Tema 3: Perceptrón multicapa. Retropropagación

RETROPROPAGACIÓN / BACKPROPAGATION

- *Backpropagation* o la regla delta generalizada es un algoritmo de entrenamiento para redes *feedforward* multicapa.

- Se trata de un método por descenso de gradiente para minimizar el error cuadrático total de la salida.

- El objetivo del algoritmo es proporcionar un balance adecuado entre **memorización** (responder correctamente a los patrones que se han utilizado en el entrenamiento) y **generalización** (proporcionar una respuesta razonable para patrones que son similares pero no idénticos a los del entrenamiento).

- El entrenamiento consta de 3 partes: (i) la propagación hacia delante del patrón de entrenamiento de entrada, (ii) el cálculo del error asociado y su retropropagación, y (iiii) el ajuste de los pesos.

RETROPROPAGACIÓN / BACKPROPAGATION

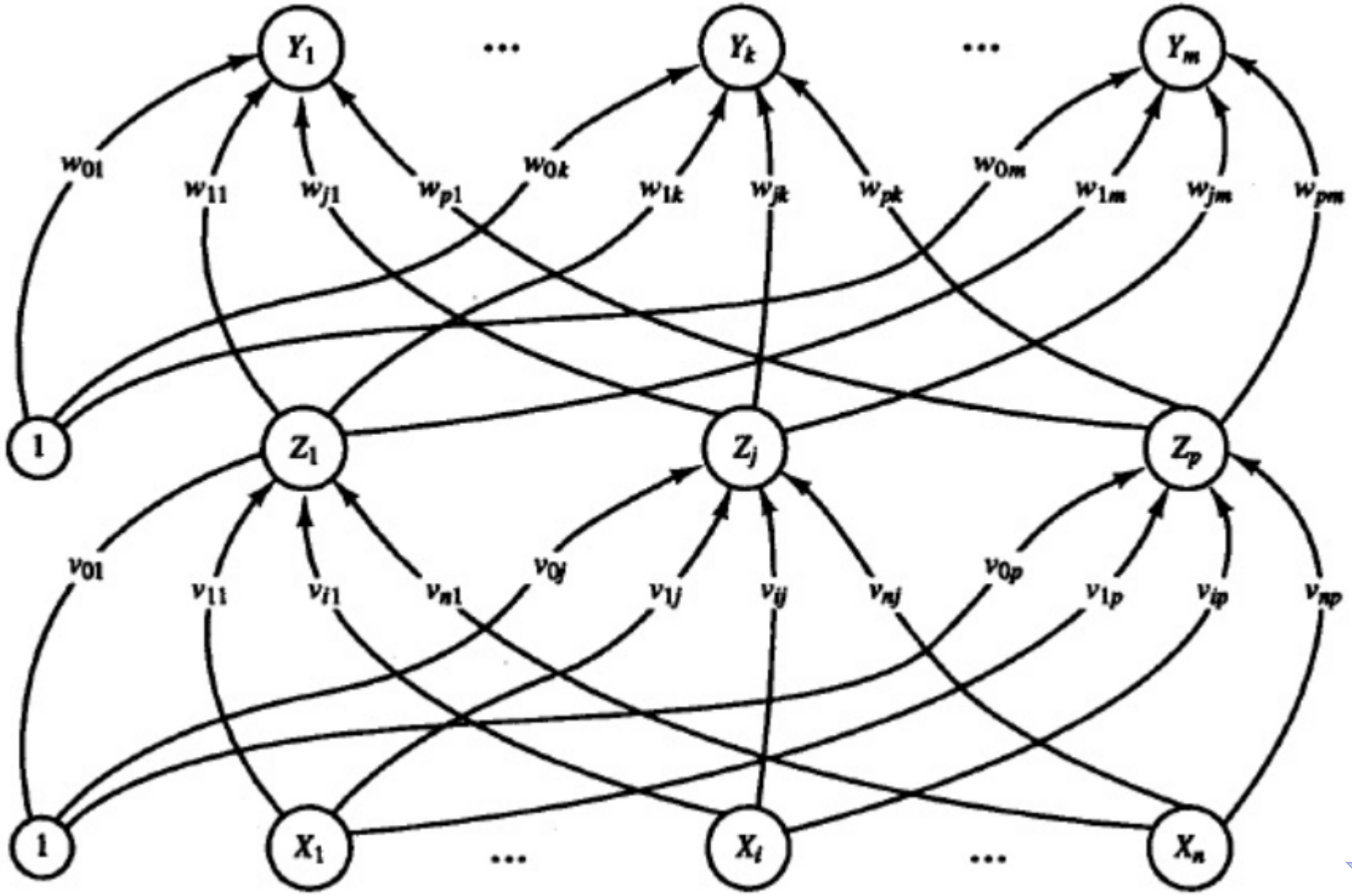
- Después del aprendizaje, la fase de explotación sólo tiene que realizar los cálculos de la propagación hacia delante.

- El entrenamiento puede ser lento pero la fase de explotación se ejecuta muy rápidamente en una red ya entrenada.

- Una red monocapa tiene muchas limitaciones en el aprendizaje, mientras que una red multicapa (con una o más capas ocultas) puede aprender casi cualquier mapa de entrada/salida.

- En la mayor parte de las aplicaciones basta con una sola capa oculta.

ARQUITECTURA DE LA RED MULTICAPA



Fase feedforward

Fase de retropropagación

NOMENCLATURA DE RETROPROPAGACIÓN

x Vector de entrada de entrenamiento $\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_n)$

t Vector objetivo de salida $\mathbf{t} = (t_1, \ldots, t_k, \ldots, t_m)$

δ_k Corrección de error para el peso w_{jk} que se debe al error en la neurona de salida Y_k ; también es la información sobre el error en la neurona Y_k que se retropropaga a las neuronas ocultas que envían información a la neurona Y_k .

δ_j Corrección de error para el peso v_{ij} que se debe a la retropropagación de la información de error de la capa de salida a la neurona oculta Z_j .

α Tasa de aprendizaje

X_i Neurona de entrada i (para una neurona de entrada, la señal de entrada y la de salida es la misma: x_i)

NOMENCLATURA DE RETROPROPAGACIÓN

v_{0j} Sesgo para la neurona oculta j

Z_j Neurona oculta j

La entrada neta a Z_j es $z\_in_j = v_{0j} + \sum_i x_i v_{ij}$

La señal de salida (activación) de Z_j es $z_j = f(z\_in_j)$

w_{0k} Sesgo para la neurona de salida k

Y_k Neurona de salida k

La entrada neta a Y_k es $y\_in_k = w_{0k} + \sum_j z_j w_{jk}$

La señal de salida (activación) de Y_k es $y_k = f(y\_in_k)$

FUNCIÓN DE TRANSFERENCIA

- En retropropagación, la función de transferencia o de activación tiene que ser continua, diferenciable y monótonamente no-decreciente.

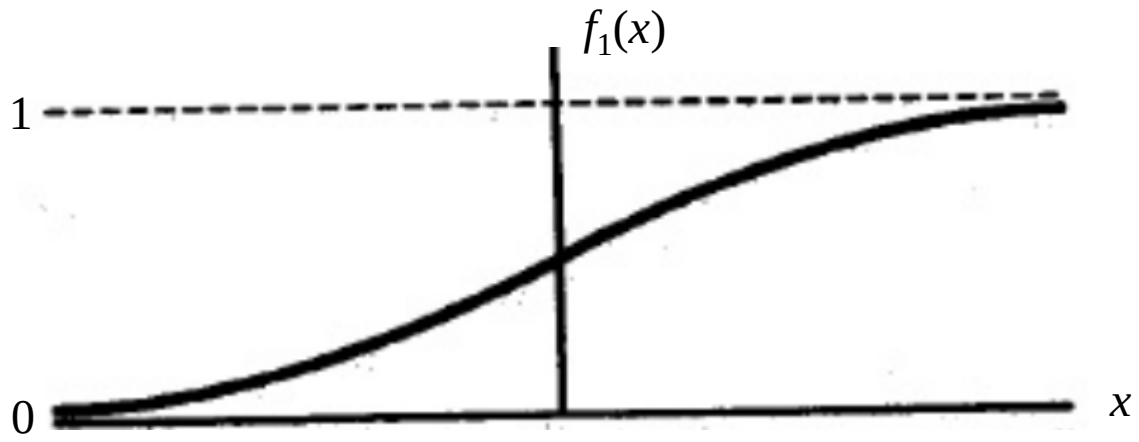
- Es conveniente que no tenga un coste computacional alto.

- Para la mayoría de las funciones utilizadas, el valor de su derivada se puede expresar en términos del valor de la función (lo que reduce el coste computacional).

- La función normalmente satura, es decir, se aproxima al valor máximo y mínimo de forma asintótica.

FUNCIÓN DE TRANSFERENCIA

Sigmoide binaria

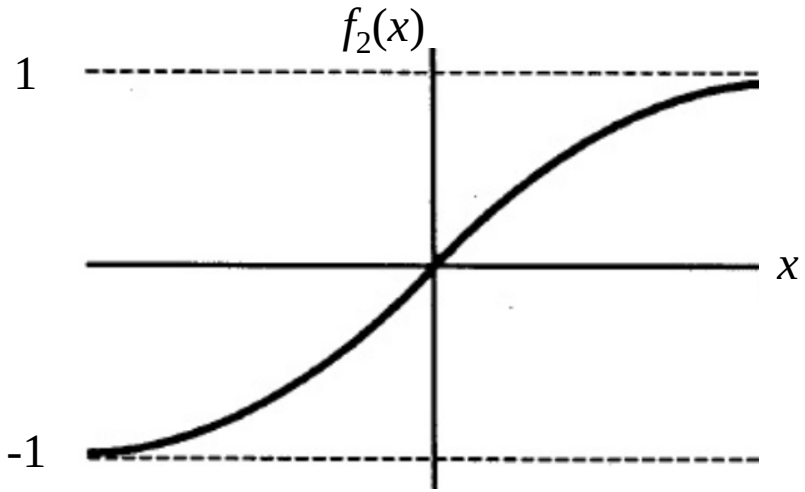


$$f_1(x) = \frac{1}{1+\exp(-x)}$$

$$f_1'(x) = f_1(x)[1 - f_1(x)]$$

FUNCIÓN DE TRANSFERENCIA

Sigmoide bipolar



$$f_2(x) = \frac{2}{1+\exp(-x)} - 1$$

$$f_2'(x) = \frac{1}{2}[1+ f_2(x)][1- f_2(x)]$$

Algoritmo de retropropagación (1/3)

Paso 0: Inicializar todos los pesos y sesgos (valores aleatorios pequeños).
Establecer la tasa de aprendizaje α (un valor pequeño).

Paso 1: Mientras que la condición de parada sea falsa, ejecutar pasos 2-9:

Paso 2: Para cada par de entrenamiento, ejecutar los pasos 3-8:

Feedforward

Paso 3: Establecer las activaciones a las neuronas de entrada

$x_i = s_i \quad (i=1\ldots n)$

Paso 4: Calcular la respuesta de las neuronas de la capa oculta:

$z_j \qquad (j=1\ldots p)$

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij} \qquad z_j = f(z\_in_j)$$

Paso 5: Calcular la respuesta de las neuronas de salida:

$y_k \qquad (k=1\ldots m)$

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk} \qquad y_k = f(y\_in_k)$$

Algoritmo de retropropagación (2/3)

Retropropagación del error

Paso 6: Cada neurona de salida (Y_k, $k=1\ldots m$) recibe un patrón objetivo que corresponde al patrón de entrada de entrenamiento, calcula el error: $\delta_k = (t_k - y_k) f'(y\_in_k)$,
calcula su corrección de peso (utilizada para actualizar w_{jk} más tarde): $\Delta w_{jk} = \alpha \delta_k z_j$,
calcula su corrección de sesgo (utilizada para actualizar w_{0k} más tarde): $\Delta w_{0k} = \alpha \delta_k$
y envía δ_k a las neuronas de la capa anterior.

Paso 7: Cada neurona de la capa oculta (Z_j, $j=1\ldots p$) suma sus entradas delta (de las neuronas de la capa posterior)

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

lo multiplica por la derivada de su función de activación para calcular el error: $\delta_j = \delta\_in_j f'(z\_in_j)$,
calcula su corrección de peso (utilizada para actualizar v_{ij} más tarde): $\Delta v_{ij} = \alpha \delta_j x_i$,
y calcula su corrección de sesgo para actualizar v_{0j} más tarde):

$$\Delta v_{0j} = \alpha \delta_j$$

Algoritmo de retropropagación (3/3)

Actualizar pesos y sesgos

Paso 8: Cada neurona de salida (Y_k, k=1…m) actualiza sus pesos y sesgo (j=0,…,p):

$$w_{jk}\,(\text{nuevo}) = w_{jk}\,(\text{anterior}) + \Delta w_{jk}$$

Cada neurona de la capa oculta (Z_j, j=1…p) actualiza sus pesos y sesgo (i=0,…,n):

$$v_{ij}\,(\text{nuevo}) = v_{ij}\,(\text{anterior}) + \Delta v_{ij}$$

Paso 9: Comprobar la condición de parada.

Fase de explotación

Paso 0: Inicializar todos los pesos (de acuerdo al resultado de entrenamiento).

Paso 1: Para cada vector de entrada, ejecutar pasos 2-4:

Paso 2: Para $i=1\ldots n$: establecer la activación de la neurona de entrada $x_i = s_i$

Paso 3: Para $j=1\ldots p$:

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij} \qquad z_j = f(z\_in_j)$$

Paso 4: Para $k=1\ldots m$:

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j w_{jk} \qquad y_k = f(y\_in_k)$$

OBSERVACIONES

• Es importante apreciar la distinción entre los deltas para las neuronas de salida (paso 6: δ_k) y los deltas para las neuronas de la capa oculta (paso 7: δ_j)

• Una época es un ciclo que involucra a todos los patrones de entrenamiento. Normalmente hacen falta muchas épocas en *backpropagation* para implementar el aprendizaje.

• Una alternativa a la versión de *backprogagation* anterior es lo que se conoce como *batch backpropagatio*n o entrenamiento por lotes, en la cual las actualizaciones de los pesos se acumulan durante una época antes de ser aplicadas.

• El gradiente negativo del error proporciona la dirección en la que decrece más rápidamente.

Opciones para la retropropagación

1. Elección de pesos y sesgos iniciales:

- La elección de pesos iniciales puede provocar que se alcance un mínimo global del error o sólo un mínimo local, y también tiene influencia en el número de épocas para llegar a la convergencia.
- La actualización de los pesos entre dos neuronas depende de la derivada de la función de activación de la neurona receptora y de la activación de la neurona emisora. Es importante elegir los pesos iniciales de forma que ni las activaciones, ni las derivadas de las activaciones, sean cero.
- Los valores de los pesos iniciales no pueden ser muy grandes, o las señales de entrada iniciales caerán en una región donde la derivada de la función sigmoidal tiene un valor muy pequeño (región de saturación).
- Tampoco pueden ser muy pequeños, pues resultará en un aprendizaje muy lento.

Opciones para la retropropagación

1. Elección de pesos y sesgos iniciales (continúa):

- Un método muy extendido es inicializar los pesos (y los sesgos) a valores aleatorios entre -0.5 y 0.5 (o entre -1 y 1)

Los valores pueden ser positivos o negativos, puesto que después del entrenamiento la regla de aprendizaje también produce este tipo de valores.

Opciones para la retropropagación

1. Elección de pesos y sesgos iniciales (continúa):

- Inicialización Nguyen-Widrow (para mejorar el aprendizaje de las neuronas de la capa oculta distribuyendo los pesos de forma que la entrada neta a cada neurona de la capa oculta esté en el rango en el que esa neurona aprende mejor). La función de activación es $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Los pesos de la capa oculta a las neuronas de salida se inicializan a valores aleatorias entre -0.5 y 0.5 como arriba.

Se calcula el factor de escala $\beta = 0.7(p)^{1/n}$ donde n es el número de neuronas de entrada y p en número de neuronas de la capa oculta.

Para cada neurona de la capa oculta ($j=1\ldots, p$), se inicializan los pesos: $v_{ij}(\text{nuevo})$ = número aleatorio entre -0.5 y 0.5 (o entre $-\gamma$ y γ) y se calcula:

$$\left\|\mathbf{v}_j(anterior)\right\| = \sqrt{v_{1j}(anterior)^2 + v_{2j}(anterior)^2 + \ldots + v_{nj}(anterior)^2}$$

$$v_{ij} = \frac{\beta v_{ij}(anterior)}{\left\|\mathbf{v}_j(anterior)\right\|}$$

El sesgo v_{0j} se establece con un número aleatorio entre $-\beta$ y β

Opciones para la retropropagación

2. Duración del entrenamiento

• Normalmente el entrenamiento se realiza mientras el error cuadrático total no alcance un mínimo. Para esto se puede utilizar un conjunto de patrones de entrenamiento y otro (distinto) de comprobación del error durante el entrenamiento (conjunto de validación), y un último conjunto de test.

• Cuando el error comienza a crecer, la red memoriza los patrones de entrenamiento de forma demasiado específica (y por tanto pierde capacidad para generalizar). En ese momento hay que parar el entrenamiento.

• También se puede establecer un criterio para parar cuando se han alcanzado un número máximo de épocas.

Opciones para la retropropagación

3. Número de patrones de entrenamiento

• Existen recomendaciones para establecer relaciones entre el número de patrones de entrenamiento (P), el número de pesos a entrenar (W) y la expectativa de precisión de la clasificación e.

• ¿En qué circunstancias una red que se entrena para clasificar correctamente un porcentaje dado de patrones de entrenamiento también clasificará correctamente patrones de test que se han obtenido de la misma forma que los de entrenamiento?

• Respuesta: con el número suficiente de patrones $P=W/e$. La red se entrena para clasificar correctamente $1-(e/2)$ de los patrones de entrenamiento, con $0<e\leq 1/8$. W es el número de pesos.

• Por ejemplo si tenemos una red con 80 pesos y queremos que e=0.1, necesitaremos 800 patrones de entrenamiento para asegurarnos que se van a clasificar el 90% de los patrones de test, asumiendo que la red se entrenó para clasificar el 95% de los patrones de entrenamiento correctamente.

Opciones para la retropropagación

4. Representación de los datos

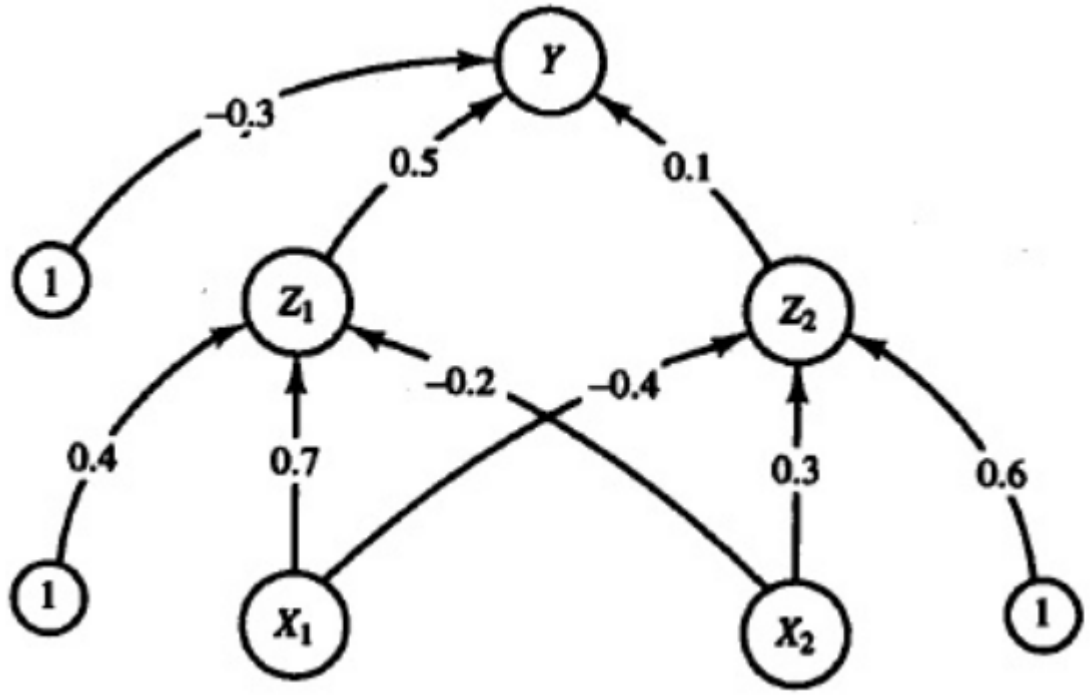
• En general es mejor la representación bipolar que la binaria.

• Se puede utilizar codificaciones continuas en vez de binarias o bipolares, aunque son más difíciles de entrenar y requieren normalización. Para algunos tipos de aplicaciones esta codificación es imprescindible.

• También se puede establecer un criterio para parar cuando se han alcanzado un número máximo de épocas.

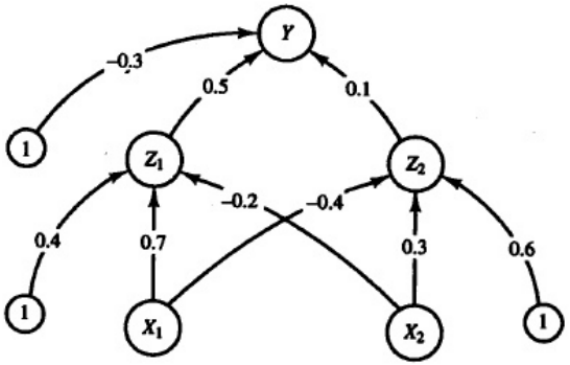
Ejercicio:

Calcular los nuevos valores de los pesos con retropropagación cuando se presenta el patrón (-1,1) y el objetivo de salida es 1. Utilizar α=0.25 y sigmoide bipolar para la función de transferencia.



$(x_1,x_2)=(-1,1) \rightarrow 1$

$\alpha=0.25$



$$f(x)=\frac{2}{1+\exp(-x)}-1$$

$$f'(x)=\frac{1}{2}[1+f(x)][1-f(x)]$$

Paso 0: Inicializar todos los pesos y sesgos a los valores especificados en el ejemplo
Establecer la tasa de aprendizaje $\alpha=0.25$.

Paso 1: Mientras que la condición de parada sea falsa, ejecutar pasos 2-9:

Paso 2: Para cada par de entrenamiento, ejecutar los pasos 3-8:

Feedforward

Paso 3: Establecer las activaciones a las neuronas de entrada

$x_1 = -1; x_2 = 1$

Paso 4: Calcular la respuesta de las neuronas de la capa oculta:

$$z\_in_1 = v_{01} + \sum_{i=1}^{2} x_i v_{ij} = 0.4 + (-1)\cdot 0.7 + 1\cdot(-0.2) = -0.5 \qquad z_1 = f(z\_in_1) = -0.25$$

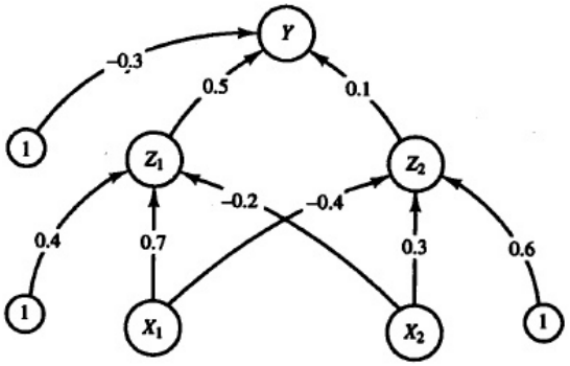
$$z\_in_2 = v_{02} + \sum_{i=1}^{2} x_i v_{ij} = 0.6 + (-1)\cdot(-0.4) + 1\cdot(0.3) = 1.3 \qquad z_2 = f(z\_in_2) = 0.57$$

Paso 5: Calcular la respuesta de la neurona de salida:

$$y\_in = w_0 + \sum_{j=1}^{2} z_j w_j = -0.3 + (-0.25)\cdot 0.5 + 0.57\cdot 0.1 = -0.37 \qquad y = f(y\_in) = -0.18$$

$(x_1,x_2)=(-1,1) \rightarrow 1$

$\alpha=0.25$



$$f(x) = \frac{2}{1+\exp(-x)} - 1$$

$$f'(x) = \frac{1}{2}[1+f(x)][1-f(x)]$$

Retropropagación del error

Paso 6: $\delta_k = (t_k - y_k) f'(y\_in_k) =$

$= [1-(-0.18)]\, f'(-0.37) = 1.18 \cdot 0.5 \cdot [1+(-0.18)] \cdot [1-(-0.18)] = 0.57$

Actualización de pesos $w_{jk} = \alpha \delta_k z_j$,

$\Delta w_{01} = 0.25 \cdot 0.57 = 0.1425$

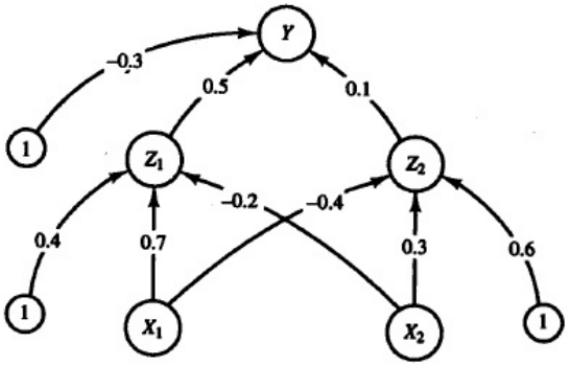
$\Delta w_{11} = 0.25 \cdot 0.57 \cdot (-0.25) = -0.0356$

$\Delta w_{21} = 0.25 \cdot 0.57 \cdot 0.57 = 0.0812$

y envía δ_k a las neuronas de la capa anterior.

$(x_1,x_2)=(-1,1) \rightarrow 1$

$\alpha=0.25$



$$f(x) = \frac{2}{1+\exp(-x)} - 1$$

$$f'(x) = \frac{1}{2}[1+f(x)][1-f(x)]$$

Retropropagación del error

Paso 7: Cada neurona de la capa oculta (Z_j, $j=1\ldots p$) suma sus entradas delta (de las neuronas de la capa posterior)

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

$$\delta\_in_1 = 0.57 \cdot 0.5 = 0.285$$

$$\delta\_in_2 = 0.57 \cdot 0.1 = 0.057$$

lo multiplica por la derivada de su función de activación para calcular el error: $\delta_j = \delta\_in_j f'(z\_in_j)$

$$\delta_1 = \delta\_in_1 f'(z\_in_1) = 0.285 \cdot f'(-0.5) = 0.13$$

$$\delta_2 = \delta\_in_2 f'(z\_in_2) = 0.057 \cdot f'(1.3) = 0.02$$

calcula su corrección de peso (utilizada para actualizar v_{ij} más tarde): $\Delta v_{ij} = \alpha \delta_j x_i$,

$\Delta v_{01} = 0.25 \cdot 0.13 = 0.0325$
$\Delta v_{02} = 0.25 \cdot (0.02) = 0.005$
$\Delta v_{11} = 0.25 \cdot 0.13 \cdot (-1) = -0.0325$
$\Delta v_{12} = 0.25 \cdot (0.02) \cdot (-1) = -0.005$
$\Delta v_{21} = 0.25 \cdot 0.13 \cdot 1 = 0.0325$
$\Delta v_{22} = 0.25 \cdot (0.02) \cdot 1 = 0.005$

Actualizar pesos y sesgos

Paso 8: Cada neurona de salida (Y_k, k=1…m) actualiza sus pesos y sesgo (j=0,…,p):

$$w_{jk}\,(\text{nuevo}) = w_{jk}\,(\text{anterior}) + \Delta w_{jk}$$

Cada neurona de la capa oculta (Z_j, j=1…p) actualiza sus pesos y sesgo (i=0,…,n):

$$v_{ij}\,(\text{nuevo}) = v_{ij}\,(\text{anterior}) + \Delta v_{ij}$$

w_{01} (nuevo) = -0.3 +0.1425=-0.1575
w_{11} (nuevo) = 0.5+(-0.0342) =0.4658
w_{21} (nuevo) = 0.1 + 0.0821 = 0.182

v_{01} (nuevo)= 0.4+0.0325=0.4325
……..

Paso 9: Comprobar la condición de parada.

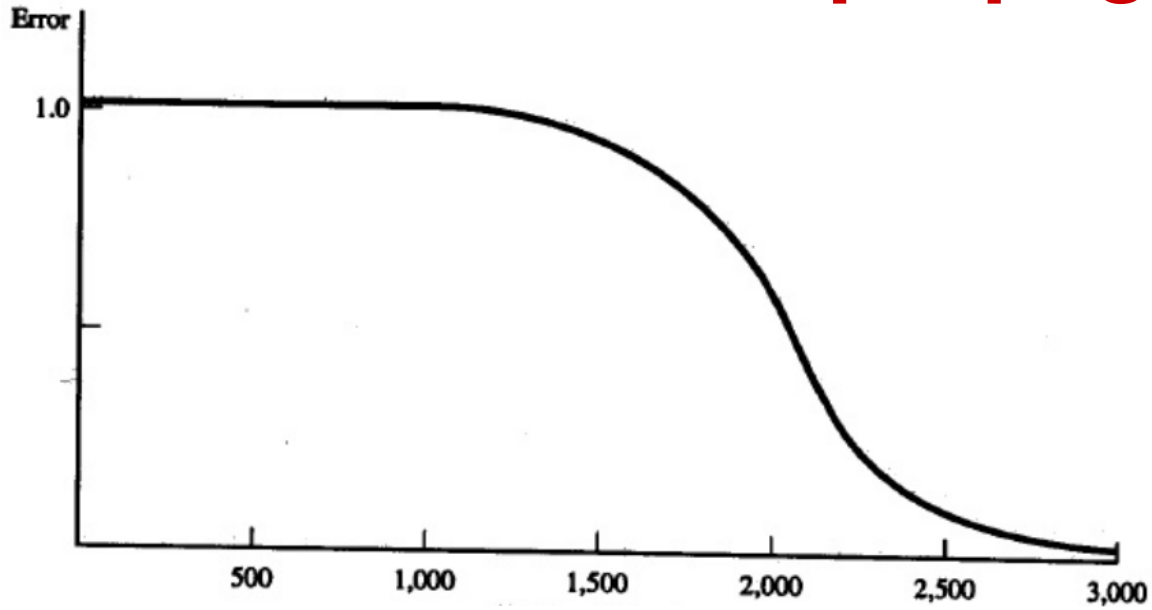
Función XOR con retropropagación

• Arquitectura: 2 neuronas de entrada, 4 en la capa oculta, 1 de salida.

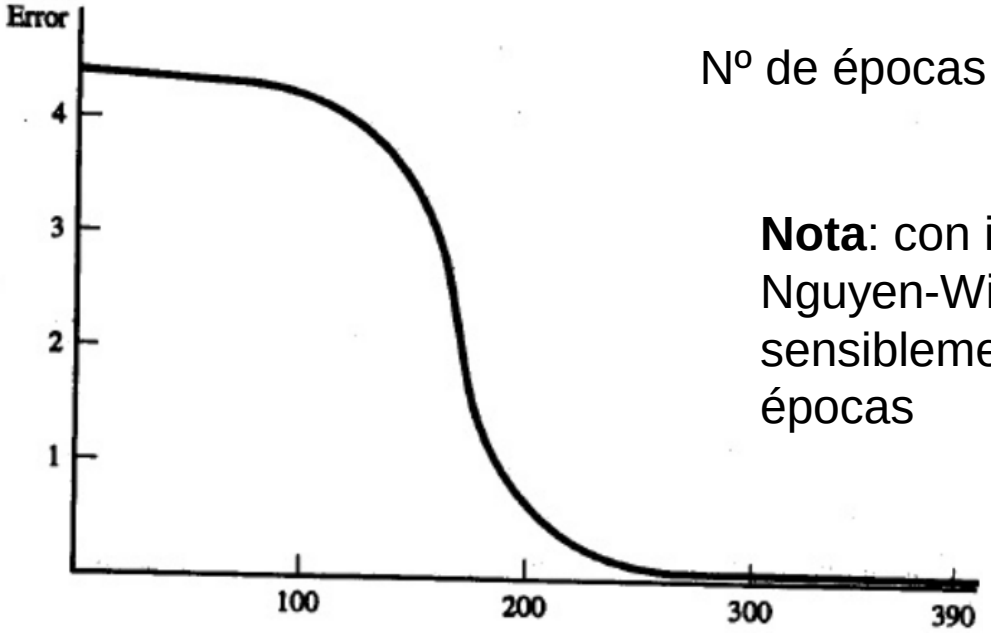
• Probamos con codificación binaria y con bipolar
• Pesos iniciales aleatorias entre -0.5 y 0.5:
• Tasa de aprendizaje $\alpha=0.02$
• Condición de parada: error cuadrático total para los cuatro patrones de entrenamiento < 0.05.

Resultados XOR con retropropagación

Codificación binaria



Codificación bipolar



Nº de épocas

Nota: con inicialización de Nguyen-Widrow se reduce sensiblemente este nº de épocas

Derivación de la regla de retropropagación (1/2)

- El error cuadrático a minimizar es:

$$E = \frac{1}{2}\sum_k [t_k - y_k]^2 \quad \text{donde para una neurona } K \quad y\_in_K = \sum_j z_j w_{jK}$$

- El gradiente de E (un vector que consiste de las derivadas parciales de E con respecto a cada peso) proporciona la dirección de más rápido crecimiento de este error,

$$\frac{\partial E}{\partial w_{JK}} = \frac{1}{2}\frac{\partial}{\partial w_{JK}}\sum_k [t_k - y_k]^2 = \frac{1}{2}\frac{\partial}{\partial w_{JK}}[t_K - f(y\_in_K)]^2$$

$$= -[t_K - y_K]\frac{\partial}{\partial w_{JK}} f(y\_in_K) = -[t_K - y_K] f'(y\_in_K)\frac{\partial}{\partial w_{JK}}(y\_in_K)$$

$$= -[t_K - y_K] f'(y\_in_K) z_J$$

- Y por tanto $\Delta w_{jk} = -\alpha \frac{\partial E}{\partial w_{jk}} = \alpha \delta_k z_j \quad \text{con } \delta_k = [t_k - y_k] f'(y\_in_k)$

Derivación de la regla de retropropagación (2/2)

- Para los pesos a las neuronas de la capa oculta:

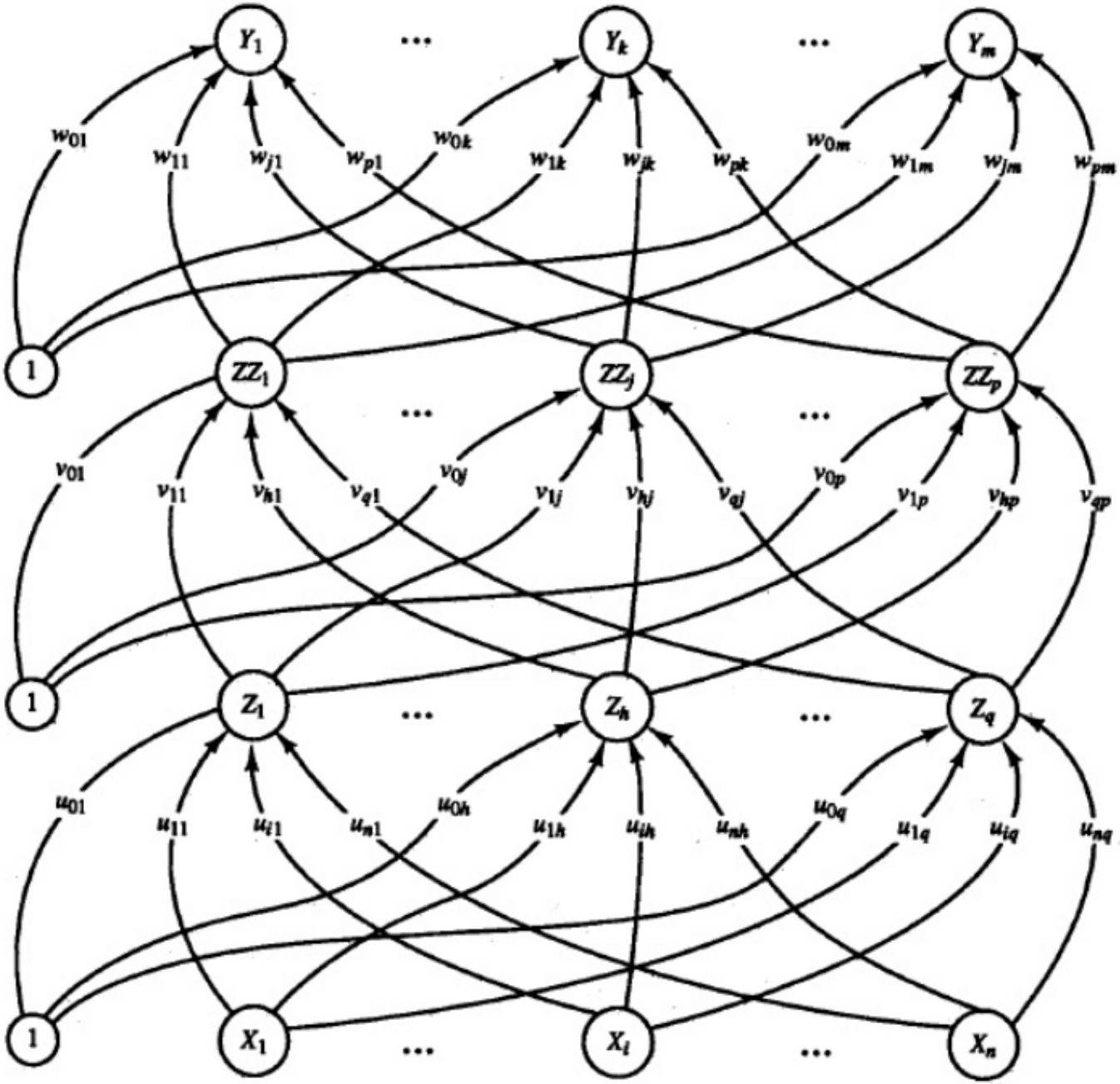
$$\frac{\partial E}{\partial v_{IJ}} = -\sum_k [t_k - y_k] \frac{\partial}{\partial v_{IJ}} y_k = -\sum_k [t_k - y_k] f'(y\_in_k) \frac{\partial}{\partial v_{IJ}} (y\_in_K)$$

$$= -\sum_k \delta_k \frac{\partial}{\partial v_{IJ}} (y\_in_K) = -\sum_k \delta_k w_{JK} \frac{\partial}{\partial v_{IJ}} z_J$$

$$= -\sum_k \delta_k w_{JK} f'(z\_in_J) x_I$$

- Y por tanto $\Delta v_{ij} = -\alpha \frac{\partial E}{\partial v_{ij}} = \alpha \delta_j x_i$ con $\delta_j = \sum_k \delta_k w_{jk} f'(z\_in_j)$

Retropropagación con dos capas ocultas



Algoritmo de retropropagación para dos capas ocultas (1/3)

Paso 0: Inicializar todos los pesos y sesgos
Establecer la tasa de aprendizaje α

Paso 1: Mientras que la condición de parada sea falsa, ejecutar pasos 2-9:

Paso 2: Para cada par de entrenamiento, ejecutar los pasos 3-8:

Feedforward

Paso 3: Establecer las activaciones a las neuronas de entrada

$x_i = s_i \quad (i=1\ldots n)$

Paso 4: Calcular la respuesta de las neuronas de las capas ocultas:

$(h=1\ldots q) \qquad z\_in_h = u_{0h} + \sum_{i=1}^{n} x_i u_{ih} \qquad z_h = f(z\_in_h)$

$(j=1\ldots p) \qquad zz\_in_j = v_{0j} + \sum_{h=1}^{n} z_h v_{hj} \qquad zz_j = f(zz\_in_j)$

Paso 5: Calcular la respuesta de las neuronas de salida:

$(k=1\ldots m)$

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} zz_j w_{jk} \qquad y_k = f(y\_in_k)$$

Algoritmo de retropropagación para dos capas ocultas (2/3)

Retropropagación del error

Paso 6: Cada neurona de salida (Y_k, $k=1\ldots m$) recibe un patrón objetivo que corresponde al patrón de entrada de entrenamiento, calcula el error: $\delta_k = (t_k - y_k) f'(y\_in_k)$,
calcula su corrección de peso (utilizada para actualizar w_{jk} más tarde): $\Delta w_{jk} = \alpha \delta_k z z_j$,
calcula su corrección de sesgo (utilizada para actualizar w_{0k} más tarde): $\Delta w_{0k} = \alpha \delta_k$
y envía δ_k a las neuronas de la capa anterior.

Paso 7: Cada neurona de la 2ª capa oculta (ZZ_j, $j=1\ldots p$) suma sus entradas delta (de las neuronas de la capa posterior)

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

lo multiplica por la derivada de su función de activación para calcular el error: $\delta_j = \delta\_in_j f'(zz\_in_j)$,
calcula su corrección de peso (utilizada para actualizar v_{hj} más tarde): $\Delta v_{nj} = \alpha \delta_j Z_n$,
y calcula su corrección de sesgo para actualizar v_{0j} más tarde):

$$\Delta v_{0j} = \alpha \delta_j$$

y envía δ_j a las neuronas de la capa anterior.

Algoritmo de retropropagación para dos capas ocultas (3/3)

Paso 8: Cada neurona de la 1ª capa oculta (Z_h, h=1…q) suma sus entradas delta (de las neuronas de la capa posterior)

$$\delta\_in_h = \sum_{j=1}^{p} \delta_j v_{hj}$$

lo multiplica por la derivada de su función de activación para calcular el error: $\delta_h = \delta\_in_h f'(z\_in_h)$,
calcula su corrección de peso (utilizada para actualizar v_{ij} más tarde): $\Delta u_{ih} = \alpha \delta_h x_i$,
y calcula su corrección de sesgo para actualizar v_{0j} más tarde):

$$\Delta u_{0h} = \alpha \delta_h$$

Actualizar pesos y sesgos

Paso 9: Cada neurona de salida (Y_k, k=1…m) actualiza sus pesos y sesgo (j=0,…,p):

$$w_{jk}\,(\text{nuevo}) = w_{jk}\,(\text{anterior}) + \Delta w_{jk}$$

Cada neurona de la 2ª capa oculta (ZZ_j, j=1…p) actualiza sus pesos y sesgo (h=0,…,q):

$$v_{hj}\,(\text{nuevo}) = v_{hj}\,(\text{anterior}) + \Delta v_{hj}$$

Cada neurona de la 1ª capa oculta (Z_h, h=1…q) actualiza sus pesos y sesgo (i=0,…,n):

$$u_{ih}\,(\text{nuevo}) = u_{ih}\,(\text{anterior}) + \Delta u_{ih}$$

Paso 10: Comprobar la condición de parada.

Variantes de retropropagación

MOMENTO

• En retropropagación el cambio de un peso se puede hacer en una dirección que es la combinación del gradiente actual y del gradiente en el paso anterior (útil cuando hay patrones de entrenamiento muy distintos de la mayoría y para no alcanzar mínimos locales).

• Es conveniente utilizar una tasa de aprendizaje pequeña.

$$w_{jk}(t+1) = w_{jk}(t) + \alpha\delta_k z_j + \mu[w_{jk}(t) - w_{jk}(t-1)]$$
$$\text{o} \quad \Delta w_{jk}(t-1) = \alpha\delta_k z_j + \mu\Delta w_{jk}(t)$$

$$v_{ij}(t+1) = v_{ij}(t) + \alpha\delta_j x_i + \mu[v_{ij}(t) - v_{ij}(t-1)]$$
$$\text{o} \quad \Delta v_{ij}(t+1) = \alpha\delta_j x_i + \mu\Delta v_{ij}(t)$$

• El parámetro del momento μ *está en el rango (0,1)*

Variantes de retropropagación

TASA DE APRENDIZAJE ADAPTATIVA

- El objetivo es que cada peso tenga su tasa de aprendizaje

$$w_{jk}\,(\mathrm{t}+1) = w_{jk}\,(\mathrm{t}) + \alpha_{jk}\,(t+1)\delta_{\mathrm{k}} z_j$$

Variantes de retropropagación

OTRAS FUNCIONES DE ACTIVACIÓN

- Sigmoide para el intervalo $[a,b]$:

  Partiendo de la sigmoide binaria:

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

Definimos los parámetros $\gamma = b-a$ y $\eta = -a$

La función sigmoidal $g(x) = \gamma f(x) - \eta$ tiene el rango entre (a,b)

$$g'(x) = \frac{1}{\gamma}[\eta + g(x)][\gamma - \eta - g(x)]$$

Variantes de retropropagación

OTRAS FUNCIONES DE ACTIVACIÓN

- Sigmoide con parámetro de pendiente σ

$$f(x) = \frac{1}{1+\exp(-\sigma x)} \qquad f'(x) = \sigma\, f(x)[1 - f(x)]$$

$$\mathrm{g(x)} = \gamma\, f(x) - \eta = \frac{\gamma}{1+\exp(-\sigma\, x)} - \eta$$

$$g'(x) = \frac{\sigma}{\gamma}[\eta + g(x)][\gamma - \eta - g(x)]$$

Algunas variaciones del aprendizaje consideran σ adaptativo

LOS PERCEPTRONES MULTICAPA CON RETROPROPAGACIÓN SON APROXIMADORES UNIVERSALES

Se puede demostrar por el teorema de Kolmogorov que una red *feedforward* con una capa oculta puede representar cualquier función continua exactamente.

Retropropagación como algoritmo de aprendizaje profundo

• El algoritmo más común de aprendizaje profundo es el *backpropagation* o retropropagación.

• Puesto que un perceptrón multicapa entrenado con retropopagación es un aproximador universal, puede en principio aproximar/aprender cualquier función. En la práctica una red neuronal puede aprender mejor con múltiples capas ocultas (es decir siendo una red neuronal más profunda).

• Cuando una red tiene más de una capa, cada capa más profunda construye nuevas abstracciones por encima de las capas anteriores.

Problemas de las redes profundas

- Incrementar el número de capas suele producir dos problemas:

1. Los gradientes por los cuales desciende el cálculo de los pesos empiezan a hacerse más pequeños conforme añadimos capas, es decir se vuelven pequeños en relación a los pesos.

2. Sobre-entrenamiento (uno de los problemas principales en los paradigmas de aprendizaje automático): los datos de entrenamiento se aproximan demasiado y se pierde la capacidad de generalización.

- Para resolver este problema se utilizan distintos tipos de algoritmos de aprendizaje profundo.

Autoencoders

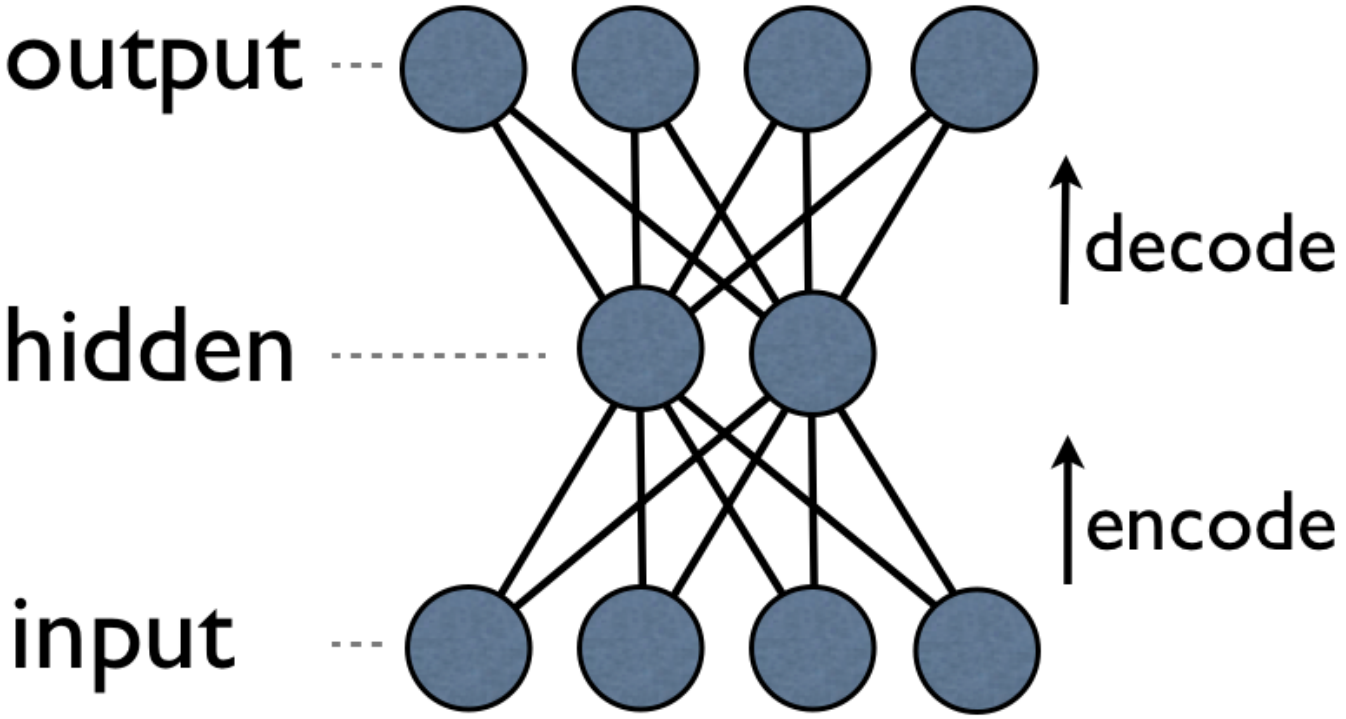
• Un autoencoder es típicamente una red neuronal de propagación hacia adelante cuyo objetivo es aprender una representación comprimida y distribuida de un conjunto de datos.

• La red se entrena para “recrear” la entrada de la red, es decir los datos de entrada y los datos objetivo son los mismos, solo que la salida está comprimida en cierta forma.

• Típicamente la arquitectura de un autoencoder puede ser una red con el mismo número de neuronas en la entrada y en la salida, con un número menor de neuronas en la capa oculta. De esta forma se obliga a la red a aprender solo las características más importantes mediante una reducción de dimensión. El objetivo es aprender la estructura interna, es decir, las principales características de los datos.

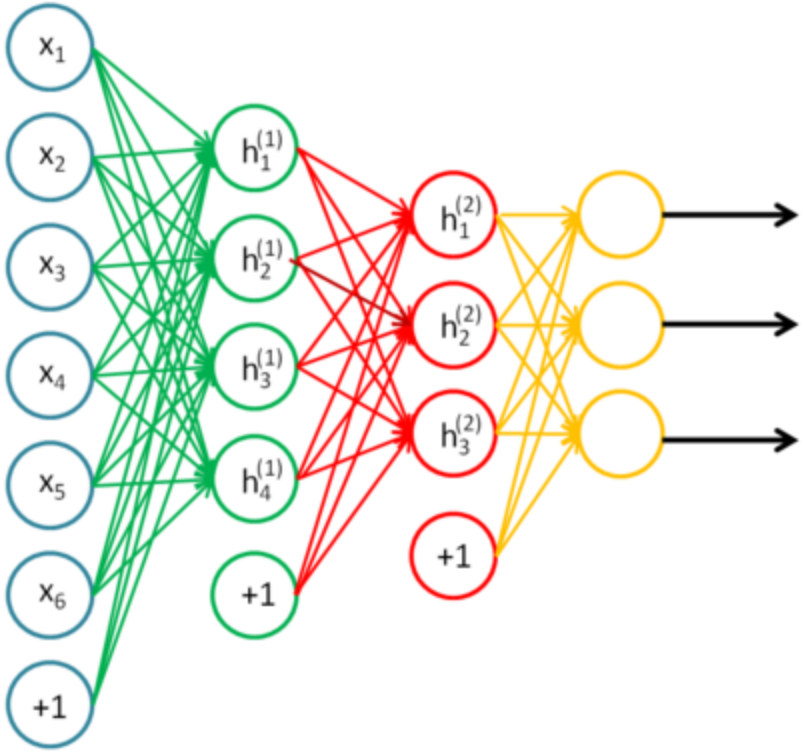
Ventaja de las representaciones compactas

Entrenando la red a aprender una representación compacta más sencilla evitamos el sobre-aprendizaje que existe con más probabilidad cuando utilizamos una representación más compleja.



Ejemplo de Red Profunda

Los autoencoders se pueden apilar para formar redes profundas. Estas redes se pueden entrenar de manera que cada capa aprende por separado para evitar los problemas de la desaparición de los gradientes o el sobre-aprendizaje que tiene la retropropagación.



Ejamplo de entrenamiento de Redes Profundas

La capa oculta *i* del autoencoder actúa como la capa de entrada del autoencoder *i*+1. Para entrenar la red se suelen utilizar algoritmos de aprendizaje “greedy” en los cuales:

1.Se entrena el primer autoencoder (*i*=1 con una capa de salida adicional) con retropropagación utilizando los patrones de entrenamiento
2.Se entrena el segundo autoencoder (*i*=2) sin tener en cuenta la primera capa oculta. La entrada a la red *i*=2 es la capa oculta de *i*=1, ya no nos interesa la capa de salida de *i*=1 y no la consideramos. La entrada se introduce en la capa de entrada de *i*=1 y se propaga a la salida de la capa *i*=2. Los pesos de *i*=2 (de la entrada a la oculta y de la oculta a la salida) se calculan con backpropagation.
3.Se repite el procedimiento anterior para todas las capas.

Ejemplo de entrenamiento de Redes Profundas

Los tres pasos anteriores se llaman pre-entrenamiento y sirven para inicializar los pesos correctamente. Sin embargo todavía no tenemos un mapa entre los datos de entrada y las salidas objetivo. Para obtenerlo, podemos entrenar ahora toda la red con retropropagación.

Los autoencoders apilados proporcionan un método efectivo de entrenamiento para inicializar los pesos de una red, y luego el perceptrón multicapa puede aprender “sin problemas” con retropropagación.