Redes Neuronales Artificiales

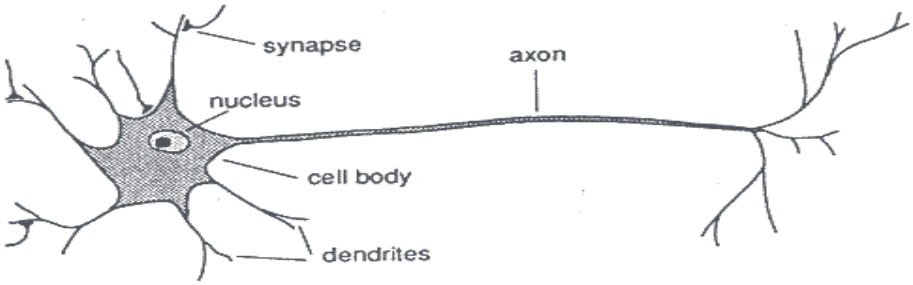
Introducción y aplicaciones en Bioinformática

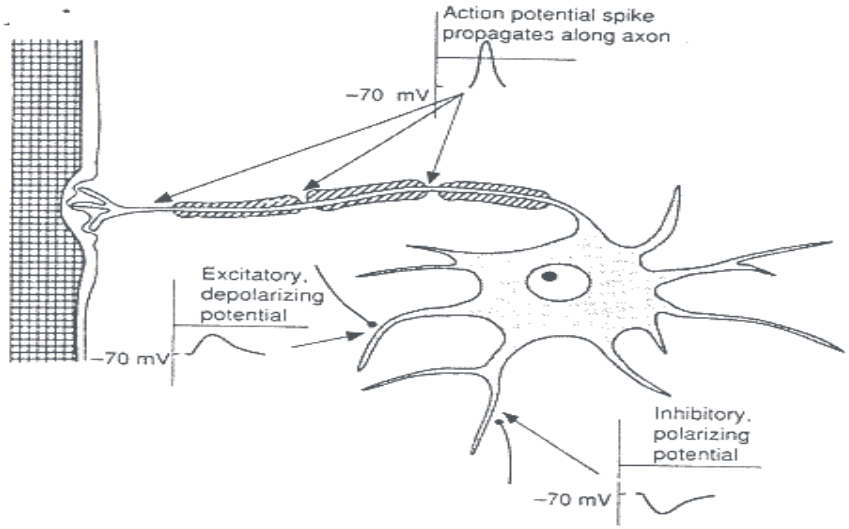
Algunos enlaces de interés

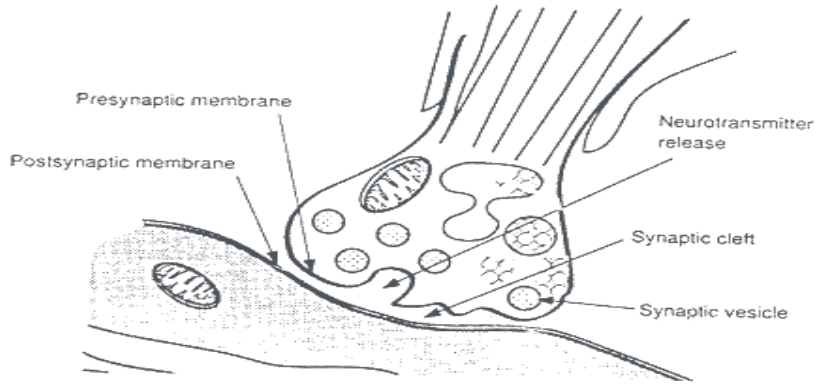
- http://www.gc.ssr.upm.es/inves/neural/ann2/anntutor.htm
- http://www.ee.umd.edu/medlab/neural/nn1.html
- http://math.chtf.stuba.sk/Books_texts_ANN.htm
- http://en.wikipedia.org/wiki/Artificial_neural_network/

Federico Morán. Julio 2007

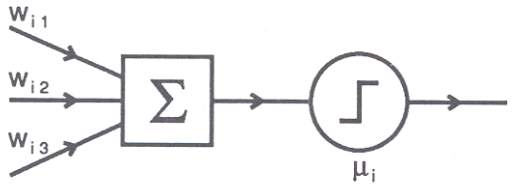
La neurona biológica

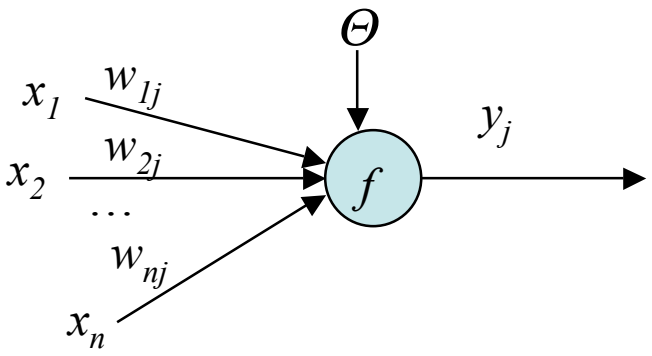






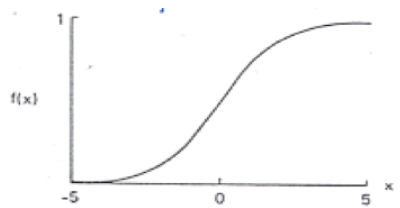
Neurona formal (McCulloch-Pitts, 1943)

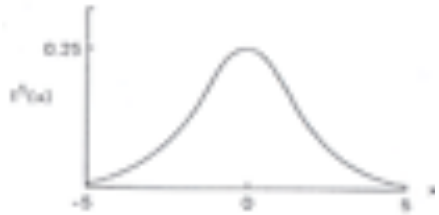




Binaria: $I_j = \sum_i w_{ij} x_i \rightarrow y_j = \begin{cases} 0 \text{ si } I_j < \Theta \\ 1 \text{ si } I_j \geq \Theta \end{cases}$

Valores reales: $I_j = \sum_i w_{ij} x_i - \Theta \rightarrow y_j = f(I_j)$ siendo f { función escalón, sigmoide, gaussiana, ... }



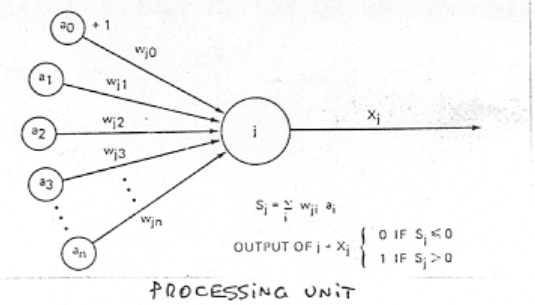


Historia de las Redes Neuronales Artificiales

(Primera Epoca)

AÑO	AUTORES	CONTRIBUCIÓN
1943	W. McCulloch, W. Pitts	Neurona formal
1949	Donald Hebb	Aprendizaje hebbiano
1950	N. Wiener	Cybernetics
1951 y 56	J. von Neuman	Automata y computacion neuronal. Redundancia
1951	M. Minsky	Neurocomputer
1957 y 62	F. Rosenblatt	Regla delta y Perceptron
1962	B. Widraw	ADALINE
1963	Winograd y Cowan	Proceso distribuido
1969	M. Minsky y S. Papert	Critica al perceptron

Perceptrón



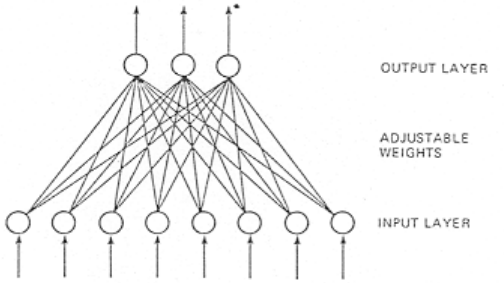


Figure 2-2. A two-layer perceptron.

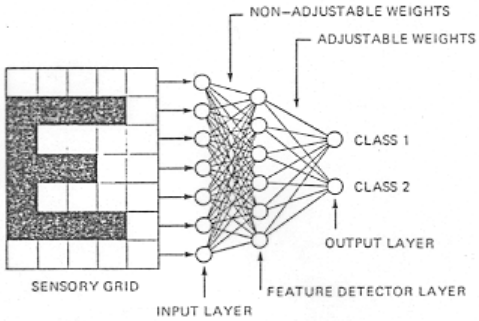


Figure 2-3. A three-layer perceptron.

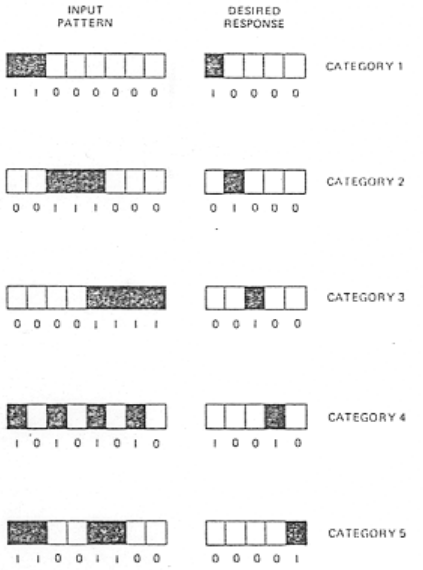


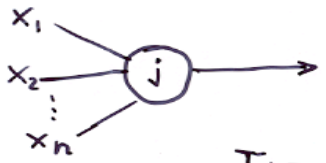
Figure 2-4. Example training set, with desired responses.

Aprendizaje

- **Supervisado**: se conoce la respuesta esperada y se puede evaluar el grado de acierto
  - Refuerzo positivo/negativo
  - Cálculo del error (regla delta)
- **No supervisado**: no se conoce si la respuesta es acertada o no; no se conoce el tipo de respuesta (clasificación)
  - Autoorganizado
  - Hebbiano

ENTRENAMIENTO <-> UTILIZACIÓN
red neuronal

Primeras redes adaptativas: El perceptrón original (Rosemblatt, 1959)



$$I_j = \sum_i w_{ij} x_i$$

$$y_j = \begin{cases} +1 & \text{si } I_j \geq \theta \\ -1 & \text{si } I_j < \theta \end{cases}$$

PRIMERA REGLA DE APRENDIZAJE (ROSENBLATT)

$$w_{ij}(\text{nuevo}) = w_{ij}(\text{viejo}) + \beta \cdot y_j \cdot x_i$$

$$\beta = \begin{cases} +1 & \text{respuesta correcta} \\ -1 & \text{" incorrecta} \end{cases} \quad \left\{ \text{Refuerzo o "crítico"} \right.$$

Procedimiento:

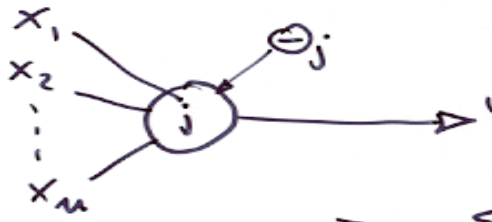
- Se presenta la muestra sucesivas veces y se van modificando los pesos hasta llegar a una situación estacionaria o resolver el problema.
- El estado final depende de los valores iniciales de los pesos. ⇒ mínimos locales.

OTRAS REGLAS DE APRENDIZAJE:

* TIPO HEBB: β = CONSTANTE
* CONVERGENTE

$$\beta = \begin{cases} = 0 & \text{si respuesta correcta} \\ \text{signo}(y_j \cdot x_i) \cdot 1 & \text{si incorrecta} \end{cases}$$

Primeras redes adaptativas: regla delta



t_j : respuesta deseada

$$I_j = \sum w_{ij} x_i$$

$$y_i = \begin{cases} +1 & \text{si } I_j \geq \theta_j \\ 0 & \text{si } I_j < \theta_j \end{cases}$$

$$w_{ij}(\text{nuevo}) = w_{ij}(\text{viejo}) + \beta\,(t_j - y_j)\cdot x_i$$

β: constante $(0 < \beta \leq 1)$ de aprendizaje

$$\delta_j = (t_j - y_j) = \begin{cases} 0 & \text{si } t_j = y_j \\ 1 & t_j = 1,\ y_j = 0 \\ -1 & t_j = 0,\ y_j = 1 \end{cases} \quad \text{(red binaria)}$$

$x_i = 1, 0$: el valor de entrada desde la neurona i.

Rosenblatt en 1962 probó que con esta regla el error siempre disminuye.

Demostración de la disminución del error en la regla delta

To be more specific, then, let

$$E_p = \frac{1}{2}\sum_j (t_{pj} - o_{pj})^2 \qquad (2)$$

be our measure of the error on input/output pattern p *and let* $E = \sum E_p$ *be our overall measure of the error. We wish to show that the delta rule implements a gradient descent in* E *when the units are linear. We will proceed by simply showing that*

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} i_{pi},$$

which is proportional to $\Delta_p w_{ji}$ *as prescribed by the delta rule. When there are no hidden units it is straightforward to compute the relevant derivative. For this purpose we use the chain rule to write the derivative as the product of two parts: the derivative of the error with respect to the output of the unit times the derivative of the output with respect to the weight.*

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}. \qquad (3)$$

The first part tells how the error changes with the output of the j*th unit and the second part tells how much changing* w_{ji} *changes that output. Now, the derivatives are easy to compute. First, from Equation 2*

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) = -\delta_{pj}. \qquad (4)$$

Not surprisingly, the contribution of unit u_j *to the error is simply proportional to* δ_{pj}*. Moreover, since we have linear units,*

$$o_{pj} = \sum_i w_{ji} i_{pi}, \qquad (5)$$

from which we conclude that

$$\frac{\partial o_{pj}}{\partial w_{ji}} = i_{pi}.$$

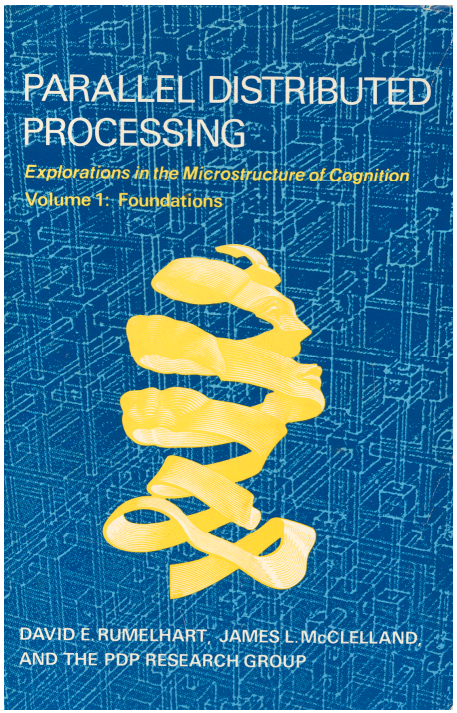
Thus, substituting back into Equation 3, we see that

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} i_{pi} \qquad (6)$$

as desired. Now, combining this with the observation that

$$\frac{\partial E}{\partial w_{ji}} = \sum_p \frac{\partial E_p}{\partial w_{ji}}$$

should lead us to conclude that the net change in w_{ji} *after one complete cycle of pattern presentations is proportional to this derivative and hence that the delta rule implements a gradient descent in* E*. In fact, this is strictly true only if the values of the weights are not changed during this cycle. By changing the weights after each pattern is presented we depart to some extent from a true gradient descent in* E*. Nevertheless, provided the learning rate (i.e., the constant of proportionality) is sufficiently small, this departure will be negligible and the delta rule will implement a very close approximation to gradient descent in sum-squared error. In particular, with small enough learning rate, the delta rule will find a set of weights minimizing this error function.*



PERCEPTRON

$(t_{jp} - x_{jp})$

t_{jp} = the target value for output unit j after presentation of pattern p

x_{jp} = the output value produced by output unit j after presentation of pattern p

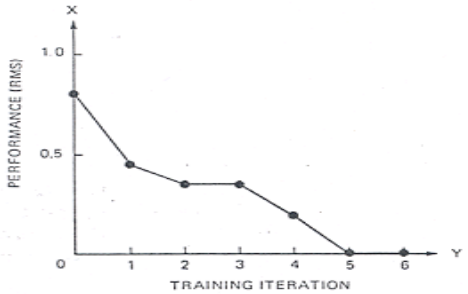
C = a small constant (the "learning rate")

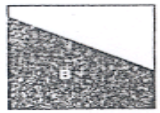
$$w_{ji}_{\text{new}} = w_{ji}_{\text{old}} + C\,(t_j - x_j)\,a_i$$

$$(t_j - x_j) = \begin{cases} 1 \text{ if } t_j \text{ is 1 and } x_j \text{ is 0} \\ 0 \text{ if } t_j = x_j \\ -1 \text{ if } t_j \text{ is 0 and } x_j \text{ is 1} \end{cases}$$

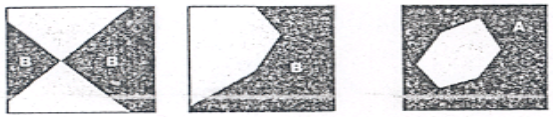
a_i = 1 or 0, the value of input unit i

$$\sqrt{\frac{\Sigma_p \Sigma_j (t_{jp} - x_{jp})^2}{n_p n_o}}$$



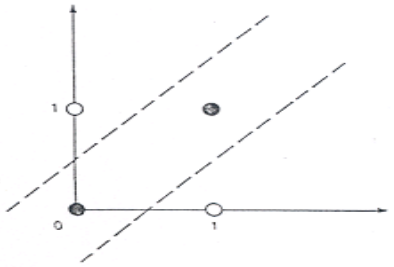


LINEARLY SEPARABLE REGIONS



NOT LINEARLY SEPARABLE BY ONE LINE

INPUT		Xor
0	0	0
0	1	1
1	0	1
1	1	0



Historia de las Redes Neuronales Artificiales

(Epoca Intermedia)

AÑO	AUTORES	CONTRIBUCIÓN
1971	T. Kohonen	Memoria asociativa
1973	Ch. von der Maslsburg	Columnas de Orientacion
1975	S.I. Amari	Aprendizaje competitivo
1976	Willsaw y von der Malsburg	Retinotopia
1977	J. Anderson	Autoasociación
1977 y 79	K. Fukushima	Cognitron y neocognitron
1978	S. Grossberg	Autoorganizacion de redes
1982	T. Kohonen	Self Organizing Map (SOM)
1982 y 84	J. Hopfield	Redes resonantes o de Hopfield
1985	Ackley, Hinton, Sejnowski	Maquina de Boltzman

Redes Resonantes o Redes de Hopfield

* ASOCIATIVE MEMORY
* COMPUTATIONAL DEVICE

BINARY HOPFIELD NET

- NETWORK STATE : $U = (U_1, U_2, \dots U_n)$.
  $(1, 0, 0, \dots)$

  $U = U(t)$

- CHARACTERISTICS

  FULLY CONNECTED (TWO DIRECTIONS)

  STRENGTH MATRIX SYMETRIC $\begin{cases} T_{ij} = T_{ji} \\ T_{ii} = 0 \end{cases}$

  RECURSIVE ⇨ RELAXATION TOWARDS STABLE STATE

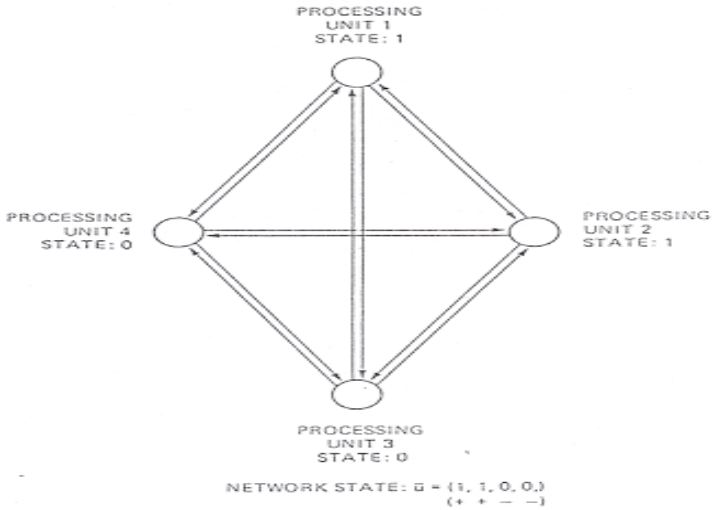
- ACTOALIZATION STATE RULE

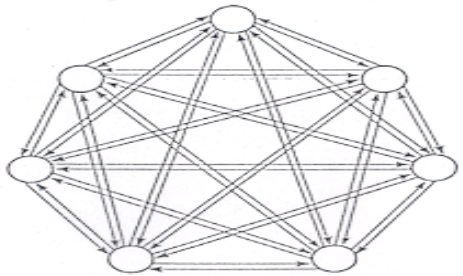
$$S_j = \sum_{i=1}^{n} U_i T_{ji} \begin{cases} \geq 0 \rightarrow U_j = 1 \\ < 0 \rightarrow U_j = 0 \end{cases} \begin{cases} = 1 \\ = -1 \end{cases}$$

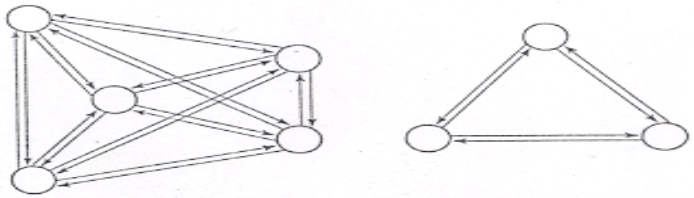
NOTE : T_{ij} = constant ⇨ no learning

$U_i \neq$ constant ⇨ "memory" iff $U_i(t) = U_i(t+1)$ $\forall i \; i = 1 \dots n$

Ejemplos de redes de Hopfield







Aplicaciones de redes de Hopfield

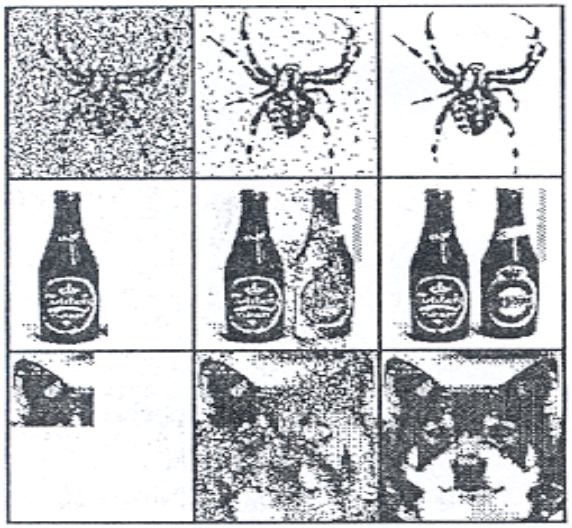


FIGURE 2.1 Example of how an associative memory can reconstruct images. These are binary images with 130 × 180 pixels. The images on the right were recalled by the memory after presentation of the corrupted images shown on the left. The middle column shows some intermediate states. A sparsely connected Hopfield network with seven stored images was used.

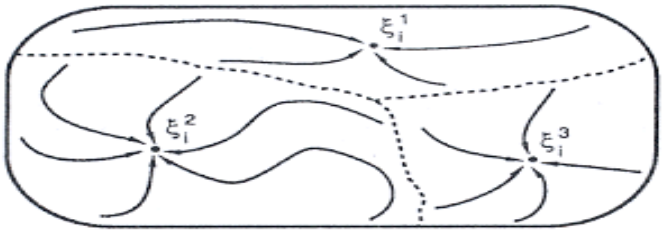


FIGURE 2.2 Schematic configuration space of a model with three attractors.

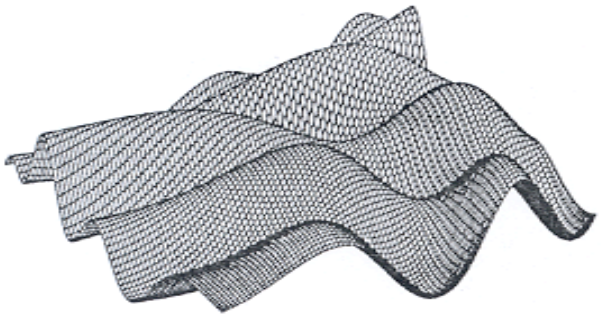
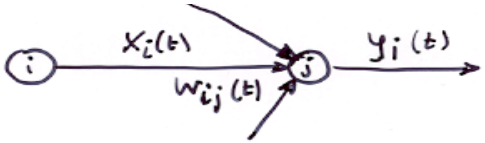


FIGURE 2.6 It is often useful (but sometimes dangerous) to think of the energy as something like this landscape. The z-axis is the energy and the 2^N corners of the hypercube (the possible states of the system) are formally represented by the x–y plane.

Aprendizaje no supervisado

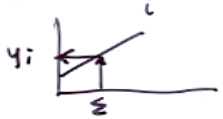
• La red presenta entrada/salida pero no hay retorno desde fuera
• La red debe “descubrir” por sí misma patrones, características, regularidades, correlaciones, etc. en la muestra o datos de entrada y codificarlos en la salida
• Tanto las unidades como las conexiones han de ser capaces de autoorganización (arquitecturas de enlaces dinámicos)
• Este tipo de redes son operativas cuando existe redundancia en la muestra: “la redundancia produce conocimiento” (Barlow, 1989)
• Tipos de tareas que se pueden realizar:
    -Modelización de procesos biológicos
    - Agrupación por familias
    - Análisis de componentes principales
    - Clasificación en clases o grupos (“*clustering*”)
    - Localización de prototipos
    - Codificación
    - Extracción de características “ocultas”
    - Organización topológica de datos

Aprendizaje Hebbiano



UNIDADES LINEALES:

$$y_j(t) = \sum_i w_{ij}(t) \cdot x_i(t) + \theta_j$$



(a) BÁSICO

$$w_{ij}(t+1) = w_{ij}(t) + \gamma \cdot y_j(t) \cdot x_i(t)$$

(b) SOPESADO

$$w_{ij}(t+1) = w_{ij}(t) + \gamma \cdot w_{ij}(t) \cdot y_j(t) \cdot x_i(t)$$

(c) NORMALIZADO

$$w_{ij}(t+1) = w_{ij}(t) + \gamma \cdot y_j(t) \left(x_i(t) - y_j(t) \cdot w_{ij}(t) \right)$$

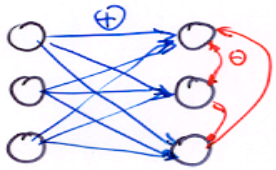
(d) ACOTADO.

REGLA BÁSICA, MANTENIENDO $w_{min} \leq w_{ij} \leq w_{max}$

(e) ANTI-HEBBIANO

PARA CONEXIONES LATERALES INHIBITORIAS

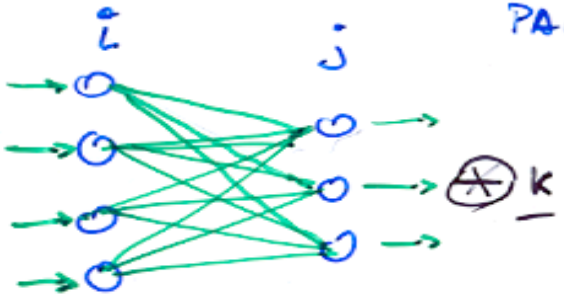
$$U_{ij}(t+1) = U_{ij}(t) - \gamma \cdot y_i(t) \cdot y_j(t)$$



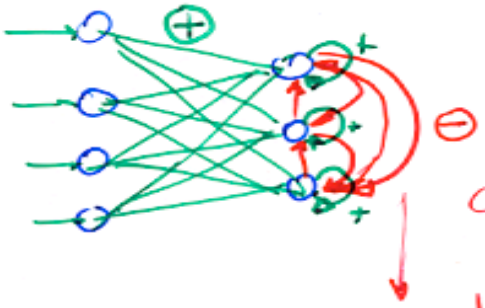
APRENDIZAJE COMPETITIVO NO SUPERVISADO

① "CLUSTERING" POR EL PROCESO "WINNER-TAKES-ALL"

[ESTE ALGORITMO ES SIMILAR A MÉTODOS CLÁSICOS COMO K-MEDIAS PARA MUESTRAS NORMALIZADAS]



$$a_j = \sum_i w_{ij} x_j$$



CONEXIONES INHIBITORIAS

$$w_{jj'} = \begin{cases} -\varepsilon & j \neq j' \\ 1 & j = j' \end{cases}$$

- SE ELIGE LA GANADORA: "k"

$$a_k > a_j \quad \forall\, j \neq k$$

- SE ASIGNAN:

$$\begin{cases} a_k = 1 \\ a_j = 0 \quad \forall\, j \neq k. \end{cases}$$

ESTE PROCESO CONVERGE A LA SELECCIÓN DE UNA ÚNICA NEURONA GANADORA.

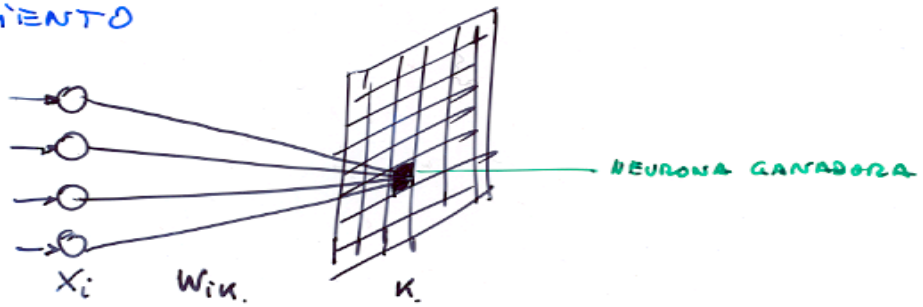
- SE MODIFICAN LOS PESOS DE LA NEURONA GANADORA

$$w_{ik}(t+1) = w_{ik}(t) + \gamma\,(x_i(t) - w_{ik}(t)) \qquad \forall\, i = 1 \dots n$$

ESTE ALGORITMO ACERCA EL VECTOR DE PESOS $\overline{w}_k$ AL VECTOR DE ENTRADA $\overline{x}$

Mapas topológicos o autoorganizativos de Kohonen (SOM)

- ESTA BASADO EN VQ CON LAS NEURONAS DE SALIDA ORGANIZADAS TOPOLÓGICAMENTE (NORMALMENTE EN MAPAS BIDIMENSIONALES)
- DATOS DE ENTRADA PRÓXIMOS → SE LOCALIZAN EN NEURONAS PRÓXIMAS
- INSPIRACIÓN BIOLÓGICA: MAPAS SOMATO SENSORIALES DEL CEREBRO.
- PROCEDIMIENTO



(a) SE PRESENTA LA MUESTRA x

(b) SE CALCULA LA NEURONA GANADORA, k, COMO LA DE DISTANCIA EUCLIDEA MINIMA. (= VQ) $\|\bar{w}_k - \bar{x}\| \leq \|\bar{w}_i - \bar{x}\| \quad \forall i$

(c) SE ADAPTAN LOS PESOS DE ESA NEURONA Y DE LAS VECINAS

$$w_v(t+1) = w_v(t) + \gamma \cdot h(v,k) \cdot (x(t) - w_v(t)) \qquad \forall v \in S$$

LA FUNCIÓN DE VECINDAD PUEDE SER:

(d) SE VA REDUCIENDO GRADUALMENTE LA VECINDAD HASTA LLEGAR A MODIFICAR SOLO UN VECTOR DE PESOS.

(e) SE TERMINA AL TÉRMINO DE UNA SERIE DE PASOS PREFIJADOS O CUANDO E NO VARÍA.

MAPAS DE KOHONEN (CONT.)

REPRESENTACIÓN DE LOS DATOS.

① Si N muestras > S número neuronas del mapa
- Una neurona puede ser activada por varias muestras
- Competición y clasificación por compresión (clustering)

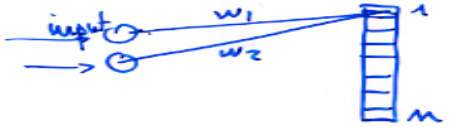
② Si N muestras < S número neuronas del mapa.

grupos de neuronas que responden mejor a una muestra

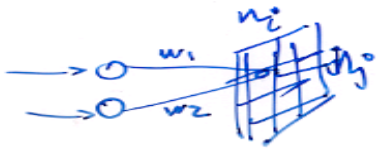
- Distribución en áreas.

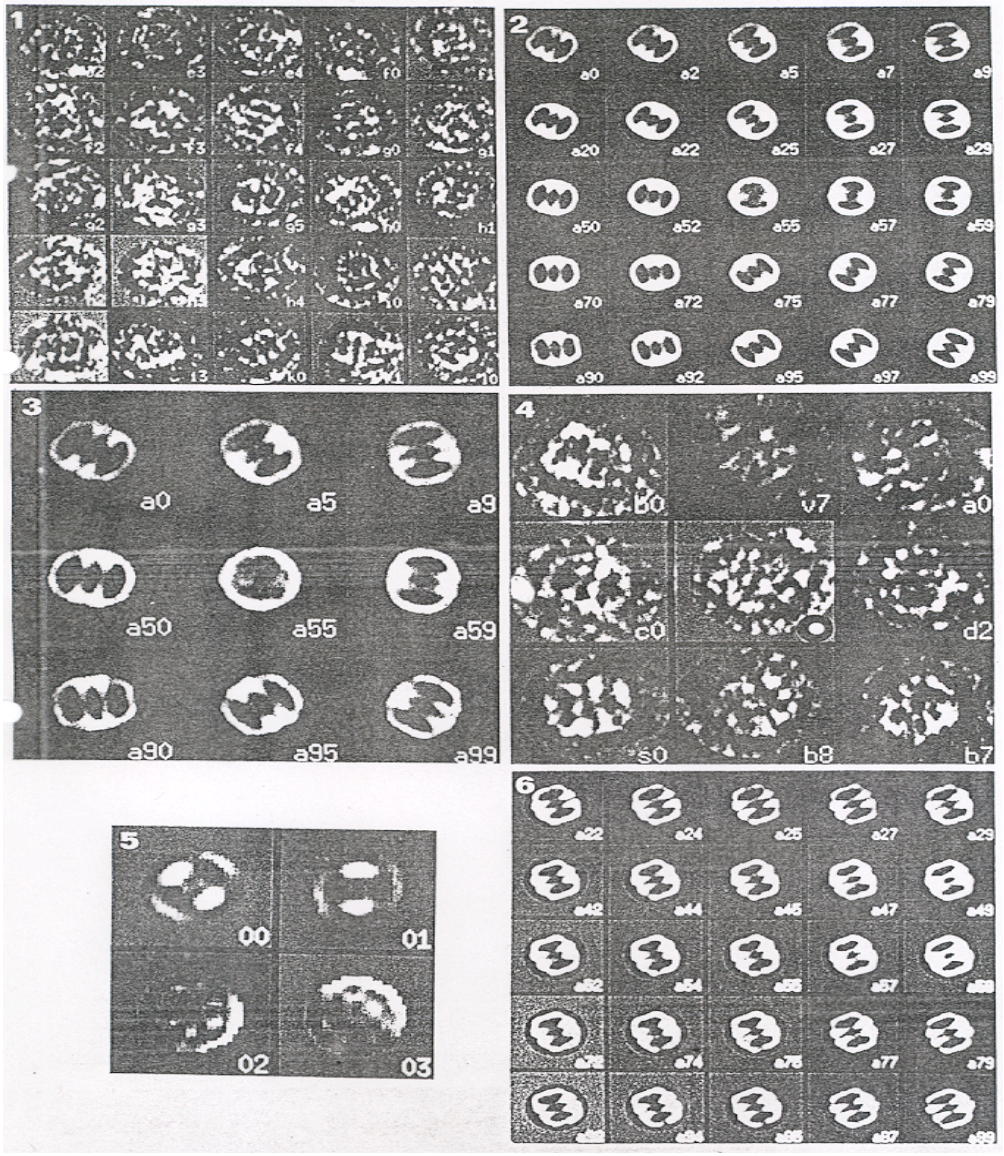
REPRESENTACIÓN DE LA CONVERGENCIA

(a) MAPA 1 DIMENSIÓN
VECTORES PESO 2 DIMENSIONES.



(b) MAPA 2 DIM. / VECTORES 2 DIM.




1
2
a0
a2
a5
a7
a9
a20
a22
a25
a27
a29
a50
a52
a55
a57
a59
a70
a72
a75
a77
a79
a90
a92
a95
a97
a99
3
a0
a5
a9
a50
a55
a59
a90
a95
a99
4
b0
v7
a0
c0
d2
s0
b8
b7
5
00
01
02
03
6
a22
a24
a25
a27
a29
a42
a44
a45
a47
a49
a54
a55
a57
a74
a75
a77
a79

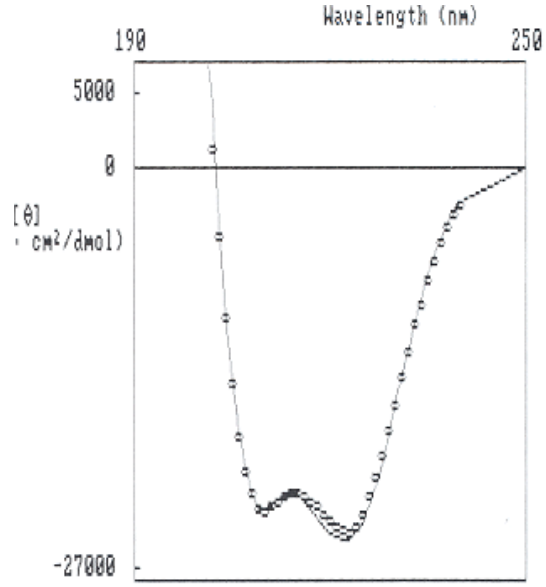
Cálculo de estructura secundaria de proteínas a partir de DC en UV lejano

CABIOS

Vol.4. no.4. 1988
Pages 479 – 482

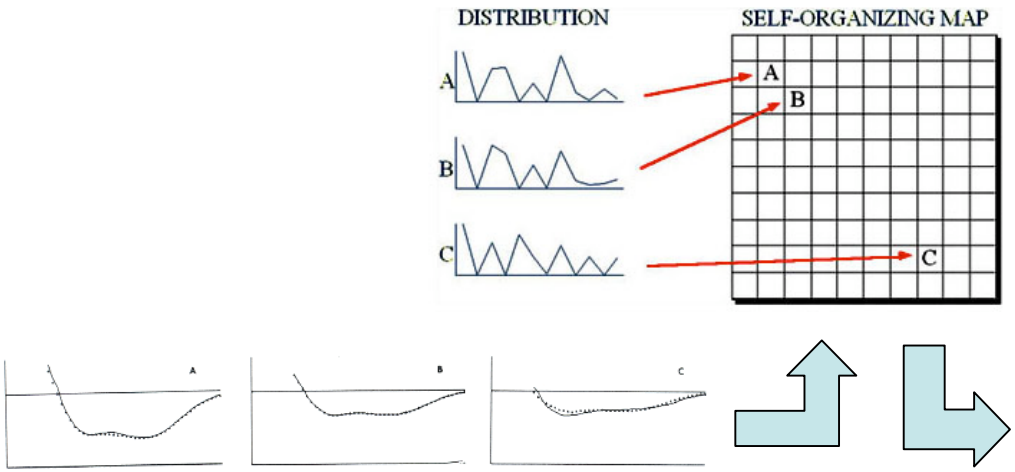
A BASIC microcomputer program to calculate the secondary structure of proteins from their circular dichroism spectrum

*Luis Menéndez-Arias, Julián Gómez-Gutiérrez, Miguel García-Ferrández[1], Alvaro García-Tejedor and Federico Morán**



Application of Vector Quantization Algorithms to Protein Classification and Secondary Structure Computation

J.J. Merelo(1), M. A. Andrade (2), C. Ureña (3), A. Prieto(1), F. Morán (2)



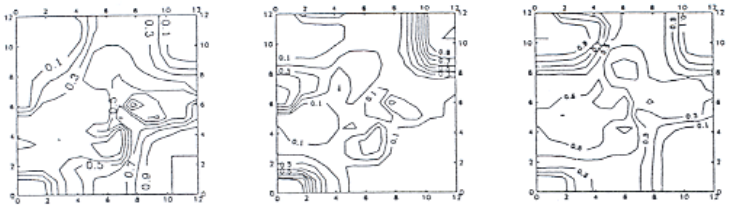
Neurocomputing 6 (1994) 443–454
Elsevier

443

NEUCOM 246

Proteinotopic feature maps

Juan J. Merelo [a, *], Miguel A. Andrade [b], Alberto Prieto [a] and Federico Morán [b]



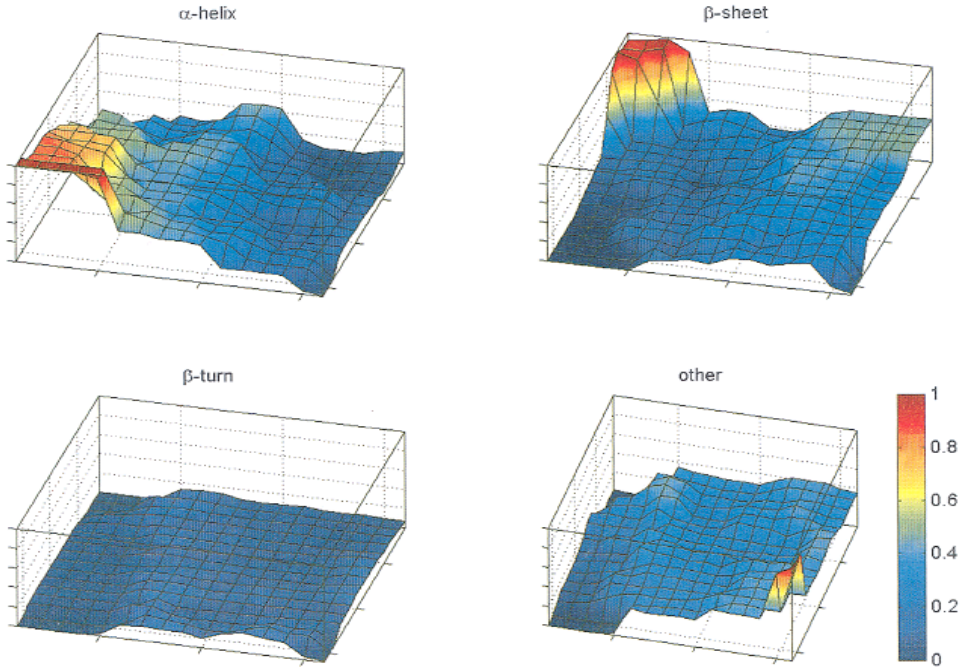
SOMCD

Cálculo de estructura secundaria de proteínas a partir de DC en UV lejano

Protein Engineering vol.6 no.4 pp.383–390, 1993

Evaluation of secondary structure of proteins from UV circular dichroism spectra using an unsupervised learning neural network

M.A.Andrade, P.Chacón, J.J.Merelo[1] and F.Morán



PROTEINS: Structure, Function, and Genetics 42:460–470 (2001)

SOMCD: Method for Evaluating Protein Secondary Structure from UV Circular Dichroism Spectra

Per Unneberg,[1] Juan J. Merelo,[2] Pablo Chacón,[3*] and Federico Morán[4*]

Historia de las Redes Neuronales Artificiales

(Segunda Epoca)

AÑO	AUTORES	CONTRIBUCIÓN
1986	Rumelhart, Hinton y Williams	Regla delta generalizada, backpropagation
1986	Rumelhart, McClelland y PDP-group	Paralell distributed processing
1986	Sejnowski y Rosenberg	NETtalk
1989	T. Kohonen	LVQ

Perceptron multicapa

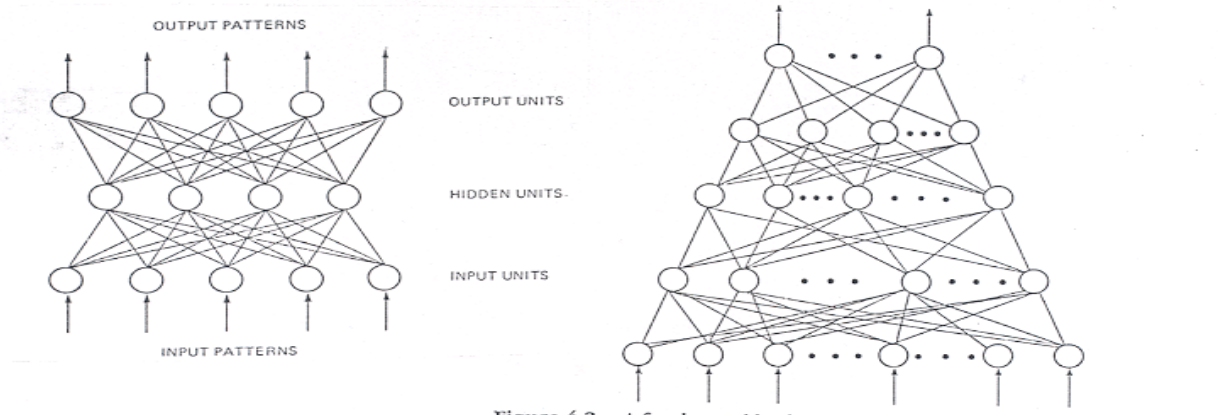


Figure 4-2. A five-layered back-propagation network, fully interconnected.

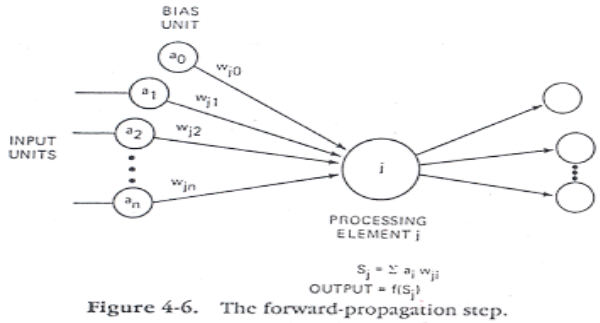
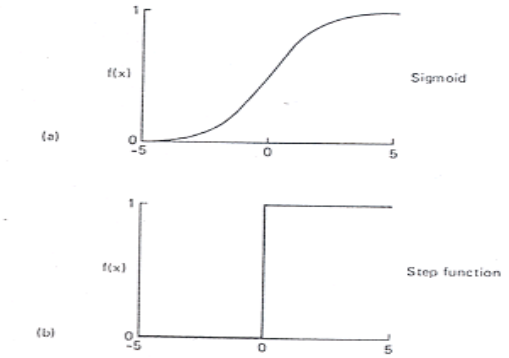
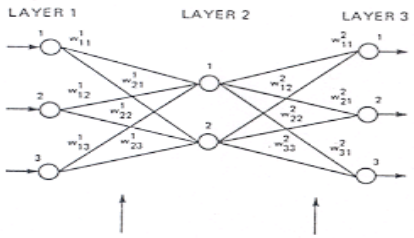
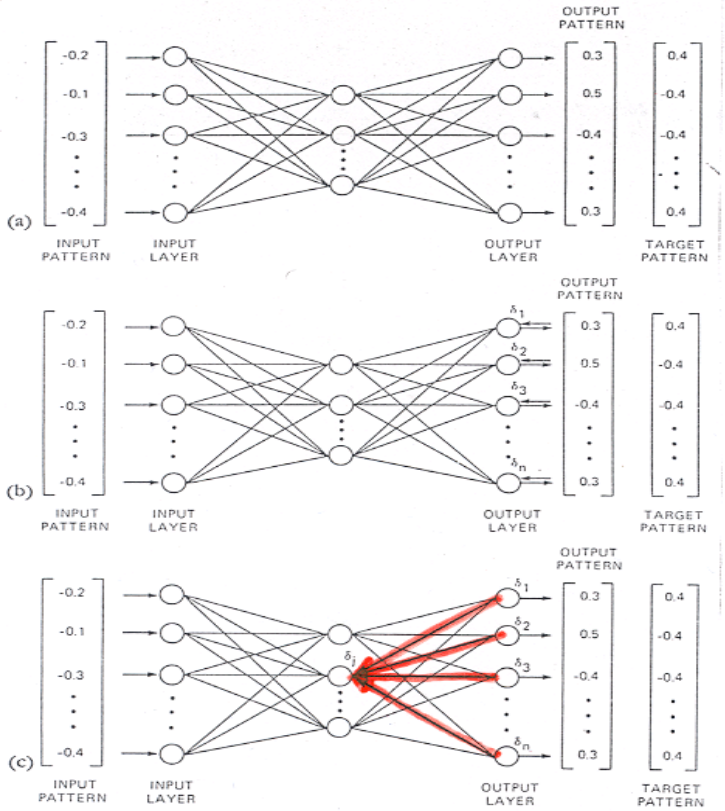


Figure 4-6. The forward-propagation step.





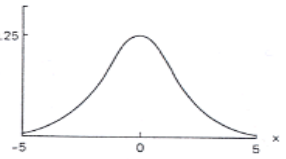
Propagación del error hacia atrás



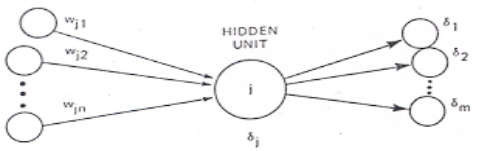
$\delta_j^o = (t_j - a_j) f'(S_j)$

t_j = the target value for unit j
a_j = the output value for unit j
$f'(x)$ = the derivative of the sigmoid function f
S_j = weighted sum of inputs to j

$f'(x) = f(x)(1 - f(x))$

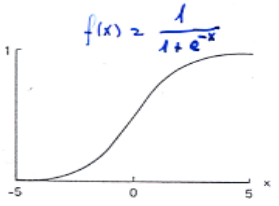


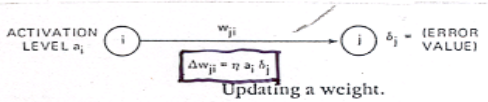
"DELTA RULE"



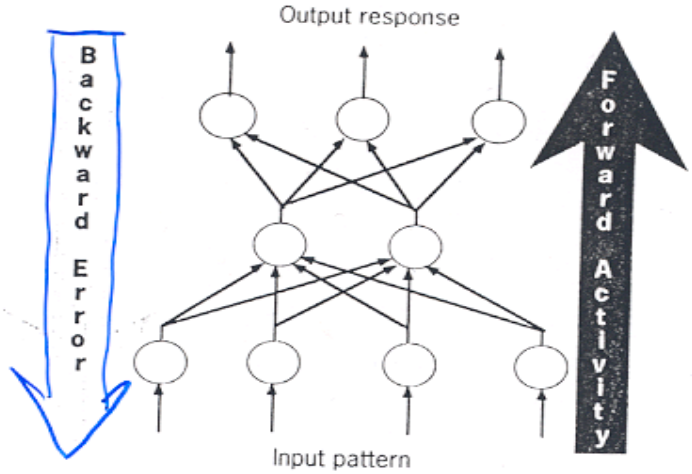
$\delta_j^h = \left[\sum_k \delta_k^o w_{kj}\right] f'(S_j)$

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

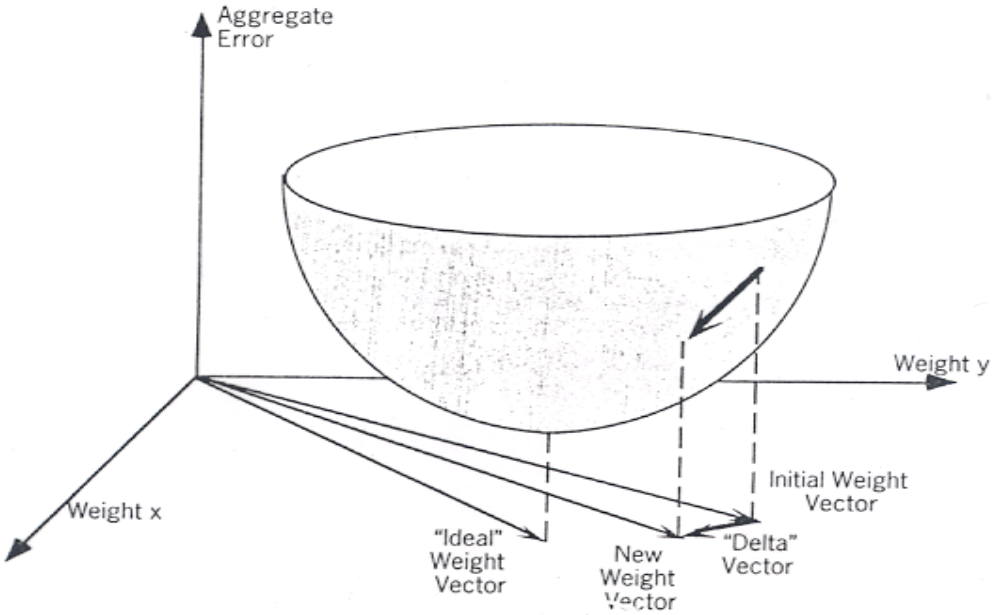




Updating a weight.



A backpropagation network trains with a two-step procedure. The activity from the input pattern flows forward through the network, and the error signal flows backward to adjust the weights.



The generalized delta rule is a gradient descent system.

Disminución del error

Demostración de la disminución del error en la regla delta generalizada

To get the correct generalization of the delta rule, we must set

$$\Delta_p w_{ji} \propto -\frac{\partial E_p}{\partial w_{ji}},$$

where E is the same sum-squared error function defined earlier. As in the standard delta rule it is again useful to see this derivative as resulting from the product of two parts: one part reflecting the change in error as a function of the change in the net input to the unit and one part representing the effect of changing a particular weight on the net input. Thus we can write

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}}. \tag{9}$$

By Equation 7 we see that the second factor is

$$\frac{\partial net_{pj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_k w_{jk} o_{pk} = o_{pi}. \tag{10}$$

Now let us define

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}}.$$

(By comparing this to Equation 4, note that this is consistent with the definition of δ_{pj} used in the original delta rule for linear units since $o_{pj} = net_{pj}$ when unit u_j is linear.) Equation 9 thus has the equivalent form

$$-\frac{\partial E_p}{\partial w_{ji}} = \delta_{pj} o_{pi}.$$

This says that to implement gradient descent in E we should make our weight changes according to

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi}, \tag{11}$$

just as in the standard delta rule. The trick is to figure out what δ_{pj} should be for each unit u_j in the network. The interesting result, which we now derive, is that there is a simple recursive computation of these δ's which can be implemented by propagating error signals backward through the network.

To compute $\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}}$, we apply the chain rule to write this partial derivative as the product of two factors, one factor reflecting the change in error as a function of the output of the unit and one reflecting the change in the output as a function of changes in the input. Thus, we have

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial net_{pj}}. \tag{12}$$

Let us compute the second factor. By Equation 8 we see that

$$\frac{\partial o_{pj}}{\partial net_{pj}} = f'_j(net_{pj}),$$

which is simply the derivative of the squashing function f_j for the jth unit, evaluated at the net input net_{pj} to that unit. To compute the first factor, we consider two cases. First, assume that unit u_j is an output unit of the network. In this case, it follows from the definition of E_p that

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}),$$

which is the same result as we obtained with the standard delta rule. Substituting for the two factors in Equation 12, we get

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj}) \tag{13}$$

for any output unit u_j. If u_j is not an output unit we use the chain rule to write

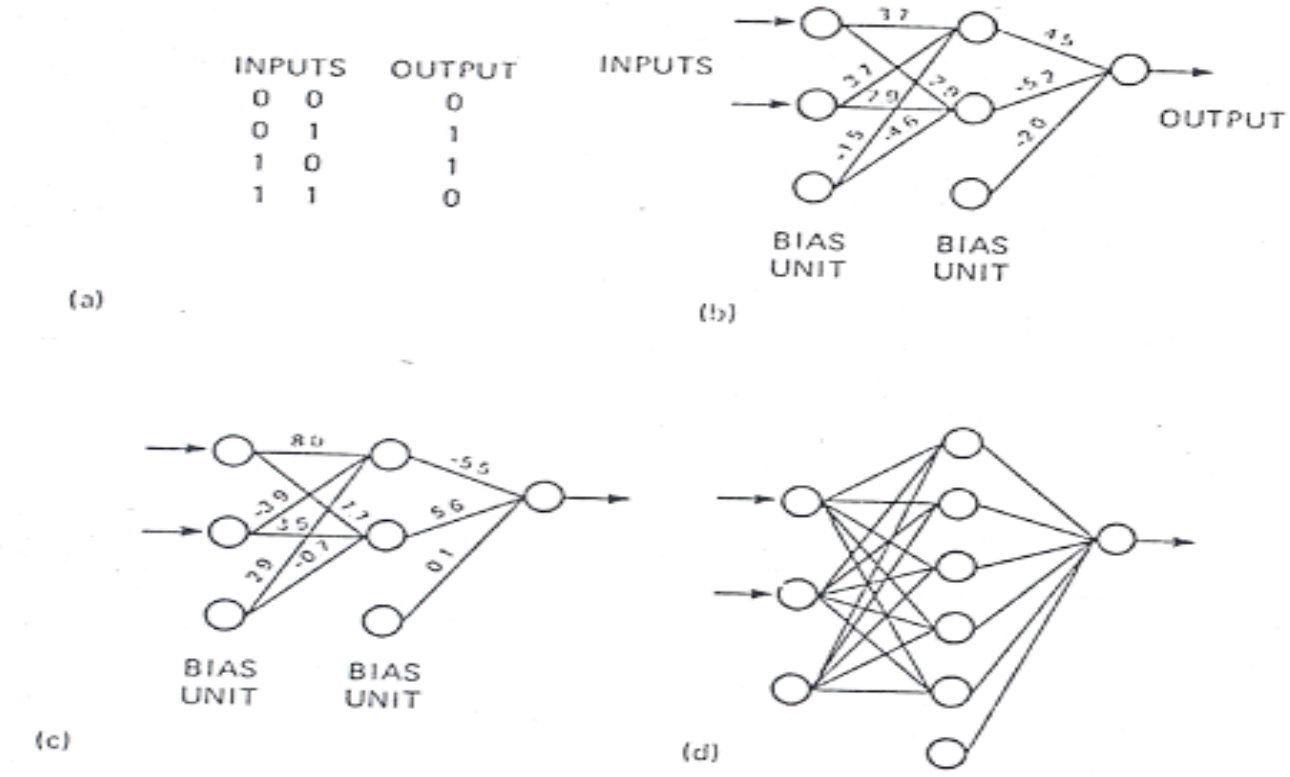
$$\sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial}{\partial o_{pj}} \sum_i w_{ki} o_{pi} = \sum_k \frac{\partial E_p}{\partial net_{pk}} w_{kj} = -\sum_k \delta_{pk} w_{kj}.$$

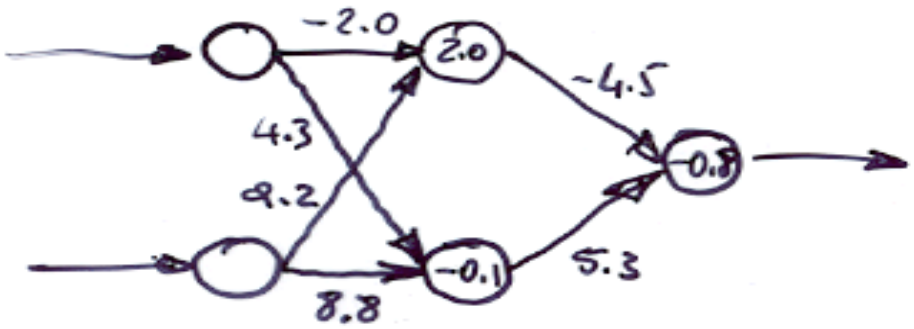
In this case, substituting for the two factors in Equation 12 yields

$$\delta_{pj} = f'_j(net_{pj}) \sum_k \delta_{pk} w_{kj} \tag{14}$$

whenever u_j is not an output unit. Equations 13 and 14 give a recursive procedure for computing the δ's for all units in the network, which are then used to compute the weight changes in the network according to Equation 11. This procedure constitutes the generalized delta rule for a feedforward network of semilinear units.

Solución al problema XOR

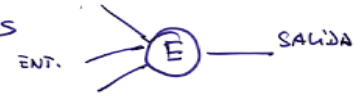


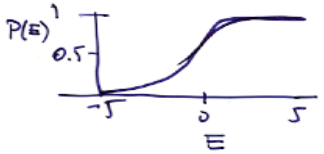


NETtalk: Sejnowski, T. J. and Rosenberg, C. R. (1986) NETtalk: a parallel network that learns to read aloud, Cognitive Science, 14, 179-211.

⇒ TRANSFORMA TEXTO ESCRITO EN FONEMAS

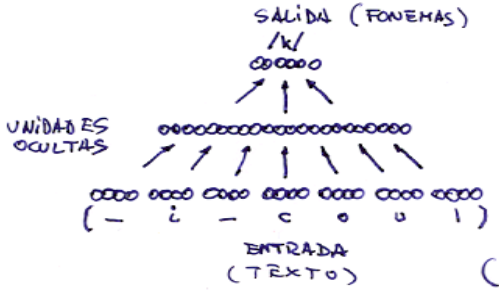
UNIDADES





$$E_i = \sum_j w_{ij} p_i \qquad p_i = P(E_i) = \frac{1}{1+e^{E_i}}$$

ARQUITECTURA



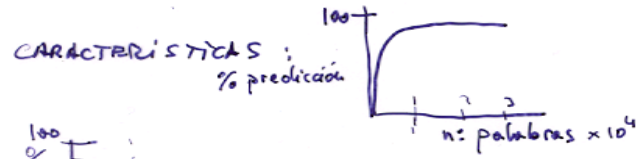
(55)
(80)
(7×29)
(TOTAL CONEX. 20.640)

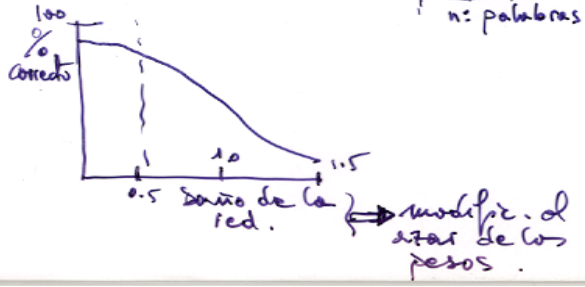
ALGORITMO DE APRENDIZAJE: "BACK-PROPAGATION"

$$w_{ij}^{(n)}(t+1) = \alpha\, w_{ij}^{(n)}(t) + (1-\alpha)\,\varepsilon\,\delta_i^{(n+1)}\, p_j^{(n)}$$

$$\begin{cases} \delta_i^{(N)} = (p_i^{*} - p_i^{(N)})\, P'(E_i^{(N)}) \\ \delta_i^{(n)} = \sum_j \delta_j^{(n+1)} w_{ji}^{(n)}\, P'(E_i^{(n)}) \end{cases}$$

CARACTERISTICAS:





Redes Neuronales en Bioinformática
Algoritmos supervisados

• Predicción de la estructura secundaria a partir de la secuencia de aminácidos de una proteína. B Rost: PHD: predicting one-dimensional protein structure by profile based neural networks. Meth. in Enzymolgy, 266, 525-539, (1996)

http://www.predictprotein.org/

http://www.cmpharm.ucsf.edu/~nomi/nnpredict.html

• Genetic Algorithm Neural Networks for Regulatory Region Identification Robert G. Beiko and Robert L. Charleboi(2005). GANN: genetic algorithm neural networks for the detection of conserved combinations of features in DNA. BMC Bioinformatics 6: 36

http://bioinformatics.org.au/gann/

• Análisis de microarrays de DNA: diagnóstico de diferentes tipos de cáncer basándose en sus expresiones génicas características. Khan et al. Nature Med. 7: 673-679 (2001)