DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Backpropagation

Fernando Berzal, berzal@acm.org

Backpropagation

- Redes neuronales multicapa
- Entrenamiento de una neurona lineal
- Entrenamiento de una neurona sigmoidal
- Entrenamiento de neuronas ocultas
- El algoritmo de propagación de errores

Redes neuronales multicapa

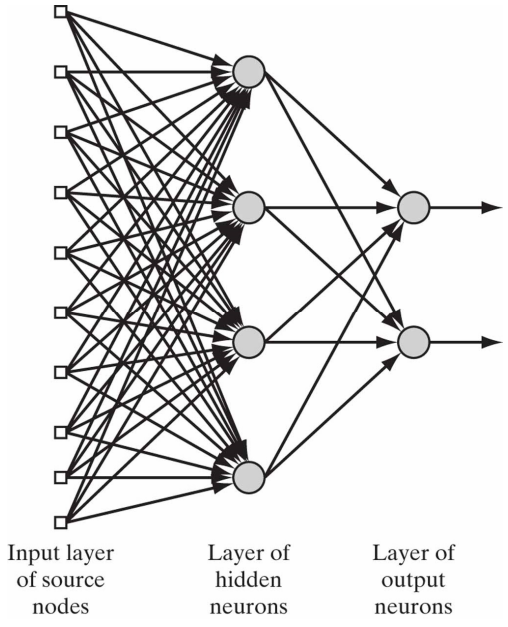
Feed-forward neural networks

Topología de red más habitual en la práctica

Sin realimentación, organizadas por capas:

- Capa de entrada
- Capa oculta
- Capa de salida

[Haykin: "Neural Networks and Learning Machines", 3rd edition]



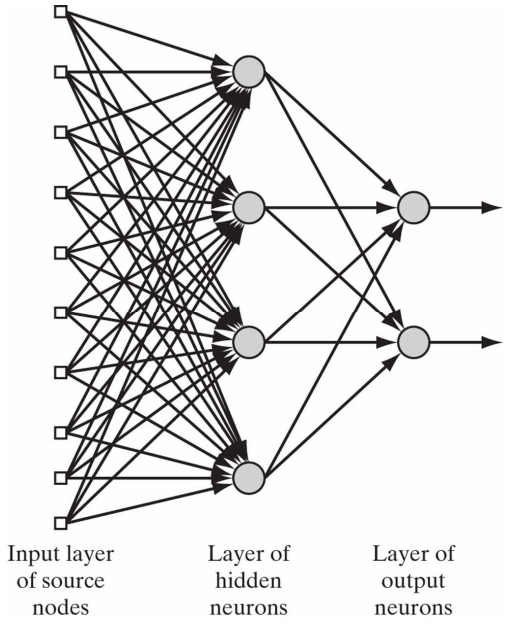
Redes neuronales multicapa

Feed-forward neural networks

Topología de red más habitual en la práctica

Si hay más de una capa oculta, se denominan **"deep" neural networks**.

[Haykin: "Neural Networks and Learning Machines", 3rd edition]



Redes neuronales multicapa

- Las redes neuronales sin neuronas ocultas (p.ej. perceptrones) están muy limitadas con respecto a lo que son capaces de aprender.
- El uso de capas de adicionales de neuronas lineales no ayuda (el resultado seguiría siendo lineal), por lo que necesitamos múltiples capas de unidades no lineales.

Redes neuronales multicapa

Problema

¿Cómo entrenamos estas redes multicapa?

- Necesitamos un algoritmo eficiente que nos permita adaptar todos los pesos de una red multicapa, no sólo los de la capa de salida.
- Aprender los pesos correspondientes a las neuronas de las capas ocultas equivale a aprender nuevas características (no presentes en el conjunto de entrenamiento), lo que resulta especialmente difícil porque nadie nos dice directamente qué es lo que deberíamos aprender en esas unidades ocultas.

Redes neuronales multicapa

Problema

¿Cómo entrenamos estas redes multicapa?

- El algoritmo utilizado por el perceptrón no puede extenderse para redes multicapa (garantiza su convergencia en una región convexa, pero no en un problema no convexo, en el que la media de dos buenas soluciones puede no ser buena).

- Las redes multicapa no usan el algoritmo de aprendizaje del perceptrón, por lo que no es correcto llamarlas perceptrones multicapa (aunque sí usual).

Redes neuronales multicapa

Problema

¿Cómo entrenamos estas redes multicapa?

- En vez de fijarnos en los cambios de los pesos, nos fijaremos en los cambios de las salidas, que intentaremos acercar a las salidas deseadas (estrategia válida para problemas no convexos).

Empezaremos por un ejemplo de juguete…

una neurona lineal

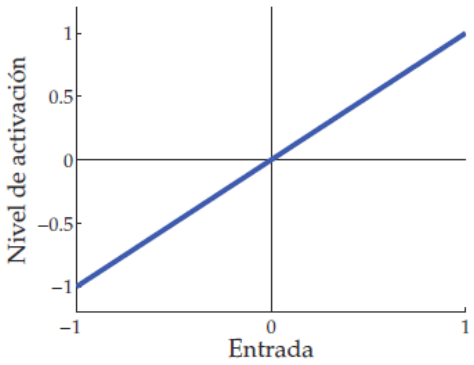
Backpropagation
Neurona lineal

Backpropagation

Neurona lineal

a.k.a. filtro lineal

$$y = \sum_i w_i x_i = \mathbf{w}^T \mathbf{x}$$



OBJETIVO:

Minimizar la suma de los errores sobre todos los ejemplos del conjunto de entrenamiento.

Podríamos resolver el problema analíticamente, pero la solución sería difícil de generalizar, por lo que diseñaremos un algoritmo iterativo (menos eficiente) que luego podamos generalizar…

Backpropagation

Neurona lineal

Todos los días tomamos café, zumo y tostadas en la cafetería, pero sólo al final de la semana nos pasan el total de la cuenta, sin darnos el precio individual de cada producto. Tras varias semanas, deberíamos ser capaces de adivinar esos precios individuales…

Algoritmo iterativo

- Empezar con una estimación (aleatoria) de los precios.
- Ajustar los precios estimados para que encajen con los importes facturados.

Backpropagation

Neurona lineal

Cada semana, el total nos impone una restricción lineal sobre los precios de las cantidades consumidas:

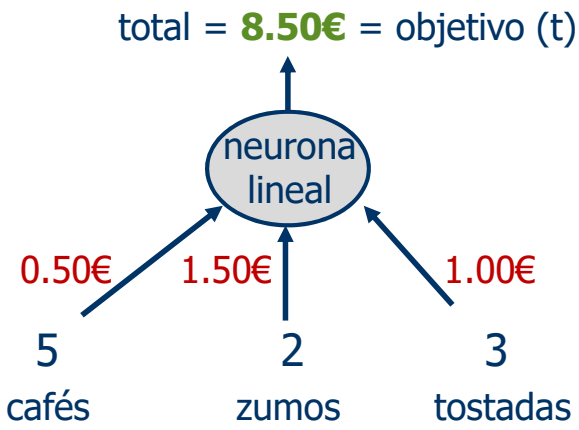
$$\text{total} = w_{\text{café}}x_{\text{café}} + w_{\text{zumo}}x_{\text{zumo}} + w_{\text{tostada}}x_{\text{tostada}}$$

- Los precios son los pesos $w = (w_{\text{café}}, w_{\text{zumo}}, w_{\text{tostada}})$.
- Las entradas corresponden a las cantidades consumidas de cada producto.

Backpropagation

Neurona lineal

Los precios reales, utilizados para pasarnos la factura:



Backpropagation

Neurona lineal

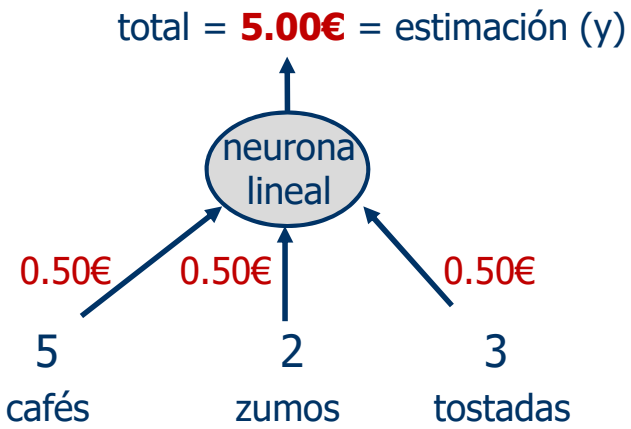
Precios estimados inicialmente (0.50€):

Error residual = 3.50€

Corrección de los pesos:

Regla delta

$$\Delta w_i = \eta \, x_i (t - y)$$



η [eta] es la tasa de aprendizaje utilizada

Backpropagation

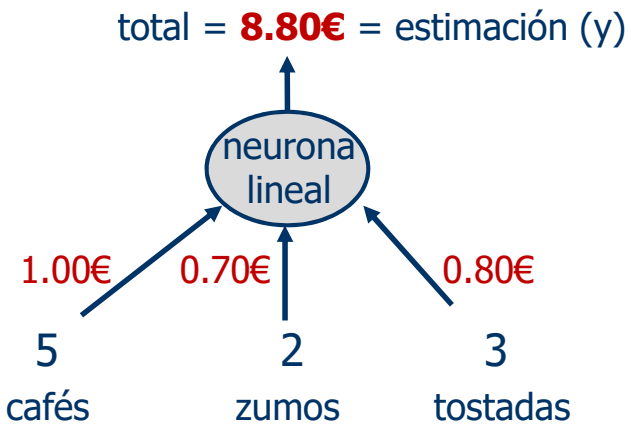
Neurona lineal

Precios ajustados usando η =1/35:

Error residual = 3.50€

Δw = (+50, +20, +30)

Nuevo error = -0.30€



NOTA:
Nuestra estimación del precio del café ha empeorado!!

Backpropagation

Neurona lineal

Derivación de la regla delta

- Error (residuos al cuadrado)

$$E = \frac{1}{2} \sum_{n \in training} (t^n - y^n)^2$$

- Derivada del error

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_n \frac{\partial y^n}{\partial w_i} \frac{dE^n}{dy^n} = -\sum_n x_i^n (t^n - y^n)$$

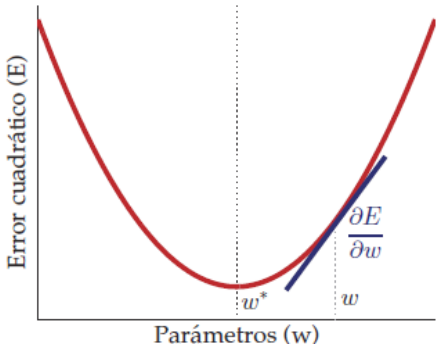
- Ajuste de los pesos en proporción al error

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_n x_i^n (t^n - y^n)$$

Backpropagation

Neurona lineal

Derivación de la regla delta



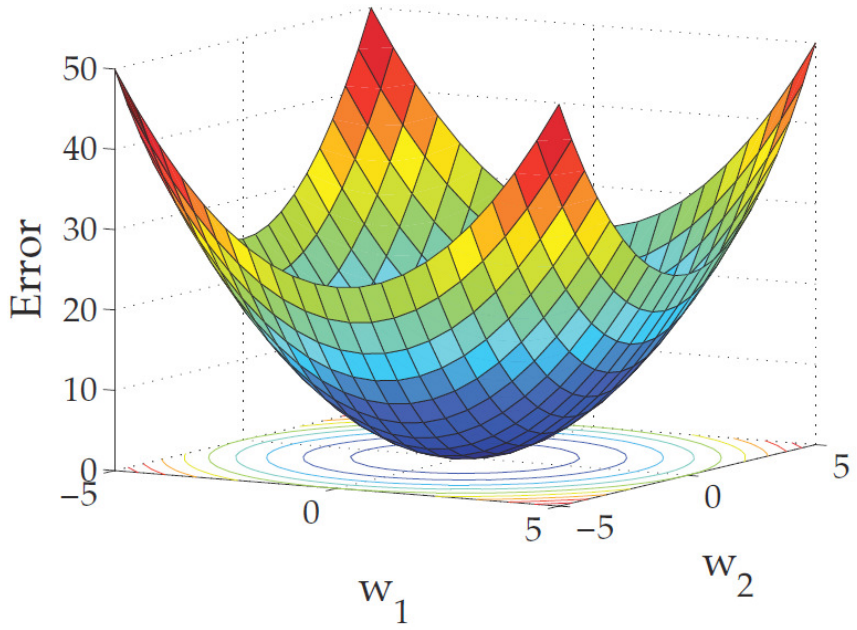
$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Ajuste de los pesos en proporción al error.

Backpropagation

Neurona lineal

Superficie de error



Backpropagation

Neurona lineal

Algoritmo de aprendizaje iterativo

- Tenemos que elegir un valor adecuado para un parámetro clave del algoritmo:
  la tasa de aprendizaje (η).
- Con una tasa de aprendizaje demasiado grande, corremos el riesgo de pasarnos del mínimo.
- Con una tasa de aprendizaje lo suficientemente pequeña, nos iremos acercando a la mejor solución posible.

Backpropagation

Neurona lineal

Algoritmo de aprendizaje iterativo

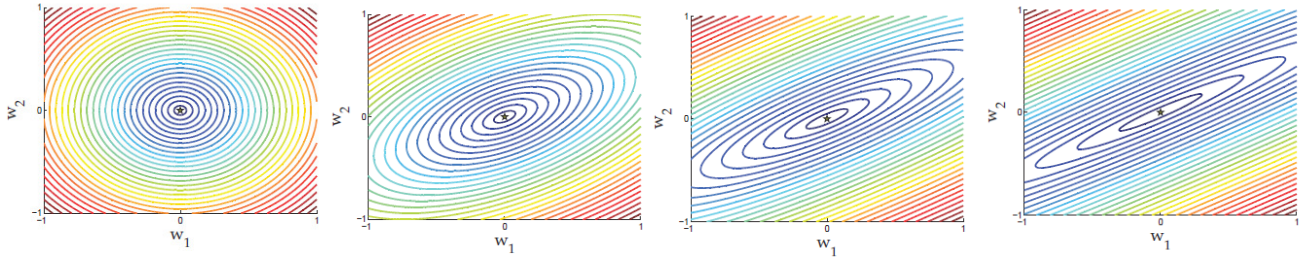
¡OJO!

El algoritmo puede ser muy lento si existe correlación entre las variables de entrada (si siempre pedimos zumo y tostadas, será difícil determinar cómo repartir el importe y determinar el precio de esos productos).

Backpropagation

Neurona lineal

Sección transversal de la superficie de error conforme aumenta la correlación entre las variables:



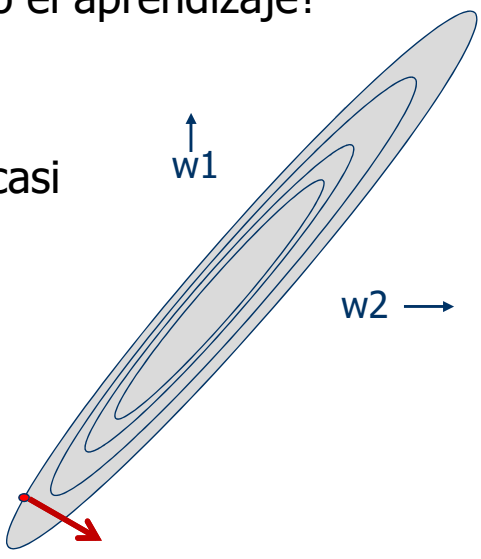
Cuanto mayor sea la correlación, más alargadas serán las elipses y menos útil resultará el gradiente (perpendicular a ellas) para guiarnos hacia el mínimo.

Backpropagation

Neurona lineal

¿Por qué puede ser muy lento el aprendizaje?

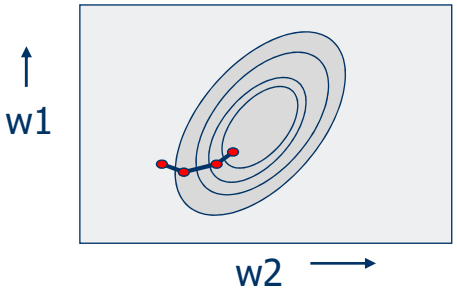
Si la elipse es muy alargada, la dirección del gradiente es casi perpendicular a la dirección hacia el mínimo...

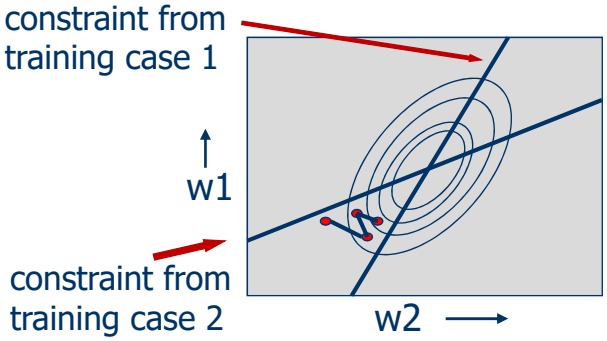


Backpropagation

Neurona lineal

Aprendizaje por lotes [batch learning] vs. online [stochastic]





Batch learning

Online learning

DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Backpropagation

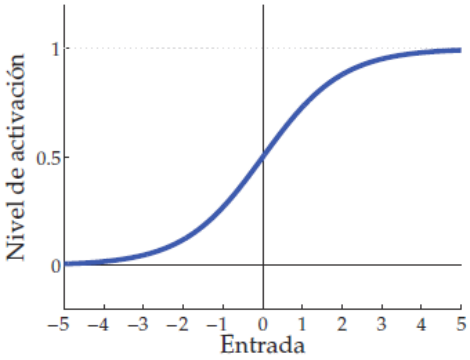
Neurona sigmoidal

Backpropagation

Neurona sigmoidal
Función logística

$$z = \sum_i x_i w_i$$

$$y = \frac{1}{1+e^{-z}}$$



Backpropagation

Neurona sigmoidal
Función logística

$$z = \sum_i x_i w_i \qquad y = \frac{1}{1+e^{-z}}$$

- Derivadas del logit (z) con respecto a pesos y entradas

$$\frac{\partial z}{\partial w_i} = x_i \qquad \frac{\partial z}{\partial x_i} = w_i$$

- Derivada de la salida (y) con respecto al logit (z)

$$\frac{dy}{dz} = y\,(1-y)$$

Backpropagation

Neurona sigmoidal

Función logística

$$y = \frac{1}{1+e^{-z}} = (1+e^{-z})^{-1}$$

$$\frac{dy}{dz} = \frac{-1(-e^{-z})}{(1+e^{-z})^2} = \left(\frac{1}{1+e^{-z}}\right)\left(\frac{e^{-z}}{1+e^{-z}}\right) = y(1-y)$$

$$\frac{e^{-z}}{1+e^{-z}} = \frac{(1+e^{-z})-1}{1+e^{-z}} = \frac{(1+e^{-z})}{1+e^{-z}} + \frac{-1}{1+e^{-z}} = 1-y$$

Backpropagation

Neurona sigmoidal

Función logística

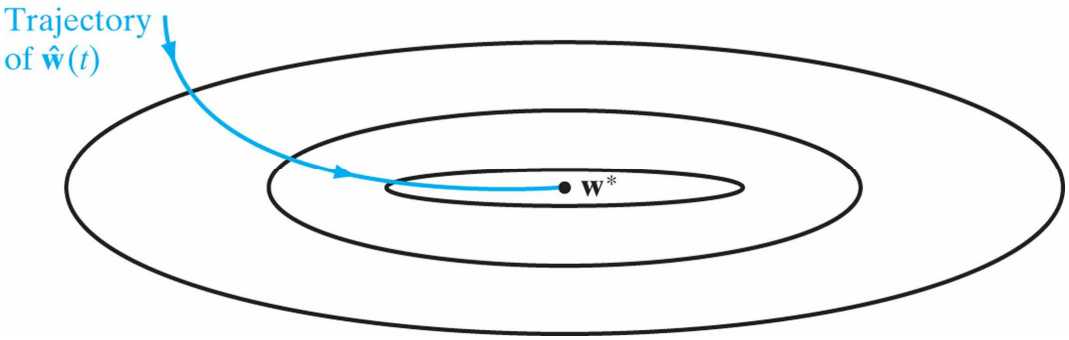
$$\frac{\partial y}{\partial w_i} = \frac{\partial z}{\partial w_i}\frac{dy}{dz} = x_i y(1-y)$$

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y^n}{\partial w_i}\frac{\partial E}{\partial y^n} = -\sum_n x_i^n y^n(1-y^n)(t^n - y^n)$$

Regla delta

Término extra (pendiente de la función logística)

Backpropagation



[Haykin: "Neural Networks and Learning Machines", 3rd edition]

DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Backpropagation
Neuronas ocultas

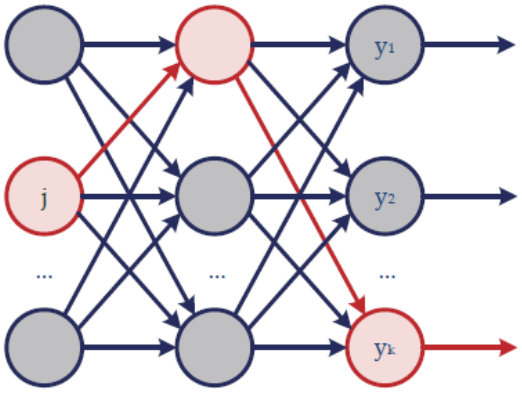
Backpropagation

Recapitulemos:

- Las redes neuronales sin unidades ocultas tienen muchas limitaciones, pero añadir características manualmente (como en el perceptrón) es tedioso.
- Nos gustaría ser capaces de encontrar características automáticamente, sin necesidad de conocer cada problema al detalle ni recurrir a un proceso de prueba y error…

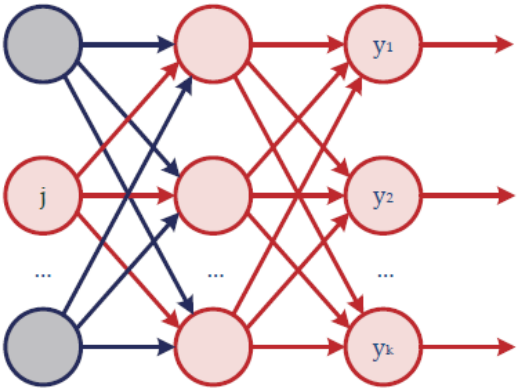
Backpropagation

Las neuronas ocultas afectan a la salida de la red…



Backpropagation

... por múltiples caminos



Backpropagation

IDEA

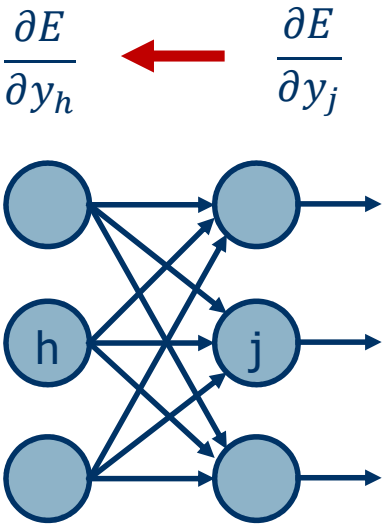
No sabemos qué deben hacer las neuronas ocultas, pero podemos calcular cómo cambia el error cuando cambia su actividad.

- En vez de utilizar la salida deseada para entrenar las neuronas ocultas, usamos la derivada del error con respecto a sus actividades ($\delta E/\delta y$).
- La actividad de cada neurona oculta puede tener efectos en muchas neuronas de salida, por lo que debemos combinarlos.
- Una vez que tenemos las derivadas del error para todas las unidades ocultas, podemos calcular las derivadas del error para sus pesos de entrada.

Backpropagation

Propagación de errores

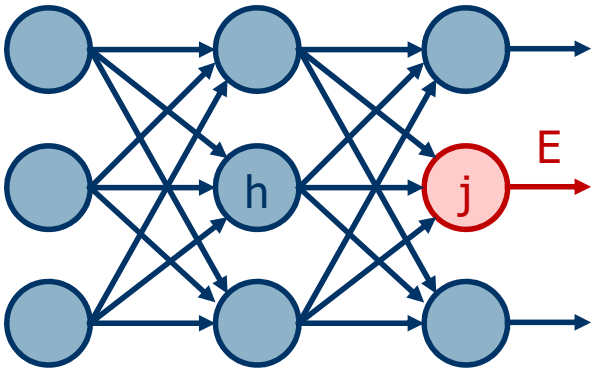
- Convertimos la discrepancia entre la salida de la red y su salida deseada en una derivada del error (∂E/∂y).
- Calculamos las derivadas del error de cada capa oculta a partir de las derivadas del error de la capa superior.
- Usamos ∂E/∂y para obtener ∂E/∂w.



Backpropagation

Propagación de errores ∂E/∂y

Conocemos el error E en la capa de salida y cómo calcular su gradiente para las neuronas de salida j.



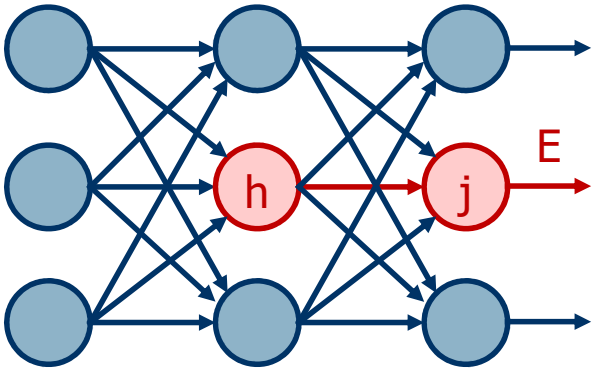
Podemos calcular cómo varía el error en función de la entrada neta de las neuronas (sigmoidales) de salida z_j:

$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j}\frac{\partial E}{\partial y_j} = f'(z_j)\frac{\partial E}{\partial y_j} = y_j(1-y_j)\frac{\partial E}{\partial y_j}$$

Backpropagation

Propagación de errores ∂E/∂y

A continuación, podemos calcular cómo fluctúa el error en función de la salida y_h de la neurona oculta (h):

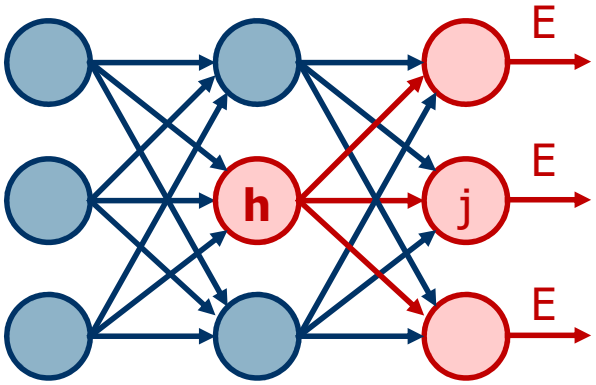


$$\frac{\partial E}{\partial y_h} = \frac{\partial z_j}{\partial y_h}\frac{\partial E}{\partial z_j} = w_{hj}\frac{\partial E}{\partial z_j}$$

Backpropagation

Propagación de errores ∂E/∂y

Pero tenemos que tener en cuenta que la neurona oculta influye en el error observado en todas las neuronas de la capa siguiente:

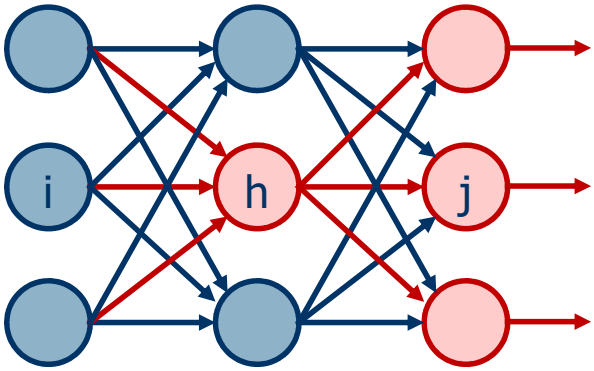


$$\frac{\partial E}{\partial y_h} = \sum_j \frac{\partial z_j}{\partial y_h}\frac{\partial E}{\partial z_j} = \sum_j w_{hj}\frac{\partial E}{\partial z_j}$$

Backpropagation

Propagación de errores ∂E/∂y

Por último, ya estamos en condiciones de calcular cómo fluctúa el error en función de los parámetros de la neurona oculta (h), los pesos w_{ij}:



$$\frac{\partial E}{\partial w_{ih}} = \frac{\partial z_h}{\partial w_{ih}} \frac{\partial E}{\partial z_h} = x_{ih} \frac{\partial E}{\partial z_h} = y_i \frac{\partial E}{\partial z_h}$$

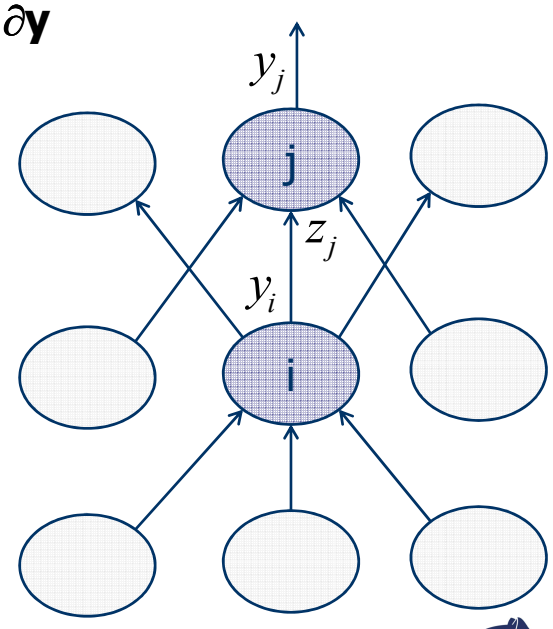
Backpropagation

Propagación de errores ∂E/∂y

$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j(1-y_j) \frac{\partial E}{\partial y_j}$$

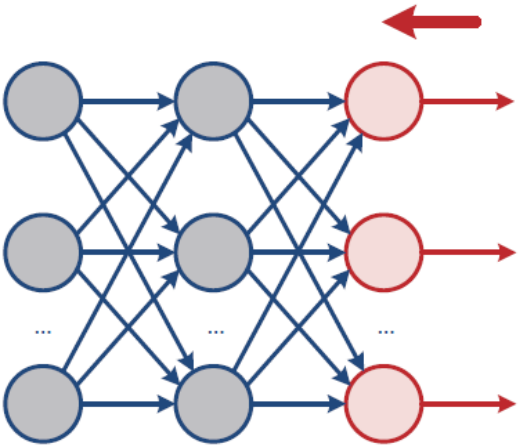
$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$



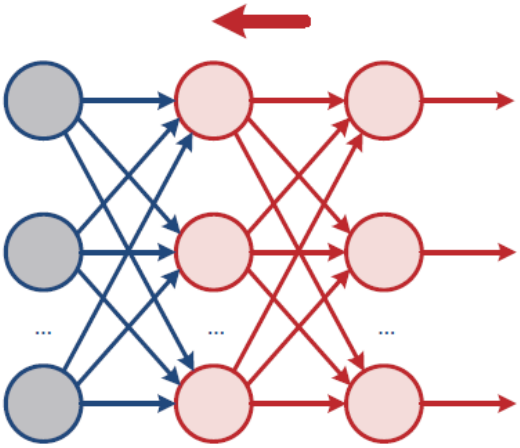
Backpropagation

Propagación de errores $\delta E/\delta y$



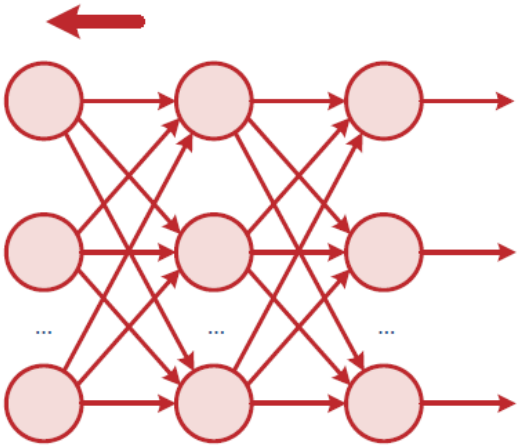
Backpropagation

Propagación de errores $\delta E/\delta y$



Backpropagation

Propagación de errores $\delta E/\delta y$



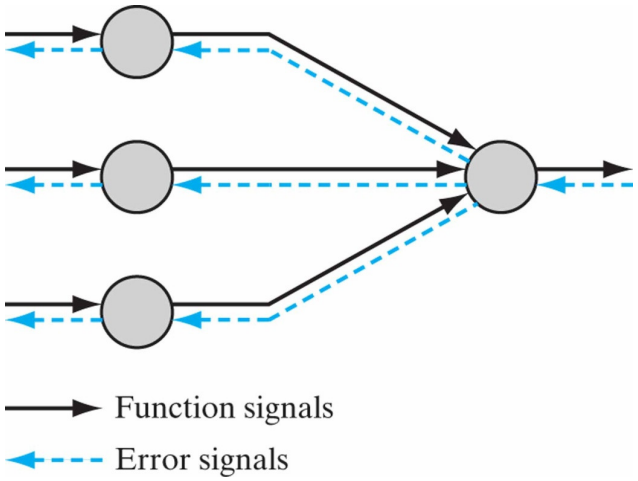
DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Backpropagation

Algoritmo de propagación de errores

Backpropagation

El algoritmo de propagación de errores nos proporciona un método eficiente para calcular las derivadas del error $\delta E/\delta w$ para cada peso.



[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Backpropagation

Algoritmo de propagación de errores

Dada la derivada del error con respecto a la salida

$$\frac{\partial E}{\partial y_j^c}$$

incorporamos la función de activación en el cálculo de los "deltas" que utilizaremos para obtener el gradiente del error:

$$\delta_j^c = \frac{\partial E}{\partial z_j^c} = \frac{\partial E}{\partial y_j^c}\frac{dy_j^c}{dz_j^c} = \frac{\partial E}{\partial y_j^c} f'(z_j^c)$$

Backpropagation

Algoritmo de propagación de errores

Calculamos todos los "deltas" recorriendo la red hacia atrás (desde la capa de salida hasta la capa de entrada):

$$\delta_j^k = \begin{cases} f'(z_j^k)\,(y_j - t_j) & \text{en las neuronas de salida,} \\ f'(z_j^k)\left(\sum_p \delta_p^{k+1} w_{pj}^{k+1}\right) & \text{en las neuronas ocultas.} \end{cases}$$

Backpropagation

Algoritmo de propagación de errores

Actualizamos los pesos de la red de acuerdo al gradiente:

$$\Delta w_{ij}^k = -\eta x_i \delta_j^k$$

Backpropagation

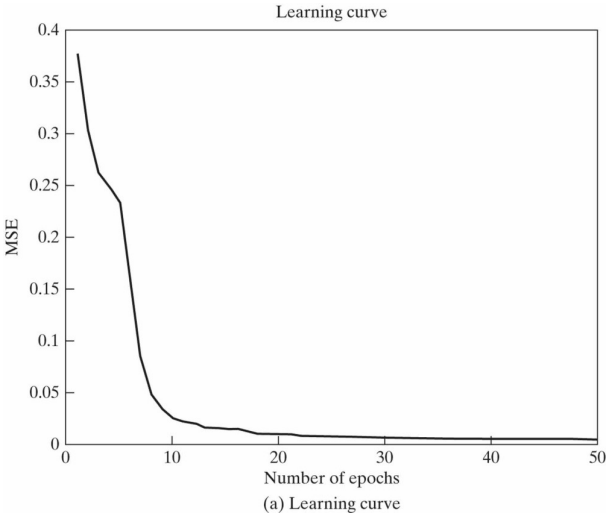
Algoritmo de propagación de errores

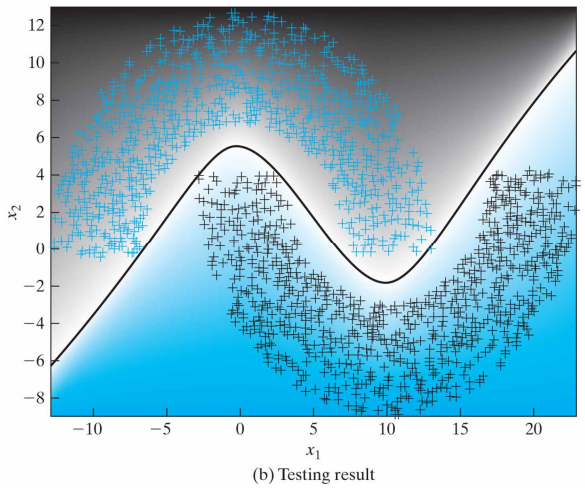
Actualizamos los pesos de la red de acuerdo al gradiente:

$$\Delta w_{ij}^k = \begin{cases} -\eta x_i^k f'(z_j^k)\,(y_j - t_j) & \text{para las neuronas de salida,} \\ -\eta x_i^k f'(z_j^k) \left(\sum_p \delta_p^{k+1} w_{pj}^{k+1} \right) & \text{para las neuronas ocultas.} \end{cases}$$

Backpropagation

Ejemplo

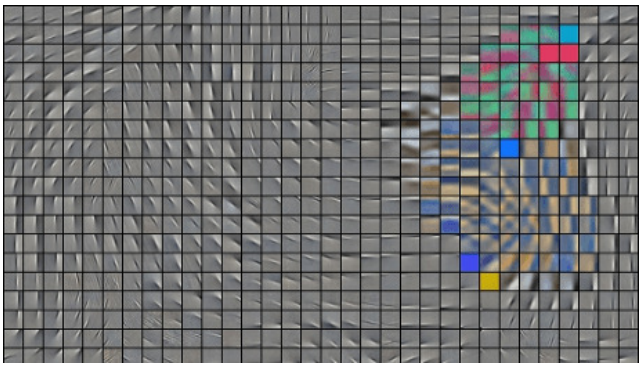




[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Cursos

Neural Networks for Machine Learning
by Geoffrey Hinton
(University of Toronto & Google)
https://www.coursera.org/course/neuralnets



UNIVERSITY OF TORONTO

Cursos

Deep Learning Specialization
by Andrew Ng, 2017

- Neural Networks and Deep Learning
- Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
- Structuring Machine Learning Projects
- Convolutional Neural Networks
- Sequence Models

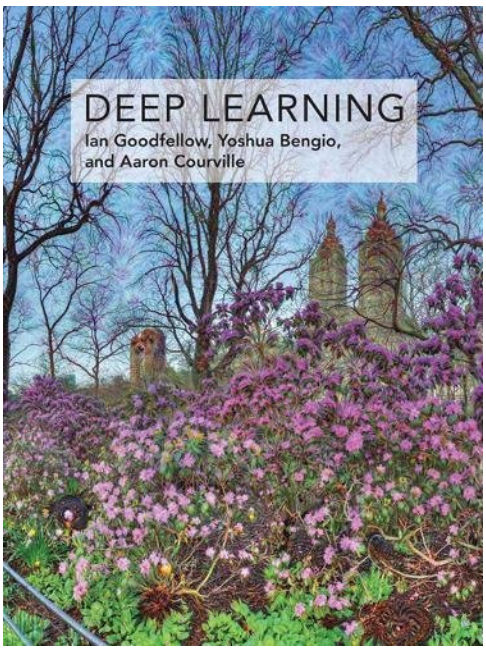
deeplearning.ai

https://www.coursera.org/specializations/deep-learning

Bibliografía

Lecturas recomendadas

Ian Goodfellow,
Yoshua Bengio
& Aaron Courville:
Deep Learning
MIT Press, 2016
ISBN 0262035618



http://www.deeplearningbook.org

Bibliografía

Lecturas recomendadas

Fernando Berzal:
Redes Neuronales
& Deep Learning

CAPÍTULO 8
Backpropagation