# Control Inteligente

## Fundamentos de las Redes Neuronales
## Parte II
## L7

Luis Moreno, Santiago Garrido, Dorin Copaci

Dpto. Ing. de Sistemas y Automática
Universidad Carlos III
Madrid

Oct 2019

# Table of contents

Some of the figures used in this presentation are from [Haykin2009]

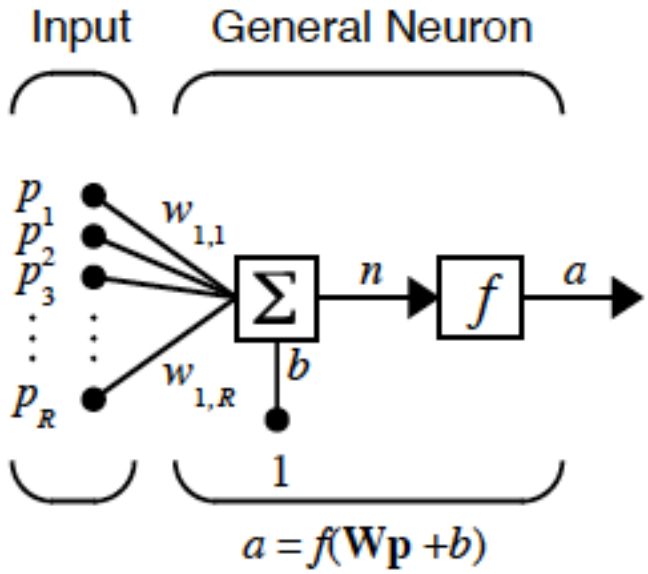
# Redes feedforward

## Perceptron multicapa

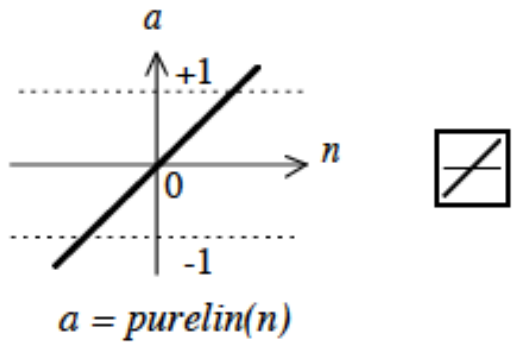
- Hemos visto anteriormente el perception de Rosenblatt, que es una red neuronal de una capa que estaba limitada a la clasificación de patrones linealmente separables.
- El entrenamiento de esta red se hace usando el algoritmo de mímimos cuadrados de Widrow y Hoff (LMS).
- Para superar las limitaciones del perceptron y del algoritmo de aprendizaje LMS se propuso una nueva estructura de red neuronal denominada perceptrón multicapa. Se caracteriza por:
  - El modelo de cada neurona en la red incluye una función de activación no-lineal que es diferenciable.
  - La red contiene una o más capas que están ocultas, y no se observan desde la entrada o salida
  - La red tiene un alto grado de conectividad cuya extensión viene determinada por los pesos sinápticos de la red.

# Redes feedforward

- Modelo de neurona




- F. de transferencia utilizadas:
  - Lineal



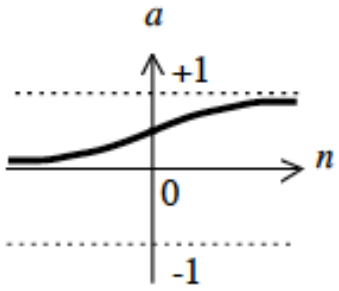
# Redes feedforward

## Funciones de transferencia no lineales

- Función logística:

$$f(n) = \frac{1}{1 + e^{-\alpha n}} \qquad (1)$$

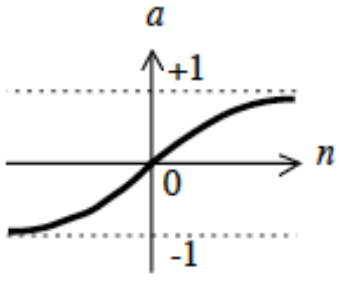
donde $\alpha$ normalmente toma valor 1.




*a = logsig(n)*

- Función tangente hiperbólica:

$$f(n) = \tanh(n) \qquad (2)$$




*a = tansig(n)*

Estas no linealidades son suaves y derivables lo que será necesario en el aprendizaje.

Derivadas de las funciones no lineales:

$$n_i = w_i^T . x_i + b_i$$

$$o_i = f(n_i)$$

- Función logística:

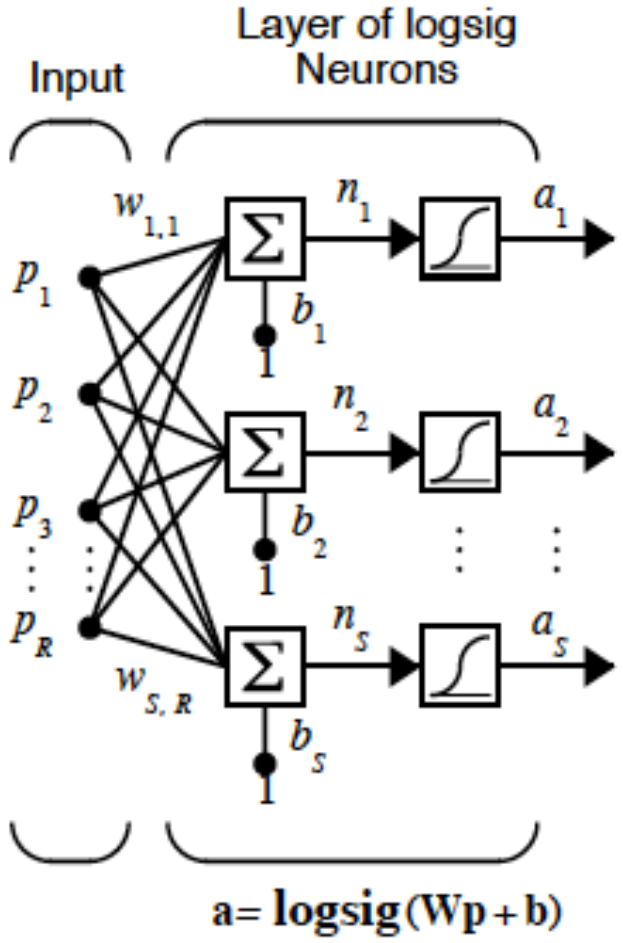
$$f'_{log}(n_i) = o(1 - o_i) \tag{3}$$

- Función tangente hiperbólica:

$$f'_{tanh}(n_i) = \frac{1}{2}(1 - o_i^2) \tag{4}$$

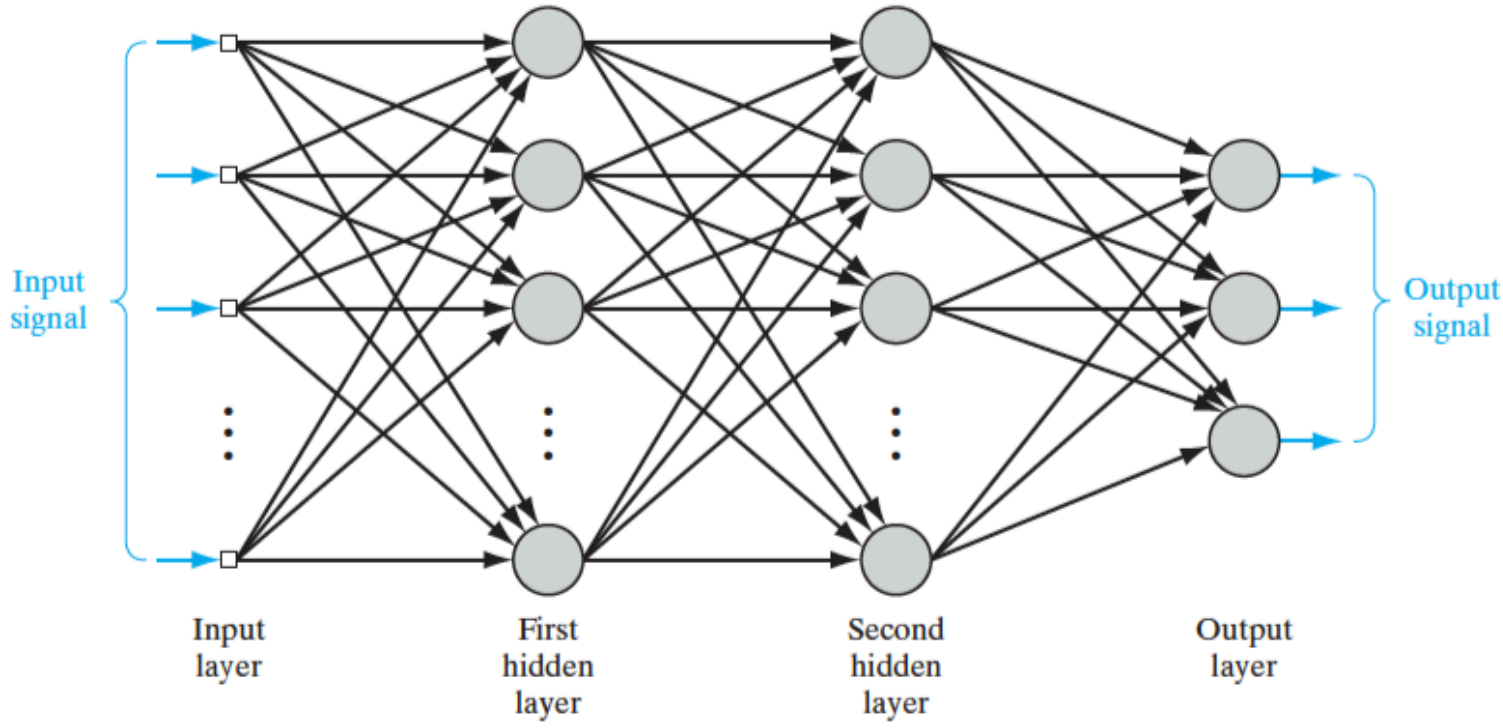
# Redes feedforward

- La estructura básica de una red feed-forward mono-capa es :
  - Suelen disponer de una o más capas ocultas de neuronas con función de salida sigmoidal seguidas por una capa final de neuronas lineales (o sigmoidales si la salida se desea en el rango (0,1)).
  - Permite *aprender relaciones no lineales y lineales* entre la entrada y la salida



# Red feedforward

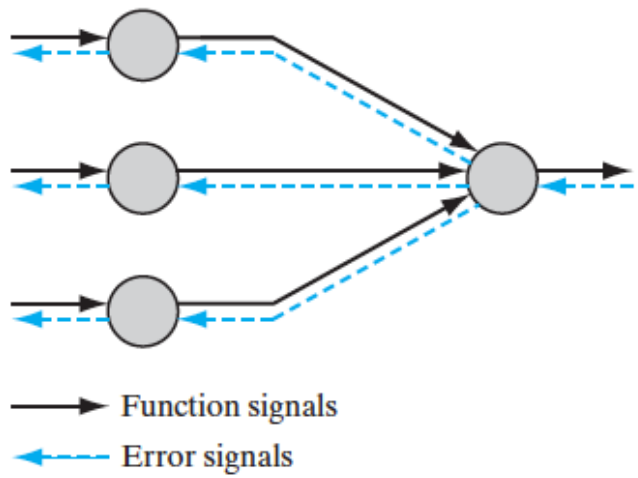
- Ejemplo de perceptrón multicapa con dos capas ocultas:



# Red feedforward

## Flujos de señal

- En el perceptrón multicapa existen dos flujos de señal:



# Red feedforward

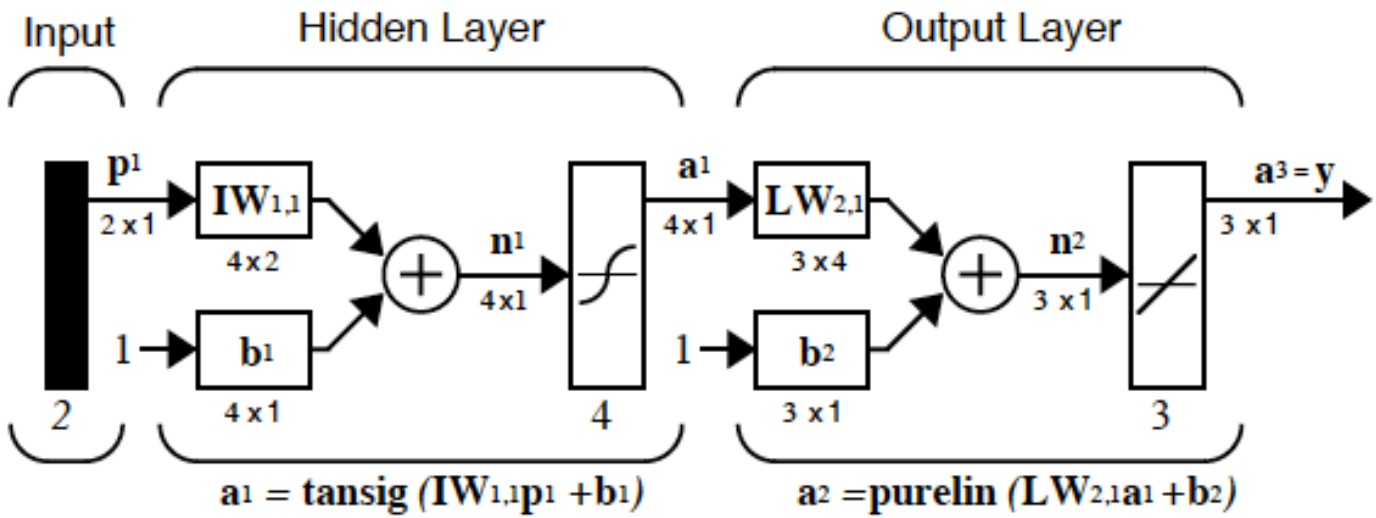
## Rol de las neuronas ocultas

- En el perceptrón multicapa las neuronas ocultas actúan como feature detectors; y juegan un papel crítico en el funcionamiento del mismo.
- A medida que el aprendizaje progresa las neuronas ocultas comienzan a "descubrir" gradualmente las salient features que caracterizan a los datos de entrenamiento.
- Esto se lleva a cabo realizando una transformación no-lineal de los datos de entrada a un nuevo espacio denominado espacio de características o feature space. En este nuevo espacio las clases de interés para la tarea de clasificación de patrones pueden ser más fácilmente separables unas de otras que los datos en el espacio de entrada original.

# Red feedforward

## Red feedforward de dos capas.

- Esta red feedforward de dos capas puede usarse como un aproximador general de funciones.
  - Aproxima bien cualquier función con un número finito de discontinuidades
  - Siempre que el número de neuronas en la capa oculta sea suficiente.



# Aprendizaje de la red

## Aprendizaje supervisado

Si definimos el conjunto de datos de entrenamiento

$$T = \{\mathbf{x}(n), \mathbf{d}(n)\}_{n=1}^{N}$$

donde $\mathbf{x}(n)$ son las entradas aplicadas a la red y $\mathbf{d}(n)$ son las salidas deseadas
Si denominamos $y_j(n)$ a la salida producida en la neurona $j$ en la capa de salida para la señal de entrada $\mathbf{x}(n)$, la señal de error en la salida de la neurona $j$ es
$e_j(n) = d_j(n) - y_j(n)$
La energía del error instantáneo de la salida $j$ es
$E_j(n) = \frac{1}{2} e_j^2(n)$

# Aprendizaje de la red

## Aprendizaje supervisado

La suma de todas las energías de error de todas las salidas nos da la energía total del error instantáneo de toda la red

$E(n) = \sum_{j \in C} E_j(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$

Y la energía del error promediada sobre el conjunto de entrenamiento es

$E_{av}(n) = \frac{1}{N} \sum_{n=1}^{N} E(n) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n)$

Estas energías dependen de los pesos sinápticos ajustables de la red.

# Aprendizaje de la red

## Aprendizaje supervisado

- El índice de prestaciones más usado para ajustar los pesos sinápticos es el *error medio cuadrático*, es decir la energía del error entre las salidas que da la red y las salidas deseadas:

$$e_{MSE} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} (d_j(n) - y_j(n))^2 \qquad (5)$$

- Se puede entrenar la red en base a este índice en dos formas básicas:
  - **Modo incremental**, se calcula el gradiente y los pesos de las conexiones se recalculan después de que cada entrada se aplique a la red.
  - **Modo batch**, todas las entradas se aplican a la red antes de que se calculen los pesos de las conexiones, este método es más rápido y da lugar a menos error.

# Aprendizaje de la red

## Modo Batch

- En este modo los ajustes de los pesos sinápticos de la red se realizan después de presentar todos los N ejemplos en el conjunto de muestras de entrenamiento $T$ que constituyen una época de entrenamiento.
- Es decir la función de coste a minimizar en el batch learning se define como la energía promedio del error $E_{av}$
- Los ajustes de los pesos sinápticos de la neuronas se hacen época-a-época . Los ejemplos en el conjunto de entrenamiento se barajan aleatoriamente antes de ejecutar cada época en la red.
- Ventajas e inconvenientes del batch learning:
  - Estimación precisa del vector gradiente (es decir, la derivada de la función de coste respecto al vector de pesos $w$) y por tanto garantiza la convergencia a un minimo local.
  - El proceso de aprendizaje puede ser paralizado.
  - Requiere mucha memoria.

# Aprendizaje de la red

## Modo incremental o on-line

- Se realiza ejemplo a ejemplo.
- La función de coste a minimizar es la energía instantánea del error $E(n)$.
- Se presenta a la red el primer par de ejemplo $\{\mathbf{x}(1), \mathbf{d}(1)\}$ en el ."epoch", y se hacen los ajustes de los pesos mediante el método del descenso de gradiente y sucesivamente se le presentan todos los pares de ejemplo.
- Esto impide la paralelización del aprendizaje pero tiene la ventaja de ser simple.

# Aprendizaje de la red

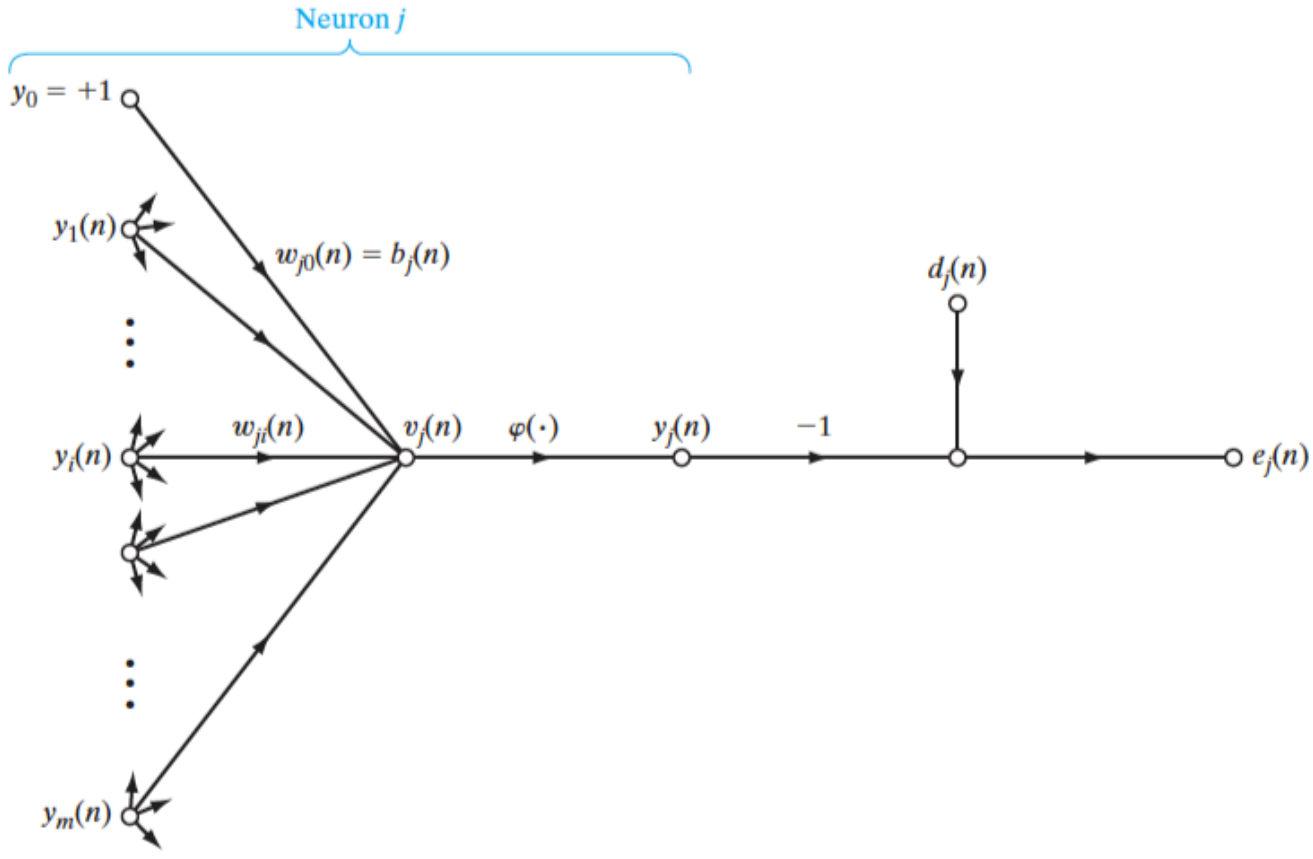
## Algoritmo backpropagation

- El entrenamiento usa el gradiente de las prestaciones de la red con respecto a los pesos de de las conexiones de la red para ajustar los pesos sinápticos.
- Se calcula mediante un algoritmo de aprendizaje denominado **Backpropagation**.

# Aprendizaje de la red

## Algoritmo backpropagation

- Veamos las señales que hay en las neurona $j$ de la capa de salida



# Aprendizaje de la red

## Algoritmo backpropagation

- El campo local inducido $v_j(n)$ producido en la entrada de la función de activación n asociada a la neurona $j$ es

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{6}$$

  donde: $m$ es el número de entradas(excluyendo el bias) aplicado a la neurona $j$.

- El peso sináptico $w_{j0}$ correspondiente a la entrada fija $+1$ es el bias $b_j$ aplicado a la neurona $j$.
- La señal de salida $y_j(n)$ de la neurona $j$ en la iteración $n$

$$y_j(n) = \phi_j(v_j(n)) \tag{7}$$

# Aprendizaje de la red

## Algoritmo backpropagation

- De forma similar al algoritmo LMS visto para el Adeline, el algoritmo de backpropagation aplica una correción $\Delta w_{ji}(n)$ a los pesos sinápticos $w_{ji}(n)$, la cual es proporcional a la derivada parcial $\frac{\partial E(n)}{\partial w_{ji}(n)}$.
- De acuerdo a la regla de la cadena el gradiente anterior se puede expresar en la forma:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{8}$$

- La $\frac{\partial E(n)}{\partial w_{ji}(n)}$ representa el factor de sensibilidad, que determina la dirección de búsqueda en espacio de pesos para las conexiones sinápticas $w_{ji}(n)$

# Aprendizaje de la red

## Algoritmo backpropagation

- Diferenciando en la expresión de $E(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n)$ con respecto a $e_j(n)$ tenemos

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \tag{9}$$

- Diferenciando en ambos lados de la expresaón de la señal de error $e_j(n) = d_j(n) - y_j(n)$ con respecto a $y_j(n)$ tenemos

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{10}$$

- Diferenciando en la expresión $y_j(n) = \phi_j(v_j(n))$ respecto a $v_j(n)$ tenemos

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \phi_j'(v_j(n)) \tag{11}$$

- Por último diferenciando $v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$ respecto a $w_{ji}(n)$ tenemos

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \tag{12}$$

# Aprendizaje de la red

## Algoritmo backpropagation

- Uniendo todas estas expresiones tenemos que

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \tag{13}$$

$$= e_j(n).(-1).\phi_j'(v_j(n)).y_i(n) \tag{14}$$

- Con lo que la corrección de los pesos sinápticos es

$$\Delta w_{ji}(n) = -\alpha \frac{\partial E(n)}{\partial w_{ji}(n)} \tag{15}$$

# Aprendizaje de la red

## Algoritmo backpropagation

- La corrección $\Delta w_{ji}(n)$ suele expresarse

$$\Delta w_{ji}(n) = -\alpha \frac{\partial E(n)}{\partial w_{ji}(n)} = \alpha \delta_j y_i(n) \tag{16}$$

donde al término $\delta_j$ se le denomina gradiente local y se define como

$$\begin{aligned} \delta_j &= \frac{\partial E(n)}{\partial v_j(n)} && (17) \\ &= \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} && (18) \\ &= e_j(n).(-1).\phi_j'(v_j(n)) && (19) \end{aligned}$$

el gradiente local apunta a los cambios requeridos en los pesos sinápticos y es el producto de la señal de error $e_j(n)$ de esa neurona por la derivada $\phi_j'(v_j(n))$ de la función de activación asociada.

- Observe que en el cálculo del ajuste de los pesos interviene la señal de error $e_j(n)$ en la salida de la neurona $j$. Este error es conocido en las neuronas de la capa de salida de la red, pero y en las intermedias como?

# Aprendizaje de la red

## Algoritmo backpropagation

- En el caso de las neuronas de las capas ocultas, la salida de la neurona $j$ está oculta.
- Aunque estas neuronas no sean visibles comparten la responsabilidad en el error que se comete en la salida de la red.
- La cuestión es como penalizar o recompensar a estas neuronas ocultas por su parte de responsabilidad en el error. Este problema se conoce como el problema de la asignación de crédito ( credit-assignment problem ).

# Aprendizaje de la red

## Algoritmo backpropagation

- Con lo visto, a la hora de calcular las correcciones de los pesos sinápticos tenemos que diferenciar dos casos
  - La neurona $j$ es de la capa de salida.
    En este caso se calcula el error $e_j(n)$ asociado a la neurona y calculando el gradiente local $\delta_j$ se determina la corrección de los pesos de la neurona $j$

$$\Delta w_{ji}(n) = \alpha \delta_j y_i(n) \quad (20)$$

$$\delta_j = e_j(n).(-1).\phi'_j(v_j(n)) \quad (21)$$

  - La neurona $j$ es de una capa oculta.
    Veamos como.

# Aprendizaje de la red

## Algoritmo backpropagation

- Si la neurona $j$ está en una capa oculta, no hay una señal de salida asignada a las neuronas de esta capa y la señal de error debe de ser calculada recursivamente hacia atrás en función de las señales de error de todas las neuronas a las cuales la neurona oculta está conectada.
- De acuerdo con la expresión vista para el gradiente local, podemos redefinirlo para una neurona $j$ oculta como

$$\delta_j = -\frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \tag{22}$$

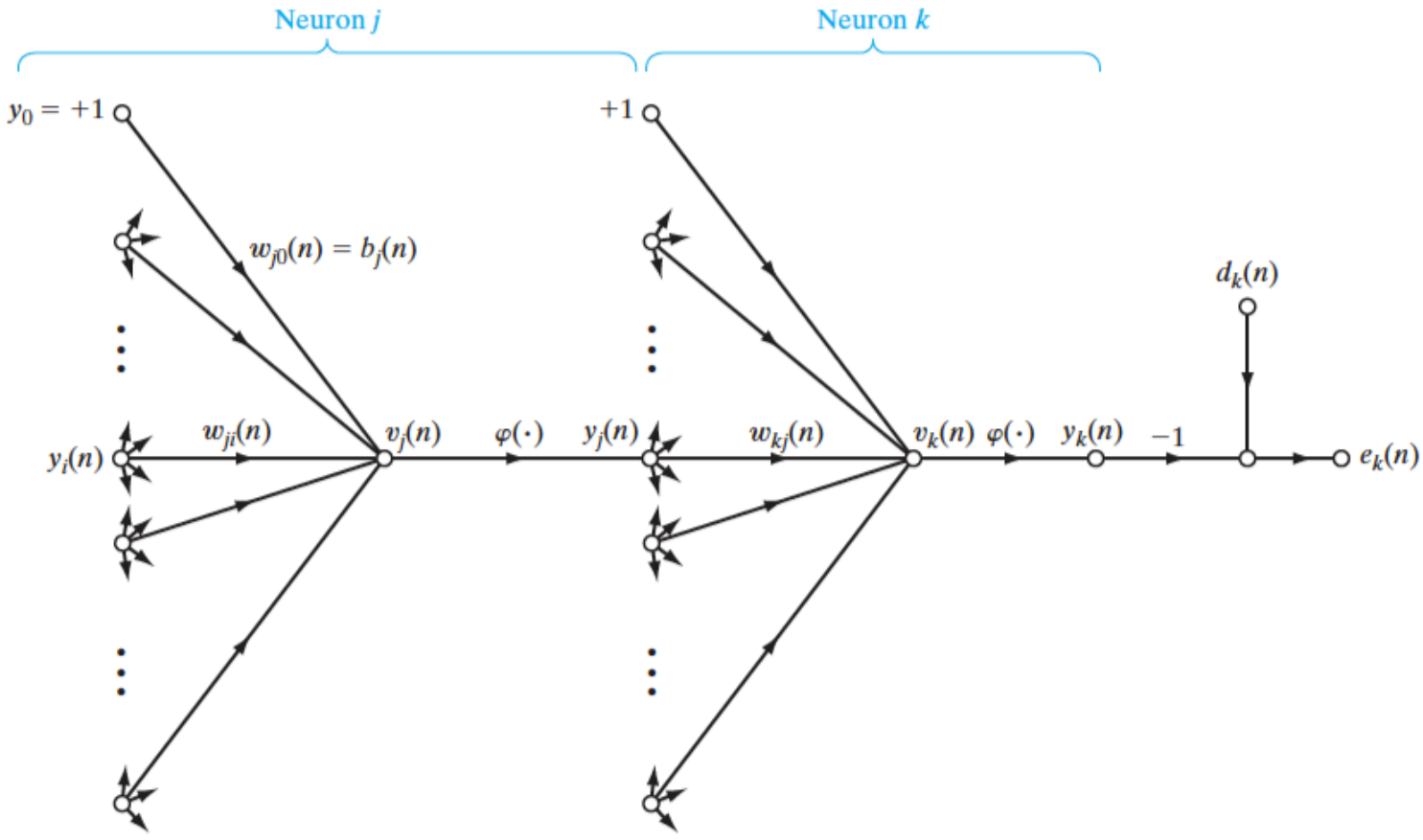
$$= -\frac{\partial E(n)}{\partial y_j(n)} \phi_j'(v_j(n)) \tag{23}$$

donde hemos usado que $\frac{\partial y_j(n)}{\partial v_j(n)} = \phi_j'(v_j(n))$.

# Aprendizaje de la red

## Algoritmo backpropagation

- Veamos las señales que hay en la neurona $k$ conectada a la neurona $j$ de la capa oculta



# Aprendizaje de la red

## Algoritmo backpropagation

- El problema es como calcular $\frac{\partial E(n)}{\partial y_j(n)}$.
- Por otra parte sabemos que para la neurona $k$

$$E(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n) \tag{24}$$

diferenciando con respecto a la señal $y_j(n)$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \tag{25}$$

# Aprendizaje de la red

## Algoritmo backpropagation

- Si ahora usamos la regla de la cadena para la derivada parcial $\frac{\partial e_k(n)}{\partial y_j(n)}$ y reescribimos la expresión nos queda lo siguiente

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \tag{26}$$

y como si observamos en la figura de la transparencia anterior tenemos que

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \phi_k(v_k(n))$$

siendo la neurona $k$ un nodo de salida.

- De aquí

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\phi_k'(v_k(n)) \tag{27}$$

# Aprendizaje de la red

## Algoritmo backpropagation

- Por otra parte en la figura podemos ver que

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n) \tag{28}$$

  donde $m$ es el número total de entradas (excluido el bias) aplicado a la neurona $k$. Aquí de nuevo el peso $w_{k0}$ es igual al bias $b_k(n)$ aplicado a la neurona $k$ y la entrada correspondiente se fija a $+1$.

- Diferenciando aquí con respecto a $y_j(n)$ obtenemos

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \tag{29}$$

# Aprendizaje de la red

## Algoritmo backpropagation

- Usando que $\frac{\partial e_k(n)}{\partial v_k(n)} = -\phi'_k(v_k(n))$ y que $\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$ obtenemos la derivada parcial $\frac{\partial E(n)}{\partial y_j(n)}$ que resulta

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k e_k(n)\phi'_k(v_k(n))w_{kj}(n) \quad (30)$$

$$= -\sum_k \delta_k(n)w_{kj}(n) \quad (31)$$

siendo $\delta_k(n)$ el gradiente local (hemos substituido $j$ por $k$).

- Finalmente usando esta expresión en la del $\delta_j$ obtenemos

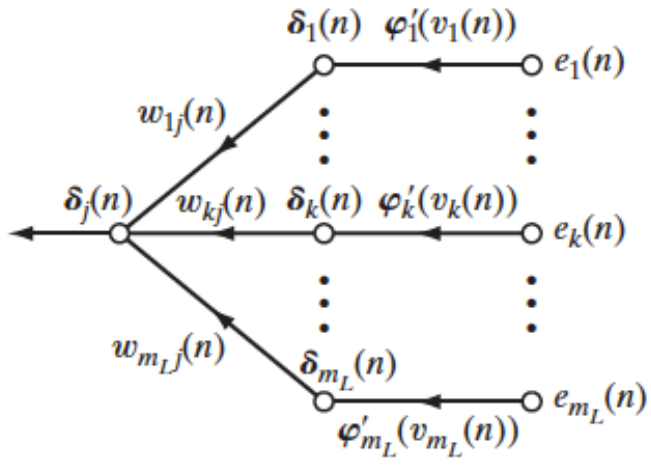
$$\delta_j = -\frac{\partial E(n)}{\partial y_j(n)}\phi'_j(v_j(n)) \quad (32)$$

$$= \phi'_j(v_j(n))\sum_k \delta_k(n)w_{kj}(n) \quad (33)$$

donde $j$ es la neurona oculta.

# Aprendizaje de la red

## Algoritmo backpropagation




- Flujo de las señales en la retropropagación del error.

# Aprendizaje de la red

## Algoritmo backpropagation

- El factor $\phi_j'(v_j(n))$ implicado en el calculo del gradiente local $\delta_j(n)$ en la expresión $\delta_j = \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$ depende únicamente de la función de activación asociada a la neurona oculta $j$.
- El factor restante, la suma sobre los $k$, depende de dos conjuntos de términos:
  - El primer conjunto son los $\delta_k(n)$ que requieren conocer el error $e_k(n)$ para todas las neuronas de la capa siguiente a la oculta que están conectadas a la neurona $j$.
  - El segundo conjunto son los $w_{kj}(n)$ que son los pesos sinápticos asociados a esas conexiones.

# Aprendizaje de la red

## Algoritmo backpropagation: resumen

- La denominada regla delta para la corrección de los pesos de las conexiones que conectan la neurona $i$ a la $j$ viene dada por

$$\Delta w_{ji}(n) = \alpha.\delta_j(n).y_i(n) \tag{34}$$

- Dependiendo de si la neurona $j$ es un nodo oculto o no:
  - Si la neurona $j$ es salida, $\delta_j(n)$ es igual al producto de la derivada $\phi'_j(v_j(n))$ por la señal de error $e_j(n)$, ambos eatán asociados con la neurona $j$;

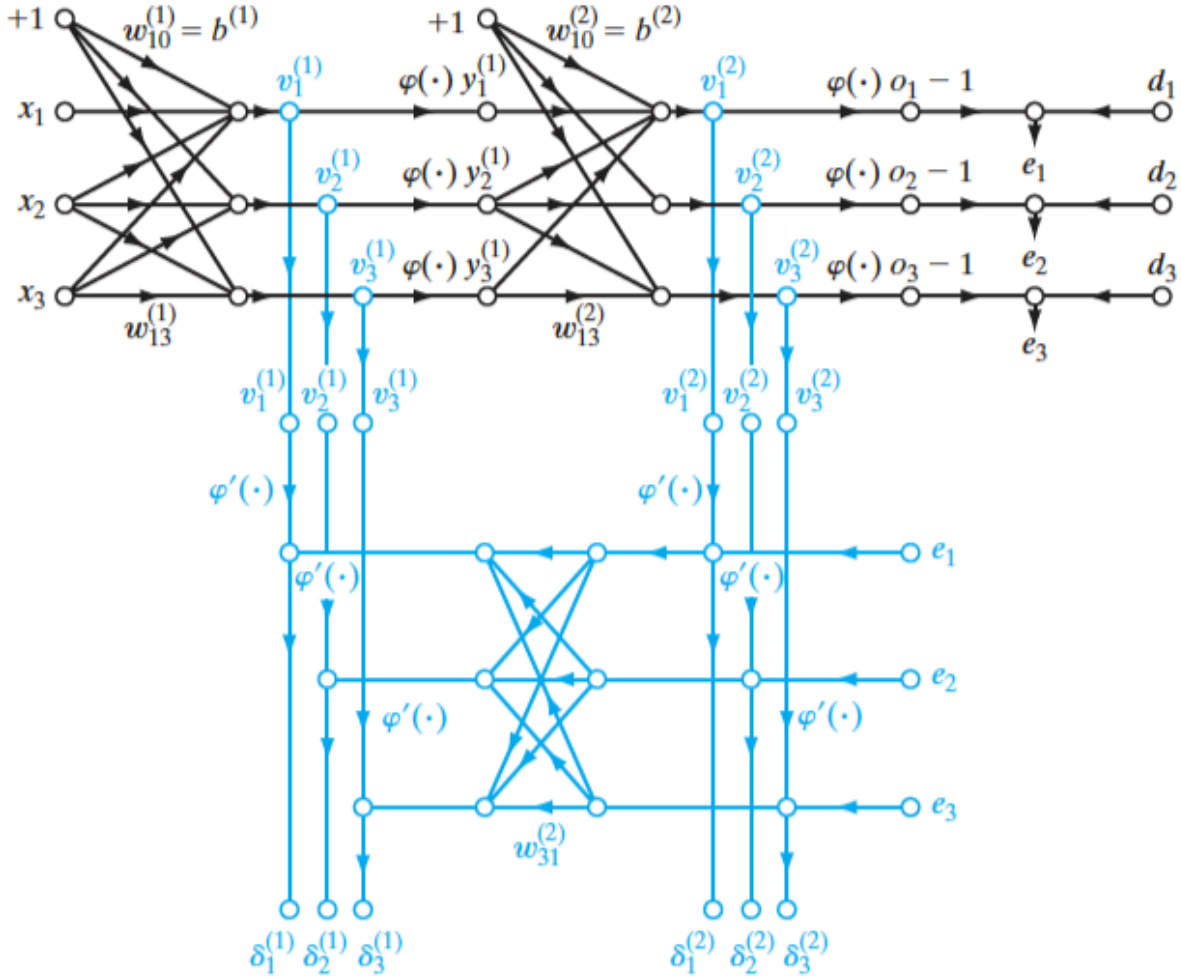
$$\delta_j \quad = \quad \phi'_j(v_j(n))e_j(n) \tag{35}$$

  - Si la neurona $j$ es oculta, $\delta_j(n)$ es igual al producto de las derivadas asociadas $\phi'_j(v_j(n))$ por la suma ponderada dos pesos $\delta$s calculados para las neuronas en la capa oculta o de salida situada inmediatamente posterior que estén conectadas a la neurona $j$.

$$\delta_j \quad = \quad \phi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{36}$$

# Aprendizaje de la red

## Algoritmo backpropagation




- Flujo de las señales en el algoritmo back propagación, arriba en negro el flujo hacia adelante y abajo el flujo en la retropropagación del error.

# Algoritmo backpropagation

## Funciones de activación

- El cálculo del gradiente local $\delta$ para cada neurona requiere conocer la derivada de la función de activación $\phi$ asociada con esa neurona. Por esto las funciones de activación deben de ser diferenciables.
- Función logística

$$\phi_j(v_j(n)) = \frac{1}{1 + exp(-av_j(n))}, \quad a > 0 \tag{37}$$

derivando

$$\phi'_j(v_j(n)) = \frac{a\ exp(-av_j(n))}{[1 + exp(-av_j(n))]^2} \tag{38}$$

como $y_j(n) = \phi_j(v_j(n))$ podemos eliminar el término $exp(-av_j(n))$ con lo que

$$\phi'_j(v_j(n)) = ay_j(n)[1 - y_j(n)] \tag{39}$$

# Algoritmo backpropagation

## Funciones de activación

- Con lo que para una neurona $j$ de la capa de salida el gradiente local queda como

$$\begin{aligned} \delta_j(n) &= e_j(n)\phi_j'(v_j(n)) && (40) \\ &= a[d_j(n) - o_j(n)]o_j(n)[1 - o_j(n)] && (41) \end{aligned}$$

donde $o_j(n)$ es la salida de la neurona $j$ de la capa de salida y $d_j$ es la salida deseada en dicha neurona, y para una neurona $j$ de una neurona de una capa oculta

$$\begin{aligned} \delta_j(n) &= \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) && (42) \\ &= a y_j(n)[a - y_j(n)] \sum_k \delta_k(n) w_{kj}(n) && (43) \end{aligned}$$

# Aprendizaje de la red

- Función tangente hiperbólica.

$$\phi_j(v_j(n)) = a\tanh(bv_j(n)), \tag{44}$$

donde $a, b$ son constantes positivas. derivando

$$\begin{aligned} \phi_j'(v_j(n)) &= ab(1-\tanh^2(bv_j(n))) && (45)\\ &= \frac{b}{a}[a-y_j(n)][a+y_j(n)] && (46)\end{aligned}$$

- Con lo que para una neurona $j$ de la capa de salida el gradiente local queda como

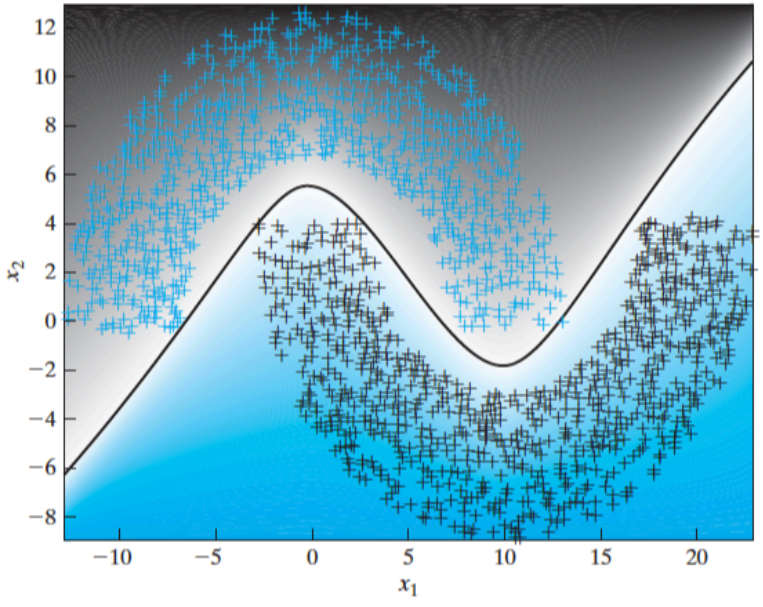
$$\begin{aligned} \delta_j(n) &= e_j(n)\phi_j'(v_j(n)) && (47)\\ &= \frac{b}{a}[d_j(n)-o_j(n)][a-o_j(n)][a+o_j(n)] && (48)\end{aligned}$$

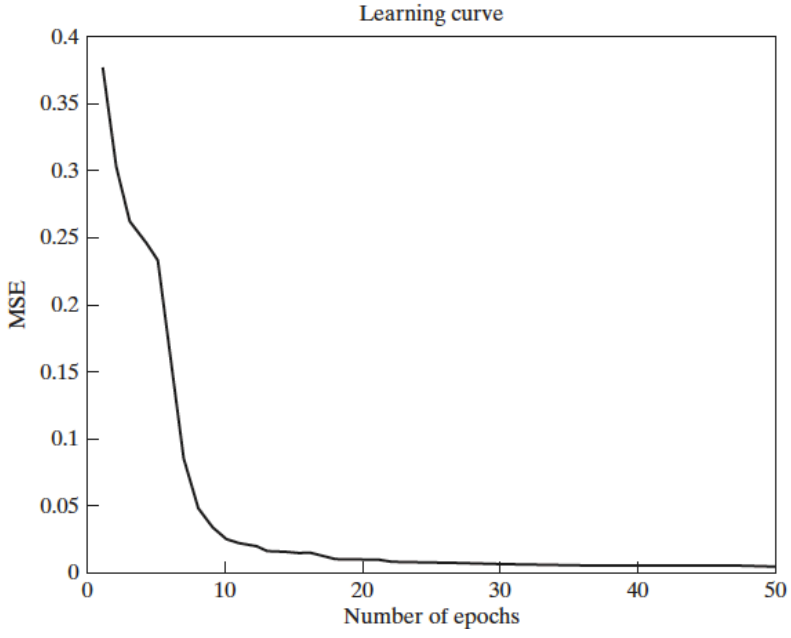
y para una neurona $j$ de una neurona de una capa oculta

$$\begin{aligned} \delta_j(n) &= \phi_j'(v_j(n))\sum_k \delta_k(n)w_{kj}(n) && (49)\\ &= \frac{b}{a}[a-y_j(n)][a+y_j(n)]\sum_k \delta_k(n)w_{kj}(n) && (50)\end{aligned}$$

# Algoritmo backpropagation

## Ejemplo




- Capas: entrada $m_0 = 2$, capa oculta $m_1 = 20$, capa de salida $m_2 = 1$, función de activación hiperbólica, learning rate $\alpha = 10^{-1}$.

# Algoritmo de Backpropagation

## Variantes

- Existen diferentes variantes del algoritmo de aprendizaje backpropagation:

| Function | Algorithm |
|---|---|
| `trainlm` | Levenberg-Marquardt |
| `trainbr` | Bayesian Regularization |
| `trainbfg` | BFGS Quasi-Newton |
| `trainrp` | Resilient Backpropagation |
| `trainscg` | Scaled Conjugate Gradient |
| `traincgb` | Conjugate Gradient with Powell/Beale Restarts |
| `traincgf` | Fletcher-Powell Conjugate Gradient |
| `traincgp` | Polak-Ribiére Conjugate Gradient |
| `trainoss` | One Step Secant |
| `traingdx` | Variable Learning Rate Gradient Descent |

# Método de aprendizaje

## Algoritmo de Backpropagation

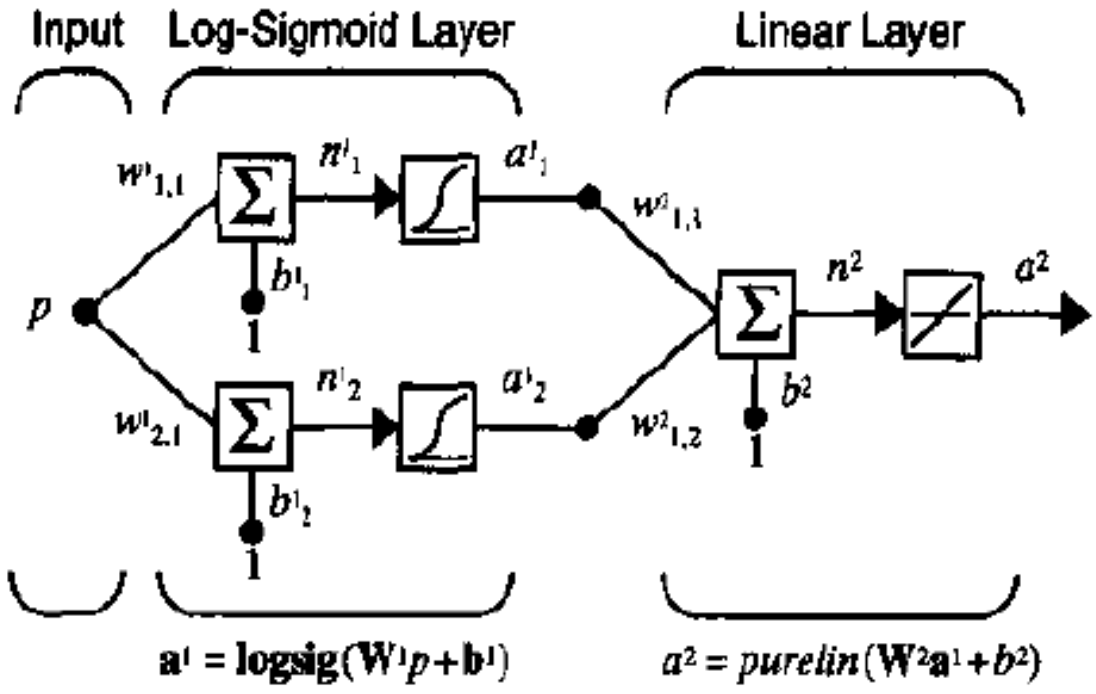
- Variantes del algoritmo de aprendizaje backpropagation (cont):

| | |
|---|---|
| `traingdm` | Gradient Descent with Momentum |
| `traingd` | Gradient Descent |

# Ejemplo de funcionamiento

- Red simple 1-2-1 (Entradas-C1-Cs).
  - Queremos aproximar la función:

$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right), \quad \text{para} \quad -2 \leqslant p \leqslant 2 \tag{51}$$

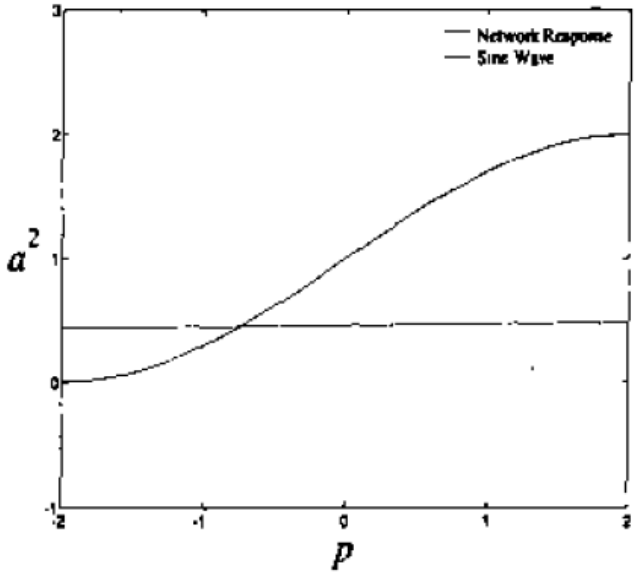


# Ejemplo de funcionamiento

- Operación
  - Para comenzar el alg. backpropagation es necesario inicializar los pesos y bias de la red (se suelen tomar valores aleatorios pequeños). Supongamos los siguientes

$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix}, \mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix}, \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix}, \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$

  - Respuesta de la red para estos valores iniciales



# Ejemplo de funcionamiento

- Algoritmo:
  - Entrada inicial $p = 1, \rightarrow \ a^0 = p = 1$ salida de la primera capa de la red para esta entrada

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) = \mathbf{logsig}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}[1] + \begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right) = \mathbf{logsig}\left(\begin{bmatrix}-0.75\\-0.54\end{bmatrix}\right)$$

$$= \begin{bmatrix}\dfrac{1}{1+e^{0.75}}\\[2ex]\dfrac{1}{1+e^{0.54}}\end{bmatrix} = \begin{bmatrix}0.321\\0.368\end{bmatrix}$$

  - salida de la segunda capa de la red

$$a^2 = f^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) = purelin\left(\begin{bmatrix}0.09 & -0.17\end{bmatrix}\begin{bmatrix}0.321\\0.368\end{bmatrix} + \begin{bmatrix}0.48\end{bmatrix}\right) = \begin{bmatrix}0.446\end{bmatrix}$$

# Ejemplo de funcionamiento

- cont.
  - el error de salida es

$$e = t - a = \{1 + \sin\left(\frac{\pi}{4}p\right)\} - a^2 = \{1 + \sin\left(\frac{\pi}{4}1\right)\} - 0.446 = 1.261$$

  - Retropropagación (backpropagate) del error, necesitamos calcular en primer lugar las derivadas de la función de transferencia $f'^1(n)$ y $f'^2(n)$.
  - Para la primera capa

$$f^1(n) = \frac{d}{dn}\left(\frac{1}{1+e^{-n}}\right) = \frac{e^{-n}}{(1+e^{-n})^2} = \left(1 - \frac{1}{1+e^{-n}}\right)\left(\frac{1}{1+e^{-n}}\right) = (1-a^1)(a^1)$$

# Ejemplo de funcionamiento

- cont.
  - y para la segunda

$$f^2(n) = \frac{d}{dn}(n) = 1$$

  - Ahora se puede comenzar con la retropropagación, empezando por la segunda capa tenemos que la sensibilidad es

$$\mathbf{s}^2 = -2\dot{\mathbf{F}}^2(\mathbf{n}^2)(\mathbf{t}-\mathbf{a}) = -2\left[f^2(n^2)\right](1.261) = -2\left[1\right](1.261) = -2.522$$

# Ejemplo de funcionamiento

- cont.
  - Para la primera capa usamos la sensibilidad de la segunda

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix} (1-a_1^1)(a_1^1) & 0 \\ 0 & (1-a_2^1)(a_2^1) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} \begin{bmatrix} -2.522 \end{bmatrix}$$

$$= \begin{bmatrix} (1-0.321)(0.321) & 0 \\ 0 & (1-0.368)(0.368) \end{bmatrix} \begin{bmatrix} 0.09 \\ -0.17 \end{bmatrix} \begin{bmatrix} -2.522 \end{bmatrix}$$

$$= \begin{bmatrix} 0.218 & 0 \\ 0 & 0.233 \end{bmatrix} \begin{bmatrix} -0.227 \\ 0.429 \end{bmatrix} = \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix}.$$

  - La parte final del algoritmo es el cálculo de los nuevos pesos, para lo que usaremos un $\alpha = 0{,}1$.

# Ejemplo de funcionamiento

- cont.
  - Calculamos los nuevos pesos, a partir de las expresiones y un $\alpha = 0{,}1$.

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha \mathbf{s}^2 (\mathbf{a}^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} \begin{bmatrix} 0.321 & 0.368 \end{bmatrix} = \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix}$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha \mathbf{s}^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix},$$

$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha \mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}.$$

  - Esto completa la primera iteración del algoritmo backpropagation.
  - A continuación se introduce una nueva entrada *p* y se realiza una nueva iteración del algoritmo.
  - Se continua hasta que la diferencia entre la salida deseada y la red este por debajo de un cierto valor.

# Configuración de la arquitectura en las redes feedforward

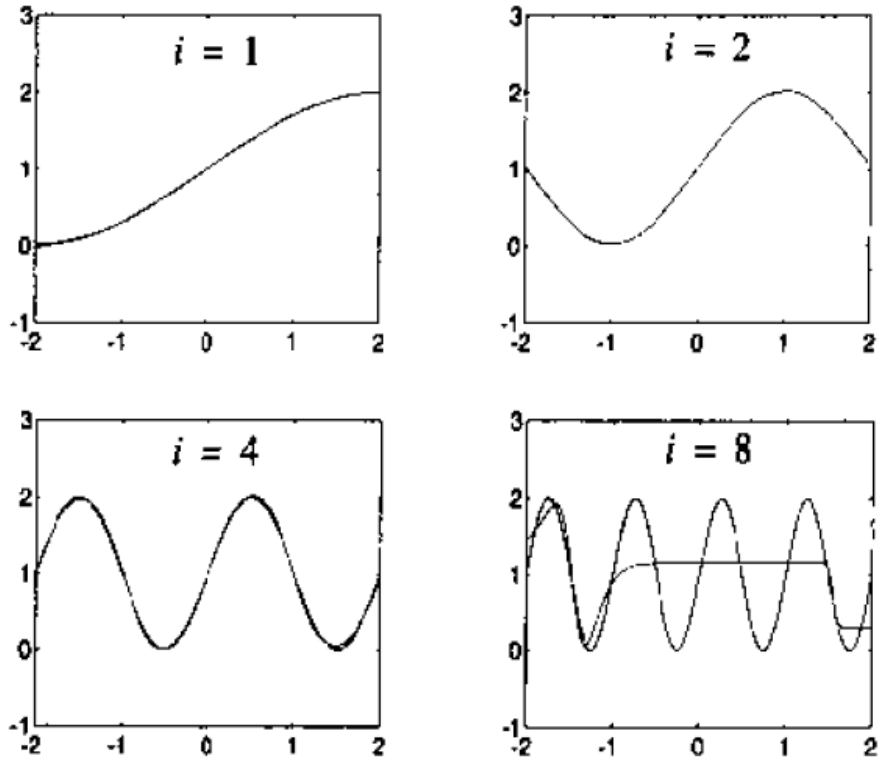
- Las redes feedforward son aproximadores generales de funciones.
  - Siempre y cuando que tengamos suficientes neuronas en las capas ocultas.
  - Como determinar el número de capas ocultas y el número de neuronas en cada capa?.
  - Veamos un ejemplo:
    - Queremos aproximar las siguientes funciones para $i = 1, 2, 4 y 8$.

$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right), \quad \text{para} \quad -2 \leqslant p \leqslant 2 \tag{52}$$

    - Al aumentar la función se hace más compleja y para el intervalo de $p$ considerado hay más ciclos.

# Configuración de la arquitectura en las redes feedforward

- Inicialmente usamos una arquitectura de red $1-3-1$ (1 entrada, 3 neuronas en la capa oculta y 1 en la de salida), es decir es una red de dos capas
- Para $i = 1, 2, 4, 8$ después de entrenar la red se obtienen las siguientes salidas



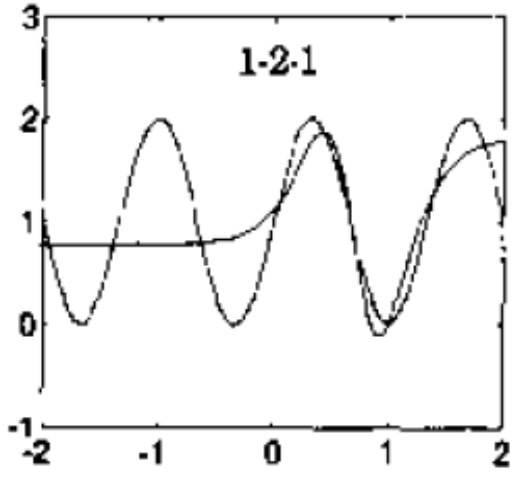
# Configuración de la arquitectura en las redes feedforward

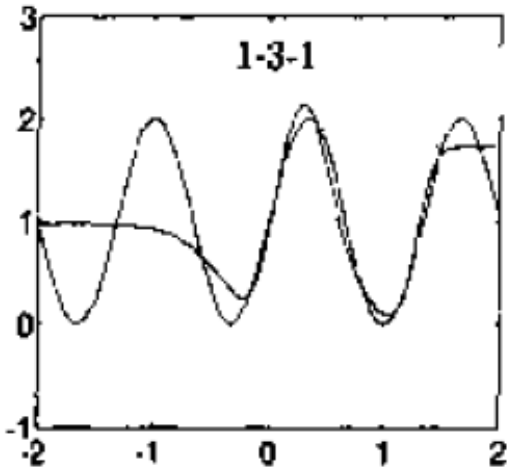
- La red $1-3-1$ para $i=8$ falla ya que no consigue ajustar la forma de la función por falta de neuronas (obsérvese que consigue aproximar una parte).
- Repitamos el proceso pero ahora para una función fija y con redes de complejidad variable

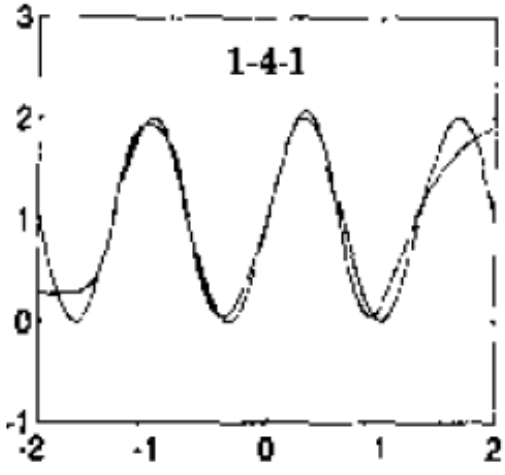
$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right), \quad \text{para} \quad -2 \leqslant p \leqslant 2 \tag{53}$$

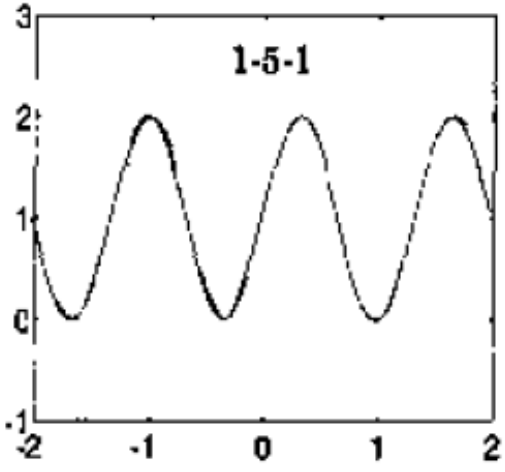
- Usaremos como en el caso anterior redes de dos capas $1-S^1-1$ con neuronas con activación log-sigmoide en la oculta y lineales en la de salida.
- Veamos que ocurre a medida que aumentamos el número de neuronas en la capa oculta de la red.

# Configuración de la arquitectura en las redes feedforward



# Configuración de la arquitectura en las redes feedforward

Conclusión:

- Es necesario ajustar la arquitectura de la red a la complejidad del problema.
- No es un enfoque razonable el sobredimensionar a priori, aparecen otros fenomenos indeseables:
  - Sobreajuste (overfitting)
  - Lentitud en el aprendizaje.

# Capacidad de generalización

- En la mayoría de los casos se entrena la red con un número finito de casos correctos.

$$(\mathbf{p}_1, \mathbf{t}_1), (\mathbf{p}_2, \mathbf{t}_2), \ldots, (\mathbf{p}_Q, \mathbf{t}_Q) \tag{54}$$

  - Normalmente este conjunto de entrenamiento es representativo de una clase mucho mayor de posibles pares entrada/salida.
  - Es importante que la red generalice lo que ha aprendido a toda la población.
  - Supongamos que el conjunto de entrenamiento lo obtenemos muestreando

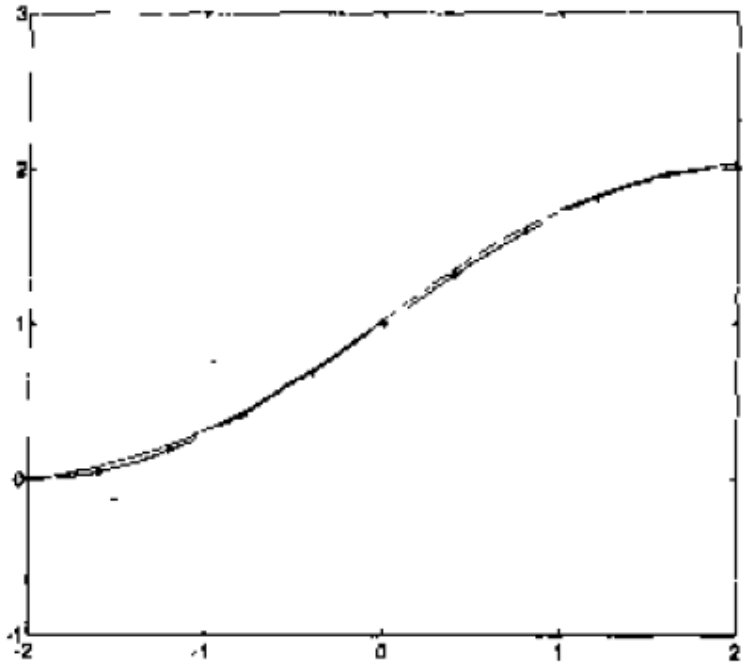
$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right), \quad \text{para} \quad -2 \leqslant p \leqslant 2 \tag{55}$$

    en los puntos

$$p = -2, -1{,}6, -1{,}2, \ldots, 1{,}6, 2$$

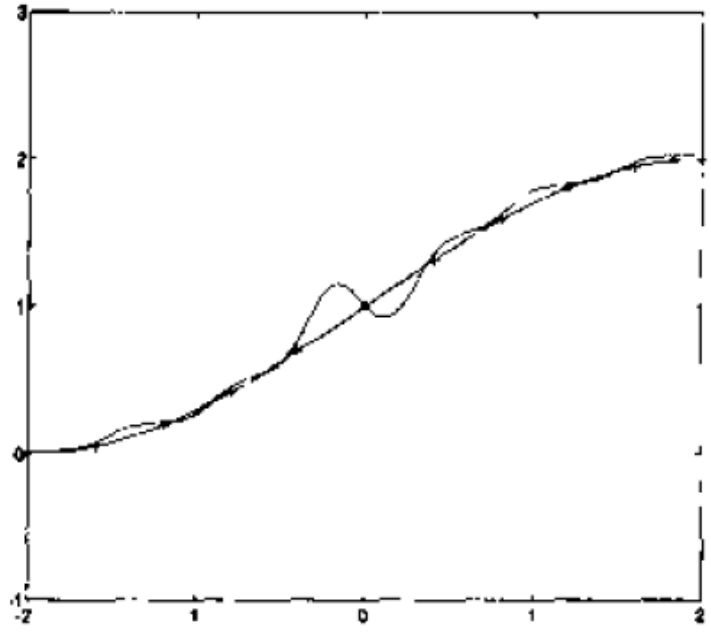
# Capacidad de generalización

- Si entrenamos una red $1-2-1$ como las anteriores con los 11 pares de entrenamiento obtenidos, la respuesta es bastante correcta.
- Incluso en los puntos intermedios no usados en el entrenamiento, es decir generaliza bien.
- Esta red tiene 7 parámetros a ajustar, 4 pesos y 3 bias y 11 ejemplos de entrenamiento.



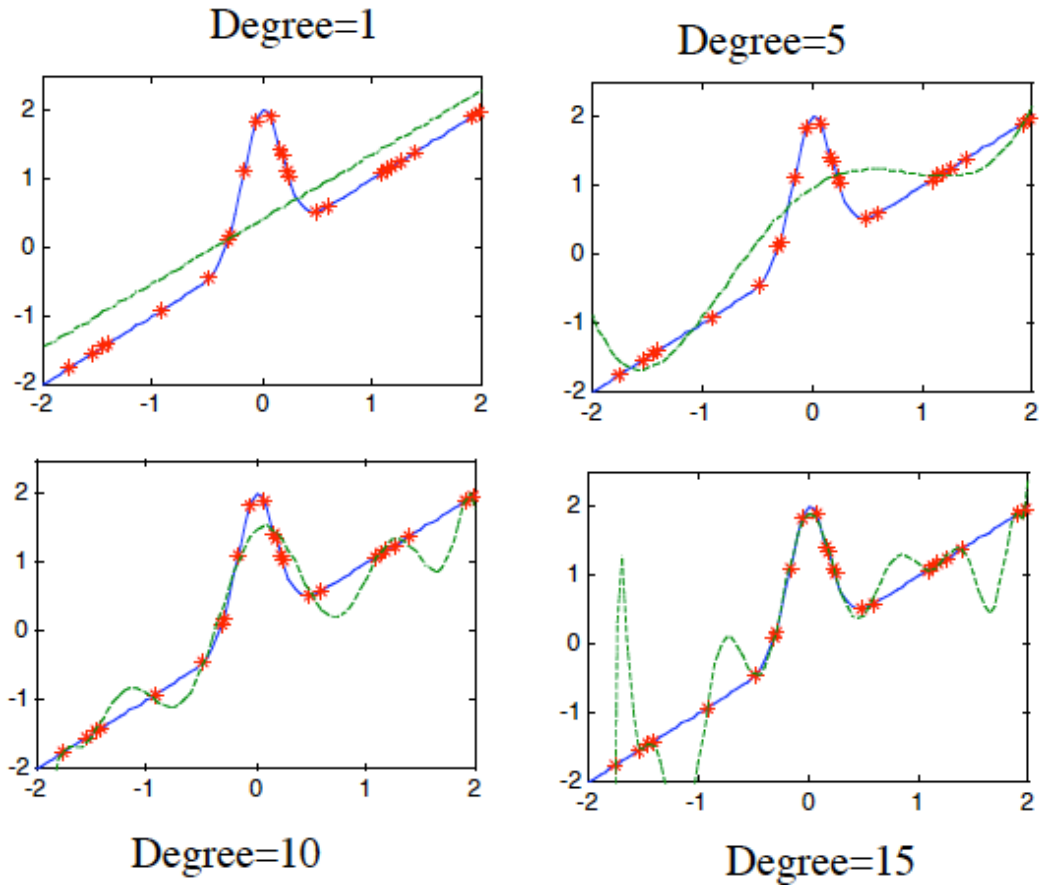
# Capacidad de generalización

- Si entrenamos una red 1-9-1 como las anteriores con los mismos 11 pares de entrenamiento obtenidos, la respuesta es correcta para los ejemplos de entrenamiento.
- Pero en los puntos intermedios no usados en el entrenamiento generaliza mal.
- Tiene demasiados parámetros a ajustar, en total 28 (18 pesos y 10 bias), y sólo 11 ejemplos de entrenamiento.
- Para que una red generalice bien debe tener menos parámetros que ejemplos en el conjunto de entrenamiento



# Overfitting (sobreajuste)

- Este problema aparece cuando ajustamos modelos teóricos a datos experimentale, es muy anterior a la redes neuronales (aparece inicialmente en ajustes de mínimos cuadrados a polinomios)



# Radial basis networks

## Redes con funciones de base radial

- Este tipo de red requiere muchas más neuronas que las redes feedforward con backpropagation, pero son mucho más fáciles de configurar.
- Existen dos grupos de redes este tipo de estructura:
  - **Redes neuronales de regresión generalizadas** (Generalized regression neural networks) GRNN.
  - **Redes neuronales probabilísticas** (Probabilistic neural networks) PNN

# Funciones de base radial

- Modelo de neurona
  - Aquí la entrada a la f. de transferencia de la red cambia sustancialmente
  - En vez de usarse la suma de los elementos del vector de entradas ponderados por los pesos
  - Se utiliza el vector de distancias entre el vector $p$ de entradas y el vector $w$ de pesos ( $\|\mathbf{dist}\|$ ) por el bias $b$

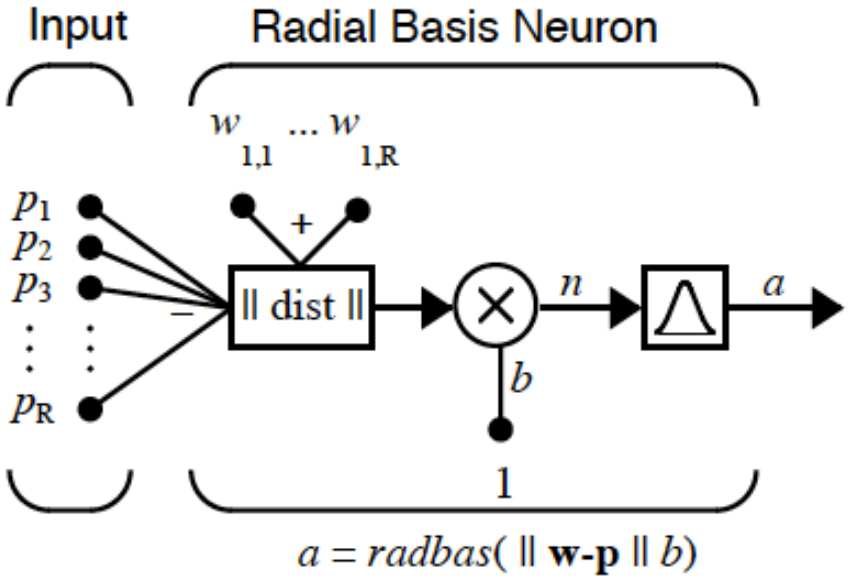
$$\|\mathbf{p} - \mathbf{w}\|.b \tag{56}$$

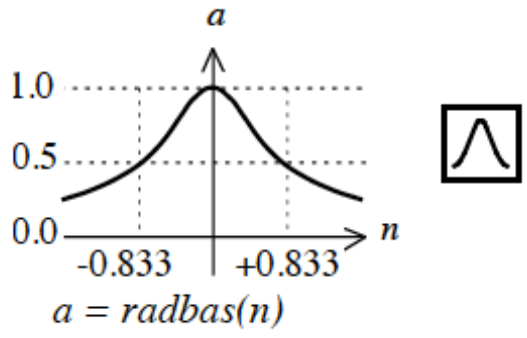
  - La función de salida es en este caso

$$radbas(n) = e^{-n^2} \tag{57}$$

# Funciones de base radial

- Modelo de neurona



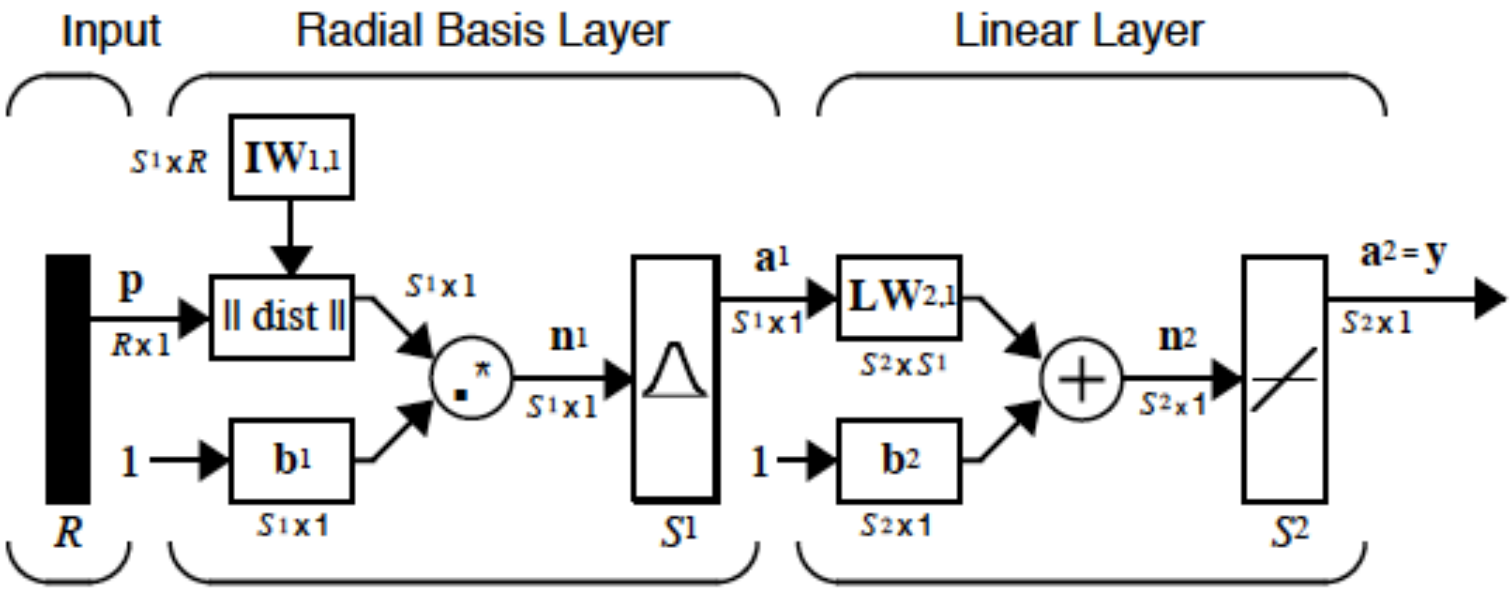



# Funciones de base radial

- Las funciones de base radial dan un valor máximo de 1 cuando la entrada a la función de activación es cero.
- Es decir a medida que la distanca entre **p** y **w** decrece la salida aumenta.
- Por tanto cada función de base radial actúa como un detector que da salida uno cuando la entrada **p** es idéntica a su vector de pesos **w**.
- El término de bias $b$ permite ajustar la sensibilidad de la neurona de base radial.
  - Ejemplo:
    Si la neurona tiene bias 0,1 dará una salida 0,5 para cualquier vector de entrada **p** que este a distancia 8,326(0,8326/$b$) de su vector de pesos **w**.

# Arquitectura de la red

- Constan de dos capas:
  - Capa oculta de neuronas de base radial $S^1$ .
  - Capa de salida lineal de $S^2$ neuronas.



# Arquitectura de la red

- Donde:
  - $R$ es el número de elementos en el vector de entrada
  - $S^1$ es el número de neuronas de base radial en la capa 1 (oculta)
  - $S^2$ es el número de neuronas en la capa 2 (de salida) $a_i,1$ es el i-ésimo elemento de $\mathbf{a}^1$ donde ${}_i\mathbf{IW}^{1,1}$ es un vector formado de la i-ésima fila de $\mathbf{IW}^{1,1}$

$$a_i,1 = radbas(\|{}_i\mathbf{IW}^{1,1} - \mathbf{p}\|b_i,1)$$
$$\mathbf{a}^2 = purelin(\mathbf{LW}^{2,1}\mathbf{a}^1 + \mathbf{b}^2) \quad (58)$$

# Idea de funcionamiento

- Si le presentamos un vector de entrada a la red, cada neurona de la capa de entrada de base radial dará una salida de acuerdo a la proximidad de la entrada al vector de pesos de la neurona.
- Las neuronas de base radial con pesos muy diferentes de la entrada darán valores próximos a cero con un efecto prácticamente despreciable sobre las neuronas de salida lineales.
- Las neuronas de base radial con salidas más altas son las que más influirán en las neuronas de la capa de salida.

# Diseño exacto

- Es posible diseñar una red con funciones de base radial que produzca error cero sobre un conjunto de vectores de entrenamiento.
- En Matlab esto se hace con la orden

$$net = newrbe(P, T, SPREAD) \tag{59}$$

- Esta función crea tantas neuronas de base radial como vectores de entrada hay en el conjunto de entrenamiento $P$ y fija los pesos de las neuronas de la capa de entrada a los $P_i$.
- Cada neurona actua como un detector del correspondiente vector de entrenamiento, si hay $Q$ vectores de entrada habra $Q$ neuronas.
- Los bias de la capa oculta de entrada se fijan todos al valor SPREAD, esto determina el área de influencia de la neurona de base radial, de forma que las neuronas dan valor 0,5 a una entrada situada a +/- SPREAD del vector de pesos.
- La capa lineal de salida se diseña para minimizar la suma del cuadrado de los errores.

$$Wb = T/[P; ones(1, Q)] \tag{60}$$

# Diseño exacto: problemas

- Un problema con $C$ restricciones (pares entrada/salida) y con $C + 1$ pesos ($C$ pesos + 1 bias) en cada neurona tiene infinitas soluciones con error cero.
- Se puede añadir alguna restricción adicional para forzar que las neuronas se solapen y den un funcionamiento global más suave.
- En general esta red produce una red con demasiadas neuronas en la capa oculta de base radial y si el número de vectores de entrada es muy grande no da una solución aceptable (lo que es muy frecuente).

# Mejora

- Un diseño más eficiente, crea una iterativamente las neuronas de base radial en la capa oculta de una en una.
- Las neuronas se añaden a la red mientras que la suma de errores cuadráticos cae por debajo de un cierto nivel de error o se alcanza un número máximo de neuronas.

$$net = newrbe(P, T, GOAL, SPREAD) \tag{61}$$

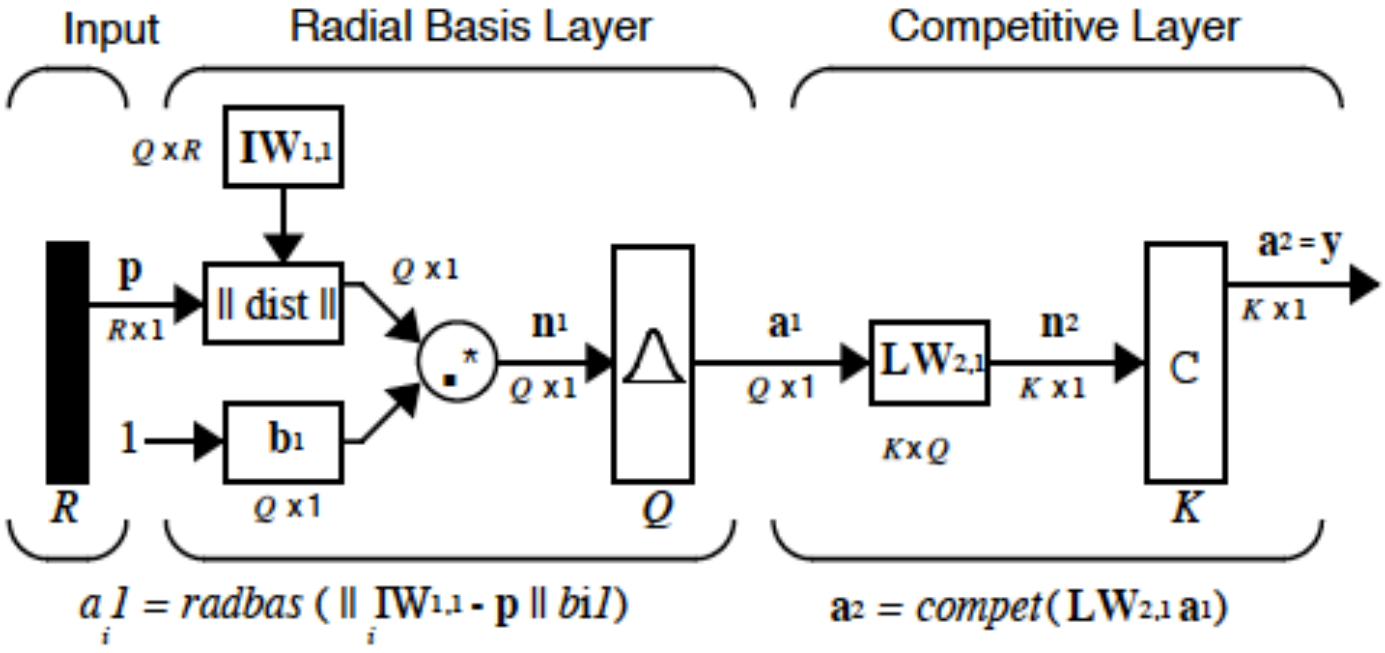
- En cada iteración este comando crea una neurona, y disminuye el error, este error se comprueba y cuando es suficientemente bajo *newrb* se finaliza.
- En general son redes mucho mayores que las feedforward, incluso la mejorada, aunque como ventaja tienen su rapidez de entrenamiento.
- En ocasiones pueden dar lugar a redes más simples que las feedforward.

# Redes neuronales probabilísticas PNN

- Se usan en problemas de clasificación.
- Su arquitectura consta de:
  - Una primera capa en la que se calculan las distancias del vector de entrada a los vectores de entrenamiento y se produce un vector cuyos elementos indican lo cerca que esta la entrada del vector de entrenamiento.
  - La segunda capa suma estas contribuciones para cada clase de entrada y da a la salida de la red un vector de probabilidades.
  - Finalmente una función de transferencia competitiva en la salida elije la mayor de estas probabilidades, produciéndo un 1 para esta clase y un 0 para todas las demas clases.

# Redes neuronales probabilísticas PNN

- Arquitectura de la red



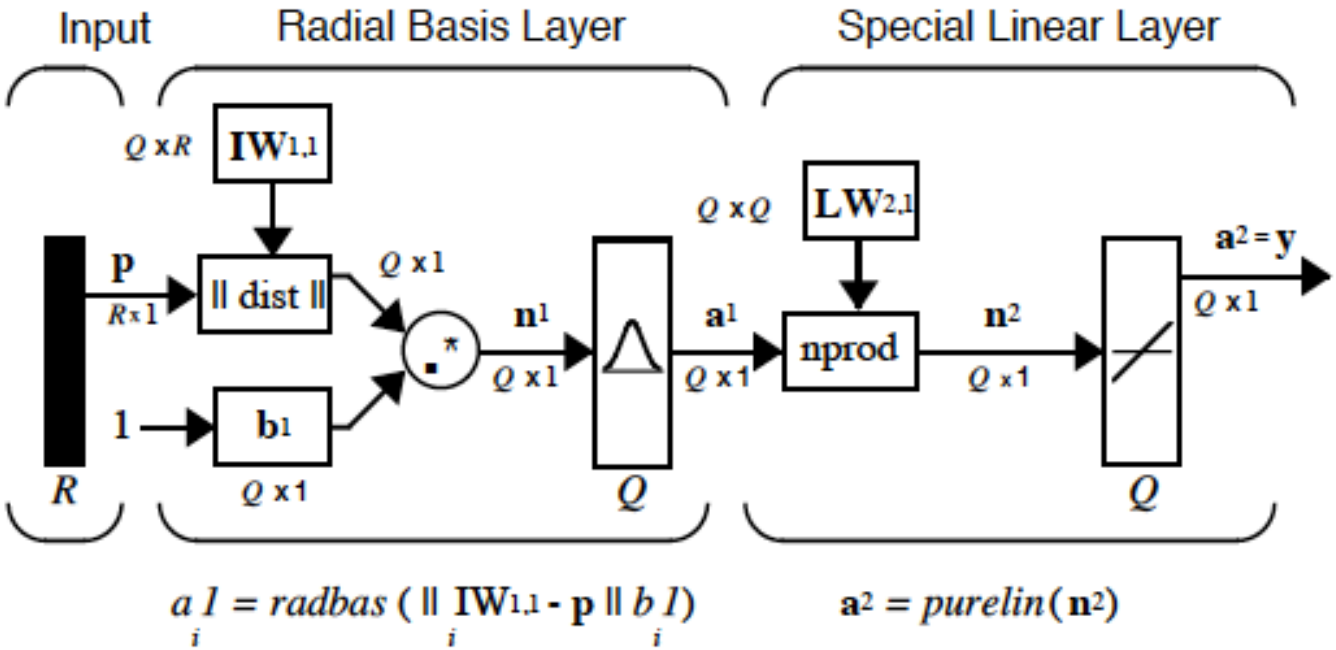

- $R$ número de elementos en el vector de entrada.
- $Q$ número de pares entrada/salida, número de neuronas en la capa 1.
- $K$ número de clases de datos de entrada, número de neuronas en la capa 2.

# Redes de regresión generalizadas GRNN

- Se suelen usar como aproximadores de funciones
- Su estructura consta de dos capas:
  - Un capa de neuronas de base radial
  - Una capa lineal especial

# Redes de regresión generalizadas GRNN

- Arquitectura de la red




- $R$ número de elementos en el vector de entrada.
- $Q$ número de neuronas en la capa 1, número de pares entrada/salida, número de neuronas en la capa 2.

# Redes de regresión generalizadas GRNN

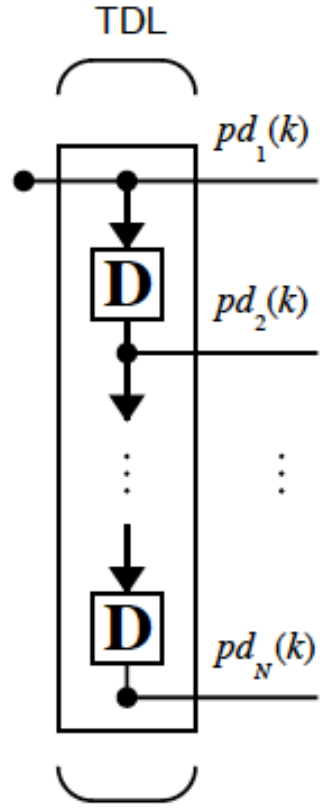
## Conceptos

- La primera capa es similar a las funciones de base radial.
- La segunda es ligeramente diferente:
- La función *nprod* (normproduct) produce un vector $n^2$ , donde cada elemento es el producto puntual (dot product) de una fila de $LW^{2,1}$ por el vector de entrada a la segunda capa $a^1$, todo ello normalizado por la suma de los elementos de $a^1$.

# Redes lineales con retardos

- Este tipo de redes también denominadas a veces como redes dinámicas suelen utilizar las señales físicas retardadas.
- Lineas de retardo.
  - En los diseños suele aparecer un elemento que genera las señales con retardo en el tiempo, es decir una especie de registro de desplazamiento pero analógico (tapped delay line TDL)
  - Genera un vector de señales retardadas en el tiemp, si la entrada es $p(k)$, el vector que se genera a la salida es:

$$(pd_1(k), pd_2(k), \frac{1}{2}., pd_N(k))^T$$

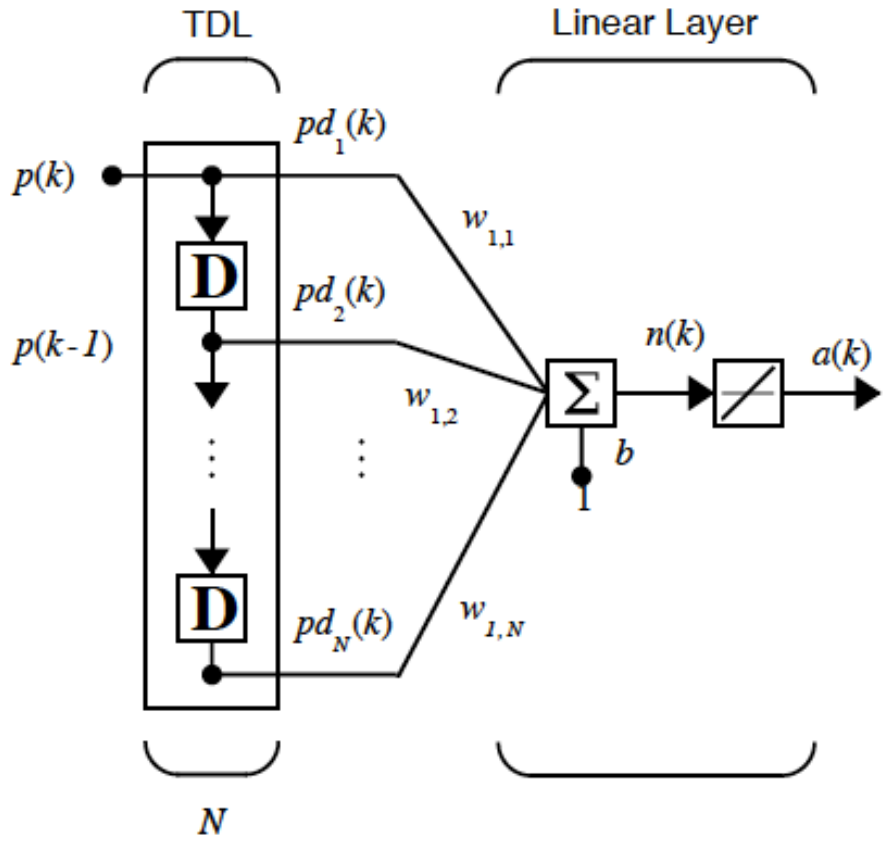


# Filtro lineal

- Combinando una línea de retardo con una neurona lineal se obtiene el típico filtro FIR
- La salida viene dada por

$$a(k) = purelin(Wp + b)$$

$$= \sum_{i=1}^{N} w_{1,i} p(k-i-1) + b$$

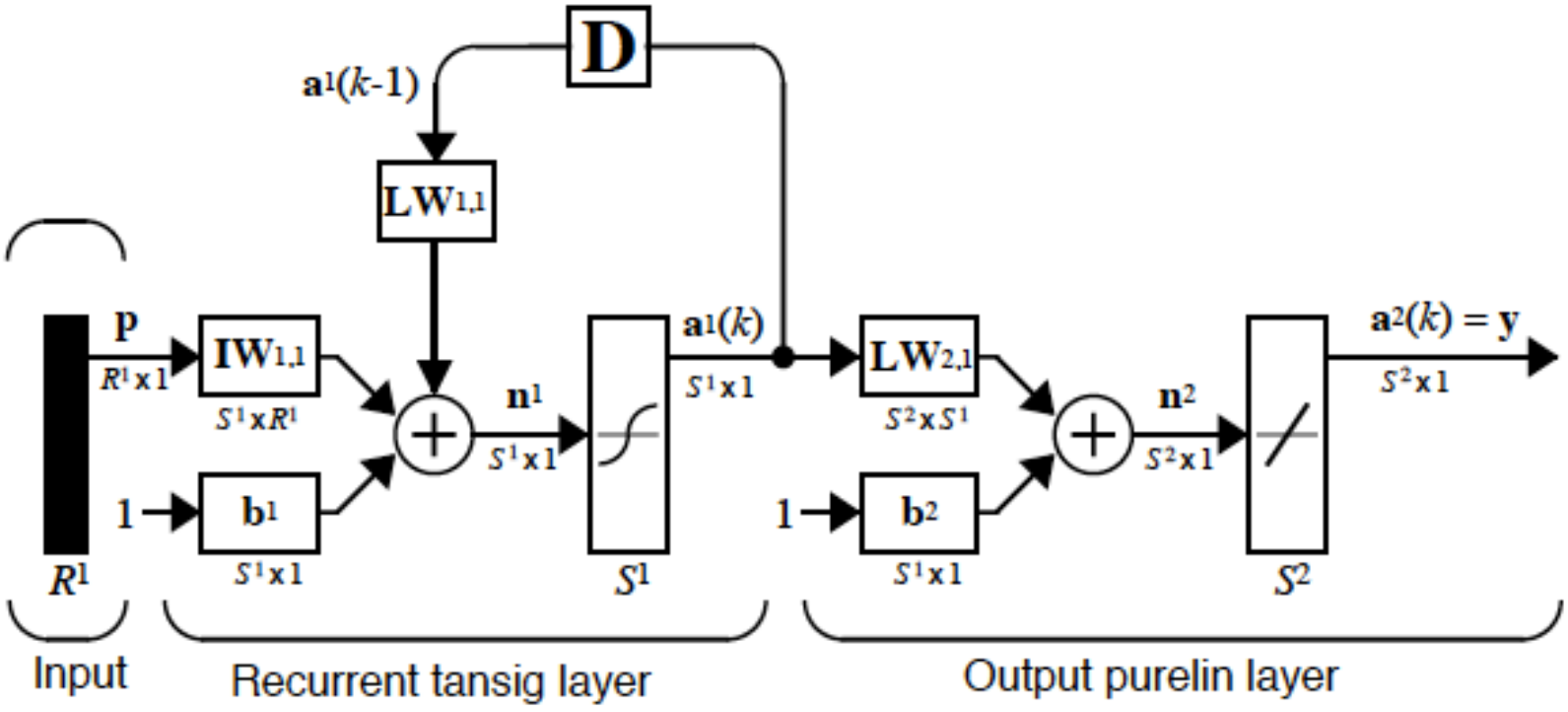


# Redes de Elman

- Las redes de Elman son un tipo de redes orientadas a sistemas dinámicos, presentan:
  - Una estructura recurrente
  - Líneas de retardo o al menos retardos
- Estas redes permiten detectar y generar patrones variables en el tiempo
- Este tipo de redes tiene dos capas y se realimenta de la salida de la primera capa a la entrada de la primera capa.
- La existencia de recurrencia en la red la da capacidad para aprender:
  - Patrones espaciales
  - Patrones temporales

# Redes de Elman

- Arquitectura de la red




$$\mathbf{a}^1(k) = \mathbf{tansig}\left(\mathbf{IW}^{1,1}\mathbf{p} + \mathbf{LW}^{1,1}\mathbf{a}^1(k\text{-}1) + \mathbf{b}^1\right) \qquad \mathbf{a}^2(k) = \mathbf{purelin}\left(\mathbf{LW}^{2,1}\mathbf{a}^1(k) + \mathbf{b}^2\right)$$

# Redes de Elman

## Aprendizaje

- Existen diversos métodos de entrenamiento, en Matlab hay dos: *train* y *adapt*.
- El **método train** funciona del siguiente modo en cada epoch:
  - Se le presenta la *secuencia entera* a la red, y las salidas de la red se calculan y comparan con las salidas deseadas (target) generándose una secuencia de error.
  - Para cada ciclo de tiempo (time step), el error es propagado hacia atrás (backpropagated) para encontrar los gradientes de errores para cada peso y bias.
  - Este gradiente es en realidad una aproximación ya que las contribuciones de pesos y bias a los errores vía la conexión recurrente retardada se desconoce.
  - Este gradiente se usa para actualizar los pesos mediante la función de entrenamiento backpropagation.

# Redes de Elman

## Aprendizaje

- El **método adapt** funciona del siguiente modo en cada epoch:
  - Se le presenta *un vector de entrada* a la red, y las salidas de la red se calculan y comparan con las salida deseada (target) generándose un error.
  - El error es propagado hacia atrás (backpropagated) para encontrar los gradientes de errores para cada peso y bias.
  - Este gradiente es una aproximación ya que las contribuciones de pesos y bias a los errores via la conexión recurrente retardada se desconoce.
  - Este gradiente se usa para actualizar los pesos mediante la función de entrenamiento backpropagation.

# Bibliografía


- Haykin, S. S. (2009). Neural networks and learning machines/Simon Haykin. New York: Prentice Hall,.
- Bishop, C. M.. Pattern recognition and machine learning. Springer 2006.
- Sánchez Camperos, E. N., Alanís García, A. Y. (2006). Redes Neuronales: Conceptos fundamentales y aplicaciones a control automático. Edgar Nelson Sánchez Camperos y Alma Yolanda Alanís García.
- Demuth, H. B., Beale, M. H., De Jess, O., Hagan, M. T. (2014). Neural network design. Martin Hagan.

- Fin L7