

Redes Neuronales

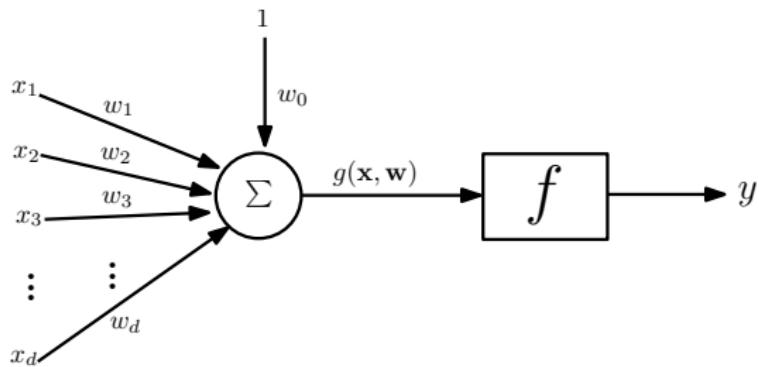
Fernando Lozano

Universidad de los Andes

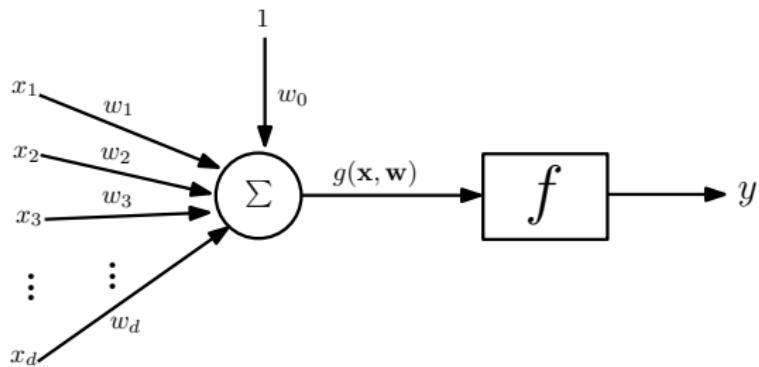
14 de septiembre de 2022



Modelo de una neurona

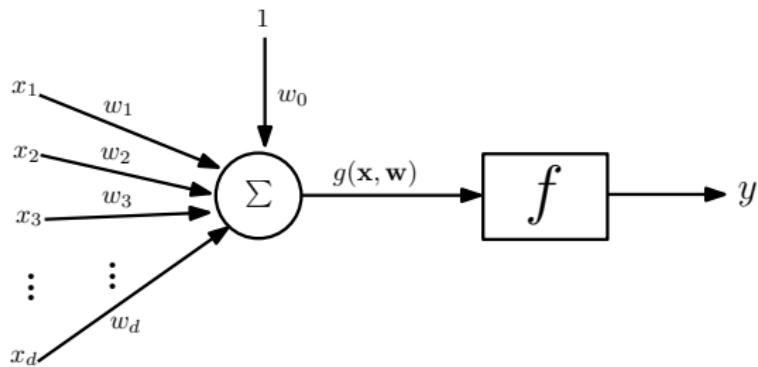


Modelo de una neurona



$$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]^T$$

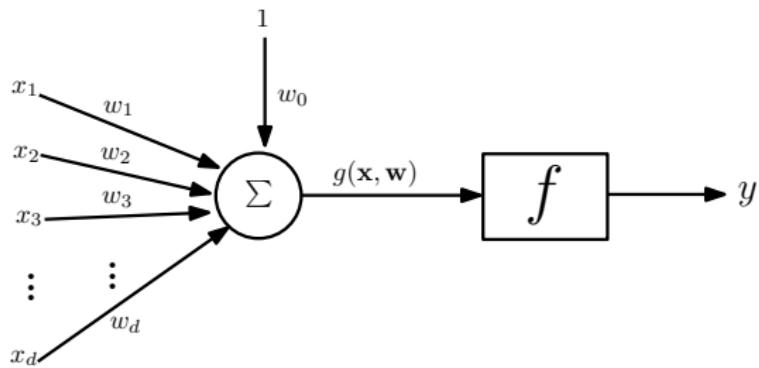
Modelo de una neurona



$$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]^T$$

$$\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]^T$$

Modelo de una neurona

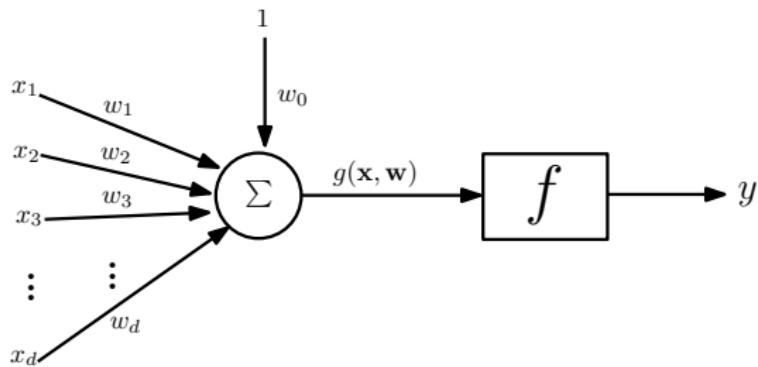


$$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]^T$$

$$\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]^T$$

$$g(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} + w_0$$

Modelo de una neurona



$$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]^T$$

$$\mathbf{w} = [w_1 \ w_2 \ \dots \ w_d]^T$$

$$g(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} + w_0$$

$$y = f(g(\mathbf{x})) = f(\mathbf{w}^T \mathbf{x} + w_0)$$

Vectores extendidos

$$\tilde{\mathbf{x}} = [1 \ x_1 \ x_2 \ \dots \ x_d]^T$$

Vectores extendidos

$$\begin{aligned}\tilde{\mathbf{x}} &= [1 \quad x_1 \quad x_2 \quad \dots \quad x_d]^T \\ \tilde{\mathbf{w}} &= [w_0 \quad w_1 \quad w_2 \quad \dots \quad w_d]^T\end{aligned}$$

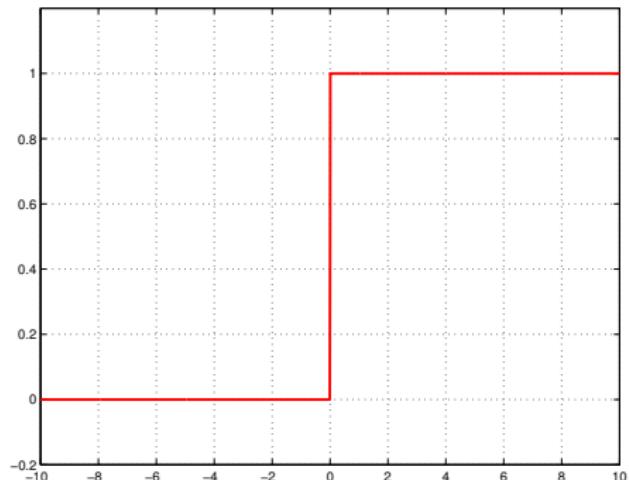
Vectores extendidos

$$\begin{aligned}\tilde{\mathbf{x}} &= [1 \quad x_1 \quad x_2 \quad \dots \quad x_d]^T \\ \tilde{\mathbf{w}} &= [w_0 \quad w_1 \quad w_2 \quad \dots \quad w_d]^T \\ g(\mathbf{x}) &= \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}\end{aligned}$$

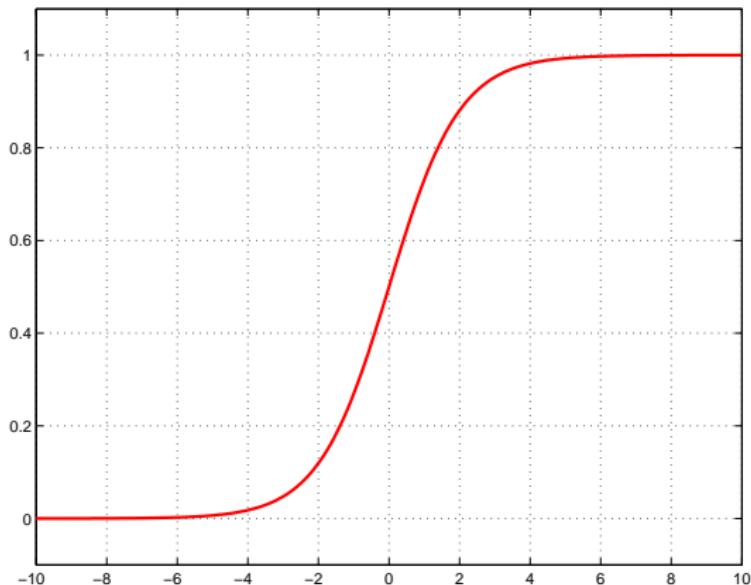
Vectores extendidos

$$\begin{aligned}\tilde{\mathbf{x}} &= [1 \quad x_1 \quad x_2 \quad \dots \quad x_d]^T \\ \tilde{\mathbf{w}} &= [w_0 \quad w_1 \quad w_2 \quad \dots \quad w_d]^T \\ g(\mathbf{x}) &= \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} \\ u(\mathbf{x}) &= f(g(\mathbf{x})) = f(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}})\end{aligned}$$

Umbral (limitador duro)

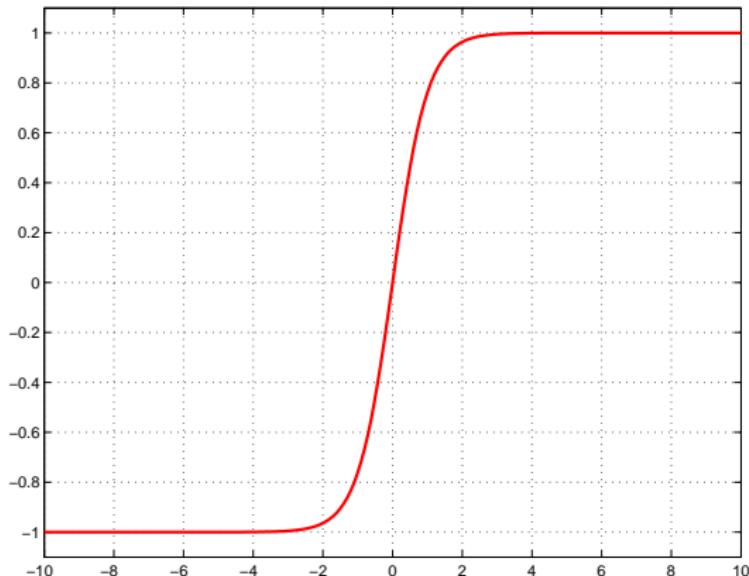


Activación sigmoidal



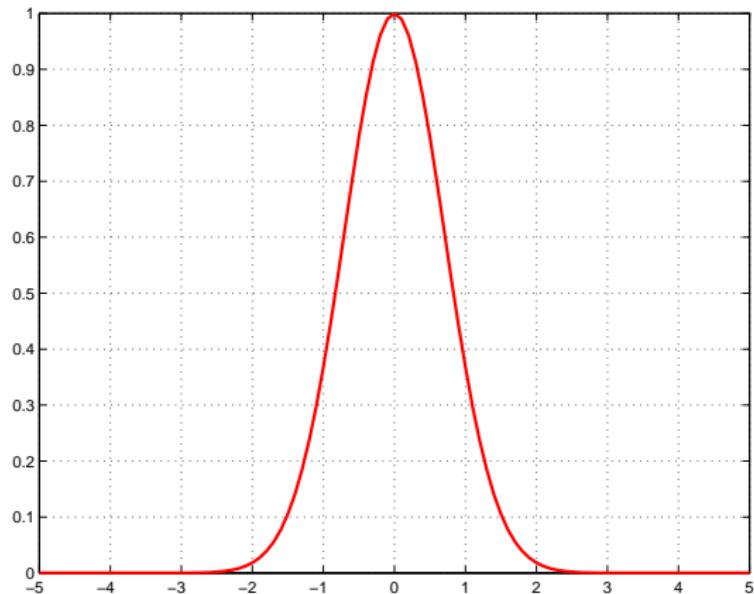
$$f_s(z) = \frac{1}{1 + e^{-\beta z}}$$

Tangente hiperbólica



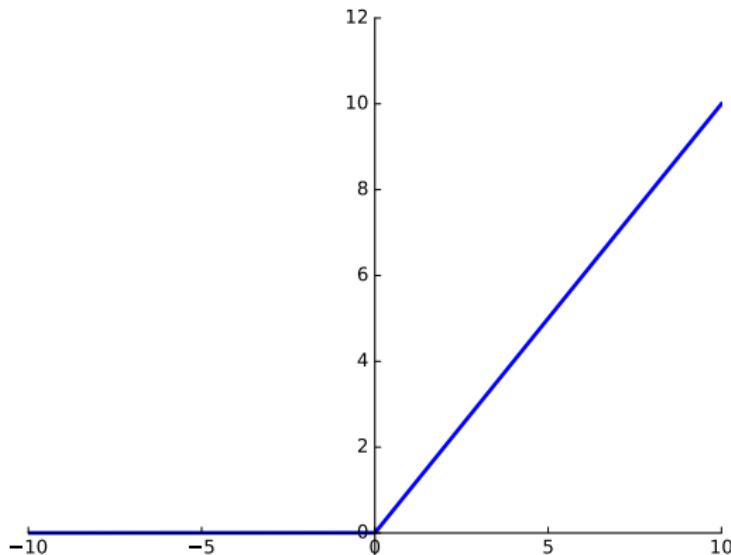
$$f_{TH}(z) = \tanh(z)$$

Función de base radial



$$f_{RBF}(z) = e^{-z^2}$$

ReLU



$$r(z) = \max \{0, z\}$$

Aplicaciones

Aplicaciones

- Clasificación binaria:

Aplicaciones

- Clasificación binaria:
 - ▶ Limitador duro: f_{LD} es etiqueta.

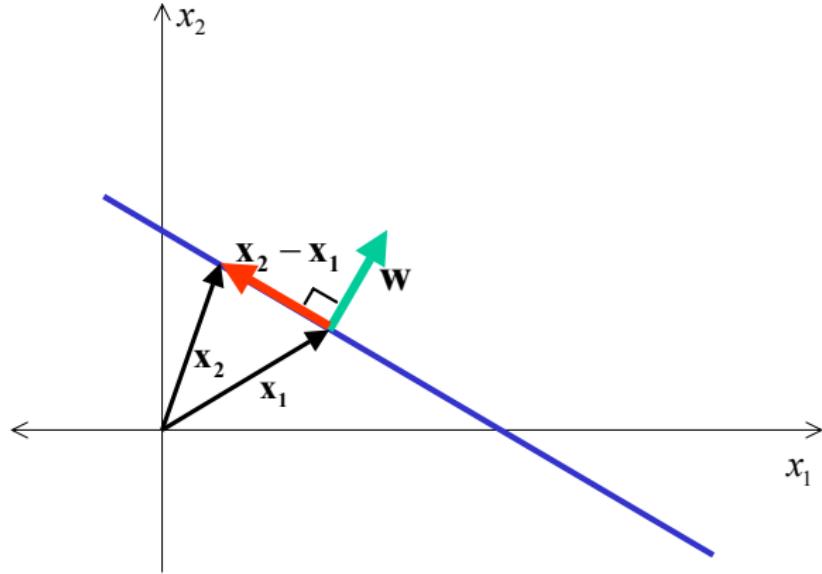
Aplicaciones

- Clasificación binaria:
 - ▶ Limitador duro: f_{LD} es etiqueta.
 - ▶ Función logística f_s es probabilidad de pertenecer a clase 1.

Aplicaciones

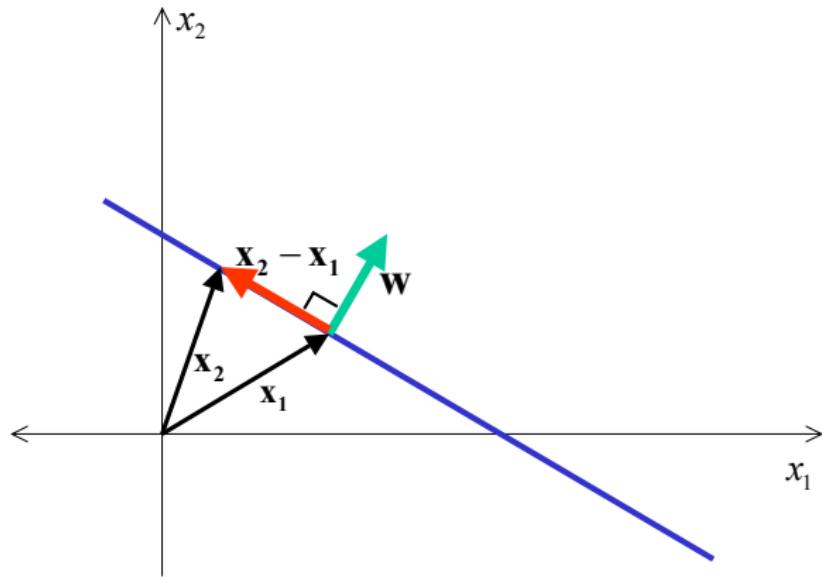
- Clasificación binaria:
 - ▶ Limitador duro: f_{LDE} es etiqueta.
 - ▶ Función logística f_s es probabilidad de pertenecer a clase 1.
- Regresión: Sigmoidal, RBF.

Interpretación geométrica 1



$$C = \{\mathbf{x} : \mathbf{w}^T \mathbf{x} + w_0 = 0\}$$

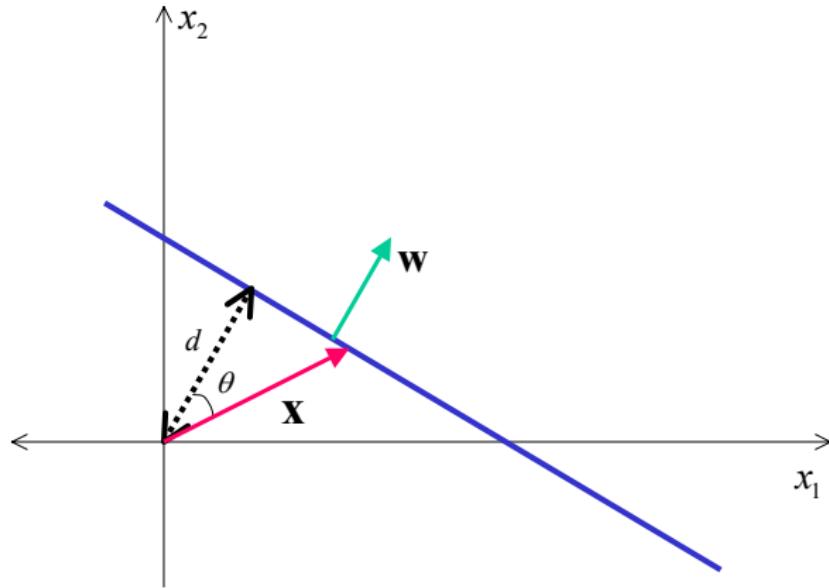
Interpretación geométrica 1



$$C = \{\mathbf{x} : \mathbf{w}^T \mathbf{x} + w_0 = 0\}$$

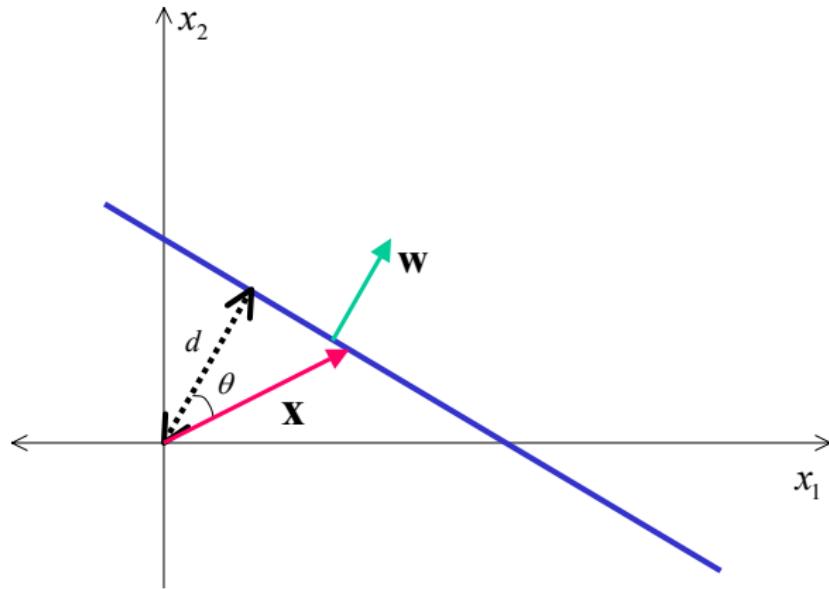
$$\mathbf{x}_1, \mathbf{x}_2 \in C \Rightarrow \mathbf{w}^T (\mathbf{x}_2 - \mathbf{x}_1) = 0$$

Interpretación geométrica 2



$$\mathbf{x} \in C \Rightarrow \mathbf{w}^T \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos(\theta)$$

Interpretación geométrica 2



$$\begin{aligned}\mathbf{x} \in C \Rightarrow \mathbf{w}^T \mathbf{x} &= \|\mathbf{w}\| \|\mathbf{x}\| \cos(\theta) \\ &= \|\mathbf{w}\| d = -w_0 \Rightarrow d = \frac{-w_0}{\|\mathbf{w}\|}\end{aligned}$$

Cuándo puede ser este modelo óptimo?

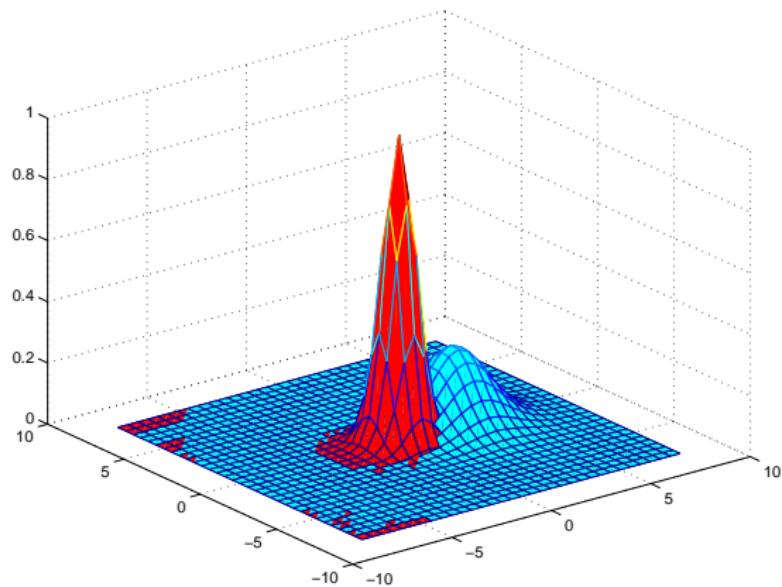
Cuándo puede ser este modelo óptimo?

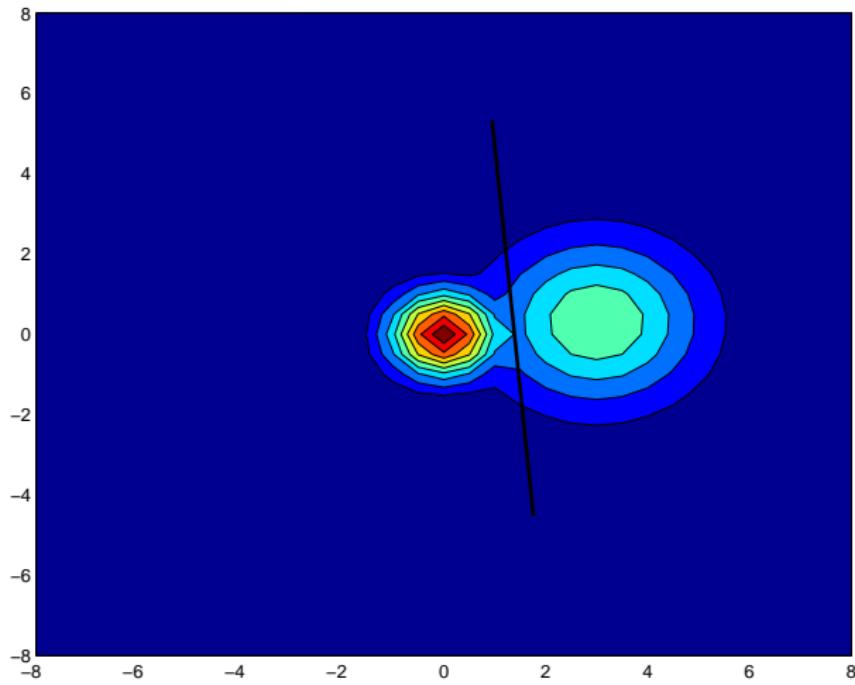
- Cuando $p_0(\mathbf{x}) \sim N(\mathbf{m}_0, \Sigma)$ y $p_1(\mathbf{x}) \sim N(\mathbf{m}_1, \Sigma)$ el clasificador óptimo de Bayes asigna \mathbf{x} a la clase 1 si:

Cuándo puede ser este modelo óptimo?

- Cuando $p_0(\mathbf{x}) \sim N(\mathbf{m}_0, \Sigma)$ y $p_1(\mathbf{x}) \sim N(\mathbf{m}_1, \Sigma)$ el clasificador óptimo de Bayes asigna \mathbf{x} a la clase 1 si:

$$\underbrace{(\mathbf{m}_1 - \mathbf{m}_0)^T \Sigma^{-1} \mathbf{x}}_{\mathbf{w}^T} > \underbrace{2 \ln \left(\frac{1-\alpha}{\alpha} \right) + \frac{1}{2} (\mathbf{m}_1^T \Sigma^{-1} \mathbf{m}_1 - \mathbf{m}_0^T \Sigma^{-1} \mathbf{m}_0)}_{-w_0}$$





Algoritmos de entrenamiento

Algoritmos de entrenamiento

- Regresión logística.

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS
 - ▶ Widrow y Hoff (1960)

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS
 - ▶ Widrow y Hoff (1960)
 - ▶ Adaptive linear networks (ADALINE).

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS
 - ▶ Widrow y Hoff (1960)
 - ▶ Adaptive linear networks (ADALINE).
- Algoritmo del Perceptrón

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS
 - ▶ Widrow y Hoff (1960)
 - ▶ Adaptive linear networks (ADALINE).
- Algoritmo del Perceptrón
 - ▶ Rosembatt (1962).

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS
 - ▶ Widrow y Hoff (1960)
 - ▶ Adaptive linear networks (ADALINE).
- Algoritmo del Perceptrón
 - ▶ Rosembatt (1962).
 - ▶ Prueba de convergencia.

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS
 - ▶ Widrow y Hoff (1960)
 - ▶ Adaptive linear networks (ADALINE).
- Algoritmo del Perceptrón
 - ▶ Rosembatt (1962).
 - ▶ Prueba de convergencia.
- Perceptrón con bolsillo

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS
 - ▶ Widrow y Hoff (1960)
 - ▶ Adaptive linear networks (ADALINE).
- Algoritmo del Perceptrón
 - ▶ Rosembatt (1962).
 - ▶ Prueba de convergencia.
- Perceptrón con bolsillo
 - ▶ Gallant (1986)

Algoritmos de entrenamiento

- Regresión logística.
- Algoritmo LMS
 - ▶ Widrow y Hoff (1960)
 - ▶ Adaptive linear networks (ADALINE).
- Algoritmo del Perceptrón
 - ▶ Rosembatt (1962).
 - ▶ Prueba de convergencia.
- Perceptrón con bolsillo
 - ▶ Gallant (1986)
 - ▶ Para datos no linealmente separables.

El perceptrón

- Conjunto de datos:

$$\{\mathbf{x}_i, y_i\}, \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

El perceptrón

- Conjunto de datos:

$$\{\mathbf{x}_i, y_i\}, \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

- (\mathbf{x}_i, y_i) es clasificado correctamente si:

$$g(\mathbf{w}, \mathbf{x}_i)y_i > 0$$

El perceptrón

- Conjunto de datos:

$$\{\mathbf{x}_i, y_i\}, \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

- (\mathbf{x}_i, y_i) es clasificado correctamente si:

$$\begin{aligned} g(\mathbf{w}, \mathbf{x}_i) y_i &> 0 \\ (\mathbf{w}^T \mathbf{x}_i) y_i &> 0 \end{aligned}$$

El perceptrón

- Conjunto de datos:

$$\{\mathbf{x}_i, y_i\}, \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

- (\mathbf{x}_i, y_i) es clasificado correctamente si:

$$g(\mathbf{w}, \mathbf{x}_i)y_i > 0$$

$$(\mathbf{w}^T \mathbf{x}_i)y_i > 0$$

- Criterio de error del perceptrón:

El perceptrón

- Conjunto de datos:

$$\{\mathbf{x}_i, y_i\}, \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

- (\mathbf{x}_i, y_i) es clasificado correctamente si:

$$\begin{aligned} g(\mathbf{w}, \mathbf{x}_i)y_i &> 0 \\ (\mathbf{w}^T \mathbf{x}_i)y_i &> 0 \end{aligned}$$

- Criterio de error del perceptrón:

$$E(\mathbf{w}) = - \sum_{\mathbf{x}_i \in \mathbf{M}} (\mathbf{w}^T \mathbf{x}_i)y_i, \quad \mathbf{M} = \{\mathbf{x}_i : (\mathbf{w}^T \mathbf{x}_i)y_i < 0\}$$

Criterio de error del perceptrón

- $E(\mathbf{w})$ es una función lineal a trozos.

Criterio de error del perceptrón

- $E(\mathbf{w})$ es una función lineal a trozos.
- $E(\mathbf{w})$ es una función continua.

Criterio de error del perceptrón

- $E(\mathbf{w})$ es una función lineal a trozos.
- $E(\mathbf{w})$ es una función continua.
- $\nabla_{\mathbf{w}} E$ es una función discontinua.

Criterio de error del perceptrón

- $E(\mathbf{w})$ es una función lineal a trozos.
- $E(\mathbf{w})$ es una función continua.
- $\nabla_{\mathbf{w}} E$ es una función discontinua.
- Sin embargo, en los puntos donde es continua, $-\nabla_{\mathbf{w}} E$ es una dirección de descenso en la superficie de error.

Algoritmo del perceptrón

- Procedimiento iterativo:

Algoritmo del perceptrón

- Procedimiento iterativo:

- ➊ Comenzar en:

$$\mathbf{w}_0 = 0$$

Algoritmo del perceptrón

- Procedimiento iterativo:

- ① Comenzar en:

$$\mathbf{w}_0 = 0$$

- ② Búsqueda de gradiente: Ir “hacia abajo de la colina”.

Algoritmo del perceptrón

- Procedimiento iterativo:

① Comenzar en:

$$\mathbf{w}_0 = 0$$

② Búsqueda de gradiente: Ir “hacia abajo de la colina”.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \nabla_{\mathbf{w}} E|_{\mathbf{w}_k}$$

Algoritmo del perceptrón

- Procedimiento iterativo:

- ① Comenzar en:

$$\mathbf{w}_0 = 0$$

- ② Búsqueda de gradiente: Ir “hacia abajo de la colina”.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \nabla_{\mathbf{w}} E|_{\mathbf{w}_k}$$

- Nuevamente, $\nabla_{\mathbf{w}} E|_{\mathbf{w}_k}$ no se calcula exactamente:

Algoritmo del perceptrón

- Procedimiento iterativo:

① Comenzar en:

$$\mathbf{w}_0 = 0$$

② Búsqueda de gradiente: Ir “hacia abajo de la colina”.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \nabla_{\mathbf{w}} E|_{\mathbf{w}_k}$$

- Nuevamente, $\nabla_{\mathbf{w}} E|_{\mathbf{w}_k}$ no se calcula exactamente:

$$\nabla_{\mathbf{w}} E = - \sum_{\mathbf{x}_i \in \mathcal{M}} \mathbf{x}_i y_i$$

Algoritmo del perceptrón

- Procedimiento iterativo:

① Comenzar en:

$$\mathbf{w}_0 = 0$$

② Búsqueda de gradiente: Ir “hacia abajo de la colina”.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \nabla_{\mathbf{w}} E|_{\mathbf{w}_k}$$

- Nuevamente, $\nabla_{\mathbf{w}} E|_{\mathbf{w}_k}$ no se calcula exactamente:

$$\begin{aligned}\nabla_{\mathbf{w}} E &= - \sum_{\mathbf{x}_i \in \mathcal{M}} \mathbf{x}_i y_i \\ &\approx -\mathbf{x}_i y_i\end{aligned}$$

Algoritmo del perceptrón

Incialize $\mathbf{w}_0 = 0$

Algoritmo del perceptrón

Incialize $\mathbf{w}_0 = 0$

repeat

Escoja (\mathbf{x}_i, y_i) al azar

Algoritmo del perceptrón

Incialize $\mathbf{w}_0 = 0$

repeat

Escoja (\mathbf{x}_i, y_i) al azar

if $(\mathbf{w}^T \mathbf{x}_i)y_i \leq 0$ **then**

Algoritmo del perceptrón

Incialize $\mathbf{w}_0 = 0$

repeat

Escoja (\mathbf{x}_i, y_i) al azar

if $(\mathbf{w}^T \mathbf{x}_i)y_i \leq 0$ **then**

$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{x}_i y_i$

Algoritmo del perceptrón

Incialize $\mathbf{w}_0 = 0$

repeat

Escoja (\mathbf{x}_i, y_i) al azar

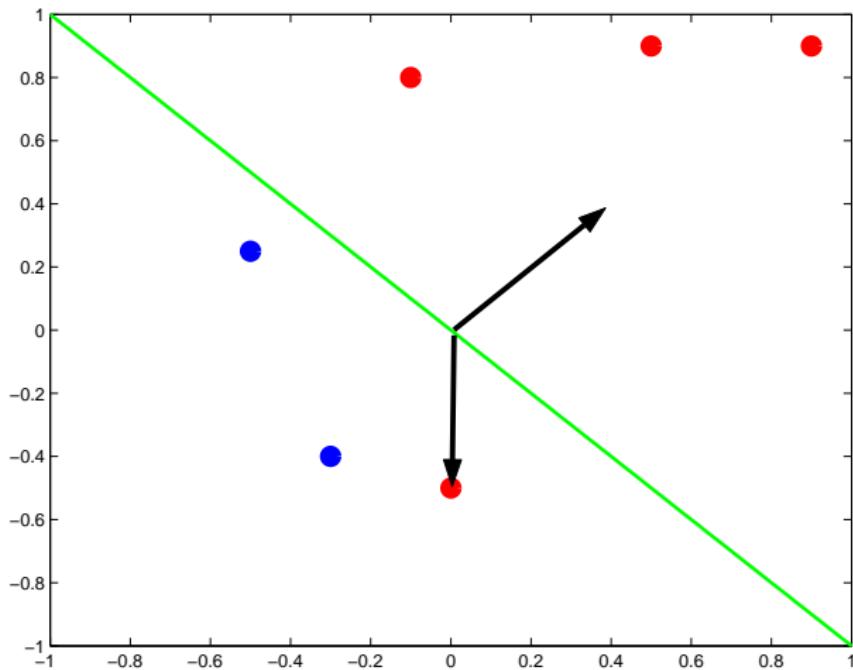
if $(\mathbf{w}^T \mathbf{x}_i)y_i \leq 0$ **then**

$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{x}_i y_i$

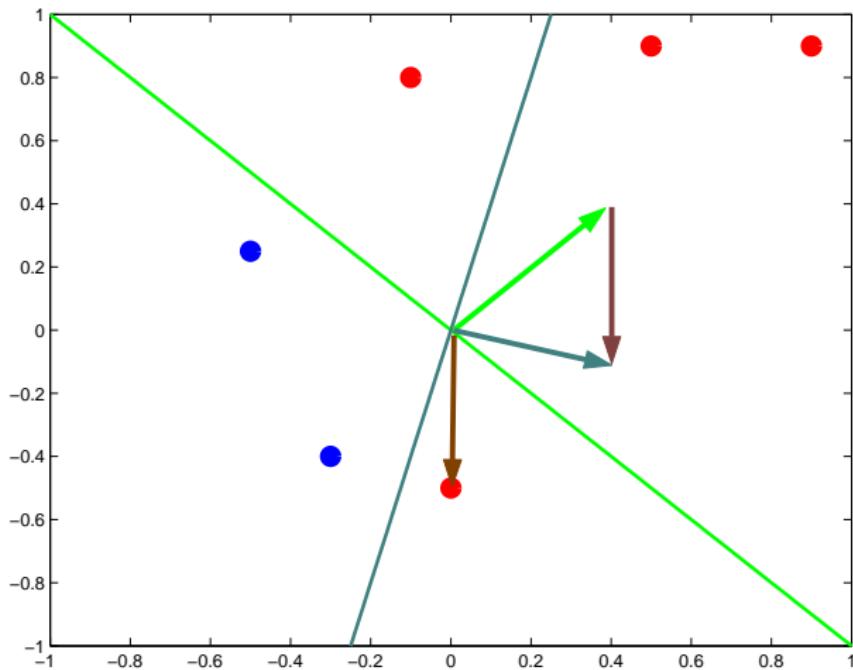
end if

until Convergencia.

Interpretación geométrica



Interpretación geométrica



Convergencia

- S.P.D.G Cambie los \mathbf{x}_i para los cuales $\hat{\mathbf{w}}^T \mathbf{x}_i < 0$ por $-\mathbf{x}_i$. Con este cambio $\mathbf{w}^T \mathbf{x}_i > 0$ indica clasificación correcta.

Convergencia

- S.P.D.G Cambie los \mathbf{x}_i para los cuales $\hat{\mathbf{w}}^T \mathbf{x}_i < 0$ por $-\mathbf{x}_i$. Con este cambio $\mathbf{w}^T \mathbf{x}_i > 0$ indica clasificación correcta.

Teorema

Suponga:

- $\|\mathbf{x}_i\| \leq K \in \mathbb{R}, \quad i = 1 \dots, n.$

Convergencia

- S.P.D.G Cambie los \mathbf{x}_i para los cuales $\hat{\mathbf{w}}^T \mathbf{x}_i < 0$ por $-\mathbf{x}_i$. Con este cambio $\mathbf{w}^T \mathbf{x}_i > 0$ indica clasificación correcta.

Teorema

Suponga:

- $\|\mathbf{x}_i\| \leq K \in \mathbb{R}, \quad i = 1, \dots, n.$
- $\exists \hat{\mathbf{w}} \in \mathbb{R}^{d+1}, \delta > 0 \quad \text{tal que} \quad \hat{\mathbf{w}}^T \mathbf{x}_i \geq \delta \quad i = 1, \dots, n.$

Convergencia

- S.P.D.G Cambie los \mathbf{x}_i para los cuales $\hat{\mathbf{w}}^T \mathbf{x}_i < 0$ por $-\mathbf{x}_i$. Con este cambio $\mathbf{w}^T \mathbf{x}_i > 0$ indica clasificación correcta.

Teorema

Suponga:

- $\|\mathbf{x}_i\| \leq K \in \mathbb{R}, \quad i = 1, \dots, n.$
- $\exists \hat{\mathbf{w}} \in \mathbb{R}^{d+1}, \delta > 0 \quad \text{tal que} \quad \hat{\mathbf{w}}^T \mathbf{x}_i \geq \delta \quad i = 1, \dots, n.$

Entonces el algoritmo del perceptrón ejecuta el paso de actualización a lo sumo $\left(\frac{K\|\hat{\mathbf{w}}\|}{\delta}\right)^2$ veces.

Convergencia

- S.P.D.G Cambie los \mathbf{x}_i para los cuales $\hat{\mathbf{w}}^T \mathbf{x}_i < 0$ por $-\mathbf{x}_i$. Con este cambio $\mathbf{w}^T \mathbf{x}_i > 0$ indica clasificación correcta.

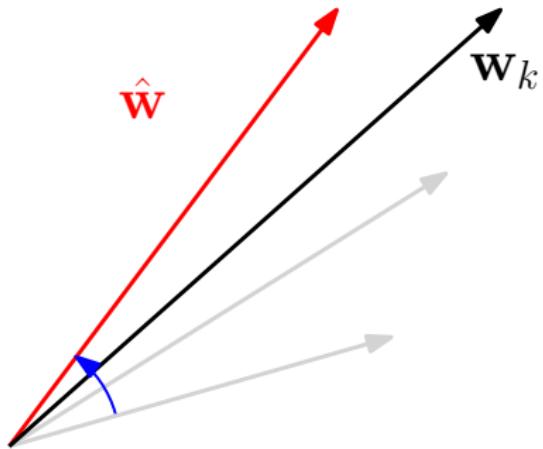
Teorema

Suponga:

- $\|\mathbf{x}_i\| \leq K \in \mathbb{R}, \quad i = 1, \dots, n.$
- $\exists \hat{\mathbf{w}} \in \mathbb{R}^{d+1}, \delta > 0 \quad \text{tal que} \quad \hat{\mathbf{w}}^T \mathbf{x}_i \geq \delta \quad i = 1, \dots, n.$

Entonces el algoritmo del perceptrón ejecuta el paso de actualización a lo sumo $\left(\frac{K\|\hat{\mathbf{w}}\|}{\delta}\right)^2$ veces.

- Es decir, para datos linealmente separables, el algoritmo del perceptrón converge en un número finito de pasos.



Demostración

Demostración

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = \mathbf{0}$ tenemos:

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = \mathbf{0}$ tenemos:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{x}_i$$

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = 0$ tenemos:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{x}_i$$

$$\hat{\mathbf{w}}^T \mathbf{w}_t = \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \hat{\mathbf{w}}^T \mathbf{x}_i \geq \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \delta$$

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = 0$ tenemos:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{x}_i$$

$$\hat{\mathbf{w}}^T \mathbf{w}_t = \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \hat{\mathbf{w}}^T \mathbf{x}_i \geq \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \delta$$

⋮

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = 0$ tenemos:

$$\begin{aligned}\mathbf{w}_t &= \mathbf{w}_{t-1} + \mathbf{x}_i \\ \hat{\mathbf{w}}^T \mathbf{w}_t &= \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \hat{\mathbf{w}}^T \mathbf{x}_i \geq \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \delta \\ &\vdots \\ \hat{\mathbf{w}}^T \mathbf{w}_t &\geq (t\delta)\end{aligned}$$

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = 0$ tenemos:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{x}_i$$

$$\hat{\mathbf{w}}^T \mathbf{w}_t = \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \hat{\mathbf{w}}^T \mathbf{x}_i \geq \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \delta$$

⋮

$$\hat{\mathbf{w}}^T \mathbf{w}_t \geq (t\delta)$$

- Similarmente:

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = 0$ tenemos:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{x}_i$$

$$\hat{\mathbf{w}}^T \mathbf{w}_t = \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \hat{\mathbf{w}}^T \mathbf{x}_i \geq \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \delta$$

⋮

$$\hat{\mathbf{w}}^T \mathbf{w}_t \geq (t\delta)$$

- Similarmente:

$$\|\mathbf{w}_t\|^2 = \mathbf{w}_t^T \mathbf{w}_t = (\mathbf{w}_{t-1} + \mathbf{x}_i)^T (\mathbf{w}_{t-1} + \mathbf{x}_i)$$

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = 0$ tenemos:

$$\begin{aligned}\mathbf{w}_t &= \mathbf{w}_{t-1} + \mathbf{x}_i \\ \hat{\mathbf{w}}^T \mathbf{w}_t &= \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \hat{\mathbf{w}}^T \mathbf{x}_i \geq \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \delta \\ &\vdots \\ \hat{\mathbf{w}}^T \mathbf{w}_t &\geq (t\delta)\end{aligned}$$

- Similarmente:

$$\begin{aligned}\|\mathbf{w}_t\|^2 &= \mathbf{w}_t^T \mathbf{w}_t = (\mathbf{w}_{t-1} + \mathbf{x}_i)^T (\mathbf{w}_{t-1} + \mathbf{x}_i) \\ &= \|\mathbf{w}_{t-1}\|^2 + 2\mathbf{w}_{t-1}^T \mathbf{x}_i + \|\mathbf{x}_i\|^2 \leq \|\mathbf{w}_{t-1}\|^2 + K^2\end{aligned}$$

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = 0$ tenemos:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{x}_i$$

$$\hat{\mathbf{w}}^T \mathbf{w}_t = \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \hat{\mathbf{w}}^T \mathbf{x}_i \geq \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \delta$$

⋮

$$\hat{\mathbf{w}}^T \mathbf{w}_t \geq (t\delta)$$

- Similarmente:

$$\begin{aligned}\|\mathbf{w}_t\|^2 &= \mathbf{w}_t^T \mathbf{w}_t = (\mathbf{w}_{t-1} + \mathbf{x}_i)^T (\mathbf{w}_{t-1} + \mathbf{x}_i) \\ &= \|\mathbf{w}_{t-1}\|^2 + 2\mathbf{w}_{t-1}^T \mathbf{x}_i + \|\mathbf{x}_i\|^2 \leq \|\mathbf{w}_{t-1}\|^2 + K^2\end{aligned}$$

⋮

Demostración

- Sea t el número de correcciones a \mathbf{w} . Con $\mathbf{w}_0 = 0$ tenemos:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \mathbf{x}_i$$

$$\hat{\mathbf{w}}^T \mathbf{w}_t = \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \hat{\mathbf{w}}^T \mathbf{x}_i \geq \hat{\mathbf{w}}^T \mathbf{w}_{t-1} + \delta$$

⋮

$$\hat{\mathbf{w}}^T \mathbf{w}_t \geq (t\delta)$$

- Similarmente:

$$\begin{aligned}\|\mathbf{w}_t\|^2 &= \mathbf{w}_t^T \mathbf{w}_t = (\mathbf{w}_{t-1} + \mathbf{x}_i)^T (\mathbf{w}_{t-1} + \mathbf{x}_i) \\ &= \|\mathbf{w}_{t-1}\|^2 + 2\mathbf{w}_{t-1}^T \mathbf{x}_i + \|\mathbf{x}_i\|^2 \leq \|\mathbf{w}_{t-1}\|^2 + K^2\end{aligned}$$

⋮

$$\|\mathbf{w}_t\|^2 \leq tK^2$$

- Combinando:

- Combinando:

$$t\delta \leq \hat{\mathbf{w}}^T \mathbf{w}_t$$

- Combinando:

$$\begin{aligned} t\delta &\leq \hat{\mathbf{w}}^T \mathbf{w}_t \\ &= \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \cos\theta \end{aligned}$$

- Combinando:

$$\begin{aligned} t\delta &\leq \hat{\mathbf{w}}^T \mathbf{w}_t \\ &= \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \cos\theta \\ &\leq \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \end{aligned}$$

- Combinando:

$$\begin{aligned} t\delta &\leq \hat{\mathbf{w}}^T \mathbf{w}_t \\ &= \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \cos\theta \\ &\leq \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \\ &\leq \|\hat{\mathbf{w}}\| K \sqrt{t} \end{aligned}$$

- Combinando:

$$\begin{aligned} t\delta &\leq \hat{\mathbf{w}}^T \mathbf{w}_t \\ &= \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \cos\theta \\ &\leq \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \\ &\leq \|\hat{\mathbf{w}}\| K \sqrt{t} \\ t &\leq \left(\frac{K \|\hat{\mathbf{w}}\|}{\delta} \right)^2 \end{aligned}$$

- Combinando:

$$\begin{aligned} t\delta &\leq \hat{\mathbf{w}}^T \mathbf{w}_t \\ &= \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \cos\theta \\ &\leq \|\hat{\mathbf{w}}\| \|\mathbf{w}_t\| \\ &\leq \|\hat{\mathbf{w}}\| K \sqrt{t} \\ t &\leq \left(\frac{K \|\hat{\mathbf{w}}\|}{\delta} \right)^2 \end{aligned}$$



Redes Multicapa

- Redes con una sola capa tienen capacidad funcional limitada.

Redes Multicapa

- Redes con una sola capa tienen capacidad funcional limitada.
- Redes multicapa implementan funciones más complejas.

Redes Multicapa

- Redes con una sola capa tienen capacidad funcional limitada.
- Redes multicapa implementan funciones más complejas.
- Capacidad de construir representaciones internas de los datos de entrada.

Redes Multicapa

- Redes con una sola capa tienen capacidad funcional limitada.
- Redes multicapa implementan funciones más complejas.
- Capacidad de construir representaciones internas de los datos de entrada.
- Problema de asignación de crédito.

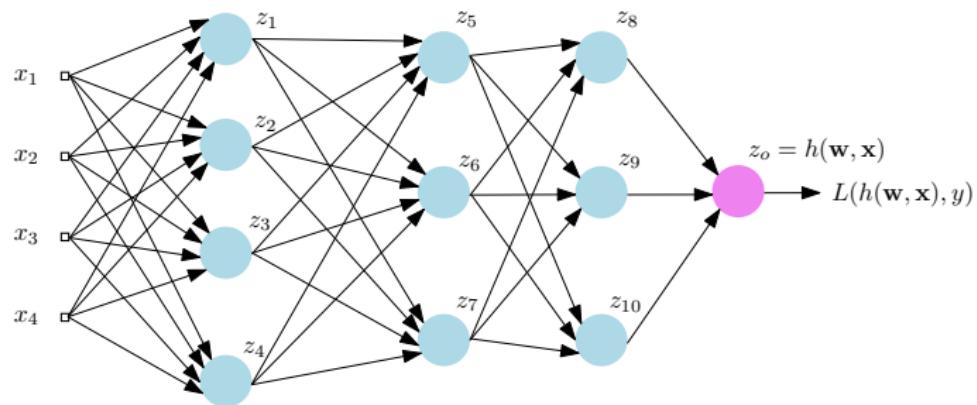
Redes Multicapa

- Redes con una sola capa tienen capacidad funcional limitada.
- Redes multicapa implementan funciones más complejas.
- Capacidad de construir representaciones internas de los datos de entrada.
- Problema de asignación de crédito.
- Algoritmos de entrenamiento más complejos

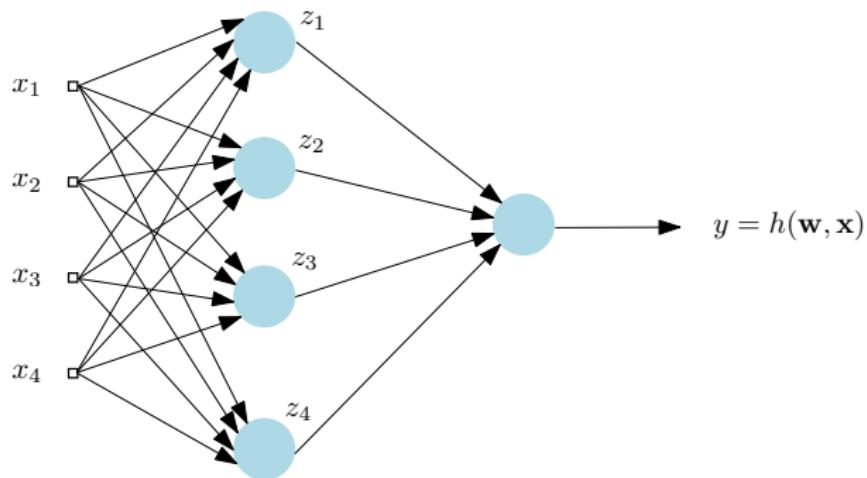
Redes Multicapa

- Redes con una sola capa tienen capacidad funcional limitada.
- Redes multicapa implementan funciones más complejas.
- Capacidad de construir representaciones internas de los datos de entrada.
- Problema de asignación de crédito.
- Algoritmos de entrenamiento más complejos

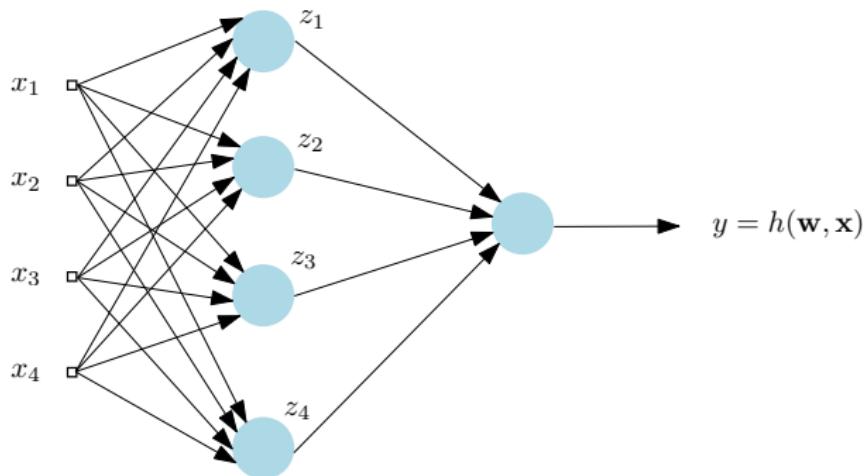
Arquitectura en Capas



Suma de funciones base



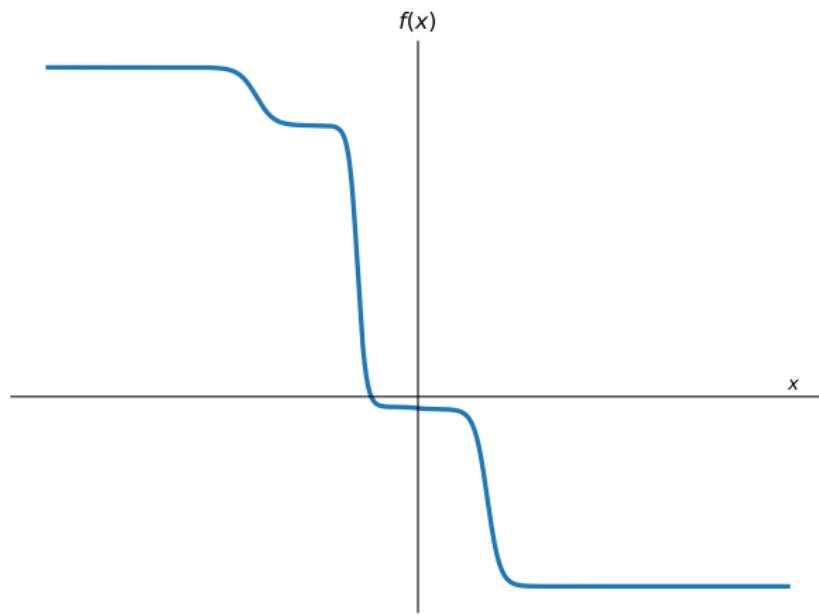
Suma de funciones base



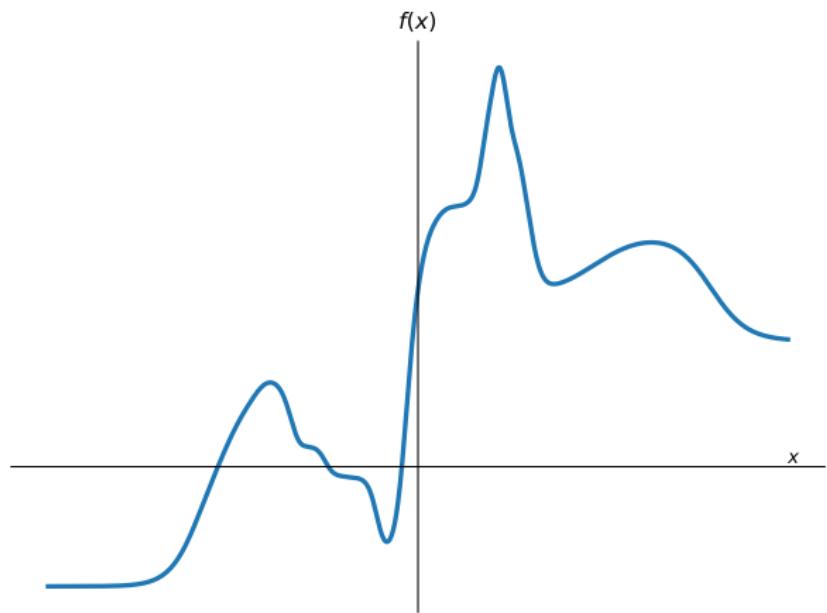
$$y = h(\mathbf{x}) = f_o \left(a_0 + \sum_{k=1}^N a_k f_k(\mathbf{w}_k^T \mathbf{x}) \right)$$

$$\stackrel{f_o(z)=z}{=} a_0 + \sum_{k=1}^N a_k f_k(\mathbf{w}_k^T \mathbf{x})$$

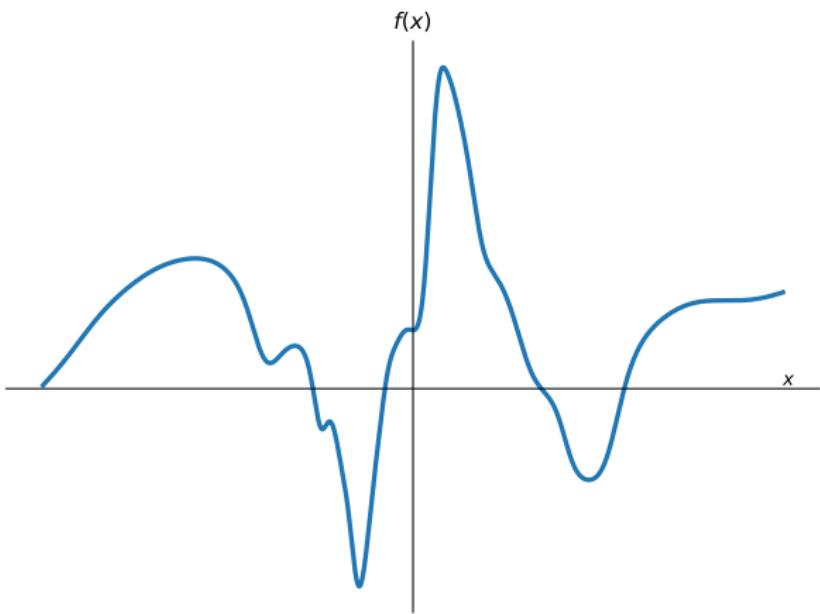
Sigmoidal, N=5



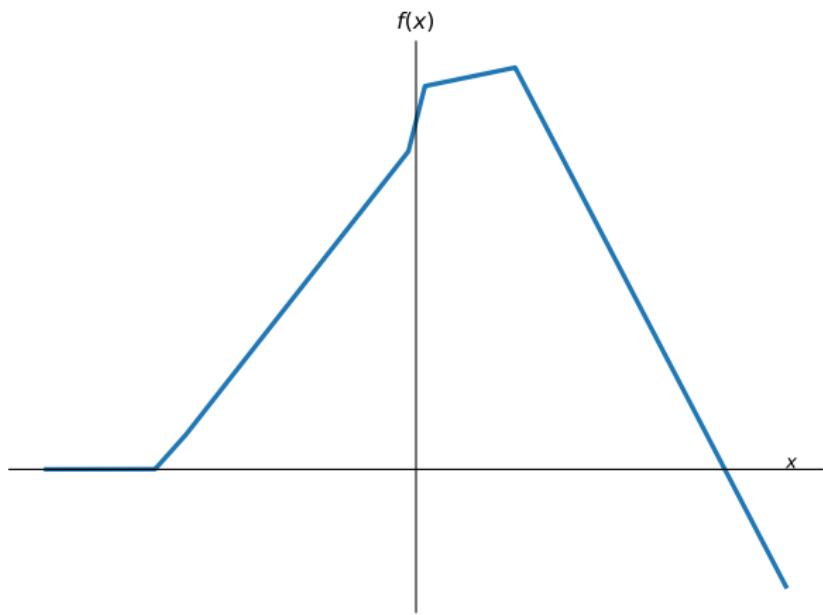
Sigmoidal, N=25



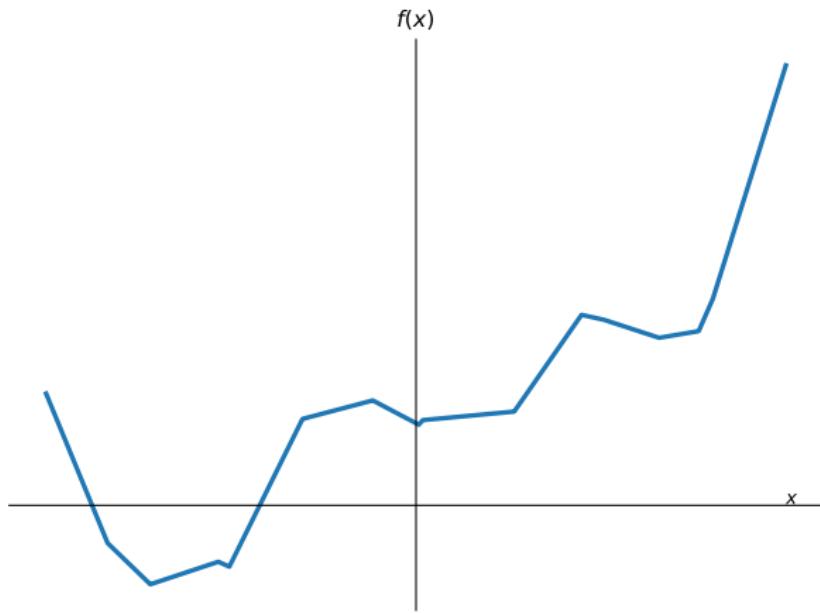
Sigmoidal, N=100



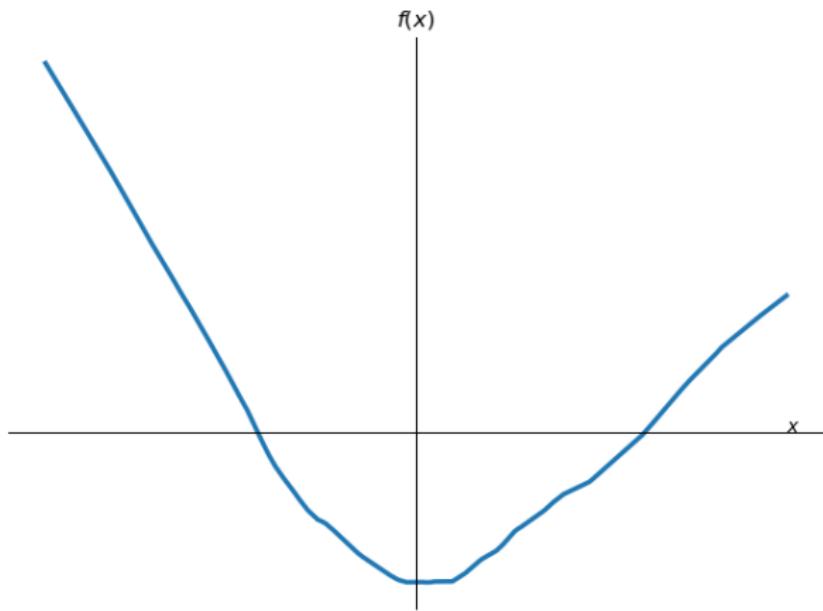
ReLU, N=5



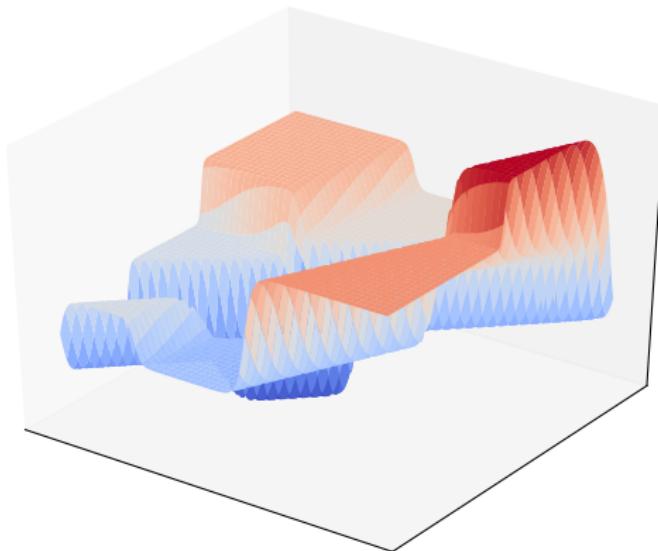
ReLU, N=25



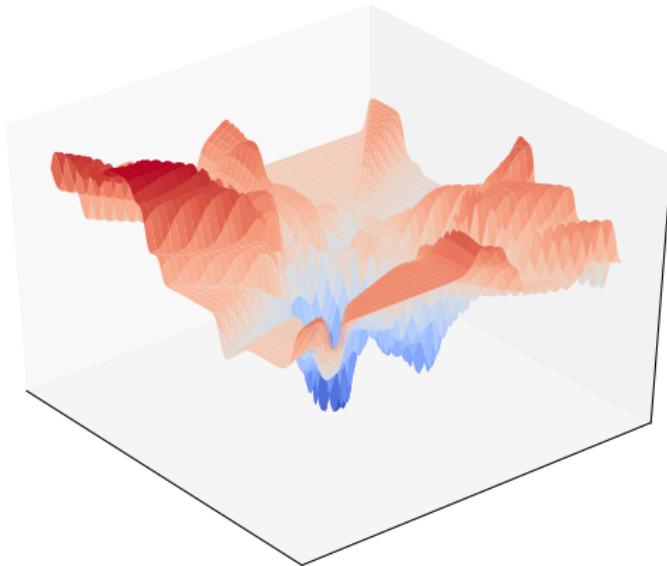
ReLU, N=100



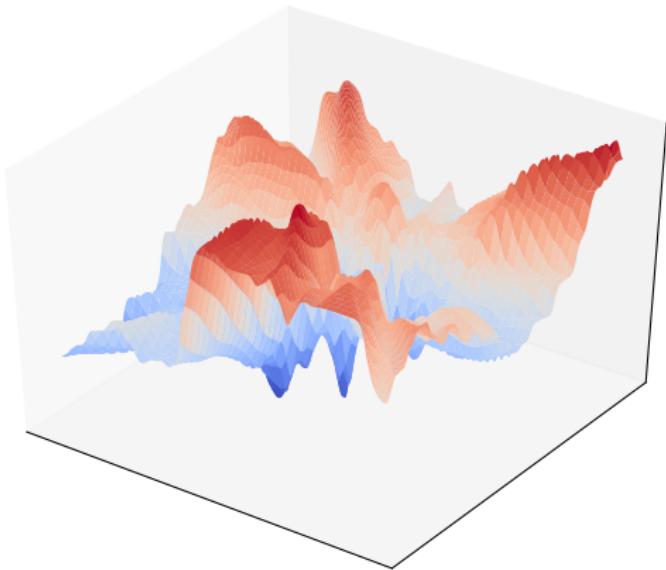
Sigmoidal, N=5



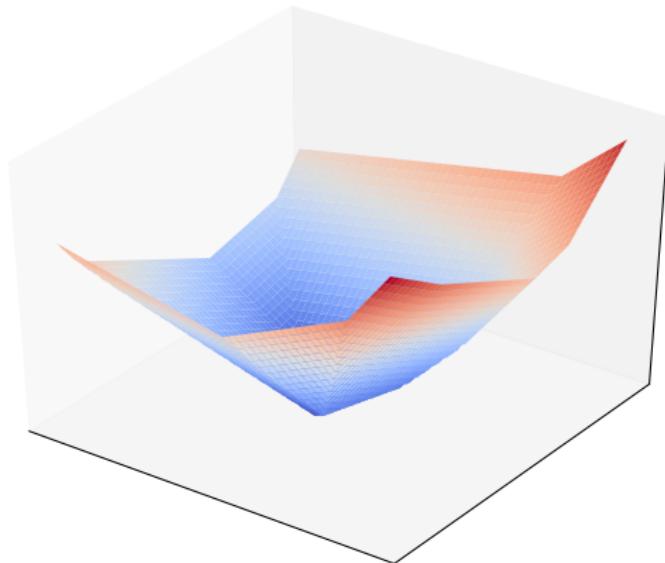
Sigmoidal, N=25



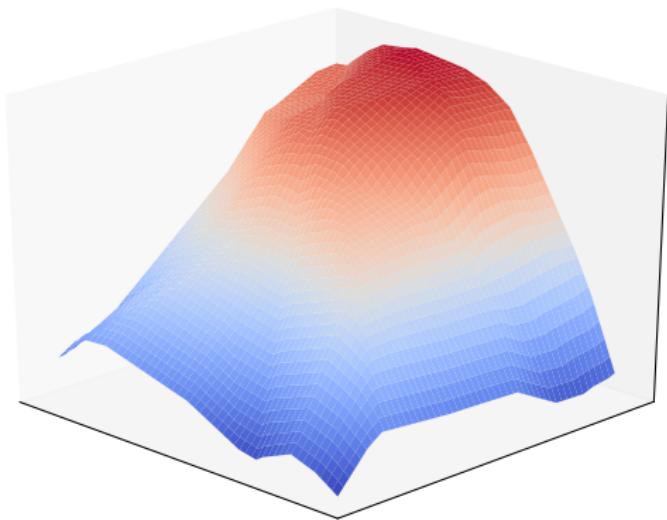
Sigmoidal, N=100



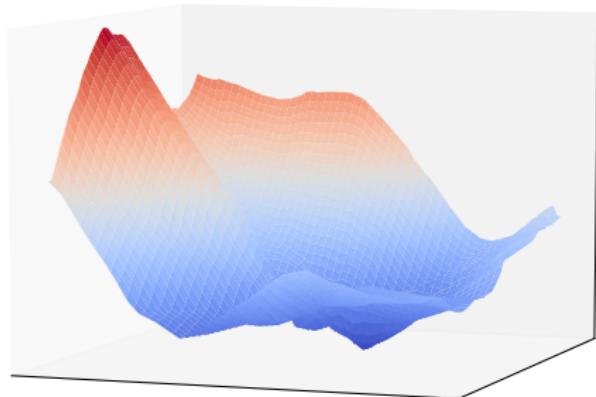
ReLU, N=5



ReLU, N=25



ReLU, N=100



Capacidad Funcional

Capacidad Funcional

- Aproximación universal : Una red con una capa escondida, es capaz de aproximar cualquier función “suave” con precisión arbitraria, si se permite incrementar el número de neuronas k indefinidamente.

Capacidad Funcional

- Aproximación universal : Una red con una capa escondida, es capaz de aproximar cualquier función “suave” con precisión arbitraria, si se permite incrementar el número de neuronas k indefinidamente.
 - ▶ MLPs: Cybenko (1989), Hornik (1991), Funahashi (1989)

Capacidad Funcional

- Aproximación universal : Una red con una capa escondida, es capaz de aproximar cualquier función “suave” con precisión arbitraria, si se permite incrementar el número de neuronas k indefinidamente.
 - ▶ MLPs: Cybenko (1989), Hornik (1991), Funahashi (1989).
 - ▶ RBFs: Hartman et.al (1990), Girossi y Pogio (1990).

Capacidad Funcional

- Aproximación universal : Una red con una capa escondida, es capaz de aproximar cualquier función “suave” con precisión arbitraria, si se permite incrementar el número de neuronas k indefinidamente.
 - ▶ MLPs: Cybenko (1989), Hornik (1991), Funahashi (1989).
 - ▶ RBFs: Hartman et.al (1990), Girossi y Pogio (1990).
- Esto **no** quiere decir que una red con una capa escondida y un número limitado de neuronas solucione cualquier problema!

Capacidad Funcional

- Aproximación universal : Una red con una capa escondida, es capaz de aproximar cualquier función “suave” con precisión arbitraria, si se permite incrementar el número de neuronas k indefinidamente.
 - ▶ MLPs: Cybenko (1989), Hornik (1991), Funahashi (1989).
 - ▶ RBFs: Hartman et.al (1990), Girossi y Pogio (1990).
- Esto **no** quiere decir que una red con una capa escondida y un número limitado de neuronas solucione cualquier problema!
- Tasa de aproximación $O\left(\frac{1}{n}\right)$ (es $O\left(\frac{1}{n^{\frac{1}{d}}}\right)$ para modelos lineales en los parámetros).

MLP: Algoritmo de entrenamiento

- Backpropagation: Werbos (1974), Rumelhart, Hinton, y Williams (1986), LeCun (1986), Parker (1985).

MLP: Algoritmo de entrenamiento

- Backpropagation: Werbos (1974), Rumelhart, Hinton, y Williams (1986), LeCun (1986), Parker (1985).
- Algoritmo iterativo de búsqueda usando gradiente (**estocástico**).

MLP: Algoritmo de entrenamiento

- Backpropagation: Werbos (1974), Rumelhart, Hinton, y Williams (1986), LeCun (1986), Parker (1985).
- Algoritmo iterativo de búsqueda usando gradiente (**estocástico**).
- Regla de la cadena.

Backpropagation

- Función de error:

$$L(\mathbf{w}) = \sum_{p=1}^n l(h(\mathbf{w}, \mathbf{x}_p), y_p) = \sum_{i=1}^n L_p$$

Backpropagation

- Función de error:

$$L(\mathbf{w}) = \sum_{p=1}^n l(h(\mathbf{w}, \mathbf{x}_p), y_p) = \sum_{i=1}^n L_p$$

- Procedimiento iterativo:

Backpropagation

- Función de error:

$$L(\mathbf{w}) = \sum_{p=1}^n l(h(\mathbf{w}, \mathbf{x}_p), y_p) = \sum_{i=1}^n L_p$$

- Procedimiento iterativo:

- ① Punto inicial \mathbf{w}_0 :

Backpropagation

- Función de error:

$$L(\mathbf{w}) = \sum_{p=1}^n l(h(\mathbf{w}, \mathbf{x}_p), y_p) = \sum_{i=1}^n L_p$$

- Procedimiento iterativo:

- ① Punto inicial \mathbf{w}_0 :
 - ★ Aleatorio.

Backpropagation

- Función de error:

$$L(\mathbf{w}) = \sum_{p=1}^n l(h(\mathbf{w}, \mathbf{x}_p), y_p) = \sum_{i=1}^n L_p$$

- Procedimiento iterativo:

- ① Punto inicial \mathbf{w}_0 :

- ★ Aleatorio.
- ★ Aleatorio + normalización.

Backpropagation

- Función de error:

$$L(\mathbf{w}) = \sum_{p=1}^n l(h(\mathbf{w}, \mathbf{x}_p), y_p) = \sum_{i=1}^n L_p$$

- Procedimiento iterativo:

- ① Punto inicial \mathbf{w}_0 :

- ★ Aleatorio.
- ★ Aleatorio + normalización.
- ★ Nguyen-Widrow.

Backpropagation

- Función de error:

$$L(\mathbf{w}) = \sum_{p=1}^n l(h(\mathbf{w}, \mathbf{x}_p), y_p) = \sum_{i=1}^n L_p$$

- Procedimiento iterativo:

- ➊ Punto inicial \mathbf{w}_0 :

- ★ Aleatorio.
- ★ Aleatorio + normalización.
- ★ Nguyen-Widrow.

- ➋ Descenso de gradiente:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}} L|_{\mathbf{w}_k}$$

Backpropagation

- Función de error:

$$L(\mathbf{w}) = \sum_{p=1}^n l(h(\mathbf{w}, \mathbf{x}_p), y_p) = \sum_{i=1}^n L_p$$

- Procedimiento iterativo:

- ① Punto inicial \mathbf{w}_0 :

- ★ Aleatorio.
- ★ Aleatorio + normalización.
- ★ Nguyen-Widrow.

- ② Descenso de gradiente:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}} L|_{\mathbf{w}_k}$$

- $\nabla_{\mathbf{w}} L|_{\mathbf{w}_k}$ puede calcularse exactamente (batch backpropagation), o estimarse con un sólo dato (on-line backpropagation).

Dificultades

Dificultades

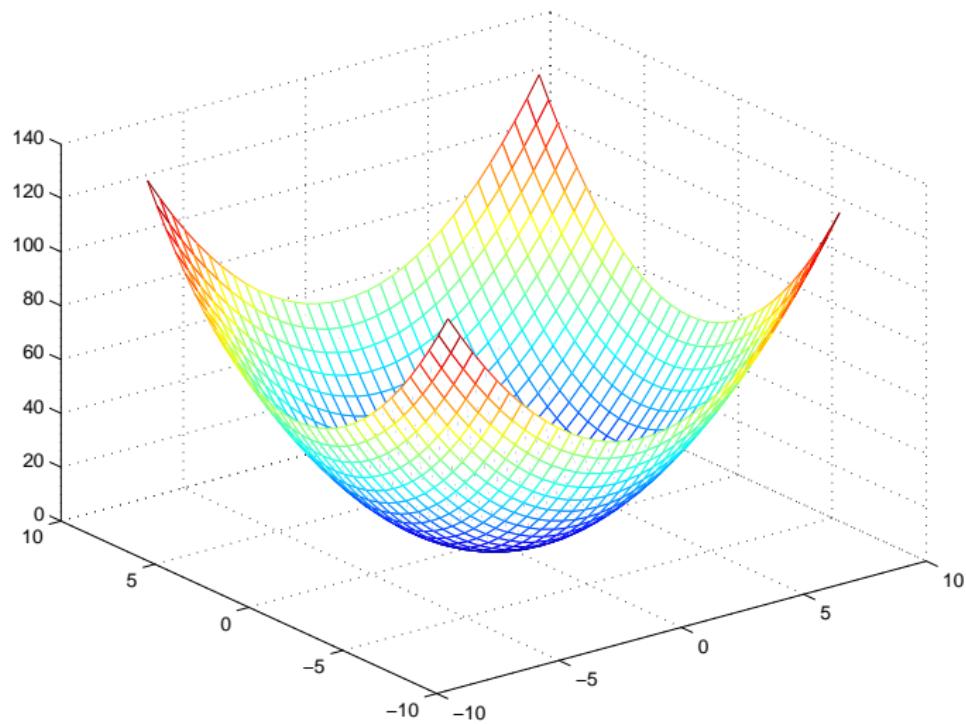
- Función de error **no es convexa**.

Dificultades

- Función de error **no es convexa**.
- No es **amigable** para algoritmos de optimización.

Dificultades

- Función de error **no es convexa**.
- No es **amigable** para algoritmos de optimización.



Forward pass

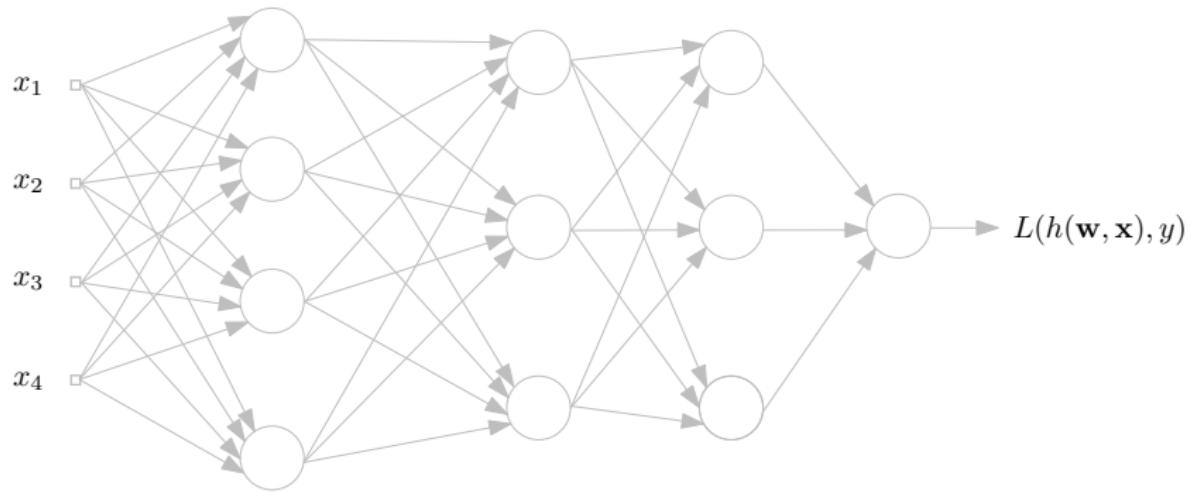
$$a_j = \sum_i w_{ij} z_i$$

Forward pass

$$a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$$

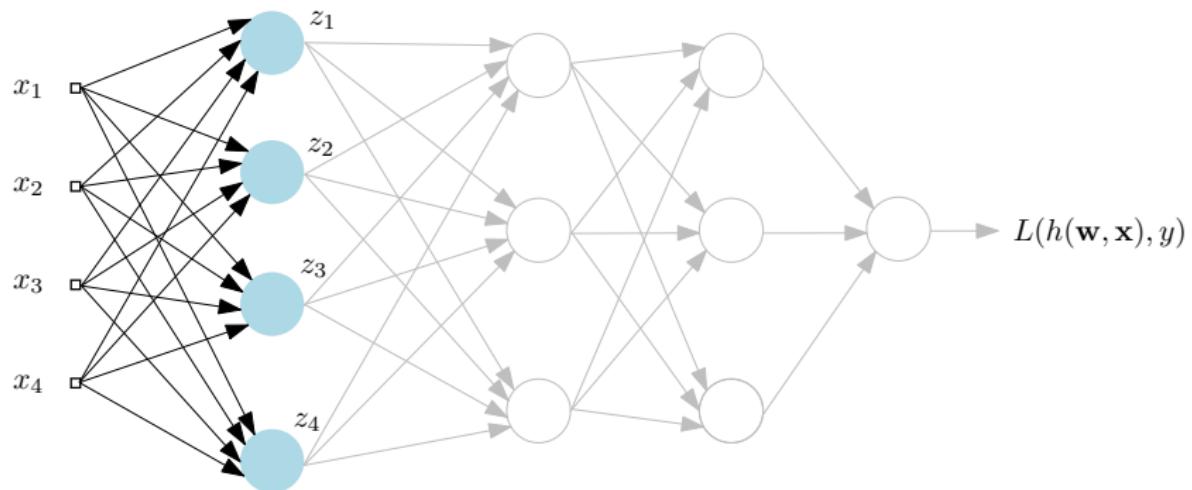
Forward pass

$$a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$$



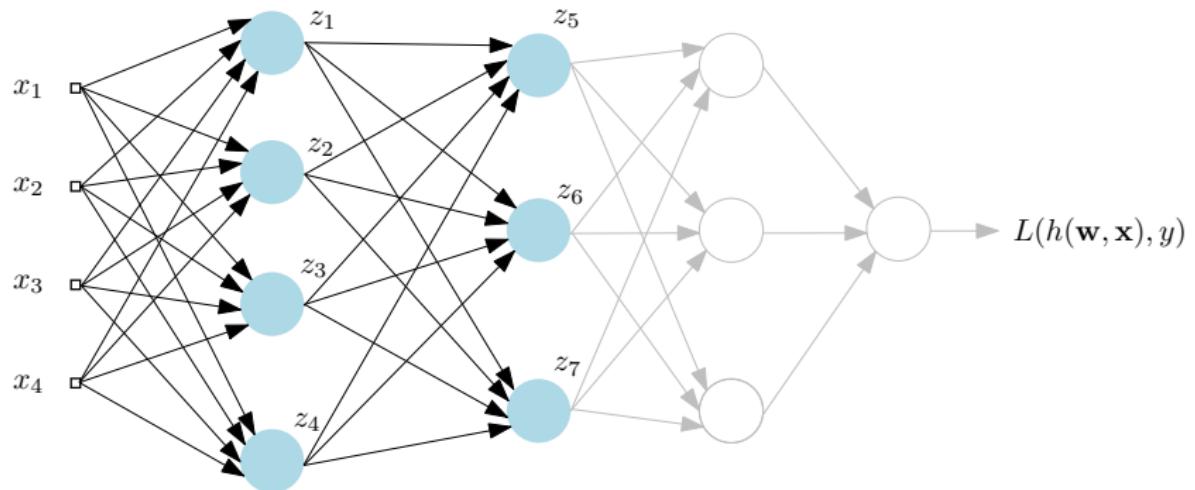
Forward pass

$$a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$$



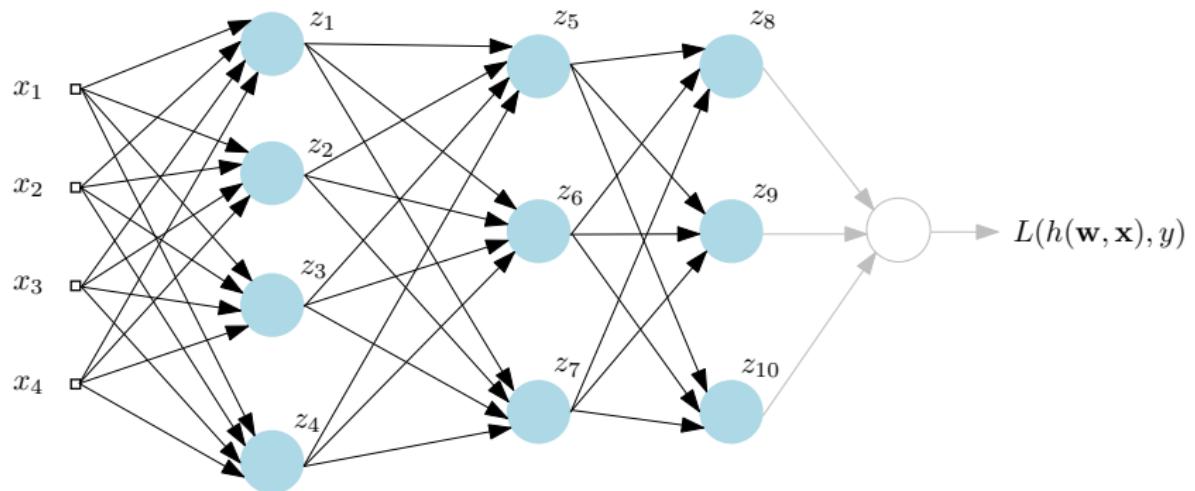
Forward pass

$$a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$$



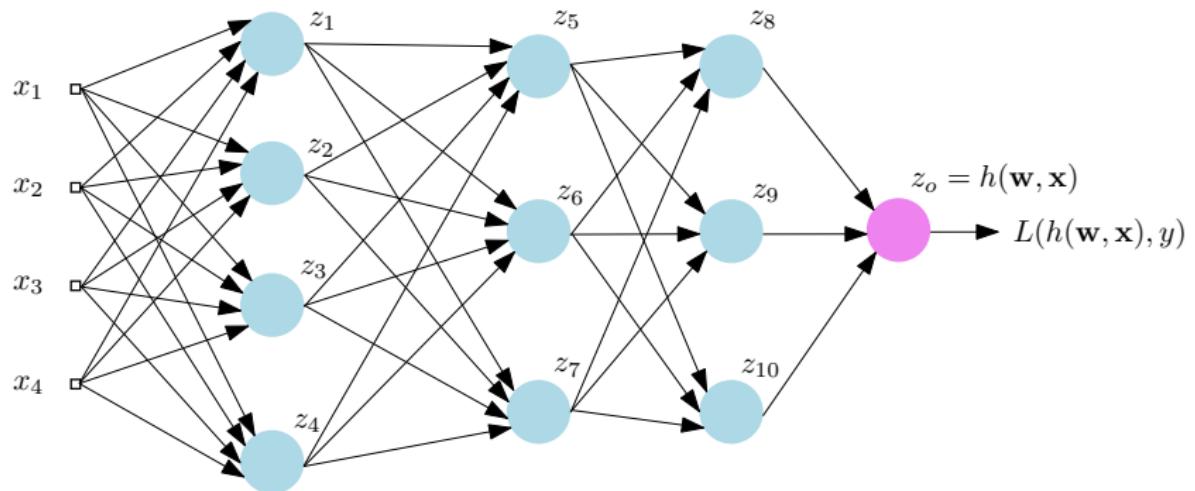
Forward pass

$$a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$$

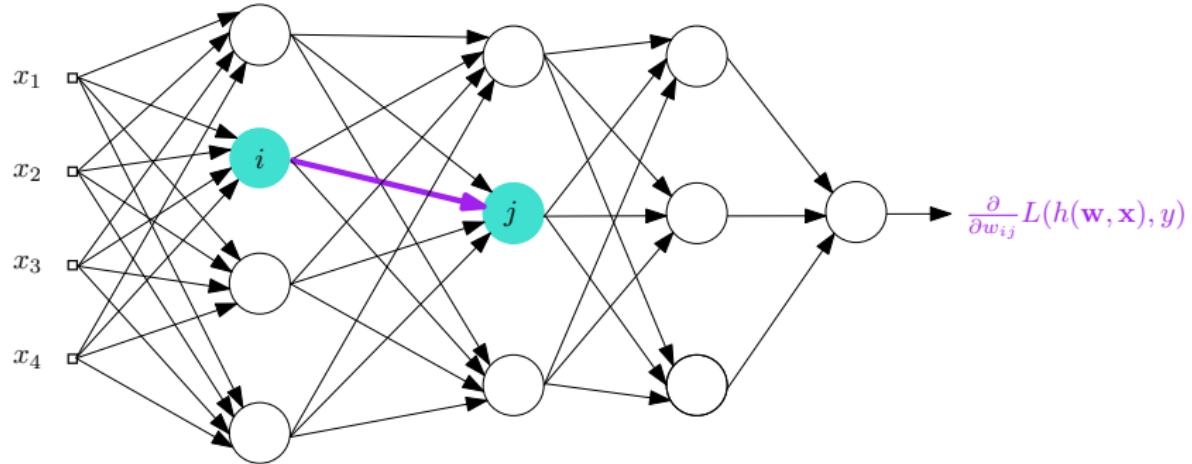


Forward pass

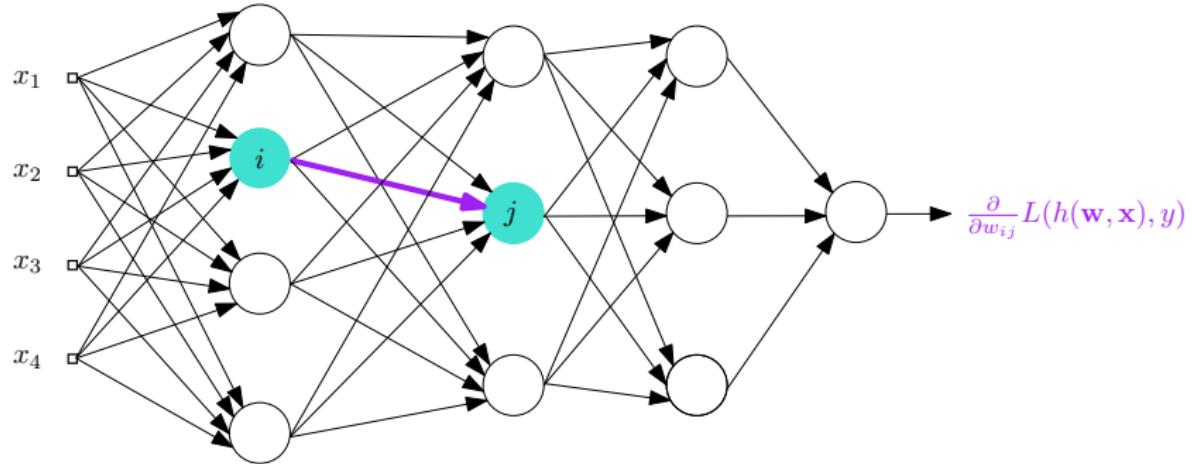
$$a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$$



Cálculo del gradiente

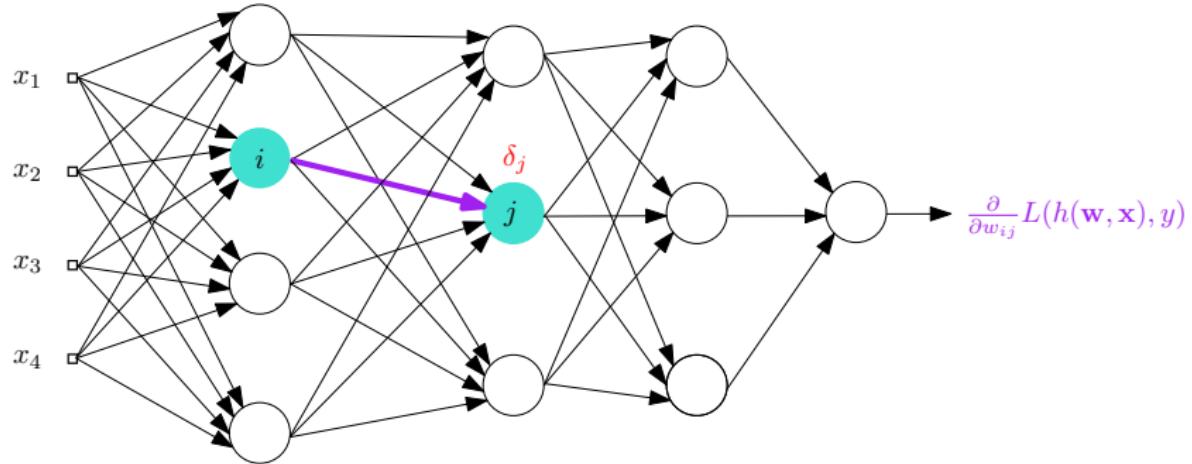


Cálculo del gradiente



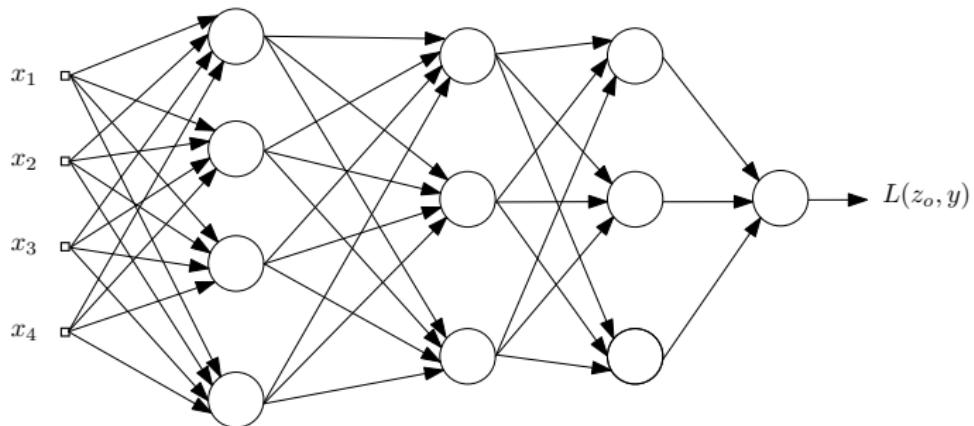
$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}$$

Cálculo del gradiente

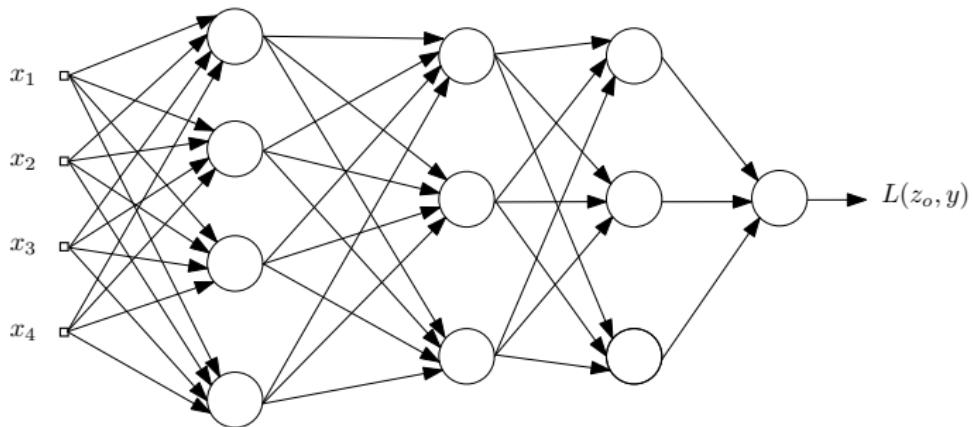


$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = \delta_j z_i$$

Cálculo de los δ

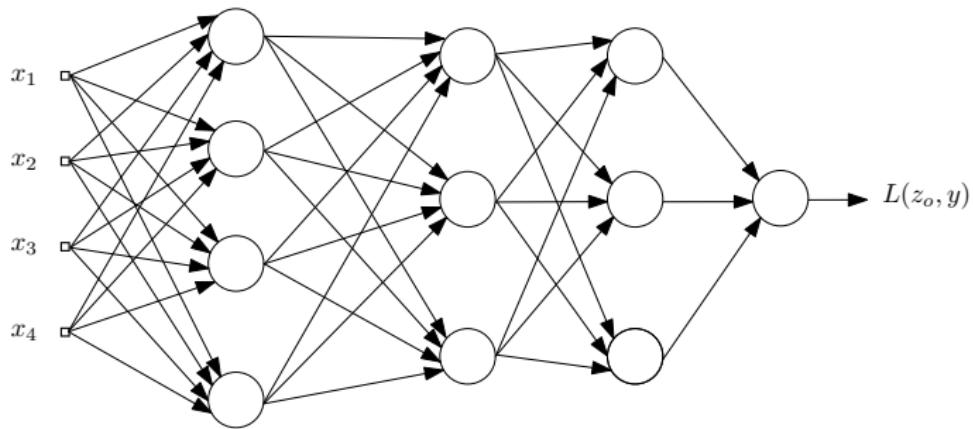


Cálculo de los δ



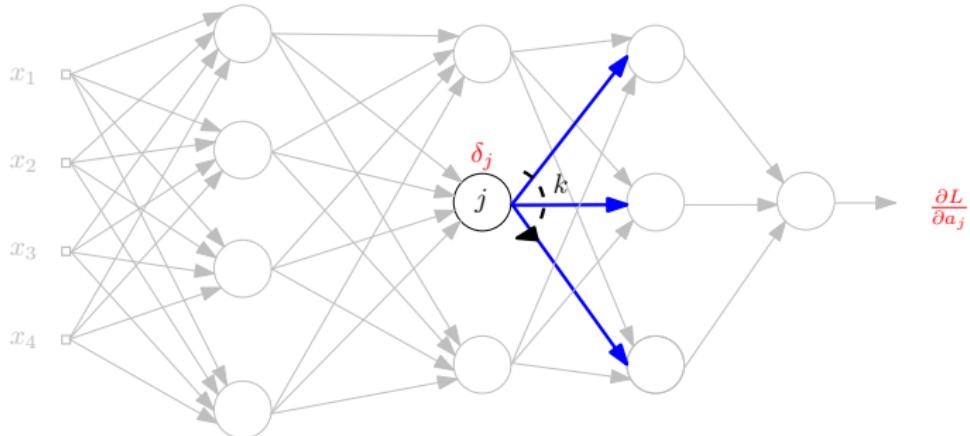
- En la salida:

Cálculo de los δ

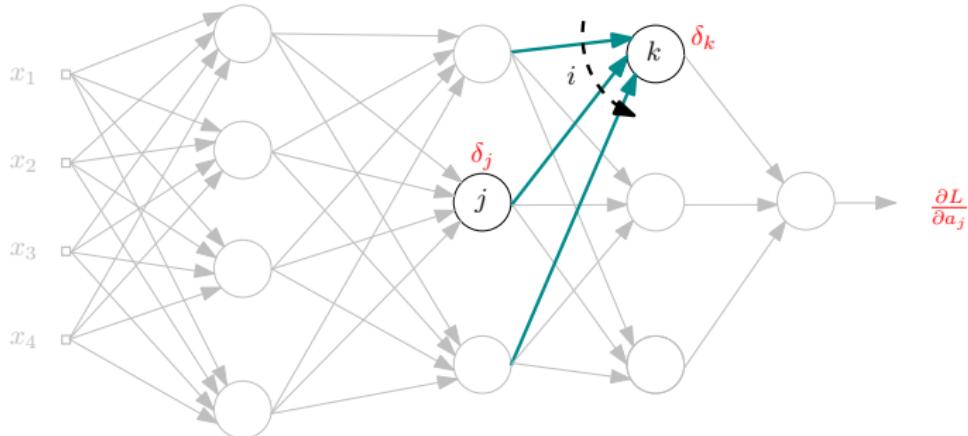


- En la salida:

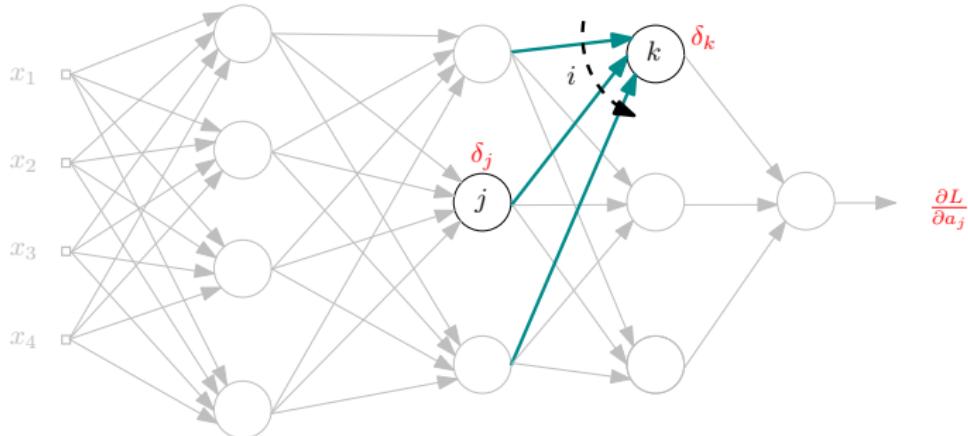
$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o}$$



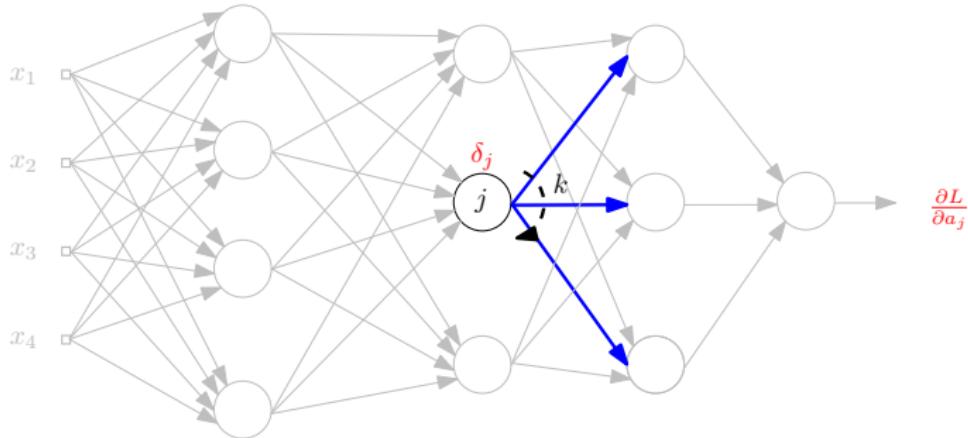
$$\delta_j = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$



$$\delta_j = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial}{\partial a_j} \sum_i w_{ik} f_i(a_i)$$



$$\delta_j = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial}{\partial a_j} \sum_i w_{ik} f_i(a_i) = f'_j(a_j) \sum_k w_{jk} \delta_k$$



$$\delta_j = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial}{\partial a_j} \sum_i w_{ik} f_i(a_i) = f'_j(a_j) \sum_k w_{jk} \delta_k$$

Ejemplo

Ejemplo

- Función de error cuadrática:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^n (h(\mathbf{w}, \mathbf{x}_p) - y_p)^2$$

Ejemplo

- Función de error cuadrática:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^n (h(\mathbf{w}, \mathbf{x}_p) - y_p)^2$$

Ejemplo

- Función de error cuadrática:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^n (h(\mathbf{w}, \mathbf{x}_p) - y_p)^2$$

- Activación Sigmoide:

$$f_i(a) = f_s(a) = \frac{1}{1 + e^{-a}}$$

Ejemplo

- Función de error cuadrática:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^n (h(\mathbf{w}, \mathbf{x}_p) - y_p)^2$$

- Activación Sísmoide:

$$f_i(a) = f_s(a) = \frac{1}{1 + e^{-a}}$$

- Salida con activación lineal:

$$f_o(a) = a$$

- En la salida:

- En la salida:

$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o}$$

- En la salida:

$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o} = h(\mathbf{w}, \mathbf{x}_p) - y_p$$

- En la salida:

$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o} = h(\mathbf{w}, \mathbf{x}_p) - y_p \leftarrow \text{Señal de error}$$

- En la salida:

$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o} = h(\mathbf{w}, \mathbf{x}_p) - y_p \leftarrow \text{Señal de error}$$

- En otro caso:

- En la salida:

$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o} = h(\mathbf{w}, \mathbf{x}_p) - y_p \leftarrow \text{Señal de error}$$

- En otro caso:

$$\delta_j = f'_s(a_j) \sum_k w_{jk} \delta_k$$

- En la salida:

$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o} = h(\mathbf{w}, \mathbf{x}_p) - y_p \leftarrow \text{Señal de error}$$

- En otro caso:

$$\begin{aligned}\delta_j &= f'_s(a_j) \sum_k w_{jk} \delta_k \\ &= z_j(1 - z_j) \sum_k w_{jk} \delta_k\end{aligned}$$

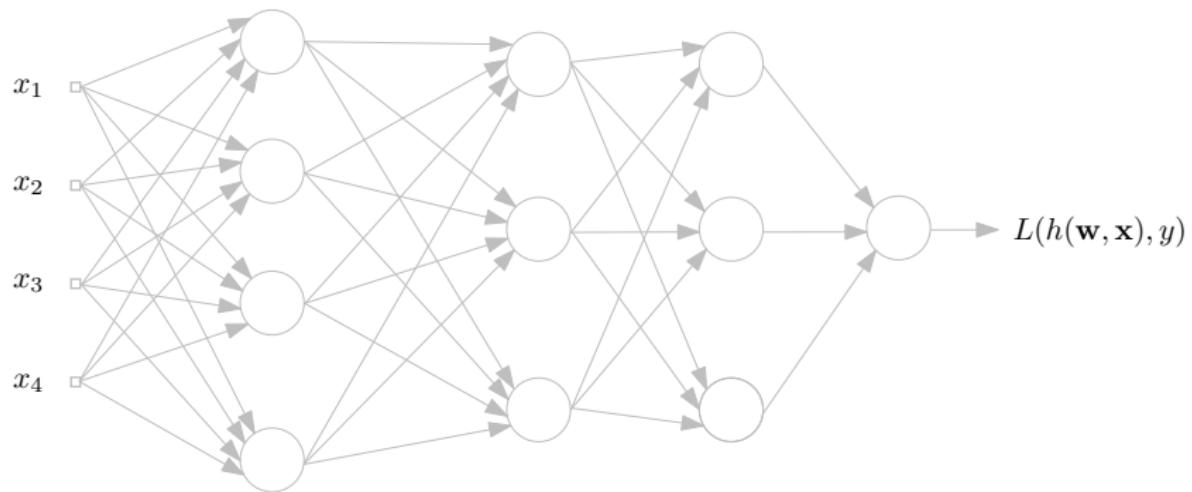
- En la salida:

$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o} = h(\mathbf{w}, \mathbf{x}_p) - y_p \leftarrow \text{Señal de error}$$

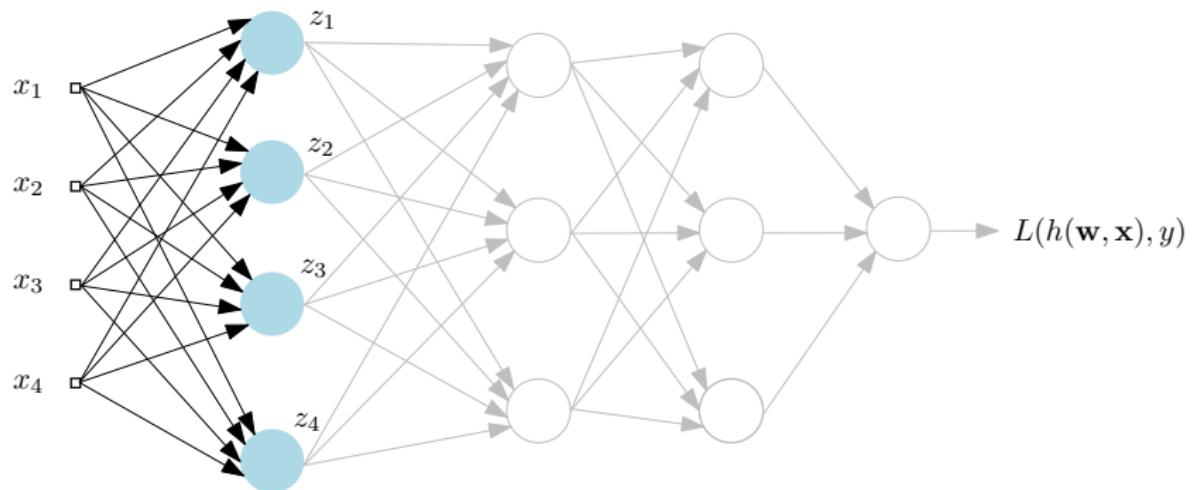
- En otro caso:

$$\begin{aligned}\delta_j &= f'_s(a_j) \sum_k w_{jk} \delta_k \\ &= z_j(1 - z_j) \sum_k w_{jk} \delta_k \leftarrow \text{Señal de error se propaga}\end{aligned}$$

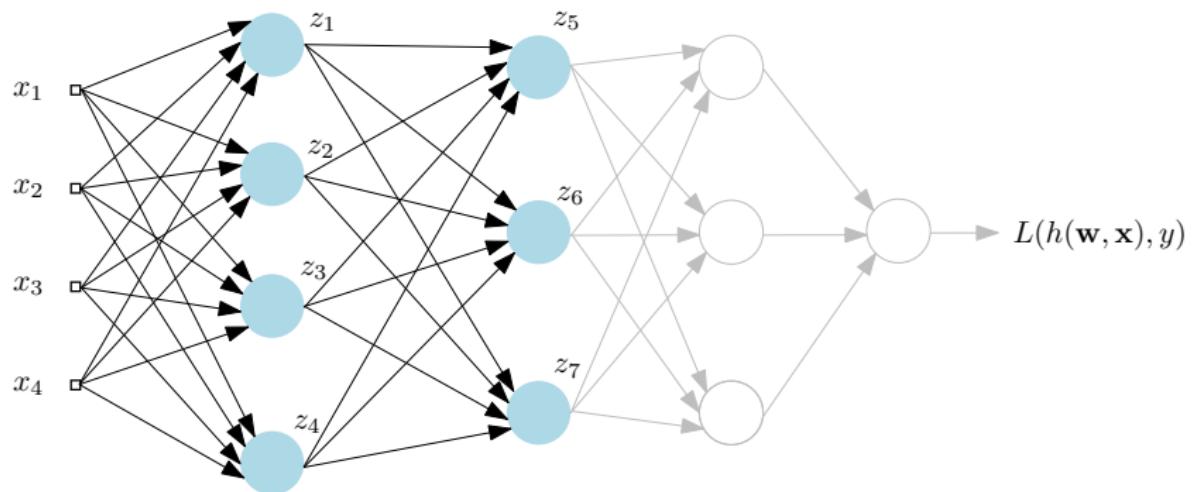
Forward pass



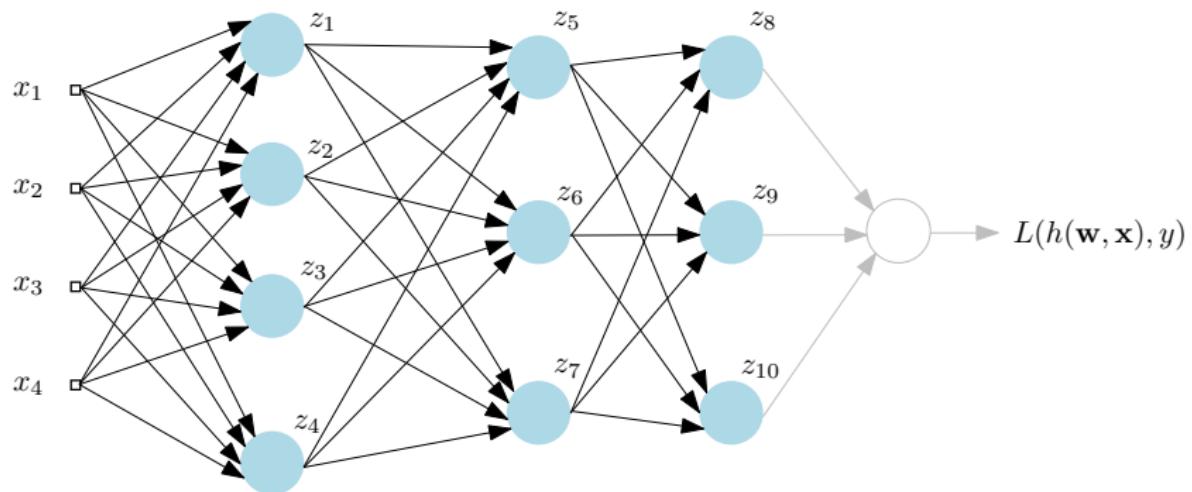
Forward pass



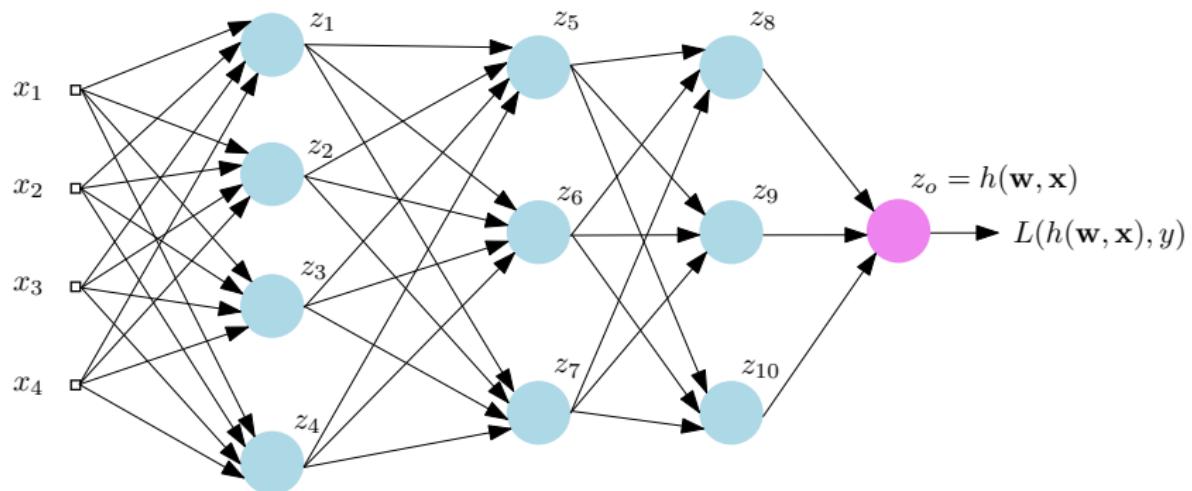
Forward pass



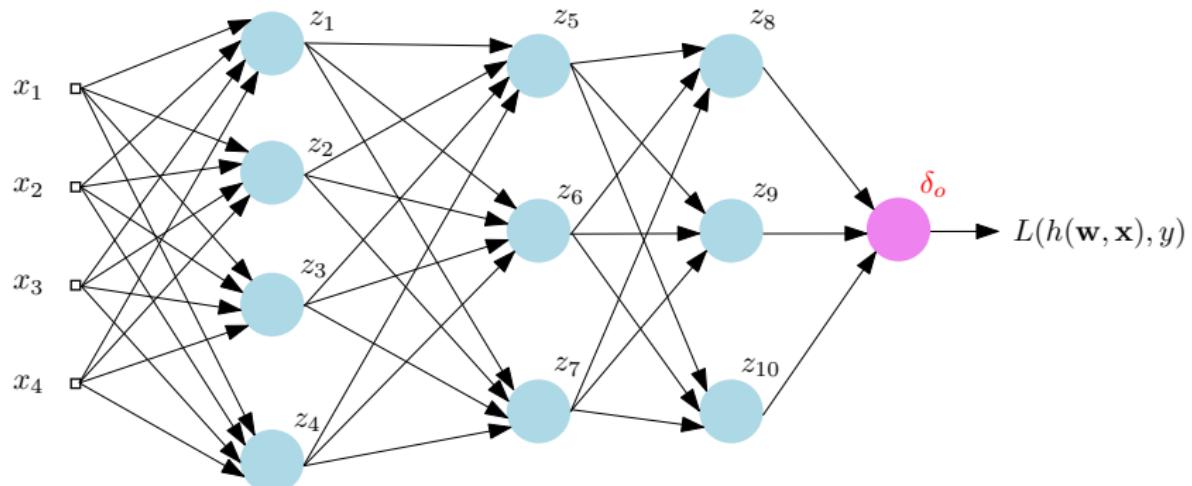
Forward pass



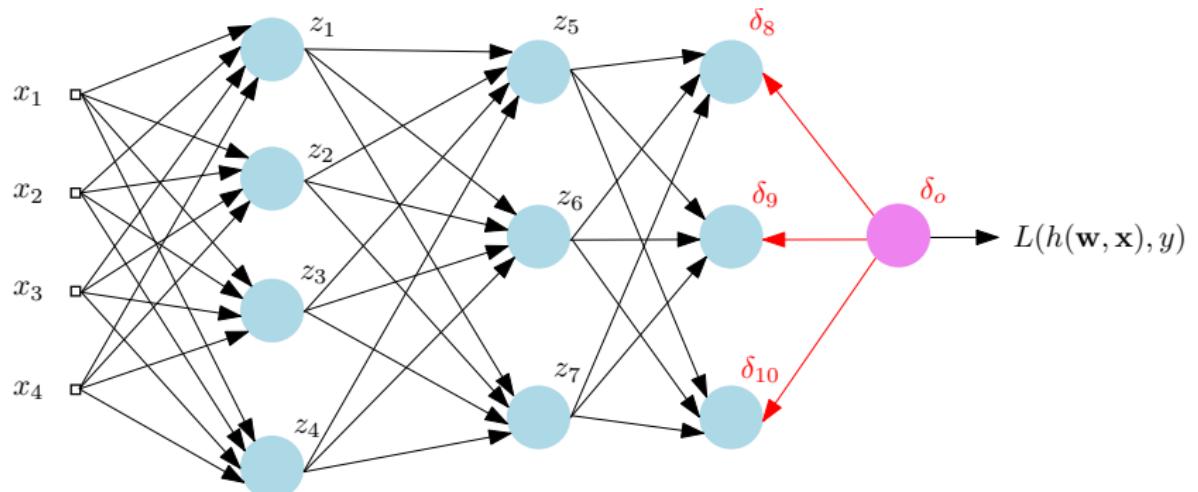
Forward pass



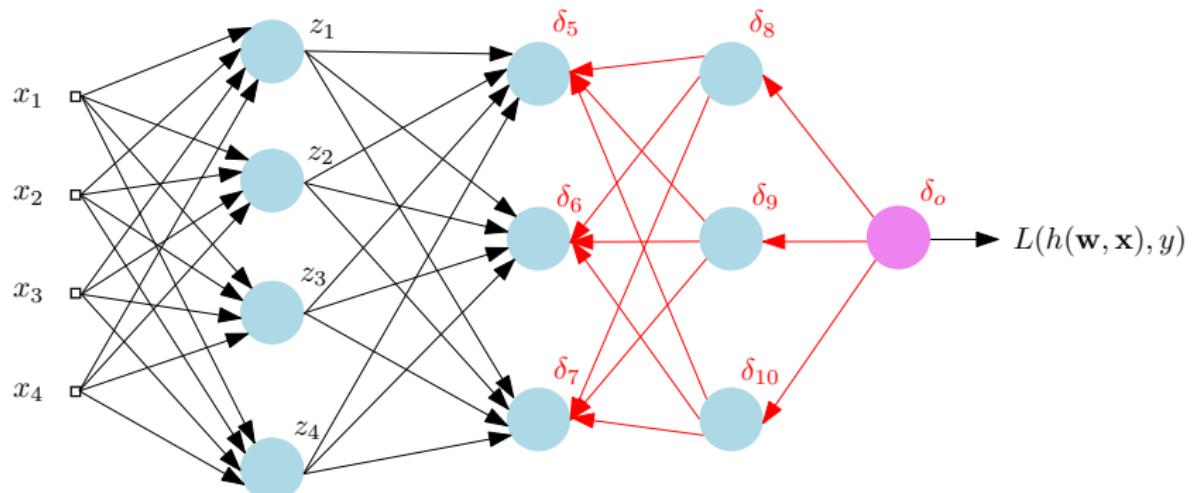
Backpropagation



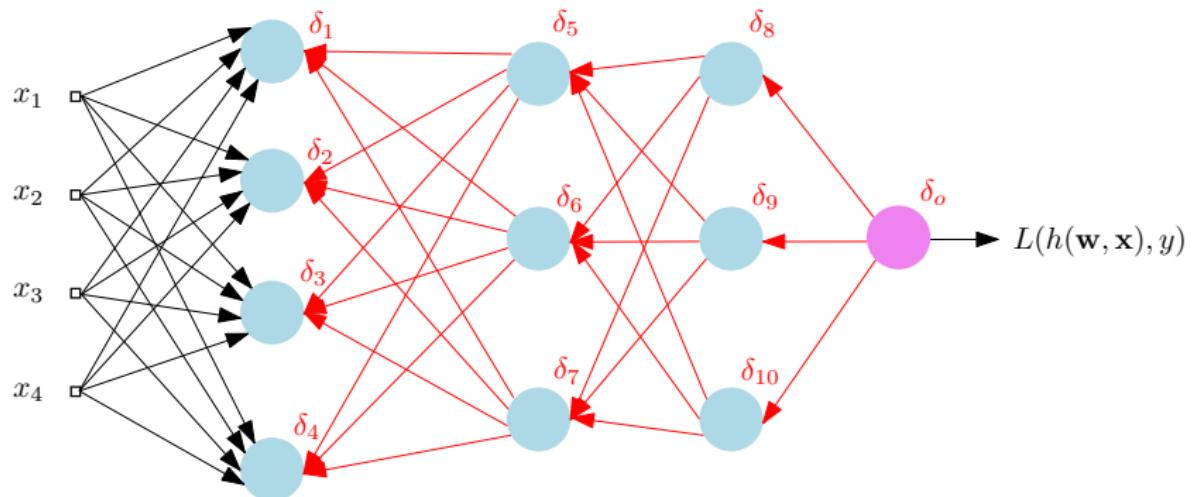
Backpropagation



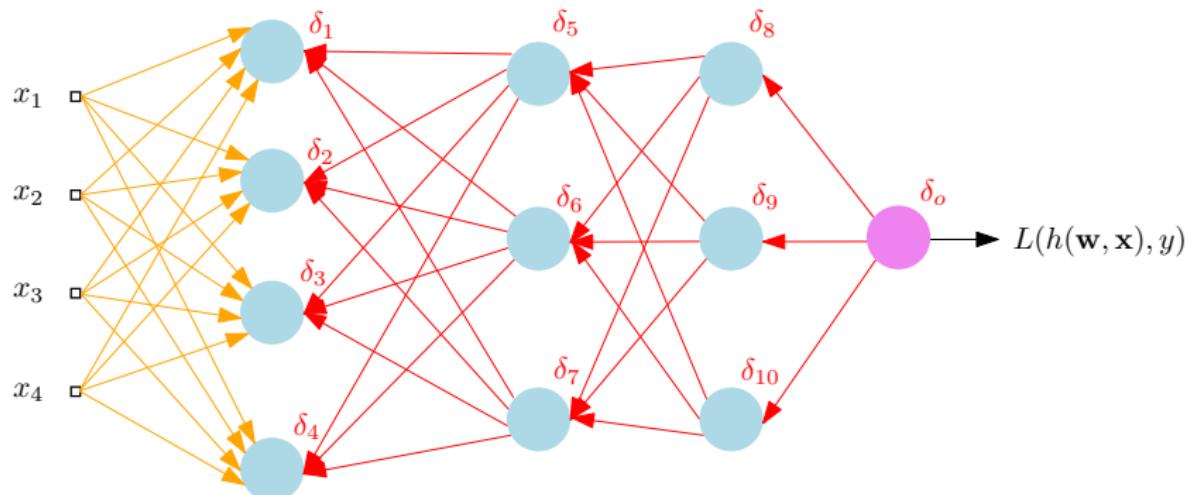
Backpropagation



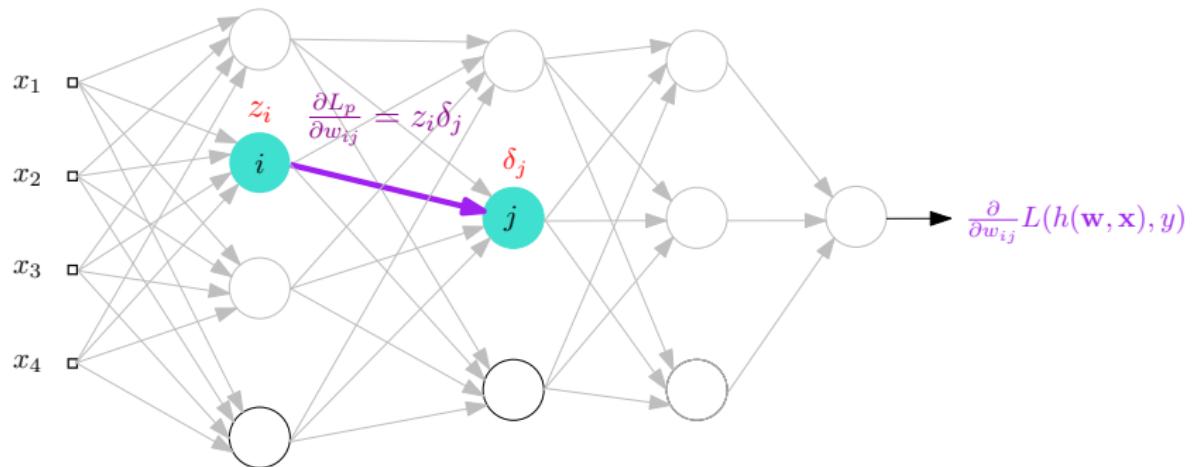
Backpropagation



Backpropagation



Actualización de los pesos



Backpropagation

- Procedimiento iterativo.

Backpropagation

- Procedimiento iterativo.
- Dados valores de los pesos de la red, calcular la salida $h(\mathbf{w}, \mathbf{x}_i)$ (paso **forward**).

Backpropagation

- Procedimiento iterativo.
- Dados valores de los pesos de la red, calcular la salida $h(\mathbf{w}, \mathbf{x}_i)$ (paso **forward**).
- Calcular los δ_j desde la salida hacia las capas anteriores.

Backpropagation

- Procedimiento iterativo.
- Dados valores de los pesos de la red, calcular la salida $h(\mathbf{w}, \mathbf{x}_i)$ (paso **forward**).
- Calcular los δ_j desde la salida hacia las capas anteriores. El error se **propaga** desde la salida hacia las capas anteriores (paso de **backpropagation**).

Backpropagation

- Procedimiento iterativo.
- Dados valores de los pesos de la red, calcular la salida $h(\mathbf{w}, \mathbf{x}_i)$ (paso **forward**).
- Calcular los δ_j desde la salida hacia las capas anteriores. El error se **propaga** desde la salida hacia las capas anteriores (paso de **backpropagation**).
- Actualizar los pesos.

Algoritmo de Backpropagation (on-line)

Incialize \mathbf{w}_0

Algoritmo de Backpropagation (on-line)

Incialize \mathbf{w}_0

repeat

Escoja (\mathbf{x}_p, y_p)

Algoritmo de Backpropagation (on-line)

Incialize \mathbf{w}_0

repeat

Escoja (\mathbf{x}_p, y_p)

Feed-forward {calcule los z_j }

Algoritmo de Backpropagation (on-line)

Incialize \mathbf{w}_0

repeat

Escoja (\mathbf{x}_p, y_p)

Feed-forward {calcule los z_j }

Back-prop {calcule los δ_j y $\nabla_{\mathbf{w}} L_p$ }

Algoritmo de Backpropagation (on-line)

Incialize \mathbf{w}_0

repeat

Escoja (\mathbf{x}_p, y_p)

Feed-forward {calcule los z_j }

Back-prop {calcule los δ_j y $\nabla_{\mathbf{w}} L_p$ }

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}} L_p|_{\mathbf{w}_k}$$

Algoritmo de Backpropagation (on-line)

Incialize \mathbf{w}_0

repeat

Escoja (\mathbf{x}_p, y_p)

Feed-forward {calcule los z_j }

Back-prop {calcule los δ_j y $\nabla_{\mathbf{w}} L_p$ }

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}} L_p|_{\mathbf{w}_k}$$

until Condición de terminación.

Algoritmo de Backpropagation (on-line)

Incialize \mathbf{w}_0

repeat

Escoja (\mathbf{x}_p, y_p)

Feed-forward {calcule los z_j }

Back-prop {calcule los δ_j y $\nabla_{\mathbf{w}} L_p$ }

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}} L_p|_{\mathbf{w}_k}$$

until Condición de terminación.

Eficiencia de Backpropagation

Eficiencia de Backpropagation

- Sea W el número de pesos en la red.

Eficiencia de Backpropagation

- Sea W el número de pesos en la red.
- Cálculo en pasos **forward** y de **backpropagation** toma $O(W)$ operaciones.

Eficiencia de Backpropagation

- Sea W el número de pesos en la red.
- Cálculo en pasos **forward** y de **backpropagation** toma $O(W)$ operaciones.
- Cálculo directo toma

Eficiencia de Backpropagation

- Sea W el número de pesos en la red.
- Cálculo en pasos **forward** y de **backpropagation** toma $O(W)$ operaciones.
- Cálculo directo toma $O(W^2)$ operaciones.

Eficiencia de Backpropagation

- Sea W el número de pesos en la red.
- Cálculo en pasos **forward** y de **backpropagation** toma $O(W)$ operaciones.
- Cálculo directo toma $O(W^2)$ operaciones.
- Multiplicar por n datos!

Backprop Matricial para red en capas

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i$

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

- ▶ $\mathbf{a}_n, \mathbf{z}_n$: neuronas en la capa n .

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

- ▶ $\mathbf{a}_n, \mathbf{z}_n$: neuronas en la capa n .
- ▶ $[\mathbf{z}'_n]_j = f'_j(a_j)$ en la capa n

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

- ▶ $\mathbf{a}_n, \mathbf{z}_n$: neuronas en la capa n .
- ▶ $[\mathbf{z}'_n]_j = f'_j(a_j)$ en la capa n
- ▶ \mathbf{W}_n : pesos en la capa n .

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

- ▶ $\mathbf{a}_n, \mathbf{z}_n$: neuronas en la capa n .
- ▶ $[\mathbf{z}'_n]_j = f'_j(a_j)$ en la capa n
- ▶ \mathbf{W}_n : pesos en la capa n .
- ▶ δ_n : “errores“ en capa n

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

- ▶ $\mathbf{a}_n, \mathbf{z}_n$: neuronas en la capa n .
- ▶ $[\mathbf{z}'_n]_j = f'_j(a_j)$ en la capa n
- ▶ \mathbf{W}_n : pesos en la capa n .
- ▶ $\boldsymbol{\delta}_n$: “errores“ en capa n
- ▶ Ecuaciones:

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

- ▶ $\mathbf{a}_n, \mathbf{z}_n$: neuronas en la capa n .
- ▶ $[\mathbf{z}'_n]_j = f'_j(a_j)$ en la capa n
- ▶ \mathbf{W}_n : pesos en la capa n .
- ▶ $\boldsymbol{\delta}_n$: “errores“ en capa n
- ▶ Ecuaciones:

- ★ $\mathbf{a}_n = \mathbf{W}_n \mathbf{z}_{n-1}, \mathbf{z}_n = f_S(\mathbf{a}_n)$

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

- ▶ $\mathbf{a}_n, \mathbf{z}_n$: neuronas en la capa n .
- ▶ $[\mathbf{z}'_n]_j = f'_j(a_j)$ en la capa n
- ▶ \mathbf{W}_n : pesos en la capa n .
- ▶ $\boldsymbol{\delta}_n$: “errores“ en capa n
- ▶ Ecuaciones:
 - ★ $\mathbf{a}_n = \mathbf{W}_n \mathbf{z}_{n-1}, \mathbf{z}_n = f_S(\mathbf{a}_n)$
 - ★ $\boldsymbol{\delta}_n = \mathbf{z}'_n \odot \mathbf{W}'_{n+1} \boldsymbol{\delta}_{n+1}$

Backprop Matricial para red en capas

- Ecuaciones de Backpropagation:

- ▶ $a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$
- ▶ $\delta_j = f'_j(a_j) \sum_k w_{jk} \delta_k$
- ▶ $\frac{\partial L}{\partial w_{ij}} = \delta_j z_i$

- Notación:

- ▶ $\mathbf{a}_n, \mathbf{z}_n$: neuronas en la capa n .
- ▶ $[\mathbf{z}'_n]_j = f'_j(a_j)$ en la capa n
- ▶ \mathbf{W}_n : pesos en la capa n .
- ▶ $\boldsymbol{\delta}_n$: “errores“ en capa n
- ▶ Ecuaciones:
 - ★ $\mathbf{a}_n = \mathbf{W}_n \mathbf{z}_{n-1}, \mathbf{z}_n = f_S(\mathbf{a}_n)$
 - ★ $\boldsymbol{\delta}_n = \mathbf{z}'_n \odot \mathbf{W}'_{n+1} \boldsymbol{\delta}_{n+1}$
 - ★ $\nabla L_n = \mathbf{z}_n \odot \boldsymbol{\delta}_{n+1}$

Momentum

- Tiene en cuenta información de gradiente local más la tendencia reciente en la superficie de error:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}} E|_{\mathbf{w}_k} + \eta (\mathbf{w}_{k-1} - \mathbf{w}_{k-2})$$

Momentum

- Tiene en cuenta información de gradiente local más la tendencia reciente en la superficie de error:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}} E|_{\mathbf{w}_k} + \eta (\mathbf{w}_{k-1} - \mathbf{w}_{k-2})$$

- En una región de la superficie de error con poca curvatura, y gradiente aproximadamente constante:

$$\begin{aligned}\Delta \mathbf{w} &\approx -\mu \nabla_{\mathbf{w}} E(1 + \eta + \eta^2 + \dots) \\ &= -\frac{\mu}{1 - \eta} \nabla_{\mathbf{w}} E\end{aligned}$$

Momentum

- Tiene en cuenta información de gradiente local más la tendencia reciente en la superficie de error:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}} E|_{\mathbf{w}_k} + \eta (\mathbf{w}_{k-1} - \mathbf{w}_{k-2})$$

- En una región de la superficie de error con poca curvatura, y gradiente aproximadamente constante:

$$\begin{aligned}\Delta \mathbf{w} &\approx -\mu \nabla_{\mathbf{w}} E(1 + \eta + \eta^2 + \dots) \\ &= -\frac{\mu}{1 - \eta} \nabla_{\mathbf{w}} E\end{aligned}$$

- En cambio, en una región de curvatura grande en la cual el descenso de gradiente es oscilatorio, las contribuciones sucesivas del término de momentum tienden a cancelarse, resultando en una tasa de aprendizaje efectiva cercana a μ .

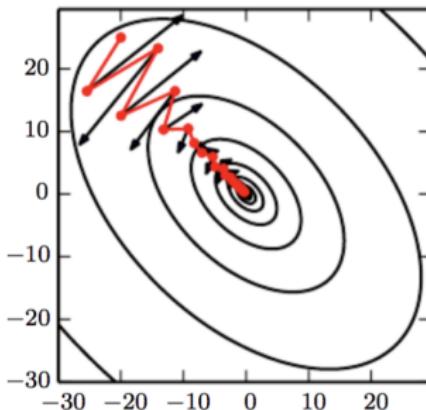


Figure 8.5: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix. The red path cutting across the contours indicates the path followed by the momentum learning rule as it minimizes this function. At each step along the way, we draw an arrow indicating the step that gradient descent would take at that point. We can see that a poorly conditioned quadratic objective looks like a long, narrow valley or canyon with steep sides. Momentum correctly traverses the canyon lengthwise, while gradient steps waste time moving back and forth across the narrow axis of the canyon. Compare also figure 4.6, which shows the behavior of gradient descent without momentum.

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter θ

Require: Small constant δ , perhaps 10^{-7} , for numerical stability

Initialize gradient accumulation variable $r = \mathbf{0}$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$.

 Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$.

 Compute update: $\Delta \theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$. (Division and square root applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta \theta$.

end while

$$\eta_k \propto \frac{1}{\sqrt{\sum (\text{valores pasados})_k^2}}$$

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ , decay rate ρ .

Require: Initial parameter θ

Require: Small constant δ , usually 10^{-6} , used to stabilize division by small numbers.

Initialize accumulation variables $r = 0$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Accumulate squared gradient: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

 Compute parameter update: $\Delta \theta = -\frac{\epsilon}{\sqrt{\delta + \mathbf{r}}} \odot \mathbf{g}$. ($\frac{1}{\sqrt{\delta + \mathbf{r}}}$ applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta \theta$

end while

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1]$.
(Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default:
 10^{-8})

Require: Initial parameters θ

Initialize 1st and 2nd moment variables $s = \mathbf{0}$, $r = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with
 corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

$t \leftarrow t + 1$

 Update biased first moment estimate: $\hat{s} \leftarrow \rho_1 s + (1 - \rho_1) \mathbf{g}$

 Update biased second moment estimate: $\hat{r} \leftarrow \rho_2 r + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

 Correct bias in first moment: $\hat{s} \leftarrow \frac{\hat{s}}{1 - \rho_1^t}$

 Correct bias in second moment: $\hat{r} \leftarrow \frac{\hat{r}}{1 - \rho_2^t}$

 Compute update: $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$ (operations applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while
