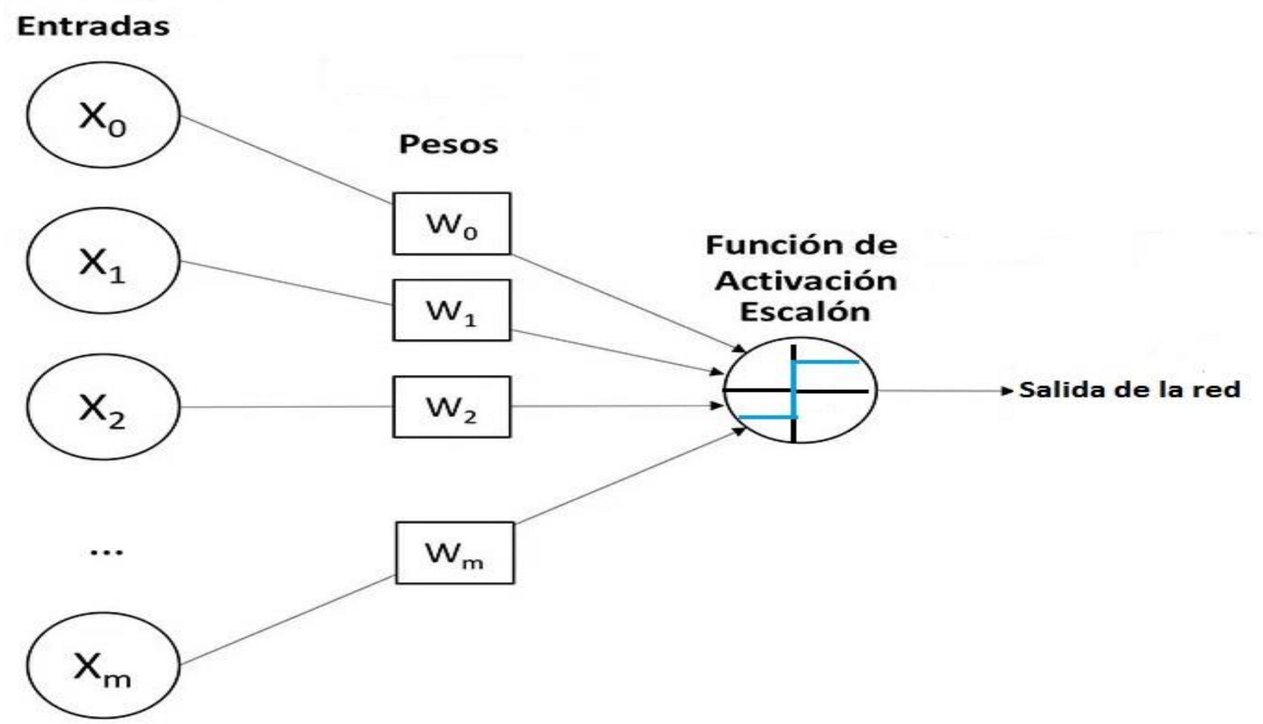
Algoritmos de Redes Neuronales

Algoritmo de Red Perceptron

Red monocapa con conexiones hacia adelante con aprendizaje supervisado.



1) Inicialización de pesos W_i $i = 1$ a m

Se asignan valores aleatorios a cada uno de los pesos W_i de las conexiones

2) Presentación de un nuevo par (entrada, salida esperada)

Presentar un patrón de entrada X_p $(x_1, x_2, \ldots\ldots, x_m)$ junto con la salida esperada $d_p(t)$

3) Cálculo de la salida actual

$Y_p(t) = f[\Sigma\, w_i(t)\, x_i(t)]$ $i = 1$ a m

siendo $f[\ ]$ la función de activación en este caso la **función escalón**

4) Adaptación de los pesos

$$W_i(t+1) = W_i(t) + \alpha\, [\underbrace{d_p(t) - y_p(t)}_{\text{error}}]\, x_i(t)\,]$$

5) Volver al paso 2

El proceso se repite hasta que el

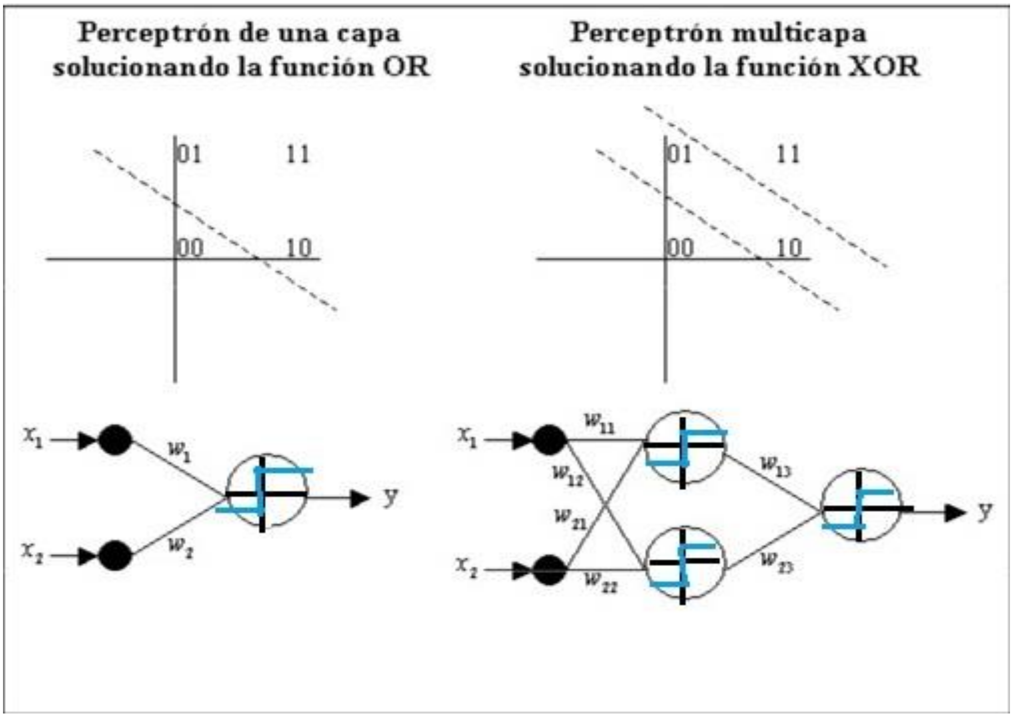
error = salida esperada d(t) – salida de la red y(t)

sea 0 para todas las entradas.

A tener en cuenta:

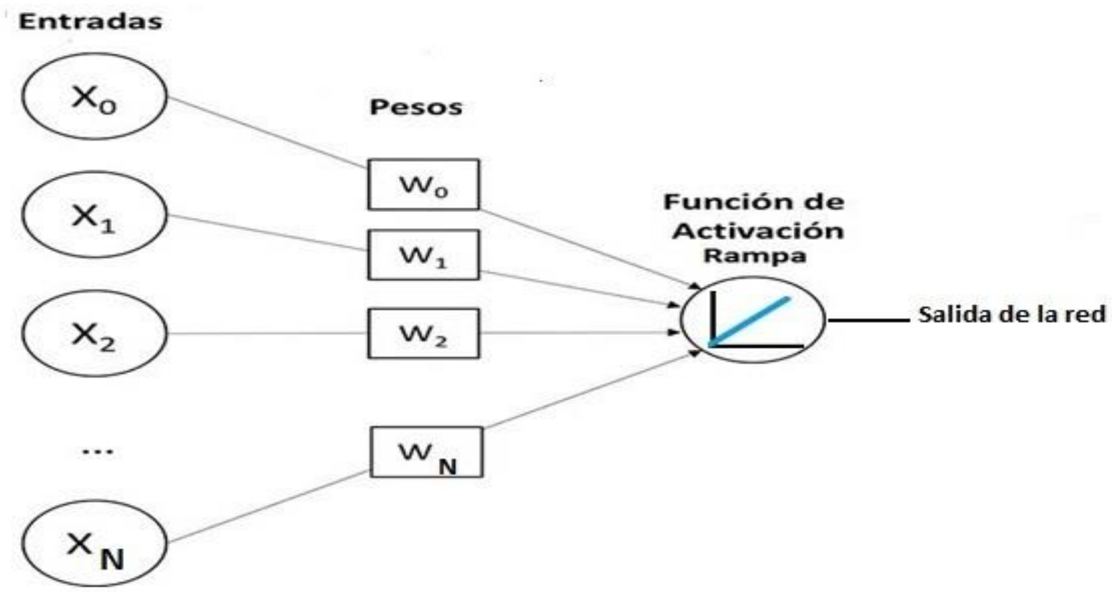
La red perceptron utiliza la función de activación escalón sólo tiene en cuenta si se ha equivocado o no.

Ejemplo de red Perceptron mono y multicapa



Algoritmo de la red Adaline (ADAptive LINear Element)

Red monocapa con conexiones hacia adelante con aprendizaje supervisado.



$X_k = (X_1, X_2, \ldots\ldots, X_N)$ K = 1 a L L es la cantidad de entradas.

1) Inicialización de pesos Se asignan **valores aleatorios** a cada uno de los pesos W_j de las conexiones j = 1 a N

2) Presentación de un nuevo par (entrada, salida esperada) Presentar un patrón de entrada X_k con k = 1 a L L es la cantidad de entradas.

3) Cálculo de la salida actual

$S_k = f\,[\Sigma\, w_j\, x_{kj}\,]$ de j = 1 a N (j es la cantidad de componentes de cada entrada)

siendo f [ ] la función de activación en este caso la **función rampa**

$e_k = (d_k - S_k)$

3) Se actualizan los pesos

$W_j(t+1) = w_j(t) + \alpha\,[e_k\; x_{kj}]$

Siendo α la tasa de aprendizaje

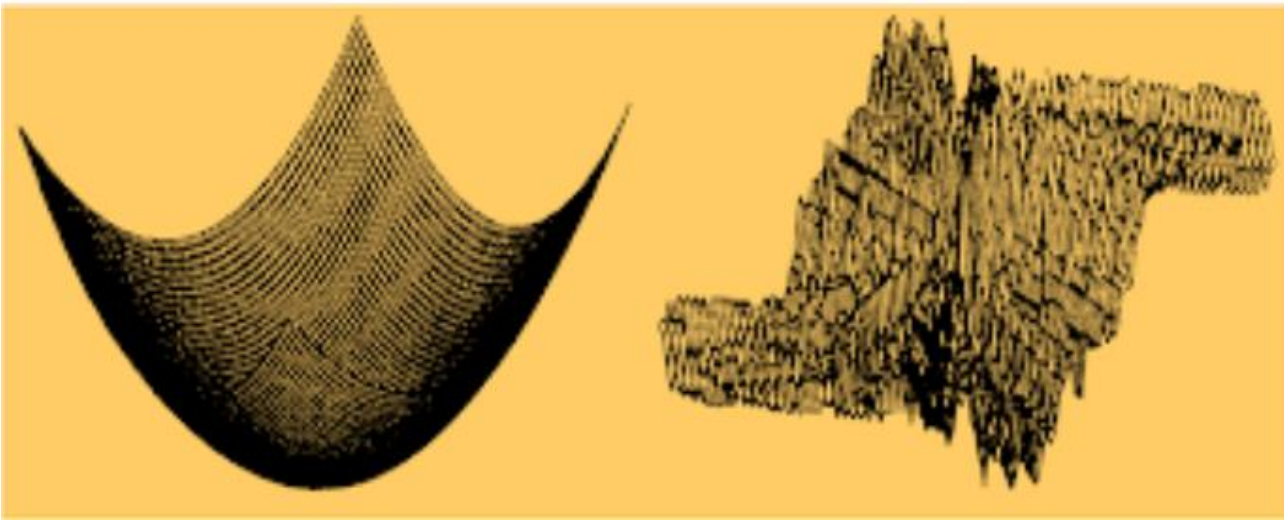
4) Se repiten los pasos 1 al 3 con todos los vectores de entrada L

5) Si el error cuadrado medio:

$<\mathrm{E}^2>_k = (1 / 2L)\, \Sigma\, e^2_k$ k = 1 a L , L es la cantidad de entradas.

es un valor aceptable termina el algoritmo, sino se vuelve al paso 1.

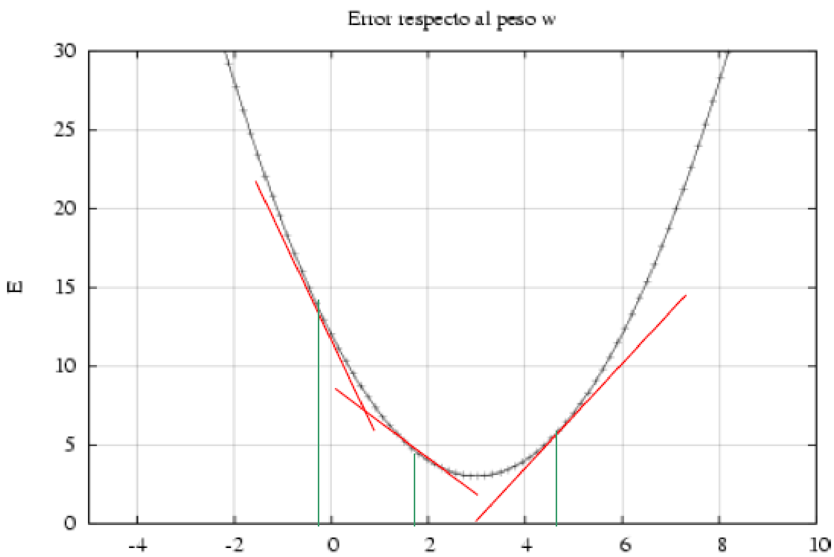
Hay una gran diferencia entre el cálculo del error en la red perceptron y en adaline ya que el error en la red perceptron se calcula como la diferencia entre la salida deseada y la salida de la red siendo esta última binaria con lo cual sólo tiene en cuenta si se ha equivocado o no.
En el caso de adaline el error es un valor númerico y permite medir **cuanto se ha equivocado** la red, siendo el cálculo del error a través del error cuadrado medio.



Superficie del error Red Adaline Superficie del error Red Perceptron

La regla de aprendizaje de ADALINE es la regla Delta, que busca el conjunto de pesos que minimiza la función de error, la idea es realizar un **cambio en cada peso proporcional a la derivada del error**, medida en el patrón actual, respecto del peso:

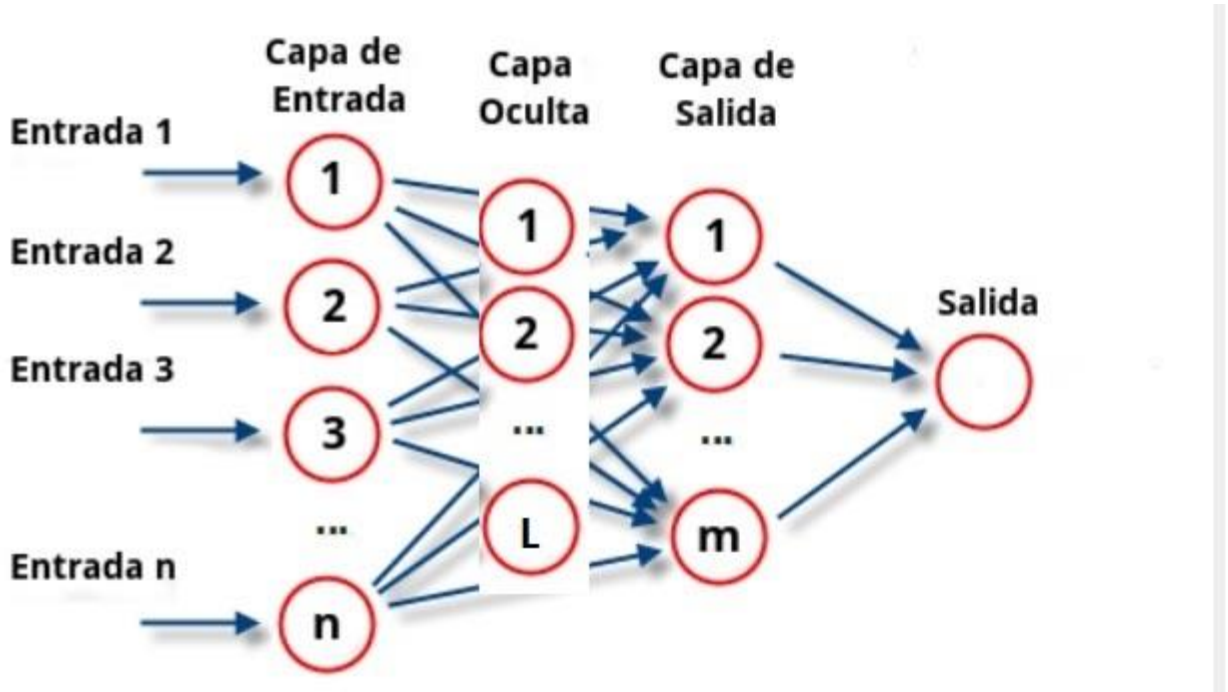
$$\Delta_p w_j = -\gamma \frac{\partial E^p}{\partial w_j}$$



Aplicando la regla de la cadena queda:

$$\Delta_p w_j = \gamma (d^p - y^p) x_j$$

Algoritmo de red backpropagation red multicapa con conexiones hacia atrás con aprendizaje supervisado



1) Establecer el error aceptable para cada entrada

2) Inicializar los pesos de la red con valores pequeños aleatorios

3) Presentar la entrada $X_p = (x_{p1}, x_{p2}, \ldots\ldots, x_{pn})$ i= 1 a n, y especificar la salida deseada que debe generar la red: $d_1, d_2, \ldots\ldots, dm$

4) Calcular la salida de la red, primero las salidas de la **capa oculta**

$$net^h_{pj} = \Sigma\, w^h_{ji}\, x_{pi} \qquad i = 1 \text{ a } n$$ (cantidad de componentes de cada entrada)

$$j = 1 \text{ a } L$$ (cantidad de neuronas ocultas)

h número de capa oculta p p-esimo vector de entrada j j-esima neurona oculta

$$y_{pj} = f^h_j\,(net^h_{pj})$$

Se realizan los mismos cálculos para obtener las salidas de las neuronas de la **capa de salida**

$$net^o_{pk} = \Sigma\, w^o_{kj}\, y_{pj} \qquad j = 1 \text{ a } L$$ (cantidad de neuronas ocultas)

$y_{pk} = f^{o}_{k} (net^{o}_{pk})$ k= 1 a m (cantidad de las neuronas de salida)

5) Calcular el error para la **capa de salida**

$$\delta^{o}_{pk} = (d_{pk} - y_{pk})\, f^{o\prime}_{k} (net^{o}_{pk})$$

Calcular el error de la **capa oculta**

$$\delta^{h}_{pj} = f^{h\prime}_{j} (net^{h}_{pj}) \sum \delta^{o}_{pk}\, w^{o}_{kj}$$ k= 1 a m (cantidad de neuronas de salida)

El error en una neurona oculta es **proporcional a la suma de los errores conocidos** que se producen en las neuronas de salida ponderada por el peso.

6) Actualización de los pesos

Para las neuronas de la **capa de salida**:

$$W^{o}_{kj}(t+1) = W^{o}_{kj}(t) + \alpha\, \delta^{o}_{pk}\, y_{pj}$$ k= 1 a m (cantidad de neuronas de salida)

y para los pesos de las neuronas de la **capa oculta**

$$W^{h}_{ji}(t+1) = W^{h}_{ji}(t) + \alpha\, \delta^{h}_{pj}\, x_{pi}$$ j = 1 a L (cantidad de neuronas ocultas)

6) El proceso se repita hasta que el término de error

$$E_p = ½ \sum (\delta^{o}_{pk})^2$$ para k = 1 a m

resulta aceptable pequeño para cada entrada

Esta red backpropagation utiliza la regla delta de aprendizaje generalizada.