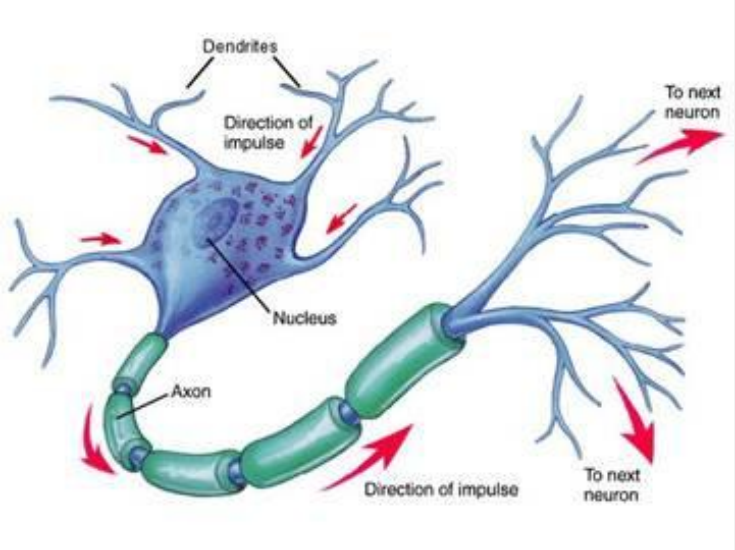
# Redes Neuronales Artificiales (RNA)

## RNA - Introducción

- Las RNA imitan en cierto modo la estructura y el modo de operación del cerebro humano.
- El cerebro está formado por millones de células llamadas neuronas. Estas neuronas biológicas son elementos simples de procesamiento, con canales de entrada de información (dendritas), un órgano de procesamiento (soma) y un canal de salida de información (axón).

# Introducción RNA - Neurona biológica

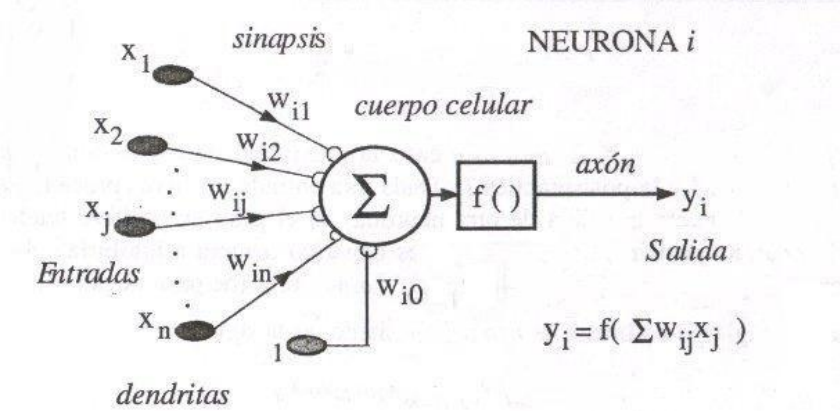



# RNA - Definición

- Las RNA son modelos basados en la interconexión de una gran cantidad de elementos simples de procesamiento (neuronas artificiales). Estos elementos de procesamiento (EP) trabajan en paralelo para obtener una eficiente performance.

- Cada neurona artificial es un procesador elemental que procesa vectores de entradas [x1, x2, x3,…….xp], comúnmente denominados patrones y produce una respuesta o salida única y.

# RNA - Neurona artificial




# RNA

- Las entradas que reciben los datos que provienen de los axones de otras neuronas corresponden a las dendritas
- Los pesos sinápticos $w_{ij}$ establecen sinapsis entre la dendrita de una neurona y el axón de otra. Cada uno de estos pesos, es un factor de importancia, un número que se modifica durante el entrenamiento de la red neuronal, y es aquí donde se almacena la información que hará que la red sirva para un propósito u otro.
- Con esas entradas y los pesos sinápticos, se suele hacer algún tipo de operación para obtener el valor de salida (valor que es función de las entradas y los pesos).

# RNA

- Las RNA tienen la capacidad de almacenar conocimientos experimentales, para su posterior utilización en la resolución de problemas reales. Este proceso tiene dos características:

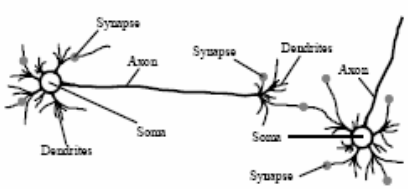
- 1) El conocimiento es adquirido por la red mediante un proceso de aprendizaje.

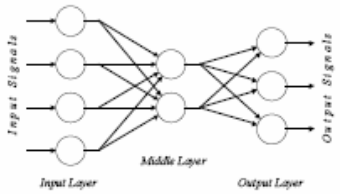
- 2) El conocimiento se almacena en la red en forma distribuida a través de los pesos que caracterizan cada conexión entre EP (se almacena en las sinapsis entre las neuronas).

# RNA

- Analogía entre una neurona biológica y una artificial:

| *Red Biológica* | *Red Artificial* |
|---|---|
| Soma | Neurona |
| Dendrita | Entrada |
| Axon | Salida |
| Sinapsis | Peso |



## Clasificación de las RNA

- La forma mas común de clasificarlas es a través del tipo de aprendizaje:

- **Aprendizaje supervisado**: en este tipo de aprendizaje se le proporciona a la RNA una serie de ejemplos, consistentes en patrones de entrada, junto con la salida que debería dar la red. El proceso de entrenamiento consiste en el ajuste de los pesos, para que la salida de la red sea lo más parecida posible a la salida deseada. Es por ello que en cada iteración se use alguna función que calcule el error o el grado de acierto que está cometiendo la red.

## Clasificación de las RNA

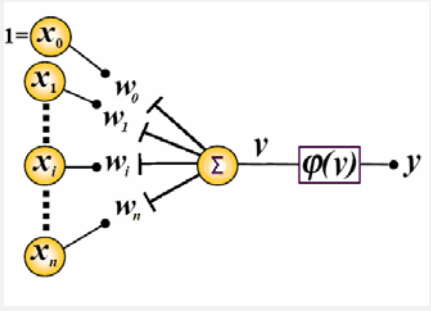
- **Aprendizaje no supervisado o autoorganizado (self-organized):** en este tipo de aprendizaje se presenta a la red una serie de ejemplos, pero no se presenta la respuesta deseada. Lo que hace la RNA es reconocer regularidades en el conjunto de entradas.

# Procesos de Aprendizaje

- La red es estimulada por el ambiente.
- La red genera cambios en su estructura en función de los estímulos recibidos.
- Con la nueva estructura, la red responde al ambiente de una manera diferente.
- Un conjunto de reglas bien definidas, que logran la solución del problema de aprendizaje de una red, se denomina algoritmo de aprendizaje.

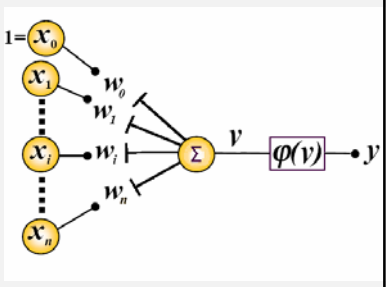
# Perceptrón Simple

- El Perceptrón Simple es una RNA formada por una sola neurona artificial, cuyas entradas son las n componentes de un vector patrón, una función de transferencia de tipo umbral no lineal y una única salida binaria.



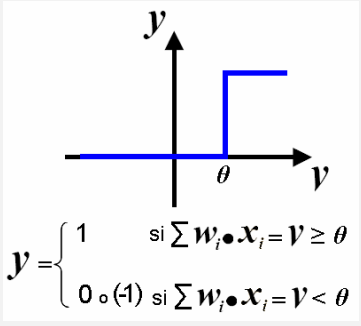
# Perceptrón Simple

- Donde:
- $x_1, x_2, \ldots\ldots x_p$ son las p componentes del vector patrón del dominio de trabajo.
- $w_1, w_2, \ldots\ldots w_p$ son los pesos correspondientes a cada componente del patrón de entrada.
- $x_0$ es la unidad de bías comúnmente toma un valor igual a 1 o -1.
- $w_0$ es el peso correspondiente a la unidad de bías.
- $v = \sum w_i \cdot x_i$
- φ función de transferencia de tipo escalón y salida única.



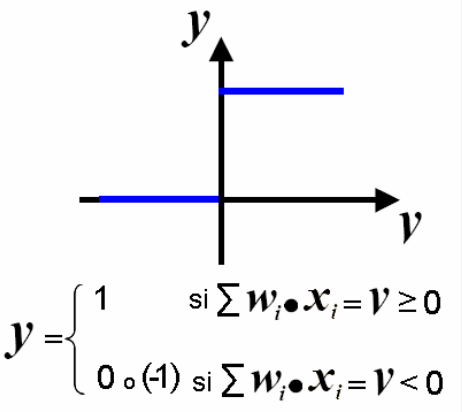

# Perceptrón Simple - Características

- El vector patrón de entrada pertenece a una clase I o una clase II.
- La función de transferencia es del tipo escalón con umbral. Esta puede tomar el valor 0 ó el 1 (también puede ser 1 ó (-1)).



## Perceptrón Simple - Características

- Para normalizar eliminamos el umbral:




## Proceso de Aprendizaje

- **Procesos de Reglas o corrección del error**
- Modifica los pesos en forma proporcional al error mismo, para corregir el error en la salida correspondiente al patrón actualmente presentado.

## Proceso de Aprendizaje

- **Proceso de Corrección del Error para el PERCEPTRON SIMPLE:**
- Es un procedimiento que ofrece la posibilidad de obtener un conjunto de pesos correcto, siempre que este grupo de pesos exista.
- Se hace realizando ajustes en los pesos, para reducir la diferencia entre las salidas actuales y las deseadas.

## Proceso de Aprendizaje

- Los pesos iniciales pueden fijarse en 0 o de manera aleatoria, usualmente en un rango pequeño, y se actualizan hasta obtener una salida consistente con los datos de entrenamiento.
- Si en la iteración n, la salida actual es y(n) y la deseada es d(n), entonces el error esta dado por:
- $e(n) = d(n) - y(n)$ ; $n = 1, 2, 3, \ldots$
- El error e(n) puede tomar el valor -1, 0 o 1.

# Proceso de Aprendizaje

- La iteración n se refiere al n-ésimo patrón de entrenamiento presentado al perceptrón.
- La modificación de los pesos se realiza de la siguiente forma:

$$w_p(n+1) = w_p(n) + \eta \cdot x_p(n) \cdot e(n)$$

- Si el error e(n) es positivo (produce un 0 cuando debería ser una salida 1), se adiciona el vector de entradas al vector de pesos existente. Pero si es negativo (produce un 1 cuando debería ser una salida 0), se resta el vector de entradas al vector de pesos existente:
- $e(n) > 0$ $\quad w_p(n+1) = w_p(n) + \eta \cdot x_p(n)$
- $e(n) < 0$ $\quad w_p(n+1) = w_p(n) - \eta \cdot x_p(n)$

Donde:

- p=0,1,2……. (p componentes del patrón de entrenamiento)
- η es el coeficiente de aprendizaje, una constante positiva menor que 1.

## Algoritmo de Entrenamiento:

- Paso 1: Inicialización

Fije los pesos iniciales $w_0, w_1, w_2, \ldots, w_p$ aleatoriamente en un rango pequeño.

## Algoritmo de Entrenamiento:

- Paso 2: Activación

Active el Perceptrón aplicando los patrones de entrada $x(n)$ de a uno por vez, y la correspondiente salida deseada $d(n)$. Calcule la salida actual en la iteración $n = 1$.

$$y(n) = \varphi ( v(n)) ,$$

$$v(n) = \sum w . x ,$$

## Algoritmo de Entrenamiento:

- Paso 3: Entrenamiento de pesos

$$w(n+1) = w(n) + \Delta w(n)$$

*$\Delta w(n)$ es la corrección del peso en la iteración n.*

$$\Delta w(n) = \eta \cdot x(n).e(n)$$

$$e(n) = d(n) - y(n)$$

## Algoritmo de Entrenamiento:

- Paso 4: Iterar
- Luego de la iteración 1, vaya al paso 2 y repita el proceso hasta que converja.

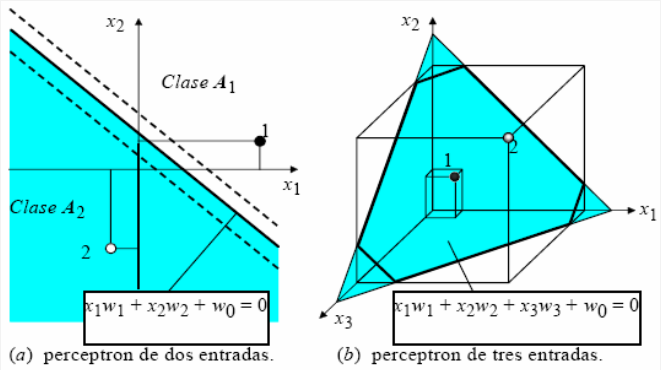
- El valor de $\eta$ no afecta la estabilidad del método de convergencia del Perceptrón y afecta al tiempo de convergencia solamente en los casos donde los valores iniciales de los pesos sean igual a cero.

# Objetivo del perceptrón

- El objetivo del perceptrón es clasificar entradas, $x_1$, $x_2$, . . ., $x_p$, en dos clases, por ejemplo A1 and A2.

# Separabilidad lineal de clases

- En el caso de un Perceptrón elemental, el espacio n-dimensional es dividido por un hiperplano en dos regiones de decisión. El hiperplano es definido por la función:

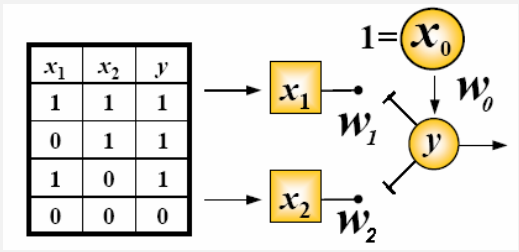


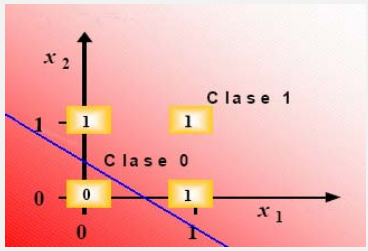

(*a*) perceptron de dos entradas. (*b*) perceptron de tres entradas.

# Ejemplos de Aplicación

- Lógica OR

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |

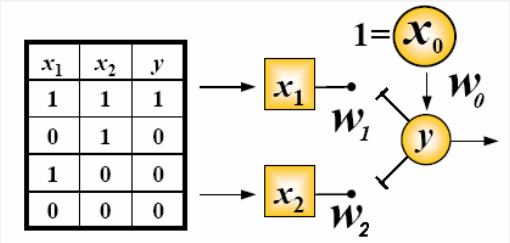


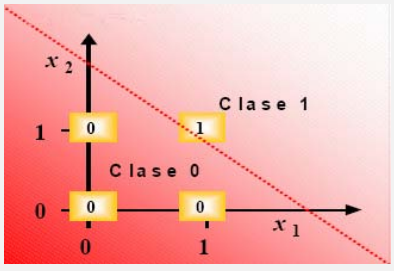

# Ejemplos de Aplicación

- Lógica AND

| $x_1$ | $x_2$ | $y$ |
|---|---|---|
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |

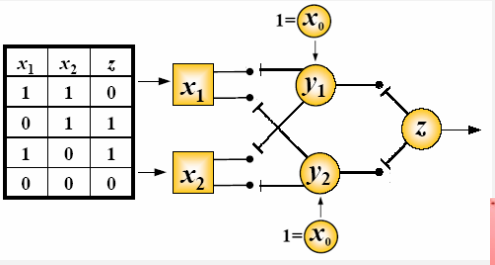



Como se puede apreciar, en el caso de la lógica “and” y la “or” estamos en presencia de problemas que son linealmente separables y pueden ser resueltos por el perceptrón simple.



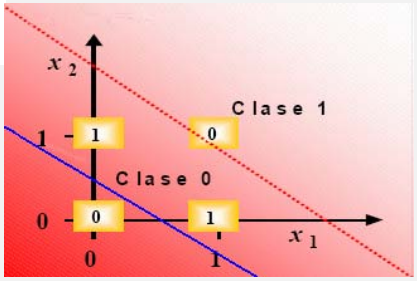
# Ejemplos de Aplicación

- Lógica XOR



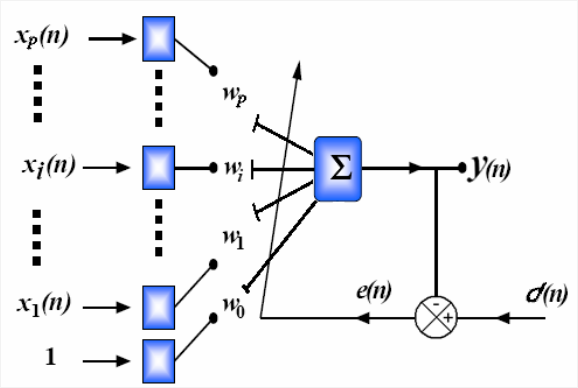

| $x_1$ | $x_2$ | $z$ |
|---|---|---|
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |

Estamos en presencia de un problema que no es linealmente separable, este no puede ser resuelto por un Perceptrón Simple.




# Adaline

- El ADALINE que por sus siglas en inglés significa ADAptive LINear Element consta de un solo elemento de procesamiento.



# Adaline

- La estructura de la adaline es similar a la del perceptrón, sólo que su función de transferencia es lineal, en lugar del escalón.
- Se alimenta con un vector de entrada y con una entrada constante igual a 1 denominada bías. Posteriormente se efectúa una suma ponderada de los valores de entrada con sus pesos asociados.
- Al igual que el perceptrón, sólo puede resolver problemas linealmente separables.
- Se usa ampliamente en aplicaciones de procesamiento de señales.

# Entrenamiento

- **Método del gradiente descendente:**
  - Se busca disminuir el error cuadrático medio J, el cual esta en función de los pesos:
  - Donde N es la cantidad de patrones de entrenamiento y *E* es la esperanza.

$$\mathrm{J} = \frac{1}{2} E\left[\mathrm{e}^2(n)\right] = \frac{1}{2}\left[\frac{1}{N}\sum_{n=1}^{N} \mathrm{e}^2(n)\right]$$

# Entrenamiento

- Donde p es la cantidad de componentes del vector patrón (se esta considerando la unidad de bías).

$$J = \frac{1}{2} E[e^2_{(n)}] = \frac{1}{2} E[(d_{(n)} - y_{(n)})^2] \quad ; \quad y_{(n)} = \sum_{k=0}^{p} w_k * x_{k(n)}$$

$$J = \frac{1}{2} E[(d_{(n)} - \sum_{k=0}^{p} w_k \cdot x_{k(n)})^2]$$

$$= \frac{1}{2} E[(d^2_{(n)} - 2 \cdot d_{(n)} \cdot \sum_{k=0}^{p} w_k \cdot x_{k(n)} + (\sum_{k=0}^{p} w_k \cdot x_{k(n)})^2)]$$

# Entrenamiento

$$J = \frac{1}{2} E[(d^2_{(n)} - 2 \cdot d_{(n)} \cdot \sum_{k=0}^{p} w_k \cdot x_{k(n)} + \sum_{j=0}^{p} \sum_{k=0}^{p} w_k \cdot w_j \cdot x_{k(n)} \cdot x_{j(n)})]$$

$$J = \frac{1}{2} E[d^2_{(n)}] - \sum_{k=0}^{p} w_k \cdot E[d_{(n)} \cdot x_{k(n)}] + \frac{1}{2} \sum_{j=0}^{p} \sum_{k=0}^{p} w_k \cdot w_j \cdot E[x_{k(n)} \cdot x_{j(n)}]$$

$$J = \frac{1}{2} \mathbf{r}d - \sum_{k=0}^{p} w_k \cdot \mathbf{r}dx_{(k)} + \frac{1}{2} \sum_{j=0}^{p} \sum_{k=0}^{p} w_k \cdot w_j \cdot \mathbf{r}x_{(j,k)}$$

# Entrenamiento

- El objetivo del método es minimizar el error mediante la modificación del vector de pesos ($w_0$, ..., $w_n$) sumándole un $\Delta\mathbf{w}$ de tal forma que nos acerquemos al error mínimo en la dirección del gradiente negativo, es decir, lo más rápidamente posible (En este caso η es el coeficiente de aprendizaje).

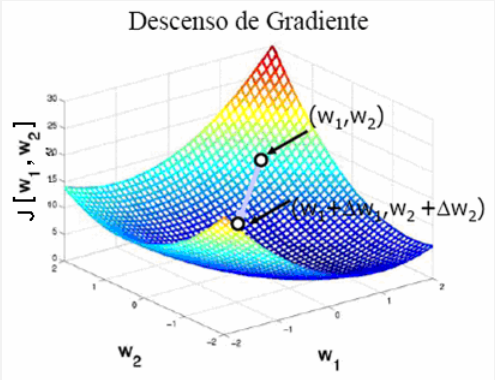
$$\Delta w_k(n) = -\eta \cdot \nabla_{W_k} J$$

$$w_k(n+1) = w_k(n) - \eta \cdot \nabla_{w_k} J$$

$$w_k(n+1) = w_k(n) + \eta \cdot \left( r dx_{(k)} - \sum_{j=0}^{p} w_{j(n)} \cdot r x_{(j,k)} \right)$$

# Entrenamiento – Descenso del Gradiente

- La utilización de este método no es factible debido a que no puedo calcular el valor de los factores "r". Entonces el algoritmo que utilizamos es el LMS.



# Algoritmo LMS

- La regla LMS (Least Mean Square). Es una aproximación del método del gradiente descendiente.
- En este caso se estiman los valores de los r:

$$\hat{r}dx_{(k,n)} = x_j(n) \cdot x_k(n)$$

$$\hat{r}x_{(j,k,n)} = x_k(n) \cdot d_{(n)}$$

$$\hat{w}_{k\,(n+1)} = \hat{w}_k(n) + \eta \cdot \left(d_{(n)} - \sum_{j=0}^{p} \hat{w}_{j(n)} \cdot x_j(n)\right) \cdot x_k(n)$$

$$\hat{w}_{k\,(n+1)} = \hat{w}_k(n) + \eta \cdot \left(d_{(n)} - y_{(n)}\right) \cdot x_k(n)$$

$$\hat{w}_{k\,(n+1)} = \hat{w}_k(n) + \eta \cdot e_{(n)} \cdot x_k(n)$$

# Algoritmo LMS

- La convergencia depende de η, malos valores de η pueden causar lentitud u oscilación.

$$\Delta \overline{w} = \eta \cdot e_{(n)} \cdot \overline{x}(n)$$

# Algoritmo LMS

- 1-) Inicializar pesos ($w_0, w_1, ..., w_p$).
- 2-) Presentar el primer vector de entrada ($x_1,...,x_p$) y la salida deseada d correspondiente.
- 3-) Calcular la salida y.

- 4-) Adaptar los pesos:

$$w_k(n+1) = w_k(n) + \eta.e(n).x_k(n).$$

- 5-) Repetir los pasos 2 a 4 hasta que el error sea aceptable.

Siguiendo este método se garantiza que, para un conjunto de entrenamiento adecuado, después de un número finito de iteraciones el error se reduce a niveles aceptables. El número de iteraciones necesarias y el nivel de error deseado dependen de cada problema particular.