

Input layer

Hidden layers

Output layer

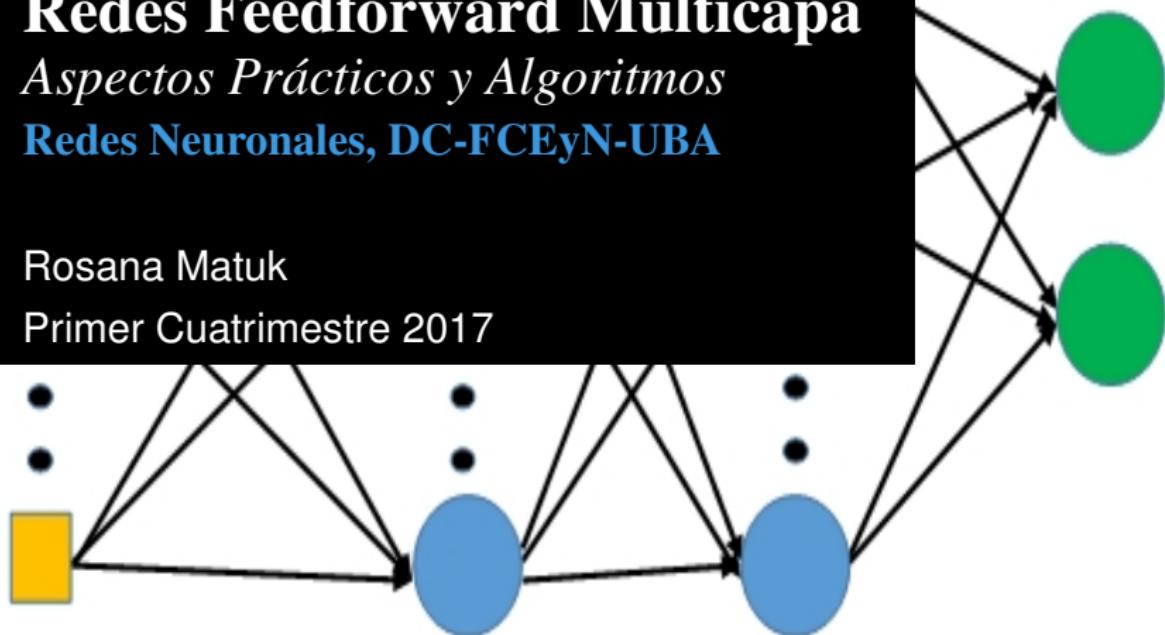
Redes Feedforward Multicapa

Aspectos Prácticos y Algoritmos

Redes Neuronales, DC-FCEyN-UBA

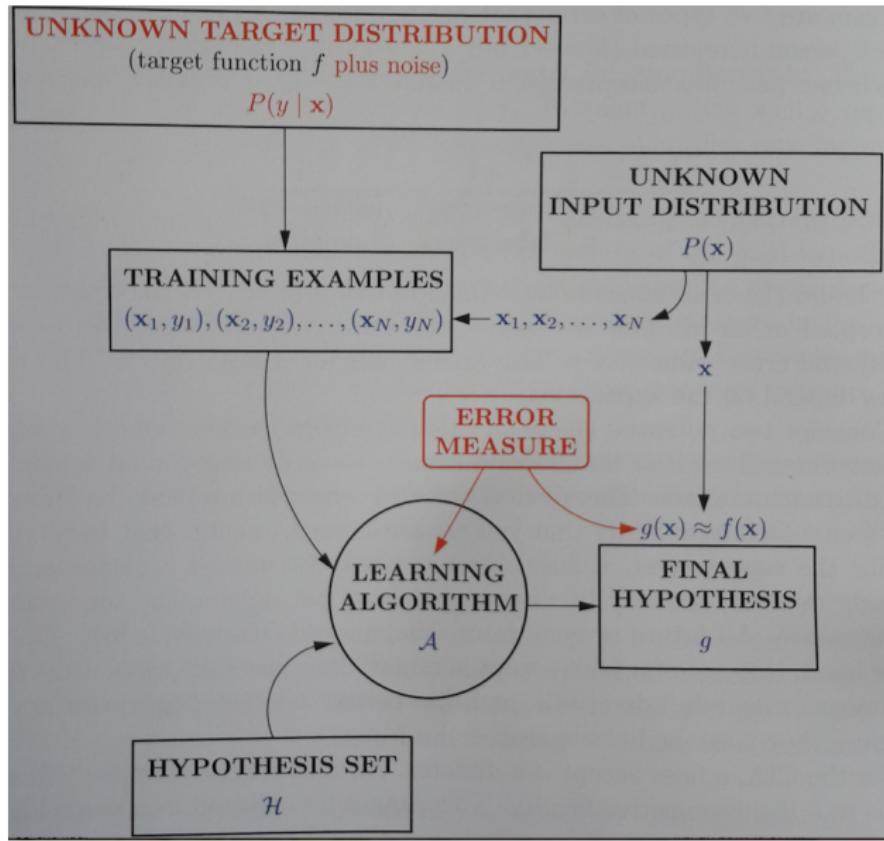
Rosana Matuk

Primer Cuatrimestre 2017

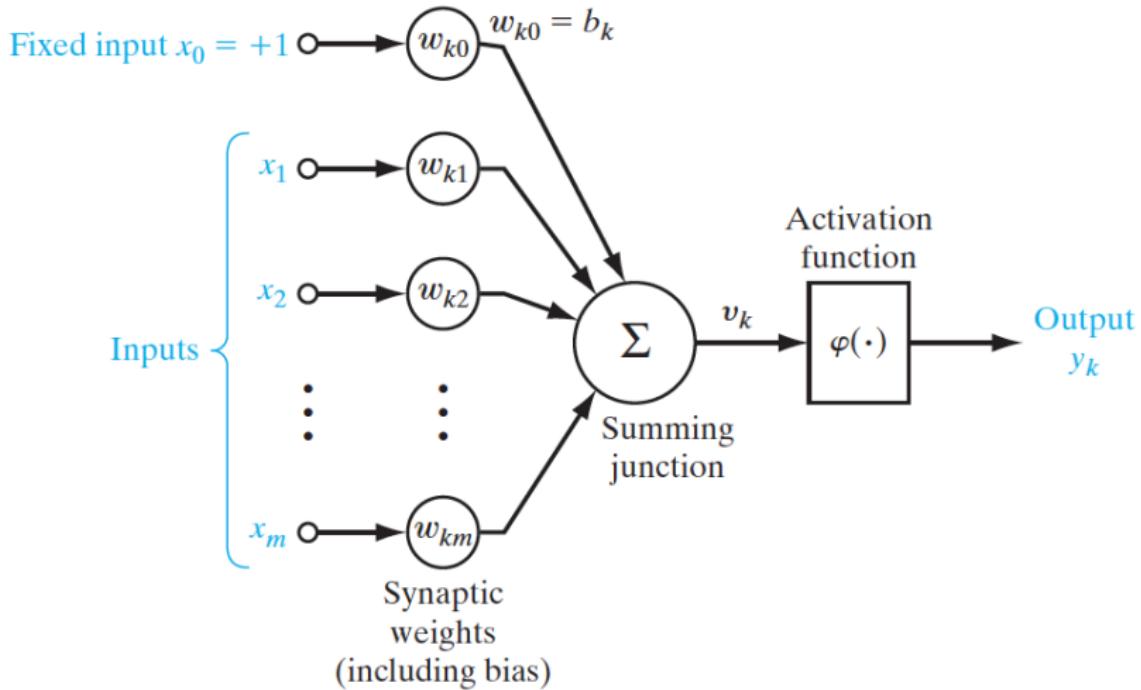


- Repaso de perceptrón simple*
- Introducción a las redes neuronales feedforward multicapa*
- Algoritmo de Backpropagation*
- Conclusiones*

Aprendizaje supervisado general



Red Neuronal Artificial Perceptrón Simple



Funciones de activación

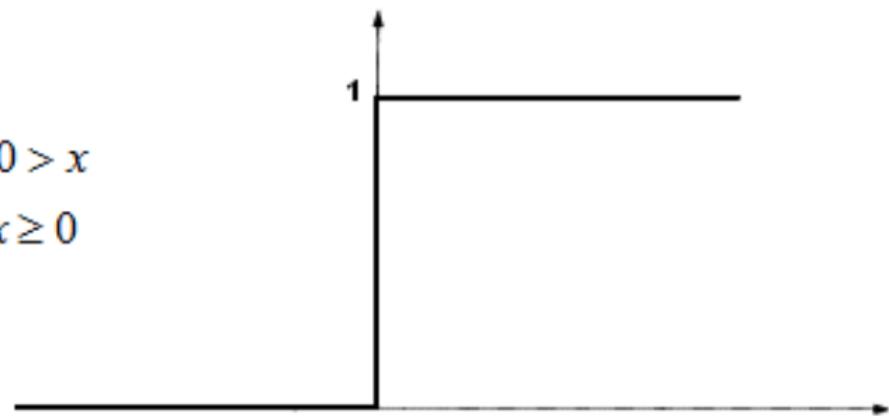
Algunas funciones de activación comúnmente usadas en redes neuronales:

- ▷ **Escalón**
- ▷ **Lineal**
- ▷ **No lineal:** Sigmoidea

Función escalón

Unit step (threshold)

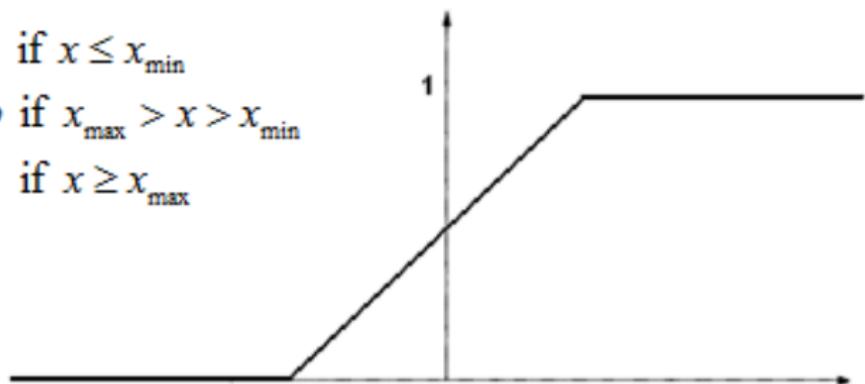
$$f(x) = \begin{cases} 0 & \text{if } 0 > x \\ 1 & \text{if } x \geq 0 \end{cases}$$



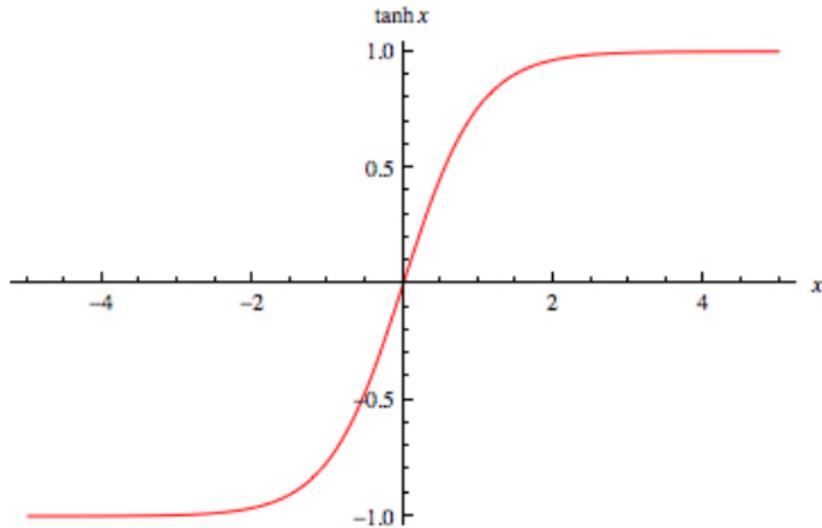
Función lineal de a tramos

Piecewise Linear

$$f(x) = \begin{cases} 0 & \text{if } x \leq x_{\min} \\ mx + b & \text{if } x_{\max} > x > x_{\min} \\ 1 & \text{if } x \geq x_{\max} \end{cases}$$



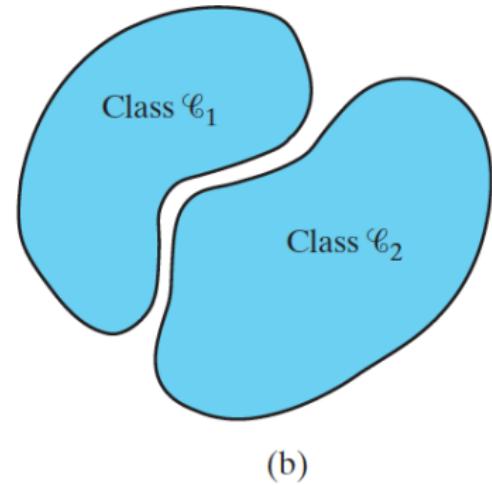
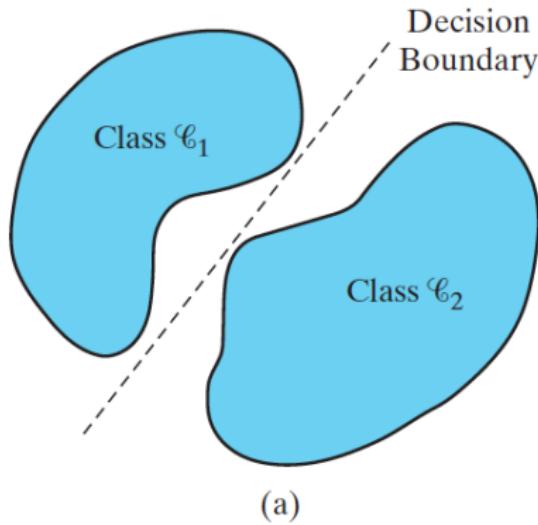
Función sigmoidea



Limitaciones del perceptrón simple (general)

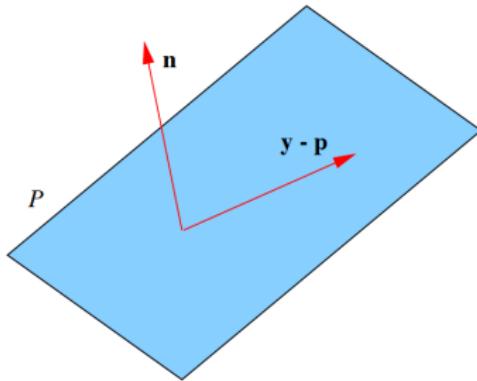
Los datos deben ser linealmente separables. Geométricamente, esta condición describe la situación en la cual existe un hiperplano en grado de separar, en el espacio vectorial de los inputs, las entradas con salidas positivas de las entradas con salidas negativas.

Separabilidad lineal



- (a) Datos linealmente separables
- (b) Datos no linealmente separables

Hiperplano



P es el conjunto de los puntos para los cuales $y - p$ es ortogonal a n .
Definición: Supongamos n y $p \in R^n$ con $n \neq 0$. El conjunto de todos los vectores $y \in R^n$, que satisfacen la ecuación $n.(y - p) = 0$ es llamado un hiperplano a través del punto p . Llamaremos n el vector normal del hiperplano.

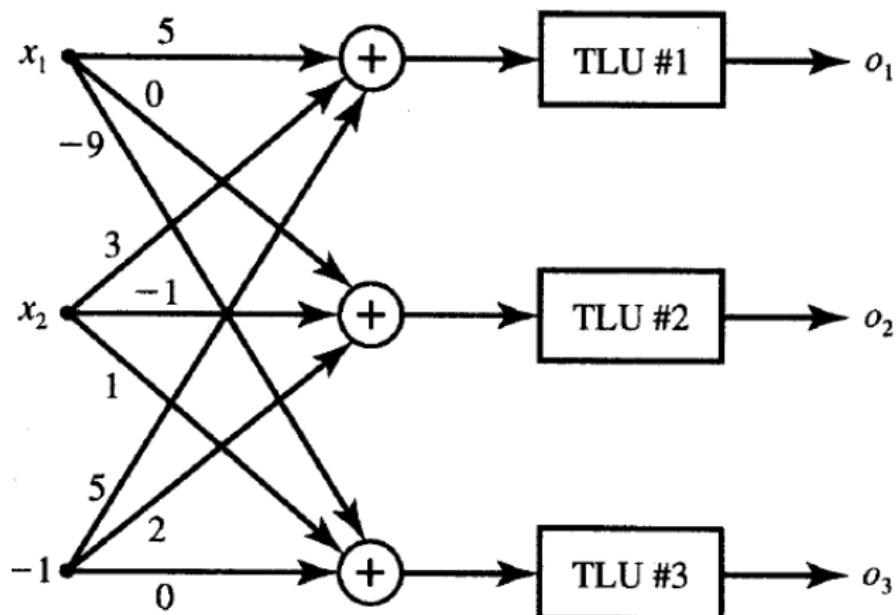
Objetivo del perceptrón: clasificar mediante el hiperplano

En perceptrón escalón buscamos:

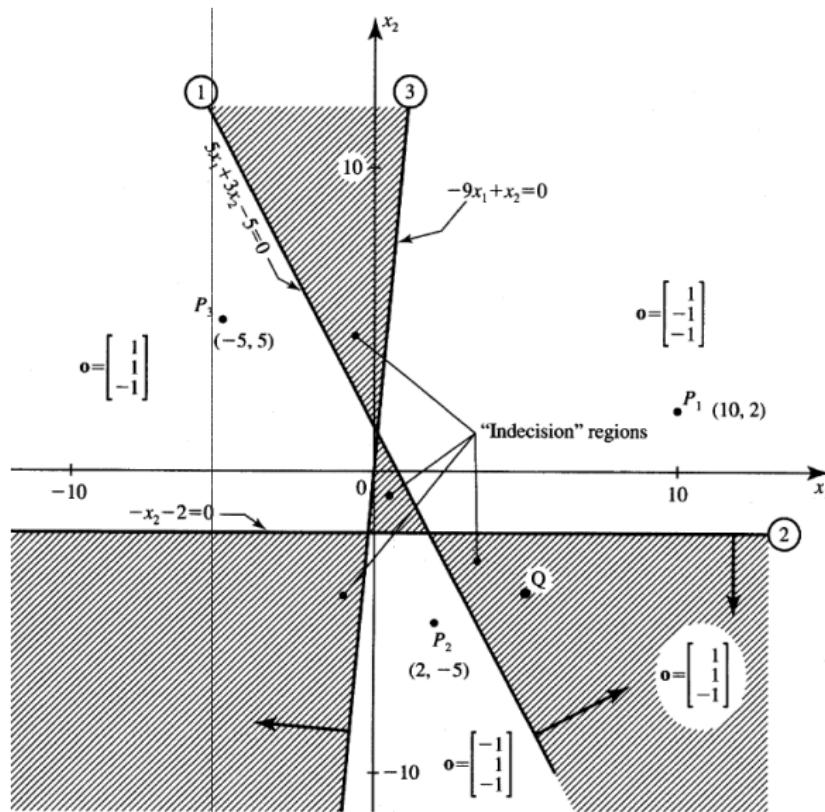
$$\text{sgn}(w \cdot \xi^\mu) = \zeta^\mu$$

que es equivalente a clasificar los patrones mediante un hiperplano.

Ejemplo perceptrón de 3 categorías



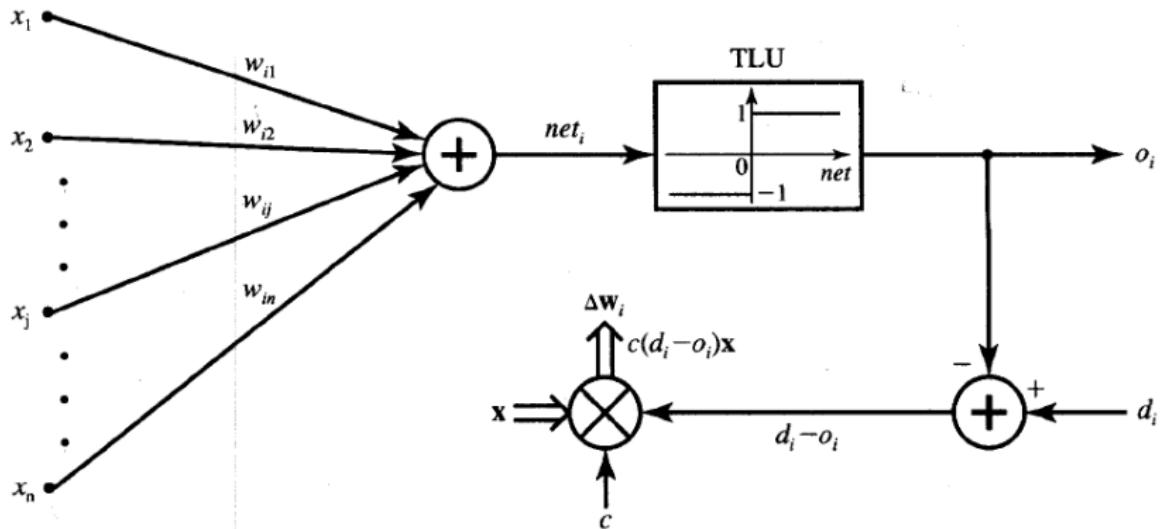
Regiones de decisión perceptrón 3 categorías



Pseudocódigo super-simplificado del perceptrón simple

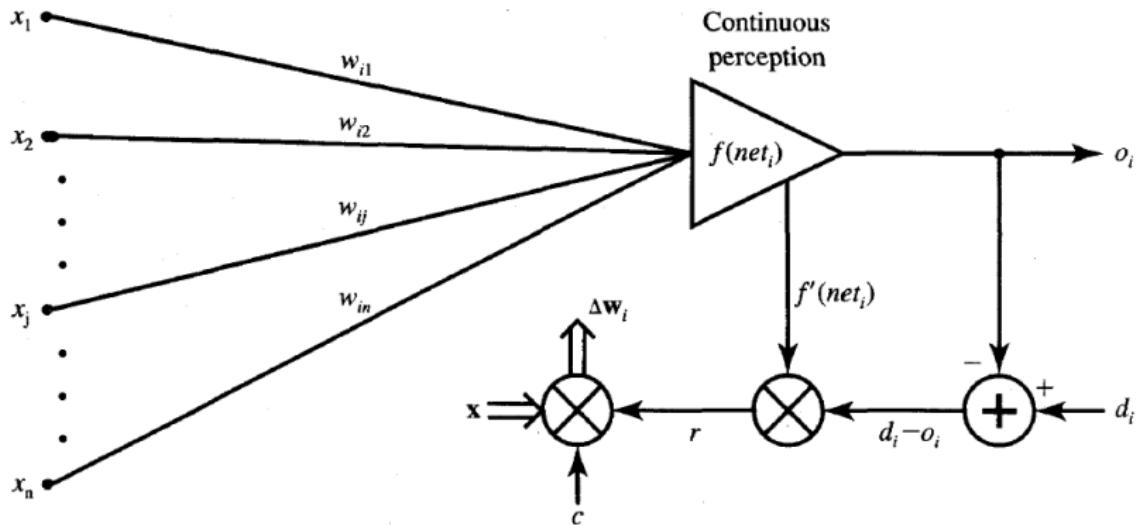
- 1) *Inicialización.* Inicializar pesos de la red
- 2) *Entrenamiento.* Mientras no se cumpla condición de corte, hacer:
 - 2.a) *Activación.* Elegir un patrón del conjunto de entrenamiento.
 - 2.b) *Cómputo de la respuesta actual.* Calcular en forma forward el output de la red neuronal.
 - 2.c) *Adaptación de los pesos.* Calcular el error de la red con respecto a la salida esperada, y actualizar los pesos de la red.

Regla de aprendizaje del Perceptrón Simple Escalón



$$\Delta \mathbf{w}_i = c [d_i - \text{sgn}(\mathbf{w}_i^T \mathbf{x})] \mathbf{x}$$

Regla de aprendizaje del Perceptrón Simple Continuo



$$\Delta \mathbf{w}_i = c(d_i - o_i)f'(\text{net}_i)\mathbf{x}$$

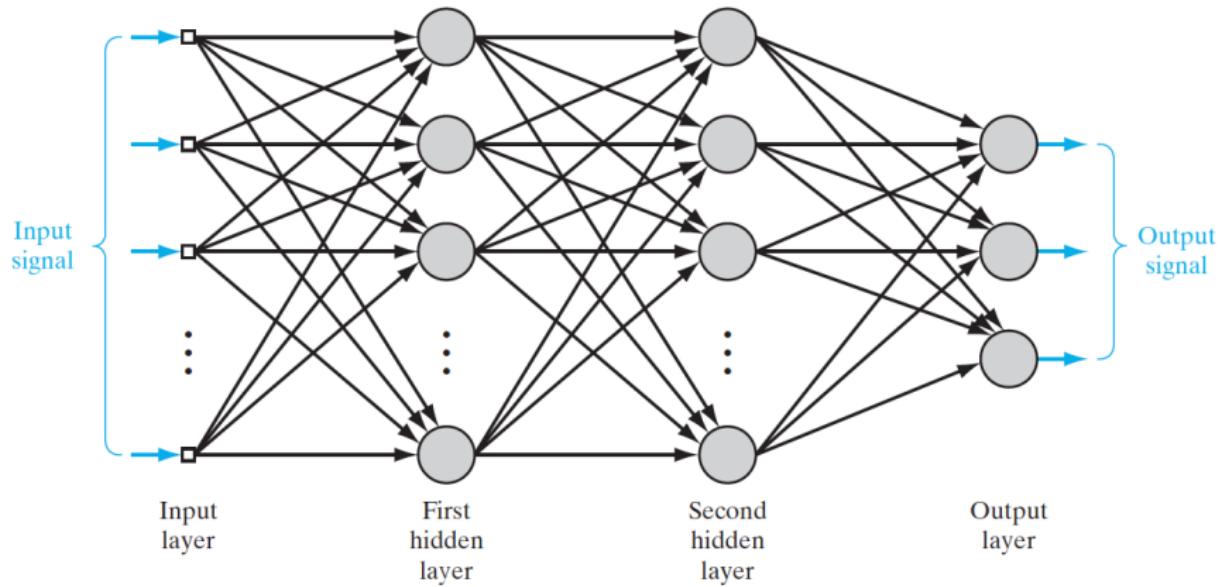
¿Cómo aprende el perceptrón simple?

La regla delta depende de la función de activación del perceptrón simple:

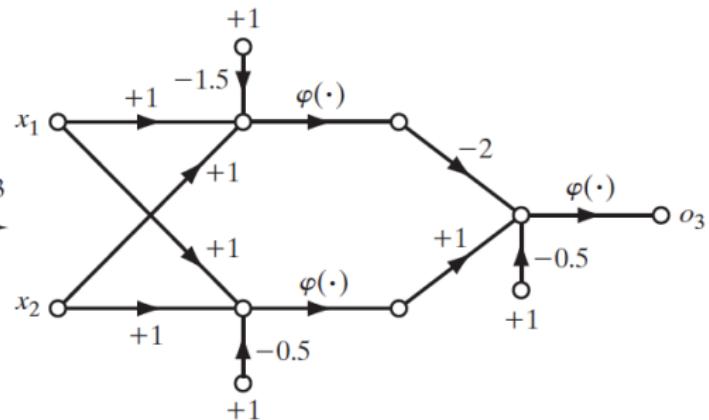
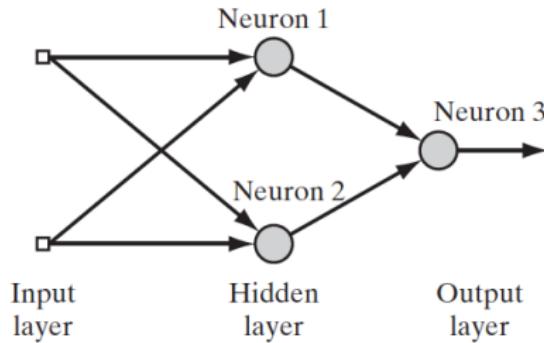
- ▷ *Función activación escalón*: aprendizaje hebbiano.
- ▷ *Función de activación diferenciable*: minimización de la función de energía mediante algún algoritmo de optimización.

- Repaso de perceptrón simple*
- Introducción a las redes neuronales feedforward multicapa*
- Algoritmo de Backpropagation*
- Conclusiones*

Red neuronal feedforward multicapa



Red neuronal feedforward multicapa: XOR



Poder de las redes feedforward multicapa

Las redes feedforward multicapa, con funciones de activación que cumplan ciertas condiciones (ver teorema), son aproximadores universales de funciones medibles de Borel (las funciones continuas en un subconjunto cerrado y acotado de R^n son de Borel). Las funciones de activación sigmoideas cumplen las condiciones del teorema.

Teorema de la Aproximación Universal (Cybenko, 1988; Hornik et al., 1989; Funahashi, 1989)

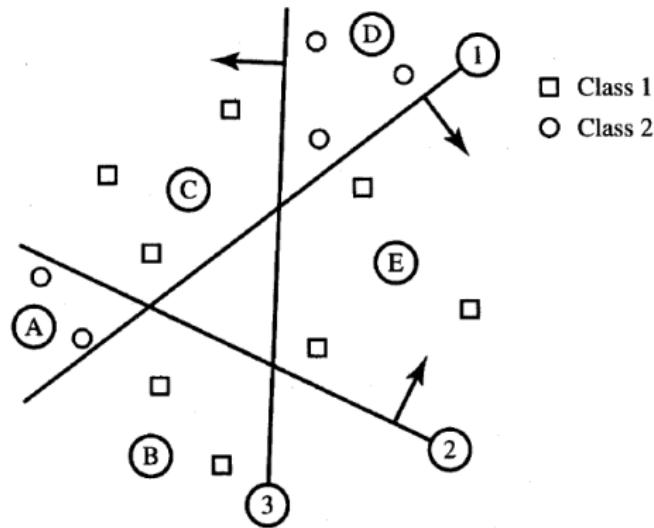
Una red feedforward multicapa, con una capa de salida con función de activación lineal, y al menos una capa oculta con funciones de activación continuas, no constantes, acotadas, y monotonamente crecientes, puede aproximar cualquier función medible de Borel, de un espacio de dimensión finita a otro con cualquier grado de error distinto de cero que se desee, si tiene la suficiente cantidad de neuronas ocultas.

Poder de las redes feedforward multicapa (cont.)

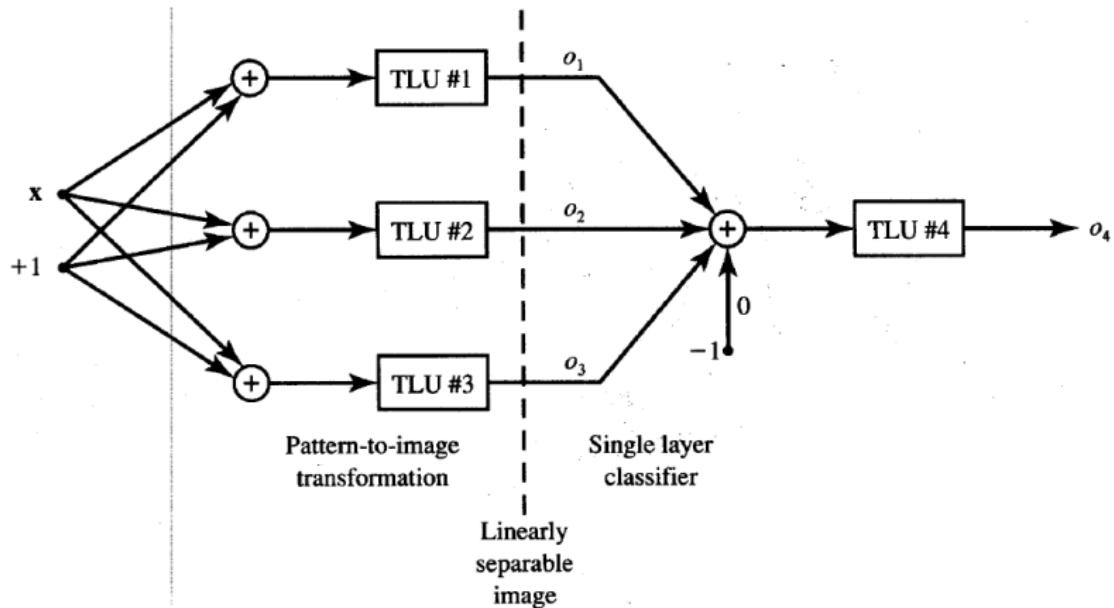
Atención: El Teorema de la Aproximación Universal es un teorema de existencia!

El Teorema de la Aproximación Universal, nos dice sólo que **existe** una red neuronal feed-forward de una sola capa oculta, que puede aproximar cualquier función continua en un subconjunto cerrado y acotado de R^n , pero no nos dice la cantidad de neuronas ocultas que debe tener, ni cómo obtener los pesos de dicha red neuronal.

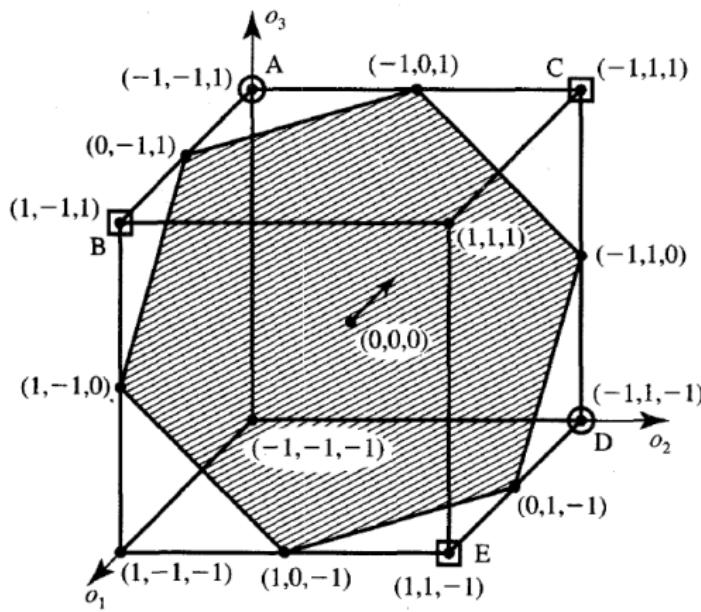
Viendo cómo funcionan: desde el espacio de los patrones...



Viendo cómo funcionan: ...transforma...



Viendo cómo funcionan: ... al espacio de las imágenes linealmente separable!



- ☒ *Repaso de perceptrón simple*
- ☒ *Introducción a las redes neuronales feedforward multicapa*
- ☐ *Algoritmo de Backpropagation*
- ☐ *Conclusiones*

Resurgimiento de las redes neuronales artificiales

NATURE VOL. 323 9 OCTOBER 1986

LETTERS TO NATURE

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California,
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,
Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure¹.

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the

more difficult when we introduce hidden units whose actual desired states are not specified by the task. (In perceptrons there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what the units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; a number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to all connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

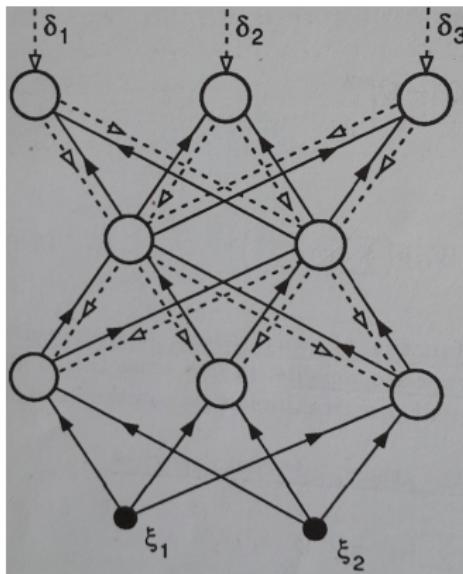
The total input, x_j , to unit j is a linear function of the output, y_i , of the units that are connected to j and of the weights, w_{ji} , on these connections

$$x_j = \sum_i y_i w_{ji}$$

Algoritmo de Backpropagation

La regla de aprendizaje está basada en minimizar la función de costo $E(w)$, siguiendo la dirección del gradiente descendente.

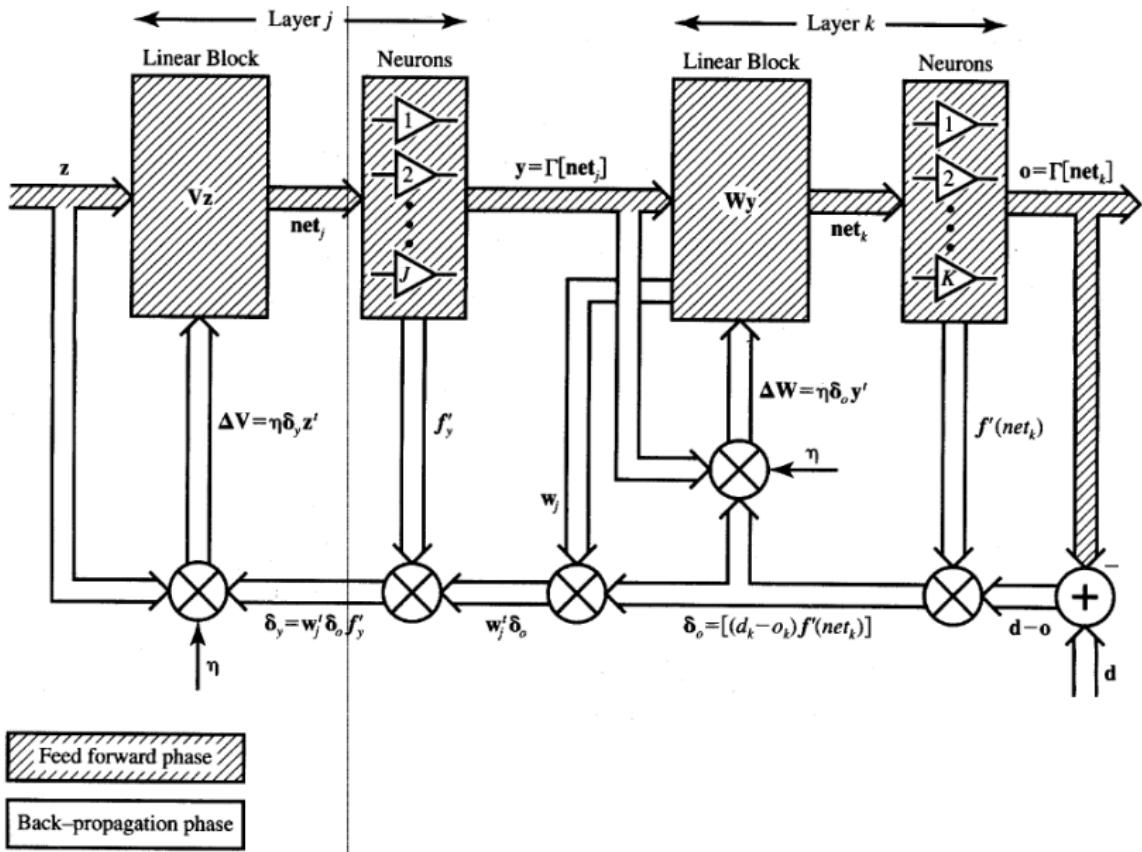
1. *Etapa forward:* Calcula el output de la red neuronal.
2. *Etapa backward:* Calcula el gradiente de $E(w)$ con respecto a los pesos de la red, y actualiza los pesos.



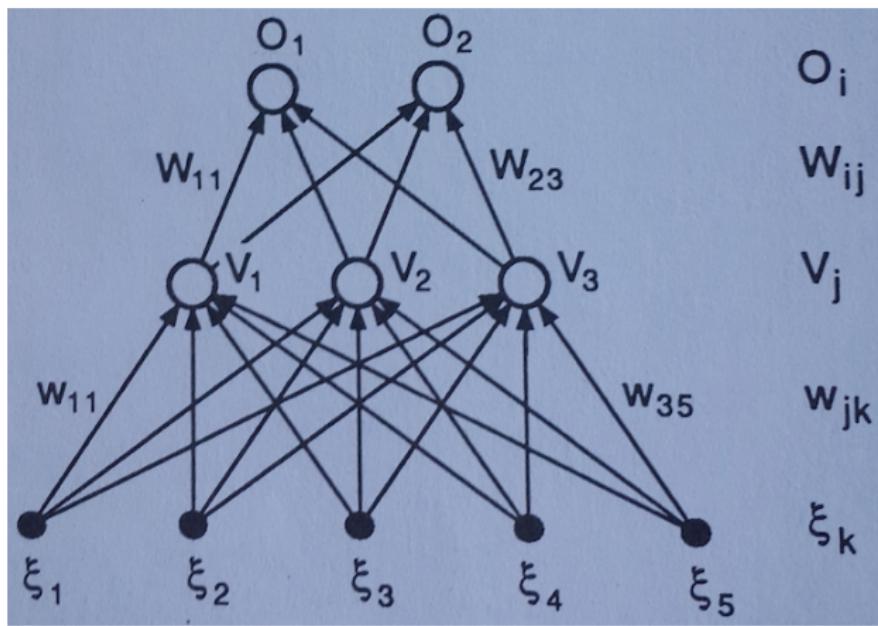
Pseudocódigo super-simplificado de la red neuronal feedforward multicapa

- 1) *Inicialización.* Inicializar pesos de la red
- 2) *Entrenamiento.* Mientras no se cumpla condición de corte, hacer:
 - 2.a) *Activación.* Elegir un patrón del conjunto de entrenamiento.
 - 2.b) *Cómputo de la respuesta actual.* Calcular en forma forward el output de la red neuronal.
 - 2.c) *Cómputo del error.* Calcular el error de la red con respecto a la salida esperada.
 - 2.d) *Cómputo de los deltas.* Calcular los deltas de la red en forma backward.
 - 2.e) *Adaptación de los pesos.* Actualizar los pesos de la red.

Algoritmo de Backpropagation (diagrama)



Algoritmo de Backpropagation



Algoritmo de Backpropagation

1. Initialize the weights to small random values.
2. Choose a pattern ξ_k^μ and apply it to the input layer ($m = 0$) so that

$$V_k^0 = \xi_k^\mu \quad \text{for all } k. \quad (6.15)$$

3. Propagate the signal forwards through the network using

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right) \quad (6.16)$$

for each i and m until the final outputs V_i^M have all been calculated.

4. Compute the deltas for the output layer

$$\delta_i^M = g'(h_i^M)[\zeta_i^\mu - V_i^M] \quad (6.17)$$

by comparing the actual outputs V_i^M with the desired ones ζ_i^μ for the pattern μ being considered.

5. Compute the deltas for the preceding layers by propagating the errors backwards

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m \quad (6.18)$$

for $m = M, M-1, \dots, 2$ until a delta has been calculated for every unit.

6. Use

$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1} \quad (6.19)$$

to update all connections according to $w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \Delta w_{ij}$.

7. Go back to step 2 and repeat for the next pattern.

Algoritmo de Backpropagation (forward)

1. Initialize the weights to small random values.
2. Choose a pattern ξ_k^μ and apply it to the input layer ($m = 0$) so that

$$V_k^0 = \xi_k^\mu \quad \text{for all } k .$$

3. Propagate the signal forwards through the network using

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right)$$

for each i and m until the final outputs V_i^M have all been calculated.

Algoritmo de Backpropagation (backward)

4. Compute the deltas for the output layer

$$\delta_i^M = g'(h_i^M)[\zeta_i^\mu - V_i^M] \quad (6.17)$$

by comparing the actual outputs V_i^M with the desired ones ζ_i^μ for the pattern μ being considered.

5. Compute the deltas for the preceding layers by propagating the errors backwards

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m \quad (6.18)$$

for $m = M, M-1, \dots, 2$ until a delta has been calculated for every unit.

6. Use

$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1} \quad (6.19)$$

to update all connections according to $w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \Delta w_{ij}$.

7. Go back to step 2 and repeat for the next pattern.

Algoritmo de Backpropagation: las fórmulas "nuevas"

$$\delta_i^M = g'(h_i^M)[\zeta_i^\mu - V_i^M]$$

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m$$

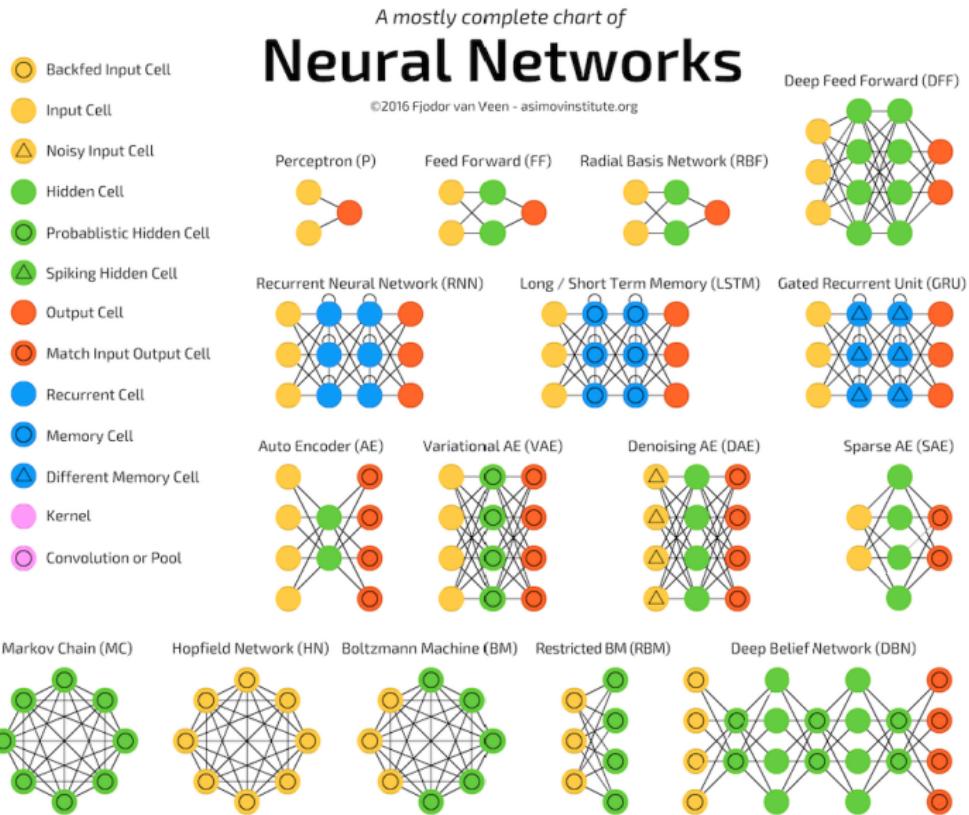
$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1}$$

- ☒ *Repaso de perceptrón simple*
- ☒ *Introducción a las redes neuronales feedforward multicapa*
- ☒ *Algoritmo de Backpropagation*
- ☐ *Conclusiones*

Bloques elementales de construcción de una red neuronal artificial

1. *Paradigma de aprendizaje*
2. *Estructura de la red neuronal*
3. *Función/es de activación*
4. *Algoritmo de aprendizaje de los pesos de la red*

Epílogo: The Neural Network Zoo¹



¹<http://www.asimovinstitute.org/neural-network-zoo/>

Bibliografía

Referencias

- Abu-Mostafa et al, *Learning from Data*, 2012.
- Haykin, *Neural Networks and Learning Machines*, 2009.
- Hertz, *Introduction to the theory of neural computation*, 1991.
- Zurada, *Introduction to artificial neural systems*, 1992.