Introducción a Machine Learning

Aprendizaje de Máquina – Redes Neuronales Artificiales

MSc. Marco Sobrevilla

Objetivo

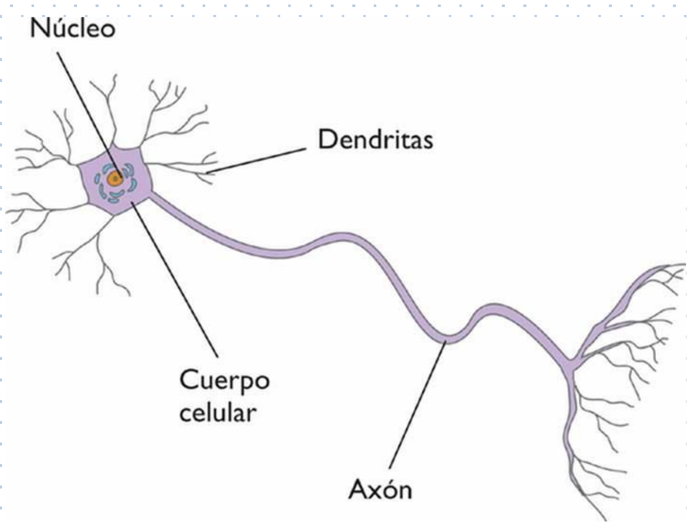
- Aprender conceptos sobre Redes Neuronales Artificiales
  - Perceptron
  - *Multilayer Perceptron*

Agenda

- Introducción
- Perceptron
- Perceptron Multicapa

Neurona Biológica

- Cerebros tienen millones de neuronas que procesan informacion
  - Red Neuronal Biológica
- Neurona es un simple procesador y la interacción con otras así como su procesamiento en paralelo otorgan las habilidades del cerebro



Introducción

- ¿Red Neuronal Artificial?
  - Basada en el concepto de las neuronas biológicas
  - Son programadas para simular el comportamiento de una neurona biológica

Red Neuronal Artificial (RNA)

- Sistema de procesamiento de información que tiene ciertas aptitudes en común con las redes neuronales biológicas:
  - *Procesamiento de Información: neuronas*
  - *Señales transferidas entre neuronas a través de conexiones*
  - *Cada conexión tiene un peso asociado que multiplica la señal transmitida*
  - *Cada neurona aplica una función de activación a su entrada de red*
    - *Obtención de salida*

Características

- Entre sus características, podemos encontrar:
  - Auto-organización y Adaptabilidad
    - Procesado robusto y adaptativo
  - Procesamiento no lineal
  - Procesamiento Paralelo

Aplicaciones de RNA

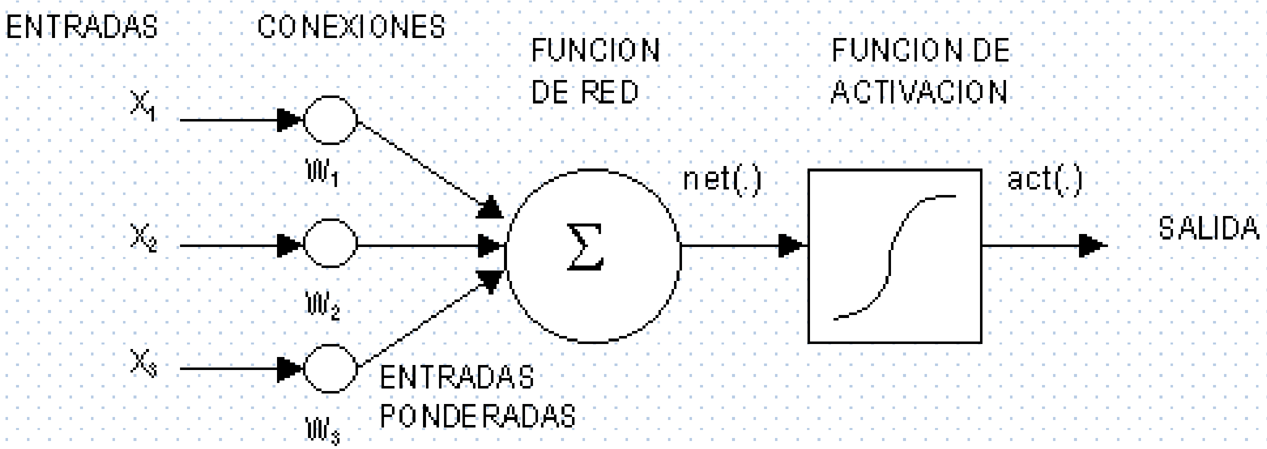
- Reconocimiento del Habla
- Clasificación de Imágenes
- Videojuegos
- Robótica
- Predicción
- Entre otros …

Un poco de historia

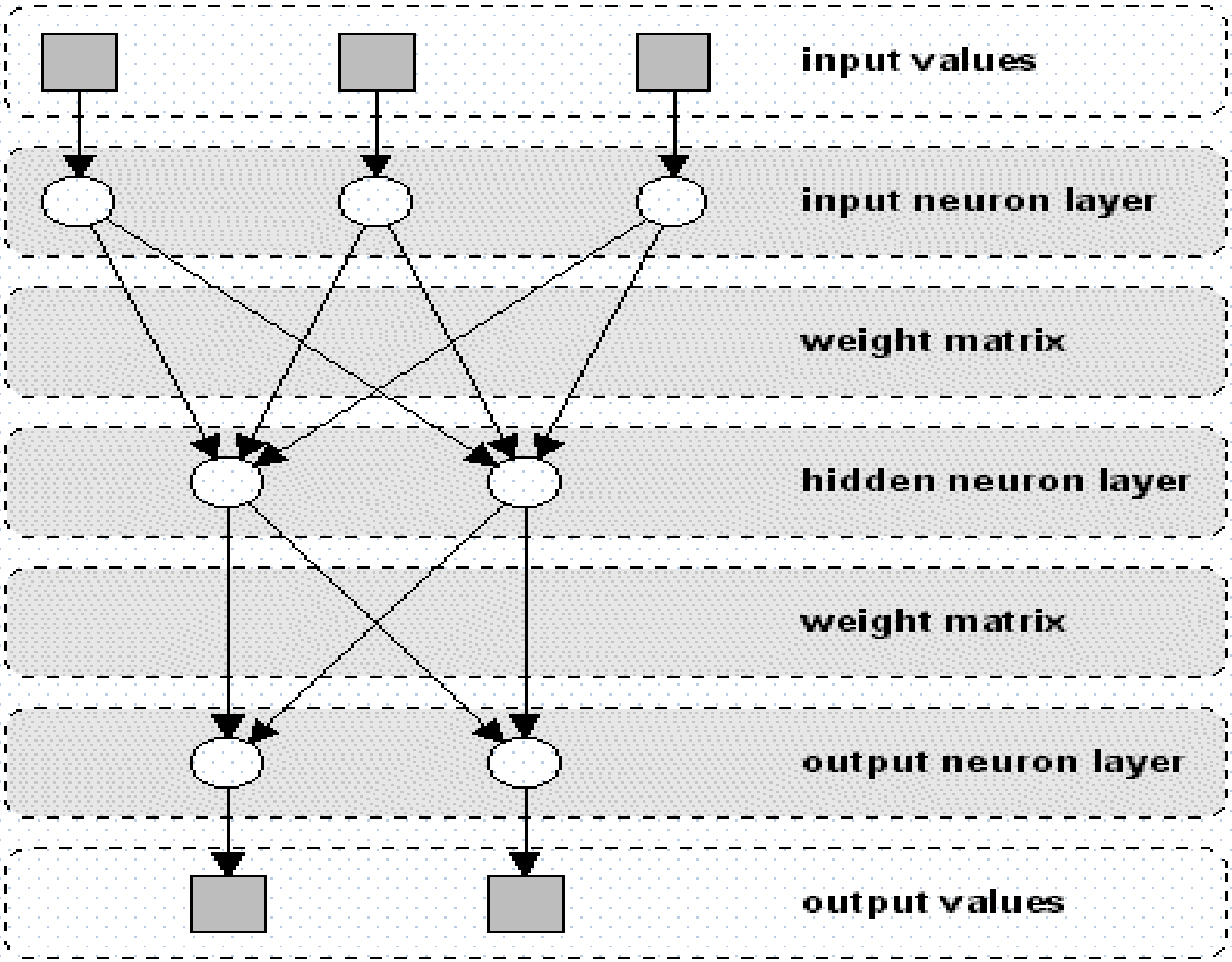
- McCullouch y Pitts (1943) propusieron un modelo computacional basado en redes neuronales biológicas
- Hebb (40's) propuso la hipótesis del aprendizaje basado en un mecanismo de plasticidad neuronal
  - Capacidad de remodelar el cerebro en función de experiencias
- Rosenblatt (1958) propuso el modelo perceptron
  - Clasificador lineal y binario
- Minsky y Papert (1969) descubrieron **problemas** en el perceptron (XOR)
- Webos (1975) propone algoritmo BackPropagation y las RNA **vuelven**

Elementos Básicos

- X1, X2 Y X3 son nodos o elementos de procesamiento
- W1, W2 Y W3 son pesos asociados a cada nodo
  - Fuerza de conexión
- Función de Red: Combinación Lineal de nodos y pesos
- Función de Activación: Define la salida real
  - Función Signo, Sigmoidal, entre otras



Estructura de Red



Aprendizaje

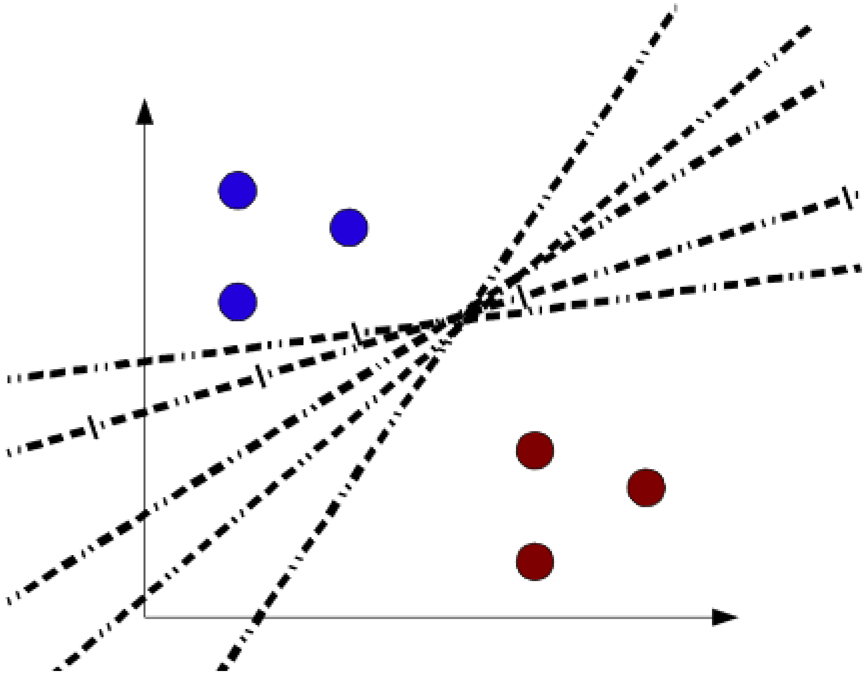
- Supervisado
  - Se presentan patrones de entrada y salida
  - *Multilayer Perceptron*
- No Supervisado
  - Solo patrones de entrada
  - Aprendizaje extrae patrones
  - Mapas auto-organizados (SOM)

Perceptron

- Clasificador Lineal y binario

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

- w: pesos
- x: vector de entrada
- b: sesgo (bias)



Perceptron - Algoritmo

- Inicializar los pesos (w) de manera aleatoria
- Para cada par «j» del conjunto de entrenamiento
  - Calcular la salida

$$y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j] = f[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}]$$

  - Adaptar los pesos

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$

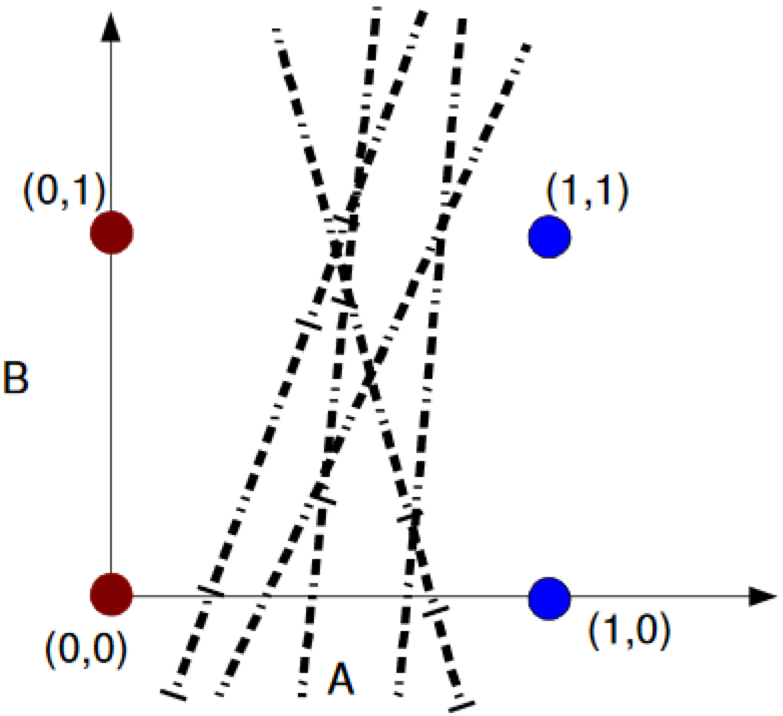
- Continuar hasta que el error sea menor que un umbral

$$d_j - y_j(t) < \gamma$$

Perceptron

- Función de Activación: f(x) > 0.5 1:0

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0



Perceptron

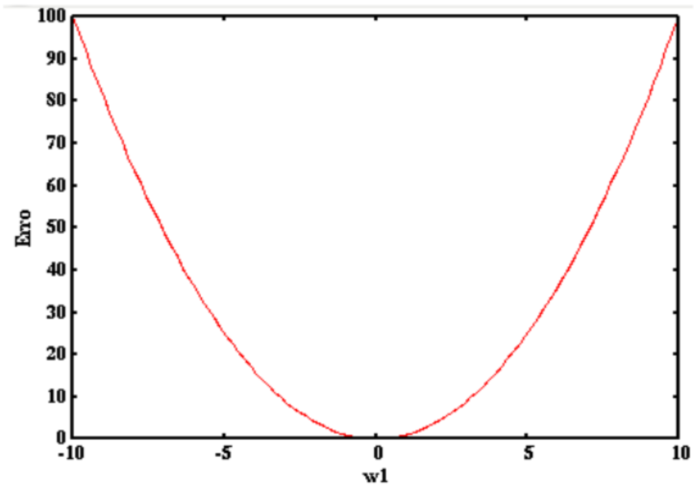
- Análisis de la Adaptación de Pesos

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$

- Consideremos el error vs el peso 1
  - Si alfa muy grande oscila
  - Debemos disminuir en pasos

  Cortos para minimizar el error

$f(x) = x^2$

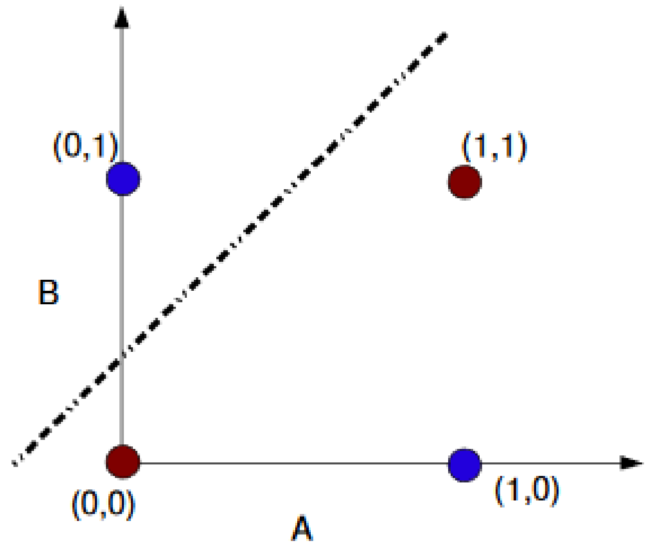


Problemas Perceptron

- ¿Qué ocurre con el XOR?

XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0



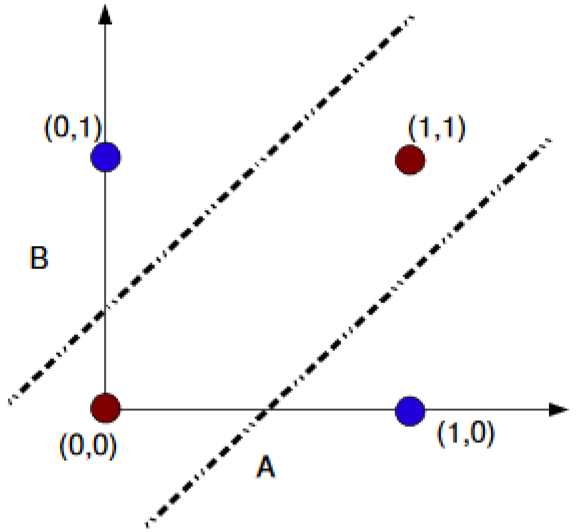
- Minsky y Papert (1969) presentan esta limitación
- Ecuación lineal representa un hiperplano

Problemas Perceptron

- ¿2 hiperplanos?
  - Regiones disjuntas pueden pertenecer a una misma clase

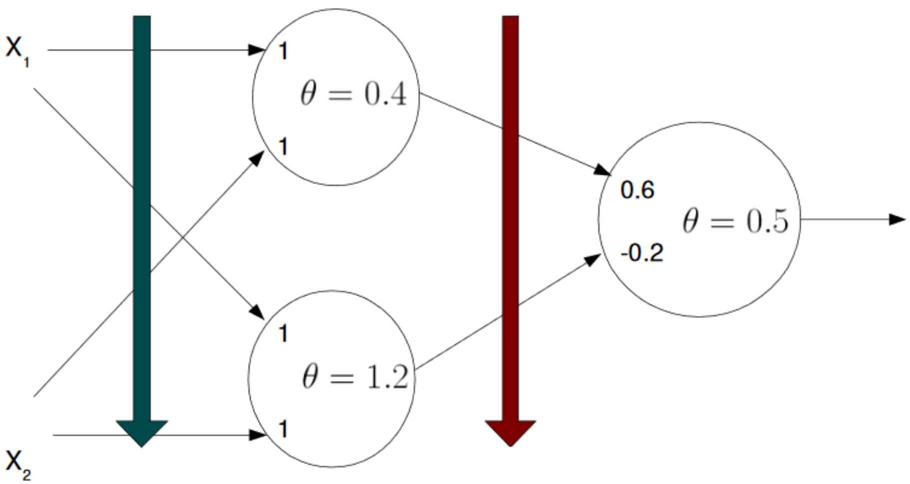
XOR Truth Table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0



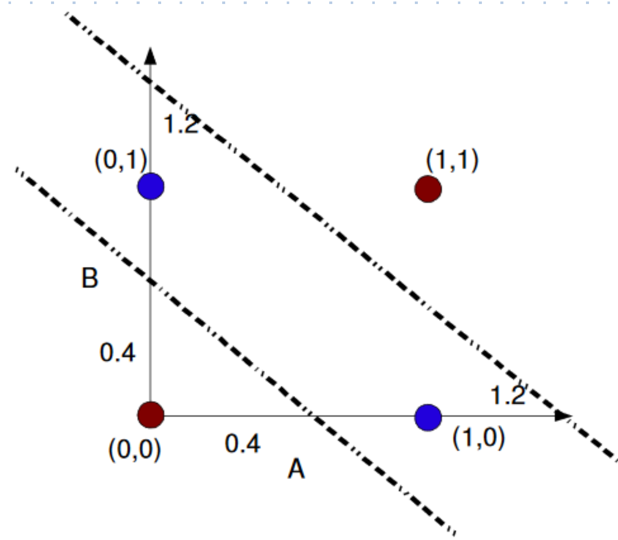
Perceptron Multicapa
Multilayer Perceptron

- Capa adicional llamada capa oculta (*Hidden Layer*)
  - *Solución al problema del XOR*



Pesos de capa escondida

Pesos de capa de salida

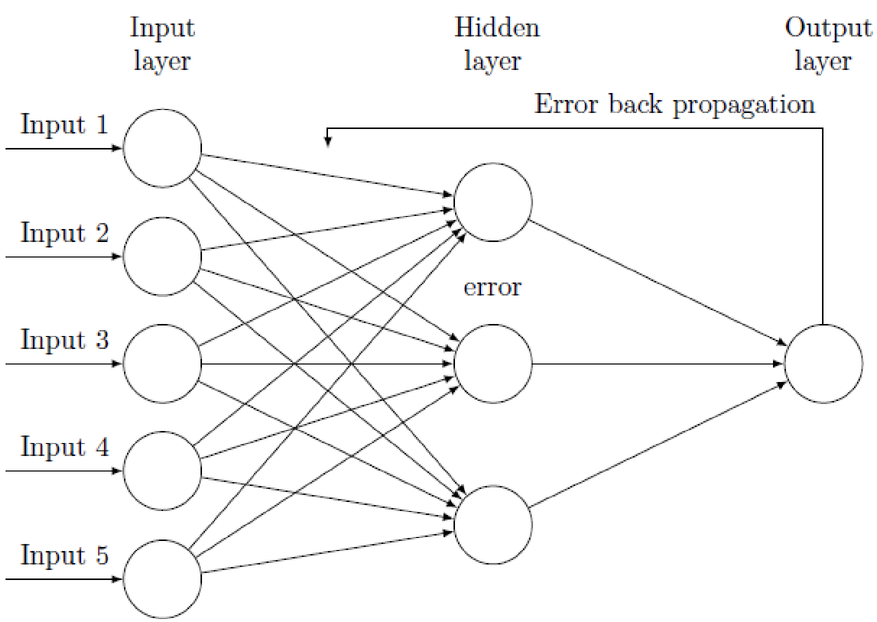


Perceptron Multicapa
Multilayer Perceptron

- Red ***Feedforward***
  - Entradas generan salidas
  - No hay retroalimentación como en otras redes
  - Algoritmo ***Backpropagation*** para entrenamiento
    - Error se propaga desde la última capa a la primera

Multilayer Perceptron
Regla Delta Generalizada

- **Entrenamiento:** En función al error encontrado en la capa de salida
  - Adaptación de pesos



Multilayer Perceptron
Regla Delta Generalizada

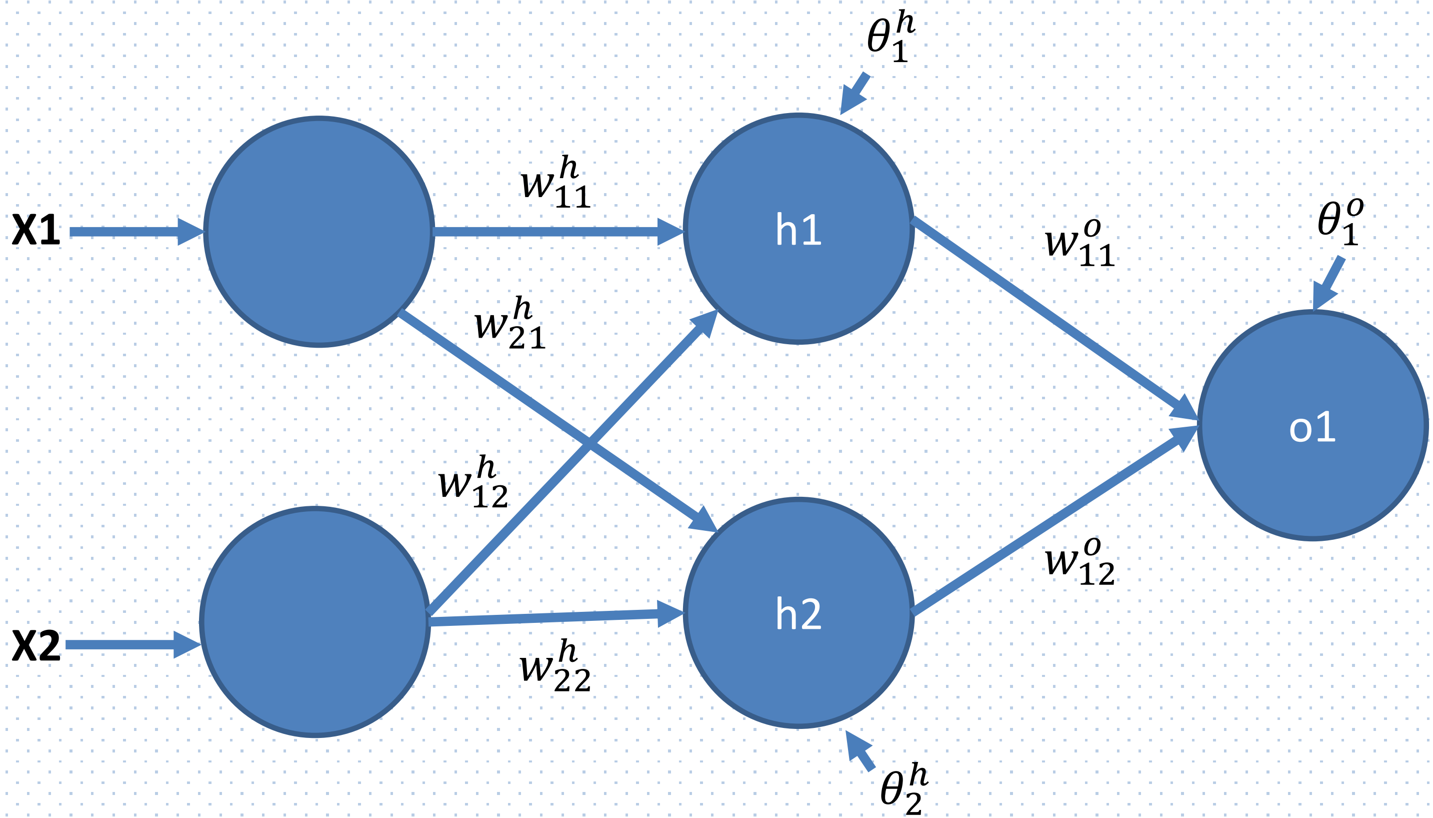
- Capa de Entrada es simple
  - *Neuronas solo dan valores para la capa escondida*
- Capa Escondida calcula:

$$\mathbf{net}_{pj}^{h} = \sum_{i=1}^{N} w_{ji}^{h} x_{pi} + \theta_j^h$$

- Capa de Salida calcula

$$\mathbf{net}_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o} i_{pj} + \theta_k^o$$

Multilayer Perceptron
Regla Delta Generalizada



$$\mathbf{net}_{pj}^h = \sum_{i=1}^{N} w_{ji}^h x_{pi} + \theta_j^h$$

Multilayer Perceptron
Regla Delta Generalizada

- **Actualización de los pesos de la capa de salida**
  - En la capa de salida pueden haber varias neuronas

$$\boldsymbol{\delta_{pk} = (y_{pk} - o_{pk})}$$

  - $\boldsymbol{y_{pk}}$: salida esperada de la neurona «k» para el vector de entrada «p»
  - $\boldsymbol{o_{pk}}$: salida producida de la neurona «k» para el vector de entrada «p»
  - ***p:*** indica el vector de entrada usado en el entrenamiento
  - ***k:*** Indica la neurona de la capa de salida

Multilayer Perceptron
Regla Delta Generalizada

- **Actualización de los pesos de la capa de salida**
  - *Minimizar la suma de errores cuadráticos para todas las unidades de salida considerando la entrada «p»*

$$\mathbf{E}_p = \frac{1}{2}\sum_{k=1}^{M}\delta_{pk}^2$$

- Se quiere reducir el error y sabemos que el error depende de los pesos «w»

Multilayer Perceptron
Regla Delta Generalizada

$$\mathbf{E}_p = \frac{1}{2} \sum_{k=1}^{M} \delta_{pk}^2$$

$$\delta_{pk} = (y_{pk} - o_{pk})$$

$$\mathbf{E}_p = \frac{1}{2} \sum_{k=1}^{M} (y_{pk} - o_{pk})^2$$

Multilayer Perceptron
Regla Delta Generalizada

- Derivando el error en función de los pesos (w):

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x) \text{ ou } \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

- Se tiene:

$$E_{pk} = \frac{1}{2}(y_{pk} - o_{pk})^2$$

$$f(g(x)) = \frac{1}{2}(y_{pk} - o_{pk})^2$$

$$g(x) = y_{pk} - o_{pk}$$

y

$$f'(g(x)) = 2 \cdot \frac{1}{2}(y_{pk} - o_{pk})$$

$$g'(x) = 0 - o'_{pk}$$

Multilayer Perceptron
Regla Delta Generalizada

$$f'(g(x)) = 2 \cdot \frac{1}{2}(y_{pk} - o_{pk})$$
$$g'(x) = 0 - o'_{pk}$$

- *Recordando que :*

$$o_{pk} = f_k^o(\mathbf{net}_{pk}^o)$$ ***entonces*** $$o'_{pk} = \frac{\partial f_k^o}{\partial \mathbf{net}_{pk}^o} \cdot \frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o}$$

$$f'(g(x))g'(x) = \left(2 \cdot \frac{1}{2}(y_{pk} - o_{pk})\right) \cdot \left(0 - \frac{\partial f_k^o}{\partial \mathbf{net}_{pk}^o} \frac{\partial \mathbf{net}_{pk}^o}{\partial w_{kj}^o}\right)$$

resultado

$$E_{pk} = \frac{1}{2}(y_{pk} - o_{pk})^2$$ ***y*** $$\frac{\partial E_{pk}}{\partial w_{pj}^o} = -(y_{pk} - o_{pk}) \frac{\delta f_k^o}{\partial \mathbf{net}_{pk}^o} i_{pj}$$

Multilayer Perceptron
Regla Delta Generalizada

- Falta derivar $\dfrac{\delta f_k^o}{\partial \mathbf{net}_{pk}^o}$

- *Si:* $f_k^o(\mathbf{net}_k^o) = (1 + e^{-\mathbf{net}_{jk}^o})^{-1}$ *entonces* $f_k'^o(\mathbf{net}_k^o) = f_k^o(1 - f_k^o)$

$$\downarrow$$

$$f_k'^o(\mathbf{net}_k^o) = o_{pk}(1 - o_{pk})$$

- *Entonces:*

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})o_{pk}(1 - o_{pk})i_{pj}$$

Multilayer Perceptron
Regla Delta Generalizada

- Finalmente, se tienen las fórmulas para el delta y la actualización de pesos de la capa de salida

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

Multilayer Perceptron
Regla Delta Generalizada

- **Actualización de los pesos de la capa oculta**
  - **El error de la capa de salida afecta a la capa oculta**

- El error medio en la capa de salida es dado por:

$$
\begin{aligned}
E_p &= \frac{1}{2}\sum_k (y_{pk} - o_{pk})^2 \\
&= \frac{1}{2}\sum_k (y_{pk} - f_k^o(\mathbf{net}_{pk}^o))^2 \\
&= \frac{1}{2}\sum_k \left(y_{pk} - f_k^o \left(\sum_j w_{kj}^o \boxed{i_{pj}} + \theta_k^o \right)\right)^2
\end{aligned}
$$

Multilayer Perceptron
Regla Delta Generalizada

- Así, definimos la variación del error en función de la cámara oculta:

$$\frac{\partial E_p}{\partial w_{ji}^h} = \frac{1}{2}\sum_k \frac{\partial}{\partial w_{ji}^h}(y_{pk} - o_{pk})^2$$
$$= -\sum_k (y_{pk} - o_{pk})\frac{\partial o_{pk}}{\partial \mathbf{net}_{pk}^o}\frac{\partial \mathbf{net}_{pk}^o}{\partial i_{pj}}\frac{\partial i_{pj}}{\partial \mathbf{net}_{pj}^h}\frac{\partial \mathbf{net}_{pj}^h}{\partial w_{ji}^h}$$

- A partir de la ecuación obtenemos:

$$\frac{\partial E_p}{\partial w_{ji}^h} = -\sum_k (y_{pk} - o_{pk}) f_k^{o\prime}(\mathbf{net}_{pk}^o) w_{kj}^o f_j^{h\prime}(\mathbf{net}_{pj}^h) x_{pi}$$

Multilayer Perceptron
Regla Delta Generalizada

- Siendo:

$$\mathbf{net}^o_{pk} = \sum_{j=1} w^o_{kj} i_{pj} + \theta^o_k$$

$$\mathbf{net}^h_{pi} = \sum_{j=1}^{L} w^h_{kj} x_{pi} + \theta^h_k$$

$$\frac{\partial E_p}{\partial w^h_{ji}} = \frac{1}{2} \sum_k \frac{\partial}{\partial w^h_{ji}} (y_{pk} - o_{pk})^2$$

$$= -\sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \mathbf{net}^o_{pk}} \boxed{\frac{\partial \mathbf{net}^o_{pk}}{\partial i_{pj}}} \frac{\partial i_{pj}}{\partial \mathbf{net}^h_{pj}} \boxed{\frac{\partial \mathbf{net}^h_{pj}}{\partial w^h_{ji}}}$$

$$\frac{\partial E_p}{\partial w^h_{ji}} = -\sum_k (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk}) \boxed{w^o_{kj}} f^{h\prime}_j(\mathbf{net}^h_{pj}) \boxed{x_{pi}}$$

Multilayer Perceptron
Regla Delta Generalizada

- Se puede calcular los pesos de las neuronas de la capa intermedia de la siguiente forma:

$$\triangle_p w_{ji}^h = \eta f_j^{h\prime}(\mathbf{net}_{pj}^h) x_{pi} \sum_k (y_{pk} - o_{pk}) f_k^{o\prime}(\mathbf{net}_{pk}^o) w_{kj}^o$$

$$\triangle_p w_{ji}^h = \eta f_j^{h\prime}(\mathbf{net}_{pj}^h) x_{pi} \sum_k \delta_{pk}^o w_{kj}^o$$

- Así, la adaptación de los pesos de la camada escondida dependen del error en la capa de salida

Multilayer Perceptron
Regla Delta Generalizada

- Finalmente, si la función de activación es sigmoidal:

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

- Capa de salida:

$$\delta^o_{pk} = (y_{pk} - o_{pk}) f^{o\prime}_k(\mathbf{net}^o_{pk})$$

$$w^o_{kj}(t+1) = w^o_{kj}(t) + \eta \delta^o_{pk} i_{pj}$$

¡IMPORTANTE!

- Capa Intermedia

$$\delta^h_{pj} = f^{h\prime}_j(\mathbf{net}^h_{pj}) \sum_k \delta^o_{pk} w^o_{kj}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \eta \delta^h_{pj} x_i$$

Entrenamiento

- Iterando ☺
- Se requieren muchas iteraciones con muchos ejemplos (también pueden ser “épocas”)
- Una época representa el período de entrenamiento con todos los ejemplos de un *dataset*
- Puede ser bastante lento
- Es posible procesamiento paralelo ☺

Entrenamiento y Test

- ¿Cuántos ejemplos son necesarios?
  - Cuanto más, mejor ☺

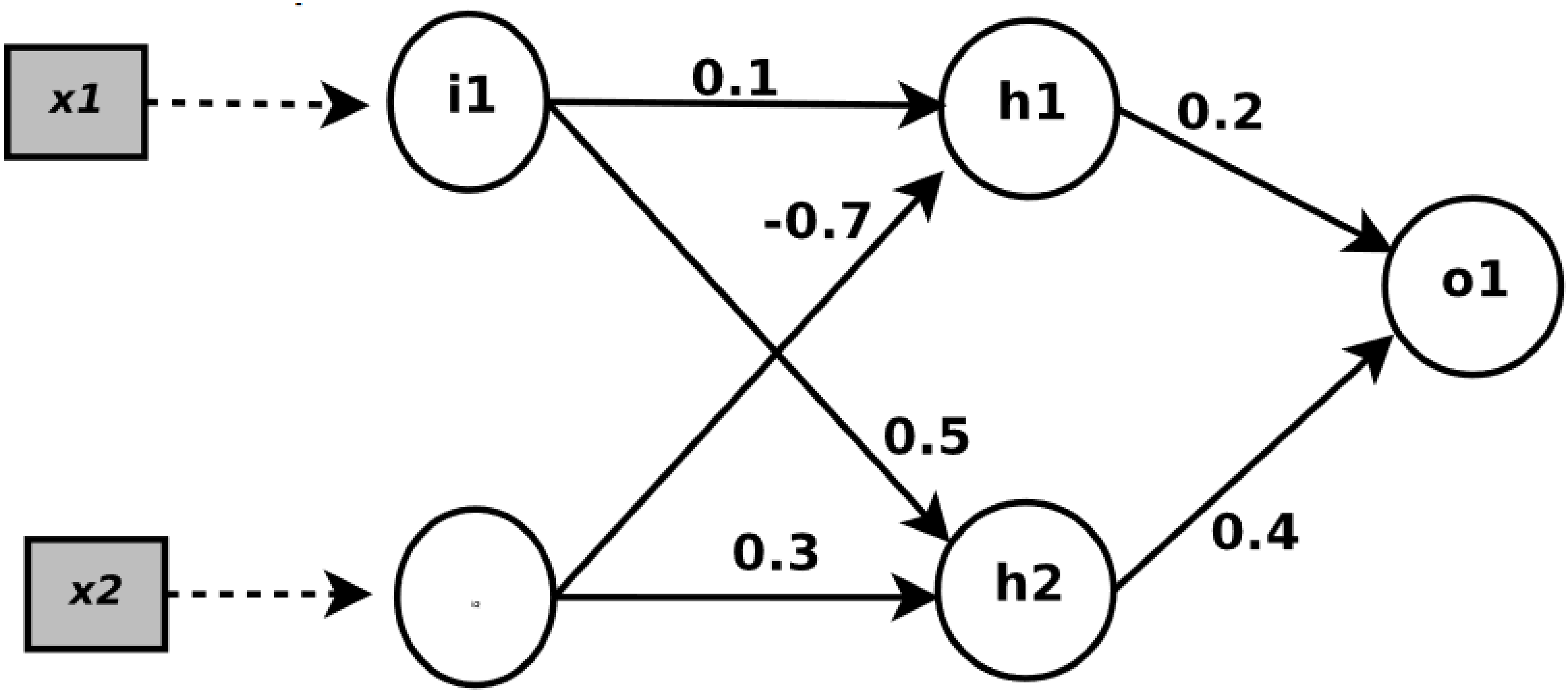
- *Dataset* de entrenamiento y test diferentes
  - **Capacidad de Generalización**
  - Minimizar el error en ejemplos de prueba

Ejercicio 1

- Entrenamiento de un perceptrón multicapa para realizar la operación XOR
- Descripción de la red
  - 1 capa oculta
  - 2 neuronas en capa de entrada
  - 2 neuronas en capa oculta
  - 1 neurona en capa de salida
- Tasa de Aprendizaje
  - alfa = 0.25

Ejercicio 1

- Esquema de Red Neuronal con pesos aleatorios iniciales



Ejercicio 2

- Ahora programe su propio perceptron multicapa y pruebe la ejecución del mismo conjunto de datos ☺

Conjunto de entrenamiento

	Entradas		Salida
	x_1	x_2	t_1
e_1	0	1	1
e_2	1	0	1
e_3	1	1	0
e_4	0	0	0

boost
accelerate your learning

Fin ☺