Redes Neuronales

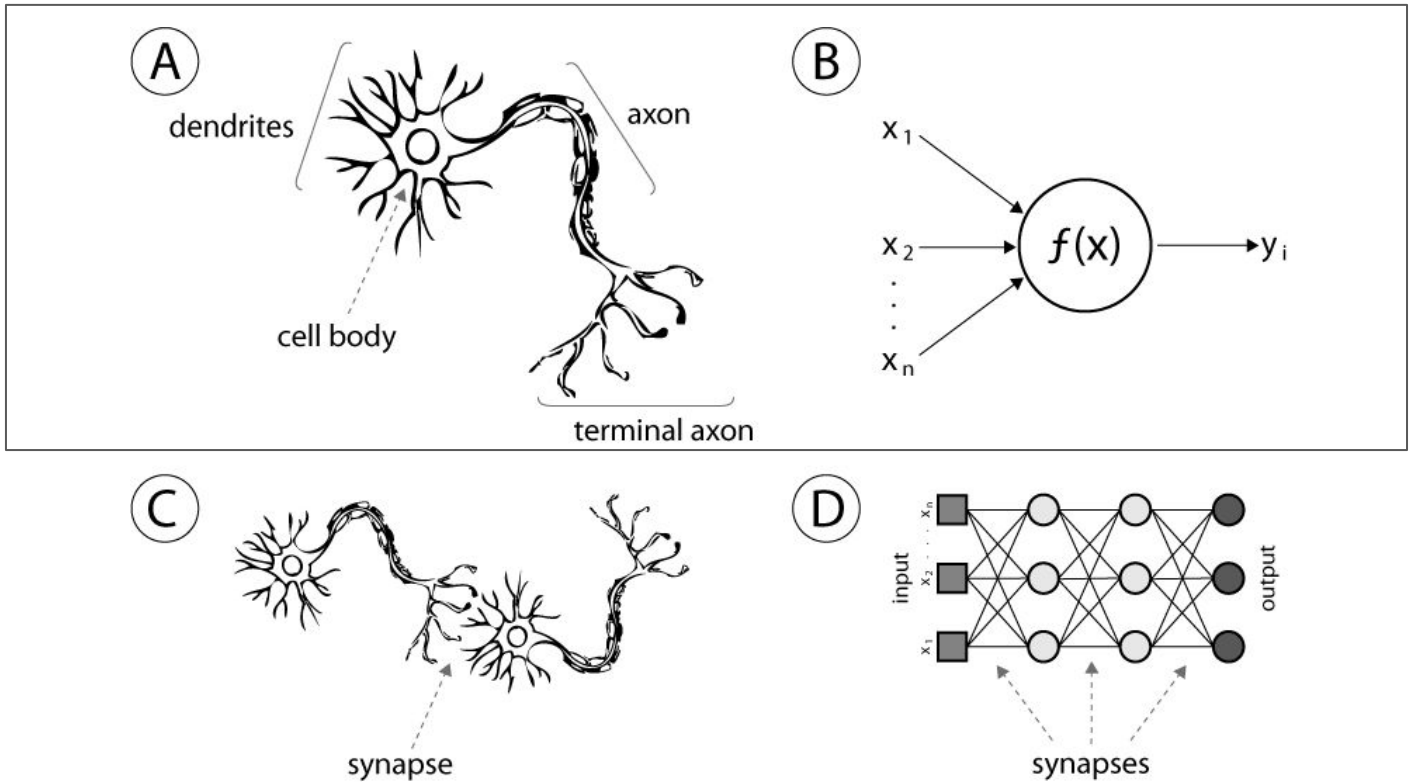
Dr. Álvaro Pardo

Universidad Católica del Uruguay
apardo@ucu.edu.uy
@AlvaroPardoUy

Temario

1. Modelo matemático de una neurona artificial. Redes Neuronales.
2. Entrenamiento de una red neuronal. Función de costo. Backpropagation.
3. Algoritmos para el entrenamiento de redes neuronales

Neurona Artificial

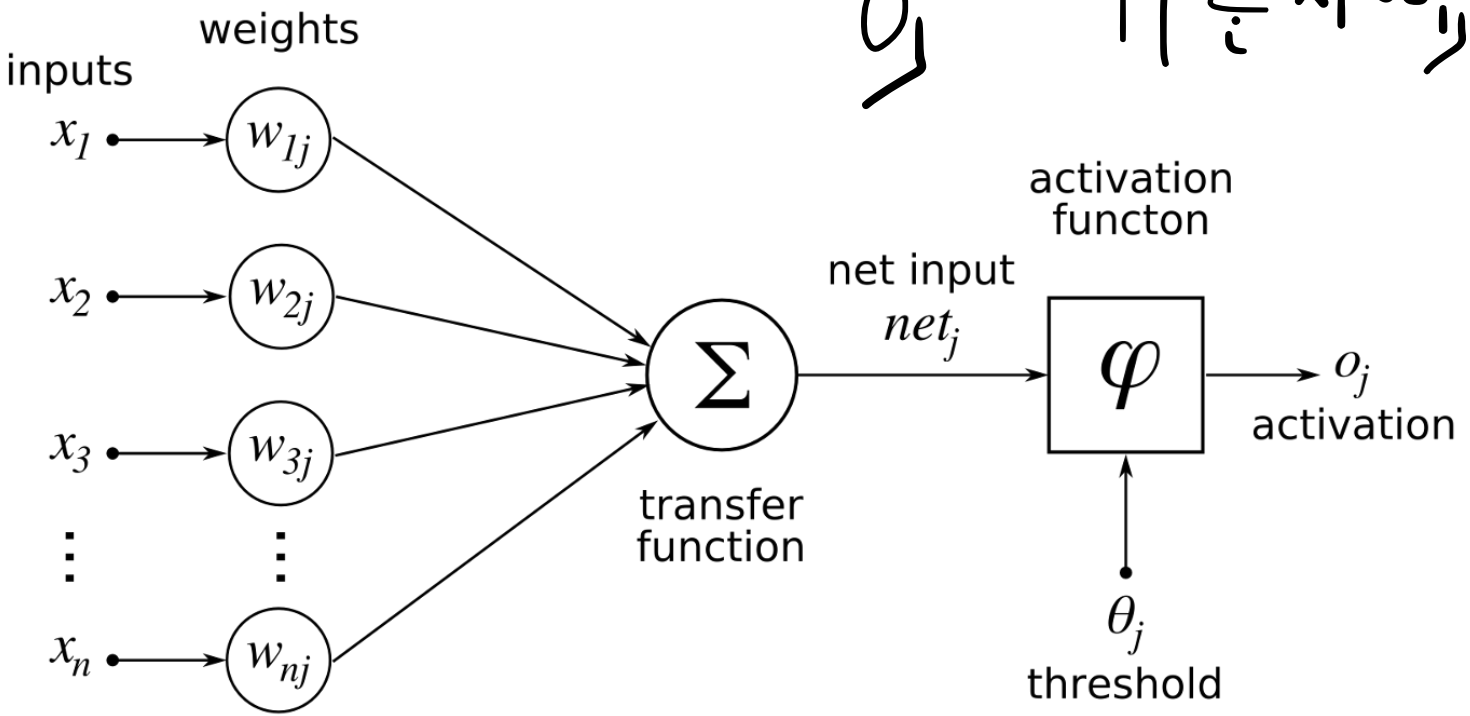


https://medium.com/@ivanliljeqvist/the-essence-of-artificial-neural-networks-5de300c995d6

Neurona Artificial

Modelo matemático:

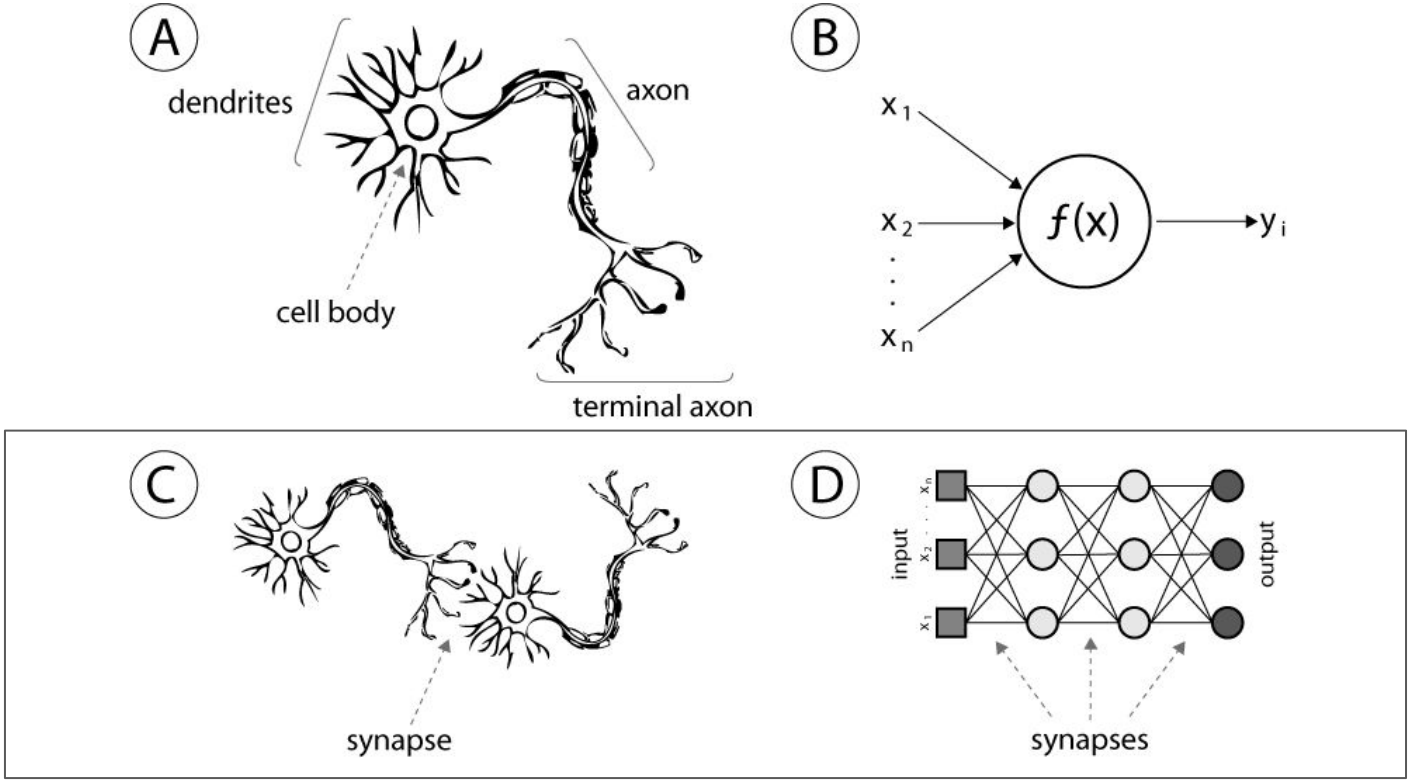
$$y_j = \varphi\left(\sum_i x_i w_{ij}\right) \geq \theta$$



https://medium.com/@ivanliljeqvist/the-essence-of-artificial-neural-networks-5de300c995d6

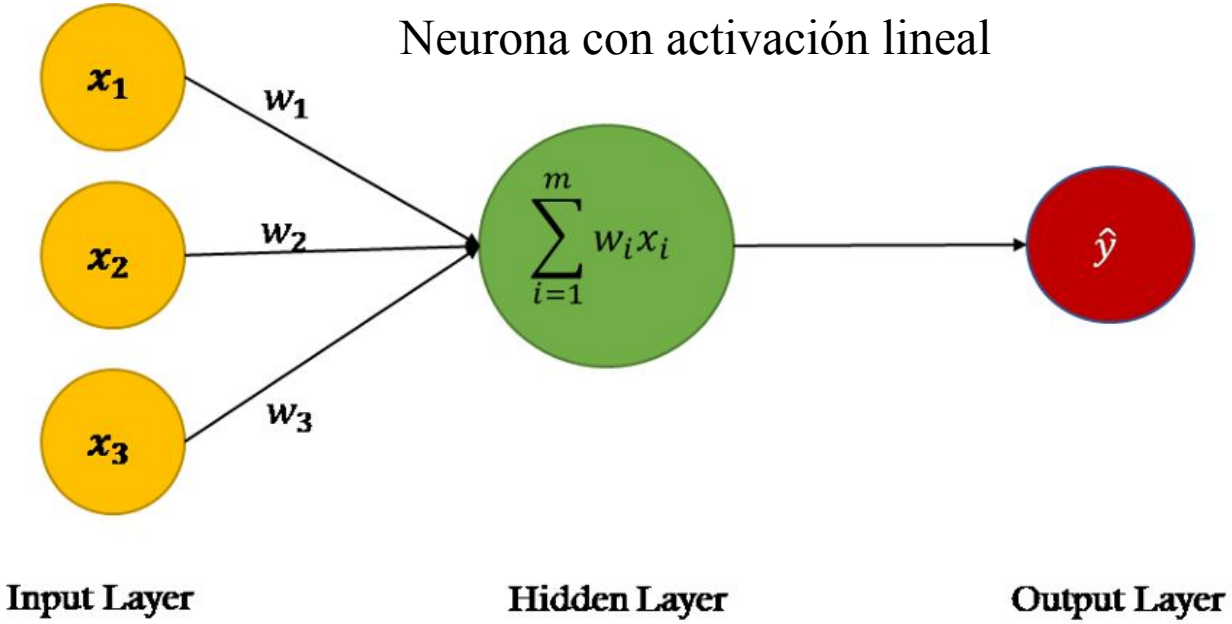
¿Cómo es una red neuronal?

Red Neuronal



https://medium.com/@ivanliljeqvist/the-essence-of-artificial-neural-networks-5de300c995d6

Entrenamiento: Caso Lineal



https://towardsdatascience.com/single-neuron-training-3fc7f84d67d

Train

$(\vec{x}^{(n)}, t^{(n)})$

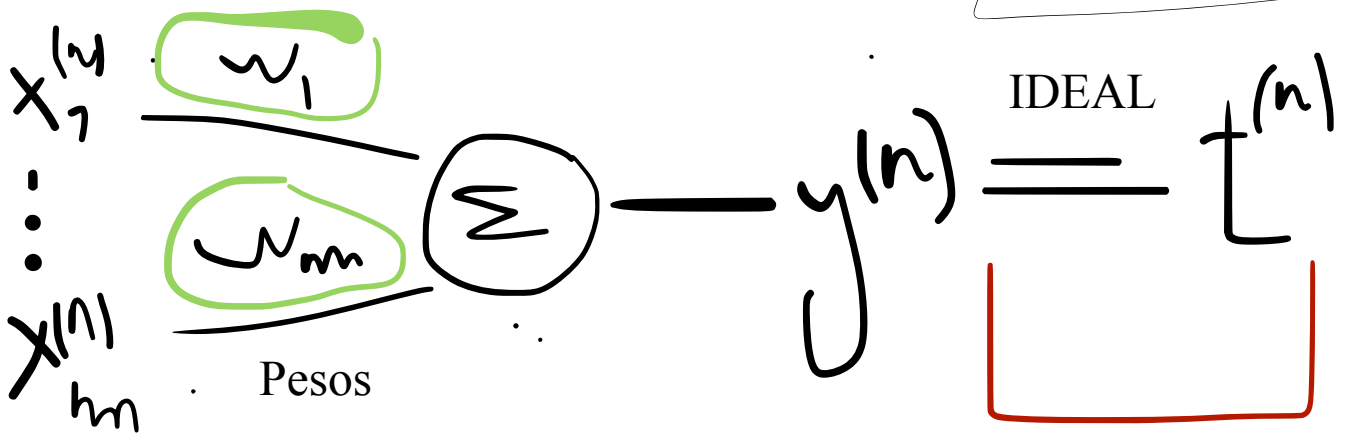
Entrada Salida

F. objetivo: minimiza la diferencia entre la salida de la red y el target y

COSTO

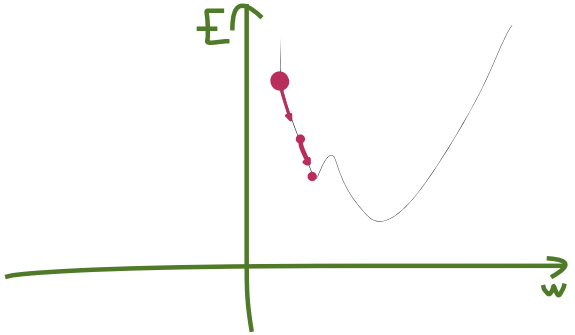
$E = \frac{1}{2} \sum_{n} (t^n - y^n)^2$

OBJETIVO $\min_{\{w_i\}} E$

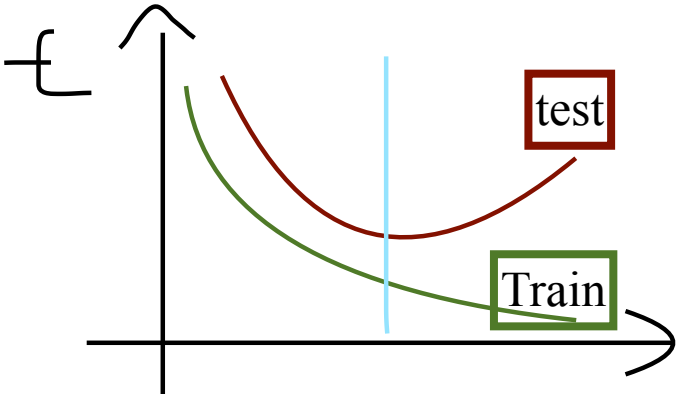


INCOGNITAS = PESOS

Solución al problema de min: descenso por gradiente



$$w_{k+1} = w_k - \eta \frac{dE}{d\omega}(\omega_k)$$



Salida

$$y = \sum_i w_i x_i = \mathbf{w}^\top \mathbf{x}$$

Error (MSE)

$$E = \frac{1}{2} \sum_{n \in train} (t^{(n)} - y^{(n)})^2$$

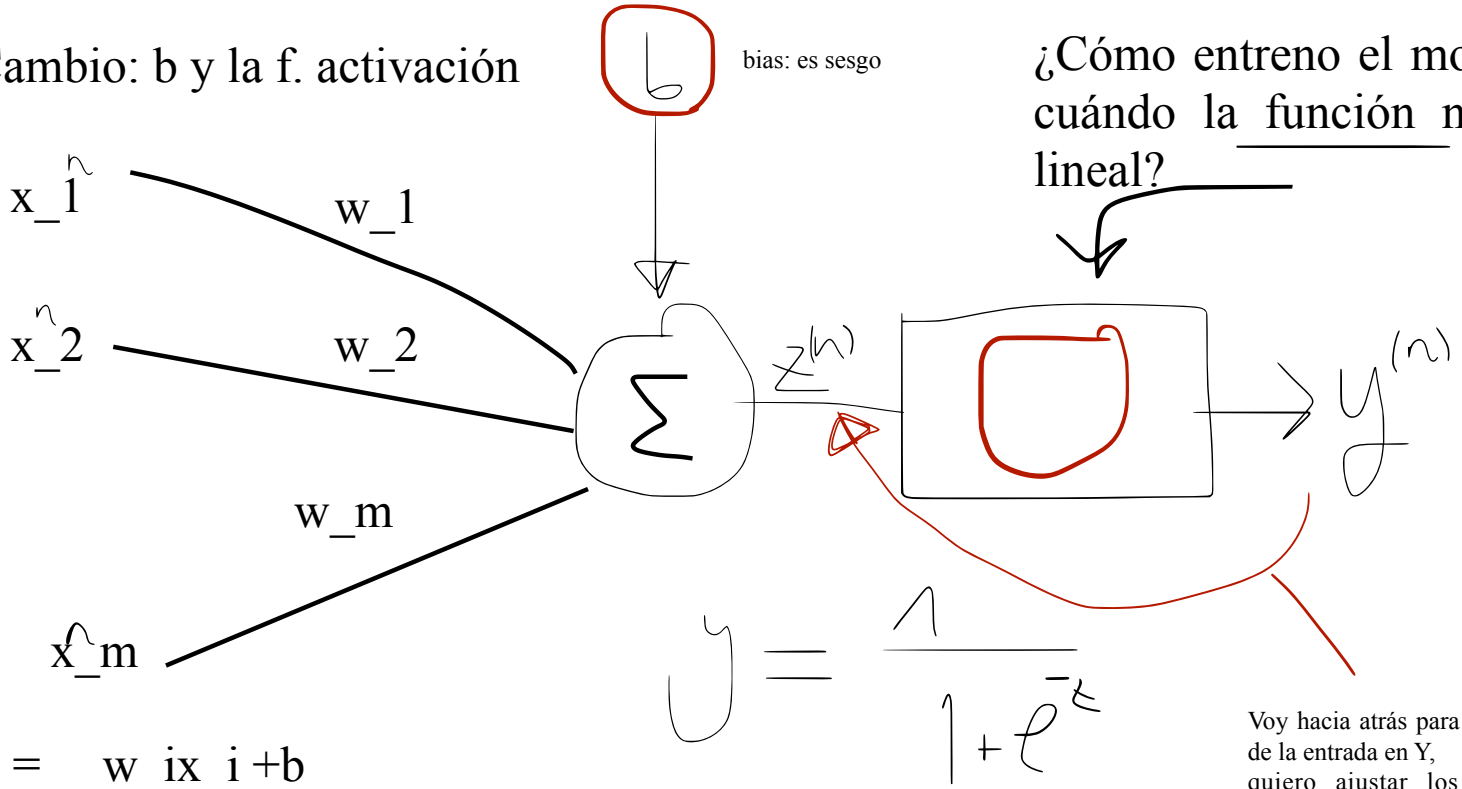
Derivada respecto a w_i

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial y^{(n)}}{\partial w_i} \frac{dE^{(n)}}{dy^{(n)}}$$

Ajuste del peso w_i

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \sum_n \eta x_i^{(n)} (t^{(n)} - y^{(n)})$$

Cambio: b y la f. activación



¿Cómo entreno el modelo cuándo la función no es lineal?

$$y = \frac{1}{1 + e^{-z}}$$

Voy hacia atrás para saber el impacto de la entrada en Y,
quiero ajustar los pesos w para optimizar el funcionamiento de la neurona, es decir, min E

obs: $z = w_i x_i + b$

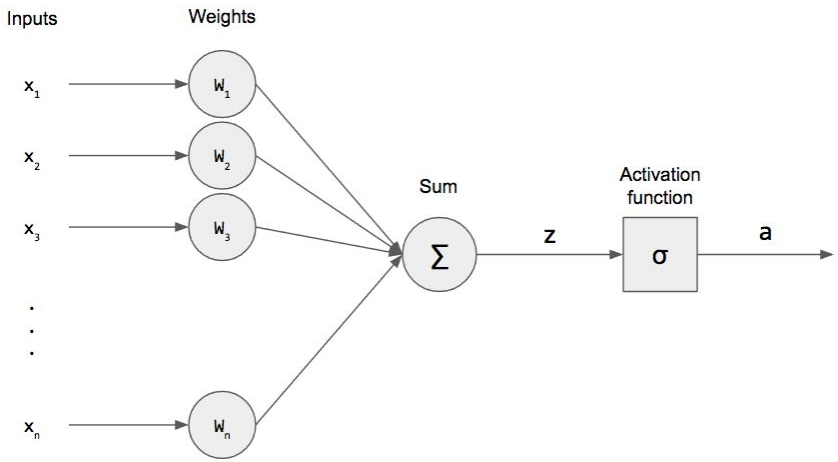
$= (b, w1, \ldots, wm) \begin{pmatrix} 1 \\ x1 \\ xm \end{pmatrix}$

$= (\vec{w}^{\top} \vec{x})$

FUNCIÓN DE ACTIVACIÓN

La función de activación permite resolver el problema de la clasificación.
Está entre (0,1) o entre(-1,1)
cuánto más cerca de uno: una clase y cerca de 0 o -1 es de la otra

Entrenamiento: Caso No Lineal



$$a = \sigma\left(\sum_{k=1}^{n} w_i x_i + w_0\right)$$

z: salida de la neurona antes de la activación

$$z = b + \sum_i w_i x_i$$

y: luego de la activación sigmoidal

$$y = \frac{1}{1 + e^{-z}}$$

dy/dz: derivada de la salida y respecto a la entrada a la activación z

$$\frac{dy}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2} = y(1 - y)$$

Derivada de la salida y respecto al peso w_i

$$\frac{\partial y}{\partial w_i} = x_i y(1 - y)$$

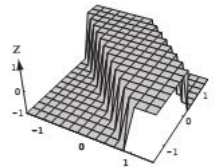
Si f no es sigmoidal hay que cambiar este paso

Ajuste del peso w_i

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^{(n)}}{\partial w_i} \frac{\partial E}{\partial y^{(n)}} = -\sum_n x_i^{(n)} y^{(n)} (1 - y^{(n)})(t^{(n)} - y^{(n)})$$

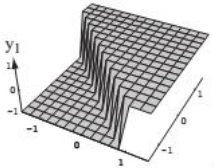
Clasificación: Intuición

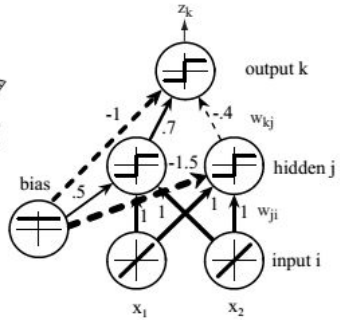
Combinación de ambas

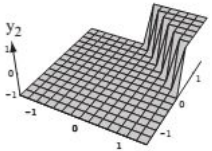


Contstrucción de una frontera de decisión con dos neuronas

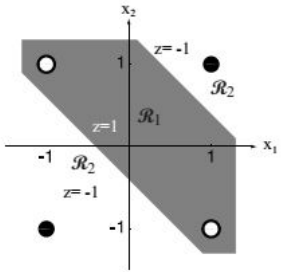
Función que genera la salida de una neurona



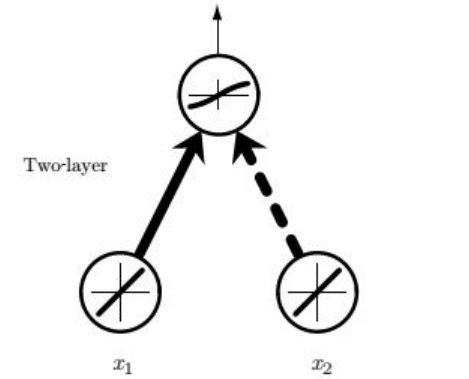


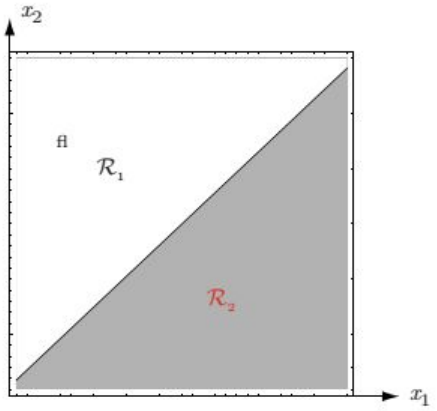


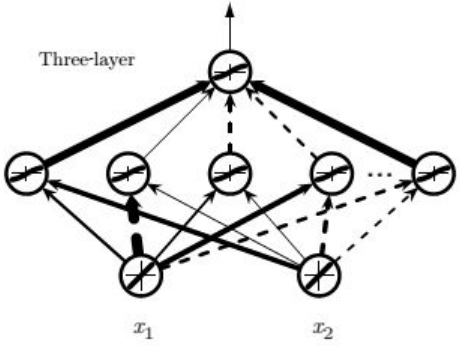
Función que genera la salida de otra neurona

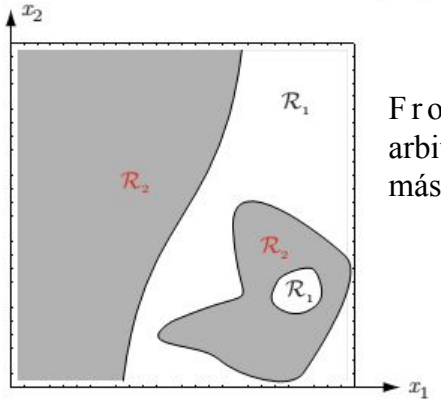


Extraída de Pattern Classification, Duda, Hart & Stork.





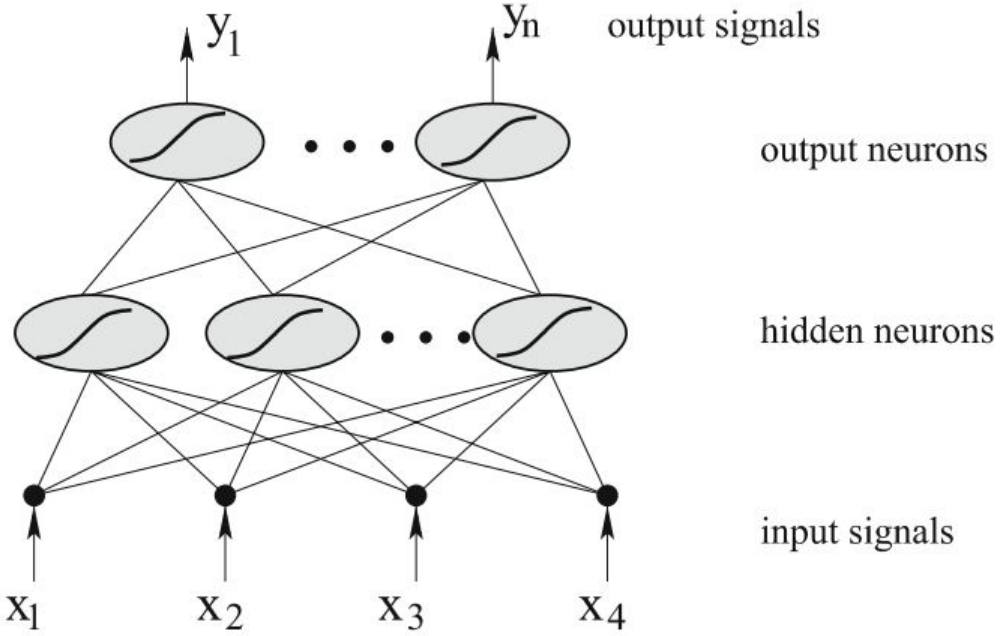




Fronteras de decisión arbitrarias con más de una neurona

Extraída de Pattern Classification, Duda, Hart & Stork.

Red Neuronal



Ref.: M. Kubat, An Introduction to Machine Learning.

Entrenamiento de una red

- Dado un vector de entradas x_n se calcula el error entre lo deseado (target) y lo obtenido:

$$\frac{1}{m}\sum_{j=1}^{m}(y_j(x_n) - t_j(x_n))^2$$

- *m* es la cantidad de neuronas (clases) en la capa de salida.
- El error total para todos los datos de entrenamiento es:

$$J(w_{ij}^k) = \sum_{n=1}^{N}\left(\frac{1}{m}\sum_{j=1}^{m}(y_j(x_n) - t_j(x_n))^2\right)$$

Entrenamiento

- El entrenamiento implica obtener los pesos que minimizan el error.
- Descenso de gradiente:

$$w_{ij}^{k+1} = w_{ij}^{k} - \eta \frac{\partial J(w_{ij}^{k})}{\partial w_{ij}^{k}}$$

Descenso por Gradiente: Pros y Contras

$$\theta^{k+1} = \theta^k - \eta \nabla L(\theta^k)$$

+ Fácil de entender
+ Fácil de implementar (backpropagation)
- Puede quedar atrapado en mínimos locales (problema conocido)
- Puede demorar en converger
- Si la red es grande los gradientes pueden consumir mucha memoria y complicar los requerimientos de HW.

Stochastic Gradient Descent (SGD)

En el caso de GD clásico es necesario computar la loss respecto a todos los datos de entrenamiento. Esto es costoso si se tienen muchos datos de entrenamiento; cada iteración implica pasar todos los datos por la red y computar las derivadas.

SGD ataca el problema procediendo de forma aleatoria. Se selecciona un punto al azar en cada iteración.

Para mitigar el avance ruidoso de SGD se puede trabajar con **mini-batches** o usar técnicas de regularización del avance.

Batch, Epoch, Mini-Batch, Iteration

Una ***epoch*** (época) implica una pasada por todos los datos del conjunto de entrenamiento.

En el modo ***batch*** todos los datos de entrenamiento se usan para calcular el gradiente y ejecutar una iteración.

Se puede procesar mediante ***mini-batches*** y en cada iteración se usan un subconjunto de los datos del conjunto de entrenamiento para estimar el gradiente. Varias iteraciones son necesarias para completar una *epoch*.

Mini-Batch Gradient Descent

Mini batch se puede aplicar tanto al GD clásico como a SGD. Se basa en utilizar un conjunto reducido de datos de entrenamiento para las estimaciones del gradiente.

Los datos de entrenamiento se dividen en mini-batches y los parámetros se actualizan luego de procesar cada mini-batch.

Los parámetros se actualizan con frecuencia luego de cada mini-batch.

El consumo de memoria se puede regular con el tamaño del mini-batch.

Ejemplo: Fraude

Momentum

$$v^{k+1} = mv^k - \eta \nabla L(\theta^k)$$
$$\theta^{k+1} = \theta^k + v^{k+1}$$

Genera una memoria en función de la velocidad del paso anterior y el parámetro de momento *m*.

Reduce las oscilaciones (especialmente crítico en SGD) y acelera la convergencia.

El momento se considera un hiperparámetro que debe ser seleccionado.

Nesterov Accelerated Gradient

El uso de momento puede complicar la convergencia si producto de este se pierden los mínimos locales (se sigue de largo).

En vez de calcular el gradiente en el punto actual Nesterov propone hacerlo en un punto intermedio a partir del punto actual y la velocidad anterior.

Evita perder mínimos locales y tiende a minimizar al acercarse a los mismos.

$$v^{k+1} = mv^k - \eta \nabla L(\theta^k + mv^k)$$
$$\theta^{k+1} = \theta^k + v^{k+1}$$

AdaDelta

En vez de promediar el valor de los gradientes para todo tiempo lo hace en una ventana mediante una ecuación de recurrencia con un factor de decaimiento.

$$E[g_k^2] = \gamma E[g_{k-1}^2] + (1-\gamma) g_k^2$$

$$\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{E[g_k^2] + \epsilon}} g_k$$

ADAM (Adaptive Moment Estimation)

Es común que en optimización se utilicen momentos de primer y segundo orden como mejora a los clásicos métodos de descenso por gradiente. Se busca reducir la velocidad al llegar a un punto cercano a mínimos locales. Es similar a AdaDelta pero se usan estimaciones de primer y segundo orden.

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k$$
$$v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^2$$

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k}$$
$$\hat{v}_k = \frac{v_k}{1-\beta_2^k}$$
$$\theta_{k+1} = \theta_k - \frac{\eta}{\sqrt{\hat{v}_k} + \epsilon} \hat{m}_k$$

Ejemplo Clasificación de Dígitos

Softmax

- Es una función de activación que hace que todas las salidas de la capa estén en el rango [0,1] y sumen 1. Es decir, se comportan como probabilidades.
- Si s_i es el valor estimado por la neurona i de la cada de salida la activación softmax se define como:

$$f(\vec{s})_i = \frac{e^{s_i}}{\sum_i e^{s_i}}$$

- Observación: La salida i del softmax depende de todos los valores s_i dada la normalización (denominador).

Categorical Cross Entropy Loss

Problemas de clasificación multiclase

- Es una función de loss que se optimiza durante el entrenamiento de la red.

$$CE = -\sum_{i=1}^{C} t_i \log(f(\vec{s})_i)$$

- f(s)_i son las salidas de la capa softmax (están entre 0 y 1 y suman 1) y t_i los valores de clase en el conjunto de entrenamiento (que son 1 o 0, pertenecen únicamente a una de las C clases, one-hot).
- La CE aumenta cuando las f(s)_i se alejan de t_i.

CE: Ejemplo para C = 2 (dos clases)

- Supongamos que t_1 = 1 (t_2 = 0):

$$CE = -t_1 \log(f(\vec{s})_1) - t_2 \log(f(\vec{s})_2) = -\log(f(\vec{s})_1)$$

- La gráfica muestra la ecuación anterior tomando como variable f(s)_1. A medida que f(s)_1 se aleja de 1 el valor de CE crece. El mínimo se da cuando f(s)_1 = 1.