Redes Neuronales

CC52A - Inteligencia Artificial

Gonzalo Ríos D.

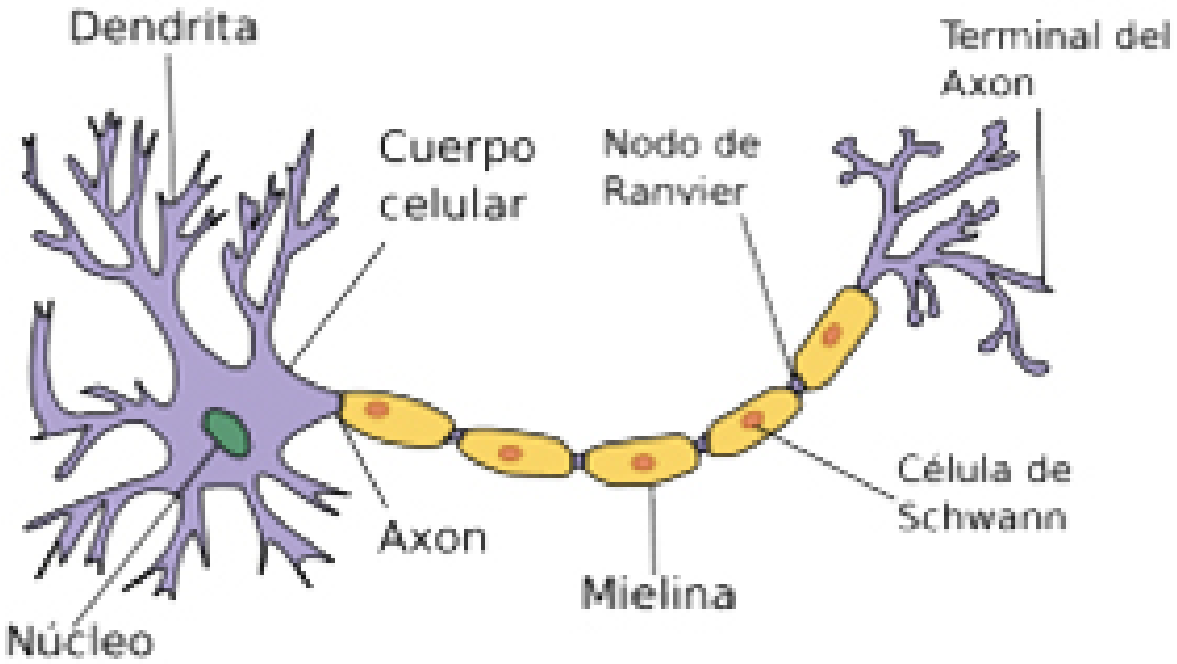
DCC - UChile

Otoño 2011

Redes Neuronales

Introducción

- Una red neuronal puede ser descrita como un modelo de regresión no lineal cuya estructura se inspira en el funcionamiento del sistema nervioso.
- Las redes neuronales emulan el comportamiento de las neuronas humanas.

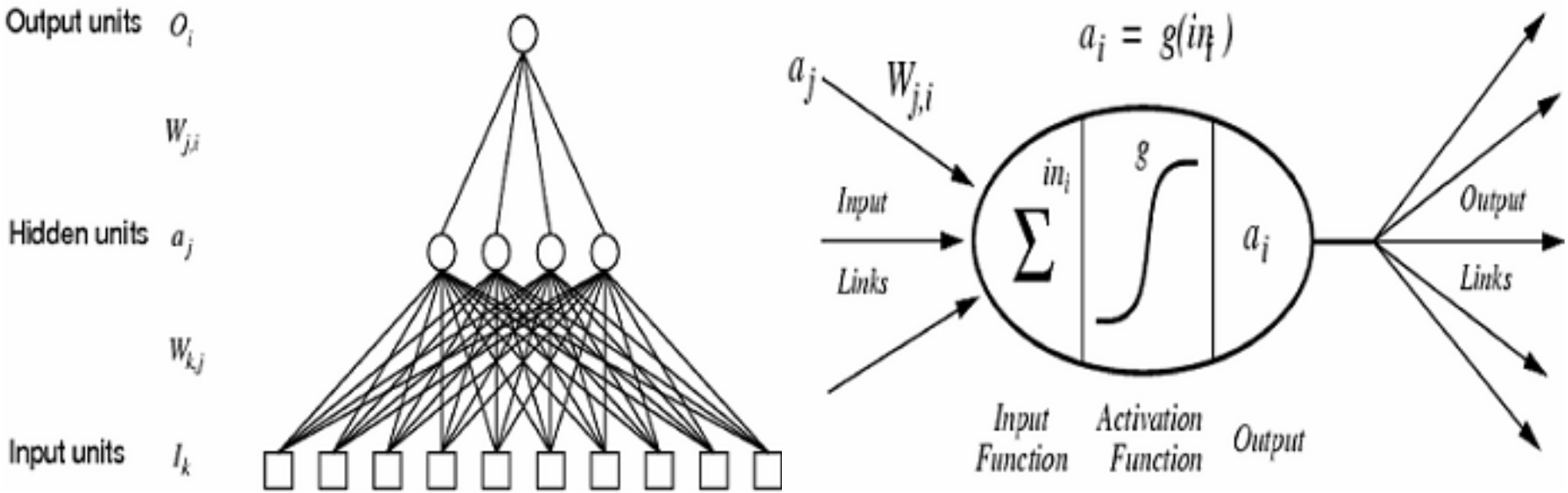


- Método general y práctico para aprendizaje supervisado
- Aprenden una función $f : X \rightarrow Y$, donde no es necesario conocer apriori su "forma".
- No tienen una interpretabilidad clara, funcionan como cajas negras.
- Requiere muchos recursos computacionales para ser entrenadas.
- La aplicación del modelo es eficiente y requiere pocos recursos computacionales.
- Funcionan con datos de entrada ruidosos y complejos.
- Aplicaciones Comunes:
  - Reconocimiento de fonemas en señales de voz
  - Reconocimiento de caracteres desde escritura manual
  - Clasificación de imágenes
  - Predicción financiera

Redes Neuronales

Estructura de una Red Neuronal

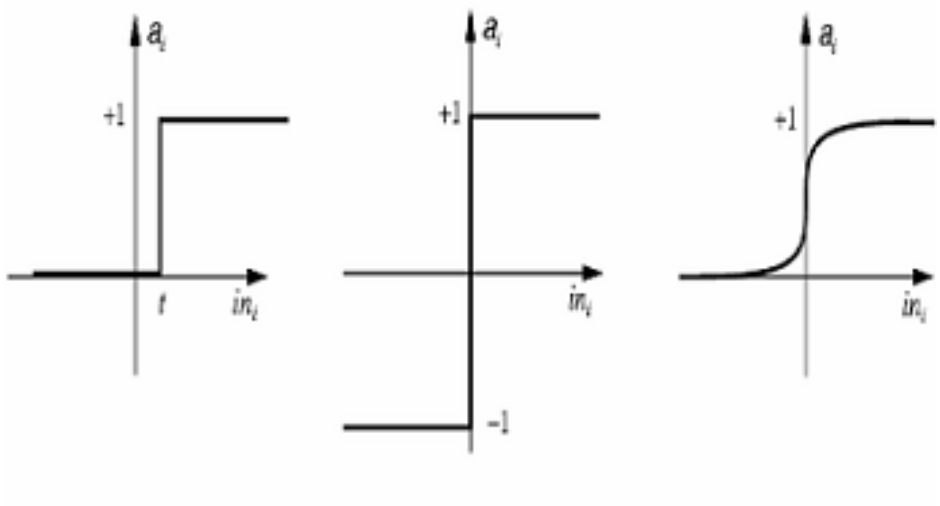
En términos generales, una red consiste en un gran número de unidades simples de proceso, denominas neuronas, que actúan en paralelo, estan agrupadas en capas y están conectadas mediante vínculos ponderados.



Cada neurona recibe inputs desde otras neuronas y genera un resultado que depende sólo de la información localmente disponible, que servirá de input para otras neuronas.

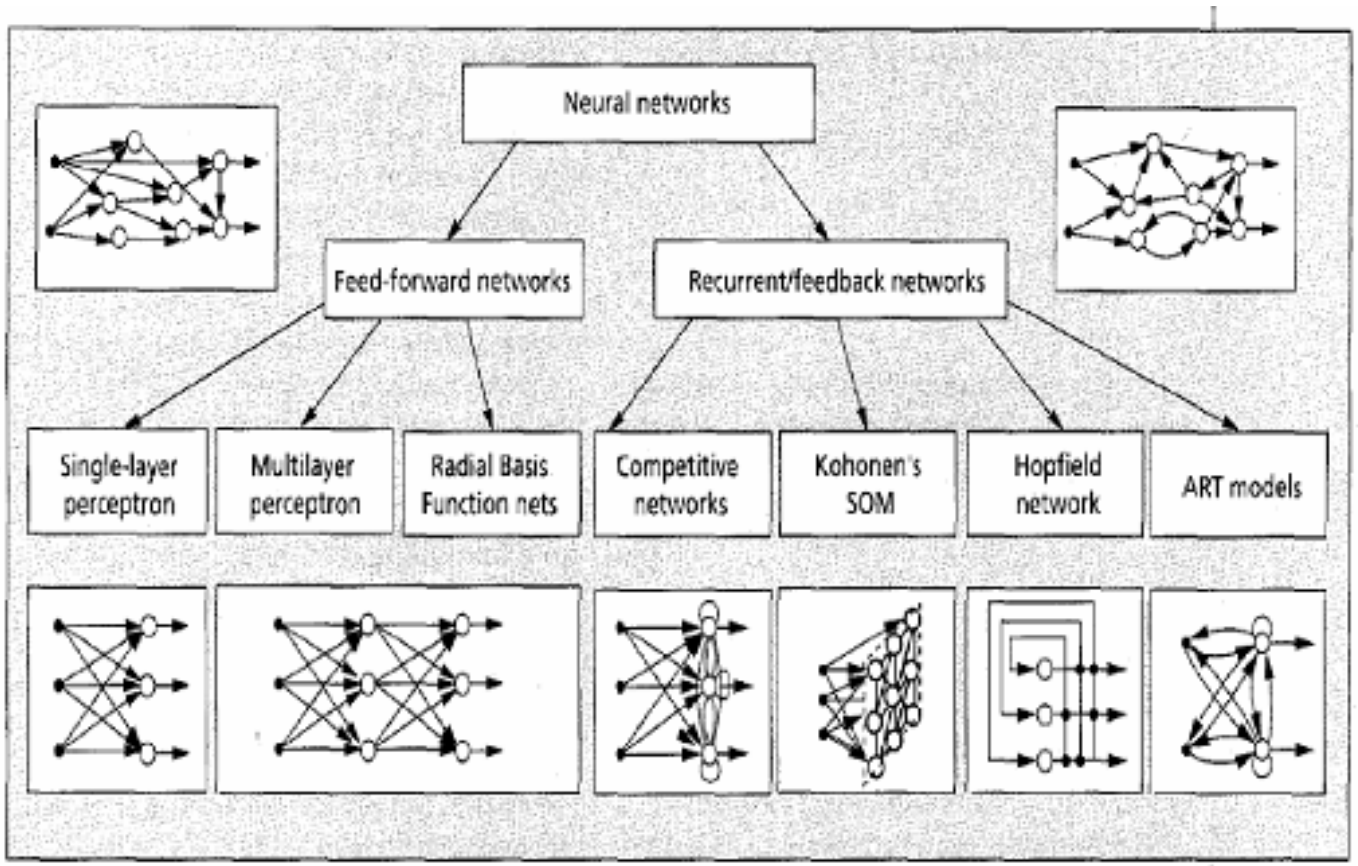
Definición

La llamada función de activación, es una función que emula el umbral presente en el sistema nervioso, que si la respuesta de una neurona no es lo suficiente mente grande, entonces esta no afecta en las siguientes neuronas. Las funciones más usuadas son la función escalón, signo, sigmoideal, gaussiana y lineal.

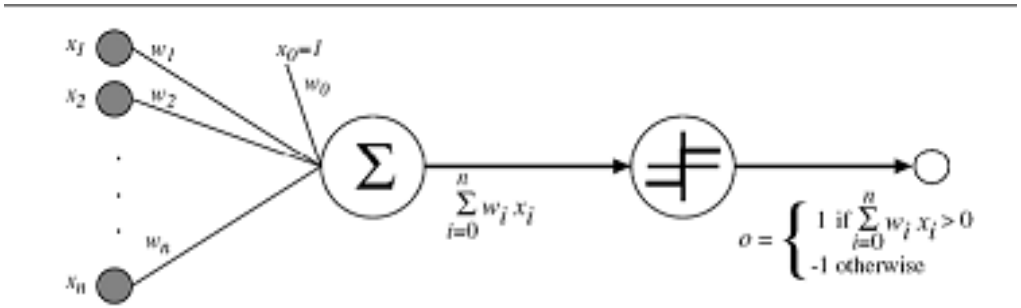


(a) Step function (b) Sign function (c) Sigmoid function

Existen diversos tipos de redes neuronales, en donde podemos distinguir dos grandes grupos: estáticas y dinámicas. Las primeras solo aprenden los "pesos", mientras que las segundas van adaptando su estructura.



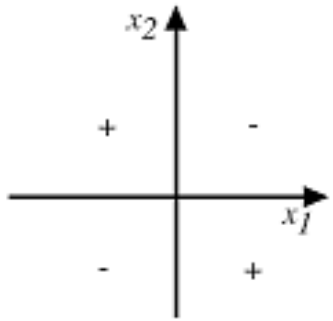
La red neuronal más básico es el perceptrón, que es un modelo que define un límite lineal de decisión para las clases {-1,1}



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

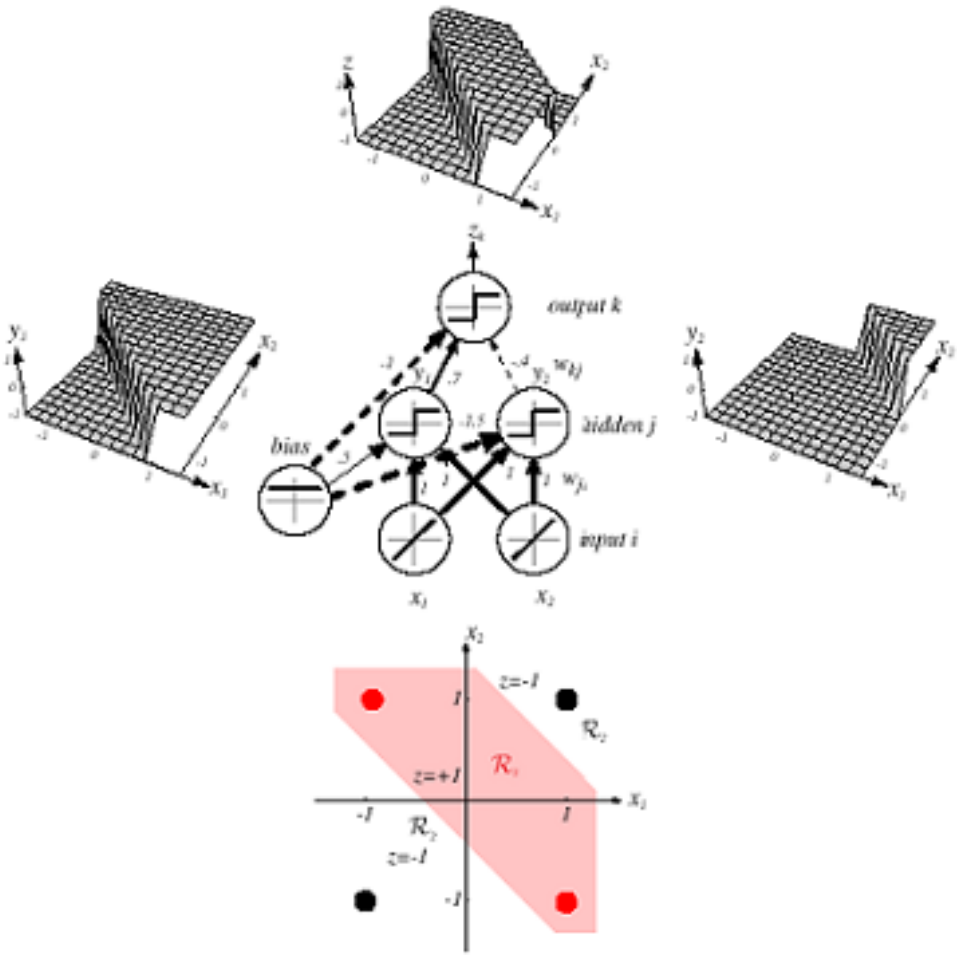
Puede representar cualquier límite de decisión lineal y la combinación de varios perceptrones lineales en un red neuronal permite modelar límites de decisión con forma de poliedros.

- Un perceptrón lineal puede representar todas las funciones Booleanas primitivas: AND,NAND, OR, NOR
- Por ejemplo, para AND basta w_0=-0,8 y w_1=w_2=0,5
- Un perceptrón no puede representar XOR



- Una combinación de perceptrones permite modelar cualquier función Booleana.

Modelación de XOR usando Perceptrones Lineales



- Cada dato de entrenamiento es un par de la forma $(\vec{x}, t)$, donde $\vec{x}$ es un vector de valores de input y t es el valor de la clase en {1,-1}.
- El problema de entrenamiento consiste en estimar los pesos w_i.

Definición

Regla de Entrenamiento de un Perceptrón Lineal

- $w_i \leftarrow w_i + \Delta w_i$
- $\Delta w_i = \eta(t - o)x_i$
- $t = c(\vec{x})$ *es el output de un dato de entrenamiento*
- o *es el output del Perceptrón*
- η *es una constante llamada tasa de aprendizaje*

Proposición

Este procedimiento converge si las clases son linealmente separables en los datos de entrenamiento y la tasa de aprendizaje es suficientemente pequeña.

Hay tres casos posibles en el entrenamiento:

1. $t - o = 0$ (no hay error)
2. $t - o = 2$ (el Perceptrón dió -1, pero el valor real era 1)
3. $t - o = -2$ (el Perceptrón dió 1, pero el valor real era -1)

Si $t - o = 2$, tenemos que actualizar los pesos de modo que ahora $\vec{w} \cdot \vec{x} > 0$, pero $\Delta w_i = \eta(t - o)x_i$ tiene el mismo signo que x_i, luego, si η es pequeño, no hay oscilación.

- El entrenamiento no converge si el límite de decisión no es lineal. Necesitamos encontrar el Perceptrón que minimice el error. Esto requiere definir noción de error para un Perceptrón.
- Consideramos sólo la unidad lineal del Perceptrón (equivalente un modelo de regresión lineal):

$$o = w_0 + w_1 x_1 + \ldots + w_n x_n$$

- Un buen modelo debe minimizar el error cuadrado

$$E(\vec{w}, D) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Usando la regla del gradiente obtenemos

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Proposición

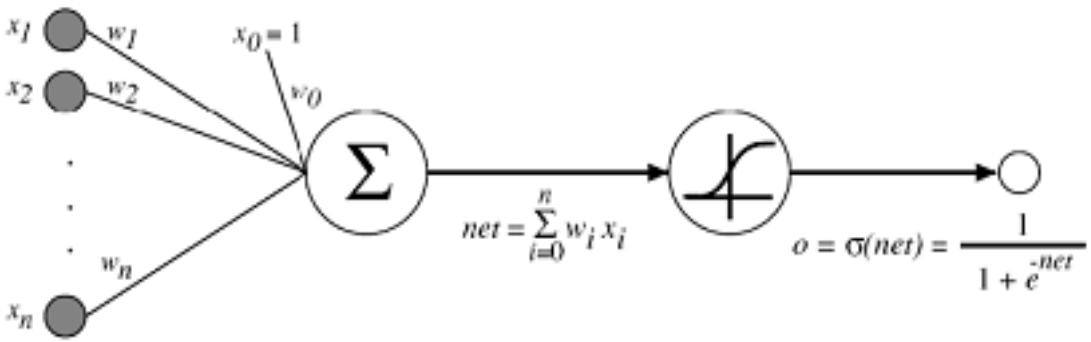
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 = \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$
$$= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) = \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x}_d)$$
$$= \sum_{d\in D}(t_d - o_d)(-x_{d,i})$$

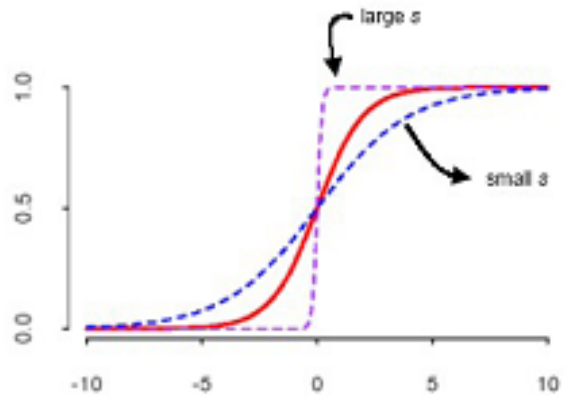
- La diferencia entre la regla de entrenamiento simple y la regla del gradiente es que en el primer caso los pesos se actualizan "dato a dato", mientras que en la regla del gradiente los pesos se actualizan usando "todos los datos" por vez.
- Si las clases son linealmente separables, el entrenamiento de un perceptrón lineal usando la regla simple converge a cero error.
- El entrenamiento de un perceptrón lineal usando la regla del gradiente converge a error cuadrático mínimo.

Un Perceptrón Sigmoidal permiten modelar funciones de salida continua, funciones de probabilidad (regresión logística)
$\sigma(x) = \frac{1}{1+e^{-x}} \in [0,1]$ es la función sigmoidal

Proposición

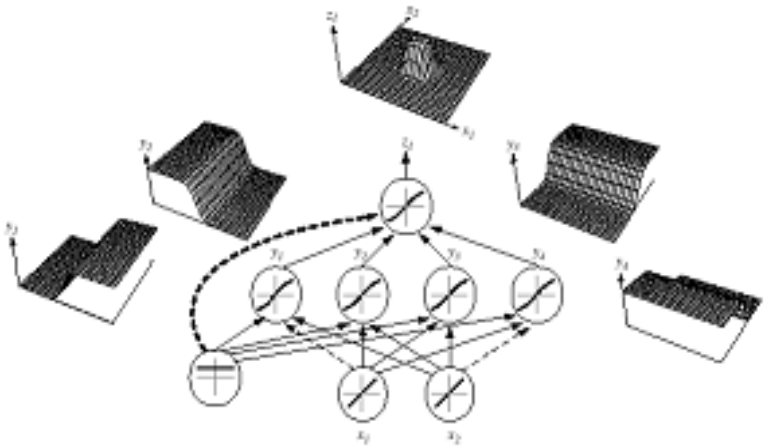
$$\frac{d\sigma(x)}{dx} = \frac{1}{1+e^{-x}}(-e^{-x}) = \sigma(x)(1-\sigma(x))$$



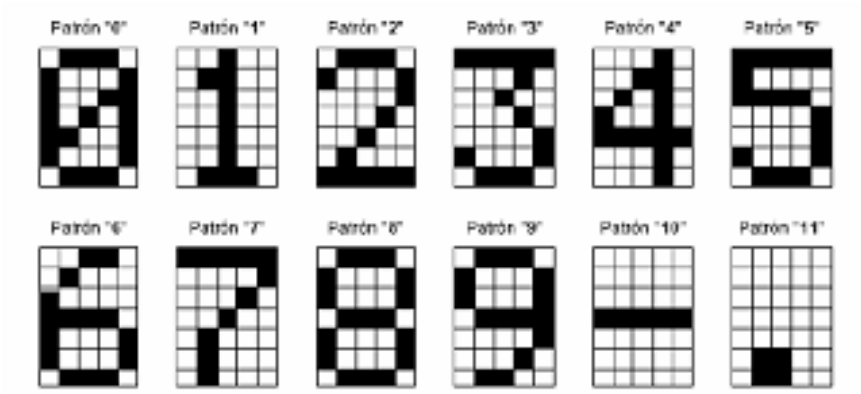


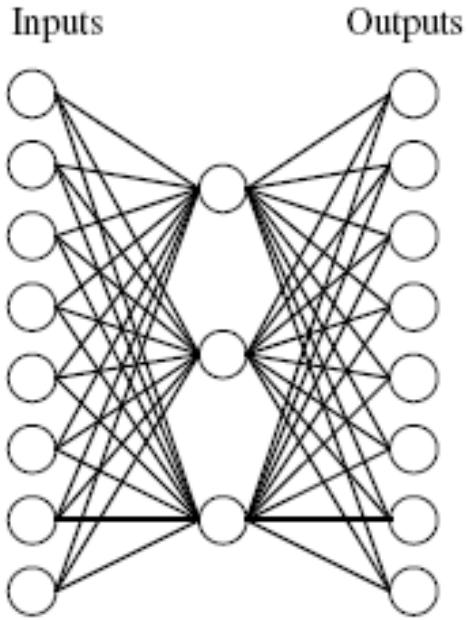
Proposición

- *Toda función booleana puede ser modelada con una red neuronal con un nivel oculto, pero con un número de unidades exponencial*
- *Toda función continuas acotada puede ser aproximada con error fijo usando una red de unidades sigmoidales de un nivel oculto*
- *Cualquier función puede ser aproximada con error fijo usando una red de unidades sigmoidales de dos niveles ocultos (regresión)*



Una aplicación típica es la de reconocimiento de patrones (clasificación)



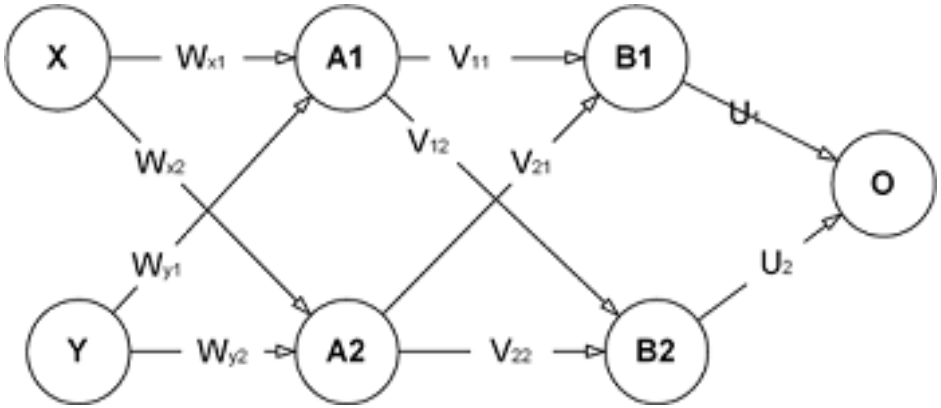


Proposición

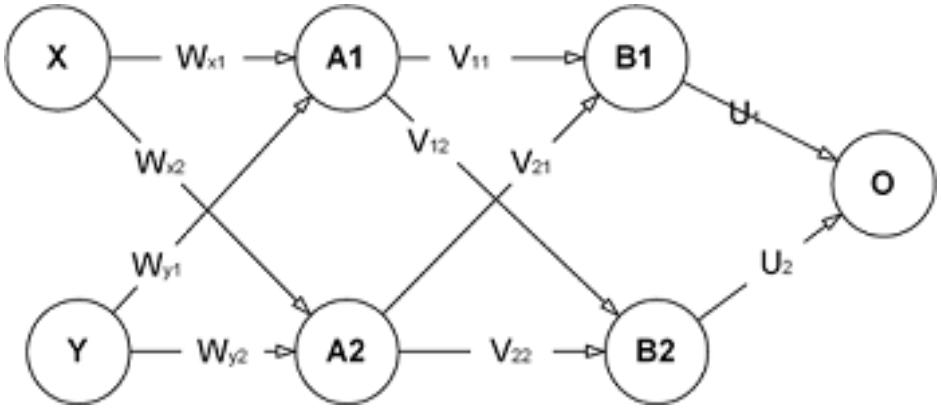
Para un Perceptrón Sigmoidal se tiene que:

$$\frac{\partial E}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{d,i}$$

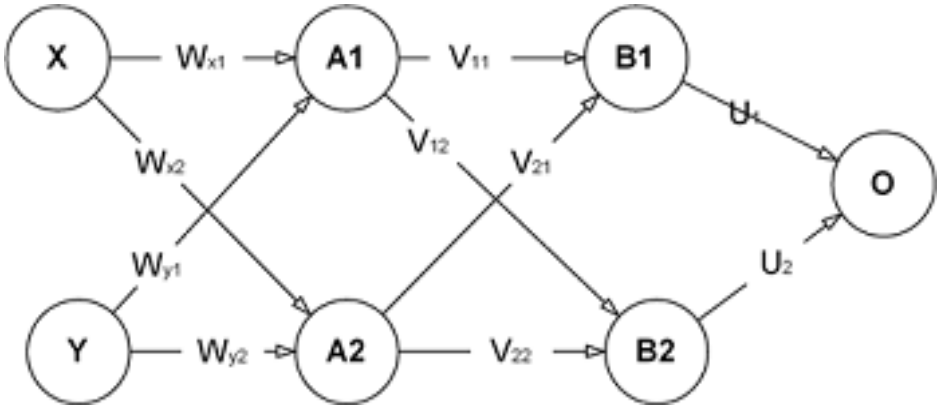
- Para una Red de Unidades Sigmoidales (o una red en general) se debe realizar un algoritmo de Retropropagación (Backpropagation).
- La idea principal del algoritmo es ir entrenando la red desde la última capa, hasta la primera capa.
- Para entrenar la última capa, se ocupa la regla de un Perceptrón Sigmoidal.
- Luego, al entrenar la capa k, se procede a entrenar la capa $k-1$ de forma recursiva, siguiendo la idea de la regla de la cadena.



- $E = \frac{1}{2}(t - O)^2$
- $\frac{\partial E}{\partial U_1} = -(t - O)\frac{\partial O}{\partial U_1}$, pero $O = \sigma(\vec{B} \cdot \vec{U}) \Longrightarrow \frac{\partial O}{\partial U_1} = O(1 - O)\frac{\partial(\vec{B} \cdot \vec{U})}{\partial U_1}$
- $\frac{\partial E}{\partial U_1} = -(t - O)O(1 - O)B_1$
- $\frac{\partial E}{\partial V_{11}} = -(t - O)\frac{\partial O}{\partial V_{11}} \Longrightarrow \frac{\partial O}{\partial V_{11}} = O(1 - O)\frac{\partial(\vec{B} \cdot \vec{U})}{\partial V_{11}}$
- $\vec{B} \cdot \vec{U} = B_1 U_1 + B_2 U_2 \Longrightarrow \frac{\partial(\vec{B} \cdot \vec{U})}{\partial V_{11}} = U_1 \frac{\partial B_1}{\partial V_{11}}$
- $B_1 = \sigma(\vec{A} \cdot \vec{V}_1) \Longrightarrow \frac{\partial B_1}{\partial V_{11}} = B_1(1 - B_1)\frac{\partial(\vec{A} \cdot \vec{V}_1)}{\partial V_{11}} = B_1(1 - B_1)A_1$



- $\frac{\partial E}{\partial W_{x1}} = -(t-O)\frac{\partial O}{\partial W_{x1}} = -(t-O)O(1-O)\frac{\partial(\vec{B}\cdot\vec{U})}{\partial W_{x1}}$
- $\vec{B}\cdot\vec{U} = B_1U_1 + B_2U_2 \Longrightarrow \frac{\partial(\vec{B}\cdot\vec{U})}{\partial W_{x1}} = U_1\frac{\partial B_1}{\partial W_{x1}} + U_2\frac{\partial B_2}{\partial W_{x1}}$
- $B_1 = \sigma(\vec{A}\cdot\vec{V}_1) \Longrightarrow \frac{\partial B_1}{\partial W_{x1}} = B_1(1-B_1)\frac{\partial(\vec{A}\cdot\vec{V}_1)}{\partial W_{x1}}$
- $\vec{A}\cdot\vec{V}_1 = A_1V_{11} + A_2V_{21} \Longrightarrow \frac{\partial(\vec{A}\cdot\vec{V}_1)}{\partial W_{x1}} = V_{11}\frac{\partial A_1}{\partial W_{x1}}$
- $A_1 = \sigma(\vec{X}\cdot\vec{W}_1) \Longrightarrow \frac{\partial A_1}{\partial W_{x1}} = A_1(1-A_1)\frac{\partial(\vec{X}\cdot\vec{W}_1)}{\partial W_{x1}}$
- $\frac{\partial(\vec{X}\cdot\vec{W}_1)}{\partial W_{x1}} = X$



$$\frac{\partial E}{\partial U_1} = -(t-O)O(1-O)B_1$$

$$\frac{\partial E}{\partial V_{11}} = -(t-O)O(1-O)U_1B_1(1-B_1)A_1$$

$$\frac{\partial E}{\partial W_{x1}} = -(t-O)O(1-O)[U_1B_1(1-B_1)V_{11}A_1(1-A_1)X + U_2B_2(1-B_2)V_{12}A_1(1-A_1)X]$$

Para cada dato de entrenamiento d

1. Computar el dato en la red y obtener los valores de output
2. Para cada neurona output k

   $$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

   End k
3. Para cada capa $j = n...1$

   Para cada neurona h de la capa j

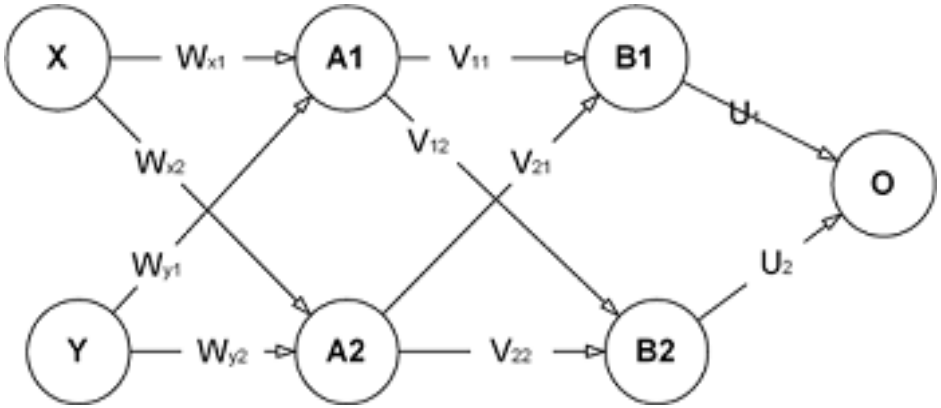
   $$\delta_h \leftarrow o_h(1-o_h) \sum_{i \in outputs(h)} w_{h,i}\delta_i$$

   End h

   End j
4. Para cada peso $w_{i,j}$

   $$w_{i,j} \leftarrow w_{i,j} + \eta \delta_j x_{i,j}$$

   End $w_{i,j}$



$$\delta_O \leftarrow O(1-O)(t-O)$$
$$\delta_{B1} \leftarrow B_1(1-B_1)U_1 O(1-O)(t-O)$$
$$\delta_{B2} \leftarrow B_2(1-B_2)U_2 O(1-O)(t-O)$$
$$\delta_{A1} \leftarrow A_1(1-A_1)[V_{11}B_1(1-B_1)U_1 O(1-O)(t-O) + V_{12}B_2(1-B_2)U_2 O(1-O)(t-O)]$$
$$\delta_{A2} \leftarrow A_2(1-A_2)[V_{21}B_1(1-B_1)U_1 O(1-O)(t-O) + V_{22}B_2(1-B_2)U_2 O(1-O)(t-O)]$$

- Minimiza error sobre datos de entrenamiento
- Encuentra un mínimo local no necesariamente global
- Tiende al Sobreajuste
- Lento de entrenar (puede tomar cientos de iteraciones)
- Complejidad: $O(d * \frac{n^2}{c})$, d: numero de datos, n: número de neuronas, c: número de capas
- Uso de la red después de entrenarse es rápido