Introducción a las Redes Neuronales

Tomás Arredondo Vidal
Depto. Electronica UTFSM
4/5/12

Introducción a las Redes Neuronales

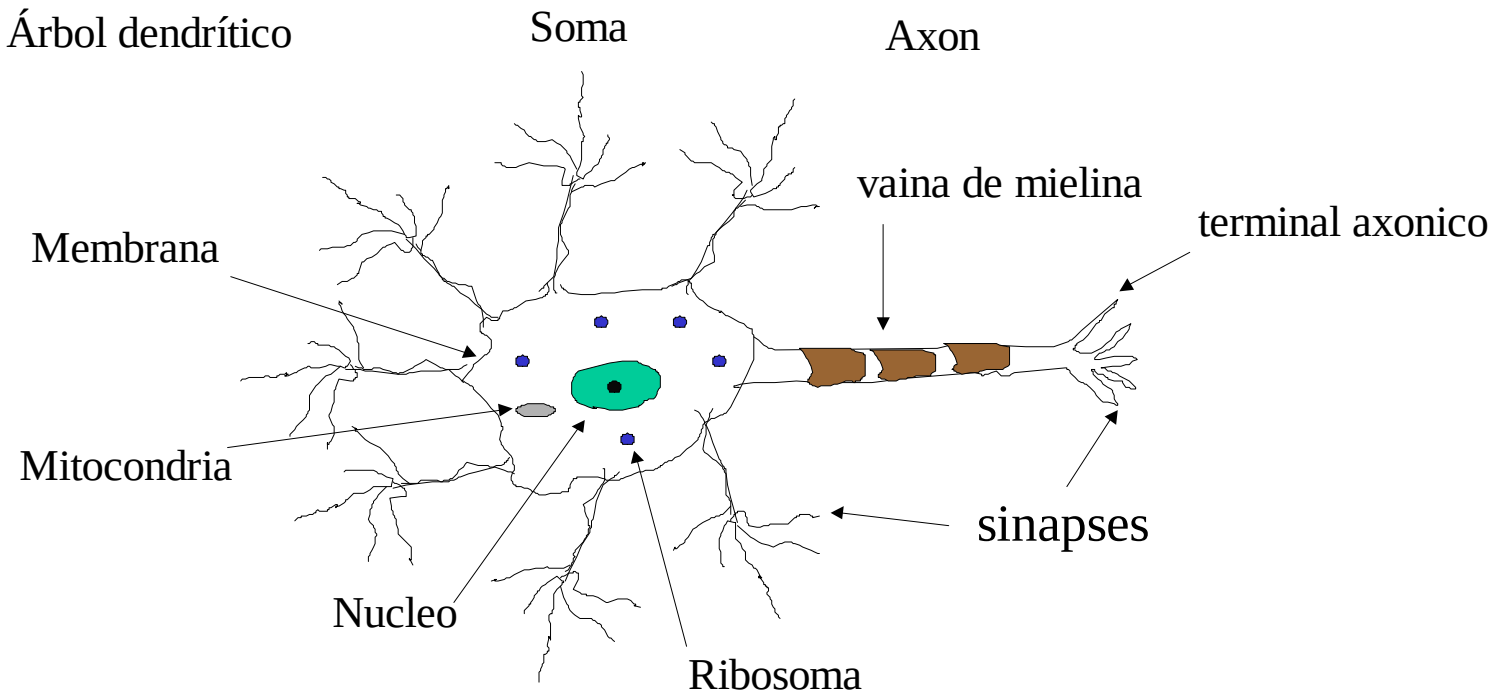
Contenidos

- Introducción a las neuronas
- Introducción a las redes neuronales artificiales (ANN)
- Introducción a algunos algoritmos de aprendizaje de las ANNs

Introducción a las Neuronas

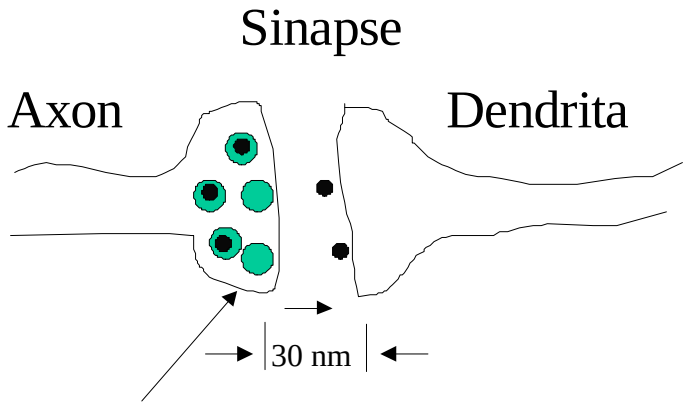
- En nuestro sistema nervioso existen células llamadas neuronas que son una unidad de procesamiento que recibe un estimulo eléctrico de otras neuronas principalmente a través de su árbol dendrítico
- El estimulo eléctrico recibido al pasar de un cierto umbral causa que la neurona a su vez envie una señal eléctrica a través de su axon a otras sucesivas neuronas
- Basado en la fuerza de interconexión entre las neuronas la red puede ser capaz de cómputos y detección de patrones complejos

Neuronas Reales



Sinapsis

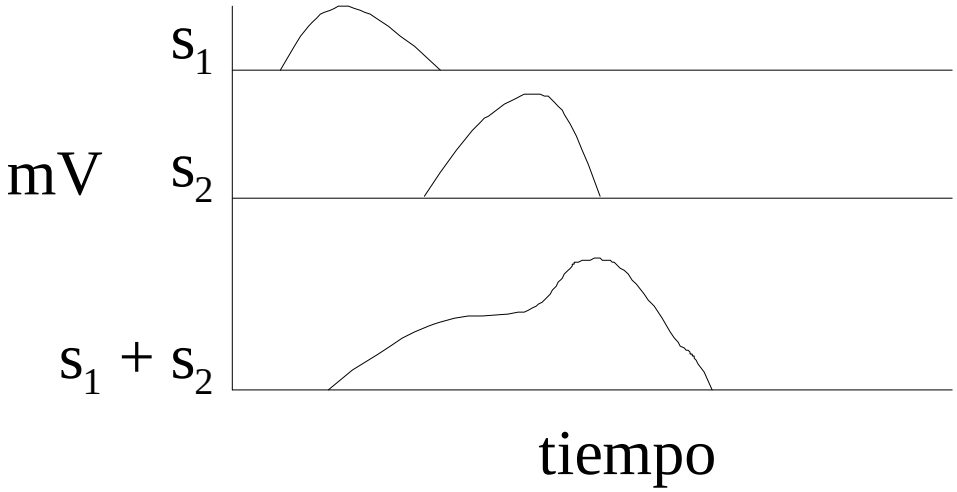
- Cuando el árbol dendrítico recibe impulsos eléctricos sobre un umbral voltaje especifico la neurona manda una señal eléctrica a través del axon a los terminales axonicos
- En esos terminales se conectan los terminales axonicos (a través de sinapsis) a otras neuronas
- Los terminales axonicos se pueden conectar a través de sinapsis a diferentes partes de la neurona pero típicamente se considera solo la conexión a otras dendritas



- Dependiendo del neurotransmisor (ion) el potencial inducido en terminal postsinaptico (dendrita) puede ser positivo (excitado) o negativo (inhibidor)

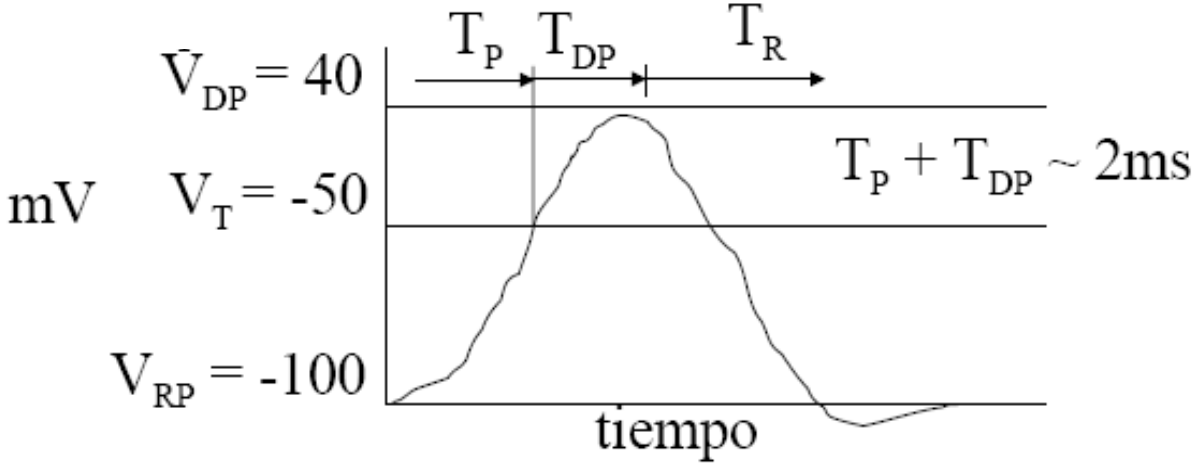
Neuronas Temporales

- Las neuronas tienen un tiempo de activación del orden de milisegundos comparado con nanosegundos para hardware computacional
- El cerebro humano tiene alrededor de 10^{11} neuronas con un promedio de 10^4 conexiones cada una
- Todas las señales de input de otras neuronas se suman de manera espaciotemporal por la neurona



Polarización de las Neuronas

- Cuando la neurona es estimulada pasado un voltaje umbral (V_T) se despolariza (T_{DP}) la membrana y las bombas de sodio-potasio paran de actuar, la membrana cambia de estado y se convierte mas permeable al ion positivo de NA+ que al entrar a la neurona cambia el potencial a través de la membrana neuronal a positivo (+ 40mV)
- Después hay un periodo refractario (T_R) de ~ 1ms en el cual se cierran los canales de NA+, la membrana vuelve a ser permeable a K+, las bombas se activan y la neurona logra su voltaje inicial a través del escape de K+ en este periodo la neurona no es susceptible a estímulos sinápticos

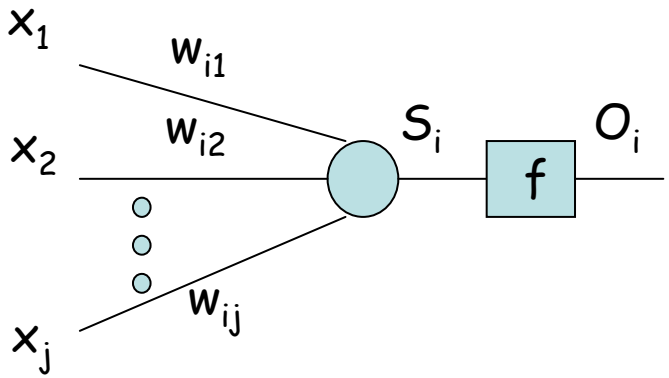


Modelo de una Neurona

- McCulloch and Pitts (1943) desarrollaron el modelo original de una neurona esta incluyo diferentes inputs (x), pesos para esos inputs (w), una función nolinear (f) de transferencia y el Output (O)
- En el modelo original no se incluyo un profesor (Teacher) para entrenar a la red
- La función no lineal de transferencia usada era el escalón (hard limit o step function)
- Se incluyo un input de bias para evitar outputs no deseados cuando los inputs son zero

Modelo MP de una Neurona

- Se modela la neurona como una red con pesos (conexiones sinápticas) desde los nodos j al nodo i, w_{ij}

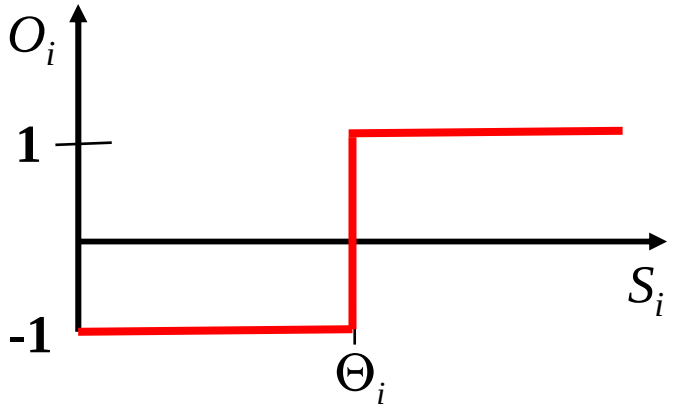


- Input neto a la neurona es:

$$S_i = \sum_j w_{ij} x_j$$

- Función de output f es sgn():

$$O_i = \begin{matrix} -1 \text{if } S_i < \Theta_i \\ 1 \text{if } S_i \geq \Theta_i \end{matrix}$$



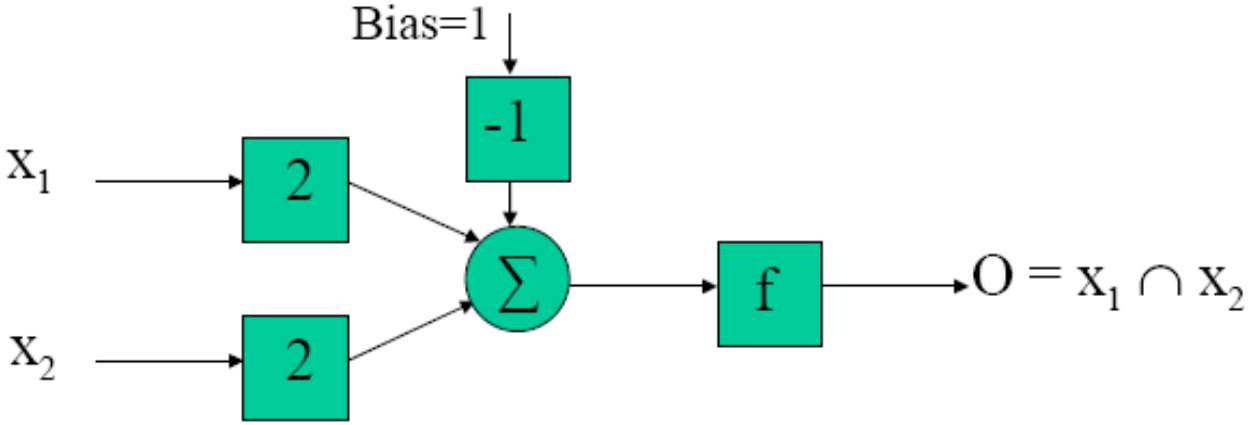
(Θ_i es umbral para neurona i)

Computación Neuronal

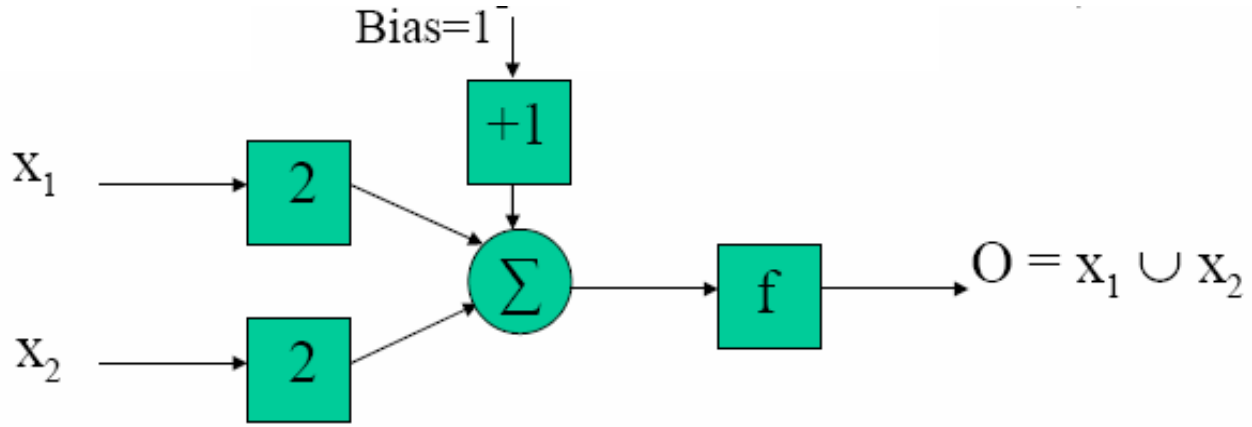
- McCullough y Pitts (1943) mostraron que este modelo podría calcular funciones lógicas para poder construir maquinas de estados finitos
- Pueden construir compuertas lógicas (AND, OR, NOT)
- Con estas compuertas pueden calcular cualquier función lógica usando una red de dos niveles AND-OR
- También se denomina perceptron de una capa

Computación Neuronal

X_1	X_2	Y
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1



X_1	X_2	Y
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	+1



Introducción a las Redes Neuronales

Contenidos

- Introducción a las neuronas
- Introducción a las redes neuronales artificiales (ANN)
- Introducción a algunos algoritmos de aprendizaje de las ANNs

Redes Neuronales Artificiales (ANNs)

- Las ANNs son un paradigma para hacer computo y para la detección de patrones basado en la interconexión paralela de neuronas artificiales
- Las ANNs son un modelo basado en los complejos sistemas nerviosos de los animales y seres humanos con su gran cantidad de interconexiones y paralelismo
- Los comportamientos inteligentes de estos son una propiedad emergente del gran numero de unidades no un resultado de reglas simbólicas o algoritmos

Redes Neuronales Artificiales (ANNs)

- Las características previamente expuestas de las neuronas se duplican para crear redes de neuronas artificiales que tienen capacidades similares de procesar información en forma paralela.
- Algunas capacidades de las redes neuronales son:
  1. Aprendizaje
  2. Clasificación
  3. Almacenaje de información
  4. Interpolarización
  5. Adaptación

Redes Neuronales Artificiales (ANNs)

- Las redes neuronales artificiales recrean o modelan las características de las neuronas ya mencionadas en sistemas computacionales de software y hardware
- Usan grupos de muchas neuronas artificiales (como la neurona McCulloch-Pitts o MP)
- Usan principios de computación colectiva y descentralizada (paralela)
- Son capaces de aprender

Redes Neuronales Artificiales (ANNs)

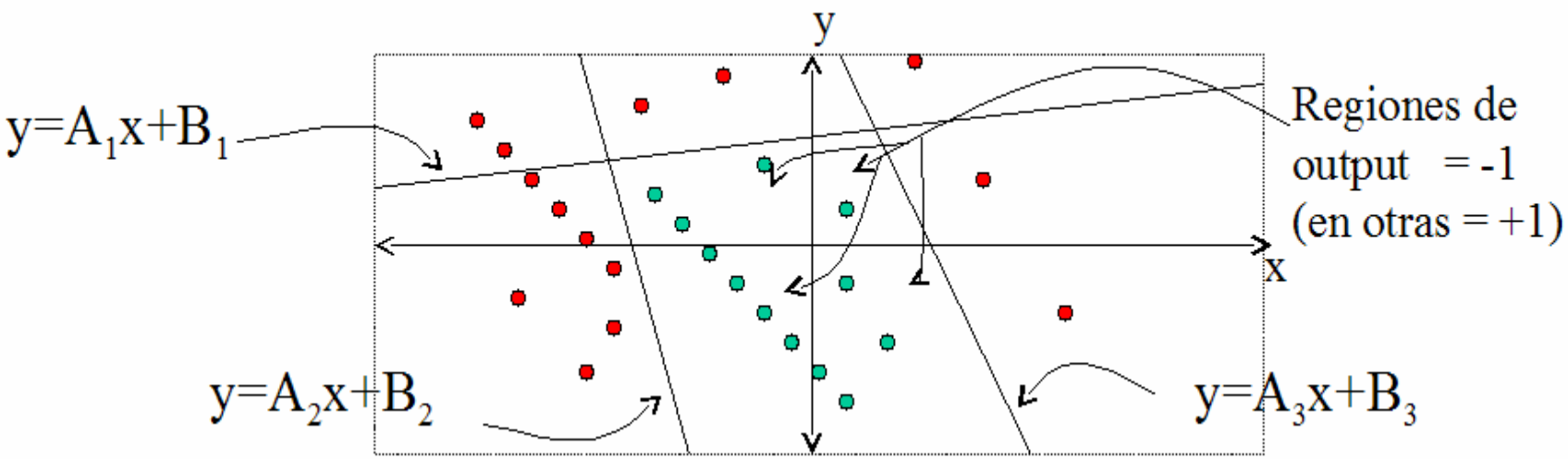
- Operaciones robustas y resistentes a fallas de neuronas individuales
- La red hace asociaciones automáticamente
- La red crea el “programa” al ajustar los pesos durante su aprendizaje
- Al operar requieren de sincronizidad o que las señales entren al mismo tiempo a todas las neuronas en el mismo nivel (no así las biológicas que usan la suma espacio-temporal)

Redes Neuronales Multicapa

- El modelo de una red neuronal con múltiples capas difiere de las que hemos visto hasta ahora en que incluye neuronas en capas escondidas
- Una capa escondida significa que no es visible ni al input ni al output
- Un perceptron de dos capas puede discernir regiones poligonales, uno de tres o mas capas puede discernir regiones arbitrarias (Multi Layer Perceptron o MLP)

Redes Neuronales Multicapa

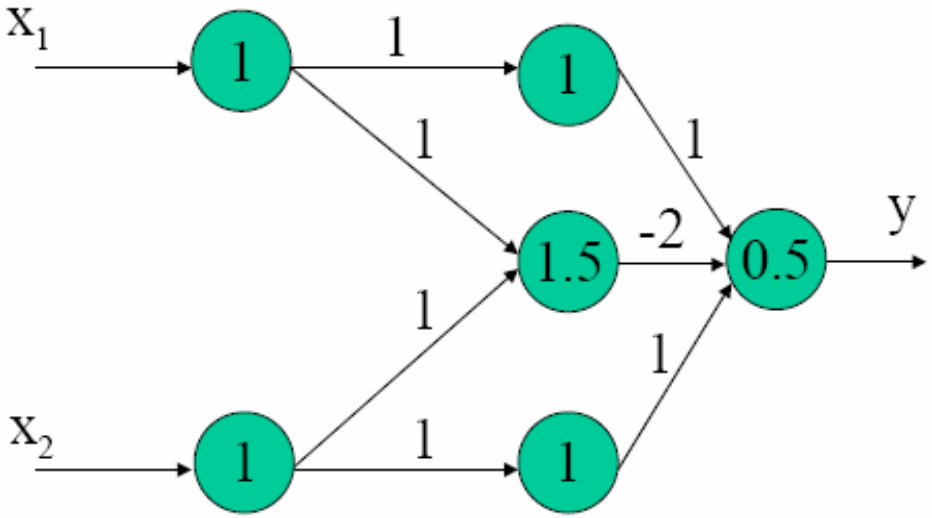
- Estos ANN tienen la capacidad de separar inputs en múltiples funciones lineales y de esta manera pueden detectar patrones mas complejos que redes de una función lineal como las vistas anteriormente



Redes Neuronales Multicapa

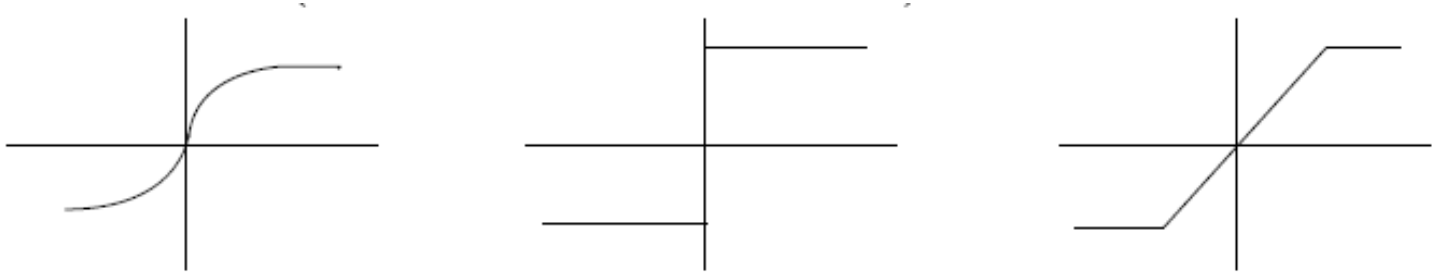
- Este ejemplo de un multilayer neural network es capaz de resolver el problema del XOR
- Los valores sobre las líneas indican pesos y los en los círculos indican umbrales (thresholds)
- La función no lineal es un step function con valores de 1 si el umbral el excedido y de 0 si el umbral no es excedido
- Ejemplo XOR (hay otras soluciones posibles):

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0



Funciones No Lineales

- Las típicas funciones no lineales (f) usadas en redes neuronales incluyen: sigmoides, limitante duro (step function), y rampa
- Los sigmoides f(s) son:
  – monotonicos (sin discontinuidades)
  – bounded (limitados)
  – Tiene derivados simples f’(s)=kf(s)[1-f(s)]
  – no linear
- Los limitantes duros y rampas son:
  – no monotonicos (tiene discontinuidades)
  – no tienen derivados simples
  – linear (dentro de las áreas con limites)

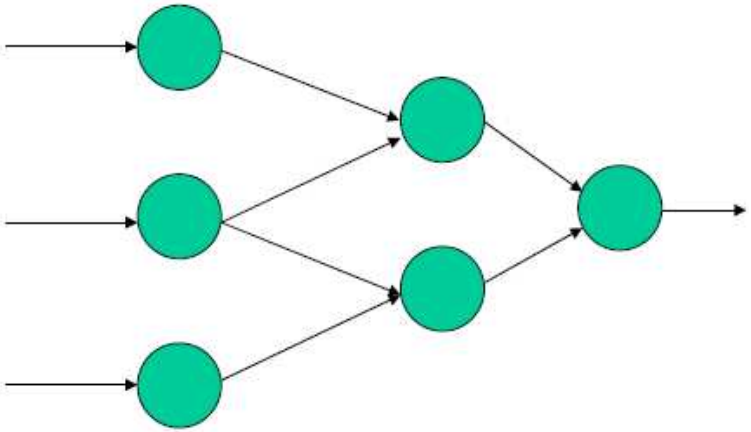


Consideraciones Parametricas

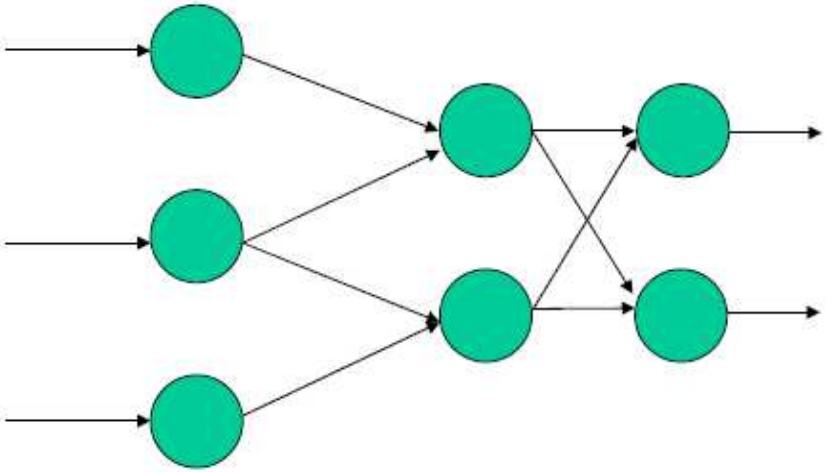
- Tamaño: cantidad de neuronas para resolver un problema
- Tipo o paradigma: numero de capas escondidas, numero de neuronas en cada capa
- Algoritmo de aprendizaje
- Numero de iteraciones durante el aprendizaje
- Numero de cálculos por cada iteración
- Velocidad para reconocer un patrón
- Capacidad de reconocimiento de patrones diferentes
- Grado de adaptabilidad (después de entrenamiento)
- Si requiere de Bias
- Valores de umbral
- Limites (boundaries) de pesos sinápticos
- Selección de función no linear (f)
- Inmunidad necesaria de la red al ruido
- Valores finales de los pesos (programa final) de la red

Paradigmas Multicapa

- Multilayer feedforward con una capa escondida

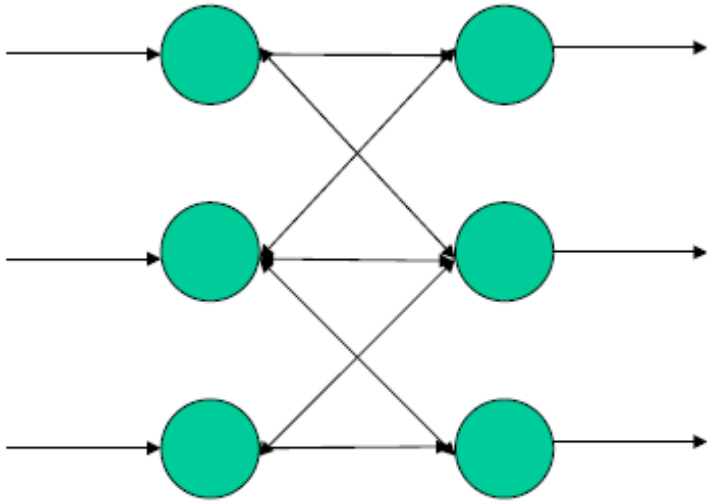


- Multilayer feedforward con capas escondidas y competitiva/cooperativa



Paradigmas Multicapa

- Bilayer feedforward/backward sin capas escondidas



- Multilayer feedforward/backward
- Monolayer with feedback
- Otras… (aquí entra la imaginacion)

Introducción a las Redes Neuronales

Contenidos

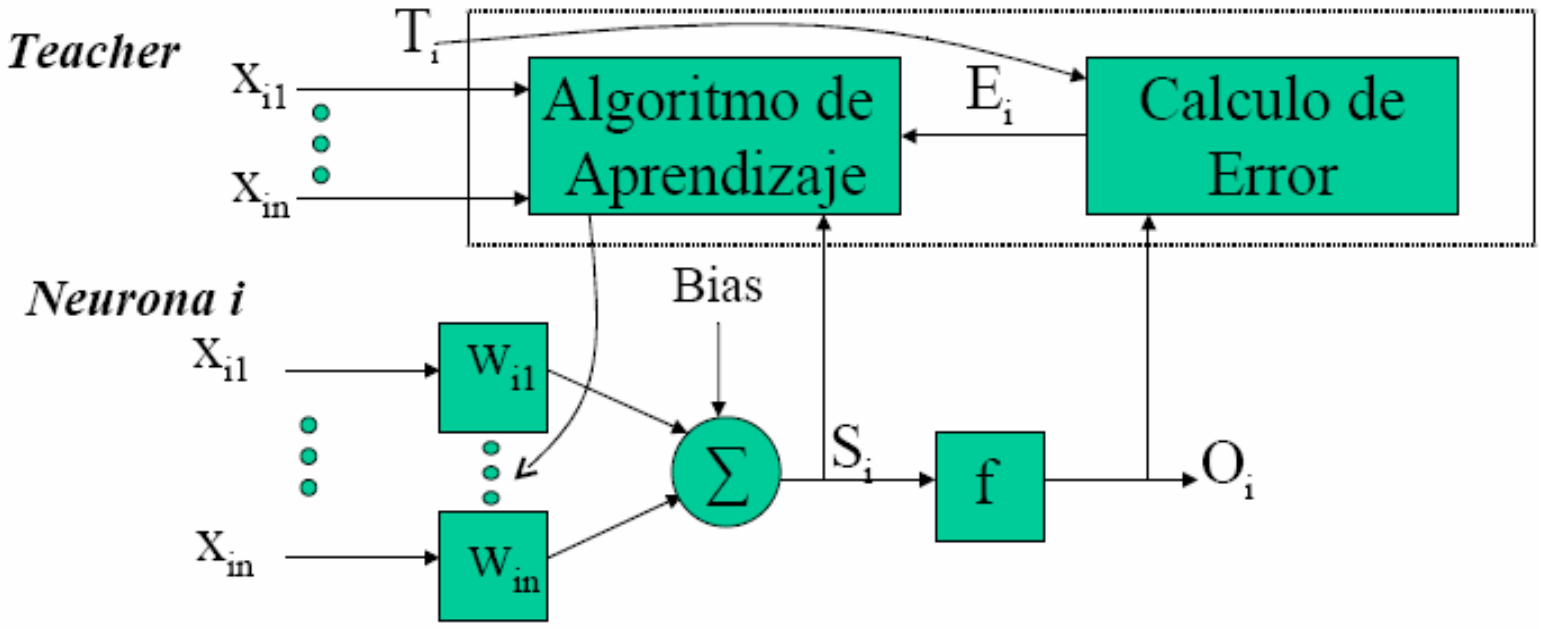
- Introducción a las neuronas
- Introducción a las redes neuronales artificiales (ANN)
- Introducción a algunos algoritmos de aprendizaje de las ANNs

Algoritmos de Aprendizaje

- El proceso de aprendizaje es el proceso por el cual la red neuronal se adapta al estimulo modificando sus pesos y eventualmente produce un resultado esperado
- Hay dos tipos de aprendizaje: supervisado y sin supervisión
- Aprendizaje supervisado incluye un supervisor (o teacher) que le indica a la red cuan cerca esta de aprender y va modificando los pesos neuronales
- Sin supervisión solamente se forman grupos de clasificación de acuerdo a indicaciones o propiedades, no se calcula un error (E)
- Se puede usar el error o el gradiente de error para que la red aprenda
- La dirección del gradiente es hacia incrementar el Error por ende se cambian los pesos hacia la dirección inversa

Aprendizaje Supervisado

- El modelo de aprendizaje con profesor (teacher) significa que durante el aprendizaje hay circuitos (o programas) externos a la neurona que comparan el output deseado (T_i) con el actual (O_i) y producen el error entre ellas ($E_i = T_i - O_i$)
- Usando el error (E_i) un algoritmo de aprendizaje modifica los valores de los pesos uno por uno tratando de minimizar el error, este proceso requiere de muchas iteraciones



Aprendizaje usando Perceptron Training

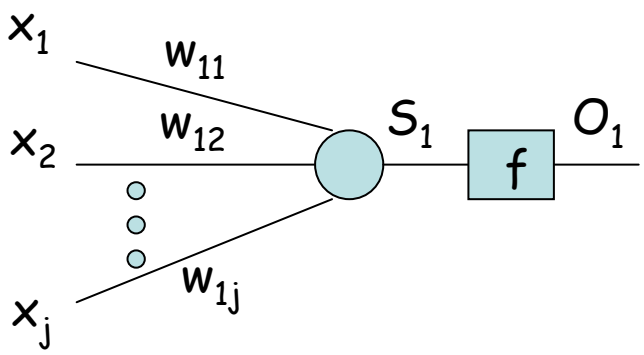
- Se asumen ejemplos de aprendizaje que dan el output deseado para una neurona dada un conjunto de inputs conocidos
- Se aprenden pesos para que cada neurona produzca el output correcto
- Algoritmo es iterativo
- Se usa en perceptrones de una capa
- Red aprende basado en el Error: E = T – O
- Dado que la red aprende usando E se puede usar una función de transferencia no diferenciable
- El parámetro η es una constante pequeña positiva conocida como taza de aprendizaje

Perceptron Learning Procedure

- Actualizar los pesos de la neurona i de acuerdo a:

$$w_{ij}(k+1) = w_{ij}(k) + \eta(T_i - O_i)x_j$$

  - η es la taza de aprendizaje
  - T_i es el valor del profesor de neurona i



- Equivalente a las reglas:
  - Si $T_i > O_i$ se quiere incrementar O_i entonces si $x_j > 0$ se incrementa w_i pero si $x_j < 0$ se reduce w_i
  - Si $T_i < O_i$ se quiere reducir O_i entonces si $x_j > 0$ se reduce w_i pero si $x_j < 0$ se incrementa w_i

Perceptron Learning

Algoritmo:

1- Asignar umbrales y pesos a valores iniciales aleatorios

2- $\mathbf{w}_{ij}(k+1) = \mathbf{w}_{ij}(k) + \eta(E(k))\mathbf{x}_j$

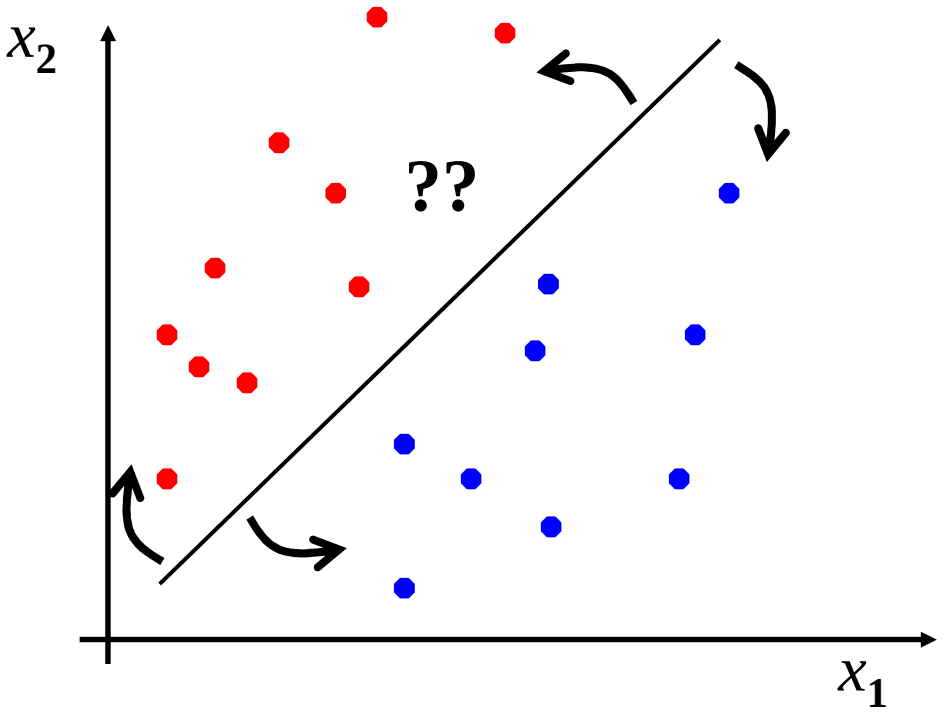
3- Se puede usar cualquier función no lineal (f) ya que el aprendizaje no depende del calculo de un derivado

4- Se continua iterando hasta que se llega a un output deseado, que se llego a un numero de iteraciones o que los pesos no están cambiado

Perceptron como un Separador Lineal

- El perceptron implementa una función lineal del umbral, de esta forma busca un separador lineal que discrimina entre las clases

$$w_{11}x_1 + w_{12}x_2 > \Theta_1$$



$$x_2 > -\frac{w_{11}}{w_{12}}x_1 + \frac{\Theta_1}{w_{12}}$$

$$O_1 = \begin{matrix} -1 \text{if } S_1 < \Theta_1 \\ 1 \text{if } S_1 \geq \Theta_1 \end{matrix}$$

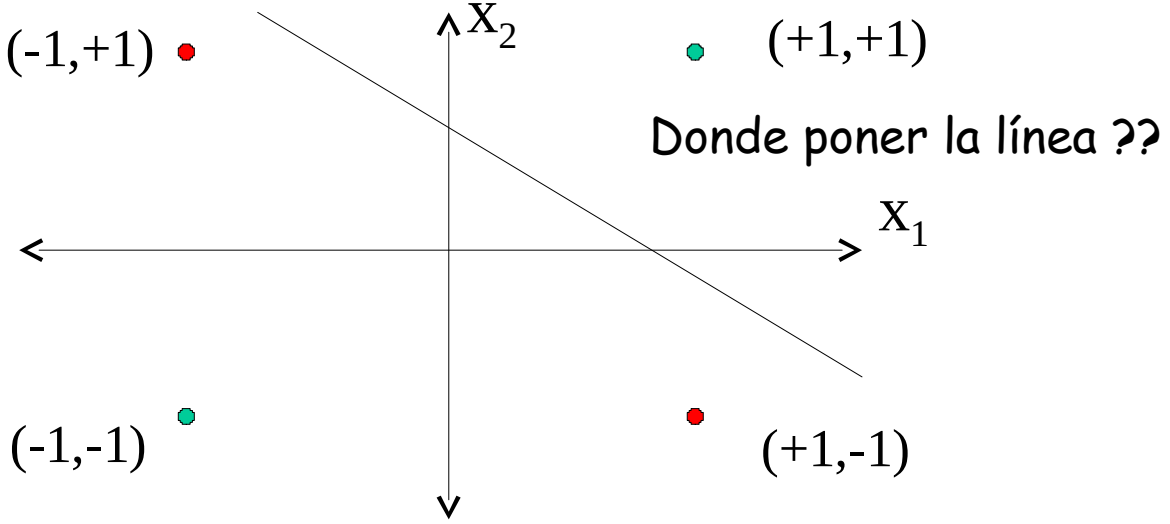
Un *hyperplano* en espacio de n dimensiones

Perceptron como un Separador Lineal II

- Un problema es que para aprender patrones diferentes hay que entrenar la red completamente de nuevo
- Otro problema de esta red es la incapacidad de aprender funciones no linealmente separables (Ej. XOR) (Minsky 1969)
- Esta incapacidad bajo el interés por el perceptron hasta que se utilizo en múltiples capas

XOR

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

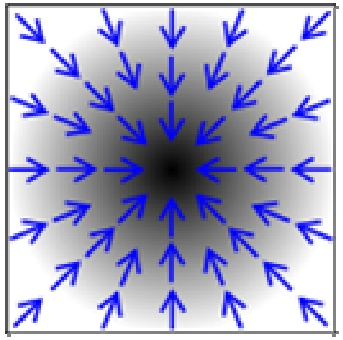


Perceptron: Convergencia y Ciclos

- **Perceptron convergence theorem**: Si los datos son linealmente separables y un set de pesos existen que sean consistentes con los datos, entonces el perceptron eventualmente va a converger en un conjunto de pesos consistentes con los datos.
- **Perceptron cycling theorem**: Si los datos no son linealmente separables el algoritmo eventualmente va a repetir un conjunto de pesos y umbral

Espacio de Búsqueda de Pesos como Minimización

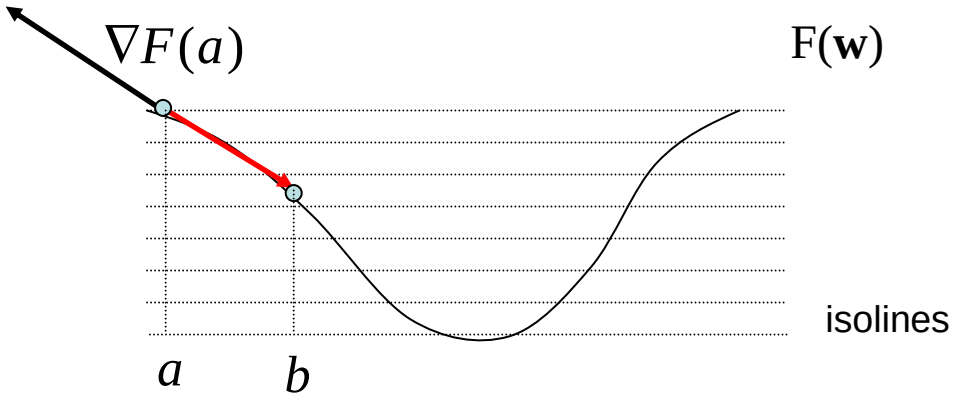
- El espacio de búsqueda es el set de pesos **w** y un umbral
- Objetivo es minimizar el error de clasificación del conjunto de entrenamiento
- Esta basado en la idea de que el gradiente de una función escalar F(**w**) es un vector que apunta en la dirección del mayor incremento de F con magnitud dada por la mayor tasa de cambio de F

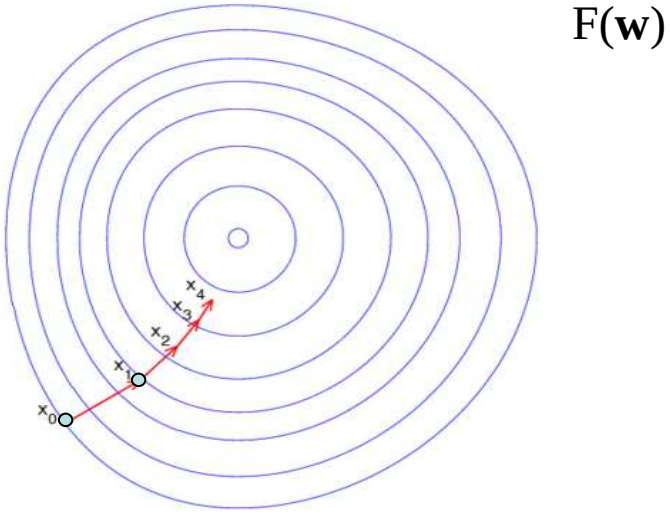


Espacio de Búsqueda de Pesos como Minimización

- Al ir en la dirección contraria al gradiente:
- Dependiendo de la superficie de búsqueda y de usar una tasa de aprendizaje pequeña
- Se llega de a un punto b para el cual F(a) ≥ F(b):

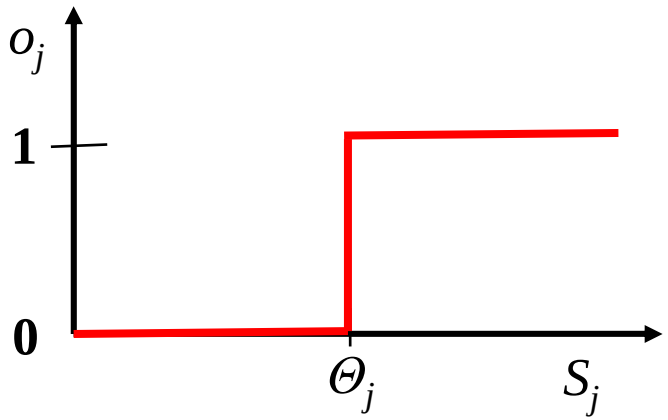
$$b = a - \gamma \nabla F(a)$$





Aprendizaje usando Gradiente

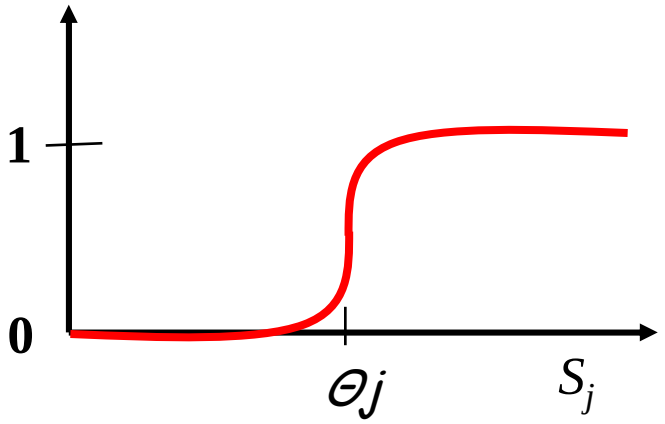
- Pero para hacer esto se necesita una función de output que sea diferenciable con respecto a sus inputs y pesos
- La función step (o sign) no es diferenciable en el umbral



Función de Output Diferenciable

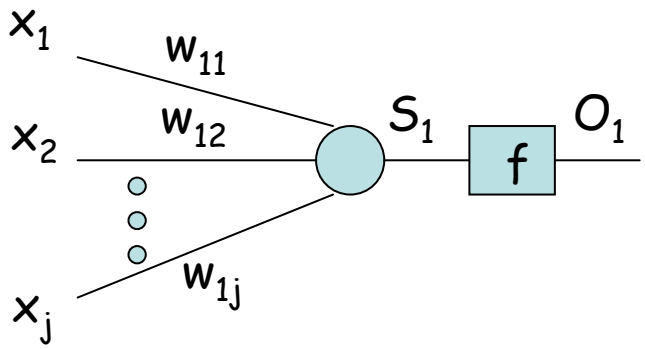
- Sigmoide es una función de activación no lineal diferenciable

$$o_j = \frac{1}{1 + e^{-(S_j - \Theta_j)}}$$



También se puede usar función de output tanh o Gausiana

ADALINE



- ADALINE (Widrow) es un modelo de una neurona:

$$S = \sum_{i=1}^{n} w_i x_i + w_0$$

- Es entrenado usando el Delta Rule o Least Mean Squared Error [Mitchell]:

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (T_d - S_d)^2$$

- Se calcula el gradiente del error E(**w**) y se modifican los pesos en la dirección contraria (D es el set de ejemplos):

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_{d \in D} 2(T_d - S_d)(-x_{id}) = -\sum_{d \in D} (T_d - S_d)(x_{id})$$

- Finalmente:

$$w_i(k+1) = w_i(k) + \Delta w_i(k) = w_i(k) - \mu \frac{\partial E}{\partial w_i} = w_i(k) + \mu \sum_{d \in D} (T_d - S_d)(x_{id})$$

Delta Learning con función no lineal sigmoide

- Ha sido usado con redes de 1 o mas capas [Kartalopoulos]
- Basado en minimización del Least Squared Error (LSE)
- Objetivo es expresar la diferencia entre el output deseado (T) y el actual (O o f(S)) en termino de los inputs y los pesos
- Utiliza la función no lineal sigmoide : $f_s(x) = 1/(1+e^{-x})$
- Su derivado : $f'(x) = [1 - f(x)] f(x)$
- Y el LMS error: $E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (T_d - f(S_d))^2$
- Se calcula el gradiente del error en termino de los pesos :

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_{d \in D} 2(T_d - f(S_d)) f'(S_d)(S') = \sum_{d \in D} (T_d - f(S_d)) f'(S_d)(-x_{id})$$

- Finalmente:

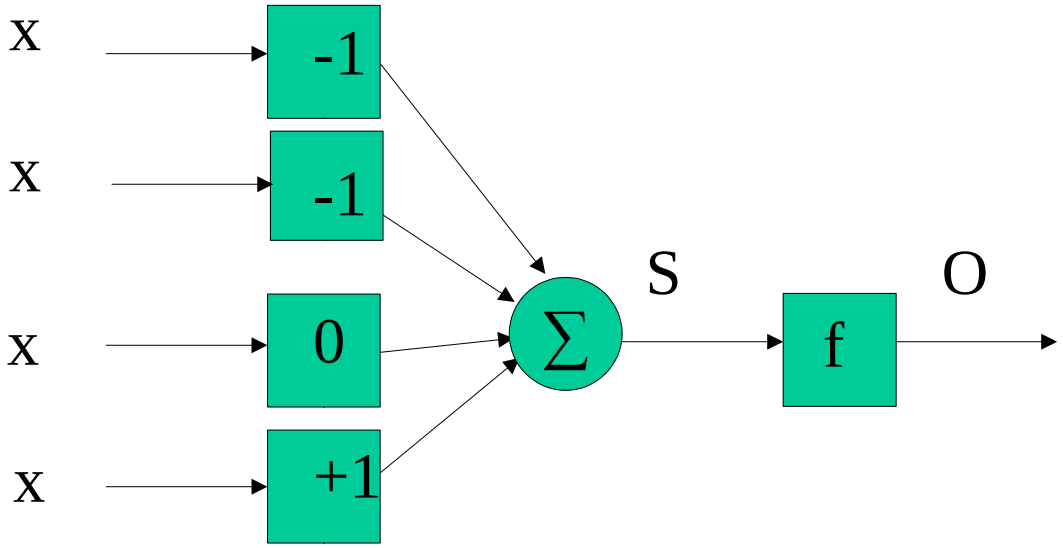
$$w_i(k+1) = w_i(k) - \mu \frac{\partial E}{\partial w_i} = w_i(k) + \mu \sum_{d \in D} (T_d - f(S_d)) f'(S_d)(x_{id})$$

Ejemplo: delta learning algorithm

- 1 neurona (i = 1), iteración inicial = 1:

  $\mathbf{x}(1)_1 = (+1, -1, +1, -1)$, $\mathbf{w}(1)_1 = (+1, -1, 0, -1)$, $T_1 = 0.55$, $\mu = 0.1$, $g = 1$, $f(S) = (1 + e^{-gS})^{-1}$, $f'(S) = [1 - f(S)]f(S)$

- Lo que se modifican son los pesos $\mathbf{w_1}$ basados en los valores de $\mathbf{x_1}$, T_1 , f(S) y μ.



Ejemplo: delta learning algorithm

- Calculando S, f(S) y modificando los pesos

  $S(1)_1 = x(1)_1 \bullet w(1)_1 = (+1, -1, +1, -1)\ (+1, -1, 0, -1)^T = 3$

  $O(1)_1 = f(S(1)_1) = f(3) = 0.952574$

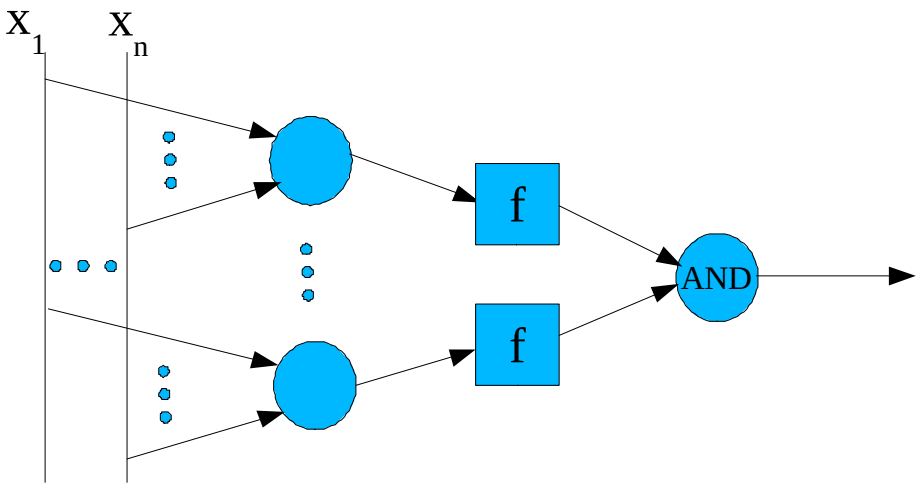
  $w(2)_1 = w(1)_1 + \mu[T - O(1)_1] f'(S(1)_1)\ x(1)_1$

  $= (0.998181, -0.998181, -0.001819, -0.998181)$
- Iteración = Iteración + 1

  basado en w(2)1 recalcular O(2)1, f '(x1w(2)1), w(3),...etc
- Parar cuando el error es menor o igual al valor deseado

MADALINE

- El modelo MADALINE usa muchos ADALINEs en parallelo y una neurona de salida
- Usa un método de selección del output basado en los inputs (Ej. mayoría o AND)

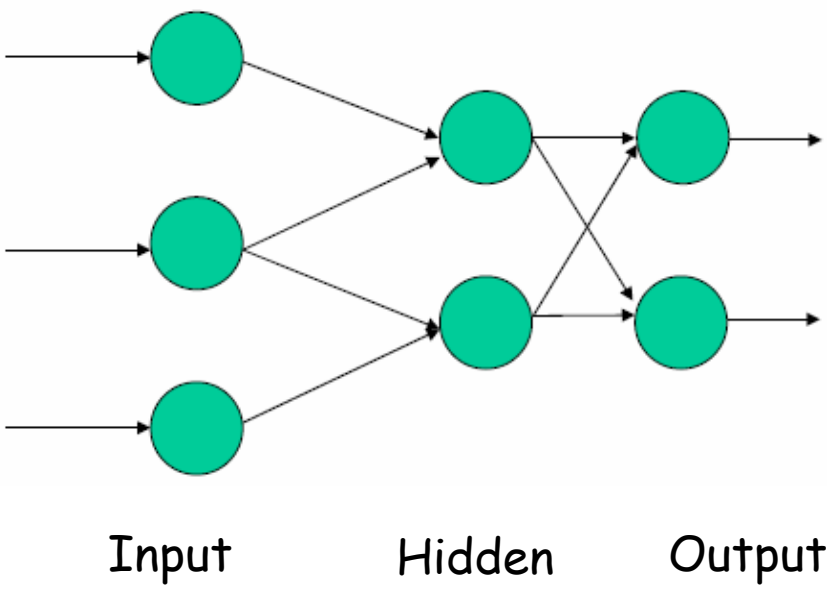


Otros Algoritmos

- Reforzado:
  - Se le indica a la red que ha pasado o fallado o sea el valor de E es P/F o 1/0
- Competitivo:
  - Hay múltiples neuronas en la capa del output y se entrena la red para que una neurona en el output este correcta y las otras no producen outputs dependiendo del estimulo

Redes Multi Capa

- Redes con múltiples capas pueden representar funciones arbitrarias
- Cualquier función Booleana puede ser representada con una capa escondida (incluyendo XOR)



Backpropagation

- Método usado para el aprendizaje de redes de múltiples capas
- Este es un algoritmo muy usado en redes neuronales de aprendizaje supervisado, la idea es modificar los pesos basado en el derivado del Error:

$$w_{ij}(k+1) = w_{ij}(k) - \mu \frac{\partial E}{\partial w_{ij}}$$

- La función total de Error se define como la suma de las diferencias al cuadrado medio para todos los patrones presentados k:

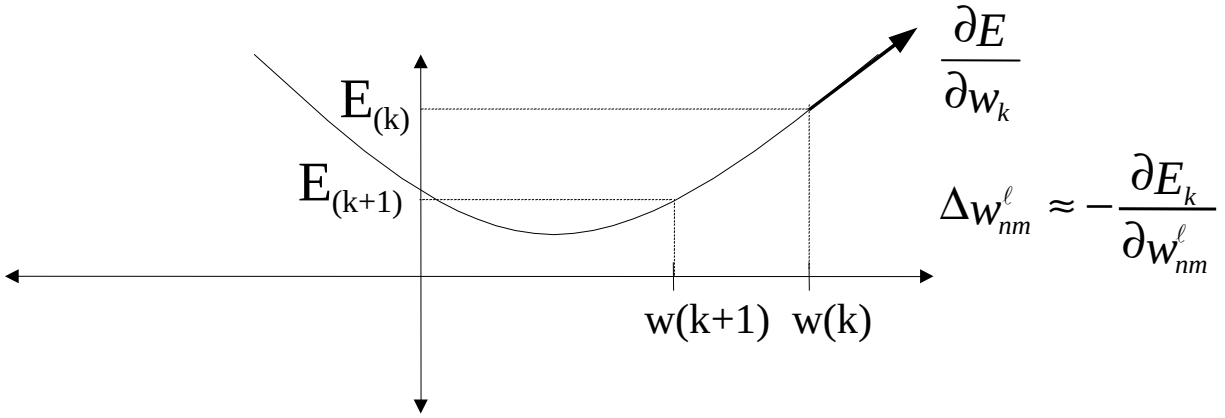
$$E = \sum_{k=1}^{K} E_k = \sum_{k=1}^{K} \left(\frac{1}{2} \sum_{i=1}^{N_L} [T_i(k) - O_i^L(k)]^2\right)$$

- Una función no lineal de transferencia típicamente usada es el sigmoide f(x):

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

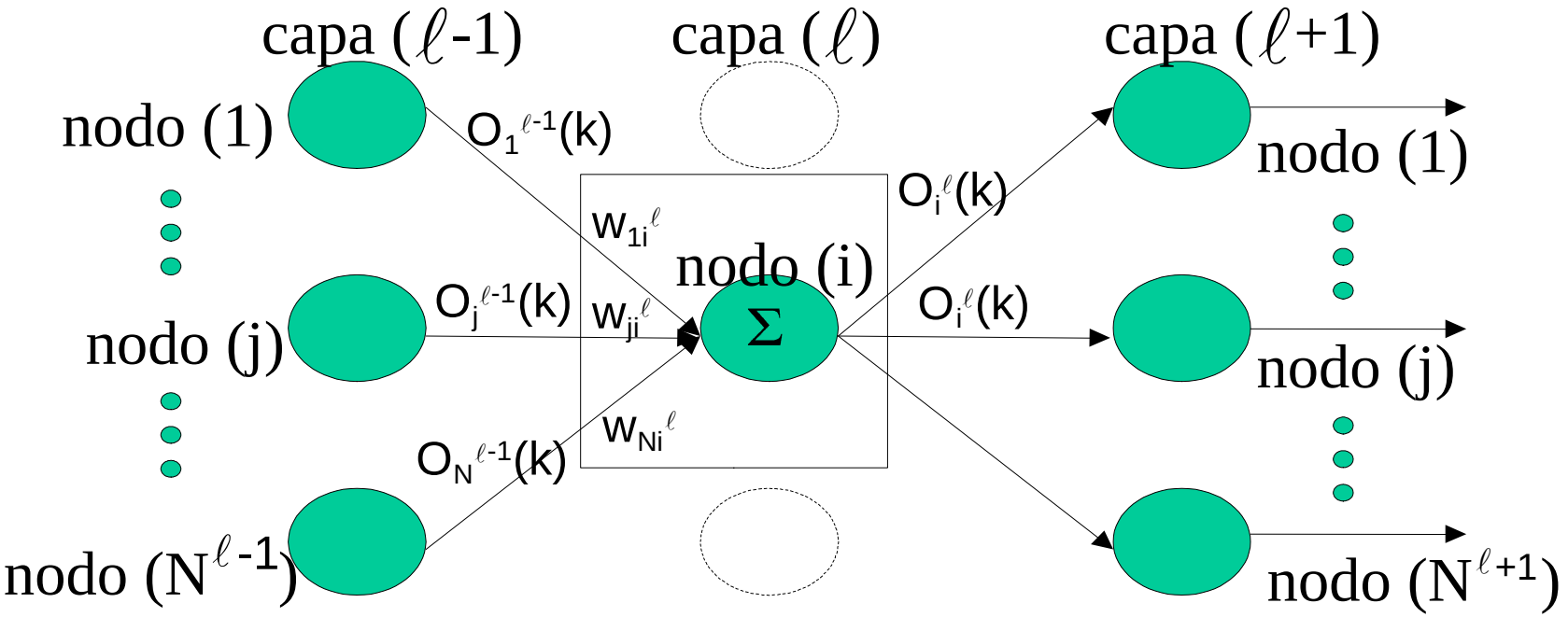
Backpropagation (cont)

- Este algoritmo (Werbos 1974) usa “gradient descent learning” que usa el gradiente del error con respecto al peso actual (w_{ij}) para la modificación del peso (Δw_{ij}) en cada nodo i en el aprendizaje del algoritmo
- En este algoritmo se empieza haciendo cómputos en la capa de output (L)
- Se procede capa a capa hacia la capa de input (1) cambiando los pesos en cada capa uno por uno



Backpropagation: modelo de la red

- a la red se le presentan K patrones de aprendizaje x_k en el cual $1 < k < K$
- el nodo i en la capa ℓ tiene como resultado del input x_k el valor $S_i^{\ell}(x_k)$
- el output del nodo i en la capa ℓ es $O_i^{\ell} = f(S_i^{\ell}(x_k))$
- el nodo j en la capa previa (ℓ -1) tienen un output de $O_j^{\ell-1}$
- el peso entre el nodo j en la capa (ℓ -1) y el nodo i en la capa ℓ es w_{ij}^{ℓ}
- la capa de output es la capa L, la capa del input es la capa 1



Algoritmo Backpropagation

1. Inicializar los pesos y los valores de bias a valores aleatorios pequeños
2. Presentar input de entrenamiento a la red (x(k), T(k)) en la generación k
3. Calcular outputs de cada neurona i ($1< i < N_\ell$) en cada capa ℓ ($1 < \ell <$L) desde el input layer y procediendo hacia el output layer (L)

$$O_i^\ell(k) = f\left(\sum_{m=1}^{N_{\ell-1}} w_{im}^\ell O_m^{\ell-1}(k)\right)$$

4. Calcular el gradiente δ_i^ℓ y la diferencia Δw_{ij} por cada input de las neuronas en cada capa desde la capa de output hasta la de input

ℓ=L $\quad \delta_i^L = (T_i - O_i^L)O_i^L(1-O_i^L), \qquad \Delta w_{ij}^L = \mu \delta_i^L O_j^{L-1}$

$\ell \neq$L $\quad \delta_i^\ell = \left(\sum_{r=1}^{N_\ell} \delta_r^{\ell+1} w_{ri}^{\ell+1}\right) O_i^\ell (1-O_i^\ell), \quad \Delta w_{ij}^\ell = \mu \delta_i^\ell O_j^{\ell-1}$

Algoritmo Backpropagation (cont)

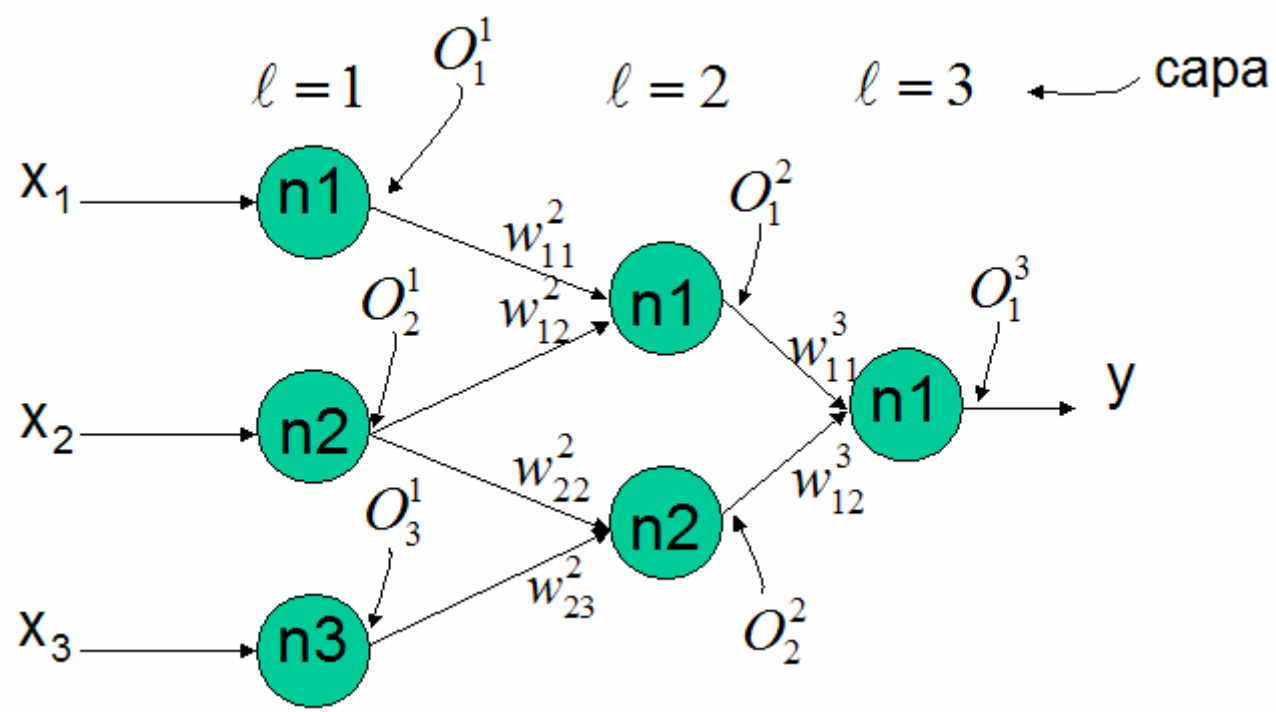
5. Actualizar los pesos de acuerdo al método de descenso de gradiente

$$w_{ij}^{\ell}(k+1) = w_{ij}^{\ell}(k) + \Delta w_{ij}^{\ell}(k)$$

6. Repetir pasos calculando los outputs de la red (próxima iteración) parar cuando el error es suficientemente pequeño

Ejemplo1: Backpropagation

- El siguiente ejemplo usa el algoritmo recientemente descrito en conjunto con la siguiente red neural usando back propagation en un feedforward NN con una capa escondida (ℓ = 2) y un output (y = O_1^3)



Ejemplo1: Backpropagation (cont)

- Inicializar los valores de los pesos en cada capa a un valor aleatorio pequeño (ej. alrededor de 1)
- Presentar inputs (ej. $k=1$, $\mathbf{x}(k)=\mathbf{x}(1)=\{1, 1, 0\}$, $T=1$, $\mu=.1$)
- Calcular outputs de cada neurona en cada capa de 1 a 3:
  - $O_1^1 = 1$, $O_2^1 = 1$, $O_3^1 = 0$
  - $O_1^2 = f(w_{11}^1 O_1^1 + w_{12}^{12} O_2^1)$, $O_2^2 = f(w_{22}^1 O_2^1 + w_{23}^2 O_3^1)$
  - $O_1^3 = f(w_{11}^3 O_1^2 + w_{12}^3 O_2^2)$
- Calcular el gradiente δ_j^ℓ y la diferencia Δw_{ij}^ℓ para cada input de las neuronas empezando por la ultima capa ($\ell=3$) hacia la primera ($\ell=1$)
  1. $\delta_1^3 = (T - O_1^3)O_1^3(1 - O_1^3)$, $\Delta w_{11}^3 = \mu\, \delta_1^3 O_1^2$, $\Delta w_{12}^3 = \mu\, \delta_1^3 O_2^2$
  2. $\delta_1^2 = (\Sigma\, \delta_r^3 w_{r1}^3)O_1^2(1 - O_1^2) = (\delta_1^3 w_{11}^3) O_1^2(1 - O_1^2)$
  3. $\delta_2^2 = (\Sigma\, \delta_r^3 w_{r2}^3)O_2^2(1 - O_2^2) = (\delta_1^3 w_{12}^3) O_2^2(1 - O_2^2)$
  4. $\Delta w_{ij}^2 = \mu\, \delta_i^2 O_j^1 \rightarrow$ $\Delta w_{11}^2 = \mu\, \delta_1^2 O_1^1$, $\Delta w_{12}^2 = \mu\, \delta_1^2 O_2^1$, $\Delta w_{22}^2 = \mu\, \delta_2^2 O_2^1$, $\Delta w_{23}^2 = \mu\, \delta_2^2 O_3^1$

Ejemplo1: Backpropagation (cont)

- Actualizar los pesos usando: $w_{ij}^{\ell}(k+1) = w_{ij}^{\ell}(k) + \Delta w_{ij}^{\ell}(k)$
- Entonces por ejemplo : $w_{11}^{2}(2) = w_{11}^{2}(1) + \Delta w_{11}^{2}(1)$
- Calcular todos los pesos desde $w_{11}^{2}(2)$ hasta $w_{12}^{3}(2)$
- Ahora calcular Y y determinar el error E(2). Si el E(2) es menor que lo deseado terminar el proceso de otra manera volver al paso 2 para repetir el proceso y generar E(3) etc...

Ejemplo2: Backpropagation

- Ej: Para que la red aprenda un set de ocho patrones x_i y T_i :
- $x_1(k)=\{0, 0, 0\}$, $T_1=1$, $x_2(k)=\{0, 0, 1\}$, $T_2=0$,
- $x_3(k)=\{0, 1, 0\}$, $T_3=0$, $x_4(k)=\{0, 1, 1\}$, $T_4=1$,
- $x_5(k)=\{1, 0, 0\}$, $T_5=1$, $x_6(k)=\{1, 0, 1\}$, $T_6=0$,
- $x_7(k)=\{1, 1, 0\}$, $T_7=1$, $x_8(k)=\{1, 1, 1\}$, $T_8=1$

- Son ocho miembros de este set: x_i, Ti, $0 \leq i \leq 8$

A- Se calculan (ver el Ej. anterior) ocho outputs: $O_1^3[i]$ y ocho errores ($E_i = | O_1^3 [i] – T_i |$) : E_1, E_2,..., E_8

C- Si el max(E_1,..., E_8) es bajo un valor entonces la red aprendió (Ej. 0.05%)

D- Si la red no aprendió se calculan los gradientes y modifican los pesos (ocho veces) basado en los T_i y O_1^3

E- Se repiten los pasos de A a D hasta que las generaciones (k) se exhausten o hasta que la red aprenda

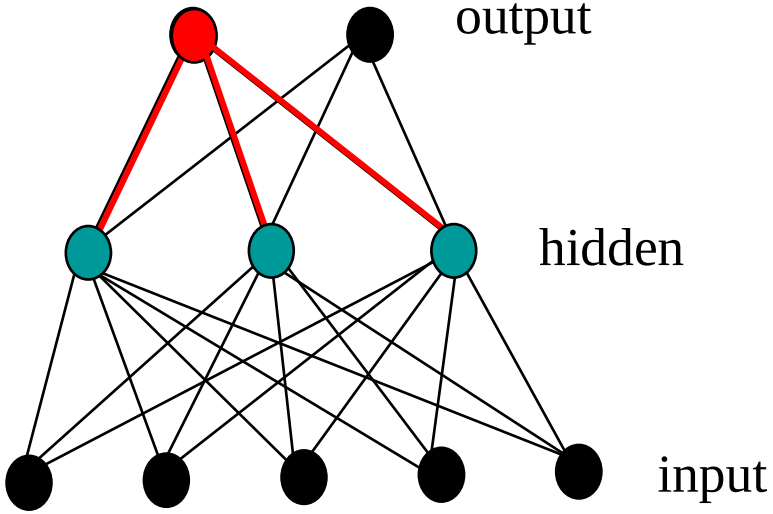
Ejemplo3: Backpropagation

- Primero calcular el error de neuronas de output y usar esto para cambiar pesos de entrada a estas neuronas

Output actual: o_i=0.2
Output correcto: t_i=1.0
Error $\delta_i^L = o_i(1–o_i)(t_i–o_i)$
0.2(1–0.2)(1–0.2)=0.128

Actualizar pesos entre nodos ***j*** *e* ***i***

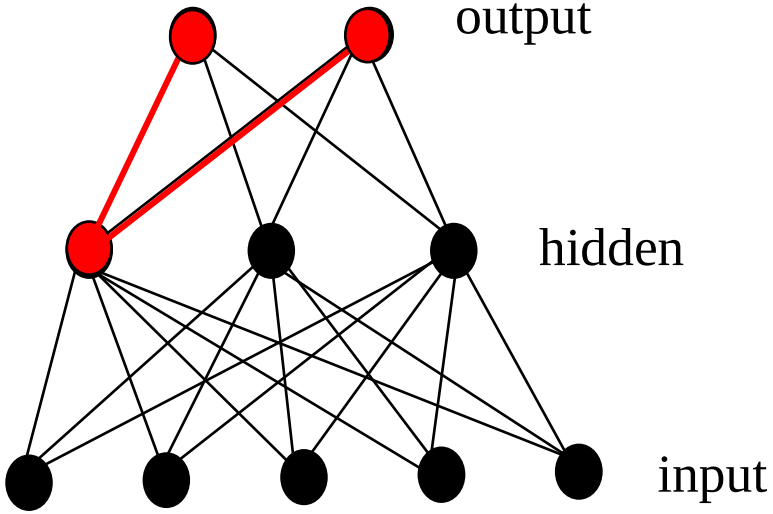
$$\Delta w_{ij} = \eta \delta_i^{\,L} o_j^{\,L-1}$$



Ejemplo3: Backpropagation

- Después se calcula el error para capas escondidas basado en los errores de unidades de output

$$\delta_i^{\ell} = o_i^{\ell}(1-o_i^{\ell})\sum_k \delta_k^{\ell+1} w_{ki}^{\ell+1}$$

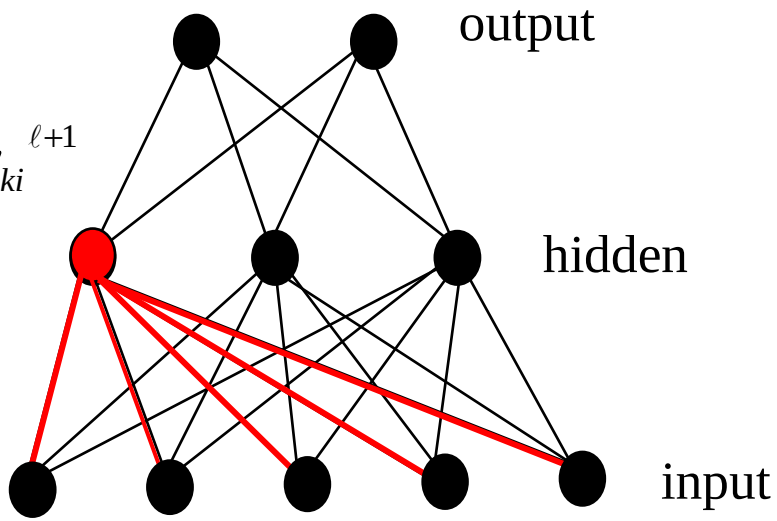


Ejemplo3: Backpropagation

- Finalmente se actualiza la capa de input de pesos basados en los errores calculados para capa escondida

$$\delta_i^{\ell} = o_i^{\ell}(1-o_i^{\ell})\sum_k \delta_k^{\ell+1} w_{ki}^{\ell+1}$$

$$\Delta w_{ij} = \eta \delta_i^{\ell} o_j^{\ell-1}$$



Algoritmo Backpropagation

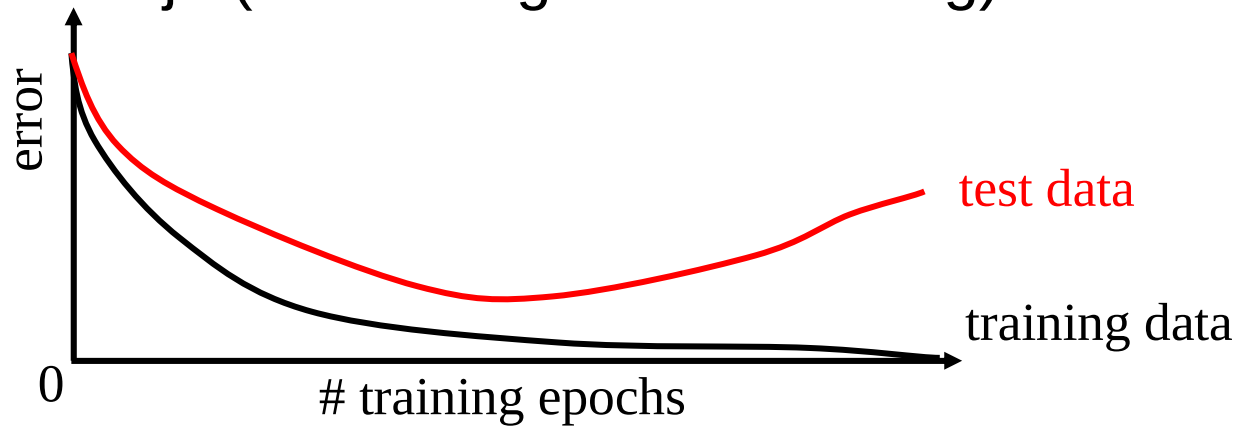
- No es garantizado que converja a zero en error de entrenamiento, puede converger a mínima local o oscilar
- En la practica converge a un bajo error para muchas redes
- Muchas épocas (miles de iteraciones) pueden ser requeridas
- Para evitar mínima local usar varias corridas (runs) con pesos inicializados aleatoriamente (*random restarts*).
  - Tomar resultados de red que da menor error

Poder de Representación

- **Funciones Booleanas**: Cualquier función booleana se puede representar con una red de dos capas con suficientes neuronas en la capa escondida
- **Funciones Continuas**: Cualquier función continua limitada (bounded) puede ser aproximada con un error arbitrariamente pequeño con una red de dos capas
  - Funciones sigmoidales puede actuar como set de funciones bases para componer funciones mas complejas (como análisis Fourier)
- **Funciones Arbitrarias**: Cualquier función puede ser aproximada con una red de tres capas

Sobre Aprendizaje (Over-Training)

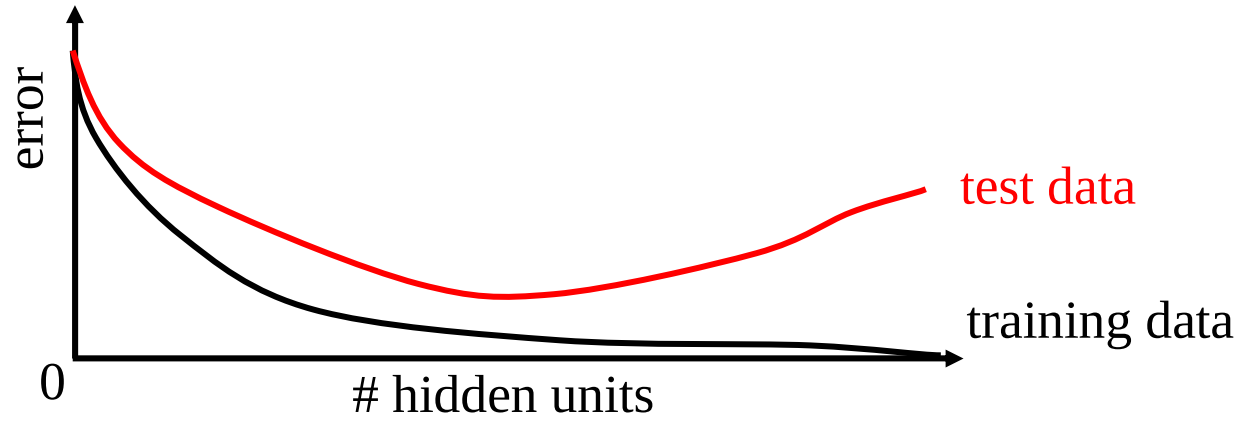
- Correr muchas iteraciones puede causar sobre aprendizaje (over-fitting o over-training).



- Hay que mantener un set de validación para probar la precisión después de cada iteración.
- Se detiene el entrenamiento cuando iteraciones adicionales incrementan el error de validación.

Determinando el Numero de Neuronas Escondidas

- Muy pocas neuronas escondidas causan que la red no pueda aprender la función deseada correctamente.
- Muchos nodos escondidos causan sobre aprendizaje.



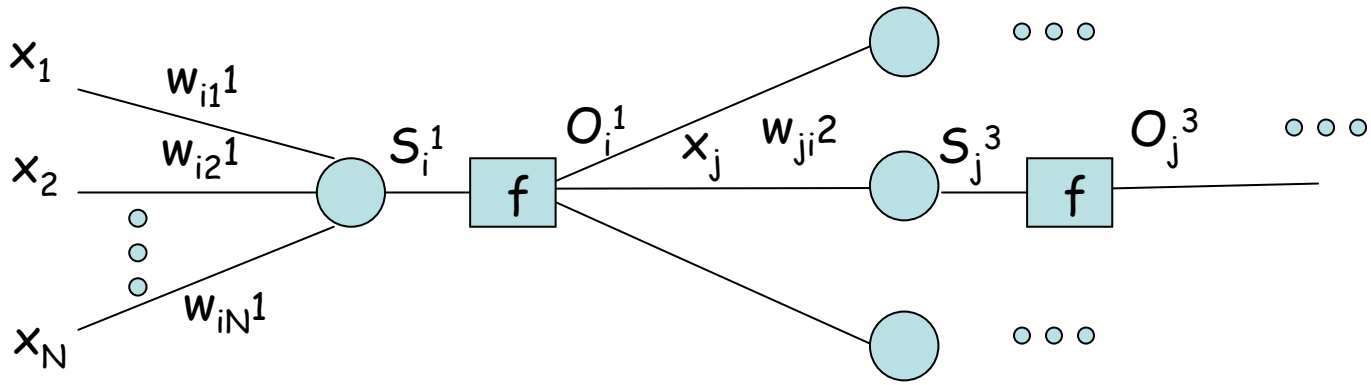
- Se debe obtener empíricamente con diferentes configuraciones de la red.

Aprendizaje no supervisado

- Hebbian learning
  - los pesos en una neurona es incrementada de acuerdo a la actividad correlacionada entre su input y actividad post sinaptica (dado su output)
  - Neuronas que disparan juntas se conectan mas fuertemente
  - En este algoritmo el valor Ti no existe
  - Los pesos tienen valores iniciales pequeños
  - Se comienza con una tasa de aprendizaje pequeña

$$\Delta w_i = \eta x_i y$$

$$y = \sum_j w_j x_j$$

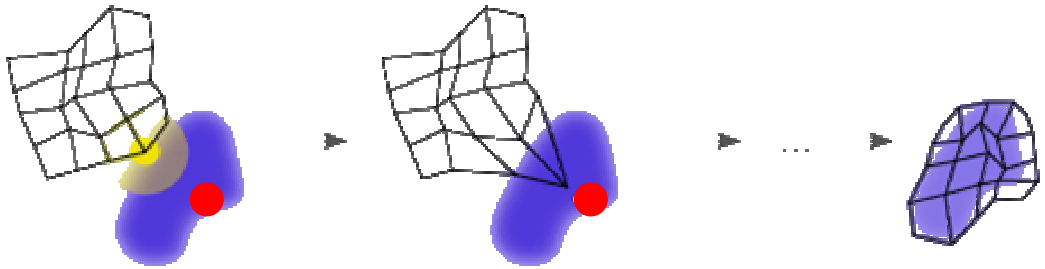


Aprendizaje no supervisado

- Self Organizing Map (SOM)
  - Introducidos por Teuvo Kohonen como Kohonen Maps, Un mapa SOM es un tipos de red neuronal que se entrena sin supervisión para producir una salida discretizada de baja dimensión de las entradas (i.e. un mapa)
  - Usan una función de vecindad para preservar propiedades del espacio de entrada
  - Las redes SOM operan en dos modos, entrenamiento y mapeo, en entrenamiento se elije la neurona que mas aproxima al vector de entrada como la solución para esa entrada y se repite iterativamente hasta tener neuronas asociadas a cada entrada
  - En la etapa de mapeo se entrega una entrada y la red la clasifica

Aprendizaje no supervisado

- Self Organizing Map (SOM)
  - El procedimiento para poner un vector desde el espacio de datos al mapa es encontrar el nodo con el vector de pesos más similar al vector de entrada y asignarle los valores del vector de entrada.



  - En el ejemplo, se ve como el nodo mas cercano (en amarillo) al vector objetivo (en rojo) es asignado ese valor, se puede ponderar el grado de acercamiento de los nodos dada su distancia (e.g. Euclideana) al vector (i.e. punto) objetivo.
  - Eventualmente la red SOM tiende a tomar los valores de los datos de entrada.

Aprendizaje no supervisado

- Self Organizing Map (SOM)
  - Usando aprendizaje competitivo, cuando un ejemplo se muestra a la red, su distancia Euclidiana a todos los vectores de pesos de los nodos se calcula.
  - El nodo (i.e. neurona) mas similar se denomina la Best Matching Unit (BMU), los pesos de la BMU y las neuronas cercanas se ajustan en dirección del vector de entrada del ejemplo.
  - La magnitud del cambio es decreciente con tiempo y distancia a la BMU. La formula de actualización para una neurona con vector de pesos Wv(t) es:

$$Wv(t + 1) = Wv(t) + \Theta (v, t)\, \alpha(t)(D(t) - Wv(t)),$$

  - donde α(t) es un coeficiente de aprendizaje decreciente monotonicamente y D(t) es el vector de entrada.
  - La función de cercanía Θ (v, t) depende de la distancia entre el BMU y la neurona v. En los casos mas simples es 1 para todas la neuronas cerca de la BMU y 0 para las otras pero una función Gaussiana es otra posible elección.

Redes Neuronales

Referencias:

[1] Jang, J-S.R. Et al, “Neuro-Fuzzy and Soft Computing”, Prentice Hall, 1997

[2] Neelakanta, P., DeGroff, D., Neuronal Network Modeling, CRC, Boca Raton, 1994

[3] Mitchel , T., “Machine Learning”, McGraw-Hill, 1997

[4] Mooney, R., Apuntes Curso University of Texas, 2006

[5] Simulador NN: www-ra.informatik.uni-tuebingen.de/SNNS/

[6] Kartalopoulos, S., Understanding Neuronal Networks and Fuzzy Logic, IEEE Press, 1994

[7] Dowla, F., Rogers, L., Solving Problems in Environmental Engineering and Geosciences with Artificial Neuronal Networks, MIT Press, 1995

[8] http://en.wikipedia.org/wiki/Hebbian_theory

[9] http://en.wikipedia.org/wiki/Self-organizing_map