

# Redes neuronales (Parte II: Modelos)

Inteligencia artificial

---

Patricio García Báez  
Grado en Ingeniería Informática

# Índice

- Perceptrón Simple
- Perceptrón Multicapa (MLP)
- Otros modelos

# Modelos: Perceptrón Simple

## – Características

- Autor más representativo: Rosenblat (1962)
- RN de una capa de cómputo
- E reales / S bipolares o binarias
- Modelo de Neurona

### – Función de suma:

$$net_j = b_j + \sum_i x_i w_{ij}$$

### – Función de activación: bipolar o binaria

$$f(net_j) = \begin{cases} 1 \Rightarrow net_j > \phi \\ 0 \Rightarrow -\phi \leq net_j \leq \phi \\ -1 \Rightarrow net_j < -\phi \end{cases} \quad f_{act}(net_j) = \begin{cases} 1 \Rightarrow net_j > 0 \\ 0 \Rightarrow \text{otro caso} \end{cases}$$

# Modelos: Perceptrón Simple

## – Regla Aprendizaje

- Supervisado mediante Corrección de Error
- Ajuste de pesos:

$$\Delta w_{ij} = \alpha \cdot e_j \cdot x_i$$

- Donde  $e_j$  es el error ajustado a  $[-1,0,1]$  y  $\alpha$  el ratio de aprendizaje

# Modelos: Perceptrón Simple

## – Algoritmo Aprendizaje

- Inicializar pesos y bias  $\in \{-1,0,1\}$ ,  $0 < \alpha \leq 1,0$
- Repetir

– Para cada patrón

» Calcular salida de cada neurona  $j$

$$net_j = b_j + \sum_i x_i w_{ij}$$

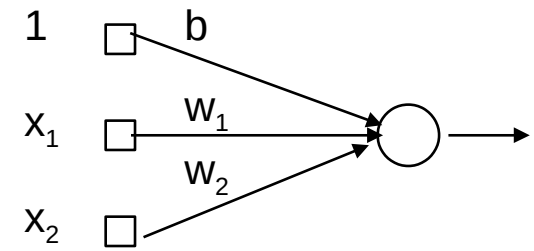
$$x_j = f(net_j) = \begin{cases} 1 \Rightarrow net_j > \phi \\ 0 \Rightarrow -\phi \leq net_j \leq \phi \\ -1 \Rightarrow net_j < -\phi \end{cases}$$

» Modificar pesos de cada conexión  $ij$

$$\Delta w_{ij} = \alpha \cdot e_j \cdot x_i$$

- Hasta no se modifiquen pesos en bucle anterior

Perceptrón para función AND



# Modelos: Perceptrón Simple

## – Ejemplo de evolución para función AND

Entrada ( $x1, x2, I$ )	Net	Salida	Objetivo	Cambio W ( $\Delta W$ )	W Actuales ( $w1, w2, b$ )	$\alpha=1, \phi=0.2$
					(0 0 0)	
(1 1 1)	0	0	1	(1 1 1)	(1 1 1)	
(1 0 1)	2	1	-1	(-1 0 -1)	(0 1 0)	
(0 1 1)	1	1	-1	(0 -1 -1)	(0 0 -1)	
(0 0 1)	-1	-1	-1	(0 0 0)	(0 0 -1)	

Resultados de la segunda pasada:

(1 1 1)	-1	-1	1	(1 1 1)	(1 1 0)
(1 0 1)	1	1	-1	(-1 0 -1)	(0 1 -1)
(0 1 1)	0	0	-1	(0 -1 -1)	(0 0 -2)
(0 0 1)	-2	-1	-1	(0 0 0)	(0 0 -2)

Resultados de la tercera pasada:

(1 1 1)	-2	-1	1	(1 1 1)	(1 1 -1)
(1 0 1)	0	0	-1	(-1 0 -1)	(0 1 -2)
(0 1 1)	-1	-1	-1	(0 0 0)	(0 1 -2)
(0 0 1)	-2	-1	-1	(0 0 0)	(0 1 -2)

# Modelos: Perceptrón Simple

## — Ejemplo de evolución para función AND

Entrada	Net	Salida	Objetivo	Cambio W	W Actuales	$\alpha=1, \phi=0.2$
$(x1, x2, I)$				$(\Delta W)$	$(w1, w2, b)$	

Resultados de la cuarta pasada:

(1 1 1)	-1	-1	1	(1 1 1)	(1 2 -1)
(1 0 1)	0	0	-1	(-1 0 -1)	(0 2 -2)
(0 1 1)	0	0	-1	(0 -1 -1)	(0 1 -3)
(0 0 1)	-3	-1	-1	(0 0 0)	(0 1 -3)

Continuamos iterando.....

Resultados de la pasada décima en la que desaparece el error:

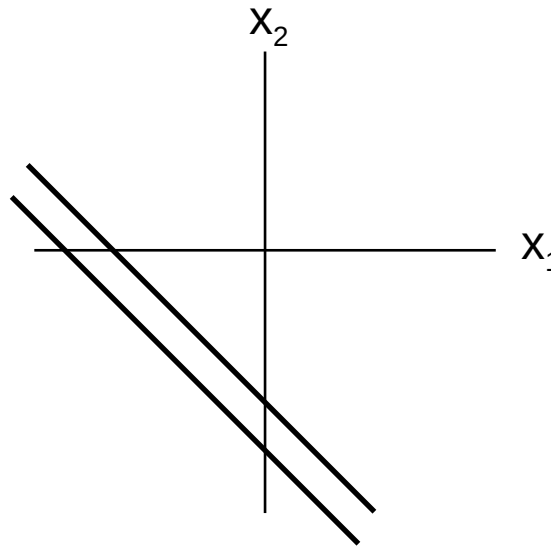
(1 1 1)	1	1	1	(0 0 0)	(2 3 -4)
(1 0 1)	-2	-1	-1	(0 0 0)	(2 3 -4)
(0 1 1)	-1	-1	-1	(0 0 0)	(2 3 -4)
(0 0 1)	-4	-1	-1	(0 0 0)	(2 3 -4)

# Modelos: Perceptrón Simple

- Separabilidad lineal

$$w_{1j}x_1 + w_{2j}x_2 + b > \phi$$

$$w_{1j}x_1 + w_{2j}x_2 + b < -\phi$$





# Modelos: Perceptrón Simple

## – Teorema de Convergencia

- Si existe un vector de pesos  $w^*$  tal que para todo patrón  $p$  perteneciente al conjunto de patrones de entrenamiento  $P$ ,  $f_{act}(x(p) \cdot w^*) = d(p)$ , entonces para cualquier  $w$  vector de pesos inicial el aprendizaje del perceptrón acabará convergiendo en un número finito de pasos a un vector de pesos (no necesariamente único ni necesariamente  $w^*$ ) que da la respuesta correcta a todos los patrones de entrenamiento

# Modelos: Perceptrón Simple

- Discusión
  - Según Minsky y Papert (1969,1988)
    - Limitaciones
      - » Separabilidad lineal
      - » Concavidad, convexidad
      - » Problemas clásicos: xor, paridad
      - » Incapacidad de generalización global en base a ejemplos localmente aprendidos
    - Conjetura de que estas limitaciones se extienden a los multicapas
    - Conjetura no justificada, véase algoritmos multicapas backpropagation y funciones de base radial

# Modelos: Perceptrón Multicapa (MLP)

## – Características

- Autores: Invención independiente de: Bryson, 1969; Werbos, 1974; Parker, 1985; Lecun, 1985; Rumelhart, Hinton, Williams, 1986
- Soporta una o más capas ocultas
- E binarias o reales / S en principio binarias, aunque las salidas se aproximan por una función continua no lineal para clasificación
- Modelo de Neurona

– Función de suma:

$$net_j = b_j + \sum_i x_i w_{ij}$$

– Función de activación: no lineal (sigmoide, ReLU), también lineal o SoftMax en salida

$$f_{act}(net) = \frac{1}{1 + \exp(-net)} \quad f_{act}(net) = \max(0, net)$$

# Modelos: Perceptrón Multicapa (MLP)

- Regla de aprendizaje
  - Supervisado mediante Corrección de Error
  - Ajuste de pesos por Retropropagación del Error (*Backpropagation*)

$$\Delta w_{ij} = \alpha \cdot \delta_j \cdot x_i$$

- Donde  $\delta_j$  es un coeficiente de error calculado para cada capa y  $\alpha$  el ratio de aprendizaje

# Modelos: Perceptrón Multicapa (MLP)

## – Algoritmo Aprendizaje (Backpropagation)

- Inicializar pesos y bias
- Repetir

### – Para cada patrón

- » Para cada capa (desde la 1ª a la última)
- » Calcular salida de cada neurona  $j$

$$net_j = b_j + \sum_i x_i w_{ij} \quad f_{act}(net_j)$$

- » Para cada conexión  $ij$  de la capa de salida

$$\delta_j = f'_{act}(net_j) \cdot e_j$$

$$\Delta w_{ij} = \alpha \cdot \delta_j \cdot x_i$$

- » Para cada conexión  $ij$  de capas anteriores

$$\delta_j = f'_{act}(net_j) \cdot \sum_{k=1}^m \delta_k \cdot w_{jk}$$

$$\Delta w_{ij} = \alpha \cdot \delta_j \cdot x_i$$

- Hasta error pequeño ó alcancemos nmi

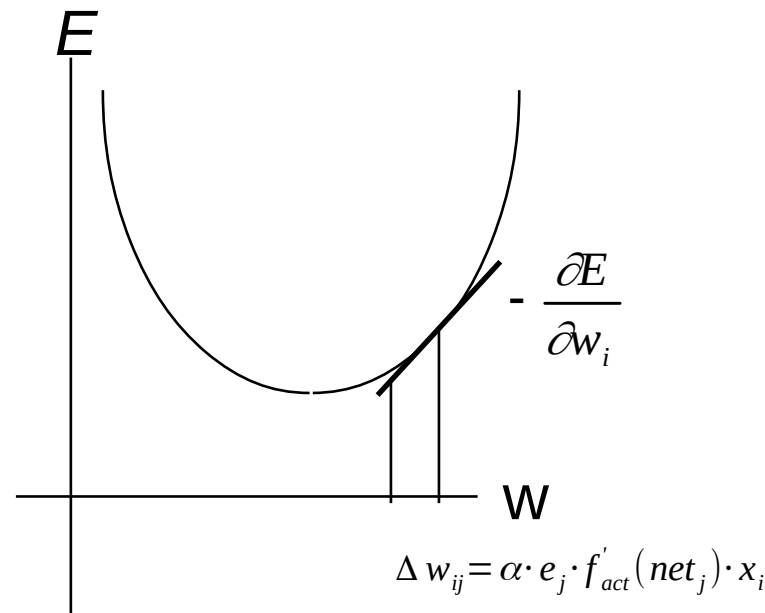
# Modelos: Perceptrón Multicapa (MLP)

- Obtención de la regla

- Error a minimizar:

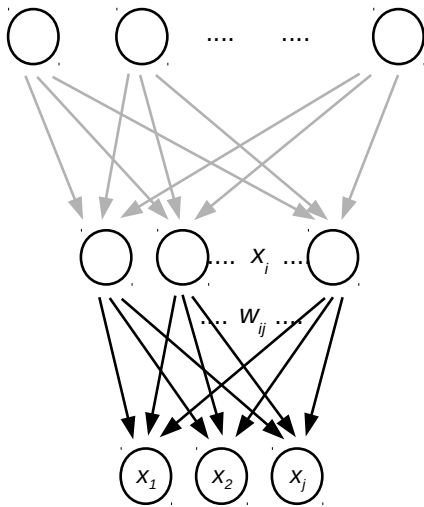
$$E = \frac{1}{2} \cdot \sum_k (d_k - x_k)^2 = \frac{1}{2} \cdot \sum_k e_k^2$$

- Minimización bajo la derivada:



# Modelos: Perceptrón Multicapa (MLP)

- Obtención de la regla
  - Para última capa:



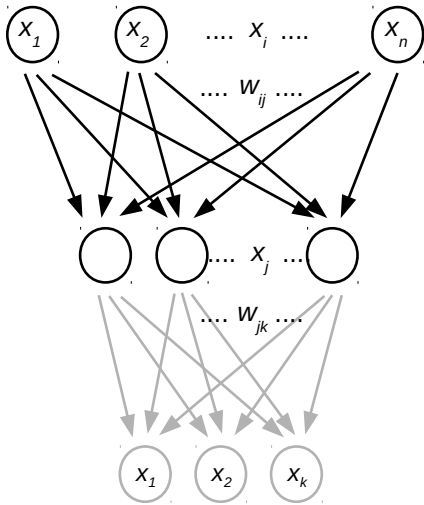
$$\begin{aligned}\frac{\partial E}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}} \frac{1}{2} \cdot \sum_k (d_k - x_k)^2 = \\ &= \frac{\partial}{\partial w_{ij}} \frac{1}{2} \cdot \sum_k (d_k - f_{act}(net_k))^2 = \\ &= -(d_j - x_j) \cdot \frac{\partial}{\partial w_{ij}} f_{act}(net_j) = \\ &= -(d_j - x_j) \cdot f'_{act}(net_j) \cdot \frac{\partial}{\partial w_{ij}} net_j = \\ &= -e_j \cdot f'_{act}(net_j) \cdot x_i = -\delta_j \cdot x_i\end{aligned}$$

donde:

$$\delta_j = e_j \cdot f'_{act}(net_j)$$

# Modelos: Perceptrón Multicapa (MLP)

- Obtención de la regla
  - Para capas ocultas:



$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \sum_k \frac{\partial E}{\partial x_k} \frac{\partial}{\partial w_{ij}} x_k = -\frac{1}{2} \cdot 2 \cdot \sum_k (d_k - x_k) \cdot \frac{\partial}{\partial w_{ij}} x_k \\ &= -\sum_k (d_k - x_k) \cdot f'_{act}(net_k) \cdot \frac{\partial}{\partial w_{ij}} net_k = \end{aligned}$$

$$= -\sum_k e_k \cdot f'_{act}(net_k) \cdot \frac{\partial}{\partial w_{ij}} net_k = -\sum_k \delta_k \cdot \frac{\partial}{\partial w_{ij}} net_k$$

$$\text{donde : } \delta_k = e_k \cdot f'_{act}(net_k)$$

entonces :

$$- \sum_k \delta_k \cdot \frac{\partial}{\partial w_{ij}} net_k = - \sum_k \delta_k \cdot \frac{\partial net_k}{\partial x_j} \frac{\partial}{\partial w_{ij}} x_j =$$

$$= - \sum_k \delta_k \cdot w_{jk} \cdot \frac{\partial}{\partial w_{ij}} x_j =$$

$$= - \sum_k \delta_k \cdot w_{jk} \cdot f'_{act}(net_j) \cdot \frac{\partial}{\partial w_{ij}} net_j =$$

$$= - \sum_k \delta_k \cdot w_{jk} \cdot f'_{act}(net_j) x_i = - \delta_j \cdot x_i$$

donde :

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



# Modelos: Perceptrón Multicapa (MLP)

## – Elecciones

- Número de capas ocultas
  - Una capa oculta suele bastar
  - Más capas pueden facilitar la tarea
- Número de neuronas ocultas
  - Cuanto menos mejor:
    - » Aprendizaje más rápido
    - » Mayor capacidad de generalización
    - » Menor posibilidad de parálisis
  - Técnicas de eliminación y/o creación de neuronas
- Funciones de activación
  - En capas ocultas: no lineales
  - En capa de salida: de acuerdo a los valores deseados

# Modelos: Perceptrón Multicapa (MLP)

## – Elecciones

- Número de pares de entrenamiento
  - Según Baum-Haussler (1989) con  $P$ =número de patrones,  $W$ =número de pesos y  $\varepsilon$  fracción de error  $\Rightarrow P > W / \varepsilon$
  - *Shuffled* de patrones
- Inicialización de pesos y bias
  - Influyen en alcanzar el error global o local y la velocidad con que se alcanza
  - Si se dan valores muy grandes  $\Rightarrow$  caída en valores de derivada pequeños (saturación)
  - Si se dan valores muy pequeños  $\Rightarrow$  salidas próximas a 0  $\Rightarrow$  lentitud
  - Experimentalmente en intervalo:

$$\left[ -0.5, 0.5 \right] \text{ o } \left[ -\frac{2.4}{\sqrt{fanin}}, \frac{2.4}{\sqrt{fanin}} \right] \text{ o } \left[ -\frac{1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}} \right]$$

# Modelos: Perceptrón Multicapa (MLP)

## – Elecciones

- Ratios de aprendizaje
  - Grande  $\Rightarrow$  parálisis o inestabilidad
  - Pequeños  $\Rightarrow$  lentitud en aprendizaje
  - Ratios adaptivos (RMSProp, Hinton (2012))
- Función de pérdida a minimizar
  - Error cuadrático medio (MSE de *Mean Squared Error*)
  - Cross-entropy loss
- Tiempo de aprendizaje
  - Hasta que el error sea menor que cota definida a priori
  - Hasta que pendiente de error sea menor que cota definida a priori
  - Hasta que pendiente de pesos sea menor que cota definida a priori
  - Jugar con 2 conjuntos, mientras no empeore el error de testeo (*Early stopping*)

# Modelos: Perceptrón Multicapa (MLP)

## – Variantes

- Momentum: Rumelhart, Hinton, Williams (1986)
  - Añade un término proporcional al último  $\Delta w$ 
$$\Delta(t+1)w_{ij} = \alpha \cdot \delta_j \cdot x_i + \eta \cdot \Delta(t)w_{ij}$$
- Quickprop: Fahlman (1988)
  - Utiliza segunda derivada para estimar mejor la posición del mínimo
- Adam: Diederik (2014)
  - Combinación de RMSProp y Momentum
- Regularización
  - Previene el sobreajuste (*Dropout Layers*)
- Batch
  - Acumula los  $\Delta w$  y los aplica al final

# Modelos: Perceptrón Multicapa (MLP)

## – Desventajas

- Convergencia no demostrada
- Lento proceso de aprendizaje
- Incertidumbre en la convergencia debido a:
  - Parálisis de red
  - Mínimos locales

# Otros modelos

- **GAN** (*Generative Adversarial Network*)
  - Sistema de dos RNs que compiten mutuamente en juego de suma cero
- **LSTM** (*Long Short-Term Memory*)
  - RN recurrente que abordar el problema de desvanecimiento de gradiente para aprender dependencias a largo plazo en secuencias de datos
- **Transformer**
  - RN que incorpora mecanismo de autoatención, dando un peso diferente a cada parte de la entrada
- **CNN** (*Convolutional Neural Network*)
  - Incorpora filtros convolucionales mas reducciones de muestreo antes de capa de clasificación









# Créditos

Esta presentación está bajo una licencia

Creative Commons Attribution-ShareAlike 4.0 International License



# Referencias

-  *Introduction to the theory of neural computation.* Hertz, Krogh, Palmer. Addison-Wesley. 1991
-  *Fundamentals of Neural Networks. Architectures, algorithms, and applications.* Fausett. Prentice-Hall. 1994
-  *Neural Networks. A comprehensive foundation.* Haykin. Macmillan. 1994
-  *Neurocomputing.* Hecht-Nielsen. Addison-Wesley. 1989
-  *Neural Computing. Theory and Practice.* Wasserman. Van Nostrand. 1989
-  *Descripción Formal de Modelos de Redes Neuronales.* J.R. Álvarez. IX Cursos de Verano de la UNED. 1998
-  *Neural Netowrks for Pattern Recognition.* C.M. Bishop. 1995
-  *Deep Learning.* I. Goodfellow, Y. Bengio, A. Courville. MIT Press. 2016