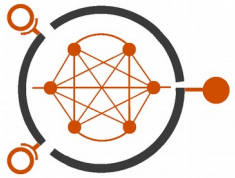
Inteligencia Artificial

Redes Neuronales

Dr. Ing. Martín G. Marchetta
martin.marchetta@ingenieria.uncuyo.edu.ar

UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO



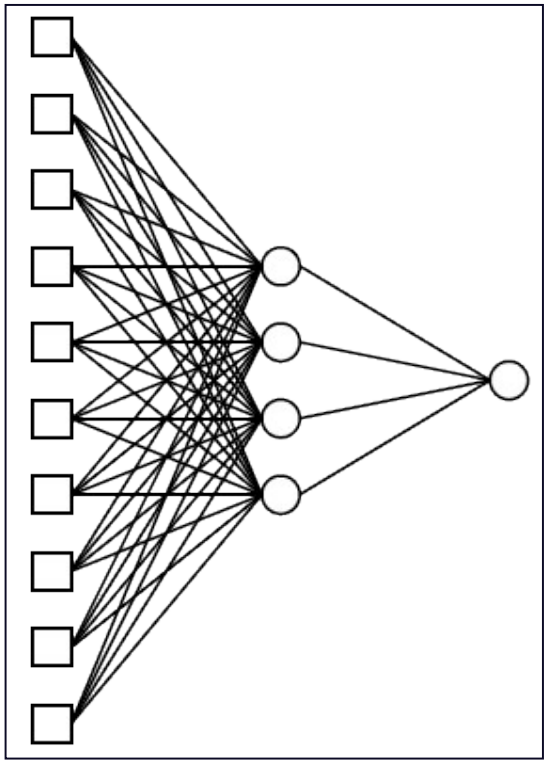
LABSIN

Introducción – ANN

- **Una neurona (biológica)**
  - Es una célula que recoge, procesa y emite señales eléctricas
  - Hipótesis: capacidad de procesamiento del cerebro se basa en redes de neuronas
- **Redes Neuronales Artificiales (ANN)**
  - También conocidos como: Artificial Neural Systems, Conexionismo, Computación Neuronal, etc.
  - Imitan la estructura del sistema nervioso
  - Enfoque bottom-up: comportamiento emergente a partir de un sistema “de bajo nivel”

Introducción – ANN

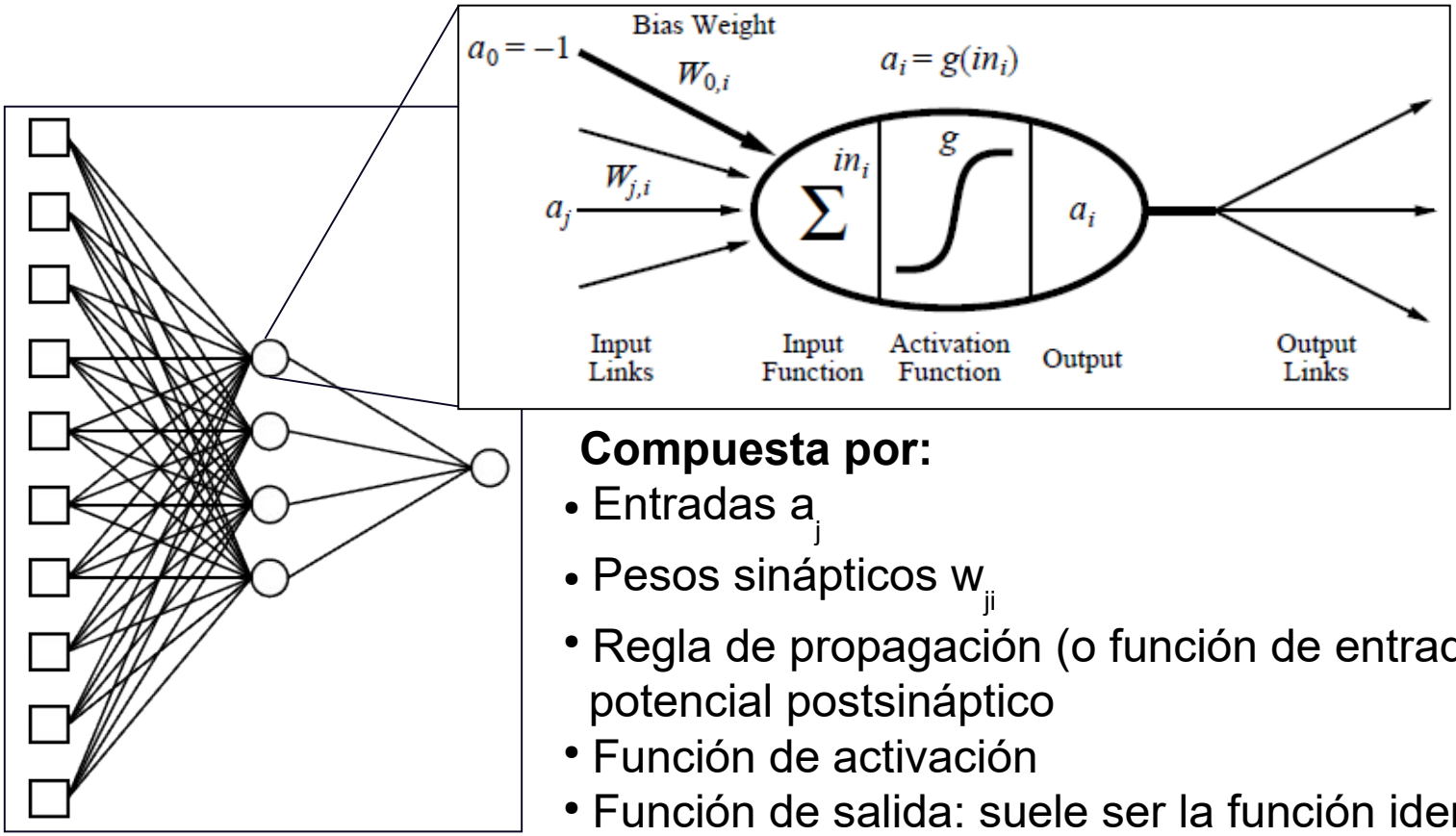
- Definición formal de Red Neuronal Artificial [Martin del Brio y Sanz, 2002]
  - Grafo dirigido
  - A cada nodo **i** se le asigna una variable de estado a_i
  - A cada conexión **j,i** se le asocia un peso w_{ji}
  - A cada nodo **i** se le asocia un umbral o sesgo θ_i
  - Para cada nodo **i** se define una función $\mathbf{g}(x_i, w_{ji}, \theta_i)$ que proporciona la salida del nodo (función de activación)



- Los nodos son las neuronas artificiales
- Las aristas son las “conexiones sinápticas” que propagan la salida de un nodo a las neuronas conectadas

Introducción – ANN

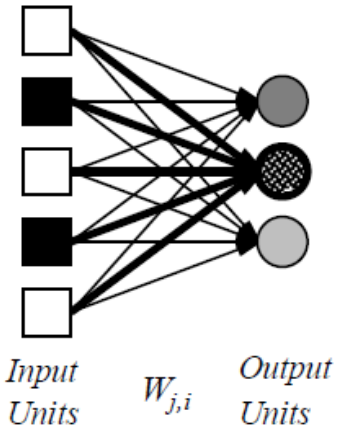
- Estructura general de una neurona artificial (McCulloch – Pitts, 1943)

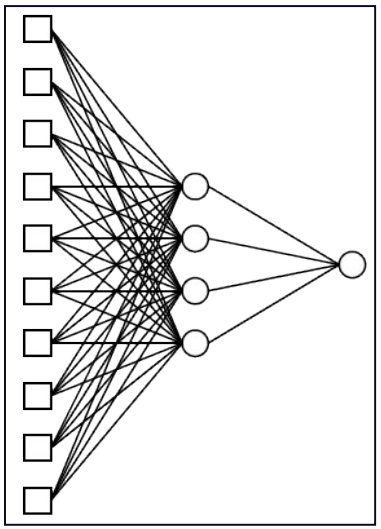


Compuesta por:

- Entradas a_j
- Pesos sinápticos w_{ji}
- Regla de propagación (o función de entrada): potencial postsináptico
- Función de activación
- Función de salida: suele ser la función identidad f(x) = x
- Salidas

- Clasificación y arquitecturas de ANN
  - Por el estilo de aprendizaje
    - Supervisados
    - No supervisados
    - Por refuerzo
    - Híbridos
  - Por la arquitectura
    - Monocapa vs. Multicapa
    - Unidireccionales vs. Realimentados





- Modos de operación de una ANN
  - Modo de aprendizaje (entrenamiento)
    - Aprendizaje: ajuste de los parámetros de la red
    - El tipo de aprendizaje define cómo se ajustan estos parámetros
    - Existen 2 niveles de aprendizaje
      - Ajuste de los valores de los parámetros (este esquema es el más común). Ej: ajuste de pesos sinápticos
      - Ajuste de la topología de la red (cantidad de nodos, conexiones, etc.)
    - La mayoría de los algoritmos de aprendizaje se basan en métodos numéricos iterativos para minimizar el error

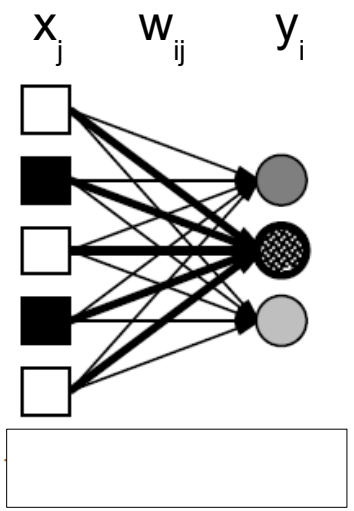
- Modos de operación de una ANN
  - Modo de ejecución
    - También llamado “recuerdo” (recall)
    - Es la operación de la ANN “en producción”
    - Generalmente en este modo de ejecución el componente de aprendizaje “se desconecta”
    - La red propaga las señales de entrada a través de su topología y produce una salida

Redes Neuronales Supervisadas

- Algunas ANN con feed-forward y aprendizaje supervisado
  - Asociador lineal con aprendizaje Hebbiano/Regla Pseudoinversa
  - Perceptrón simple
  - Adaline (ADAptive LInear NE, rebautizada ADAptive LINear Element)
  - Perceptrón Multicapa (MLP, Multi-Layer Perceptron)

- Asociador lineal
  - Objetivo: asociar p pares $\{(\mathbf{x}^{\mu}, \mathbf{t}^{\mu}) / 1 \leq \mu \leq p\}$
  - Regla de propagación $y_i = \sum_{j=0}^{n} W_{ij} x_j$
  - Función de activación = Función de salida

    Identidad → f(x) = x



- Asociador lineal (cont.)
  - Aprendizaje Hebbiano

    , $\varepsilon \in (0, 1]$ $\quad \Delta w_{ij} = \varepsilon\, y_i\, x_j$

    ε se denomina **ritmo de aprendizaje**

  - Tomando $\varepsilon = 1$, la ***regla de Hebb*** para el asociador lineal queda

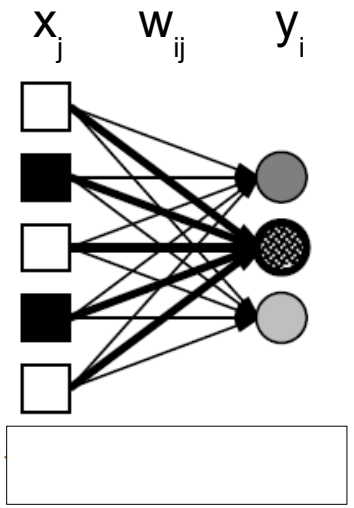
    $$\Delta w_{ij}^{\mu} = t_i^{\mu}\, x_j^{\mu} \quad \Rightarrow \quad w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij}^{\mu}$$

  - Si se parte de pesos nulos, el estado final para un peso wij es

    $$w_{ij}^{final} = 0 + \Delta w_{ij}^{1} + \ldots + \Delta w_{ij}^{p} = t_i^1 x_j^1 + \ldots + t_i^p x_j^p$$

    Matricialmente: $W = t^1 x^{1T} + t^2 x^{2T} + \ldots + t^p x^{pT}$

$$y_i = \sum_{j=0}^{n} W_{ij}\, x_j$$



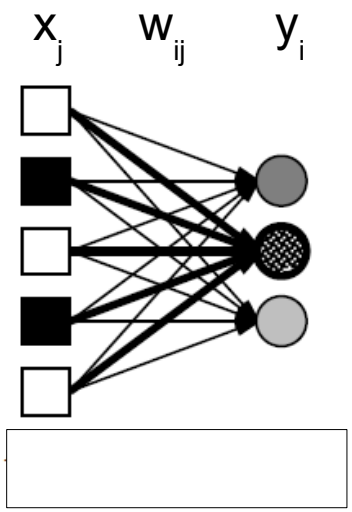
- Asociador lineal (cont.)

$$y_i = \sum_{j=0}^{n} W_{ij} x_j$$

  - Aprendizaje Hebbiano (cont.)
    - Si los vectores de entrada $\{\mathbf{x}^1, \mathbf{x}^2, \ldots \mathbf{x}^p\}$ son ortonormales (ortogonales y longitud 1), la red entrendada responderá asi:

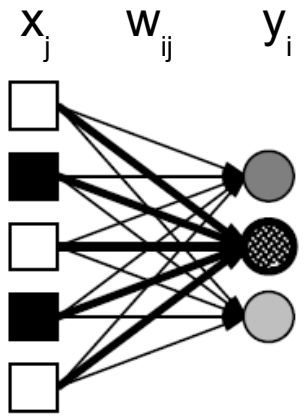
$$W\mathbf{x}^{\mu} = \left(\mathbf{t}^1 \mathbf{x}^{1T} + \ldots + \mathbf{t}^p \mathbf{x}^{pT}\right)\mathbf{x}^{\mu} =$$
$$= \mathbf{t}^1\left(\mathbf{x}^{1T}\mathbf{x}^{\mu}\right) + \ldots + \mathbf{t}^p\left(\mathbf{x}^{pT}\mathbf{x}^{\mu}\right) = \mathbf{t}^{\mu}$$

    - Problema: condiciones muy restrictivas lo hacen poco útil (entradas ortonormales)



- Perceptrón Simple
  - Red monocapa, como el asociador lineal
  - Regla de propagación

$$\sum_{j=0}^{n} w_{ij} x_j - \theta_i$$

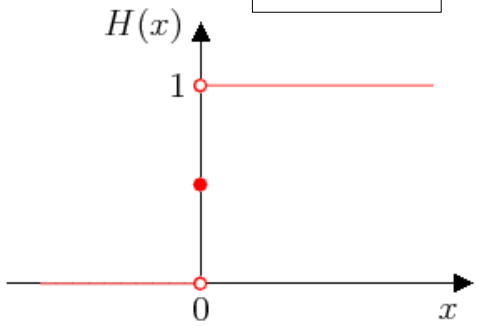


  - Función de activación: Heavyside

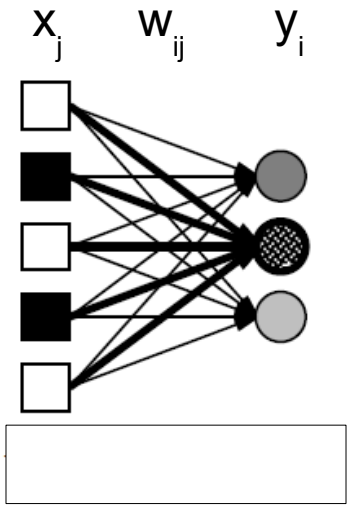
$$H(x) = \begin{cases} 1 \; si \sum_{j=0}^{n} W_{ij} x_j - \theta_i \geq 0 \\ 0 \; si \sum_{j=0}^{n} W_{ij} x_j - \theta_i < 0 \end{cases}$$

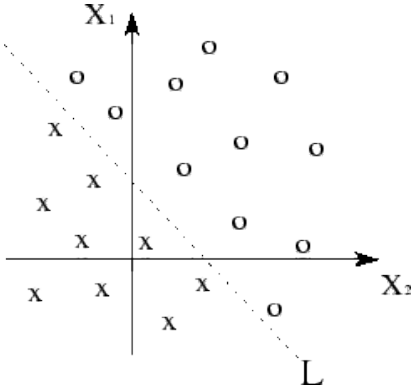
  - Función de salida: Identidad
  - Valor de salida (***n*** entradas y ***m*** salidas)

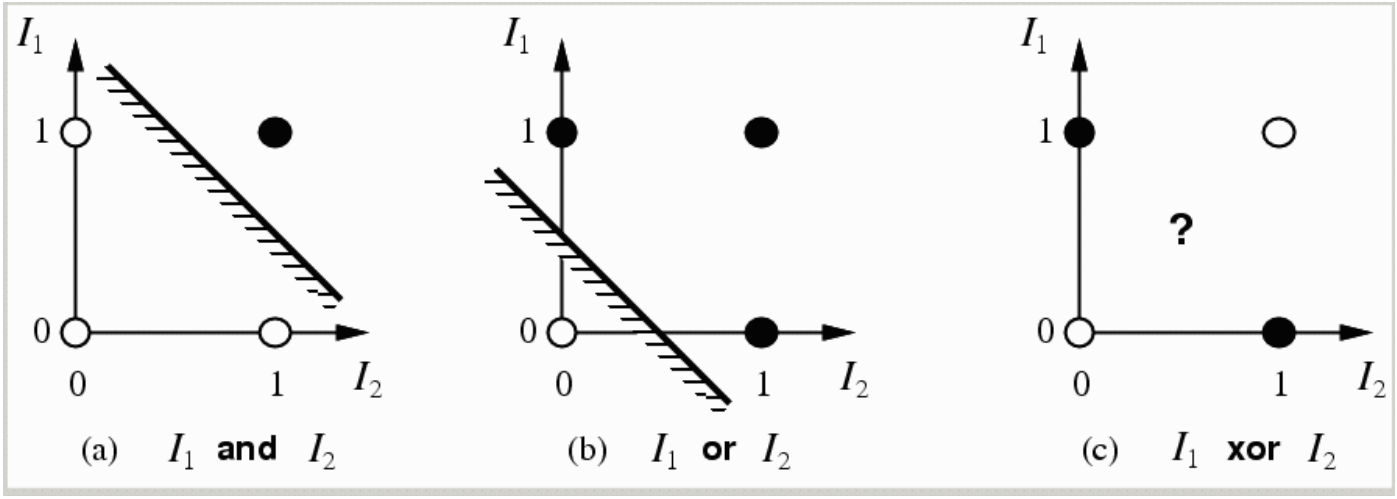
$$y_i = H\left(\sum_{j=0}^{n} W_{ij} x_j - \theta_i\right), \forall\, i, 1 \leq i \leq m, \forall\, j, 1 \leq j \leq n$$



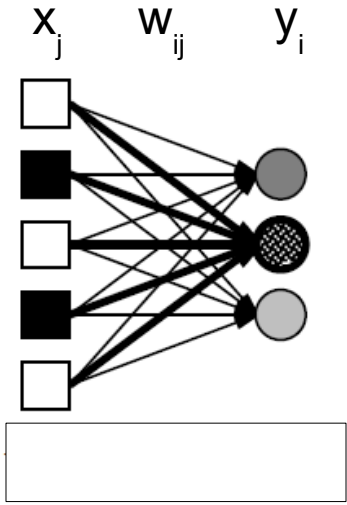
- Perceptrón Simple
  - Problema: solo discrimina patrones ***linealmente separables***







- Perceptrón Simple
  - Algoritmo de aprendizaje
    - Aprendizaje por corrección de errores
    - Ajuste en base a la diferencia entre salida esperada y salida obtenida
    - Regla de aprendizaje



$$\Delta w_{ij}^{\mu}(t) = \varepsilon \left(t_i^{\mu} - y_i^{\mu} \right) x_j^{\mu}$$

    - El error que comete la red se va reduciendo hasta proporcionar la salida deseada (si se cumplen las condiciones necesarias de separación lineal)

- ADALINE (ADAptive LInear Element)
  - Similar al perceptrón, pero la función de activación es la identidad
  - Salida (***n*** neuronas de entrada y ***m*** de salida)

$$y_i(t) = \sum_{j=1}^{n} w_{ij} x_j - \theta_i \quad , \forall i \; 1 \leq i \leq m$$

  - Limitación: al tener neuronas de respuesta lineal, sólo podrá separar patrones ***linealmente independientes***
  - La mayor ventaja respecto de las anteriores es el algoritmo de aprendizaje, basado en la minimización del error cuadrático medio (***LMS, Least Mean Squares***)

- ADALINE (cont.)
  - Algoritmo de aprendizaje
    - Descenso por el gradiente
    - La función a optimizar es una función de costo E(W), donde W son los pesos sinápticos
    - Actualización de pesos:

$$W(t+1) = W(t) - \varepsilon \nabla E(W)$$

    - Compromiso en selección de ε: aprendizaje lento vs. oscilación cerca de min/max

- ADALINE (cont.)
  - Algoritmo de aprendizaje (cont.)
    - Regla **LMS** o **Regla Widrow-Hoff**

$$E[w_{ij}] = \frac{1}{2} \sum_{\mu=1}^{p} \sum_{i=1}^{m} (t_i^\mu - y_i^\mu)^2 \quad , \text{con} \quad y_i^\mu = \sum_{j=1}^{n} w_{ij} x_j^\mu - \theta_i^\mu$$

$$\frac{\partial E[w_{ij}]}{\partial w_{ij}} = (1/2)\, 2 \sum_{\mu=1}^{p} (t_i^\mu - y_i^\mu) \left(- \frac{dy_i^\mu}{dw_{ij}} \right) =$$

$$= -(1/2)\, 2 \sum_{\mu=1}^{p} (t_i^\mu - y_i^\mu) \frac{dy_i^\mu}{dw_{ij}} = - \sum_{\mu=1}^{p} (t_i^\mu - y_i^\mu)\, x_j^\mu$$

Por lo tanto

$$\Delta w_{ij} = -\varepsilon \frac{\partial E[w_{ij}]}{\partial w_{ij}} = \varepsilon \sum_{\mu=1}^{p} (t_i^\mu - y_i^\mu)\, x_j^\mu$$

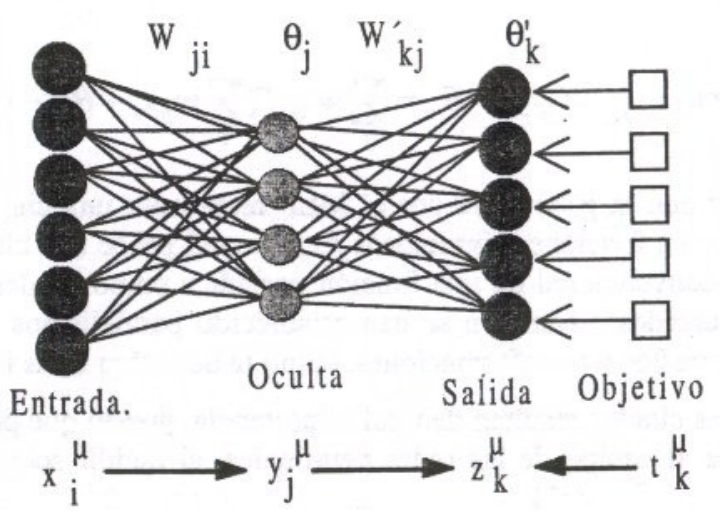
- ADALINE (cont.)
  - Dado que las neuronas son lineales, la función **E(W)** es cuadrática en los pesos, permitiendo alcanzar siempre un mínimo global (es un paraboloide)

$$E[w_{ij}] = \frac{1}{2} \sum_{\mu=1}^{p} \sum_{i=1}^{n} (t_i^{\mu} - y_i^{\mu})^2$$

  - Existe una versión multicapa de la ADALINE llamada MADALINE (Many ADALINEs) para superar las limitaciones como red monocapa

- MLP (Multi-Layer Perceptron)
  - Es una red multicapa
  - Regla de propagación
    (usualmente igual en todas las capas)

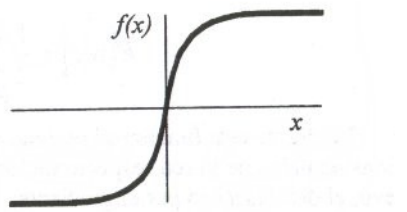
$$\sum_{i=1}^{n} w_{ji} x_i - \theta_j$$



  - Función de activación: suele ser lineal en la capa de salida, y sigmoide en la capa oculta

    Ej:

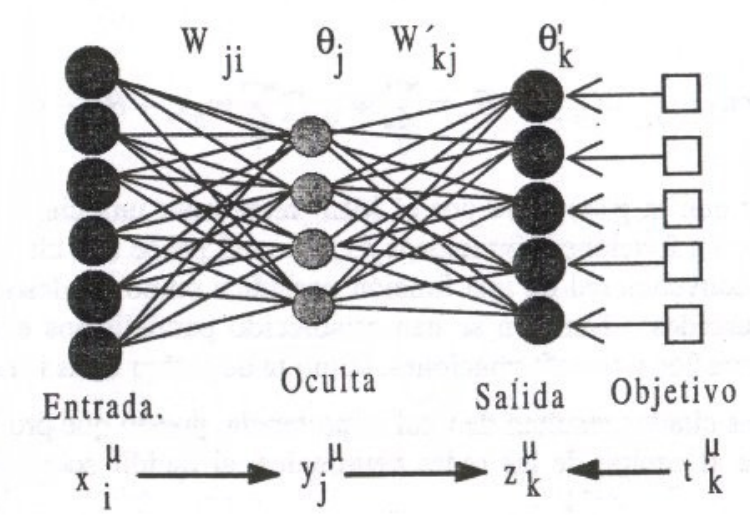
$$f(x) = \frac{1}{1+e^{-x}} \qquad f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} = \tanh(x)$$



- MLP (Multi-Layer Perceptron)
  - Salida de la red

$$z_k = g\left(\sum_j w'_{kj}\, y_j - \theta'_k\right) =$$

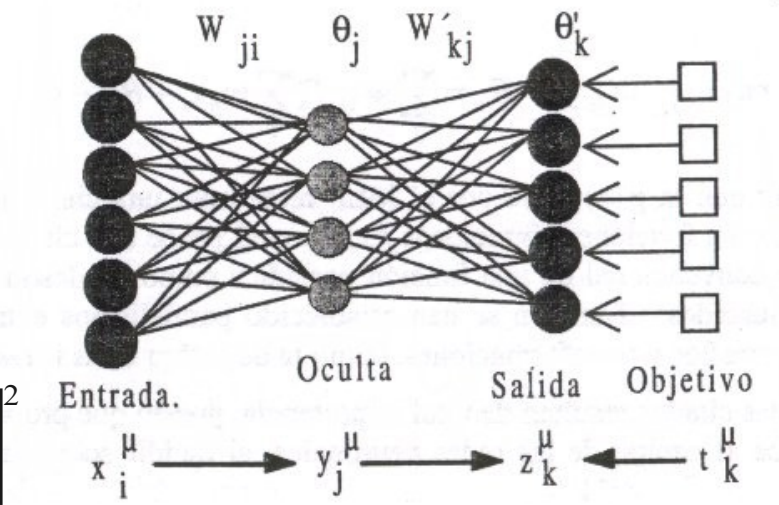
$$= g\left(\sum_j w'_{kj}\, f\left(\sum_i w_{ji}\, x_i - \theta_j\right) - \theta'_k\right)$$



donde g() es la función de activación de las neuronas de salida, y f() es la función de activación de las neuronas de la capa oculta

- MLP (Multi-Layer Perceptron)
  - Algoritmo de aprendizaje: BP (backpropagation)

$$z_k^{\mu} = g\left(\sum_j w'_{kj}\, \underbrace{f\left(\sum_i w_{ji}\, x_i^{\mu} - \theta_j\right)}_{y_j^{\mu}} - \theta'_k\right)$$



$$E(w_{ji}, w'_{kj}) = \frac{1}{2} \sum_{\mu} \sum_{k} \left[t_k^{\mu} - g\left(\sum_j w'_{kj}\, y_j^{\mu} - \theta'_k\right)\right]^2$$

$$\delta w'_{kj} = -\varepsilon \frac{\partial E}{\partial w'_{kj}} \qquad \delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w_{ji}}$$

- MLP (Multi-Layer Perceptron)
  - Algoritmo de aprendizaje: BP (backpropagation)
    Derivando respecto a los pesos y aplicando la regla de la cadena se tiene

$$\delta w'_{kj} = -\varepsilon \frac{\partial E}{\partial w'_{kj}} = \varepsilon \sum_{\mu} \Delta'^{\mu}_{k} y^{\mu}_{j}$$

$$\Delta'^{\mu}_{k} = \left[t^{\mu}_{k} - g\left(h'^{\mu}_{k} \right) \right] \frac{\partial g\left(h'^{\mu}_{k} \right)}{\partial h'^{\mu}_{k}}$$

$$h'^{\mu}_{k} = \sum w'_{kj} y^{\mu}_{j} - \theta'_{k}$$

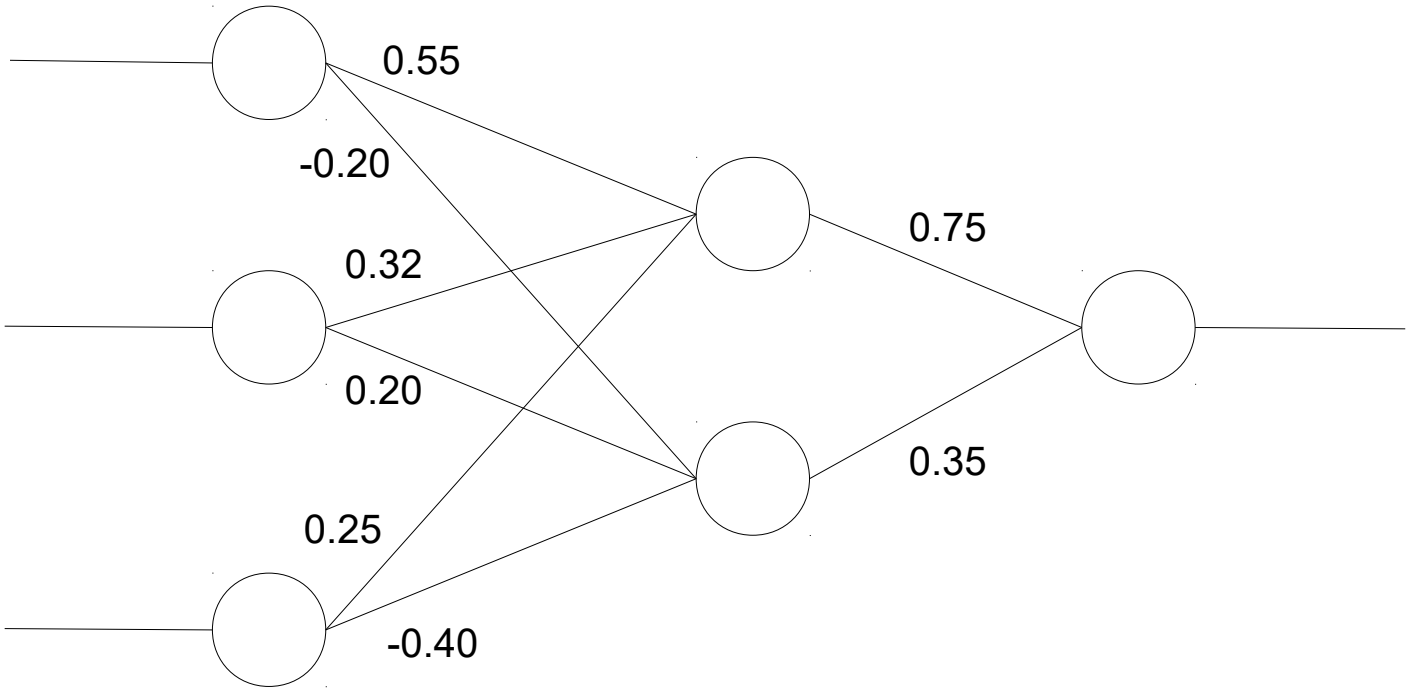
$$\delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w_{ji}} = \varepsilon \sum_{\mu} \Delta^{\mu}_{j} x^{\mu}_{i}$$

$$\Delta^{\mu}_{j} = \left[\sum_{k} \Delta'^{\mu}_{k} w'_{kj} \right] \frac{\partial f\left(h^{\mu}_{j} \right)}{\partial h^{\mu}_{j}}$$

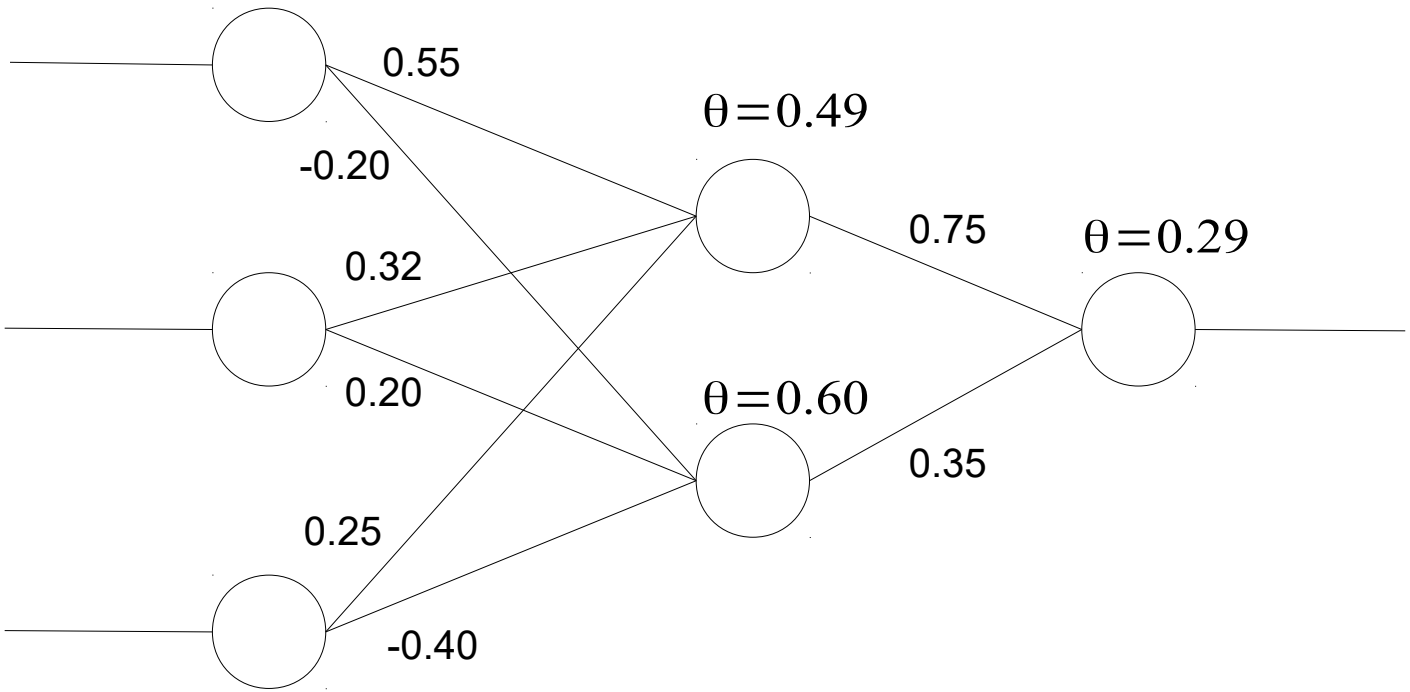
$$h^{\mu}_{j} = \sum_{i} w_{ji} x^{\mu}_{i} - \theta_{j}$$

- MLP (Multi-Layer Perceptron)
  - Cantidad de ejemplos de entrenamiento
    - Para una red con **w** pesos sinápticos se requiere aproximadamente una cantidad de patrones $\mathbf{p=w/\varepsilon}$ para obtener un error del orden de $\boldsymbol{\varepsilon}$.

- MLP: Un ejemplo
  - **Arquitectura:**
    - 3 entradas, 1 salida, 1 capa oculta
    - Función de activación capa oculta y de salida: $f(x)=g(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=\tanh(x)$
    - Derivadas: $f'(x)=g'(x)=1-\tanh(x)$



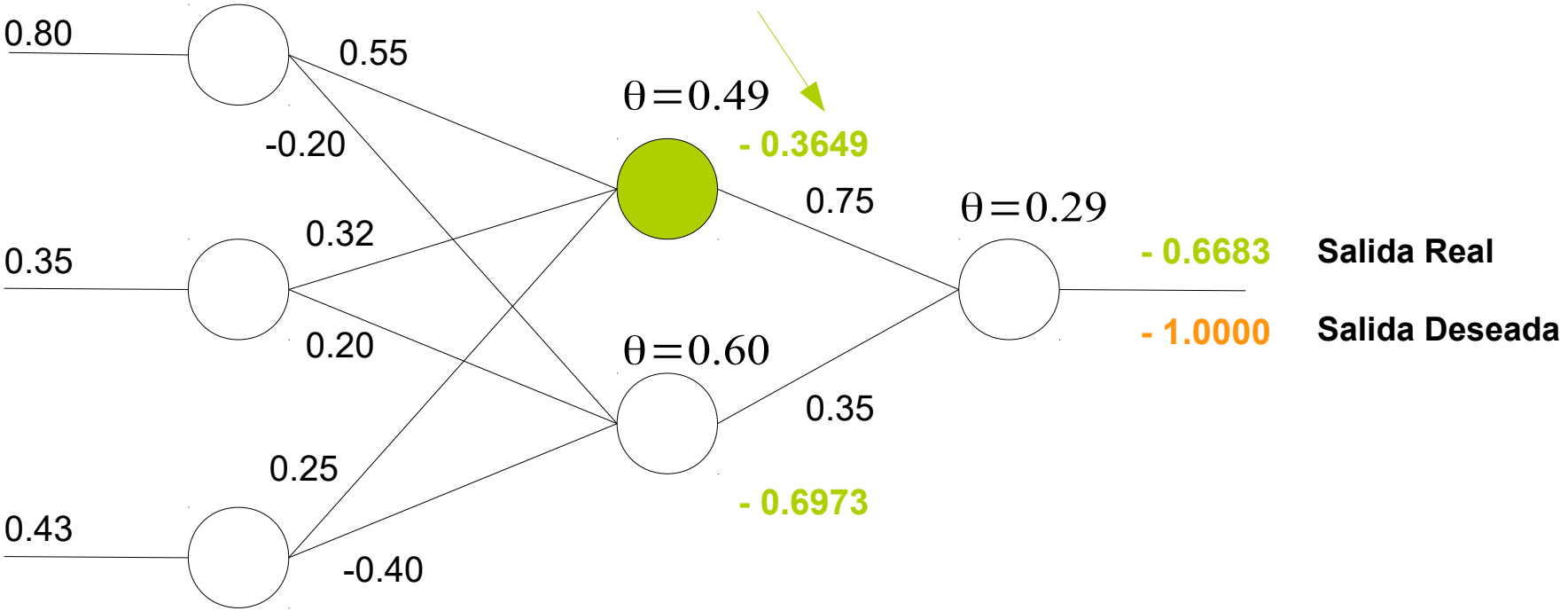
- MLP: Un ejemplo
  - **Estado inicial de la red**: pesos aleatorios, pequeños, positivos y negativos



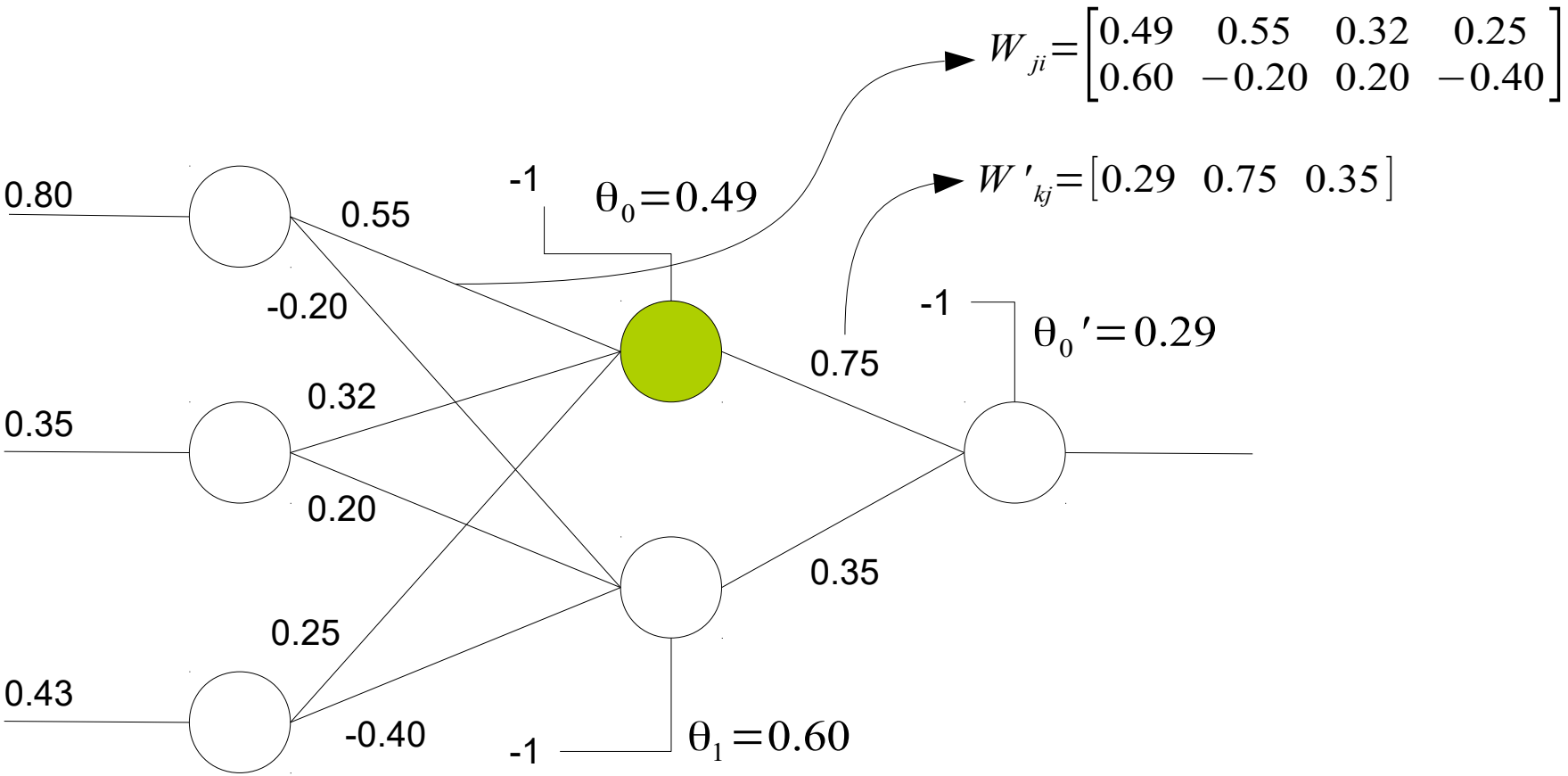
- MLP: Un ejemplo
  - **Entrenamiento**: (1) Cálculo de la salida de la red

Regla de Propagación: $\sum_{i=1}^{n} w_{ji} x_i - \theta_j = 0.55 * 0.80 + 0.32 * 0.35 + 0.25 * 0.43 - 0.49 = -0.3825$

Función de Activación: $\tanh(-0.3825) = -0.3649$



- MLP: Un ejemplo
  - **Entrenamiento**: (2) Backpropagation



- MLP: Un ejemplo
  - **Entrenamiento**: (2.a) Backpropagation para w'_{kj}

$$h'_0 = \sum_j w_{0j}\, y_j - \theta_0 = [0{,}75 \times (-0{,}3649)] + [0{,}35 \times (-0{,}6973)] - 0{,}29 = -0{,}8077$$

$g'(h'_k)=1-\tanh(h'_k)$

$$\Delta'_0 = [t_0 - g(h'_0)]\frac{\partial g(h'_0)}{\partial h'_0} = (-1.000 - (-0.6683))(1 - \tanh(-0.8077)) = -0.5534$$

$$\delta w'_{0j} = -\varepsilon \frac{\partial E}{\partial w'_{0j}} = \varepsilon\, \Delta'_0\, y_j$$

$$\delta w'_{00} = \epsilon\, \Delta'_0\, \theta'_0 = 0.30(-0.5534)(-1.0000) = 0{,}1660$$

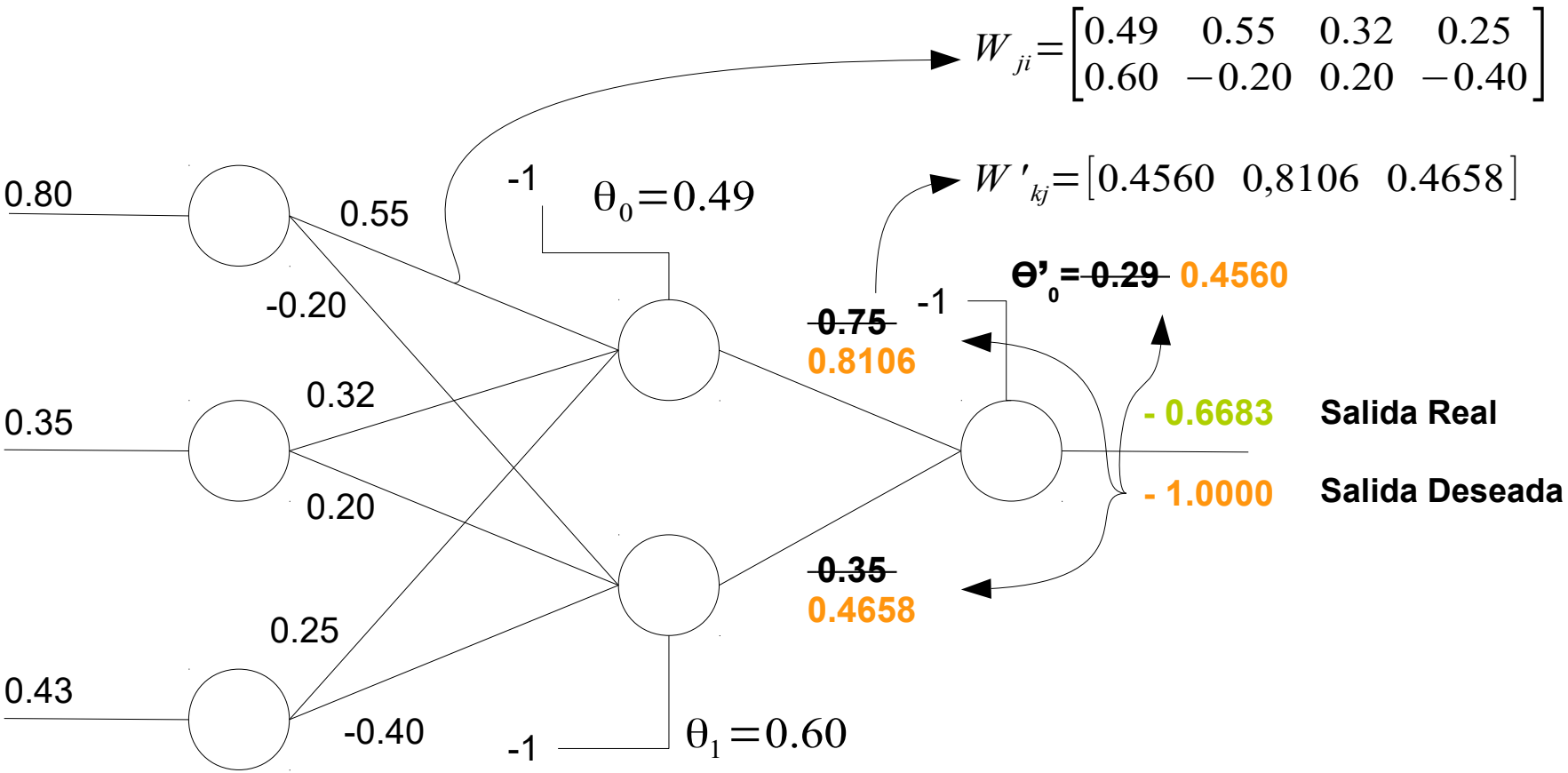
$$\delta w'_{01} = \epsilon\, \Delta'_0\, y_0 = 0.30(-0.5534)(-0.3649) = 0{,}0606$$

$$\delta w'_{02} = \epsilon\, \Delta'_0\, y_1 = 0.30(-0.5534)(-0.6973) = 0{,}1158$$

donde $\varepsilon = 0.30$

$$W'_{kj}(t+1) = W'_{kj}(t) + \delta w'_{kj} = [0.29 \;\; 0.75 \;\; 0.35] + [0.1660 \;\; 0.0606 \;\; 0.1158] = [0{,}4560 \;\; 0{,}8106 \;\; 0.4658]$$

- MLP: Un ejemplo
  - **Entrenamiento**: (2.a) Backpropagation para w'_{kj}



- MLP: Un ejemplo
  - **Entrenamiento**: (2.b) Backpropagation para w_{ji}

Primera neurona de la capa oculta (j=0)

$$h_0 = \sum_i w_{0i} x_i - \theta_0 = 0.55 \times 0.80 + 0.32 \times 0.35 + 0.25 \times 0.43 - 0.49 = 0.1695$$

$$\Delta_0 = \left[\sum_k \Delta'_k w'_{k0}\right] \frac{\partial f(h_0)}{\partial h_0} = \Delta'_0 w'_{00} f'(h_0) = (-0.5534)\,0.75\,(1 - \tanh(0.1695)) = -0.3454$$

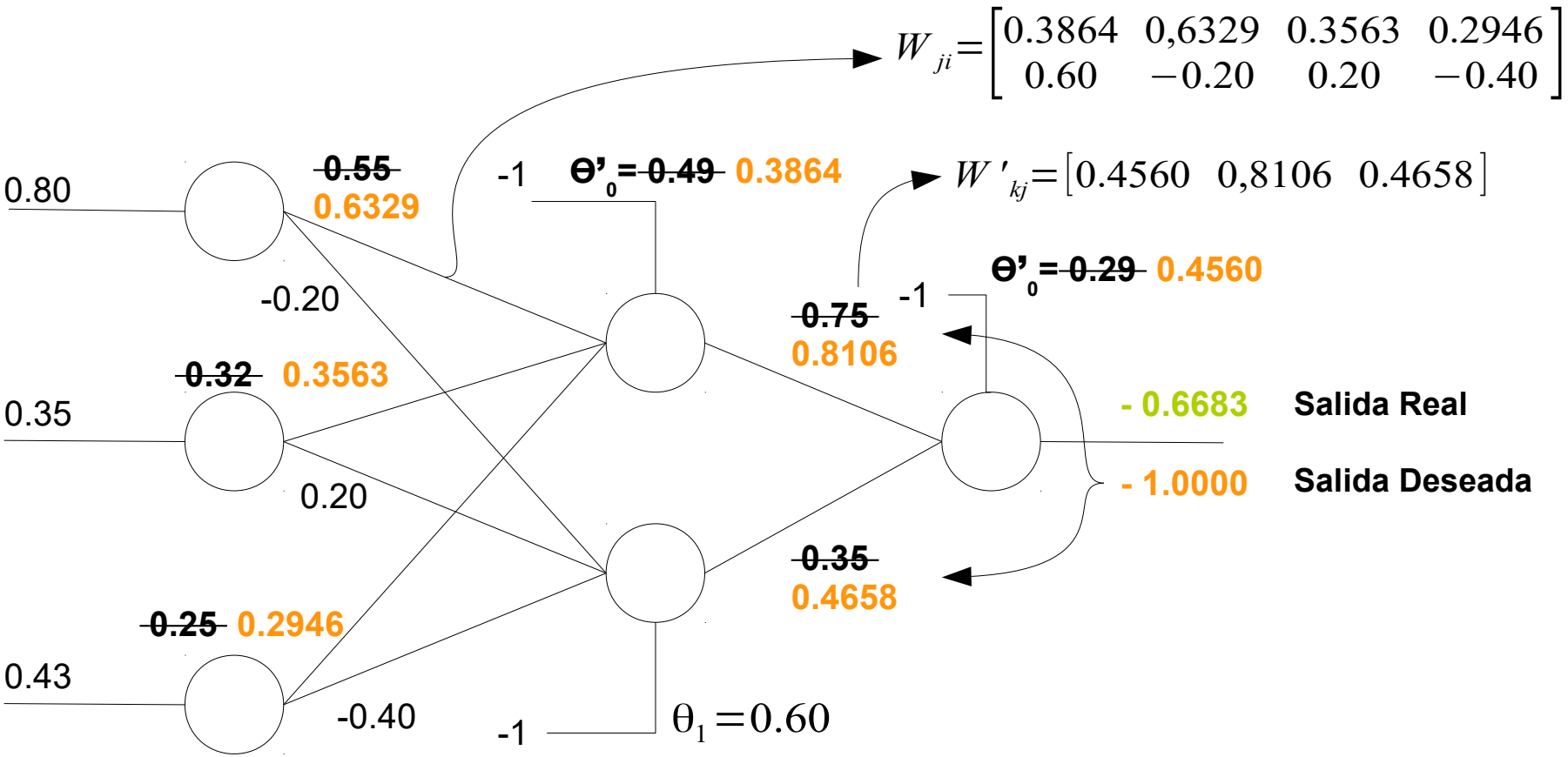
(Δ'_0, w'_{00}, $f'(h'_0)$)

$$\delta w_{0i} = -\varepsilon \frac{\partial E}{\partial w_{0i}} = \varepsilon \Delta_0 x_i$$

$$\begin{cases} \delta w_{00} = \epsilon \Delta_0 \theta_0 = 0.30(0.3454)(-1) = -0.1036 \\ \delta w_{01} = \epsilon \Delta_0 x_0 = 0.30(0.3454)(0.80) = 0.0829 \\ \delta w_{02} = \epsilon \Delta_0 x_1 = 0.30(0.3454)(0.35) = 0.0363 \\ \delta w_{03} = \epsilon \Delta_0 x_2 = 0.30(0.3454)(0.43) = 0.0446 \\ \text{donde } \varepsilon = 0.30 \end{cases}$$

$$W_{0i}(t+1) = W_{0i}(t) + \delta w_{0i} = [0.49 \;\; 0.55 \;\; 0.32 \;\; 0.25] + [-0.1036 \;\; 0.0829 \;\; 0.0363 \;\; 0.0446] = [0{,}3864 \;\; 0{,}6329 \;\; 0.3563 \;\; 0.2946]$$

- MLP: Un ejemplo
  - **Entrenamiento**: (2.a) Backpropagation para w'_{kj}



- MLP: Un ejemplo
  - **Entrenamiento**: (2.b) Backpropagation para w_{ji}

Segunda neurona de la capa oculta (j=1)

$$h_1 = \sum_i w_{1i} x_i - \theta_1 = -0.20 \times 0.80 + 0.20 \times 0.35 + 0.25 \times -0.40 - 0.60 = -0.79$$

$$\Delta_1 = \left[\sum_k \Delta'_k w'_{k1} \right] \frac{\partial f(h_1)}{\partial h_1} = \Delta'_0 w'_{01} f'(h_1) = (-0.5534)\, 0.35\, (1 - \tanh(-0.79)) = -0.3212$$

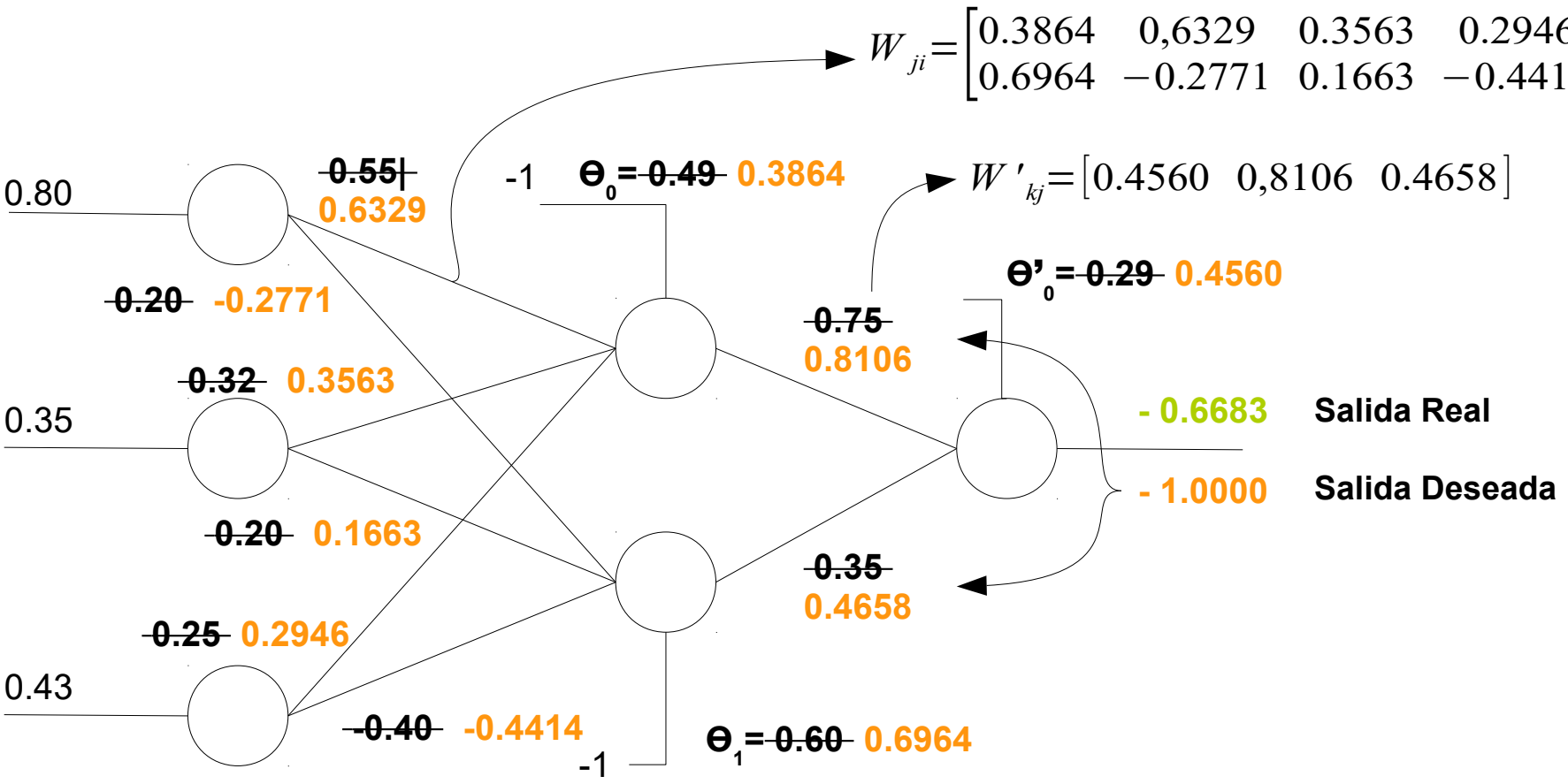
(Δ'_0, w'_{01}, $f'(h'_1)$)

$$\delta w_{1i} = -\varepsilon \frac{\partial E}{\partial w_{1i}} = \varepsilon \Delta_1 x_i$$

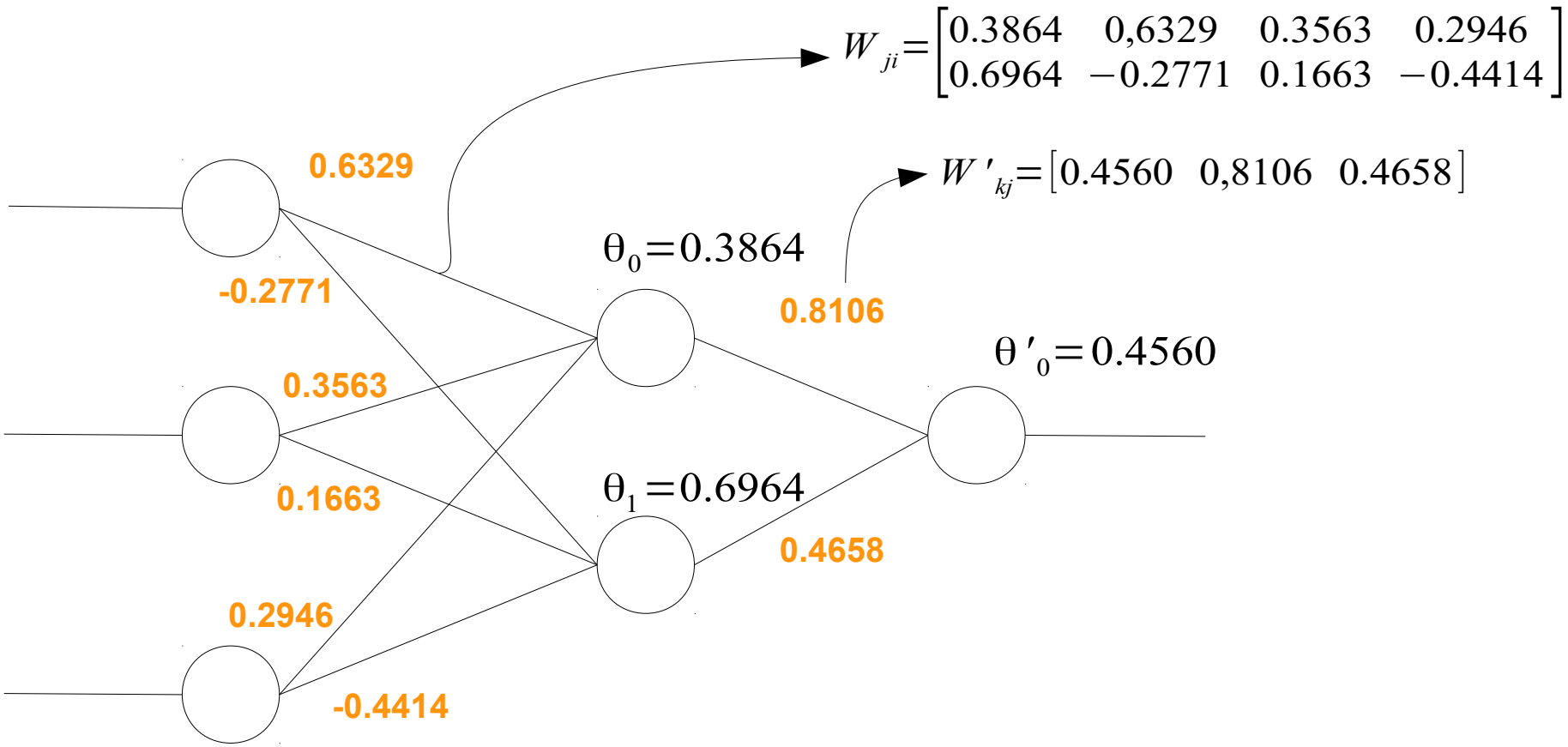
$$\begin{cases} \delta w'_{10} = \epsilon \Delta_1 \theta_1 = 0.30\,(-0.3212)(-1.0) = 0{,}0964 \\ \delta w'_{11} = \epsilon \Delta_1 x_0 = 0.30\,(-0.3212)(0.80) = -0{,}0771 \\ \delta w'_{12} = \epsilon \Delta_1 x_1 = 0.30\,(-0.3212)(0.35) = -0{,}0337 \\ \delta w'_{13} = \epsilon \Delta_1 x_2 = 0.30\,(-0.3212)(0.43) = -0{,}0414 \\ \text{donde } \varepsilon = 0.30 \end{cases}$$

$$W_{1i}(t+1) = W_{1i}(t) + \delta w_{1i} = [0.60 \;\; -0.20 \;\; 0.20 \;\; -0.40] + [0.0964 \;\; -0.0771 \;\; -0.0337 \;\; -0.0414] = [0.6964 \;\; -0.2771 \;\; 0.1663 \;\; -0{,}4414]$$

- MLP: Un ejemplo
  - **Entrenamiento**: (2.b) Backpropagation para w_{ji}



- MLP: Un ejemplo
  - **Estado de la red luego de entrenarla con 1 ejemplo**



- Aspectos prácticos de la aplicación de un ANS
  - Definir claramente el problema a resolver (tipo de entradas, tipo de salidas que se desea obtener, etc)
  - Revisión bibliográfica
  - Elegir adecuadamente la cantidad de entradas y salidas (muy pocas limitará la capacidad de generalización; demasiadas podría generar asociaciones inválidas entre entradas y salidas) →Dilema Sesgo vs. Varianza
  - Elección de conjuntos de aprendizaje y test
    - El set de entrenamiento debe cubrir razonablemente el fenómeno que se desea modelar
    - Se recomienda validación cruzada con parada temprana para evitar overfitting
    - Los ejemplos de aprendizaje no deben incluirse en el conjunto de test

- Aspectos prácticos de la aplicación de un ANS
  - Preprocesamiento de las entradas
    - Se aconseja: buena distribución, entradas con rangos parecidos y dentro del intervalo de trabajo de la función de activación de la capa oculta
    - Las entradas pueden escalarse a un intervalo [0, 1] o [-1, 1]. Enfoques
      - Por patrones (por filas): se normaliza el valor de los componentes del vector de entrada al intervalo [0, 1] linealmente en base al maximo y minimo del patron ***(solo aplicable cuando todas las entradas tienen rangos de valores parecidos)***
      - Global por columnas: se normalizan los valores de cada entrada, de acuerdo a los máximos y mínimos de esa entrada a lo largo de todos los patrones

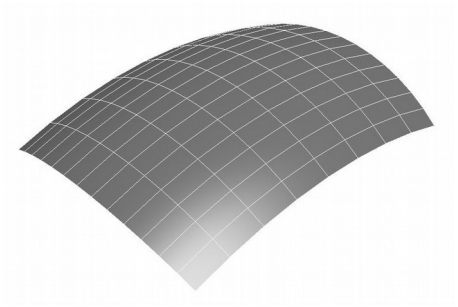
- Aspectos prácticos de la aplicación de un ANS
  - Preprocesamiento de las entradas
    - Un procesamiento que suele dar buenos resultados:
      - Estandarizar: transformar las entradas para que tengan media 0 y varianza 1.
        Cálculo: por cada componente del vector de entrada, calcular:

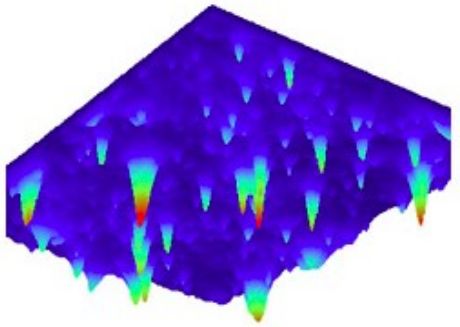
$$x' = \frac{x - \bar{x}}{\sigma}$$

      - Luego de estandarizar, escalar al intervalo [0,1] o [-1,1] mediante una transformación lineal

- Aspectos prácticos de la aplicación de un ANS
  - Entrenamiento
    - Inicialización de pesos: normalmente aleatorios, pequeños, positivos y negativos (ej: [-0.3, 0.3], [-1.1], etc)
    - Ritmo de aprendizaje lo más grande posible que no cause oscilaciones en el algoritmo. Se recomienda un ritmo de aprendizaje variable (se verá más adelante)
    - Neuronas ocultas: idealmente, la menor cantidad posible que produzca una buena respuesta (navaja de Ockham). Se recomienda probar con pocas (1 o 2 capas ocultas, con unas pocas neuronas cada una), y experimentar agregando paulatinamente neuronas a cada capa

- Velocidad de convergencia del algoritmo BP
  - Se ve afectada por algunos problemas
    - En superficies con poca pendiente, el avance es lento
    - En superficies con mucha pendiente, hay riesgo de oscilación entre picos





    - En zonas donde las derivadas de las funciones de activación son pequeñas, los ajustes de los pesos son pequeños (el tamaño del ajuste depende del valor de las derivadas)

- Mejoras para la velocidad de convergencia y estabilidad del algoritmo BP: actualización de pesos
  - Métodos de primer orden: siguen usando solo las derivadas primeras. Ej: Momentum, SAB, SuperSAB, RPROP
  - Métodos de segundo orden: usan información de las derivadas segundas. Ej: Levenberg-Marquardt
  - Ninguno de ellos domina a los otros
    - En algunos casos alguno funciona mejor que otros, y en otros casos ocurre lo contrario
    - Se debe experimentar para determinar cuál conviene en cada problema

- Actualización de pesos – Métodos de 1er orden
  - Momentum
    - Añadir al cálculo de variación de pesos un ***término de inercia***, proporcional al incremento en la iteración anterior
    - Utiliza los mismos parámetros para todos los pesos
    - Tiende a acelerar las variaciones en zonas con poca pendiente, y a hacerlas más suaves en zonas escarpadas (reduciendo/evitando oscilaciones)

$$\delta w'_{kj}(t) = -\varepsilon \frac{\partial E}{\partial w'_{kj}}(t) + \alpha \, \delta w'_{kj}(t-1)$$

$$\delta w_{ji}(t) = -\varepsilon \frac{\partial E}{\partial w_{ji}}(t) + \alpha \, \delta w_{ji}(t-1)$$

$$\alpha \in [0, 1]$$

- Actualización de pesos – Métodos de 1er orden
  - SuperSAB (Super Self-Adapting Backpropagation)
    - Agrega inercia y ritmo de aprendizaje independiente para cada peso
    - El ajuste de pesos y el cambio en el ritmo de aprendizaje dependen del signo del gradiente en iteraciones contínuas
    - El ajuste de pesos depende además de la magnitud del gradiente

$$\Delta w_{ij}(t) = \begin{cases} -\varepsilon_{ij}(t)\frac{\partial E}{\partial w_{ij}}(t) + \alpha\,\Delta w_{ij}(t-1) - decay\, w_{ij}(t), & si\ \frac{\partial E}{\partial w_{ij}}(t)\frac{\partial E}{\partial w_{ij}}(t-1) \geq 0 \\ 0 \quad en \quad otro \quad caso \quad (es \quad decir, \quad w_{ij}(t) = w_{ij}(t-1)) & \\ donde \quad decay \quad es \quad un \quad parámetro \quad del \quad modelo & \end{cases}$$

$$\varepsilon_{ij}(t) = \begin{cases} \varepsilon_{ij}(t-1)\,u, \quad si \quad \frac{\partial E}{\partial w_{ij}}(t)\frac{\partial E}{\partial w_{ij}}(t-1) \geq 0 \quad y \quad \varepsilon_{ij}(t-1) \leq \varepsilon_{max} \\ \varepsilon_{ij}(t-1)\,d, \quad en \quad otro \quad caso \\ donde \quad u \in [1.1, 1.3] \quad y \quad d \in [0.7, 0.9] \end{cases}$$

- Actualización de pesos – Métodos de 1er orden
  - RPROP (Resilient Backpropagation) – Sin weight Backtracking
    - Sólo utiliza el signo del gradiente para el tamaño del ajuste de los pesos
    - Es independiente de la magnitud del gradiente

$$\Delta_{ij}(t) = \begin{cases} \eta^{+}\,\Delta_{ij}(t-1), & si \quad \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t) > 0 \\ \eta^{-}\,\Delta_{ij}(t-1), & si \quad \frac{\partial E}{\partial w_{ij}}(t-1)\frac{\partial E}{\partial w_{ij}}(t) < 0 \\ \Delta_{ij}(t-1) & en \quad otro \quad caso \end{cases}$$

$$donde \quad 0 < \eta^{-} < 1 < \eta^{+} \; (originalmente \quad 0.5 \quad y \quad 1.2 \quad respectivamente)$$

$$\delta w_{ij}(t) = \begin{cases} -\Delta_{ij}(t) & si \quad \frac{\partial E}{\partial w_{ij}}(t) > 0 \\ +\Delta_{ij}(t) & si \quad \frac{\partial E}{\partial w_{ij}}(t) < 0 \\ 0 & en \quad otro \quad caso \end{cases}$$

Parámetros necesarios

$\Delta_0 \qquad \Delta_{max}$

Parámetro opcional

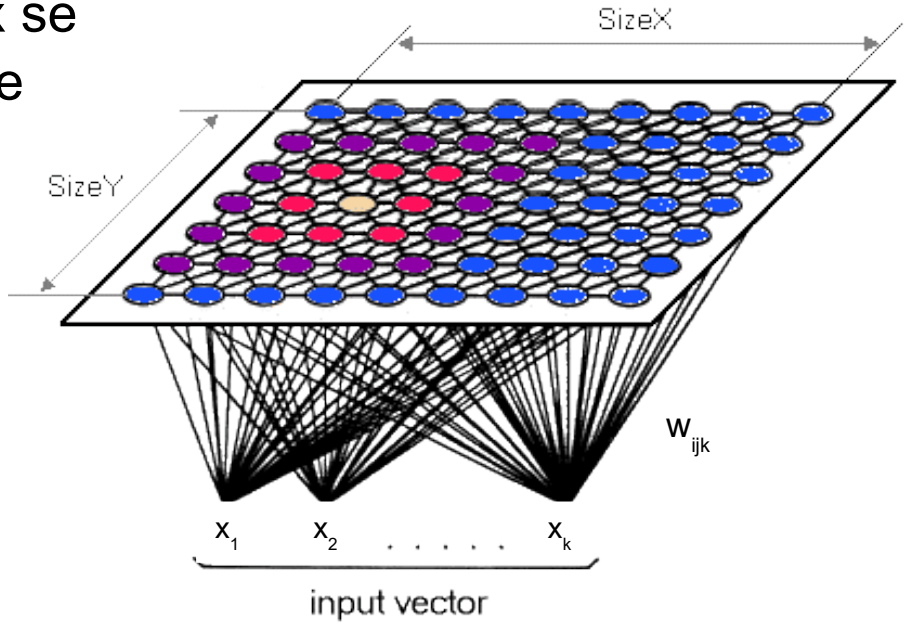
Δ_{min}

Redes Neuronales No Supervisadas

- Redes neuronales No supervisadas
  - Característica principal: durante el entrenamiento no se asocia la salida deseada con los patrones presentados
  - Estas redes se **autoorganizan** para agrupar patrones con ciertas similitudes
  - Algunas aplicaciones:
    - Análisis de similitud/novedad de patrones
    - Análisis de componentes principales
    - Clustering/Clasificación
    - Memoria asociativa
    - Codificación
    - Mapas de rasgos
    - Análisis exploratorio

- Redes neuronales No supervisadas
  - Suelen ser monocapa
  - Pueden combinarse con redes supervisadas, constituyendo redes híbridas (usualmente estas redes incluyen capas supervisadas y capas no supervisadas)
  - Hay 2 tipos
    - Redes no supervisadas Hebbianas: múltiples neuronas pueden quedar activadas
    - Redes no supervisadas Competitivas: sólo una neurona (o un pequeño grupo) pueden quedar activadas
      - También se las denomina WTA: Winner Takes All
  - En los modelos competitivos, la/s neurona/s ganadora/s se premian reforzando sus conexiones sinápticas

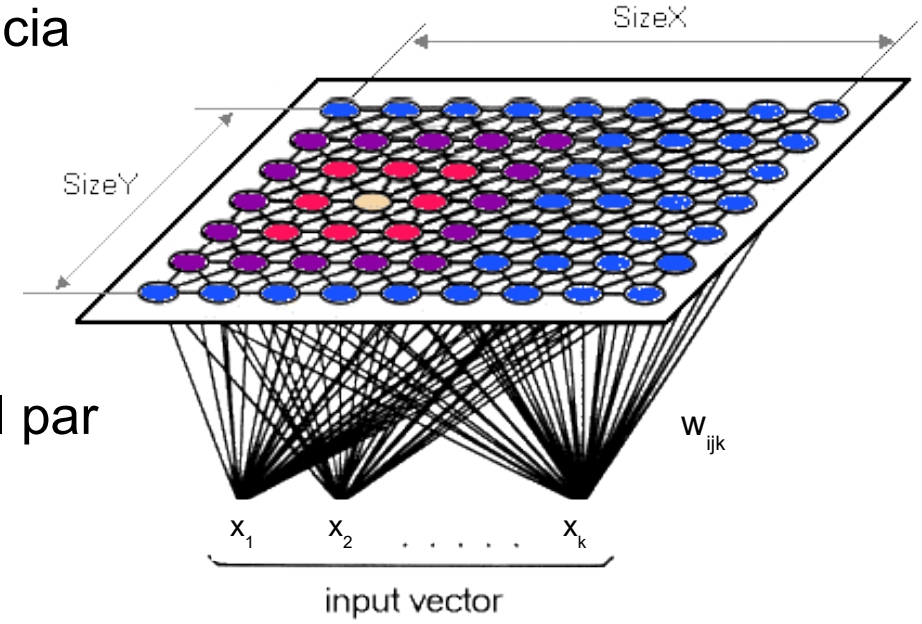
- SOFM (Self-Organizing Feature Maps)
  - 2 capas: una de entrada y otra de procesamiento/salida
  - La capa de entrada tiene un vector de ***k*** neuronas, cada una representando un elemento del vector de entrada **x**
  - La capa de procesamiento usualmente es un rectángulo con ***i*** x ***j*** neuronas
  - Cada elemento x_k de la entrada **x** se vincula con todas las neuronas de la capa de salida con un peso sináptico w_{ijk}



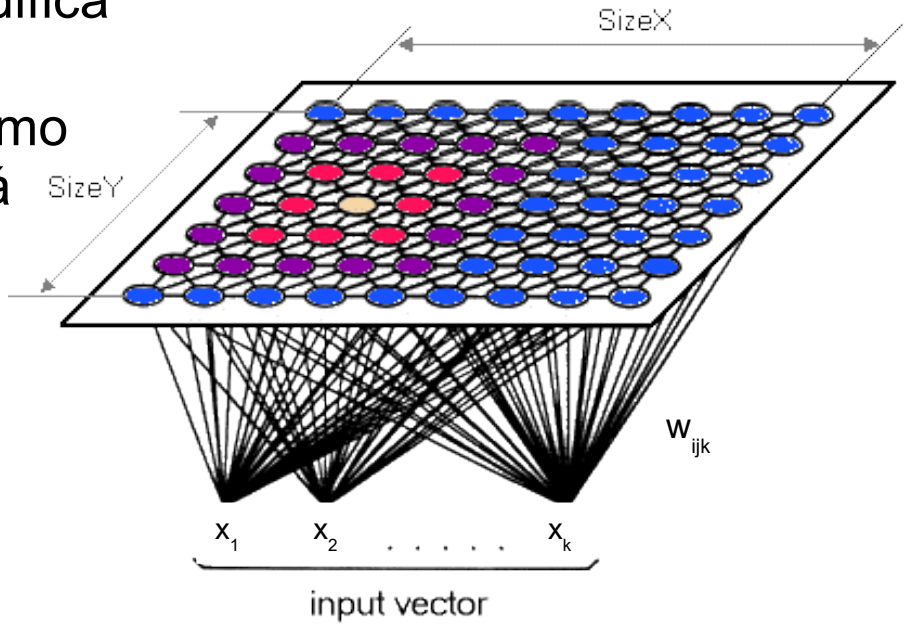
- SOFM (Self-Organizing Feature Maps)
  - **Fase de ejecución**
    - Los pesos permanecen fijos (aprendizaje se “desenchufa”)
    - Cada neurona ***(i, j)*** calcula la ***similitud*** del vector de entrada ***x*** con su vector de pesos sinápticos $\boldsymbol{w}_{ij}$
    - Existen diversas ***funciones de similitud***
    - Gana la neurona cuya distancia sea menor

$$d(\boldsymbol{w}_g, \boldsymbol{x}) = \min_{ij} \{ d(\boldsymbol{w}_{ij}, \boldsymbol{x}) \}$$

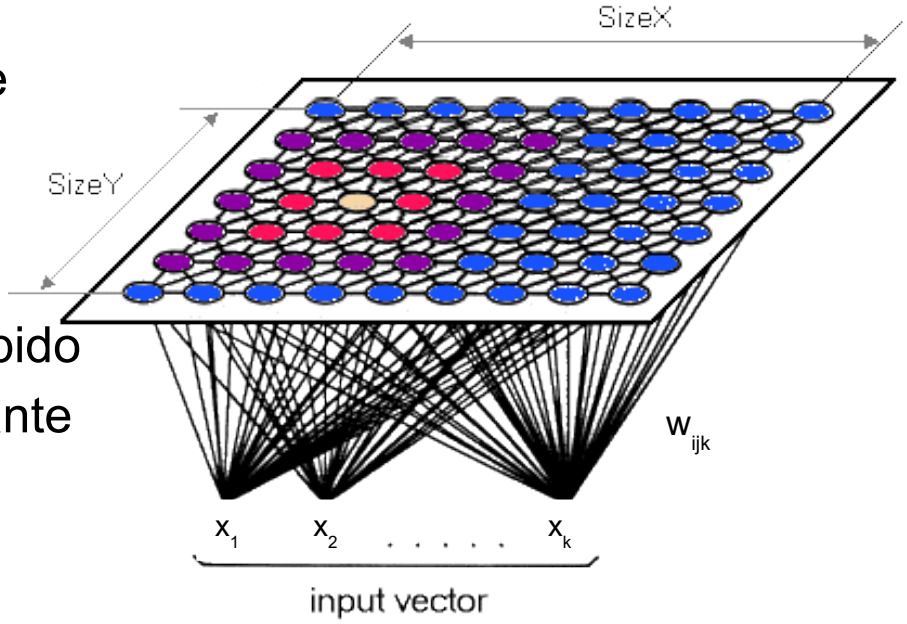
    - La neurona ganadora **g** es el par ij que minimiza la distancia



- SOFM (Self-Organizing Feature Maps)
  - **Fase de aprendizaje**
    - Cada neurona se “especializa” en un grupo o familia de patrones
    - Para el aprendizaje, se le presentan a la red múltiples patrones en secuencia
    - Por cada patrón, se calcula la neurona ganadora, y el vector de pesos $\boldsymbol{w_{ij}}$ de la misma se modifica para que se parezca un poco más a la entrada (ante el mismo patrón, la neurona se activará con más intensidad en iteraciones subsiguientes)
    - Esto representa un modelo competitivo sencillo (una sola neurona ganadora)



- SOFM (Self-Organizing Feature Maps)
  - **Fase de aprendizaje**
    - El modelo de Kohonen incorpora una ***función de vecindad*** que indica las neuronas vecinas cuyos pesos también se actualizarán
    - Patrones similares en el espacio sensorial tendrán representacones cercanas en la superficie de la red
    - La función de vecindad premia a las neuronas vecinas a la ganadora (interacciones laterales) e inhibe a las alejadas
    - El uso de esta función de vecindad tiene 2 ventajas:
      - Ritmo de convergencia + rápido
      - El sistema es más robusto ante variaciones en los valores iniciales



- SOFM (Self-Organizing Feature Maps)
  - **Procedimiento general de aprendizaje**
    - Inicialización de pesos $\boldsymbol{w_{ijk}}$. Pueden ser nulos, aleatorios (pequeños) o con un valor determinado
    - En cada iteración ***t***
      - Presentación del patrón ***x(t)*** tomado al azar del conjunto de entrenamiento
      - Cada neurona calcula la similitud entre su vector de pesos $\boldsymbol{w_{ij}}$ y la entrada ***x***. **Ejemplo:** distancia euclídea

$$d^2(\boldsymbol{w_{ij}}, \boldsymbol{x}) = \sum_{k=1}^{n} \left(w_{ijk} - x_k\right)^2$$

      - Determinación de la **neurona ganadora**

- SOFM (Self-Organizing Feature Maps)
  - **Procedimiento general de aprendizaje *(cont.)***
    - En cada iteración ***t*** *(cont.)*
      - Actualización de los pesos de la neurona ganadora y vecinas. Ej:

$$w_{ijk}(t+1) = w_{ijk}(t) + \alpha(t)\, h(|\underbrace{\boldsymbol{i}}_{(i,j)} - \underbrace{\boldsymbol{g}}_{(m,n)}|, t)\left(x_k(t) - w_{ijk}(t)\right)$$

donde

- **α(t)** es el ritmo de aprendizaje,
- **h()** es la función de vecindad que depende de la distancia de las neuronas ***i*** y ***g*** (ganadora), que vale 0 cuando ***i*** no pertenece a la vencidad de ***g***, y un n° positivo cuando sí es vecina.
- α() y **h()** disminuyen monótonamente con t

    - Luego se puede realizar una segunda etapa de **ajuste fino** del mapa, tomando α() un valor pequeño constante y **h() = 1** solo para la ganadora
    - Entre 50 y 100 iteraciones por neurona suelen ser suficientes, pero idealmente deberían ser 500 por neurona

- SOFM (Self-Organizing Feature Maps)
  - **Procedimiento general de aprendizaje *(cont.)***
    - Ritmo de aprendizaje $\alpha()$
      - Son habituales 2 opciones:
        - decreciente linealmente

$$\alpha(t) = \alpha_0 + (\alpha_f - \alpha_0)\frac{t}{t_\alpha}$$

        - decreciente exponencialmente

$$\alpha(t) = \alpha_0 \left(\frac{\alpha_f}{\alpha_0}\right)^{t/t_\alpha}$$

donde α_0 es el ritmo inicial ($|\alpha_0|<1$), α_f es el ritmo final (~0.01) y t_α el máximo número de iteraciones para llegar a α_f

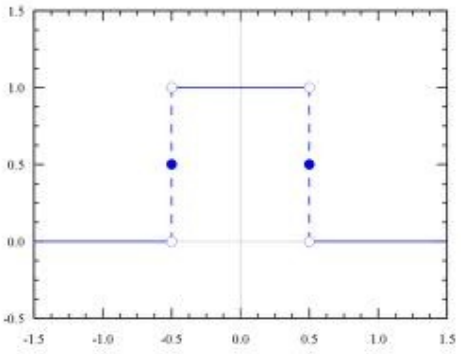
- SOFM (Self-Organizing Feature Maps)
  - **Procedimiento general de aprendizaje *(cont.)***
    - Función de vecindad $h(|\mathbf{i} - \mathbf{g}|, t)$
      - Usualmente es simétrica y centrada en la ganadora ***g*** por lo que la distancia de una neurona $\mathbf{i}=(i_i,i_j)$ a $\mathbf{g}=(g_i,g_j)$ generalmente se mide con la siguiente ecuación

$$|\boldsymbol{i}-\boldsymbol{g}|=\sqrt{(i_i-g_i)^2+(i_j-g_j)^2}$$

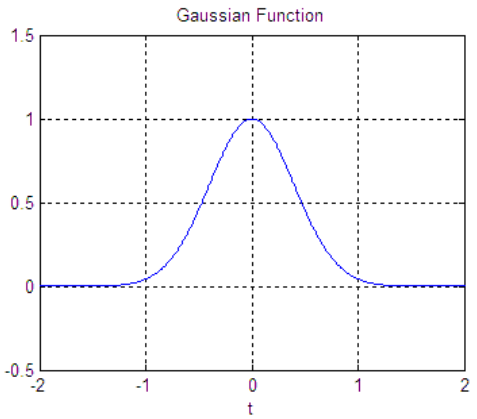
      - La función **h()** puede tomar varias formas, que dependen de una función R(t) que determina el **radio** de "vecindad"

      - Una función simple es la escalón

$$h(|\boldsymbol{i}-\boldsymbol{g}|,t)=\begin{cases}0, & si \quad |\boldsymbol{i}-\boldsymbol{g}|>R(t)\\ 1, & si \quad |\boldsymbol{i}-\boldsymbol{g}|\leq R(t)\end{cases}$$



- SOFM (Self-Organizing Feature Maps)
  - **Procedimiento general de aprendizaje**
    - Función de vecindad $h(|\mathbf{i} - \mathbf{g}|, t)$
      - Otras fuciones pueden tomar otras formas, como ser formas gaussianas



      - También pueden adoptar formas cóncavas o convexas
      - La función de vecindad en sí misma no cambia con el tiempo, pero sí la función que define el radio R(t), la cual generalmente disminuye monótonamente
        - Ej: disminución lineal

$$R(t) = R_0 + (R_f - R_0)\frac{t}{t_R}$$

donde t es la iteración, y t_R el número de iteraciones para alcanzar R_f

- SOFM (Self-Organizing Feature Maps)
  - **Consideraciones prácticas**
    - Los SOFM tal como se han descripto anteriormente, pueden utilizarse para análisis de datos (exploración de datos, monitorización de procesos, etc) y mapping (cuantificación vectorial, reducción de dimensiones, etc)
    - Puede también adaptarse la técnica para resolver problemas de ***clasificación***. Ej: utilizando el ajuste fino LVQ (*Learning Vector Quantization*)
    - Existen técnicas complementarias enfocadas en
      - Tamaño adaptativo de los mapas
      - Solución de la asimietría (“desventaja”) de neuronas en los bordes del mapa
      - Procesamiento de series de tiempo
      - Aplicación en redes híbridas: multi-capas con partes supervisadas + partes no supervisadas (Ej: red de contrapropagación, RBF, etc)

- SOFM (Self-Organizing Feature Maps)
  - Métricas de similitud
    - Correlación o producto escalar

$$C_{ij} = \sum_{k=1}^{n} w_{ijk}\, x_k$$

      - Conicide con el modelo de neurona estándar (suma ponderada)
      - Desventaja: sensible al tamaño de los vectores (en general requiere vectores normalizados para funcionar bien)

    - Coseno

$$\cos(\boldsymbol{w_{ij}}, \boldsymbol{x}) = \frac{\sum_{k=1}^{n} w_{ijk}\, x_k}{\|\boldsymbol{w_{ij}}\| . \|\boldsymbol{x}\|}$$

      - Se deriva de la correlación, y su ventaja es que es independiente del tamaño de los vectores

- SOFM (Self-Organizing Feature Maps)
  - Métricas de similitud
    - Distancia euclídea

$$d^2(\boldsymbol{w_{ij}}, \boldsymbol{x}) = \sum_{k=1}^{n} \left(w_{ijk} - x_k\right)^2$$

    - Distancia de Manhattan

$$d(\boldsymbol{w_{ij}}, \boldsymbol{x}) = \sum_{k=1}^{n} |w_{ijk} - x_k|$$

      - Ventaja: muy simple computacionalmente, lo que la hace muy rápida (no requiere potenciación, radicación ni multiplicaciones)

- SOFM (Self-Organizing Feature Maps)
  - Reglas de aprendizaje para las métricas de similitud más usadas
    - Aprendizaje para la distancia euclídea

$$\boldsymbol{w}_i(t+1)=\boldsymbol{w}_i(t)+\alpha(t)\,h(|\boldsymbol{i}-\boldsymbol{g}|,t)\,(x_k(t)-w_{ijk}(t))$$

    - Aprendizaje para la distancia Manhattan

$$\boldsymbol{w}_i(t+1)=\boldsymbol{w}_i(t)+\alpha(t)\,h(|\boldsymbol{i}-\boldsymbol{g}|,t)\,signo\,(x_k(t)-w_{ijk}(t))$$

      o bien

$$\boldsymbol{w}_i(t+1)=\boldsymbol{w}_i(t)+\Delta\,w_{ijk}(t)$$

$$con\,\Delta\,w_{ijk}(t)=\begin{cases}+\alpha(t)\,h(|\boldsymbol{i}-\boldsymbol{g}|,t) & si\,x_k(t)>w_{ijk}(t)\\ 0 & si\,x_k(t)=w_{ijk}(t)\\ -\alpha(t)\,h(|\boldsymbol{i}-\boldsymbol{g}|,t) & si\,x_k(t)<w_{ijk}(t)\end{cases}$$

- SOFM (Self-Organizing Feature Maps)
  - Reglas de aprendizaje
    - La regla de aprendizaje debe ser compatible con la métrica de similitud para que el mapa se desarrolle correctamente
    - Procedimiento sistemático para derivar reglas de aprendizaje
      - Sea $d(\boldsymbol{w}_{ij}, \boldsymbol{x})$ una función de distancia genérica, derivable
      - La neurona ganadora $\boldsymbol{g}$ cumple

$$\boldsymbol{g} = \arg\min_{ij} \left\{ d\left(\boldsymbol{w}_{ij}, \boldsymbol{x}\right) \right\}$$

      - Se define una función de error que pretende representar la diferencia entre los pesos sinápticos y las entradas del espacio sensorial

- SOFM (Self-Organizing Feature Maps)
  - Reglas de aprendizaje
    - Procedimiento sistemático para derivar reglas de aprendizaje (cont.)
      - Una **aproximación** a tal función **global** de error es

$$E'(t) = \sum_i h(|\boldsymbol{i} - \boldsymbol{g}|, t)\, f(d(\boldsymbol{x}(t), \boldsymbol{w_i}(t)))$$

con **f()** una función, usualmente identidad, incluida por generalidad

      - Aplicando el mecanismo de descenso por el gradiente, se pueden derivar actualizaciones de peso dependientes de f(), d() y h()

$$\boldsymbol{w_i}(t+1) = \boldsymbol{w_i}(t) - \lambda(t) \nabla_{w_i} E'(t)$$

con $\lambda(t)$ ritmo de aprendizaje

- SOFM (Self-Organizing Feature Maps)
  - **LVQ (*Learning Vector Quantization*)**
    - Permite ajuste fino de un mapa para aplicaciones de clasificación de patrones
    - Cada neurona o grupo de neuronas representa una ***clase***, y cada una almacena un vector de pesos $\boldsymbol{w_{ij}}$ que representa el ***prototipo*** de una clase
    - Aprendizaje: para un patrón μ dado
      - Si la neurona ganadora representa la clase del patrón, se la **premia** ajustando sus pesos **hacia** el patrón (similar al mapa de Kohonen tradicional)
      - Si la neurona ganadora no representa la clase del patrón, se la **castiga** ajustando sus pesos en el **sentido contrario** al del patrón

- SOFM (Self-Organizing Feature Maps)
  - **LVQ (*Learning Vector Quantization*)**
    - Aprendizaje: ajuste de pesos
      Suponiendo un patrón de entrada $\mathbf{x}^{\mu}$ que pertenece a una clase $C_{\mathbf{x}\mu}$ y una neurona ganadora $\mathbf{w}_c$ que representa una clase C_{wc}
      - Si $C_{wc} = C_{\mathbf{x}\mu}$ (clasificación correcta) los pesos de la neurona ganadora se actualizan con
        $$\boldsymbol{w}_c(t+1) = \boldsymbol{w}_c(t) + \alpha_t\left[\boldsymbol{x}^{\mu} - \boldsymbol{w}_c(t)\right]$$
      - Si $C_{wc} \neq C_{\mathbf{x}\mu}$ (clasificación incorrecta) los pesos de la neurona ganadora se actualizan con
        $$\boldsymbol{w}_c(t+1) = \boldsymbol{w}_c(t) - \alpha_t\left[\boldsymbol{x}^{\mu} - \boldsymbol{w}_c(t)\right]$$
      - El resto de los pesos no se modifican
      - Para evitar overfitting se puede usar parada temprana
      - Una cantidad de iteraciones de entre 50 y 200 veces la cantidad de neuronas suele ser suficiente para el entrenamiento
      - Inicialización del mapa: (a) un ejemplo de la clase; (b) k-means; (c) entrenamiento SOFM tradicional (sin ajuste fino)