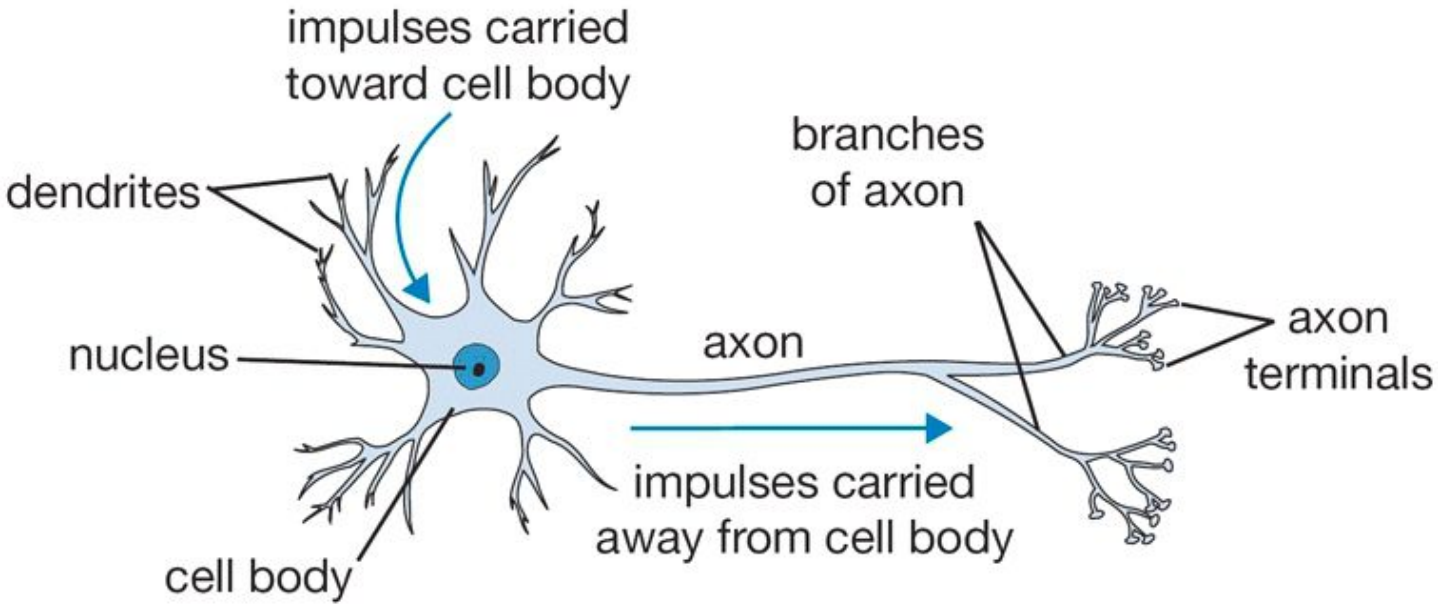
Redes Neuronales

Motivación biológica

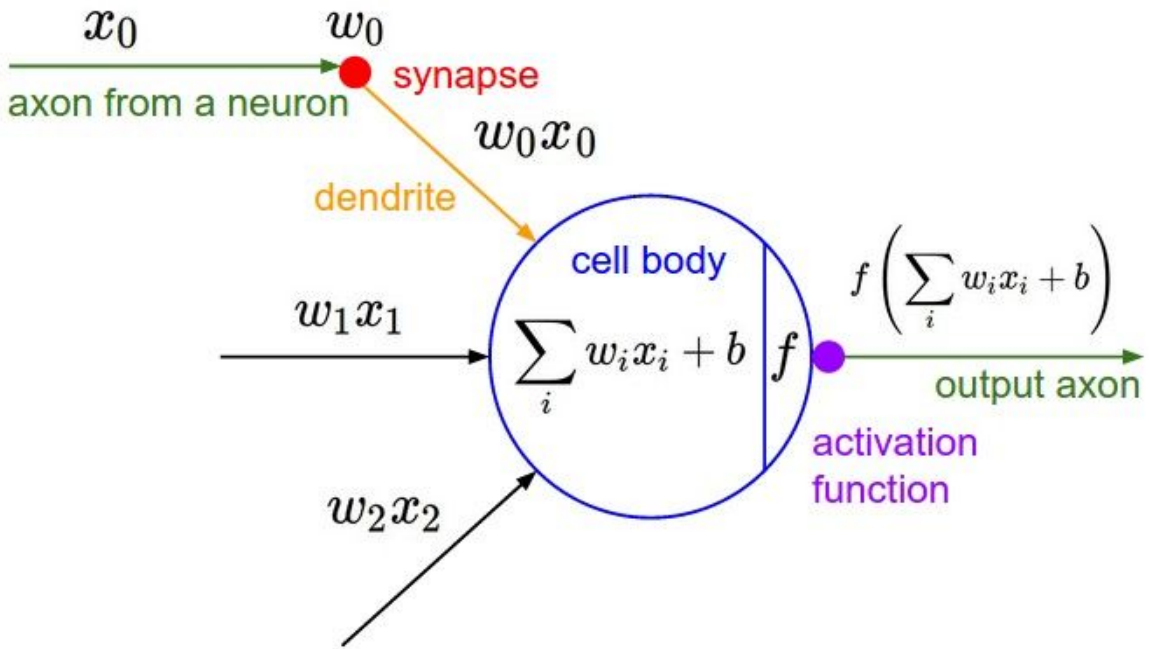
Sistema nervioso humano: 86.000 millones de neuronas

aproximadamente 10^14 - 10^15 sinapsis



Formulación matemática

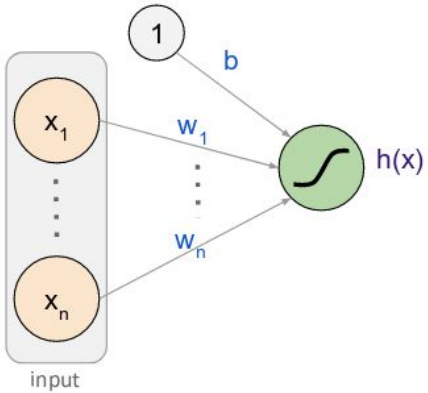
Operación lineal (preactivación) + función de activación no lineal



Formulación matemática

Pre activación lineal

$$a(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^T \mathbf{x} + b$$



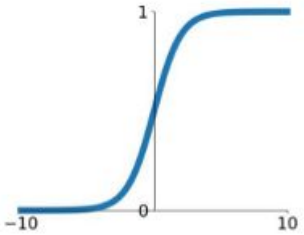
Activación no lineal

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Funciones de Activación

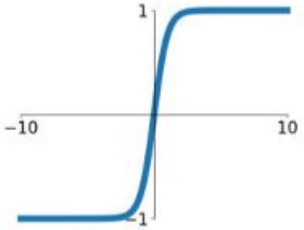
Sigmoid

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



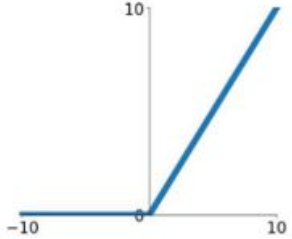
tanh

$$\tanh(x)$$



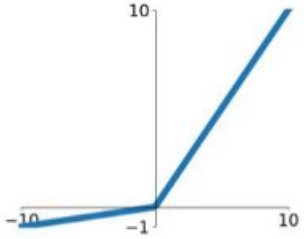
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

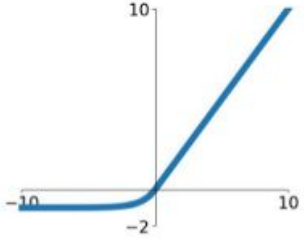


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Red neuronal

Entrada: x

Pre activación:

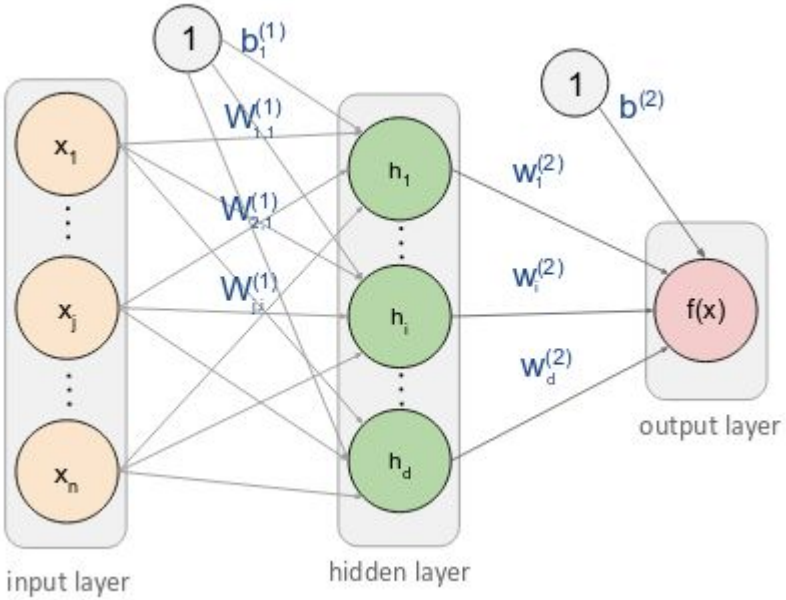
$$\mathbf{a} = \mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}$$

Activación capa oculta:

$$\mathbf{h}^{(1)}(\mathbf{x}) = g(\mathbf{a}(\mathbf{x}))$$

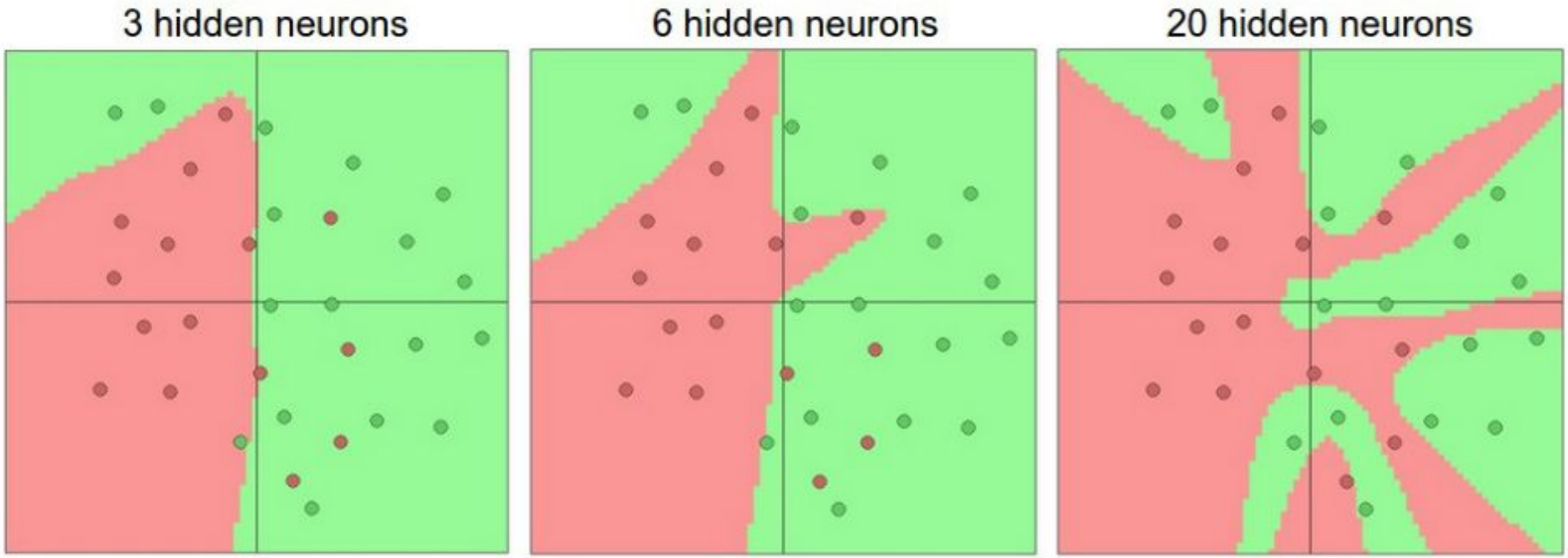
Capa de Salida:

$$f(\mathbf{x}) = o\left(\mathbf{w}^{(2)T}\mathbf{h}^{(1)} + b^{(2)}\right)$$



$W^{(1)}$ matriz de (n_features, n_hidden)
$W^{(2)}$ matriz de (n_hidden, 1)

Capacidad de la capa oculta



Cuanto más neuronas, más capacidad de ajuste tiene el modelo

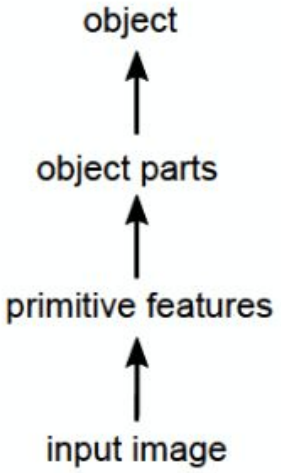
Capacidad: relacionada con cantidad de neuronas

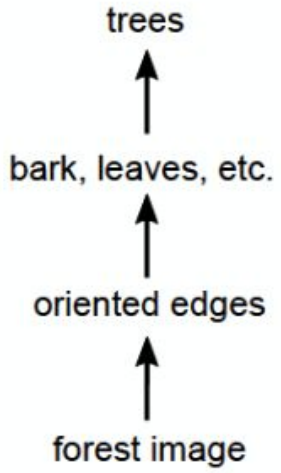
Capacidad de las Redes Neuronales

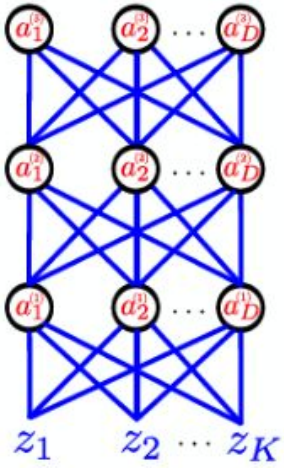
Aproximación Universal: Una RRNN con una única capa oculta y cantidad finita de Neuronas, puede aproximar cualquier función. Cybenko, 1989; Hornik 1991.

Capacidad de la red neuronal crece exponencialmente con la profundidad, pero polinomial en la cantidad de unidades ocultas. Montufar et al. 2014

Redes Neuronales Profundas







¿Cómo encontrar los pesos?

Definimos una arquitectura de red:

$$f(\mathbf{x};\theta) := f^{(n)}\left(f^{(n-1)}\left(\dots f^{(2)}\left(f^{(1)}\left(\mathbf{x}\right)\right)\dots\right)\right)$$

Lo planteamos como un problema de optimización:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} L(f(\mathbf{x}_i;\theta), y_i)$$

Con los parámetros:

$$\theta = \left[W^{(1)}, W^{(2)}, \dots, W^{(n)}\right]$$

Función de error a minimizar

Para fijar ideas, supongamos un problema de clasificación binario, se puede definir la siguiente función de error:

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(y * log(\hat{y}_i) + (1-y) * log(1-\hat{y}_i))$$

Conocida como binary cross entropy.

Hasta acá, todo lo definido es derivable (o al menos, derivable a trozos)

Utilizar descenso por gradiente!

Descenso por gradientes

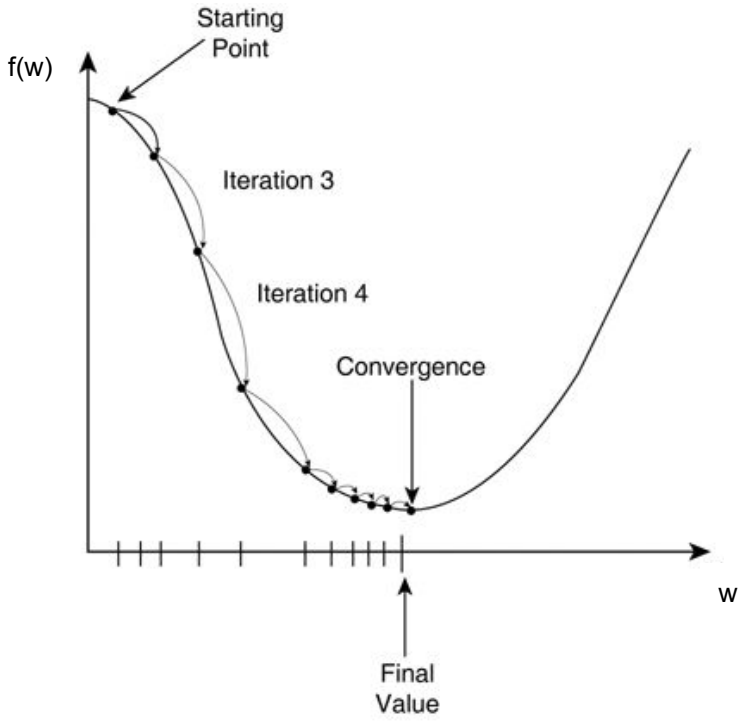
Gradiente:

- "derivada primera para funciones multidimensionales"
- Es un vector que indica la dirección en que la función crece más

Actualizar iterativamente:

$$w^{k+1} = w^k - \alpha \nabla f(w^k).$$

donde α se conoce como learning rate



Momentum

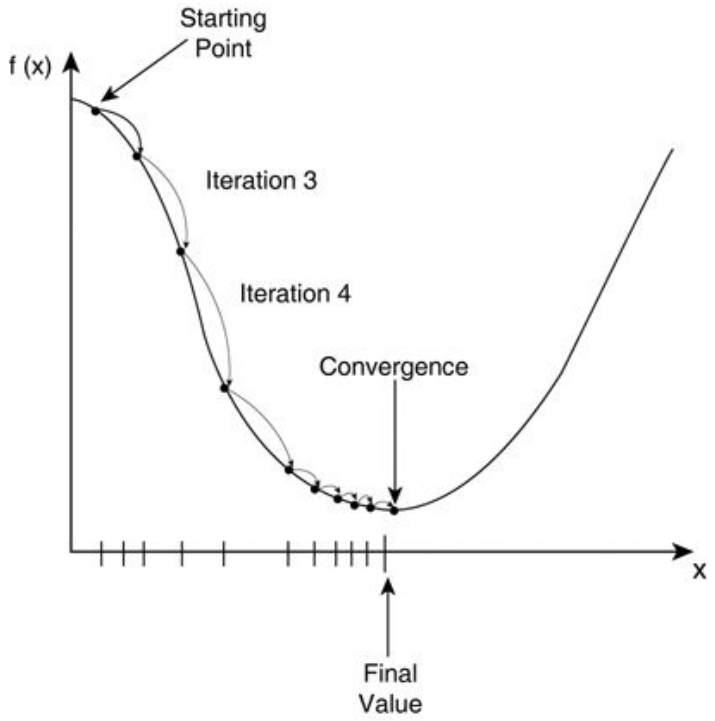
Una forma en que podemos acelerar las cosas es agregar un poco de inercia (momentum) al algoritmo:

$$z^{k+1} = \beta z^k + \nabla f(w^k)$$

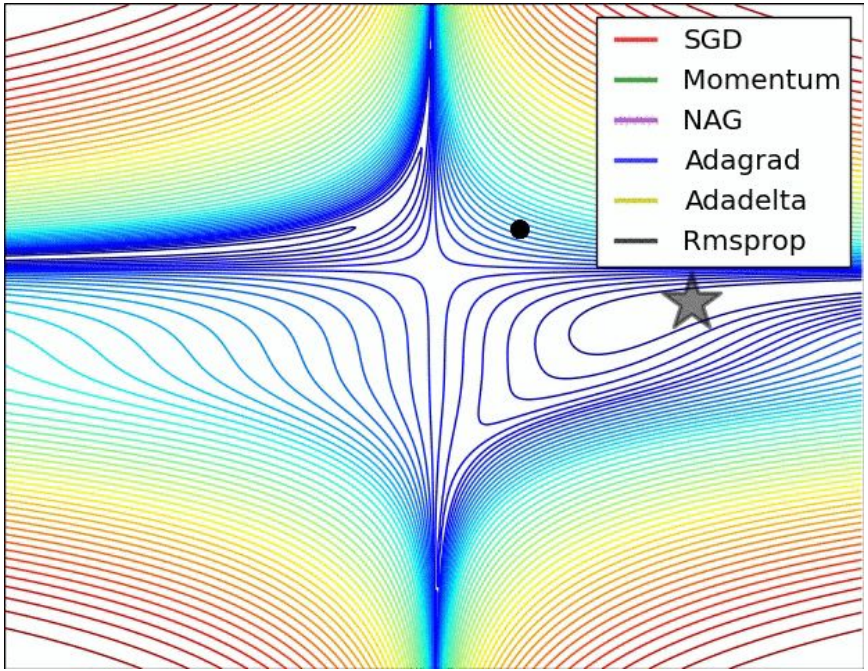
$$w^{k+1} = w^k - \alpha z^{k+1}$$

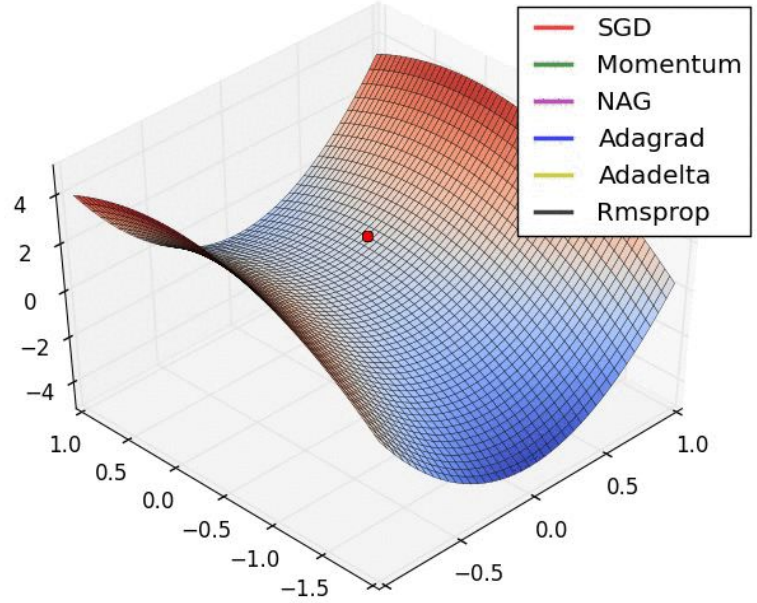
donde α se conoce como *learning rate*

y β como *inertia*



Otras variantes





Descenso por gradientes estocástico

La función de error a minimizar es la siguiente:

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{n} L(f(\mathbf{x}_i; \theta), y_i)$$

Implica calcular el gradiente para cada uno de los ejemplos entrenamiento.

Solución: minimizar la función en un batch de datos

¿Cómo se calculan los gradientes?

De manera numérica

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \qquad f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h}$$

De manera analítica: si bien la función que queremos minimizar es compleja, no es más que la composición de funciones sencillas.

Regla de la cadena

Regla de la cadena

Supongamos x es un real, f y g funciones reales:

$$y = g(x), z = f(g(x)) = f(y)$$

Entonces, la regla de la cadena dice:

$$z'(x) = f'(g(x))g'(x)$$

Es decir:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

Grafo computacional

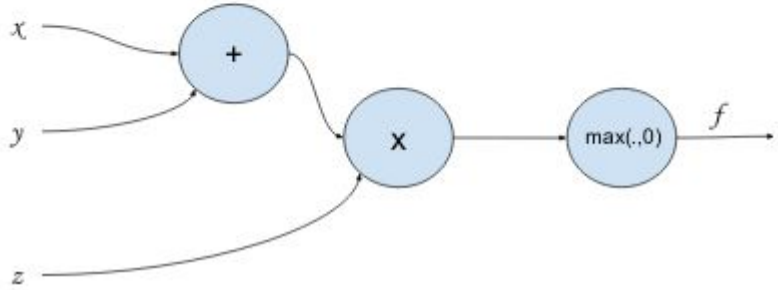
$$f(x,y,z) = \max\Big((x+y)z, 0\Big)$$

Composición de funciones elementales:

$$f_{\text{sum}}(x,y) = x+y \quad \longrightarrow \quad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

$$f_{\text{prod}}(x,y) = xy \quad \longrightarrow \quad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$f_{\max}(x) = \max(x,0) \quad \longrightarrow \quad \frac{\partial f}{\partial x} = 1_{\{x>0\}}$$



Backpropagation: forward

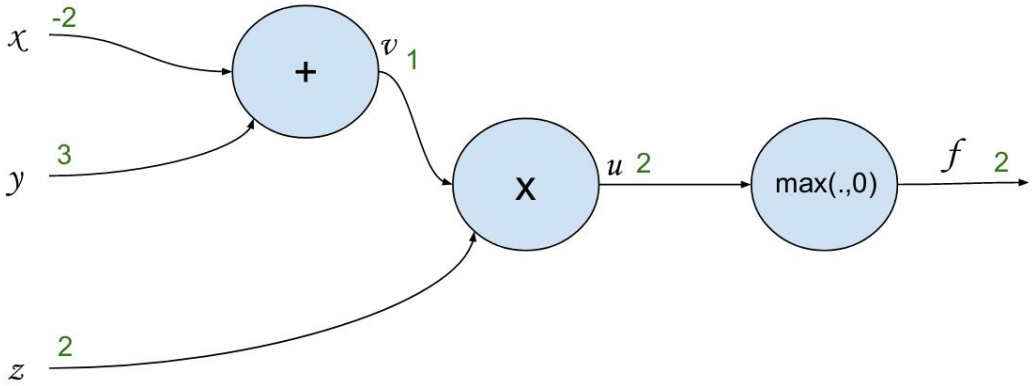
$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$v(x,y) = x+y$

$u(z,v) = zv$

$f(u) = \max(u,0)$

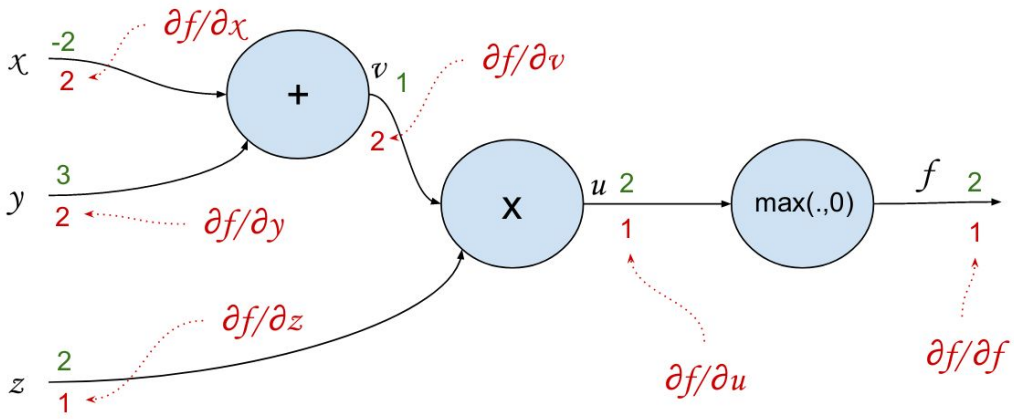
Pasada hacia delante
(Forward pass)



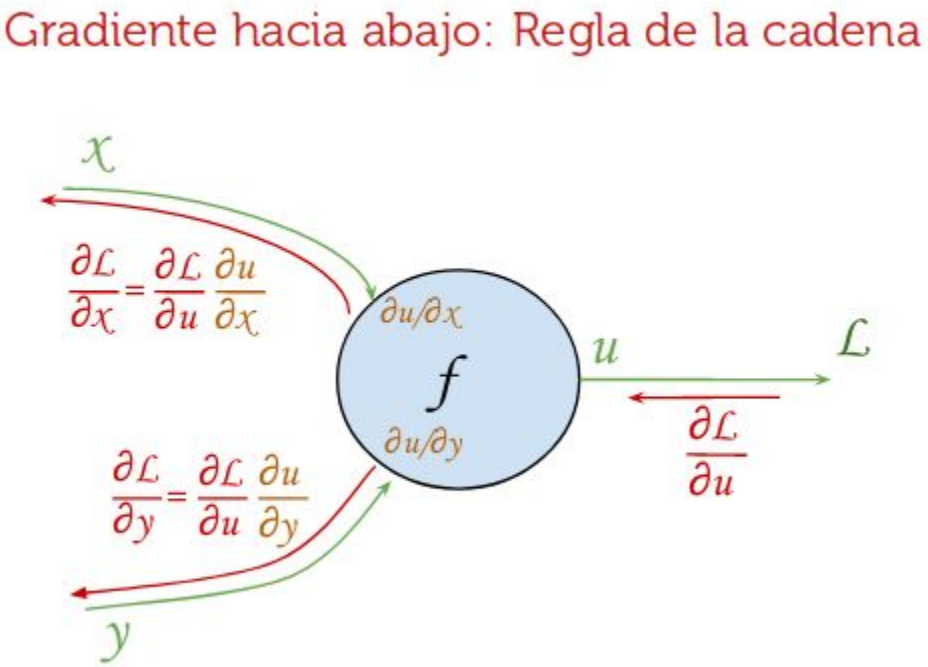
Back propagation: backward

$$f(x,y,z) = \max\Big((x+y)z, 0\Big), \quad x=-2,\ y=3,\ z=2$$

$v(x,y) = x+y$	$\longrightarrow$	$\frac{\partial v}{\partial x} = 1 \quad \frac{\partial v}{\partial y} = 1$
$u(z,v) = zv$	$\longrightarrow$	$\frac{\partial u}{\partial z} = v \quad \frac{\partial u}{\partial v} = z$
$f(u) = \max(u,0)$	$\longrightarrow$	$\frac{\partial f}{\partial u} = 1_{\{u>0\}}$



Back propagation



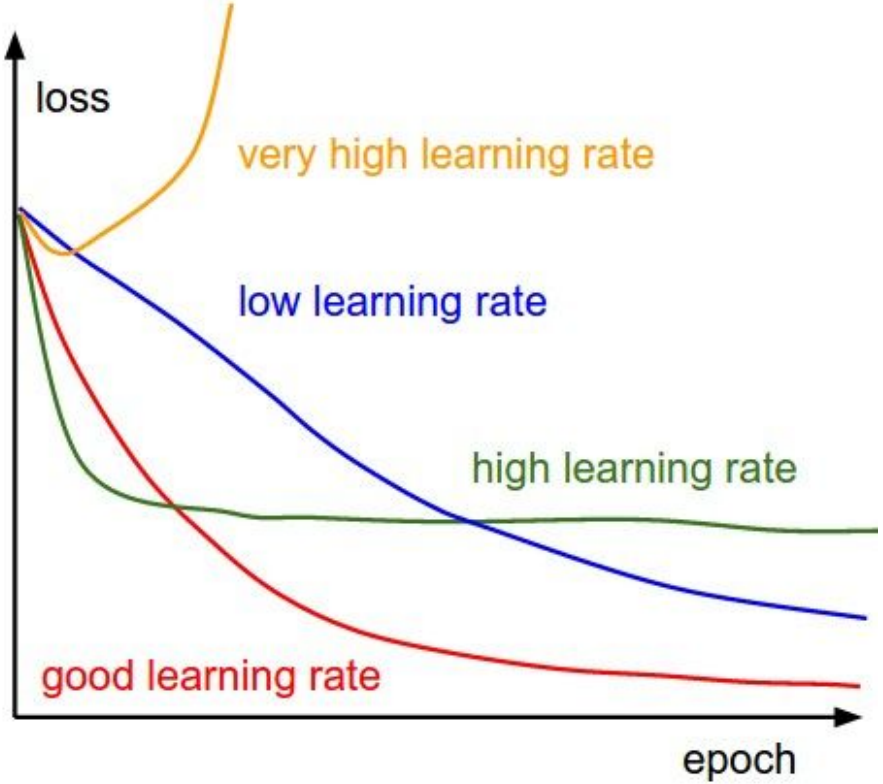
Resumen

- Una red neuronal consiste en la composición de funciones lineales con no lineales
- El problema está en encontrar los pesos correspondientes a la red
- Para esto, se utiliza descenso por gradiente estocástico, minimizando un error dado
- Para calcular los gradientes se utiliza back propagation

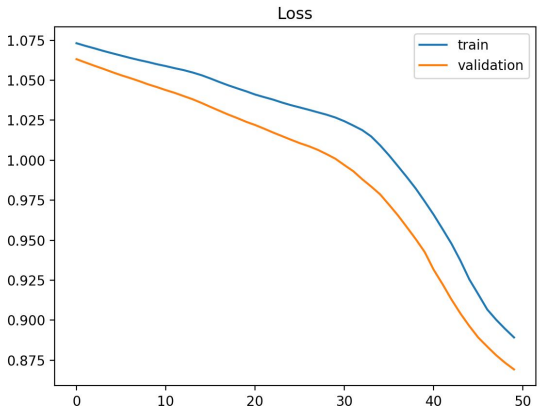
Entrenamiento: evolución del error

Learning rate es uno de los parámetros más importantes

La función de error arroja pistas a la hora de elegirlo



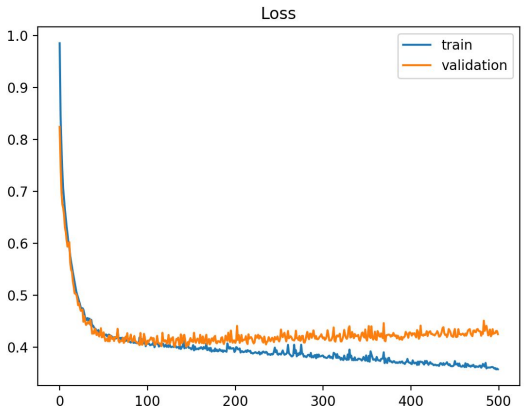
Entrenamiento: evolución del error



Underfit:

Disminuir regularización

Aumentar el modelo

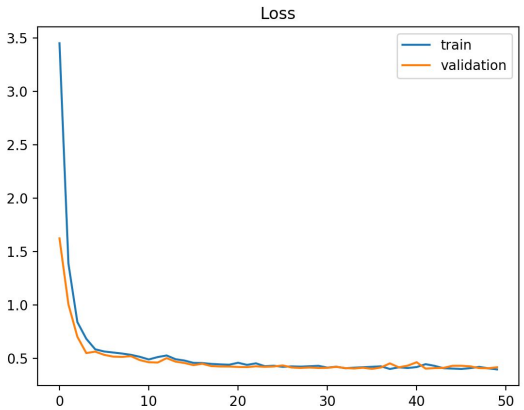


Overfit:

Aumentar regularización

Achicar el modelo

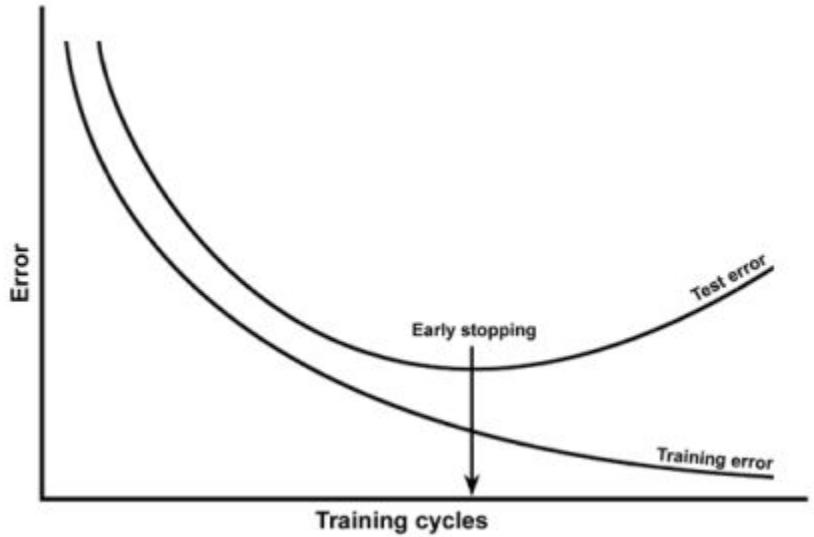
Agregar más datos



Correcto

Early stopping

Detener el entrenamiento cuando para de mejorar



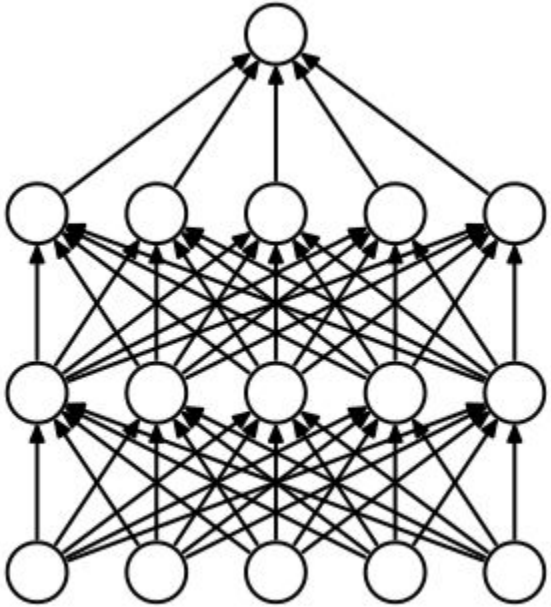
Regularización L1 y L2

Idea sencilla: modificar la función de pérdida para penalizar pesos grandes

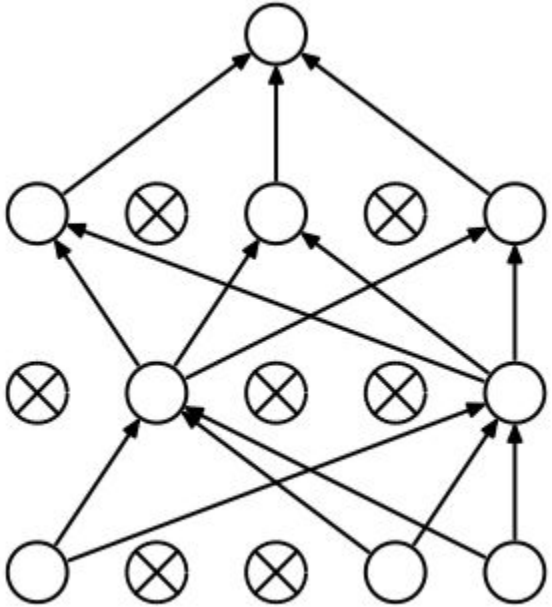
$$L(W) = \sum_{i=1}^{n} L_i(W) + \lambda R(W)$$

Donde R usualmente es la norma L1 o L2 de la matriz de pesos W

Drop out

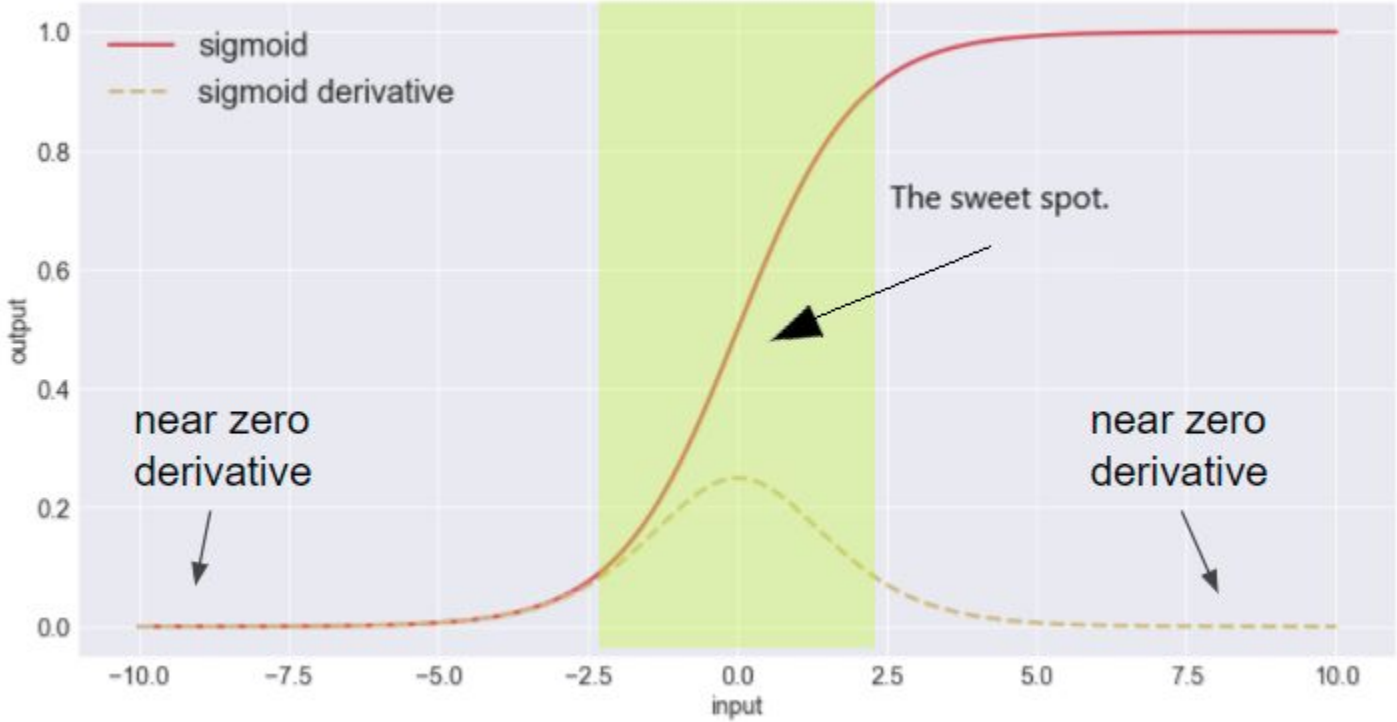


(a) Standard Neural Net



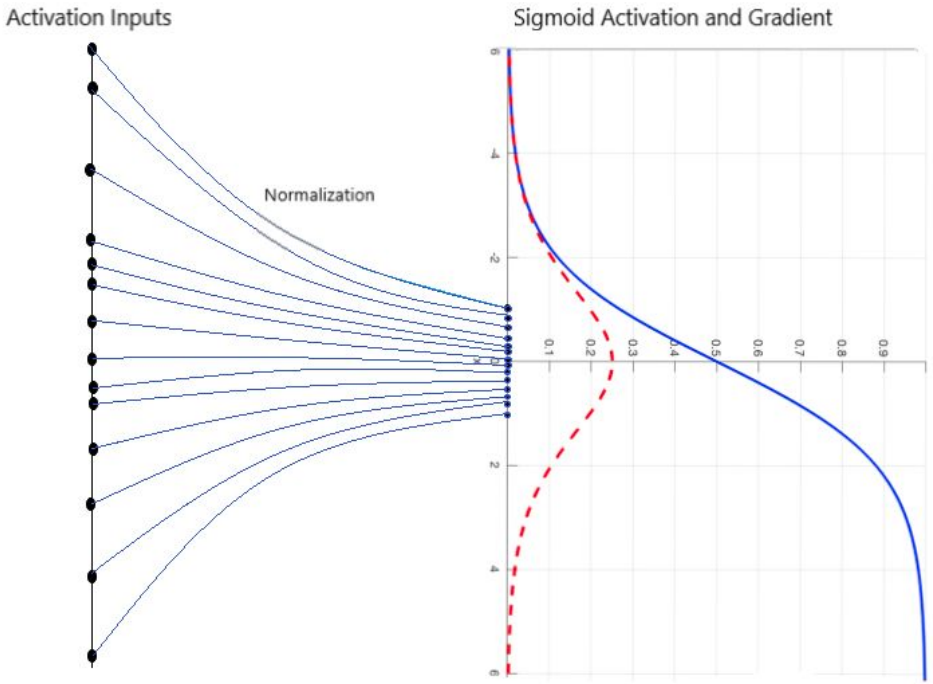
(b) After applying dropout.

Batch Normalization



Tomado de [Batch Normalization: Intuition and Implementation]

Batch Normalization



Tomado de [Batch Normalization: Intuition and Implementation]

Batch Normalization

Normalizar: quitar media y dividir entre la desviación estándar

Hay que estimarlas dinámicamente

Se debe realizar justo antes de la función de activación

Bibliografía / Recursos

curso DLvis, FIng - Udelar https://iie.fing.edu.uy/~mdelbra/DL2018/

curso cs231, Standford http://cs231n.github.io/

Why Momentum Really Works https://distill.pub/2017/momentum/

playground: https://playground.tensorflow.org/

optimizing-gradient-descent http://ruder.io/optimizing-gradient-descent/index.html