

Aproximación de funciones con redes neuronales

Fernando Lozano

Universidad de los Andes

14 de marzo de 2023



SARSA

Incialice $Q(s, a)$

repeat

 Incialice s

 Escoja a de $\mathcal{A}(s)$, de acuerdo a Q (ϵ – greedy)

repeat

 Tome acción a , observe r, s' .

 Escoja a' de $\mathcal{A}(s')$, de acuerdo a Q (ϵ – greedy)

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma Q(s', a') - Q(s, a)]$$

$s \leftarrow s', a \leftarrow a'$

until s es terminal

until ∞

Q learning

Incialice $Q(s, a)$

repeat

 Incialice s

repeat

 Escoja a de $\mathcal{A}(s)$, de acuerdo a Q (ϵ – greedy)

 Tome acción a , observe r, s' .

$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_a Q(s', a') - Q(s, a)]$$

$$s \leftarrow s'$$

until s es terminal

until ∞

RL tabular

- Almacenar/actualizar $Q(s, a)$ en arreglo en memoria.

RL tabular

- Almacenar/actualizar $Q(s, a)$ en arreglo en memoria.
- Excepto para problemas muy pequeños $|\mathcal{S}| \times |\mathcal{A}| \ggg$

RL tabular

- Almacenar/actualizar $Q(s, a)$ en arreglo en memoria.
- Excepto para problemas muy pequeños $|\mathcal{S}| \times |\mathcal{A}| \ggg$
 - ▶ Arreglo no cabe en memoria.

RL tabular

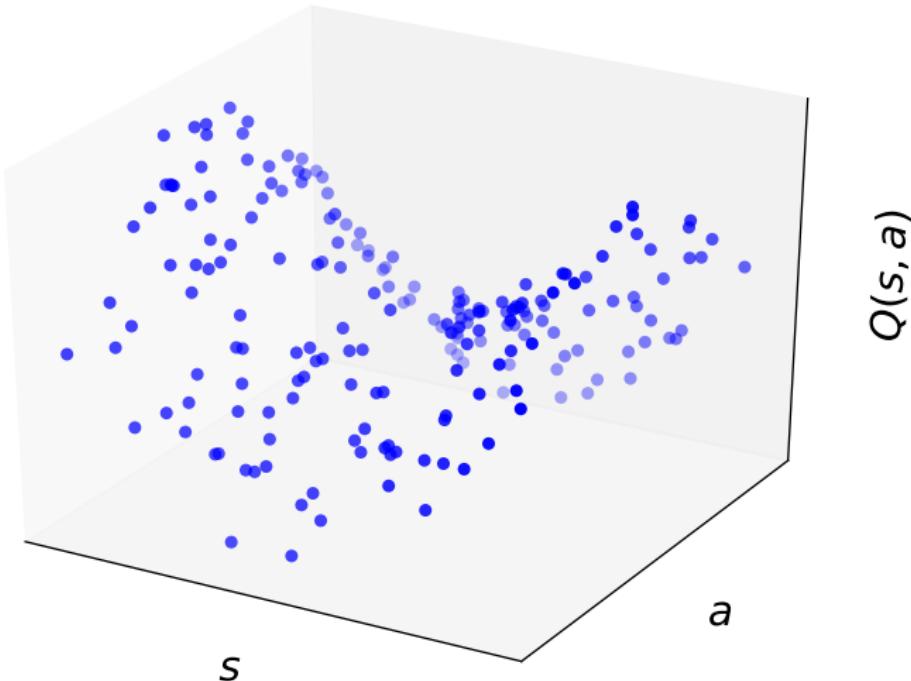
- Almacenar/actualizar $Q(s, a)$ en arreglo en memoria.
- Excepto para problemas muy pequeños $|\mathcal{S}| \times |\mathcal{A}| \ggg$
 - ▶ Arreglo no cabe en memoria.
 - ▶ Sólo una fracción muy pequeña de las entradas se actualiza.

RL tabular

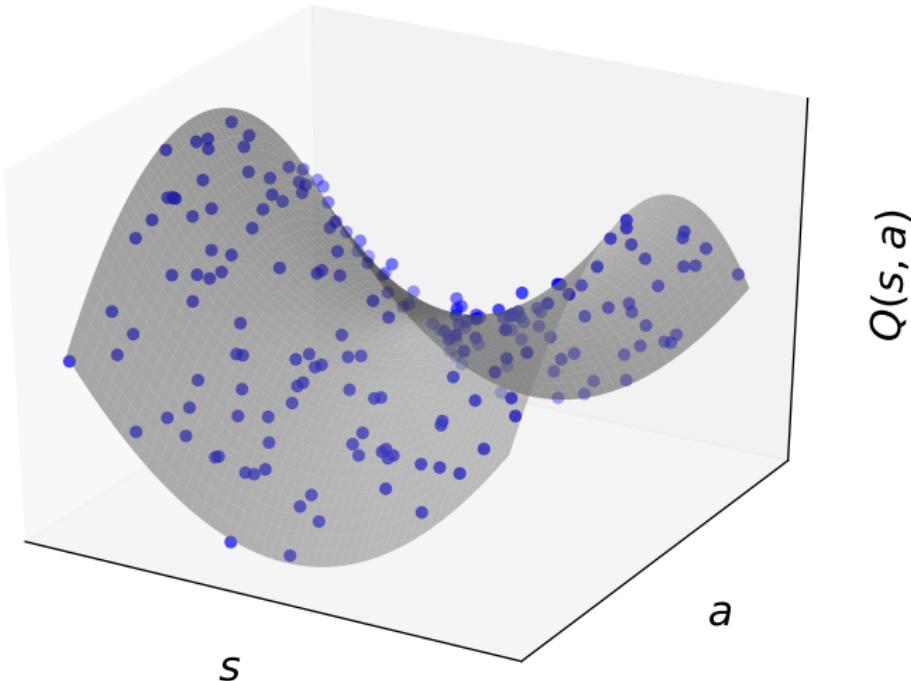
- Almacenar/actualizar $Q(s, a)$ en arreglo en memoria.
- Excepto para problemas muy pequeños $|\mathcal{S}| \times |\mathcal{A}| \ggg$
 - ▶ Arreglo no cabe en memoria.
 - ▶ Sólo una fracción muy pequeña de las entradas se actualiza.
- En cada iteración sólo se actualiza Q para un par s, a .

- Almacenar/actualizar $Q(s, a)$ en arreglo en memoria.
- Excepto para problemas muy pequeños $|\mathcal{S}| \times |\mathcal{A}| \ggg$
 - ▶ Arreglo no cabe en memoria.
 - ▶ Sólo una fracción muy pequeña de las entradas se actualiza.
- En cada iteración sólo se actualiza Q para un par s, a .
- Convergencia de algoritmos requiere visitar frecuentemente todos los estados.

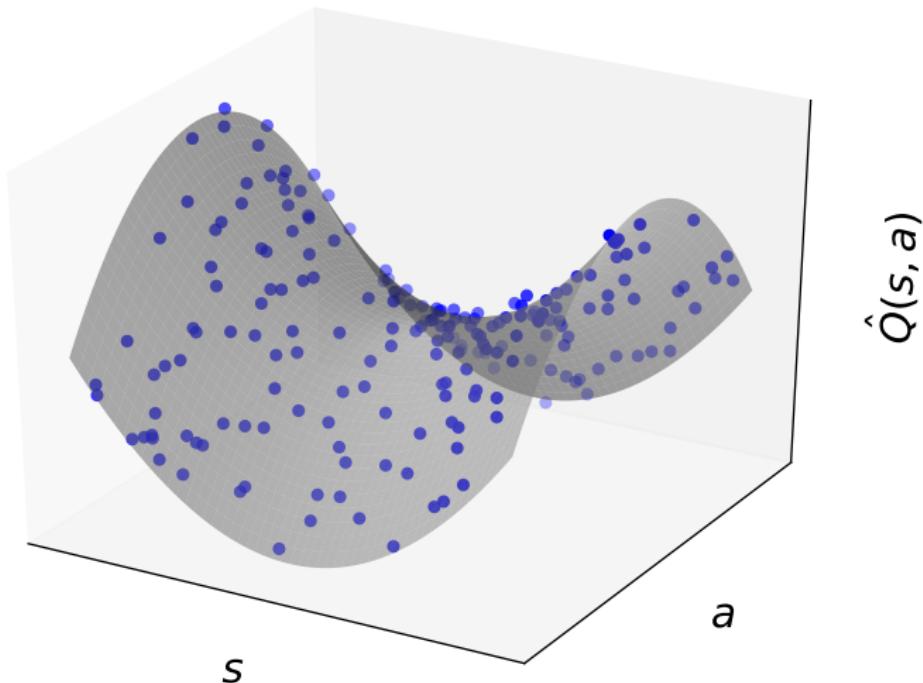
Solución: Aproximar $Q(s, a)$



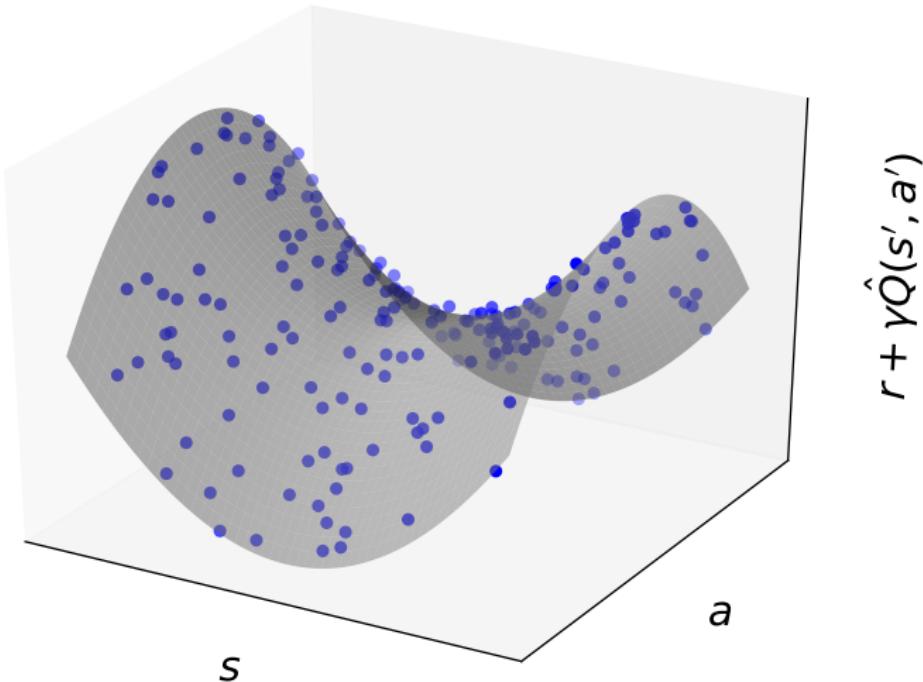
Solución: Aproximar $Q(s, a)$



Solución: Aproximar $Q(s, a)$



Solución: Aproximar $Q(s, a)$



Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.

Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.

Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.
 - ▶ Ajustar función a los datos hallando parámetros \mathbf{w} .

Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.
 - ▶ Ajustar función a los datos hallando parámetros \mathbf{w} .
 - ▶ Minimizar función de error en los datos.

Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.
 - ▶ Ajustar función a los datos hallando parámetros \mathbf{w} .
 - ▶ Minimizar función de error en los datos.
 - ▶ Selección de modelo.

Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.
 - ▶ Ajustar función a los datos hallando parámetros \mathbf{w} .
 - ▶ Minimizar función de error en los datos.
 - ▶ Selección de modelo.
- Diferencias en RL:

Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.
 - ▶ Ajustar función a los datos hallando parámetros \mathbf{w} .
 - ▶ Minimizar función de error en los datos.
 - ▶ Selección de modelo.
- Diferencias en RL:
 - ▶ Datos $\{\mathbf{x}_i, y_i\}_{i=1}^n$ no son i.i.d.

Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.
 - ▶ Ajustar función a los datos hallando parámetros \mathbf{w} .
 - ▶ Minimizar función de error en los datos.
 - ▶ Selección de modelo.
- Diferencias en RL:
 - ▶ Datos $\{\mathbf{x}_i, y_i\}_{i=1}^n$ no son i.i.d.
 - ▶ y_i son valores estimados (muestras, bootstrap).

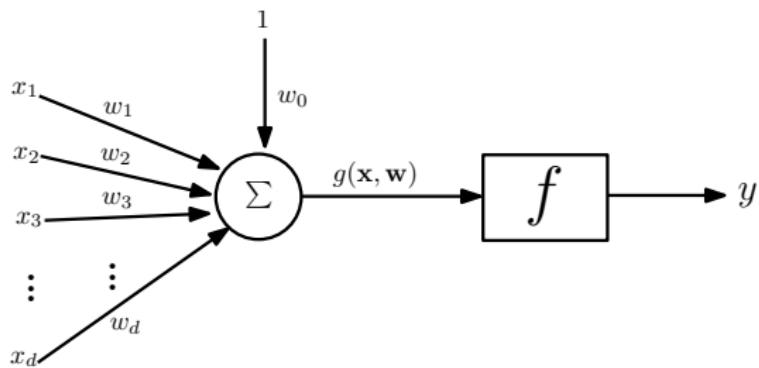
Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.
 - ▶ Ajustar función a los datos hallando parámetros \mathbf{w} .
 - ▶ Minimizar función de error en los datos.
 - ▶ Selección de modelo.
- Diferencias en RL:
 - ▶ Datos $\{\mathbf{x}_i, y_i\}_{i=1}^n$ no son i.i.d.
 - ▶ y_i son valores estimados (muestras, bootstrap).
 - ▶ y_i no son estacionarios.

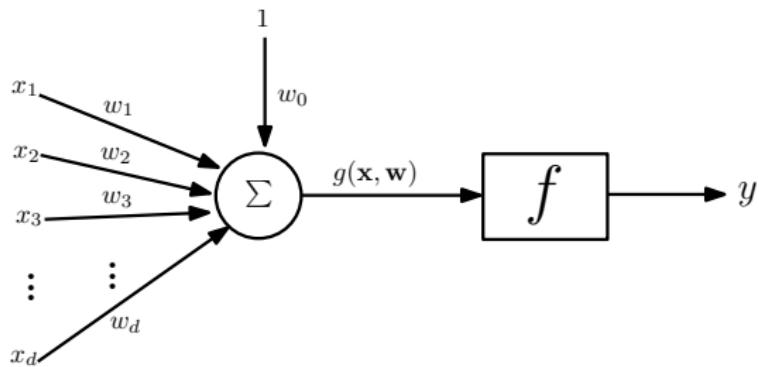
Aproximación de funciones

- Construir un modelo (función) que capture relación entre una entrada \mathbf{x} y una salida y a partir de observaciones $\{\mathbf{x}_i, y_i\}_{i=1}^n$.
 - ▶ Clase de funciones paramétrica $h(\mathbf{x}, \mathbf{w})$.
 - ▶ Ajustar función a los datos hallando parámetros \mathbf{w} .
 - ▶ Minimizar función de error en los datos.
 - ▶ Selección de modelo.
- Diferencias en RL:
 - ▶ Datos $\{\mathbf{x}_i, y_i\}_{i=1}^n$ no son i.i.d.
 - ▶ y_i son valores estimados (muestras, bootstrap).
 - ▶ y_i no son estacionarios.

Modelo de una neurona

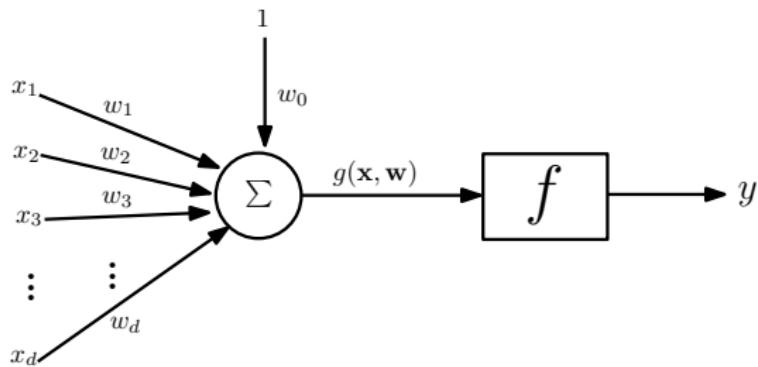


Modelo de una neurona



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

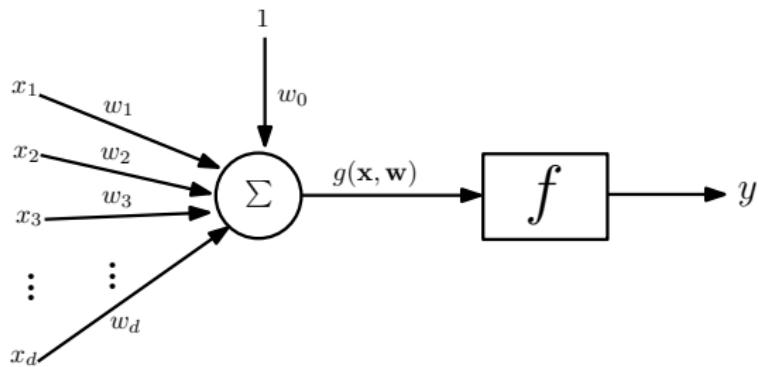
Modelo de una neurona



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

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

Modelo de una neurona

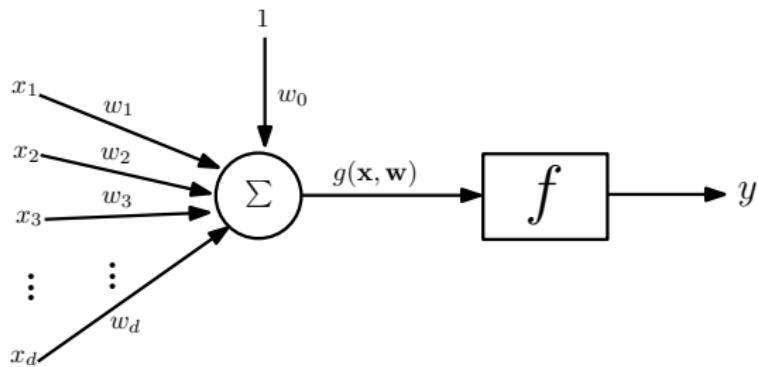


$$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_d]^T$$

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

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

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

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

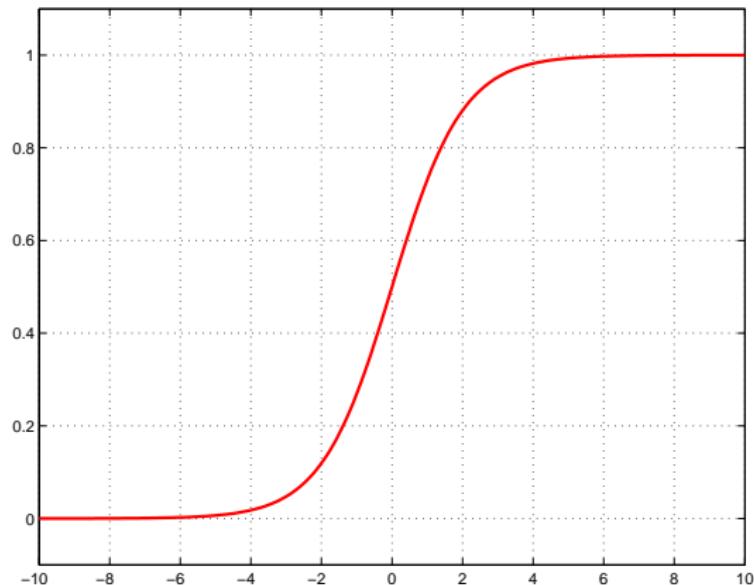
Vectores extendidos

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

Vectores extendidos

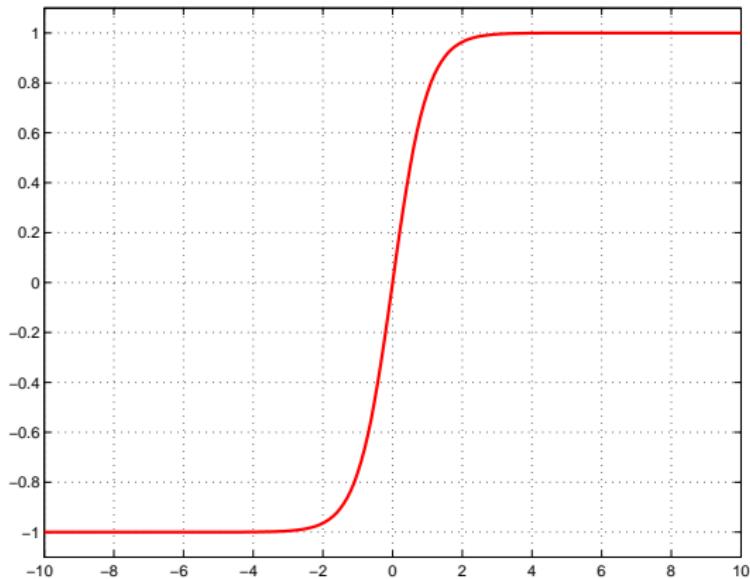
$$\begin{aligned}\tilde{\mathbf{x}} &= [1 \quad x_1 \quad x_2 \quad \dots \quad x_d]^T \\ \tilde{\mathbf{w}} &= [w_0 \quad w_1 \quad w_2 \quad \dots \quad w_d]^T \\ g(\mathbf{x}) &= \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} \\ u(\mathbf{x}) &= f(g(\mathbf{x})) = f(\tilde{\mathbf{w}}^T \tilde{\mathbf{x}})\end{aligned}$$

Activación sigmoidal



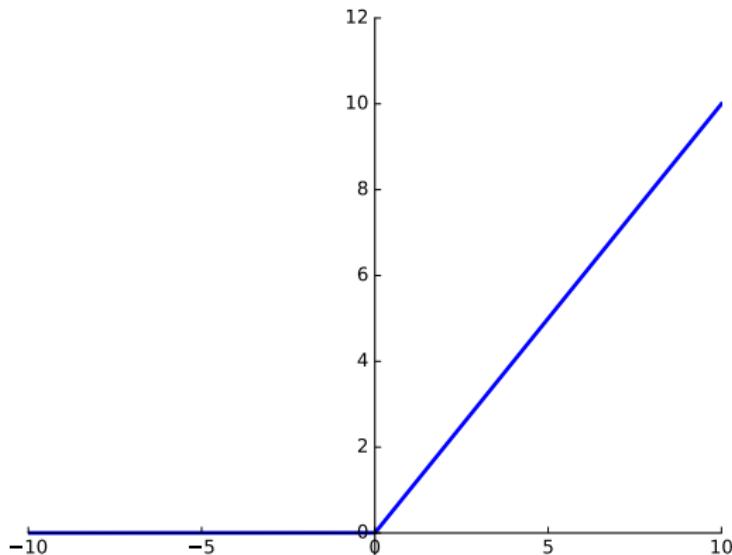
$$f_s(z) = \frac{1}{1 + e^{-\beta z}}$$

Tangente hiperbólica



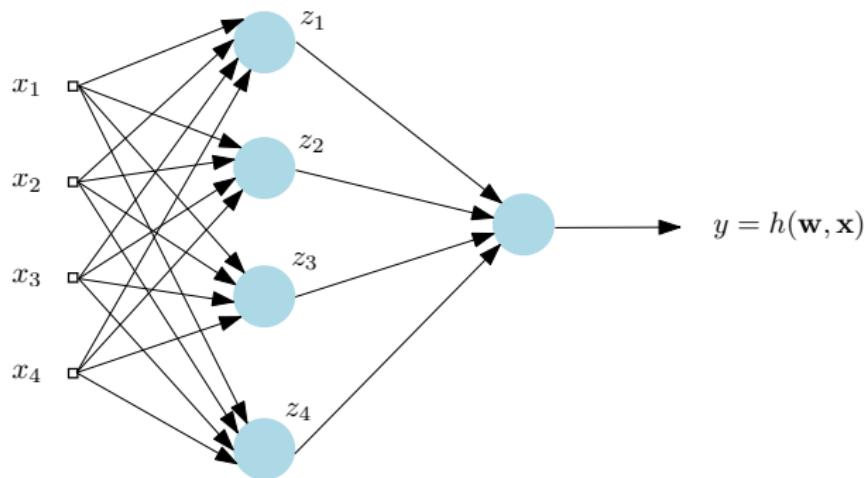
$$f_{TH}(z) = \tanh(z)$$

ReLU

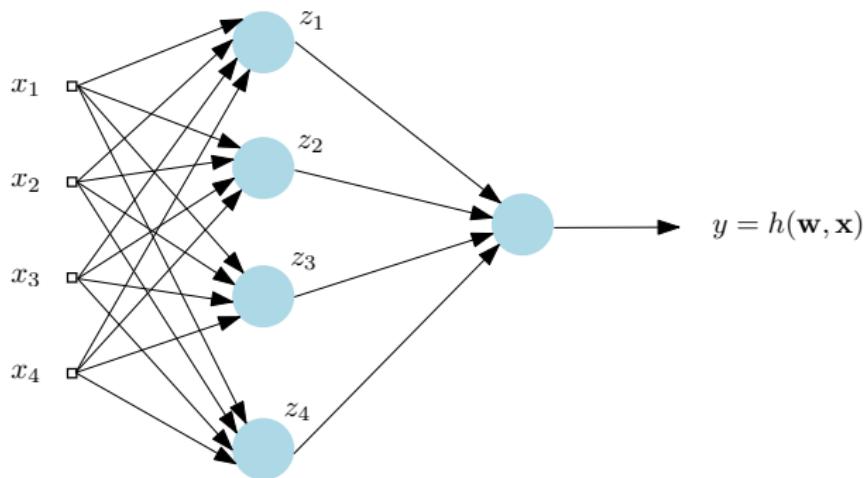


$$r(z) = \max \{0, z\}$$

Red neuronal de una capa

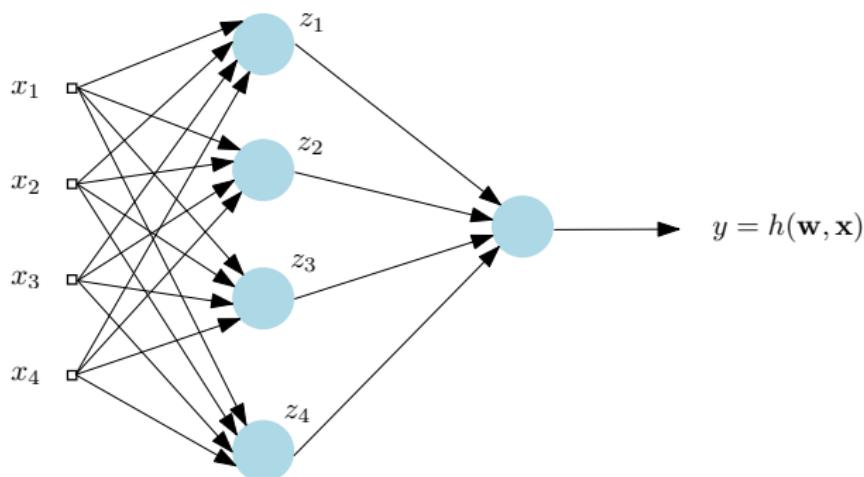


Red neuronal de una capa



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

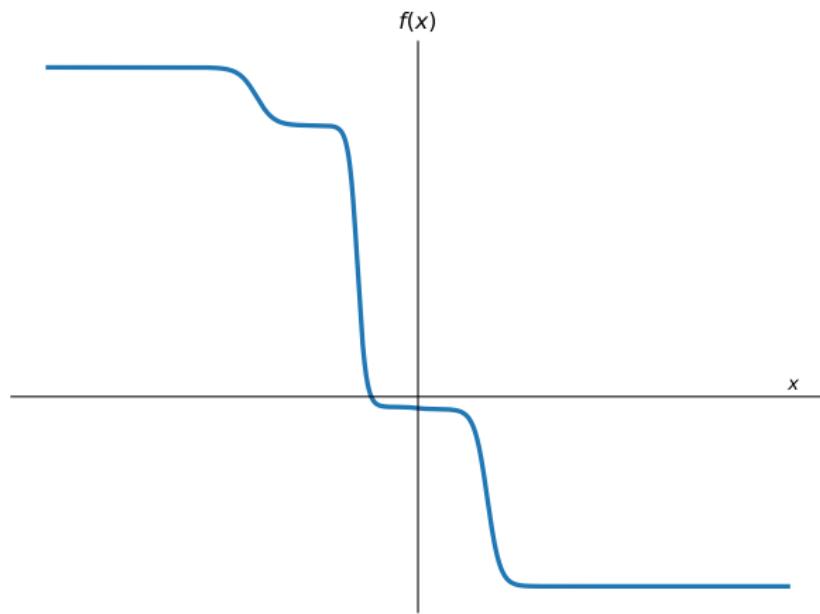
Red neuronal de una capa



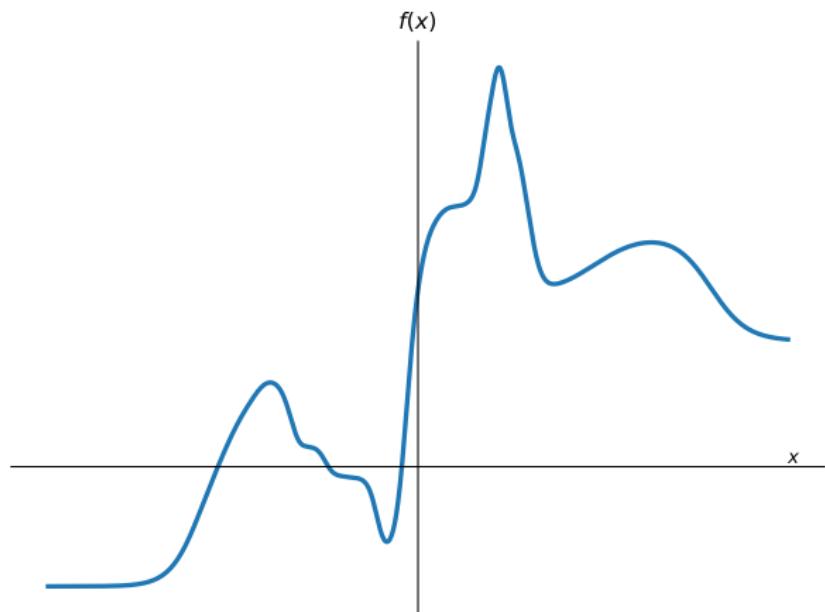
$$y = h(\mathbf{x}) = f_o \left(a_0 + \sum_{k=1}^N a_k f_k (\mathbf{w}_k^T \mathbf{x}) \right)$$

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

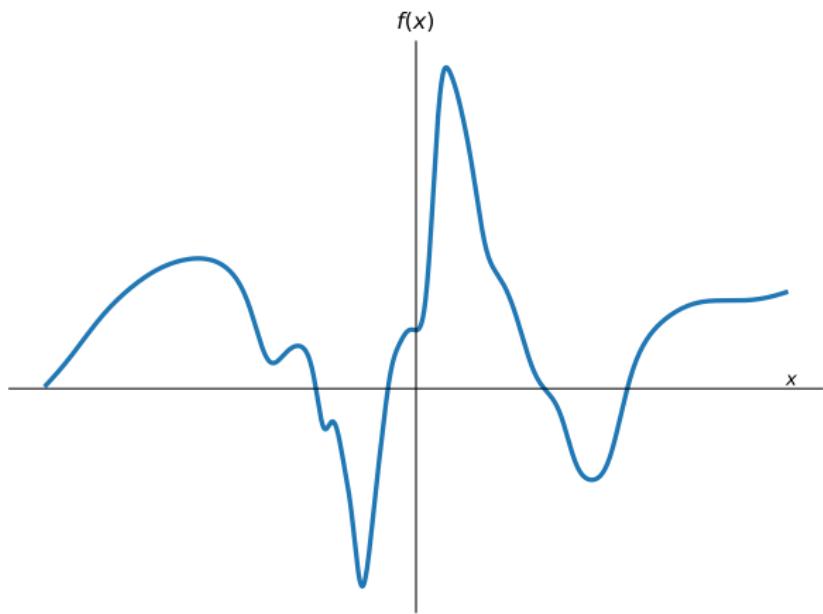
Sigmoidal, N=5



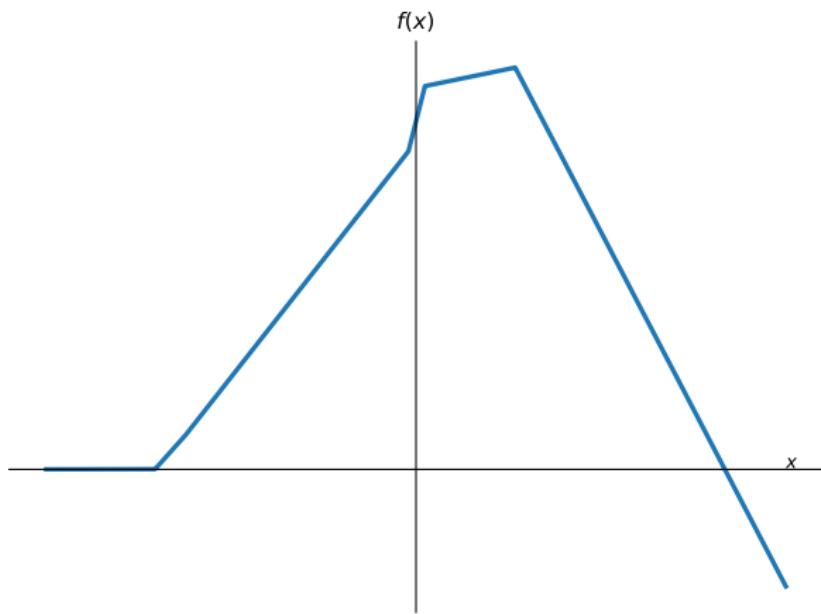
Sigmoidal, N=25



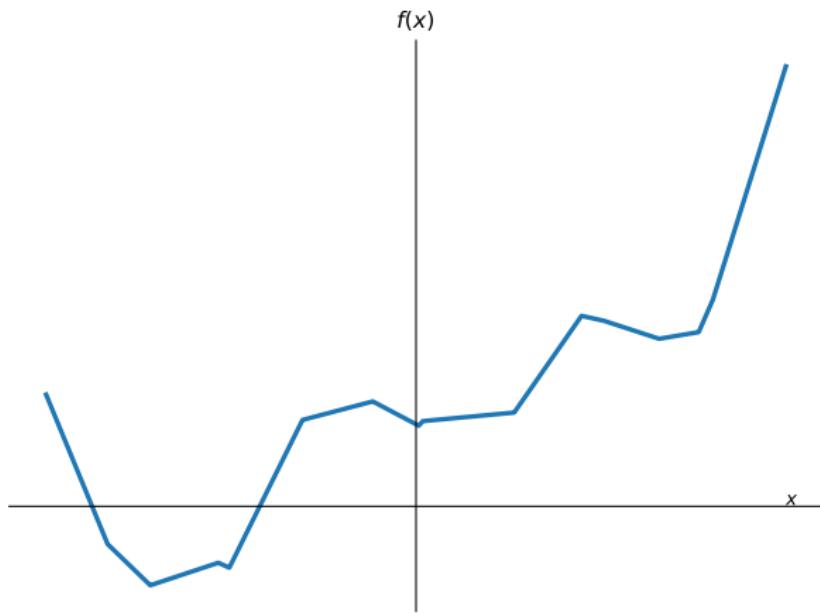
Sigmoidal, N=100



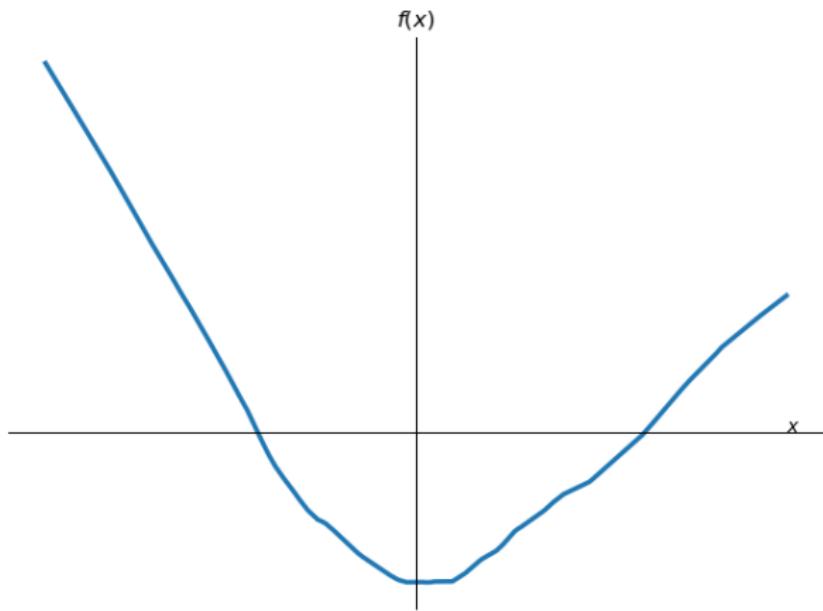
ReLU, N=5



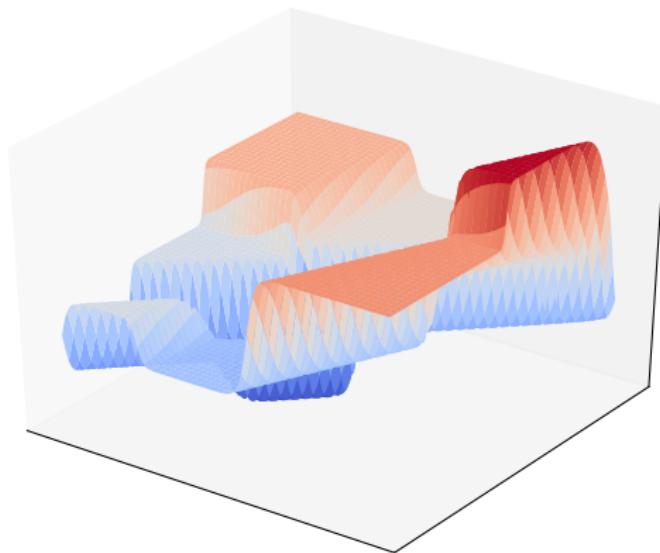
ReLU, N=25



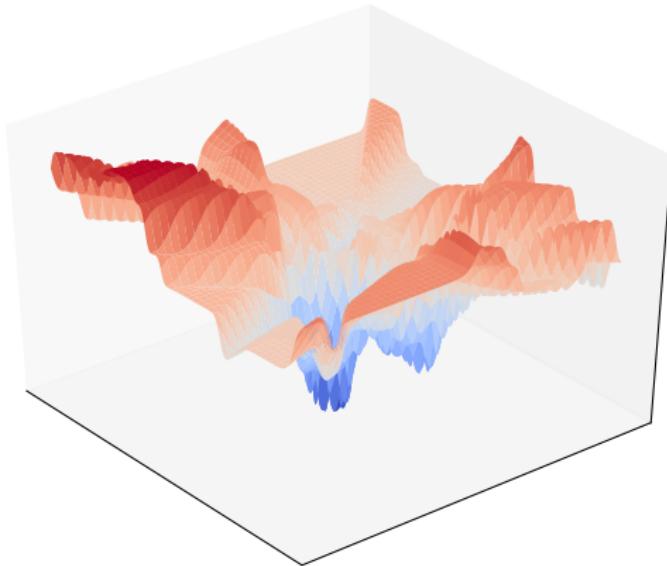
ReLU, N=100



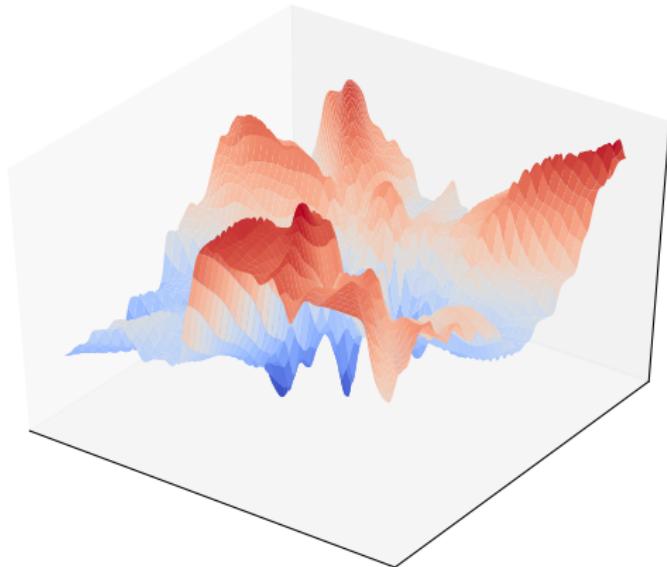
Sigmoidal, N=5



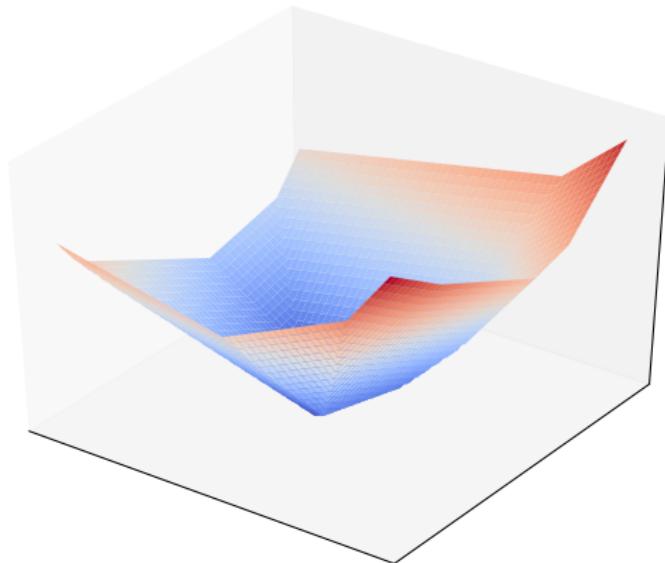
Sigmoidal, N=25



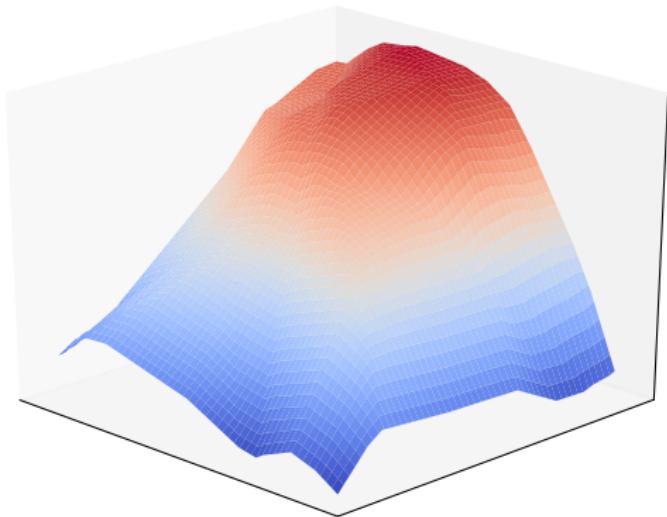
Sigmoidal, N=100



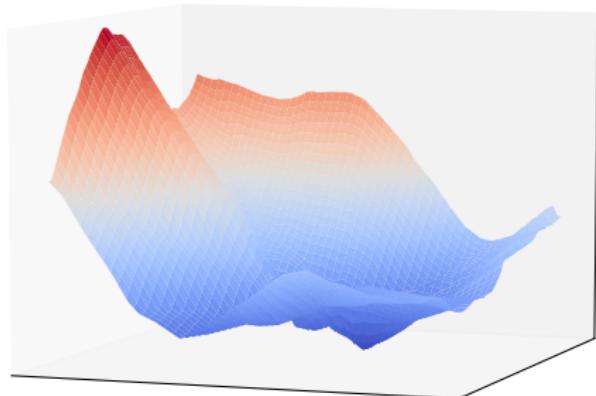
ReLU, N=5



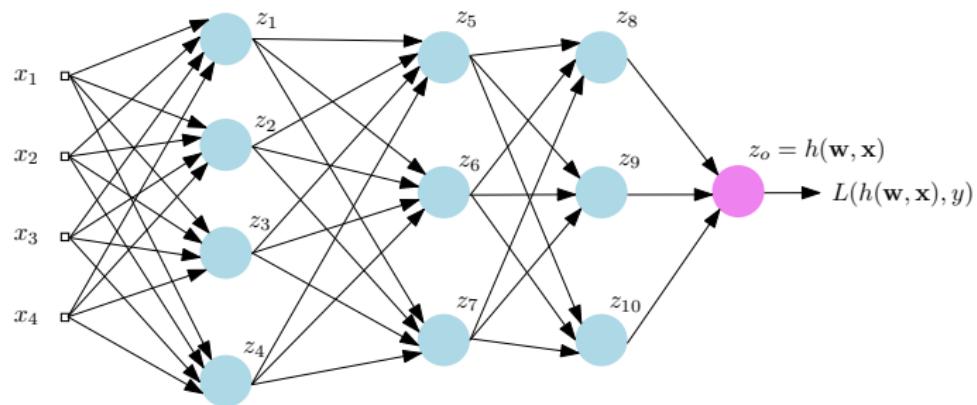
ReLU, N=25



ReLU, N=100



Arquitectura en Capas



Ajuste a los datos

Ajuste a los datos

- Minimizar función de error:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (y_i - h(\mathbf{w}, \mathbf{x}_i))^2$$

Ajuste a los datos

- Minimizar función de error:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (y_i - h(\mathbf{w}, \mathbf{x}_i))^2$$

- Descenso de gradiente

Descenso de Gradiente (GD)

Incialice \mathbf{w}_0

Descenso de Gradiente (GD)

Incialice \mathbf{w}_0

repeat

Descenso de Gradiente (GD)

Incialice \mathbf{w}_0

repeat

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \nabla_{\mathbf{w}} E(\mathbf{w}_k)$$

Descenso de Gradiente (GD)

Incialice \mathbf{w}_0

repeat

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \nabla_{\mathbf{w}} E(\mathbf{w}_k)$$

until Condición de terminación.

Descenso de Gradiente (GD)

Incialice \mathbf{w}_0

repeat

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \nabla_{\mathbf{w}} E(\mathbf{w}_k)$$

until Condición de terminación.

donde

- $\nabla_{\mathbf{w}} E(\mathbf{w})$ es el gradiente:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E(\mathbf{w})}{\partial w_1} \\ \frac{\partial E(\mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial E(\mathbf{w})}{\partial w_L} \end{bmatrix}$$

Descenso de Gradiente (GD)

Incialice \mathbf{w}_0

repeat

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta_k \nabla_{\mathbf{w}} E(\mathbf{w}_k)$$

until Condición de terminación.

donde

- $\nabla_{\mathbf{w}} E(\mathbf{w})$ es el gradiente:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E(\mathbf{w})}{\partial w_1} \\ \frac{\partial E(\mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial E(\mathbf{w})}{\partial w_L} \end{bmatrix}$$

- η_k es la **tasa de aprendizaje**.

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n \nabla_{\mathbf{w}} [(y_i - h(\mathbf{w}, \mathbf{x}_i))^2]$$

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n \nabla_{\mathbf{w}} [(y_i - h(\mathbf{w}, \mathbf{x}_i))^2] = \frac{1}{2} \sum_{i=1}^n \mathbf{g}_i$$

- Un término de gradiente por cada dato.

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n \nabla_{\mathbf{w}} [(y_i - h(\mathbf{w}, \mathbf{x}_i))^2] = \frac{1}{2} \sum_{i=1}^n \mathbf{g}_i$$

- Un término de gradiente por cada dato.
- Versión **en línea** o **estocástica**:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \frac{1}{2} \mathbf{g}_i$$

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n \nabla_{\mathbf{w}} [(y_i - h(\mathbf{w}, \mathbf{x}_i))^2] = \frac{1}{2} \sum_{i=1}^n \mathbf{g}_i$$

- Un término de gradiente por cada dato.
- Versión **en línea** o **estocástica**:

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \frac{1}{2} \mathbf{g}_i$$

- Versión **mini-batch**

$$\nabla_{\mathbf{w}} E(\mathbf{w}) \approx \frac{1}{2} \sum_{i : (\mathbf{x}_i, y_i) \in B} \mathbf{g}_i$$

donde B es un subconjunto pequeño de los datos.

Backpropagation

Backpropagation

- Descenso de gradiente para redes neuronales.

Backpropagation

- Descenso de gradiente para redes neuronales.
- Cálculo eficiente del gradiente en dos pasos:
 - ▶ Forward pass.
 - ▶ Backprop.

Backpropagation

- Descenso de gradiente para redes neuronales.
- Cálculo eficiente del gradiente en dos pasos:
 - ▶ Forward pass.
 - ▶ Backprop.
- Consideramos neuronas:
 - ▶ Activación sigmoidal:

$$f_s(z) \Rightarrow f'_s(z) = z(1 - z)$$

Backpropagation

- Descenso de gradiente para redes neuronales.
- Cálculo eficiente del gradiente en dos pasos:
 - ▶ Forward pass.
 - ▶ Backprop.
- Consideramos neuronas:
 - ▶ Activación sigmoidal:

$$f_s(z) \Rightarrow f'_s(z) = z(1 - z)$$

- ▶ Activación ReLu:

$$f_r(z) \Rightarrow f'_r(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

Backpropagation

- Descenso de gradiente para redes neuronales.
- Cálculo eficiente del gradiente en dos pasos:
 - ▶ Forward pass.
 - ▶ Backprop.
- Consideramos neuronas:
 - ▶ Activación sigmoidal:

$$f_s(z) \Rightarrow f'_s(z) = z(1 - z)$$

- ▶ Activación ReLu:

$$f_r(z) \Rightarrow f'_r(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

- ▶ Activación lineal (en la salida):

$$f_l(z) \Rightarrow f'_l(z) = 1$$

Forward pass

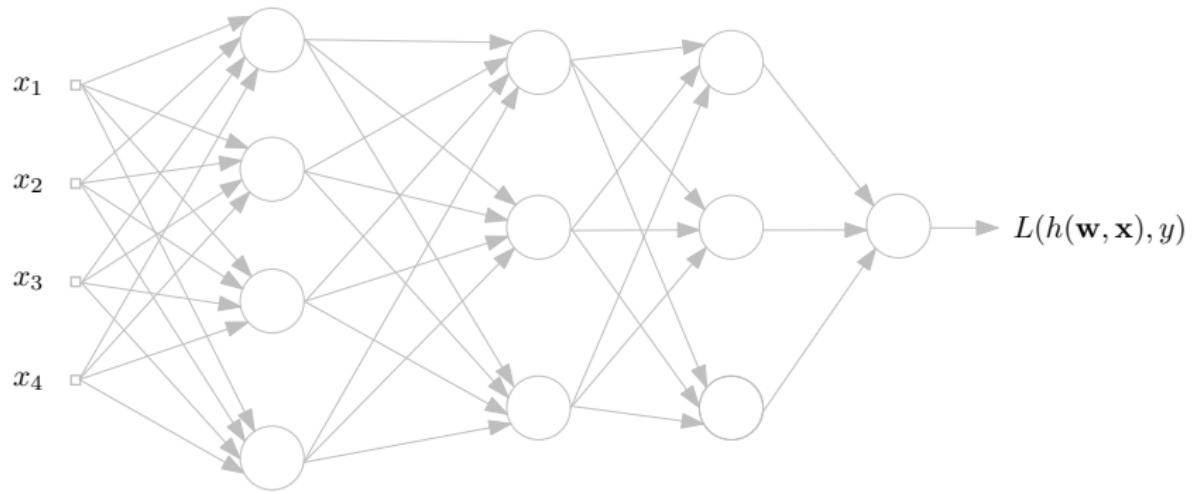
$$a_j = \sum_i w_{ij} z_i$$

Forward pass

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

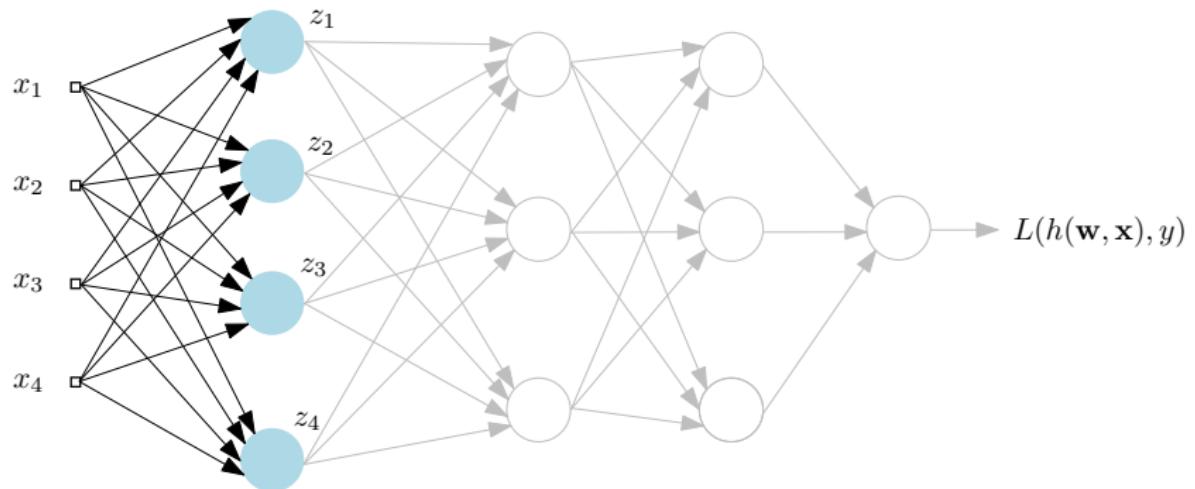
Forward pass

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



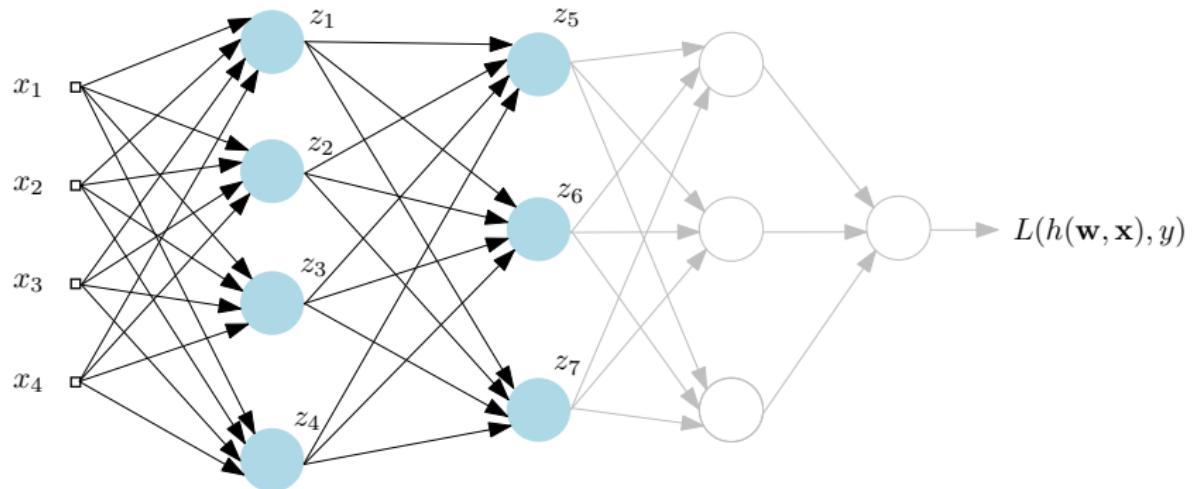
Forward pass

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



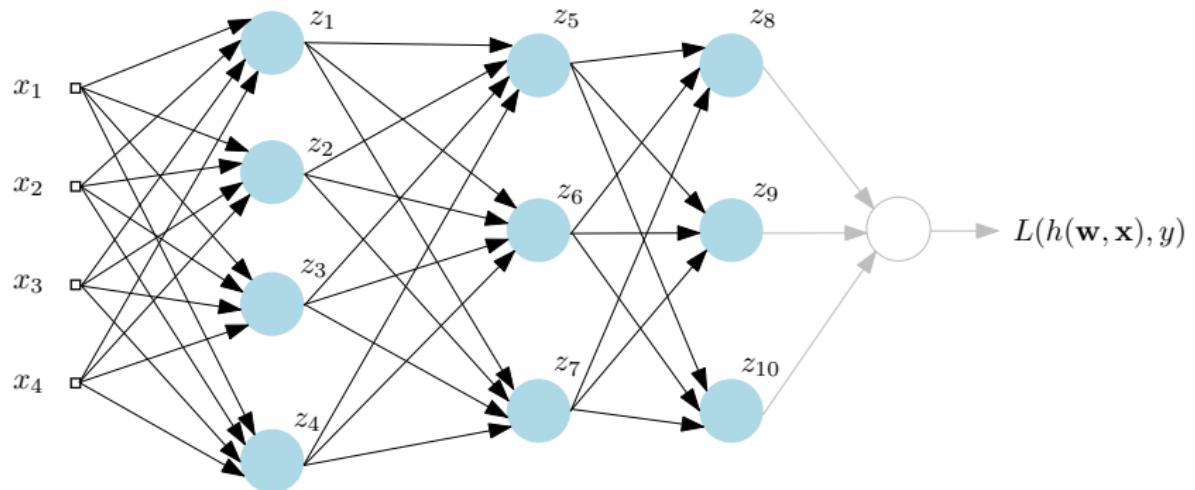
Forward pass

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



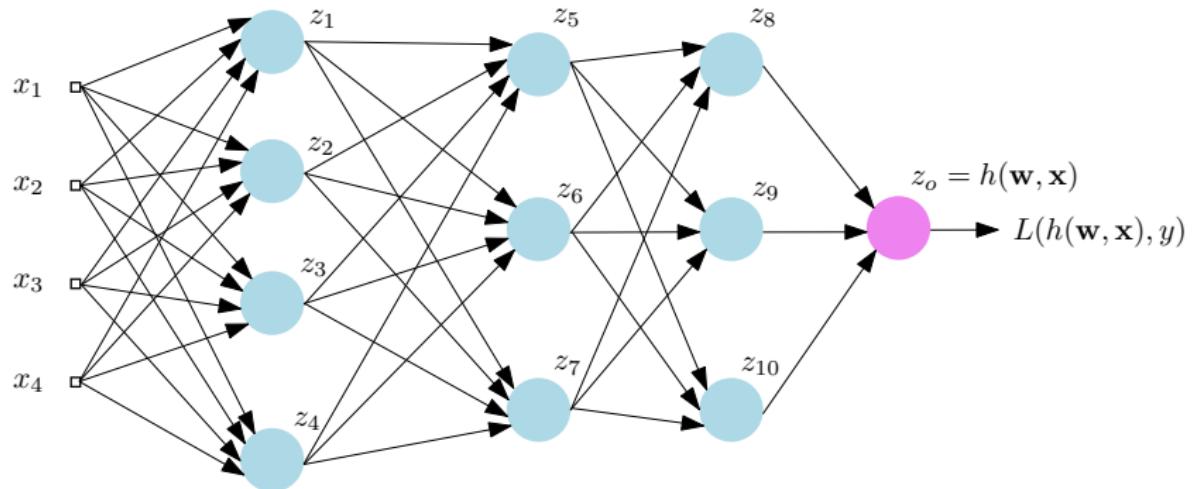
Forward pass

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

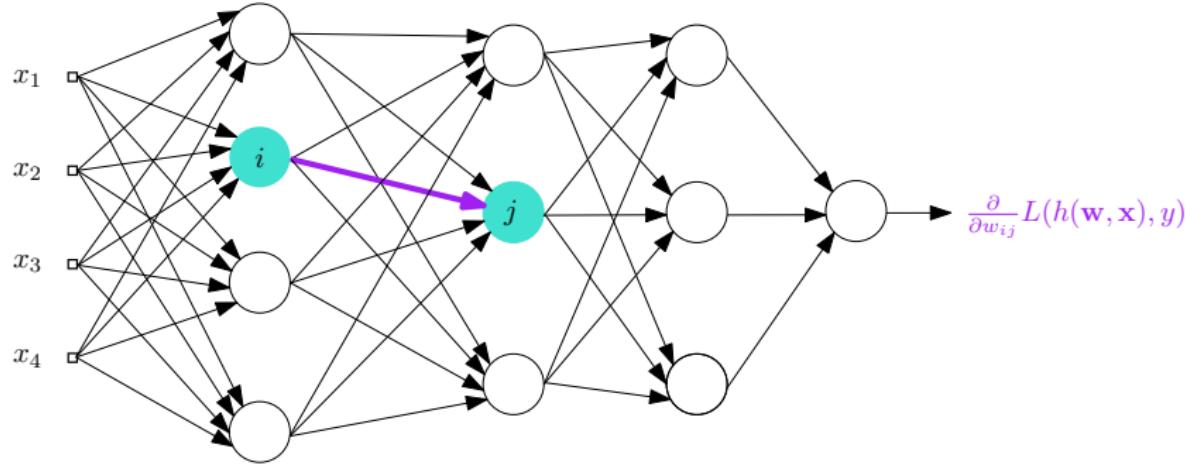


Forward pass

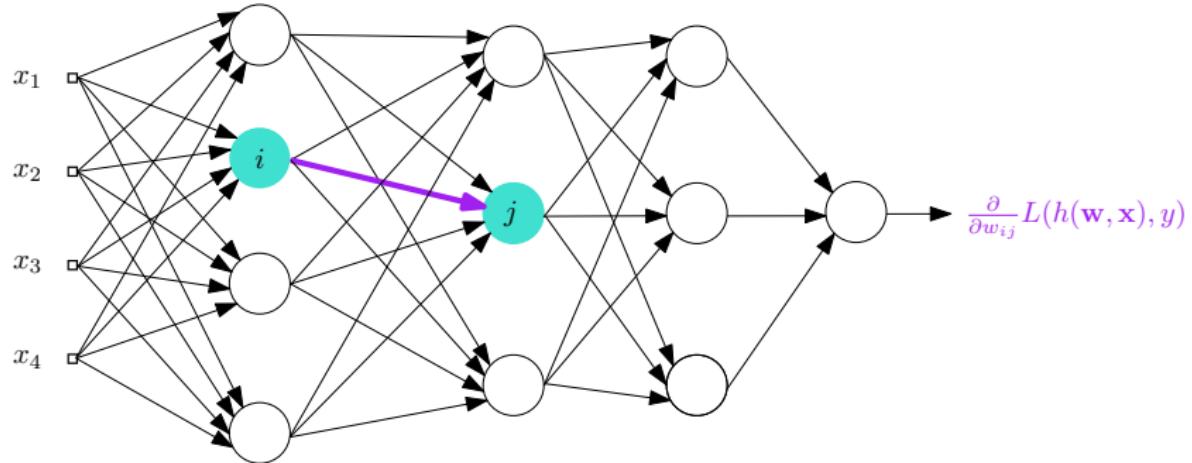
$$a_j = \sum_i w_{ij} z_i \quad z_j = f_j(a_j) \quad z_o = f_o(a_o)$$



Cálculo del gradiente

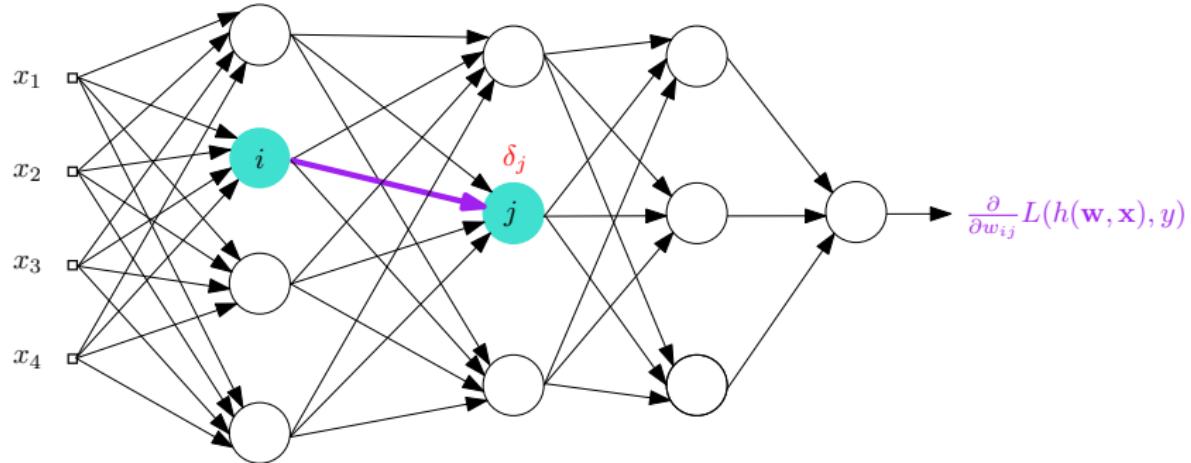


Cálculo del gradiente



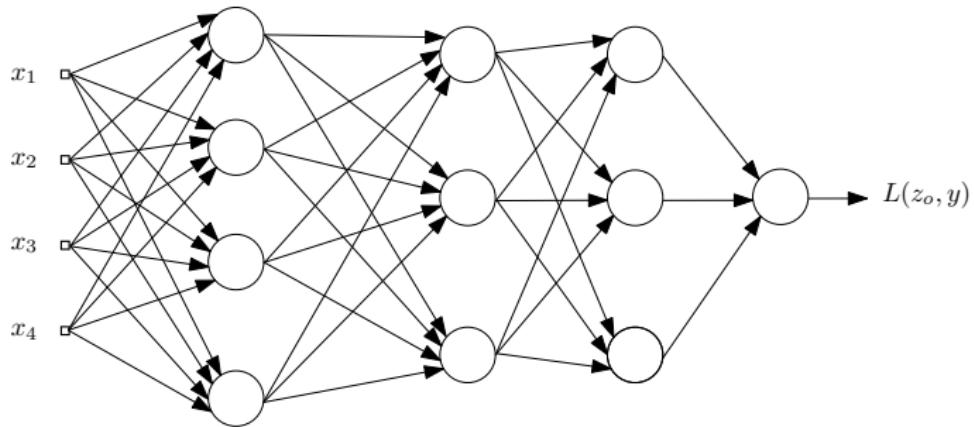
$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}$$

Cálculo del gradiente

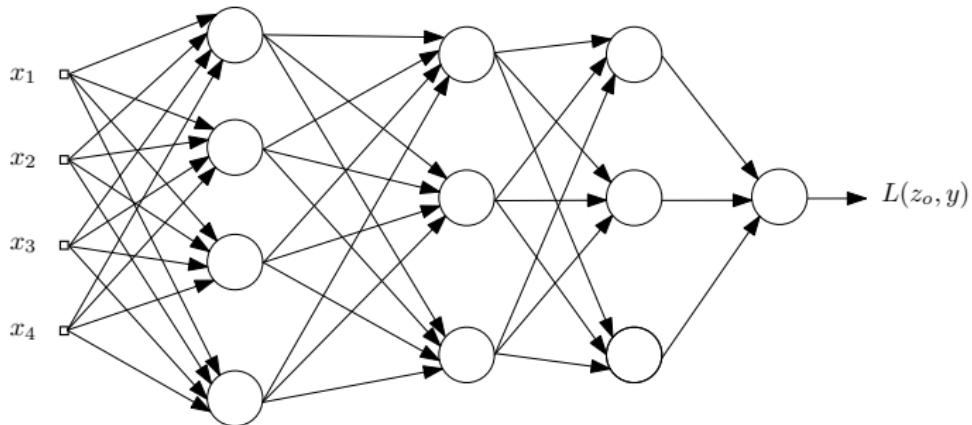


$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = \delta_j z_i$$

Cálculo de los δ

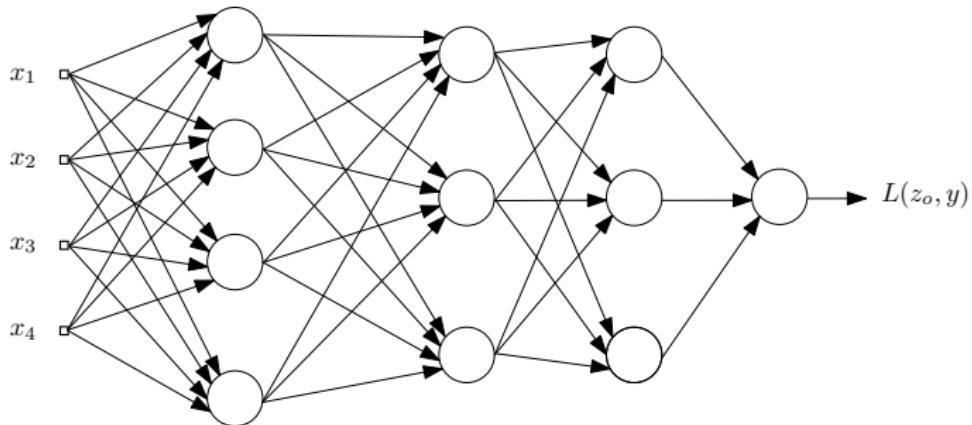


Cálculo de los δ



- En la salida:

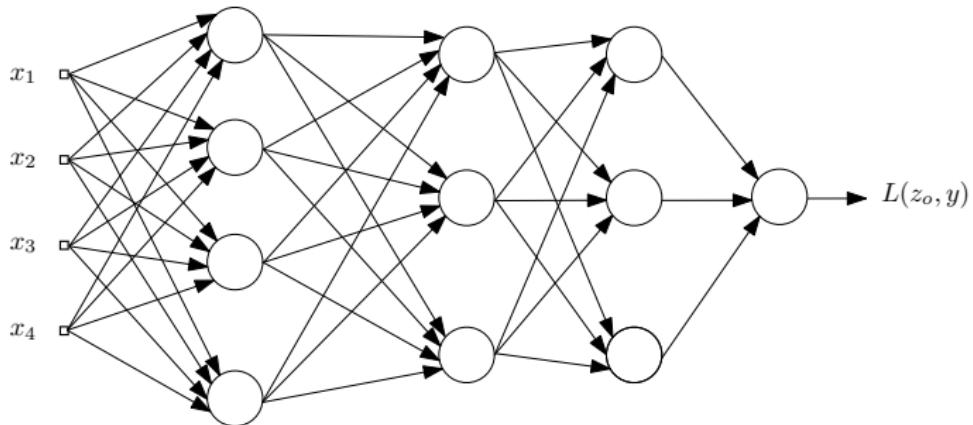
Cálculo de los δ



- En la salida:

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

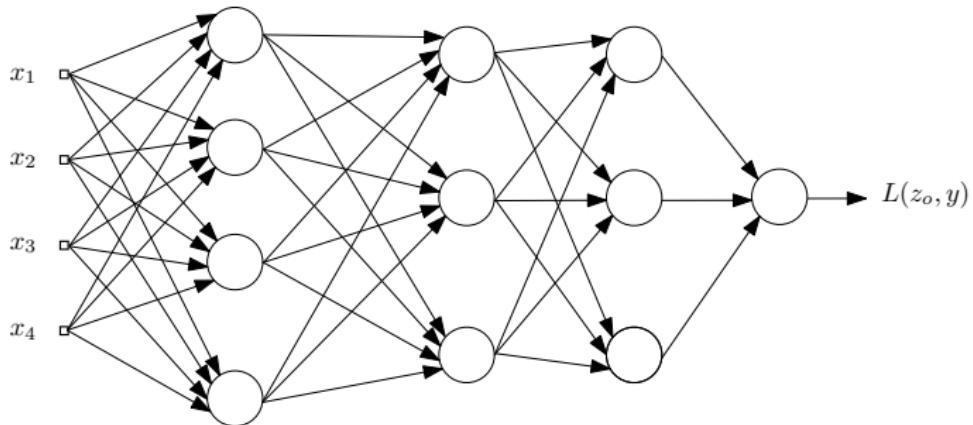
Cálculo de los δ



- En la salida:

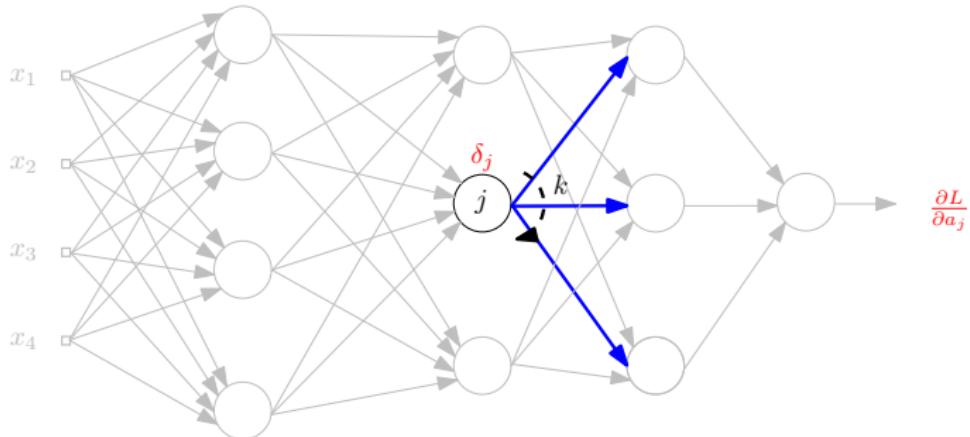
$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o} = h(\mathbf{w}, \mathbf{x}_p) - y_p$$

Cálculo de los δ

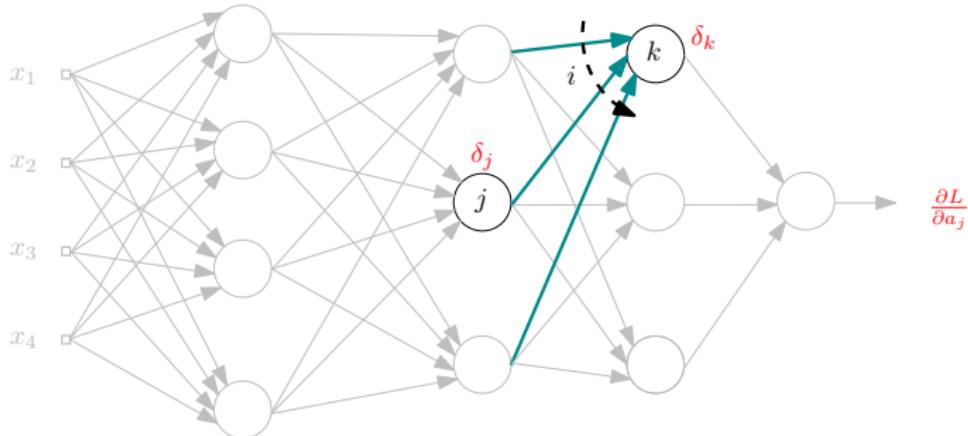


- En la salida:

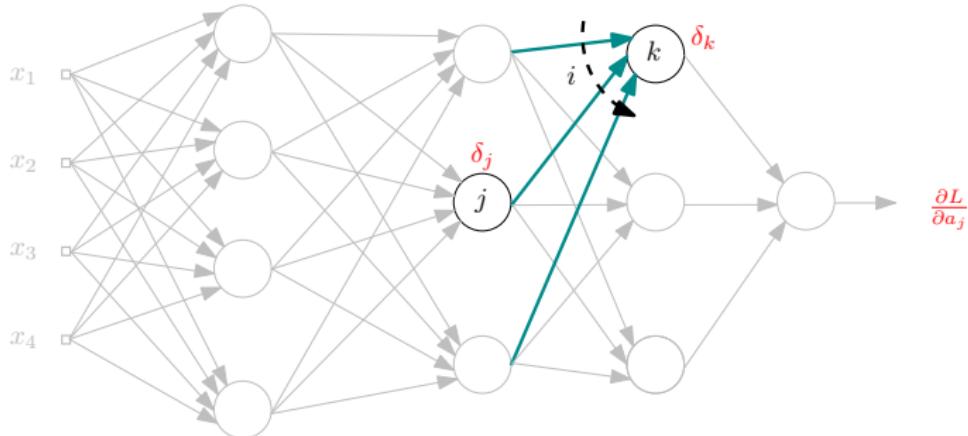
$$\delta_o = f'_o(a_o) \frac{\partial L}{\partial z_o} = h(\mathbf{w}, \mathbf{x}_p) - y_p \leftarrow \text{Señal de error}$$



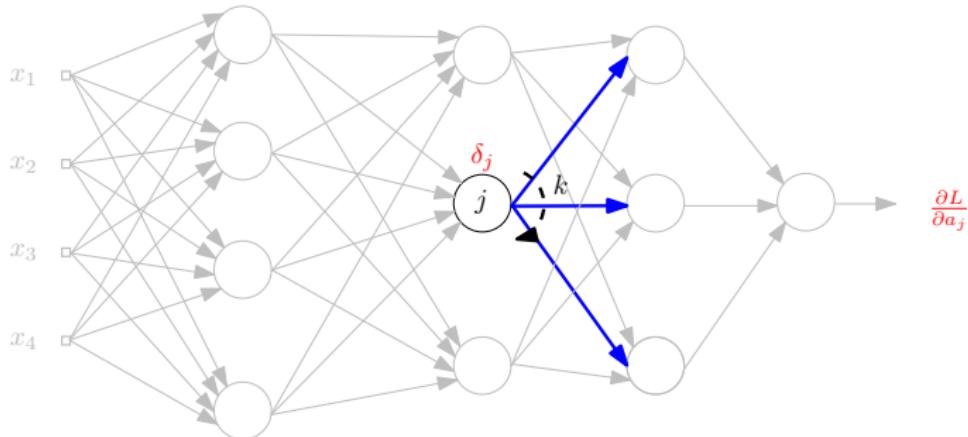
$$\delta_j = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$



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

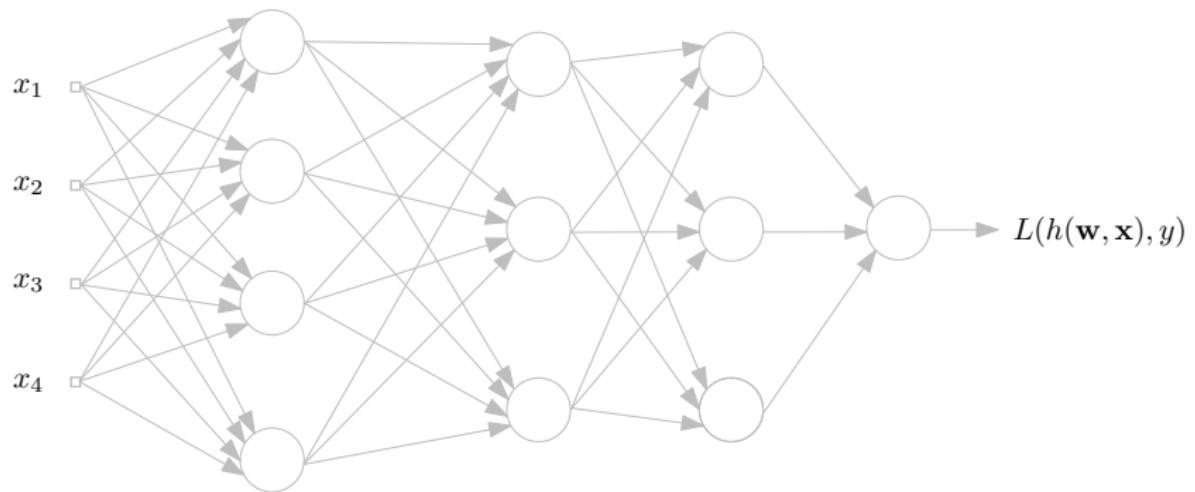


$$\delta_j = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial}{\partial a_j} \sum_i w_{ik} f_i(a_i) = f'_j(a_j) \sum_k w_{jk} \delta_k$$

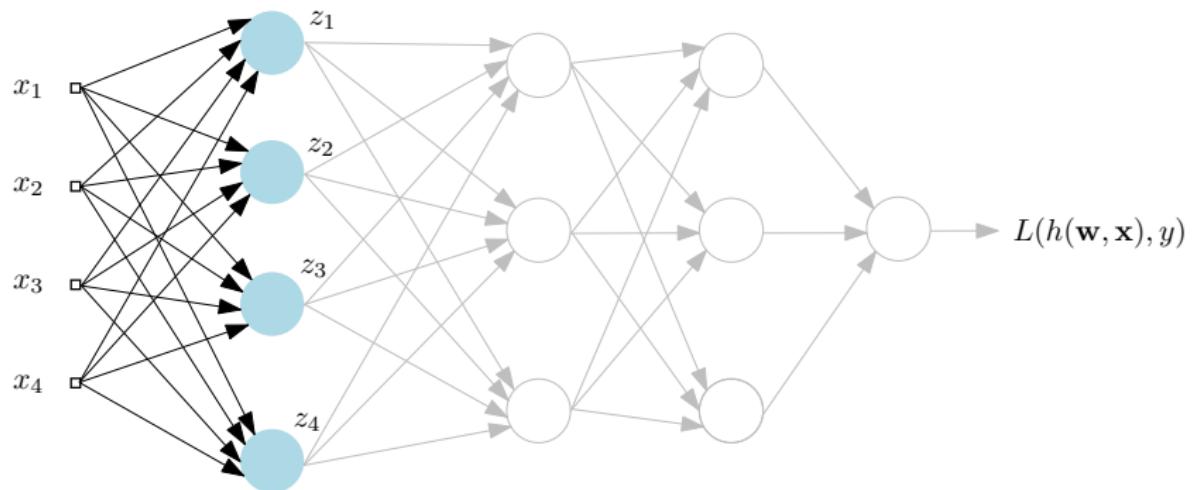


$$\delta_j = \sum_k \frac{\partial L}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial}{\partial a_j} \sum_i w_{ik} f_i(a_i) = f'_j(a_j) \sum_k w_{jk} \delta_k$$

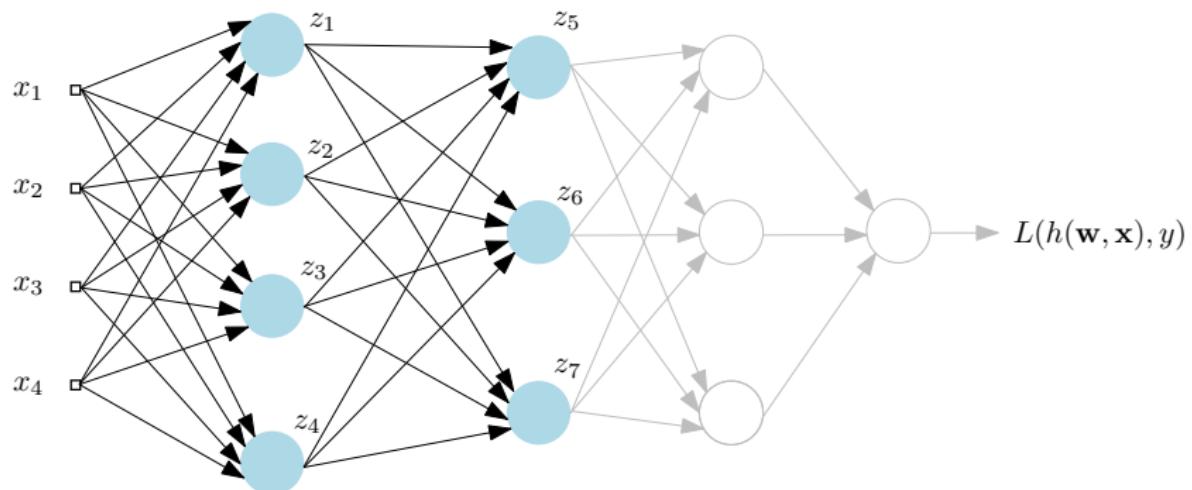
Forward pass



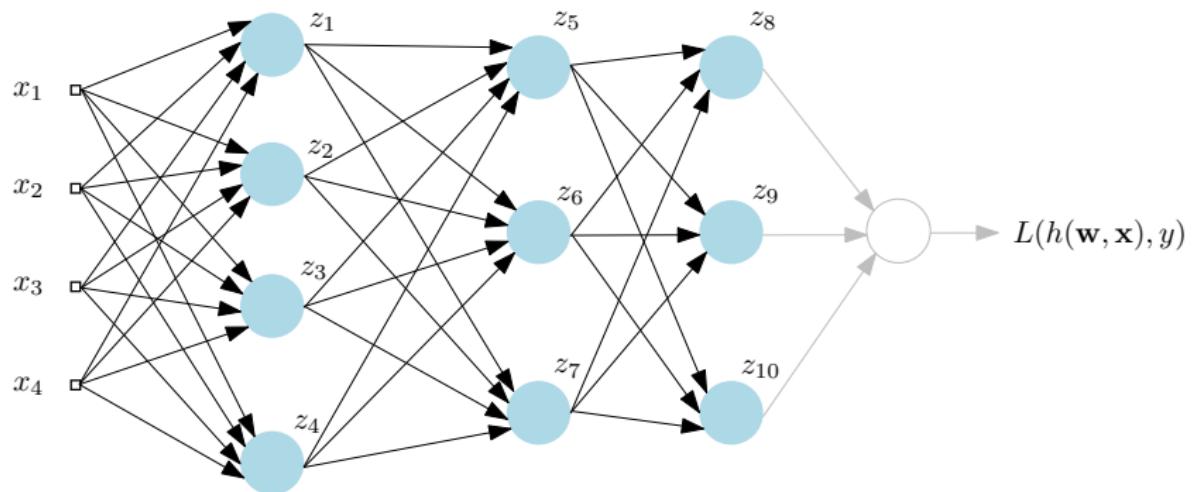
Forward pass



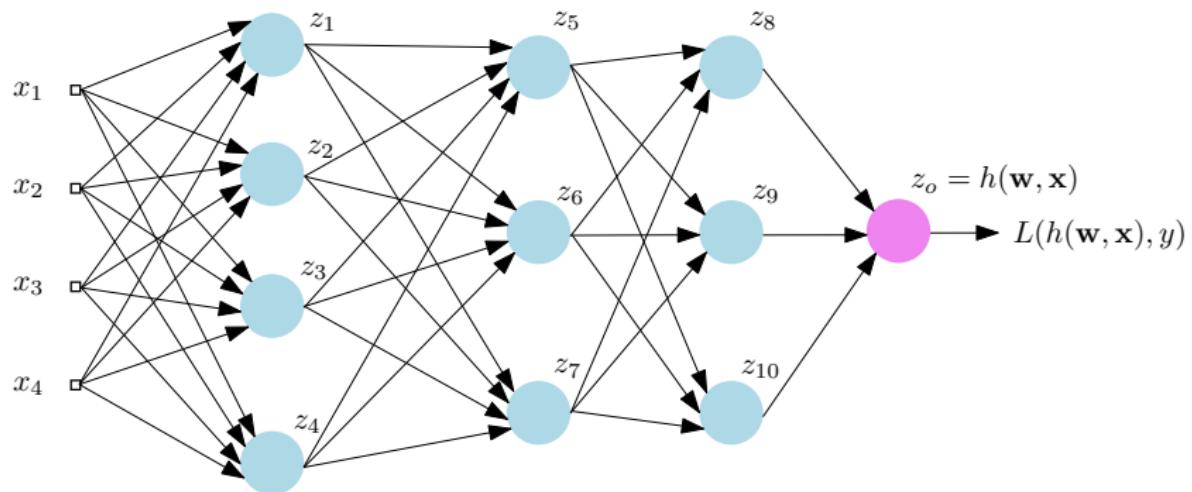
Forward pass



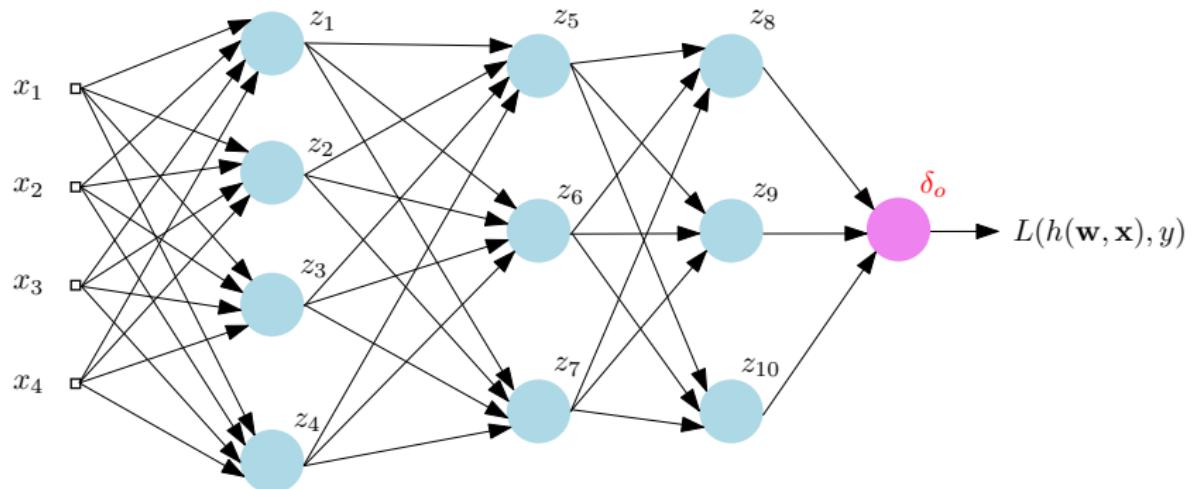
Forward pass



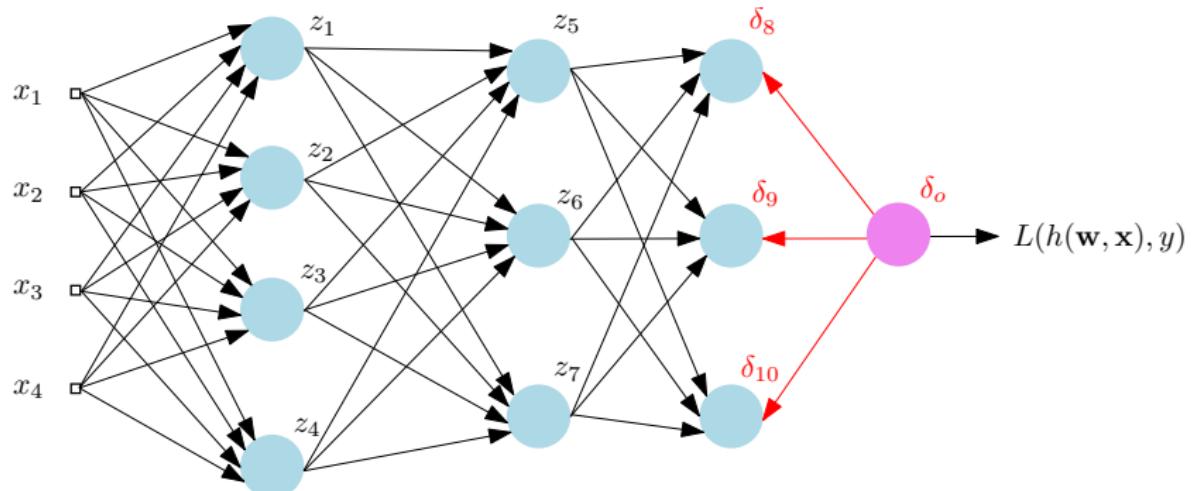
Forward pass



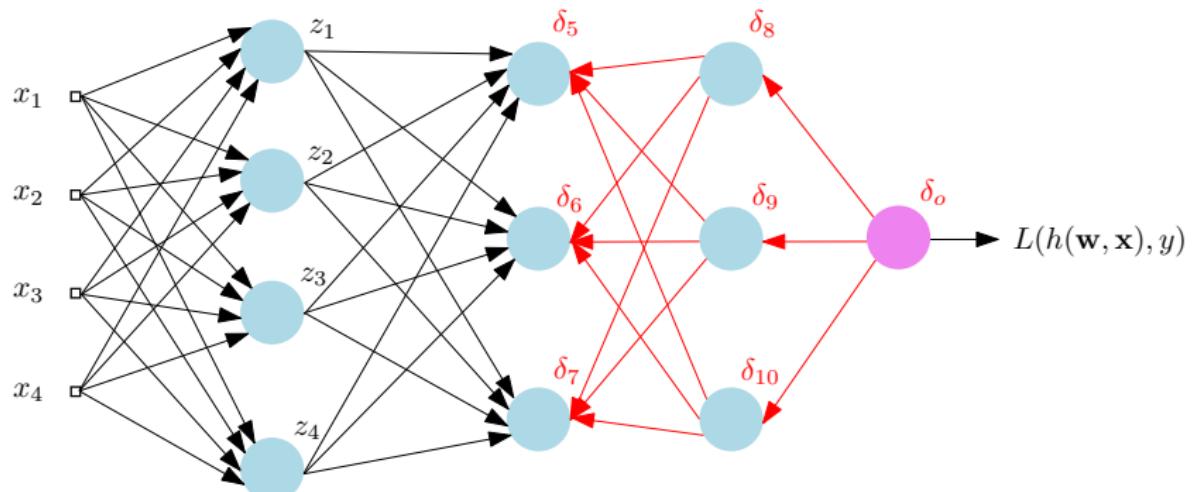
Backpropagation



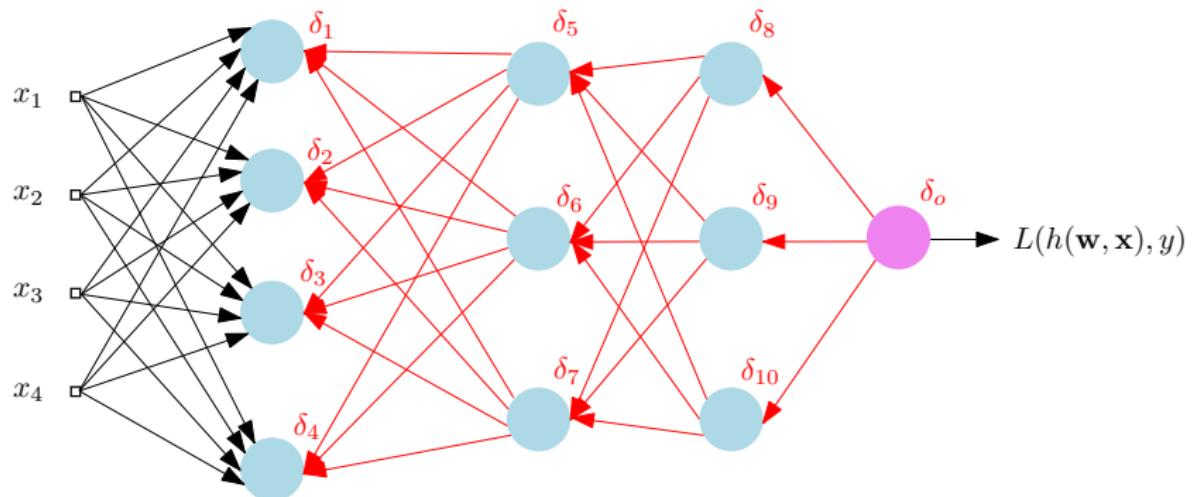
Backpropagation



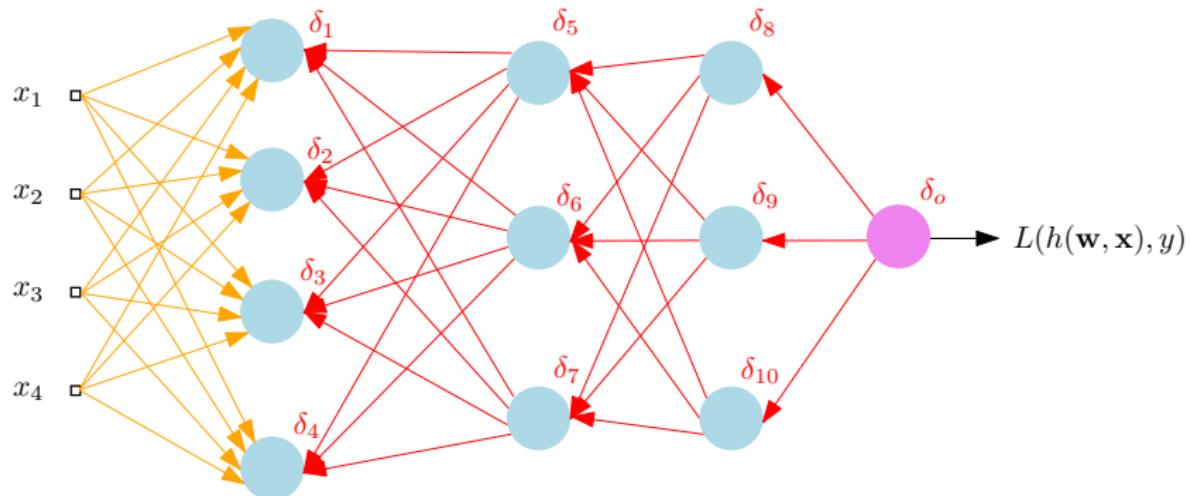
Backpropagation



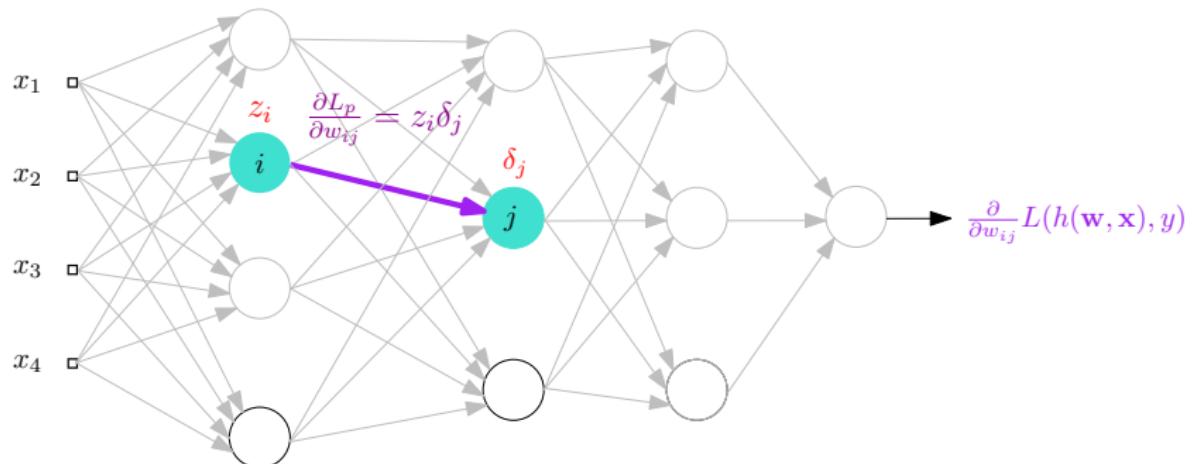
Backpropagation



Backpropagation



Actualización de los pesos



Selección de modelo e hiperparámetros

Selección de modelo e hiperparámetros

- Arquitectura de la red:
 - ▶ Número de capas, neuronas por capa.
 - ▶ Función de activación.

Selección de modelo e hiperparámetros

- Arquitectura de la red:
 - ▶ Número de capas, neuronas por capa.
 - ▶ Función de activación.
- Otros parámetros:
 - ▶ Tasa de aprendizaje.
 - ▶ Otras constantes (p.ej. momentum).
 - ▶ Regularización.

Validación cruzada

Validación cruzada

- ① Datos S se dividen en subconjuntos S_{train} and S_{test} , con $|S_{train}| = (1 - \gamma)|S|$ y $|S_{test}| = \gamma|S|$, $\gamma \in (0, 1)$.

Validación cruzada

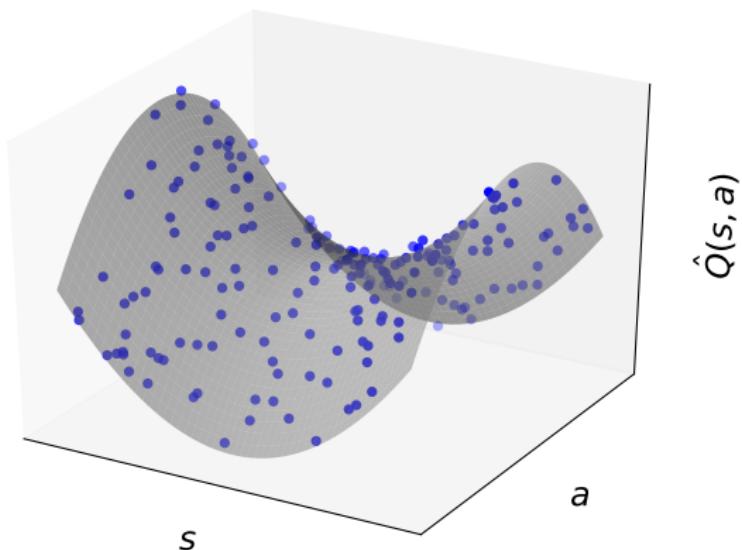
- ① Datos S se dividen en subconjuntos S_{train} and S_{test} , con $|S_{train}| = (1 - \gamma)|S|$ y $|S_{test}| = \gamma|S|$, $\gamma \in (0, 1)$.
- ② Se hallan varias redes candidatas h_d entrenando con diversas arquitecturas/parámetros en S_{train} .

Validación cruzada

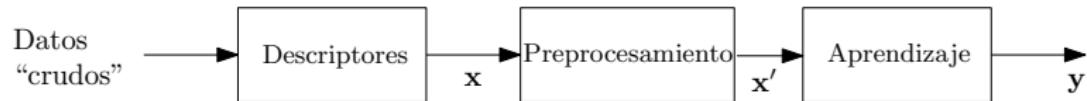
- ① Datos S se dividen en subconjuntos S_{train} and S_{test} , con $|S_{train}| = (1 - \gamma)|S|$ y $|S_{test}| = \gamma|S|$, $\gamma \in (0, 1)$.
- ② Se hallan varias redes candidatas h_d entrenando con diversas arquitecturas/parámetros en S_{train} .
- ③ Se selecciona la hipótesis candidata h_d con el menor error en S_{test} :

$$h_{d^*} = \arg \min_{\{h_1, h_2, \dots\}} E_{S_{test}}(h_d)$$

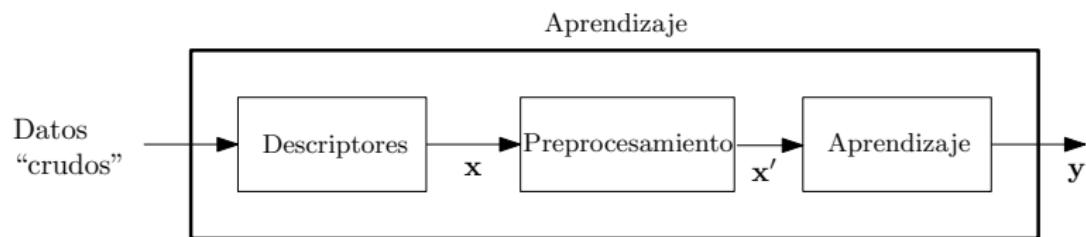
Aprendizaje de función $Q(s, a)$



Aprendizaje de Características



Aprendizaje de Características



Ejemplo: Pacman



Ejemplo: Pacman



- $s = [\text{pacman}_{xy} \quad \text{fantasmas}_{xy} \quad \text{puntos} \quad \dots]$

Ejemplo: Pacman



- $s = [\text{pacman}_{xy} \quad \text{fantasmas}_{xy} \quad \text{puntos} \quad \dots]$
- Aprendizaje automático de características:

Ejemplo: Pacman



- $s = [\text{pacman}_{xy} \quad \text{fantasmas}_{xy} \quad \text{puntos} \quad \dots]$
- Aprendizaje automático de características:
 - ▶ Imagen es entrada de la red.

Ejemplo: Pacman



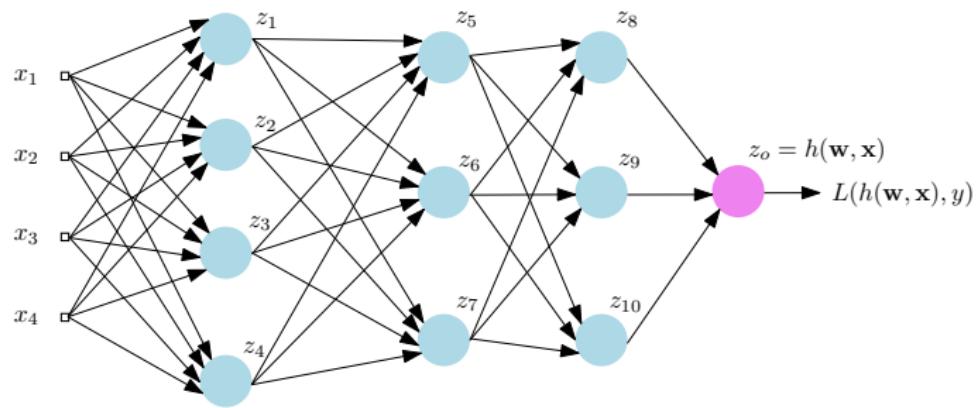
- $s = [\text{pacman}_{xy} \quad \text{fantasmas}_{xy} \quad \text{puntos} \quad \dots]$
- Aprendizaje automático de características:
 - ▶ Imagen es entrada de la red.
 - ▶ En capas **escondidas** se **aprenden** gradualmente características.

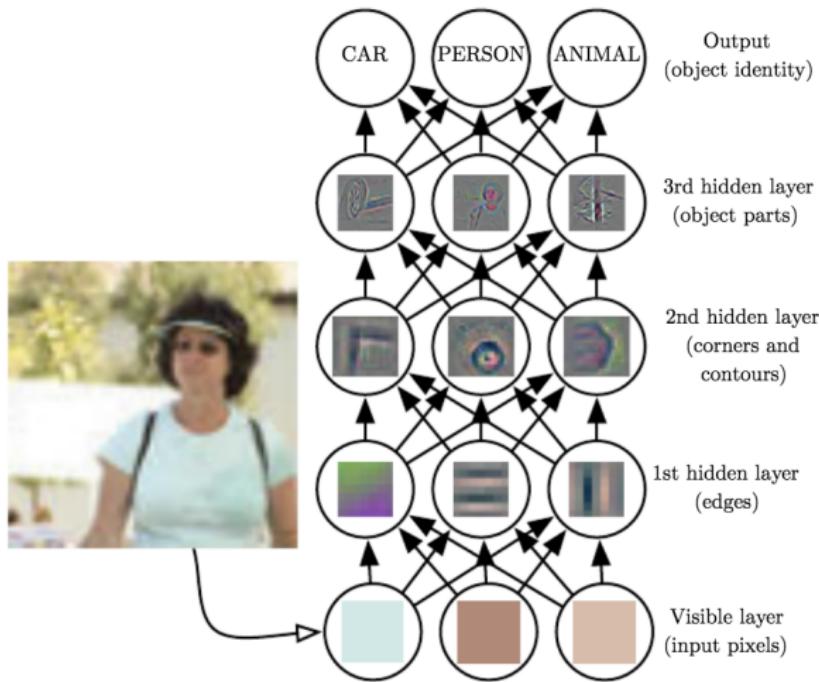
Ejemplo: Pacman



- $s = [\text{pacman}_{xy} \quad \text{fantasmas}_{xy} \quad \text{puntos} \quad \dots]$
- Aprendizaje automático de características:
 - ▶ Imagen es entrada de la red.
 - ▶ En capas **escondidas** se **aprenden** gradualmente características.

Arquitectura en Capas





Redes profundas

- Número de capas \gg

Redes profundas

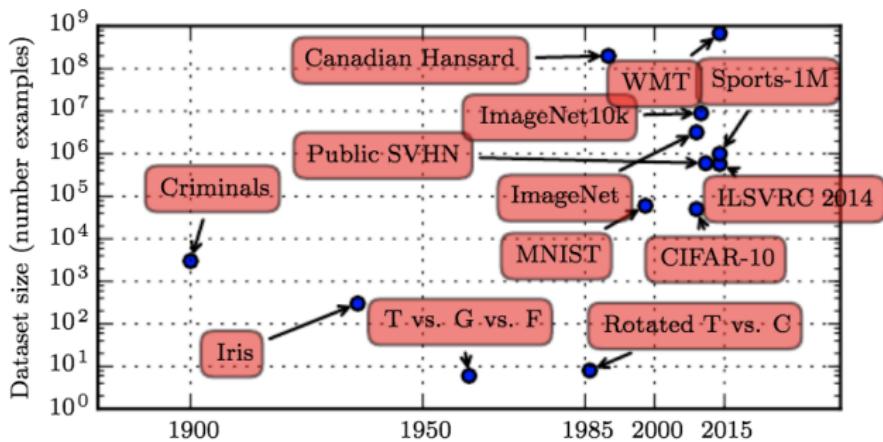
- Número de capas \gg
- Número de parámetros (pesos) \ggg

Redes profundas

- Número de capas \gg
- Número de parámetros (pesos) \ggg
- Número de datos \gg

Redes profundas

- Número de capas \gg
- Número de parámetros (pesos) \ggg
- Número de datos \ggg
- Capacidad de computación \ggg (GPUs).



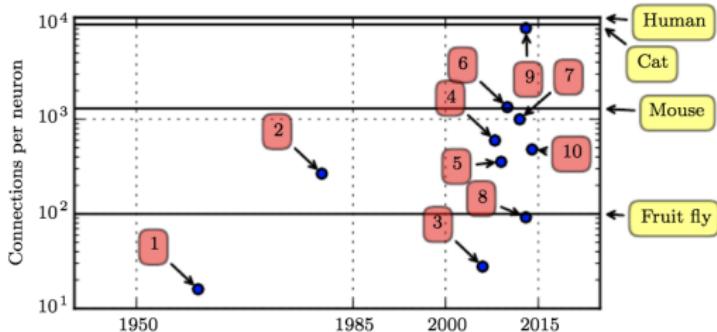


Figure 1.10: Number of connections per neuron over time. Initially, the number of connections between neurons in artificial neural networks was limited by hardware capabilities. Today, the number of connections between neurons is mostly a design consideration. Some artificial neural networks have nearly as many connections per neuron as a cat, and it is quite common for other neural networks to have as many connections per neuron as smaller mammals like mice. Even the human brain does not have an exorbitant amount of connections per neuron. Biological neural network sizes from [Wikipedia \(2015\)](#).

1. Adaptive linear element ([Widrow and Hoff, 1960](#))
2. Neocognitron ([Fukushima, 1980](#))
3. GPU-accelerated convolutional network ([Chellapilla et al., 2006](#))
4. Deep Boltzmann machine ([Salakhutdinov and Hinton, 2009a](#))
5. Unsupervised convolutional network ([Jarrett et al., 2009](#))
6. GPU-accelerated multilayer perceptron ([Ciresan et al., 2010](#))
7. Distributed autoencoder ([Le et al., 2012](#))
8. Multi-GPU convolutional network ([Krizhevsky et al., 2012](#))
9. COTS HPC unsupervised convolutional network ([Coates et al., 2013](#))
10. GoogLeNet ([Szegedy et al., 2014a](#))

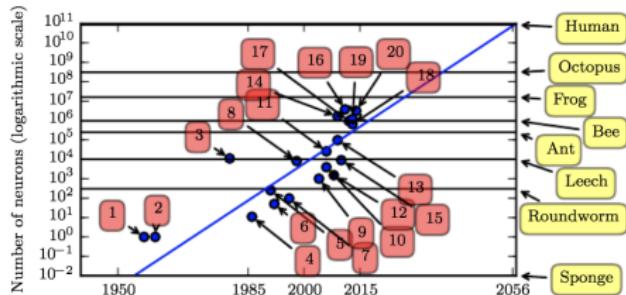


Figure 1.11: Increasing neural network size over time. Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years. Biological neural network sizes from [Wikipedia \(2015\)](#).

1. Perceptron ([Rosenblatt, 1958, 1962](#))
2. Adaptive linear element ([Widrow and Hoff, 1960](#))
3. Neocognitron ([Fukushima, 1980](#))
4. Early back-propagation network ([Rumelhart *et al.*, 1986b](#))
5. Recurrent neural network for speech recognition ([Robinson and Fallside, 1991](#))
6. Multilayer perceptron for speech recognition ([Bengio *et al.*, 1991](#))
7. Mean field sigmoid belief network ([Saul *et al.*, 1996](#))
8. LeNet-5 ([LeCun *et al.*, 1998b](#))
9. Echo state network ([Jaeger and Haas, 2004](#))
10. Deep belief network ([Hinton *et al.*, 2006](#))
11. GPU-accelerated convolutional network ([Chellapilla *et al.*, 2006](#))
12. Deep Boltzmann machine ([Salakhutdinov and Hinton, 2009a](#))
13. GPU-accelerated deep belief network ([Raina *et al.*, 2009](#))
14. Unsupervised convolutional network ([Jarrett *et al.*, 2009](#))
15. GPU-accelerated multilayer perceptron ([Ciresan *et al.*, 2010](#))
16. OMP-1 network ([Coates and Ng, 2011](#))
17. Distributed autoencoder ([Le *et al.*, 2012](#))
18. Multi-GPU convolutional network ([Krizhevsky *et al.*, 2012](#))
19. COTS HPC unsupervised convolutional network ([Coates *et al.*, 2013](#))
20. GoogLeNet ([Szegedy *et al.*, 2014a](#))

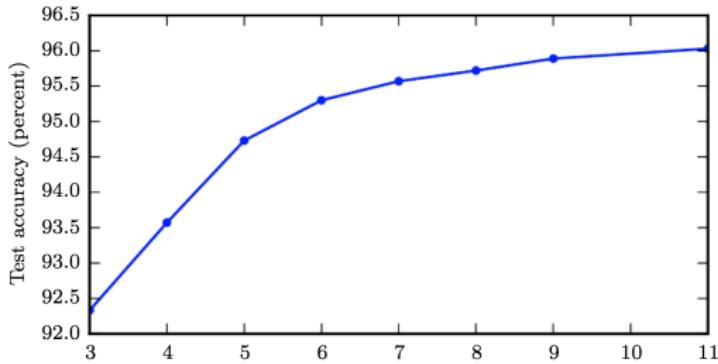
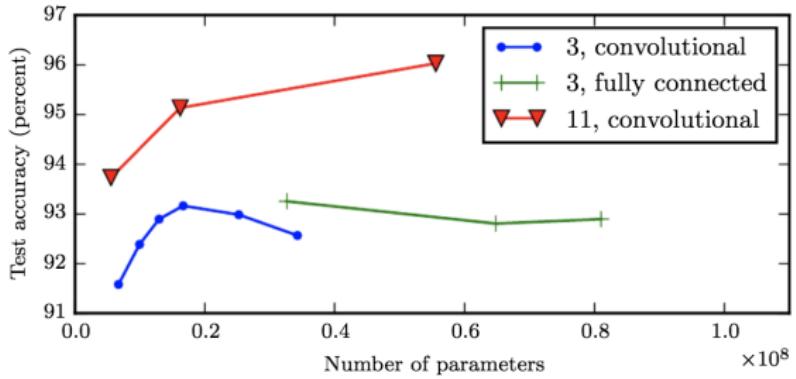


Figure 6.6: Effect of depth. Empirical results showing that deeper networks generalize better when used to transcribe multidigit numbers from photographs of addresses. Data from Goodfellow *et al.* (2014d). The test set accuracy consistently increases with increasing depth. See figure 6.7 for a control experiment demonstrating that other increases to the model size do not yield the same effect.



Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \dots$

Require: Initial parameter θ

$k \leftarrow 1$

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

 Apply update: $\theta \leftarrow \theta - \epsilon_k \hat{\mathbf{g}}$

$k \leftarrow k + 1$

end while

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ , momentum parameter α

Require: Initial parameter θ , initial velocity v

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

 Compute velocity update: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$.

 Apply update: $\theta \leftarrow \theta + \mathbf{v}$.

end while

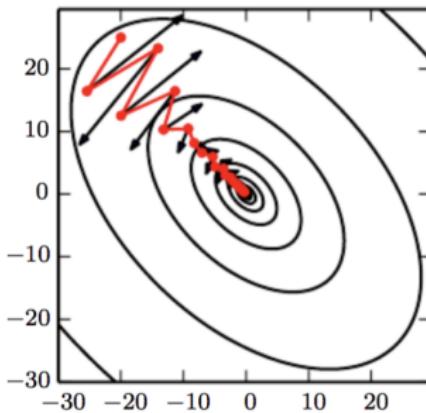


Figure 8.5: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian 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: $\hat{s} \leftarrow \frac{\hat{s}}{1 - \rho_1^t}$

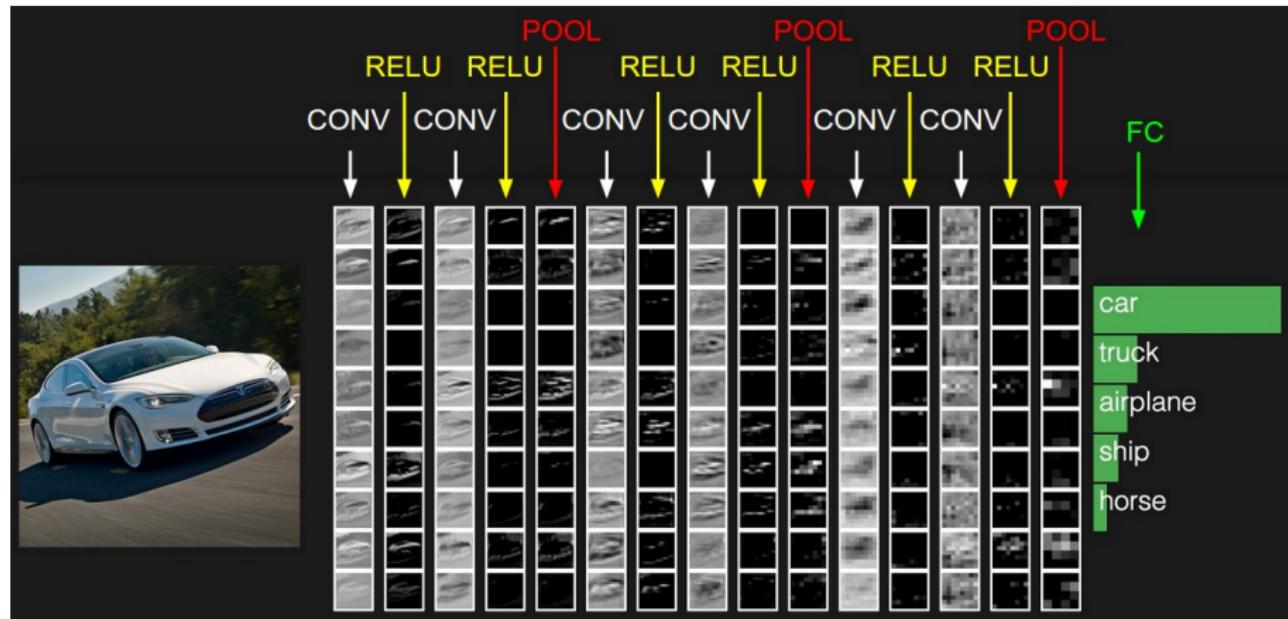
 Correct bias in second moment: $\hat{r} \leftarrow \frac{\hat{r}}{1 - \rho_2^t}$

 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

Redes Convolucionales



Redes Convolucionales

Redes Convolucionales

- Imágenes/datos en grilla 2D.

Redes Convolucionales

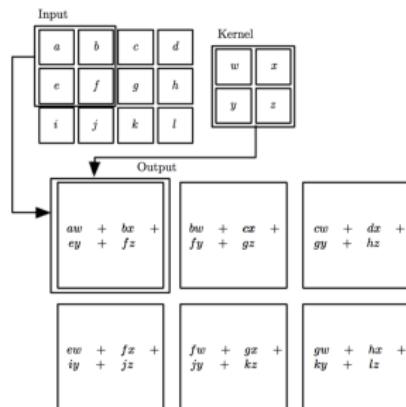
- Imágenes/datos en grilla 2D.
- Convolución 2D (correlación):

$$y[n_1, n_2] = \sum_{k_1} \sum_{k_2} I[k_1, k_2] K[n_1 + k_1, n_2 + k_2]$$

Redes Convolucionales

- Imágenes/datos en grilla 2D.
- Convolución 2D (correlación):

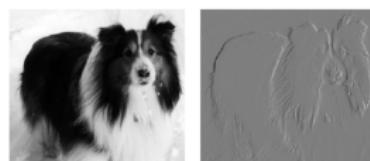
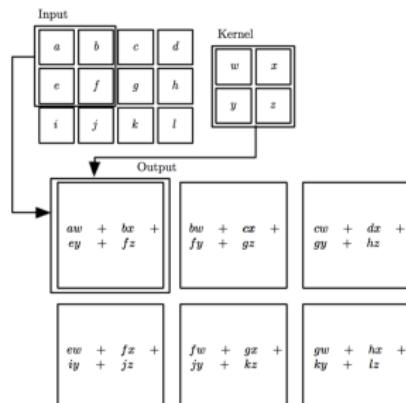
$$y[n_1, n_2] = \sum_{k_1} \sum_{k_2} I[k_1, k_2] K[n_1 + k_1, n_2 + k_2]$$



Redes Convolucionales

- Imágenes/datos en grilla 2D.
- Convolución 2D (correlación):

$$y[n_1, n_2] = \sum_{k_1} \sum_{k_2} I[k_1, k_2] K[n_1 + k_1, n_2 + k_2]$$



- Máscara → Campo receptivo.

- Máscara → Campo receptivo.
- Pesos en la red = Valores en la máscara.

- Máscara → Campo receptivo.
- Pesos en la red = Valores en la máscara.
- Conectividad dispersa.

- Máscara → Campo receptivo.
- Pesos en la red = Valores en la máscara.
- Conectividad dispersa.
- Pesos compartidos (weight sharing).

- Máscara → Campo receptivo.
- Pesos en la red = Valores en la máscara.
- Conectividad dispersa.
- Pesos compartidos (weight sharing).
- Equivarianza: misma característica a lo largo de la imagen.

Conectividad dispersa

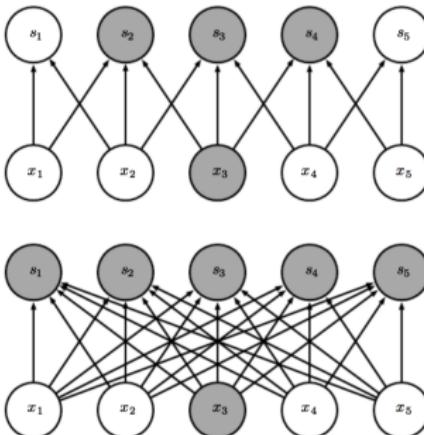


Figure 9.2: *Sparse connectivity, viewed from below:* We highlight one input unit, x_3 , and also highlight the output units in \mathbf{s} that are affected by this unit. (Top) When \mathbf{s} is formed by convolution with a kernel of width 3, only three outputs are affected by \mathbf{x} . (Bottom) When \mathbf{s} is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by x_3 .

Conectividad dispersa

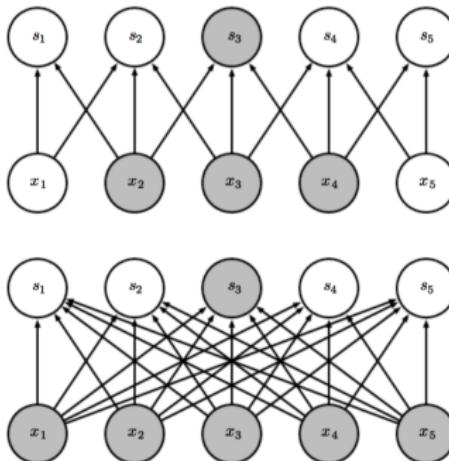


Figure 9.3: *Sparse connectivity, viewed from above:* We highlight one output unit, s_3 , and also highlight the input units in \mathbf{x} that affect this unit. These units are known as the **receptive field** of s_3 . (Top) When \mathbf{s} is formed by convolution with a kernel of width 3, only three inputs affect s_3 . (Bottom) When \mathbf{s} is formed by matrix multiplication, connectivity is no longer sparse, so all of the inputs affect s_3 .

Campo receptivo creciente

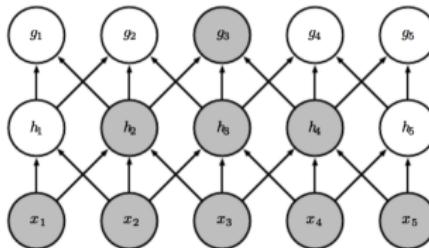


Figure 9.4: The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution (figure 9.12) or pooling (section 9.3). This means that even though *direct* connections in a convolutional net are very sparse, units in the deeper layers can be *indirectly* connected to all or most of the input image.

Pesos Compartidos

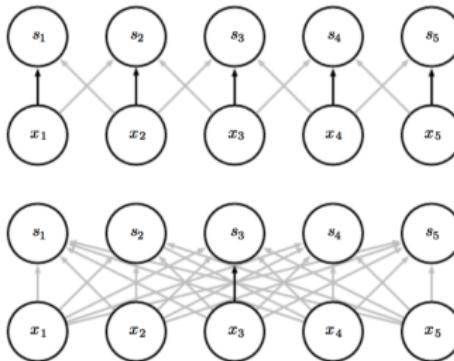
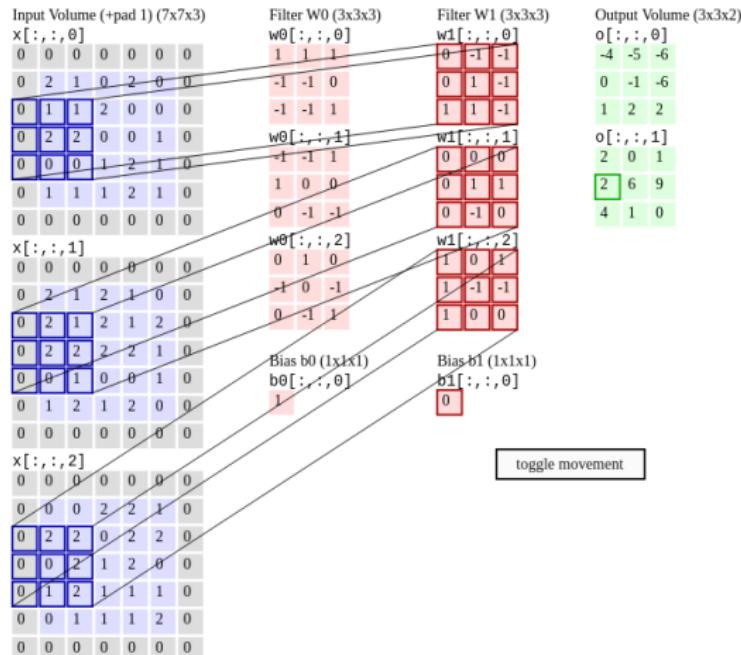


Figure 9.5: Parameter sharing: Black arrows indicate the connections that use a particular parameter in two different models. (*Top*) The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations. (*Bottom*) The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once.

Capa de convolución



Pooling

Pooling

- Reemplaza la salida en una posición dada por una estadística resumen de las salidas en una vecindad.

Pooling

- Reemplaza la salida en una posición dada por una estadística resumen de las salidas en una vecindad.
 - ▶ Máximo.

Pooling

- Reemplaza la salida en una posición dada por una estadística resumen de las salidas en una vecindad.
 - ▶ Máximo.
 - ▶ Promedio (pesado).

Pooling

- Reemplaza la salida en una posición dada por una estadística resumen de las salidas en una vecindad.
 - ▶ Máximo.
 - ▶ Promedio (pesado).
 - ▶ Norma

Pooling

- Reemplaza la salida en una posición dada por una estadística resumen de las salidas en una vecindad.
 - ▶ Máximo.
 - ▶ Promedio (pesado).
 - ▶ Norma
- Representación invariante a pequeñas traslaciones de la entrada.

Pooling

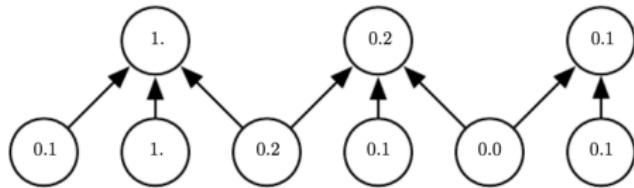
- Reemplaza la salida en una posición dada por una estadística resumen de las salidas en una vecindad.
 - ▶ Máximo.
 - ▶ Promedio (pesado).
 - ▶ Norma
- Representación invariante a pequeñas traslaciones de la entrada.
- Presencia de una característica sin importar localización exacta.

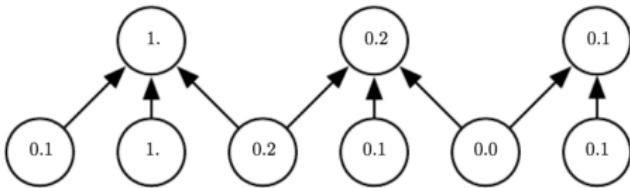
Pooling

- Reemplaza la salida en una posición dada por una estadística resumen de las salidas en una vecindad.
 - ▶ Máximo.
 - ▶ Promedio (pesado).
 - ▶ Norma
- Representación invariante a pequeñas traslaciones de la entrada.
- Presencia de una característica sin importar localización exacta.
- Pooling a través de canales: aprender otras invarianzas.

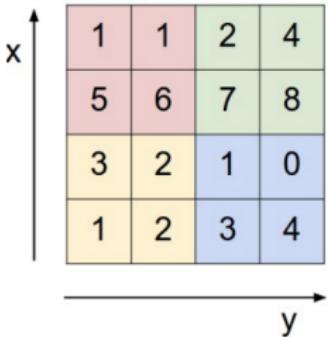
Pooling

- Reemplaza la salida en una posición dada por una estadística resumen de las salidas en una vecindad.
 - ▶ Máximo.
 - ▶ Promedio (pesado).
 - ▶ Norma
- Representación invariante a pequeñas traslaciones de la entrada.
- Presencia de una característica sin importar localización exacta.
- Pooling a través de canales: aprender otras invarianzas.
- Entradas de tamaño variante.

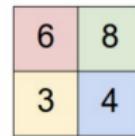


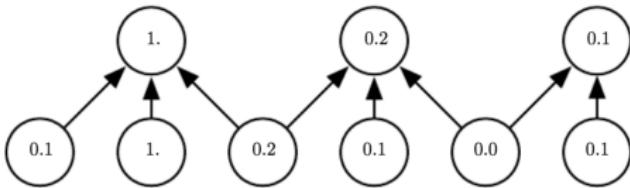


Single depth slice

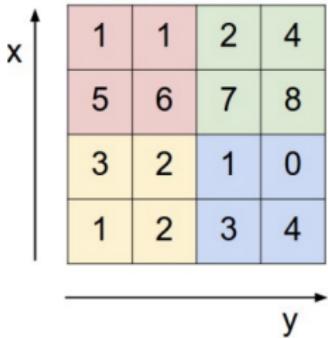


max pool with 2x2 filters
and stride 2



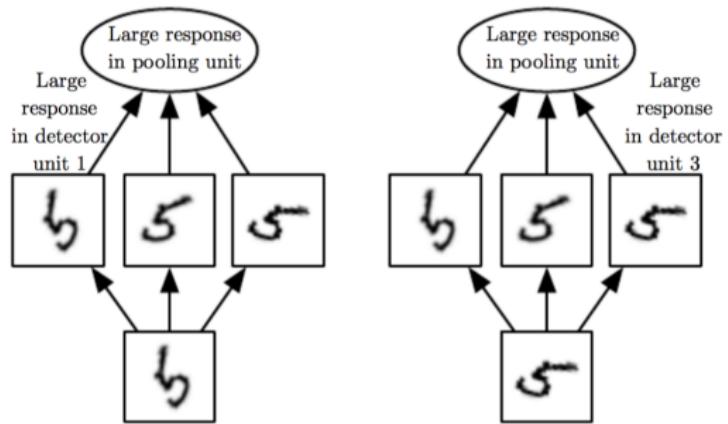


Single depth slice

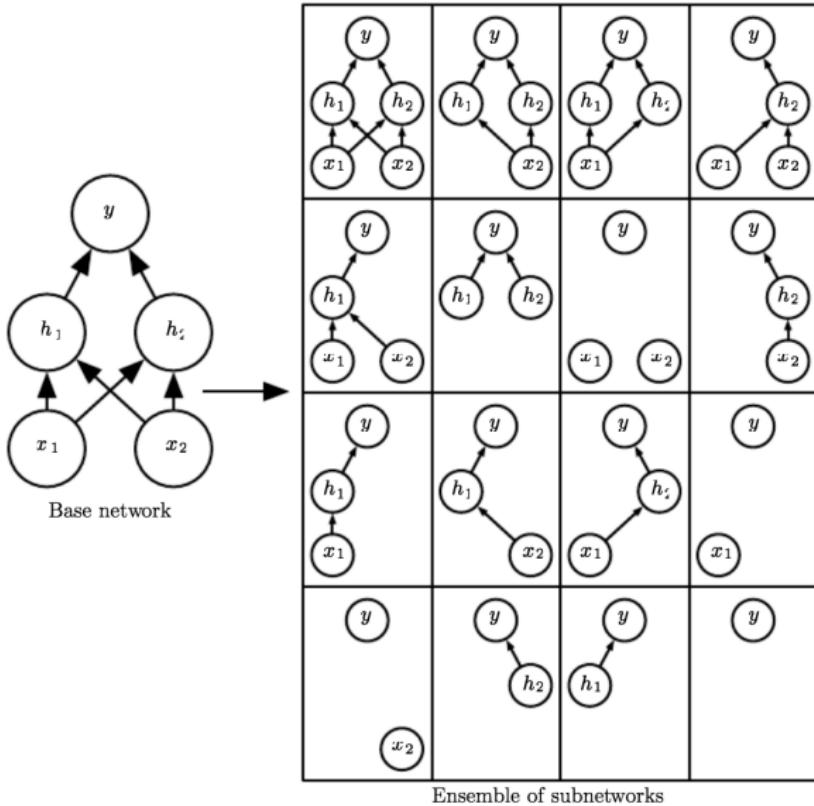


max pool with 2x2 filters
and stride 2

- Stride: reducción de la dimensión en la salida.



Dropout



Bagging (Bootstrap Aggregating)

Bagging (Bootstrap Aggregating)

- Bootstrapping (Efron, 1979):

Bagging (Bootstrap Aggregating)

- Bootstrapping (Efron, 1979):
 - ▶ Técnica para mejorar precisión de parámetros/estimativos.

Bagging (Bootstrap Aggregating)

- Bootstrapping (Efron, 1979):
 - ▶ Técnica para mejorar precisión de parámetros/estimativos.
 - ▶ Muestreo con remplazo.

Bagging (Bootstrap Aggregating)

- Bootstrapping (Efron, 1979):
 - ▶ Técnica para mejorar precisión de parámetros/estimativos.
 - ▶ Muestreo con remplazo.
 - ▶ Muestra \sim población.

Bagging (Bootstrap Aggregating)

- Bootstrapping (Efron, 1979):
 - ▶ Técnica para mejorar precisión de parámetros/estimativos.
 - ▶ Muestreo con remplazo.
 - ▶ Muestra \sim población.
- Clasificadores h_1, h_2, \dots entrenados en muestras **bootstrap** de los datos de entrenamiento.

Bagging (Bootstrap Aggregating)

- Bootstrapping (Efron, 1979):
 - ▶ Técnica para mejorar precisión de parámetros/estimativos.
 - ▶ Muestreo con remplazo.
 - ▶ Muestra \sim población.
- Clasificadores h_1, h_2, \dots entrenados en muestras **bootstrap** de los datos de entrenamiento.
- Combinación por votación.

Algorithm 1 Bagging

for $t = 1$ to T **do** Obtenga \mathcal{S}_t de \mathcal{S} muestreando con reemplazo. $h_t \leftarrow A(\mathcal{S}_t)$ **end for**Retorne $f(x) = \text{votacion } \{h_t(x)\}$

Dropout

Dropout

- Aproximación de Bagging.

Dropout

- Aproximación de Bagging.
- Clasificador combinado consiste en todas las posibles subredes de la red original.

Dropout

- Aproximación de Bagging.
- Clasificador combinado consiste en todas las posibles subredes de la red original.
- Sea μ vector de **máscara** indicando neuronas a incluir.

Dropout

- Aproximación de Bagging.
- Clasificador combinado consiste en todas las posibles subredes de la red original.
- Sea μ vector de **máscara** indicando neuronas a incluir.
- Error a minimizar: $\mathcal{E}_\mu J(\mathbf{w}, \mu)$

Dropout

- Aproximación de Bagging.
- Clasificador combinado consiste en todas las posibles subredes de la red original.
- Sea μ vector de **máscara** indicando neuronas a incluir.
- Error a minimizar: $\mathcal{E}_\mu J(\mathbf{w}, \mu)$
- Aproximación:

Dropout

- Aproximación de Bagging.
- Clasificador combinado consiste en todas las posibles subredes de la red original.
- Sea μ vector de **máscara** indicando neuronas a incluir.
- Error a minimizar: $\mathcal{E}_\mu J(\mathbf{w}, \mu)$
- Aproximación: para cada minibatch muestrear bits de μ independientemente.

Dropout

- Aproximación de Bagging.
- Clasificador combinado consiste en todas las posibles subredes de la red original.
- Sea μ vector de **máscara** indicando neuronas a incluir.
- Error a minimizar: $\mathcal{E}_\mu J(\mathbf{w}, \mu)$
- Aproximación: para cada minibatch muestrear bits de μ independientemente. Usualmente:
 - ▶ 1 con probabilidad 0.5 para neuronas escondidas.
 - ▶ 1 con probabilidad 0.8 para neuronas de entrada.

- Votación aproxima media geométrica:

$$\tilde{p}(y : \mathbf{x}) = \frac{1}{Z} \left(\prod_{\boldsymbol{\mu}} p(y : \mathbf{x}, \boldsymbol{\mu}) \right)^{\frac{1}{2^d}}$$

- Votación aproxima media geométrica:

$$\tilde{p}(y : \mathbf{x}) = \frac{1}{Z} \left(\prod_{\boldsymbol{\mu}} p(y : \mathbf{x}, \boldsymbol{\mu}) \right)^{\frac{1}{2d}}$$

- Weight scaling inference rule:

- Votación aproxima media geométrica:

$$\tilde{p}(y : \mathbf{x}) = \frac{1}{Z} \left(\prod_{\boldsymbol{\mu}} p(y : \mathbf{x}, \boldsymbol{\mu}) \right)^{\frac{1}{2^d}}$$

- Weight scaling inference rule: Evaluar red completa, con los pesos que salen de una neurona multiplicados por la probabilidad de que esté incluida.

- Votación aproxima media geométrica:

$$\tilde{p}(y : \mathbf{x}) = \frac{1}{Z} \left(\prod_{\boldsymbol{\mu}} p(y : \mathbf{x}, \boldsymbol{\mu}) \right)^{\frac{1}{2^d}}$$

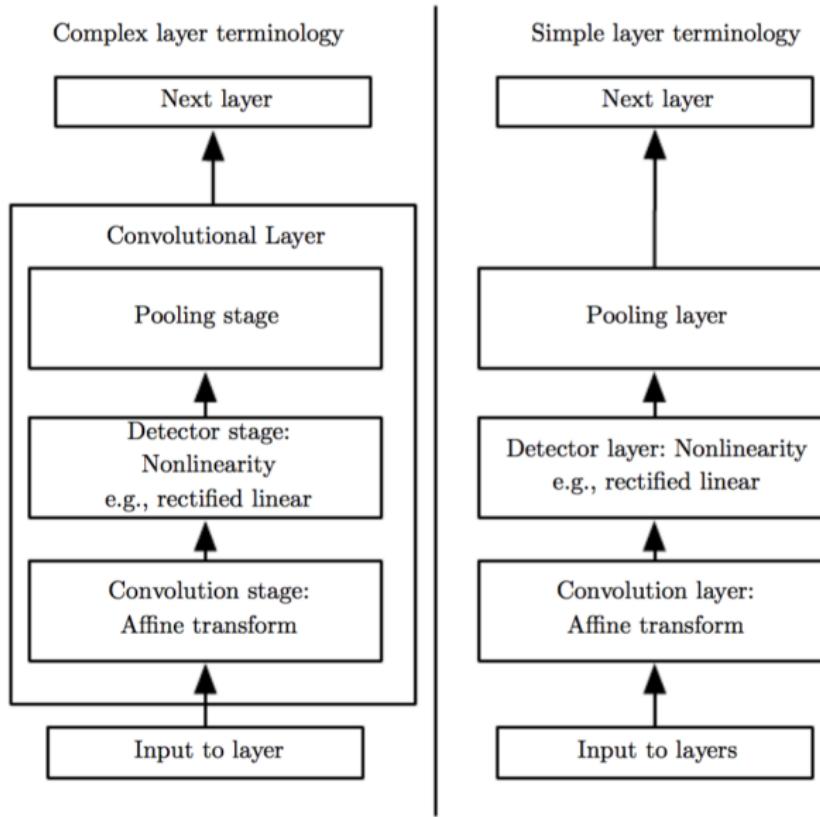
- Weight scaling inference rule: Evaluar red completa, con los pesos que salen de una neurona multiplicados por la probabilidad de que esté incluida.
- Usualmente consiste en dividir los pesos de la red entrenada por 2.

- Votación aproxima media geométrica:

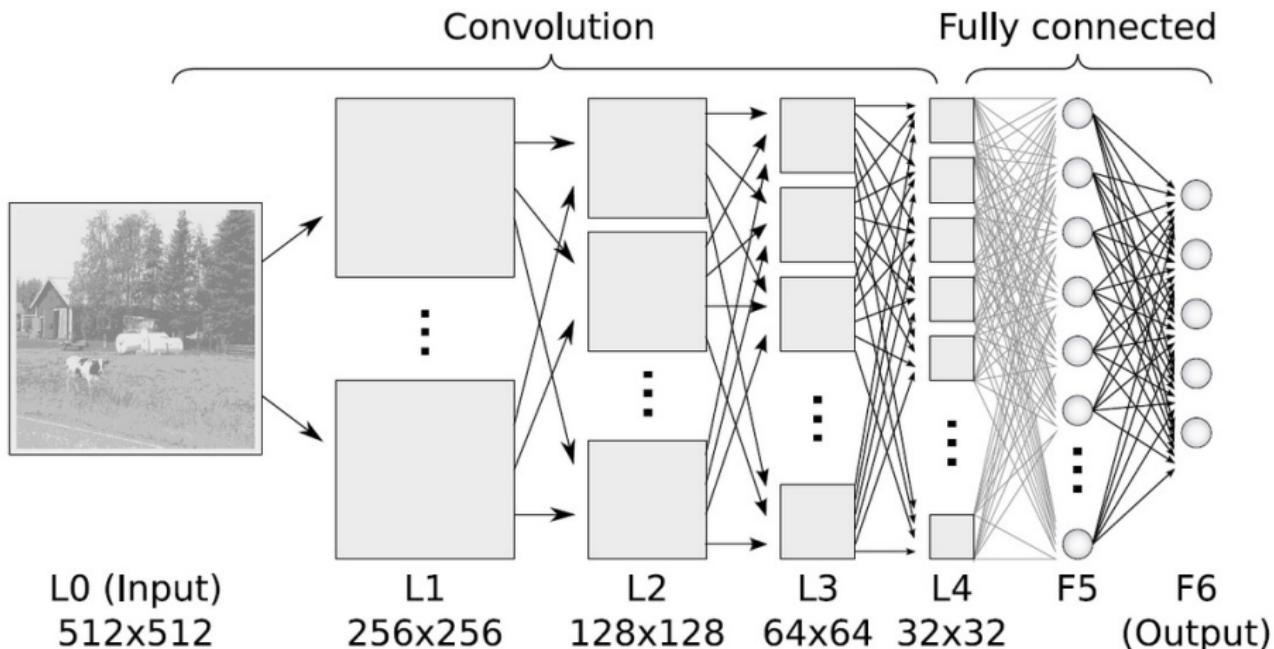
$$\tilde{p}(y : \mathbf{x}) = \frac{1}{Z} \left(\prod_{\boldsymbol{\mu}} p(y : \mathbf{x}, \boldsymbol{\mu}) \right)^{\frac{1}{2^d}}$$

- Weight scaling inference rule: Evaluar red completa, con los pesos que salen de una neurona multiplicados por la probabilidad de que esté incluida.
- Usualmente consiste en dividir los pesos de la red entrenada por 2.
- Aproximación es exacta en modelos de regresión softmax.

Capa convolucional



Arquitectura



Diseño de red convolucional

Diseño de red convolucional

- Selección de modelo!!

Diseño de red convolucional

- Selección de modelo!!
 - ① Número de capas.

Diseño de red convolucional

- Selección de modelo!!
 - ① Número de capas.
 - ② Número de filtros en cada capa.

Diseño de red convolucional

- Selección de modelo!!
 - ① Número de capas.
 - ② Número de filtros en cada capa.
 - ③ Tipo/reducción de pooling.

Diseño de red convolucional

- Selección de modelo!!
 - ① Número de capas.
 - ② Número de filtros en cada capa.
 - ③ Tipo/reducción de pooling.
 - ④ Capa totalmente conectada.

Diseño de red convolucional

- Selección de modelo!!
 - ① Número de capas.
 - ② Número de filtros en cada capa.
 - ③ Tipo/reducción de pooling.
 - ④ Capa totalmente conectada.
 - ⑤ :

Diseño de red convolucional

- Selección de modelo!!
 - ① Número de capas.
 - ② Número de filtros en cada capa.
 - ③ Tipo/reducción de pooling.
 - ④ Capa totalmente conectada.
 - ⑤ :
- Aproximación usual:

Diseño de red convolucional

- Selección de modelo!!
 - ① Número de capas.
 - ② Número de filtros en cada capa.
 - ③ Tipo/reducción de pooling.
 - ④ Capa totalmente conectada.
 - ⑤ :
- Aproximación usual:
 - ① Escoger arquitectura ya entrenada (exitosa!) en problema similar.

Diseño de red convolucional

- Selección de modelo!!
 - ① Número de capas.
 - ② Número de filtros en cada capa.
 - ③ Tipo/reducción de pooling.
 - ④ Capa totalmente conectada.
 - ⑤ :
- Aproximación usual:
 - ① Escoger arquitectura ya entrenada (exitosa!) en problema similar.
 - ② Ajuste fino (entrenar red totalmente conectada).