcal{L}$ depende de θ

me encontré que $\mathcal{L}$ no tan pequeño como me pueda

$\Rightarrow$ Problema: encontrar $\hat{\theta}$ que minimiza $\mathcal{L}(\theta)$, listo! $\min_{\theta} \mathcal{L}(\theta)$

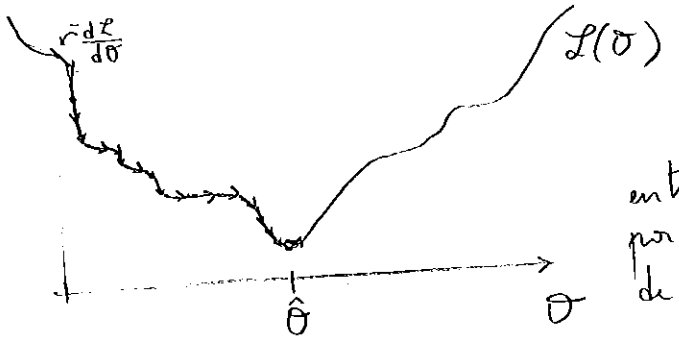
L capas, d neuronas en capa escondido (mas chico) $\Rightarrow O(Ld^2)$

$L = 10 \quad d = 10 \rightarrow 1000$
$L = 10 \quad d = 100 \rightarrow 100.000$
$L = 10 \quad d = 1000 \rightarrow 10.000.000$,

búsqueda exhaustiva no darán resultados razonables.

Idea: si muevo "un poquito" uno de los parámetros ¿como cambia la pérdida / error?
- si el error disminuye me conviene
- si el error aumenta no me conviene el cambio

Pero es muy caro hacer un cambio y evaluar $\Rightarrow$ lo hacemos analíticamente



$\frac{d\mathcal{L}}{d\theta}$ derivada

entrenamiento por descenso de gradiente:

repite e veces

$$\theta_{new} = \theta_{old} - \alpha \left.\frac{d\mathcal{L}}{d\theta}\right|_{\theta_{old}}$$

α ↑ tasa de aprendizaje

En general lo que tenemos que hacer en actualizar en un loop, como siguen:

inicializar $\hat{W}^{(i)}, \hat{b}^{(i)}, \hat{U}, \hat{c}$ de manera aleatoria, decida α

entrenamiento por descenso de gradiente:

repite e veces (o mientras alguna condición de detención no se alcance)

$$\hat{W}^{(i)} := \hat{W}^{(i)} - \alpha \left.\frac{\partial \mathcal{L}}{\partial W^{(i)}}\right|_{\hat{\theta}}$$

$$\hat{b}^{(i)} := \hat{b}^{(i)} - \alpha \left.\frac{\partial \mathcal{L}}{\partial b^{(i)}}\right|_{\hat{\theta}}$$

para i entre 1 y L

$$\hat{U} := \hat{U} - \alpha \left.\frac{\partial \mathcal{L}}{\partial U}\right|_{\hat{\theta}}$$

$$\hat{c} := \hat{c} - \alpha \left.\frac{\partial \mathcal{L}}{\partial c}\right|_{\hat{\theta}}$$

gradientes

Red $h = f(xW + b)$

$y = g(hU + c)$ asp. $g = \text{lin}$ (identidad)

$\Rightarrow y = f(xW + b)U + c \qquad x = (x_1, x_2) \in \{0,1\}^2$

¿Qué pasa si usamos f como identidad?

$\Rightarrow y = (xW + b)U + c = xWU + (bU + c)$

$\Rightarrow y = xW' + b'$

si f es la función identidad $\Rightarrow$ la red solo puede aprender una función lineal.

si $f(u) = u \cdot m + v$ (lineal) $\Rightarrow f(xW + b)$

$= (xW + b)m + v$

donde $M = \begin{pmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{pmatrix}$ $v = (v, v)$

$\Rightarrow y = ((xW + b)M + v)U + c$

$= xWMU + bMU + vU + c$

$y_{(1)} = x_{(2)} W'_{(2,1)} + b'_{(1)}$ $\qquad y = (x_1, x_2)\begin{pmatrix} w_1' \\ w_2' \end{pmatrix} + b'$

¿puede aprender XOR?

NO!

$(0\ 0)\begin{pmatrix} w_1' \\ w_2' \end{pmatrix} + b' = 0 \Rightarrow b' = 0$

$(0\ 1)$ y $(1\ 0) \Rightarrow w_1' = 1,\ w_2' = 1$

$(1\ 1) \Rightarrow w_1' + w_2' = 0$ $(\rightarrow\leftarrow)$

ya sabemos que con relu sí puede. $\Rightarrow$ no linealidad es imprescindible

¿Qué clase de funciones lineales podemos capturar?

$\varphi(x_1, x_2, \ldots, x_k)$ función booleana en k inputs $x_i \in \{0,1\}$

$\varphi \equiv \alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_r \qquad \alpha_i = l_{i1} \wedge l_{i2} \wedge \cdots \wedge l_{ir_i}$

¿puedo representar α_i como un perceptrón?

$\alpha_i(x_1, x_2, x_3, x_4, x_5) = x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \neg x_4 \wedge x_5$

$$\rightsquigarrow x_1 + x_5 - 2x_2 - 2x_3 - 2x_4 - 1$$

$$= \begin{cases} 1 & \text{si } x_1, x_5 = 1 \text{ y } x_2, x_3, x_4 = 0 \\ \leq 0 & \text{e.o.c.} \end{cases} \qquad W_i = \begin{pmatrix} 1 \\ -2 \\ -2 \\ -2 \\ 1 \end{pmatrix} \quad b = -1$$

si $\alpha_i(x_1, \ldots, x_5) = x_1 \wedge x_2 \wedge x_3 \wedge \neg x_4 \wedge \neg x_5$

$$\rightsquigarrow x_1 + x_2 + x_3 - 3x_4 - 3x_5 - 2 \qquad W_i = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -3 \\ -3 \end{pmatrix} \quad b = -2$$

$$= \begin{cases} 1 & \text{si } x_1, x_2, x_3 = 1 \text{ y } x_4, x_5 = 0 \\ \leq 0 & \text{e.o.c.} \end{cases}$$

$$\alpha_i = \bigwedge_{i \in P} x_i \wedge \bigwedge_{i \in N} \neg x_i \rightsquigarrow \sum_{i \in P} x_i - |P| \sum_{i \in N} x_i - |P| + 1$$

$\Rightarrow$ a cada α_i le puedo asignar un perceptrón con activación relu

capa de salida puede ser simplemente un hsigmoid $(h_1 + h_2 + \cdots + h_r - 0.5)$

(recordar que XOR necesita cantidad exponencial de neuronas)

Más aún:

Teo (UAT): no f una función continua de $[0,1]^k \longrightarrow [0,1]$.

Para todo $\varepsilon > 0$ existen W, b, U tq la función

$$F(x) = sig(xW + b)U \quad \text{cumple con}$$

$$\|f - F\| < \varepsilon \qquad (|f(x) - F(x)| < \varepsilon)$$

[En otras palabras, las funciones computadas por redes neuronales son "densas" en el espacio de las funciones continuas $[0,1]^k \to [0,1]$]