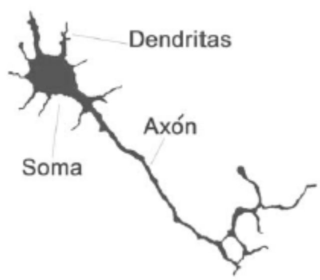
Redes Neuronales

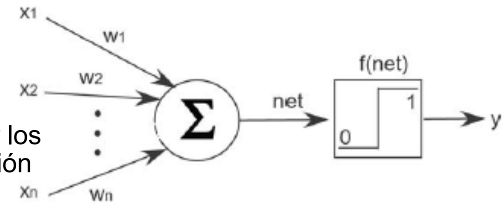
Modelo computacional

- Neurona biológica:

1. **Entradas: dendritas.**
2. **Integración: en el soma. Dispositivos "todo-o-nada" (solo se dispara salida si las entradas superan un nivel (umbral)**
3. **Salidas: el axón transporta la señal de salida a otras neuronas. Conecta con sus dendritas a través de sinapsis.**



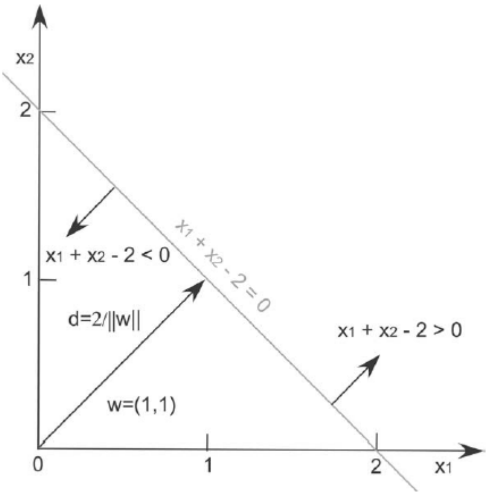
- Neurona computacional:
- Entradas: Números reales.
- Integración: suma ponderada (net) por los pesos sinápticos seguida de una función de activación f(net)
- Salida: resultado y=f(net)



Interpretación geométrica
Neuronas e hiperplanos

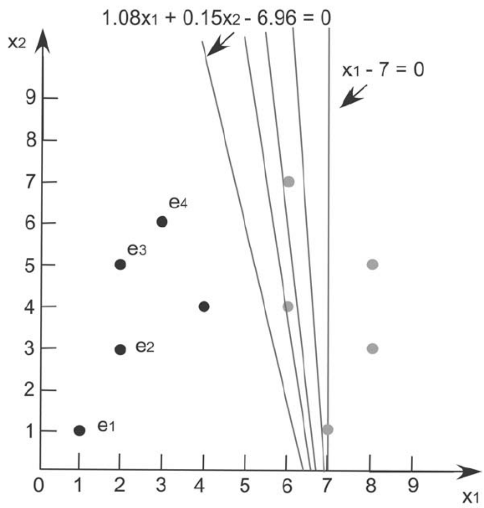
- Función de activación "umbral":
- Interpretación geométrica:
  Considerando que el umbral es un peso más con entrada fija de **-1**, la neurona define un **hiperplano** de forma que los ejemplos etiquetados con **y=1** caen al lado positivo y los etiquetados con **y=0** al lado negativo:

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$



Entrenamiento

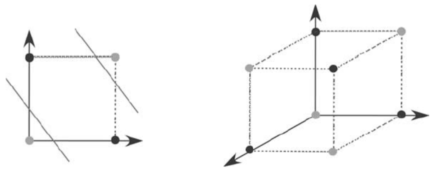
- Ajuste de hiperplanos: Dados dos conjuntos de ejemplos correspondientes a dos clases, buscaremos su separación por un hiperplano
- Regla delta:
  - Permite ajustar iterativamente el hiperplano.
  - Se asume que el incremento de los pesos es proporcional a la disparidad entre la **salida observada** y la **salida deseada**.
  - Dicha proporcionalidad viene modulada por la **constante de aprendizaje**:

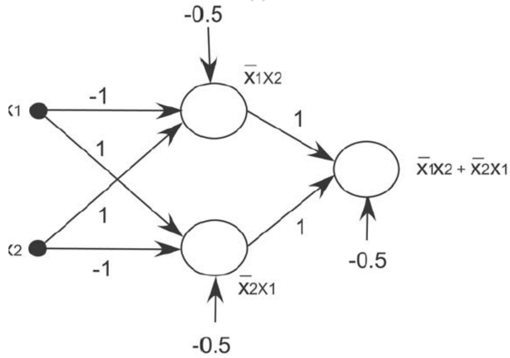


$$\Delta w_i = \eta(d-y)x_i \text{ o sea } w_i = w_i + \eta(d-y)x_i,$$

No-separabilidad lineal

- Única neurona:
  - Existen situaciones en donde un único hiperplano no puede separar los datos.
  - P.ej. cuando la frontera de decisión es curva.
- Problemas de paridad:
  - Suponiendo entradas binarias (secuencias de 0s y 1s), la neurona debería **etiquetar con 1 aquellas secuencias con un número impar de 1s** y con 0 aquellas con un número par.
  - Ej: Problema de la XOR.
  - Para resolver estos problemas es **preciso incorporar una capa adicional** de neuronas.

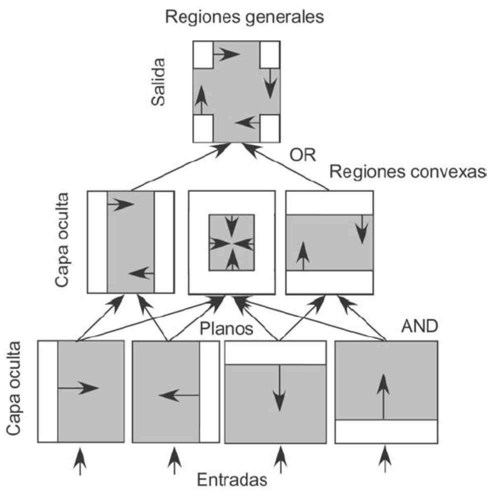




Dpnt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

Perceptrones multi-capa

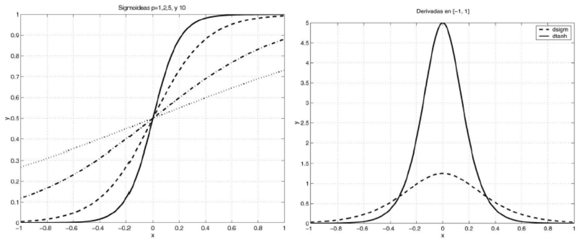
- Estructura y capacidad:
  - La capa adicional se denomina **capa oculta.**
  - Se demuestra que un perceptrón con dos capas ocultas puede **aproximar cualquier función.**
- Interpretación geométrica:
  - Problemas con regiones de decisión más complejas exigen distintas estrategias de separación.
  - Dichas estrategias las proporcionan las capas ocultas.
  - En la época en que se desarrolló esta teoría no existía un algoritmo práctico que permitiese encontrar los pesos asociados a todas y cada una de las neuronas.



Funciones de activación derivables

- Para aplicar el algoritmo de entrenamiento multicapa es necesario que la función de activación sea **derivable**
- Buscamos funciones derivables con forma **similar al escalón** del perceptrón de una sola capa

$$f(x,p) = \frac{1}{1+e^{-xp}}. \qquad f'(x,p) = \frac{\partial f(x,p)}{\partial x} = \frac{pe^{-xp}}{(1+e^{-xp})^2} = py(1-y).$$



Sigmoide

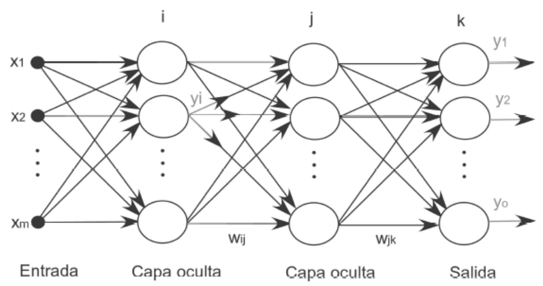
Derivada

Backpropagation: explicación heurística

- Supongamos que al clasificar un ejemplo una neurona de la última capa tiene una salida $\mathbf{y_k}$, siendo la deseada $\mathbf{d_k}$
- Dicha neurona es responsable de un error

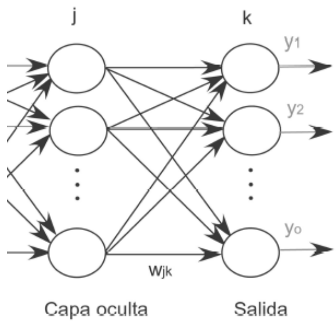
$$\delta_k = (d_k - y_k)f'(net_k),$$

- La regla de actualización de los pesos de la última capa será similar a la regla delta ya vista

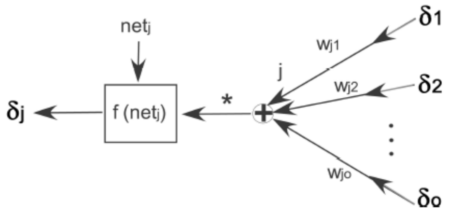


Error (delta) en capas intermedias

- Una neurona de una capa intermedia contribuye en los δ de las de la capa siguiente
- Por tanto, para calcular su δ necesitamos estos



$$\delta_j = f'(net_j)\sum_k w_{jk}\delta_k$$



Dpnt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

Backpropagation: algoritmo

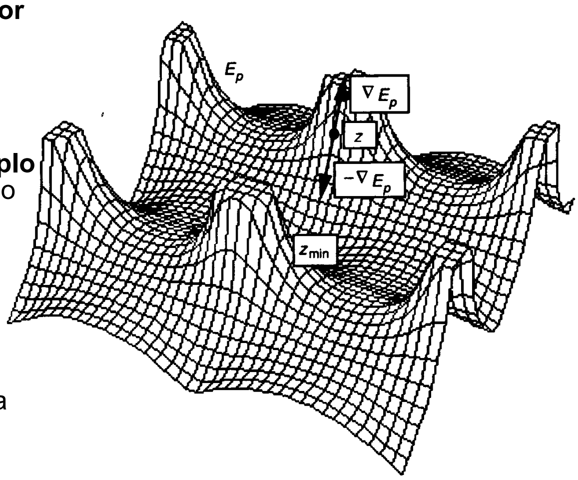
- Se aplica para cada ejemplo del conj. de entrenamiento. Se itera hasta que el error baje de un umbral
- Fases:
  - Hacia delante: cálculo de la salida de la red (los y_k). Cálculo de los δ en la última capa
  - Hacia atrás. Cálculo de los δ de la capa en función de los de la siguiente
  - Finalmente, actualización de los pesos de todas las capas

Algoritmo BACKPROPAGATION(red $ejemplos, \eta$) {
  $\{w_{ij}\} \leftarrow$ INICIALIZAR;
  Mientras $\neg$ CONVERGENCIA(red) **Hacer** {
    $e \leftarrow$ SELECCIONAREJEMPLO($ejemplos$);
    $\{y_k\} \leftarrow$ FORWARD(e);
    $\{d_k\} \leftarrow$ DESEADAS(e);
    Para cada $n_k \in$ CAPA(red, k) **Hacer** {
      $\delta_k = (d_k - y_k) f'(net_k)$;
    }
    Para $j = k - 1$ **hasta** 1 **Hacer** {
      Para $n_j \in$ CAPA(red, j) **Hacer** {
        $\delta_j = f'(net_j) \sum_{j+1} \delta_{k+1} w_{j(j+1)}$;
      }
    }
    Para $j = k$ **hasta** 1 **Hacer** {
      $w_{(j-1)j} = w_{(j-1)j} + \eta \delta_j y_{(j-1)}$;
    }
    $red \leftarrow$ ACTUALIZARRED($\{w_{ij}\}$);
  }
  Devolver red;
}

Backpropagation: derivación matemática

- El algoritmo es un **descenso por gradiente** (de ahí que necesitemos una función de activación derivable)
- Minimización del **error que se produce al clasificar un ejemplo** (encontrar los w_i óptimos). Dicho error se puede formular como

$$E = \frac{1}{2}\sum_k (d_k - y_k)^2$$



- Habrá que modificar los w_i en la dirección opuesta al **gradiente**, esto es

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

Derivación *backpropagation*: última capa

- Para un peso $\mathbf{w_{jk}}$ de la última capa (aplicando la regla de la cadena, ya que E depende de net_k que a su vez depende de $\mathbf{w_{jk}}$)

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial w_{jk}} = -\eta \frac{\partial E}{\partial net_k}\frac{\partial \sum w_{jk} y_j}{\partial w_{jk}} = -\eta \frac{\partial E}{\partial net_k} y_j$$

- Escribiendo $-\frac{\partial E}{\partial net_k}$ como δ, tenemos una fórmula equiv. a la regla delta del perceptrón de 1 capa:
- Para calcular δ_k, aplicamos de nuevo la regla de la cadena ($net_k \rightarrow y_k \rightarrow E$)

$$-\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial net_k} = -\frac{\partial E}{\partial y_k}\frac{\partial f(net_k)}{\partial net_k} = -\frac{\partial E}{\partial y_k} f'(net_k)$$

$$\frac{\partial}{\partial y_k}\frac{1}{2}\sum (d_k - y_k)^2 = -(d_k - y_k)$$

Derivación *backpropagation*: capas anteriores

- Para un peso w_{ij} de una capa anterior, el razonamiento inicial es el mismo que antes y nos lleva a

$$\Delta w_{ij} = \eta \delta_j y_i$$

- Aplicando la regla de la cadena para calcular δ

$$\delta_j = -\frac{\partial E}{\partial net_j} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j} \longrightarrow f'(net_j)$$

$$\sum_k \left(\frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial y_j}\right) = \sum_k \left(\frac{\partial E}{\partial net_k}\frac{\partial\left(\sum w_{jk} y_j\right)}{\partial y_j}\right)$$

$$= \sum_k \left(\frac{\partial E}{\partial net_k} w_{jk}\right) = -\sum_k \delta_k w_{jk}$$

Inicialización de la red

- Inicialización de los pesos de la red:
  - Arbitraria
  - Aleatoria
- **Problema** de la inicialización en los descensos por gradiente: mínimos locales
- **Solución**: entrenar la red desde distintas inicializaciones

Convergencia de backpropagation

- Una red neuronal **converge** cuando el error de validación se mantiene bajo y los ejemplos de entrenamiento no provocan cambios significativos en los pesos de la red.
- Ajuste de la constante η
  - Valores muy pequeños: convergencia lenta
  - Valores grandes: riesgo de *overshooting* (saltarnos el mínimo)

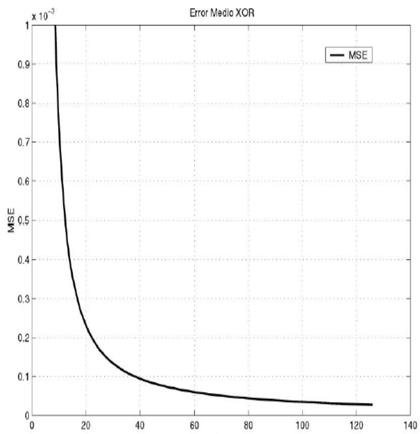
Momento: en el cambio actual influye una fracción del anterior. Mejora la convergencia evitando oscilaciones

$$w_{ij}{}^{t} = w_{ij}{}^{t} + \eta \delta_j y_i + \alpha \Delta w_{ij}{}^{(t-1)}$$

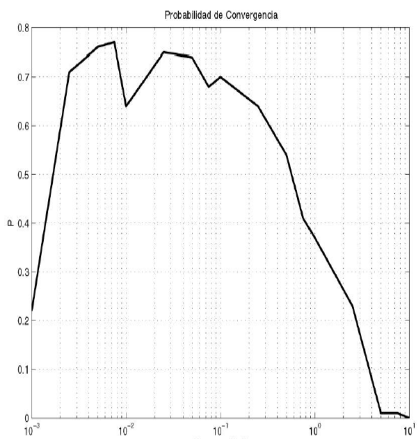
Dpnt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

Ejemplo de convergencia

- Problema del XOR



iteraciones



η

Dpnt. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial
Universitat d'Alacant
Universidad de Alicante

Bibliografía

- Escolano et al. Inteligencia Artificial. Thomson-Paraninfo 2003. Capítulo 4.
- Mitchell, Machine Learning. McGraw Hill, Computer Science Series. 1997
- Reed, Marks, Neural Smithing. MIT Press, CA Mass 1999