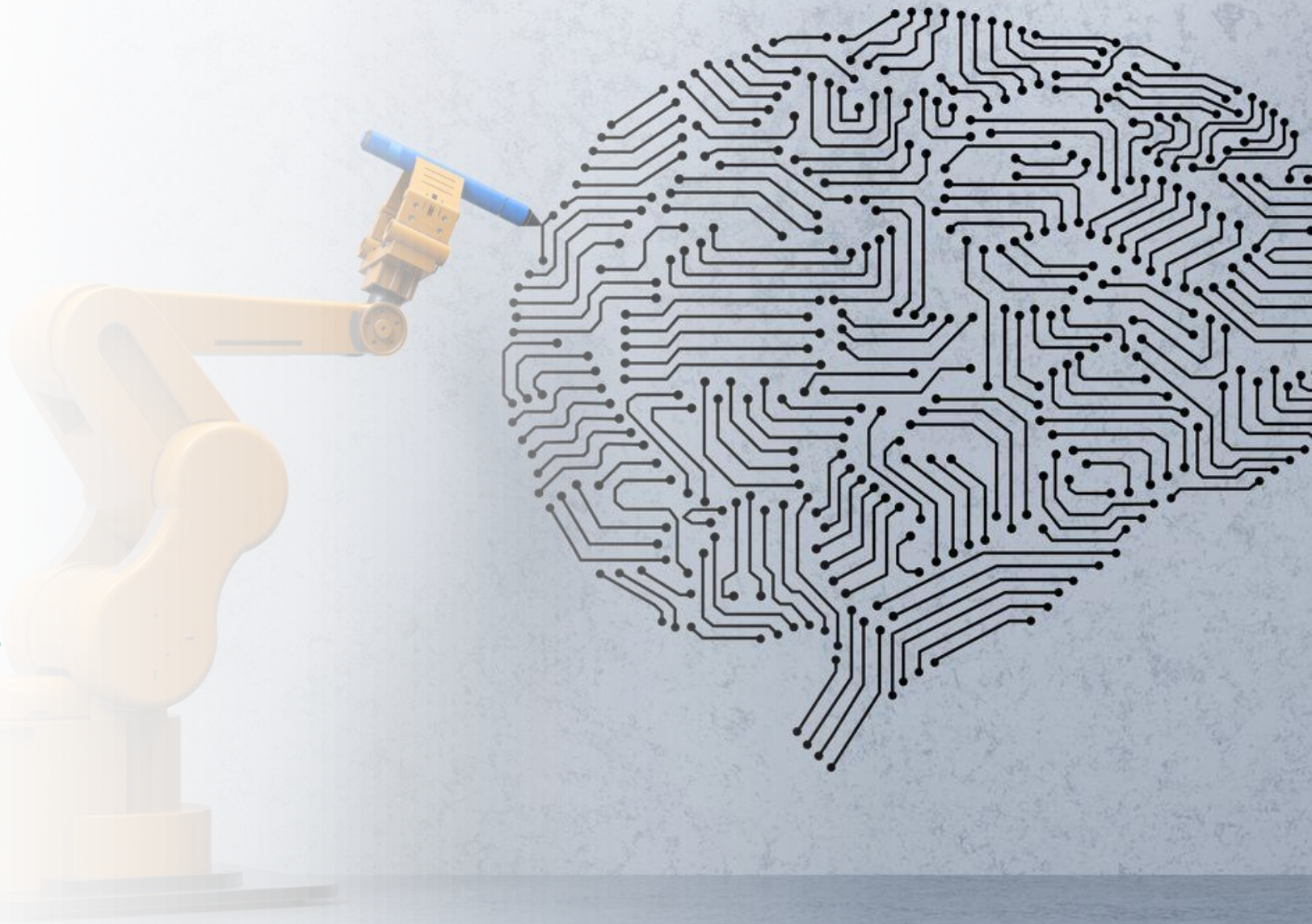
Aprendizaje por refuerzo

Clase 12: DQN



Antes de empezar…

- Dudas de tarea 2
  - Monte Carlo Control
  - SARSA
  - Q learning

Monte Carlo Control

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

El algoritmo SARSA

- Teorema: SARSA tabular converge a la función optima de acción valor, $q(s,a) \rightarrow q_*(s,a)$ si la política es GLIE

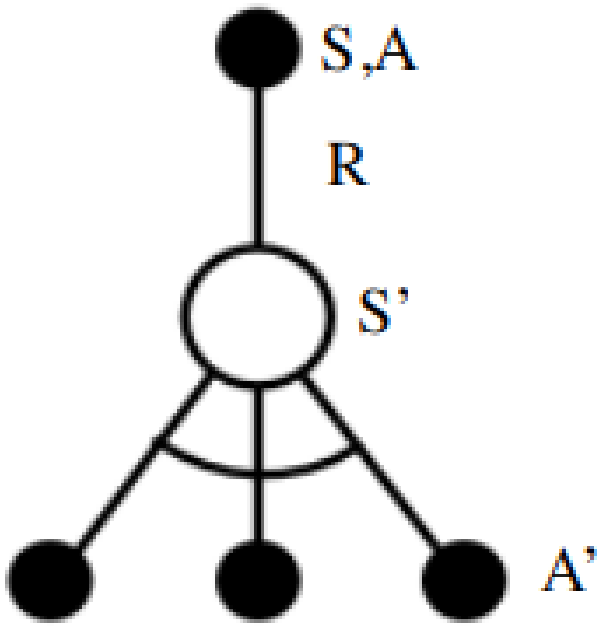
Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Algoritmo de control de aprendizaje Q

- $q_{t+1}(S_t, A_t) \leftarrow q_t(S_t, A_t) + \alpha_t(R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t))$
- Teorema
  - Control con aprendizaje Q converge a la función acción valor óptima q → q^* en el limite



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Para el día de hoy…

- Introducción a
  - Redes neuronales
  - Redes profundas convolucionales
- Implementación DQN en Pytorch

¡Llegamos!

nature › letters › article

Published: 25 February 2015

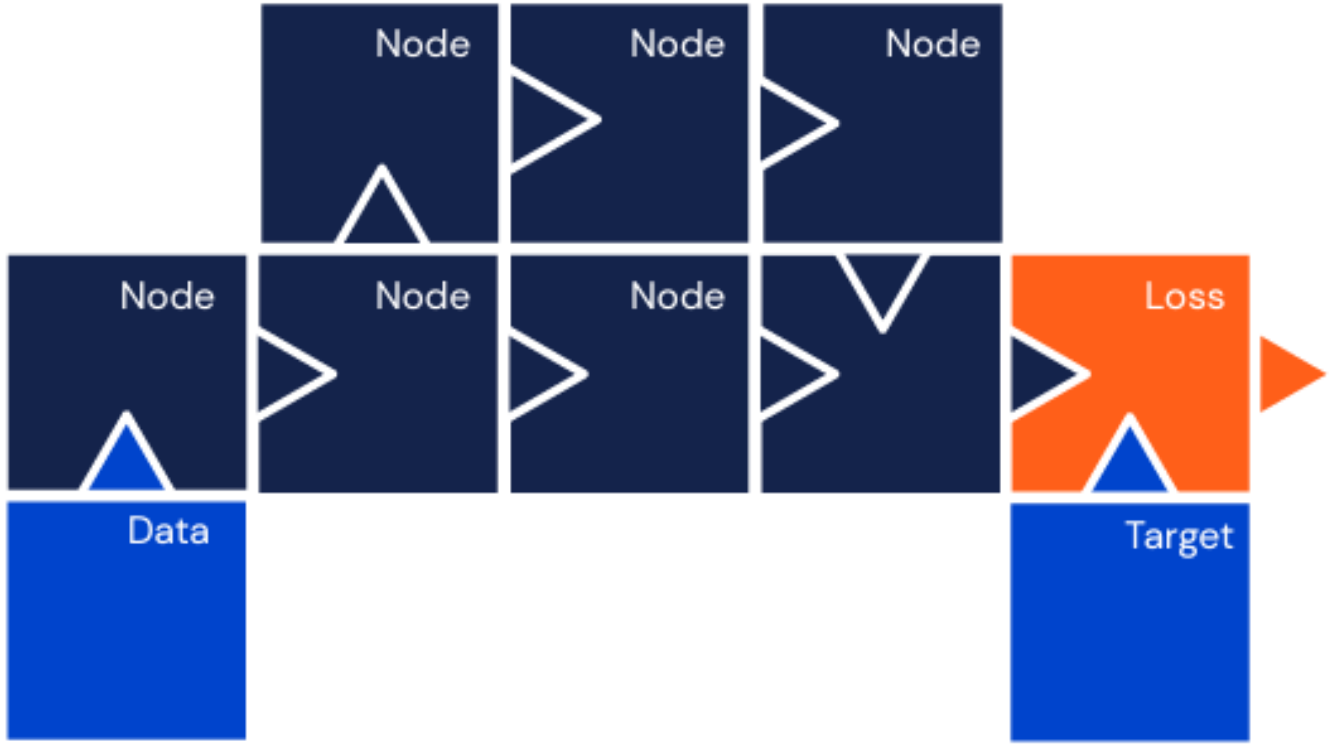
Human-level control through deep reinforcement learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

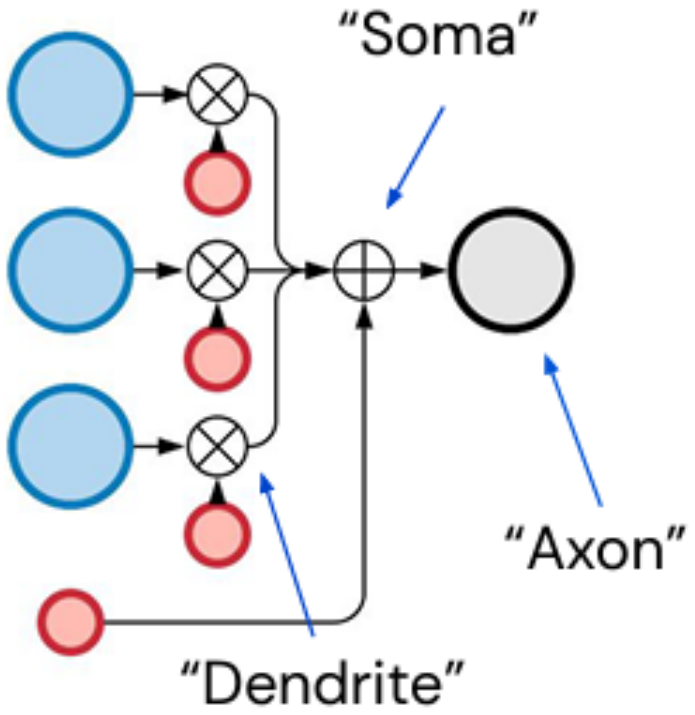
Nature **518**, 529–533 (2015) | Cite this article

405k Accesses | **8244** Citations | **1554** Altmetric | Metrics

El rompecabezas



Red neuronal artificial

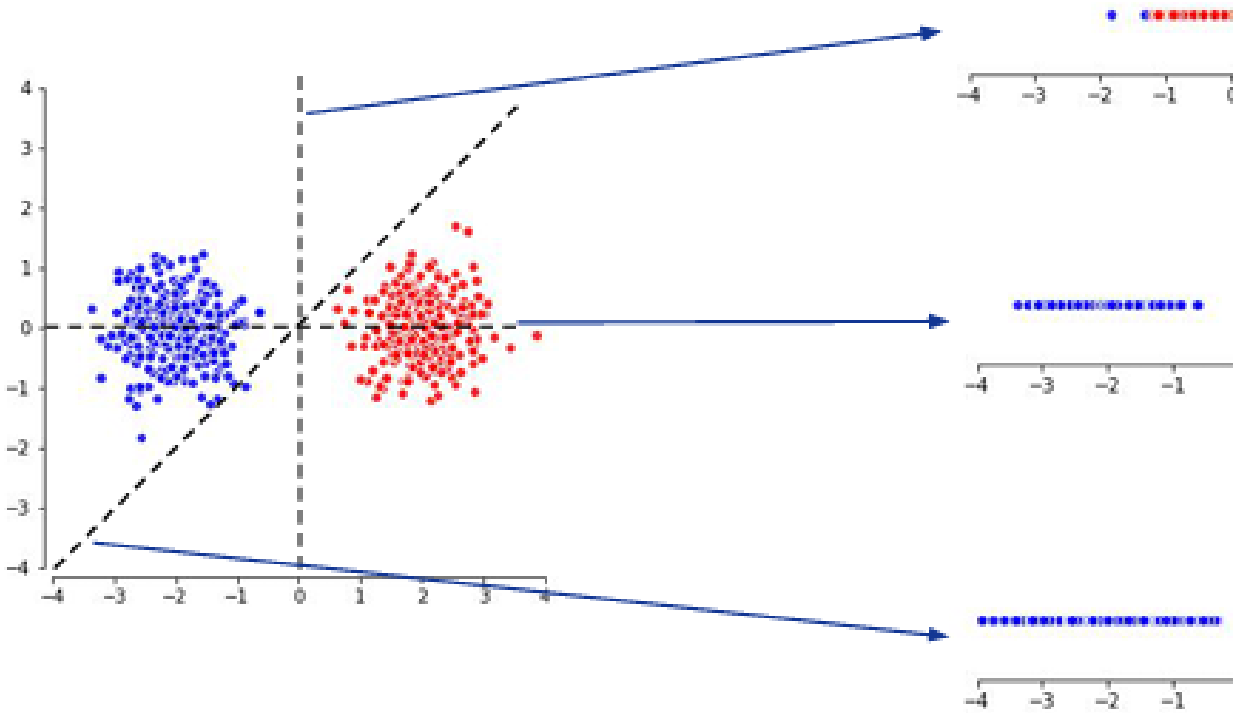


$$\sum_{i=1}^{d} \mathbf{w}_i \mathbf{x}_i$$

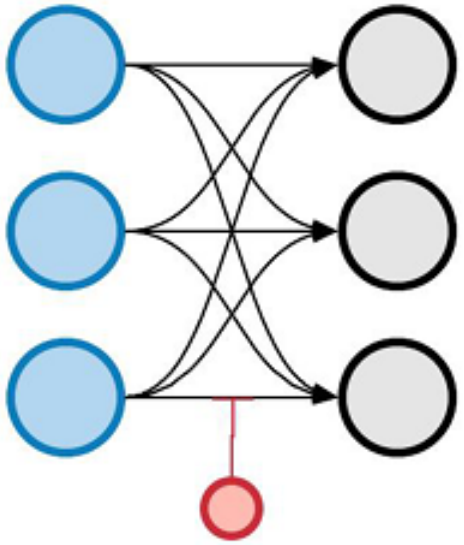
$$\sum_{i=0}^{d} \mathbf{w}_i \mathbf{x}_i$$

- El objetivo es reflejar observaciones neurofisiológicas y no reproducir su dinámica
- Fácil de componer
- Representa computación simple
- Tiene conexiones para inhibir y excitar

Neurona artificial



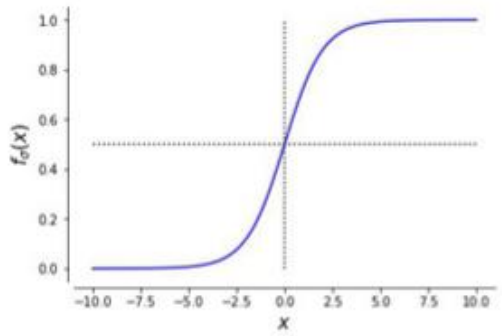
Capa lineal



$$h(\mathbf{x}, \mathbf{w}, b) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

$$f_{\text{linear}}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

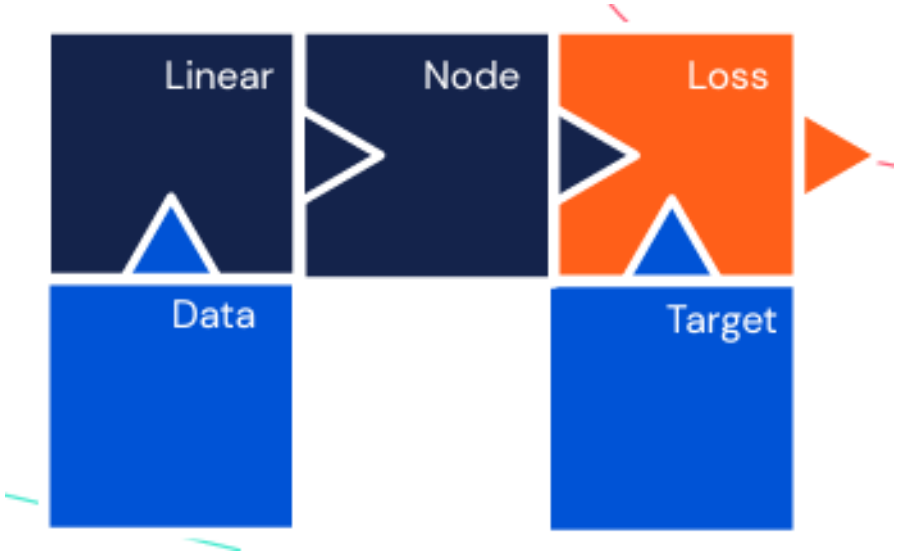
- En aprendizaje máquina linear en realidad quiere decir afin
- Las neuronas en una capa son llamadas unidad
- Los parámetros son llamados pesos



$$f_\sigma(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}$$

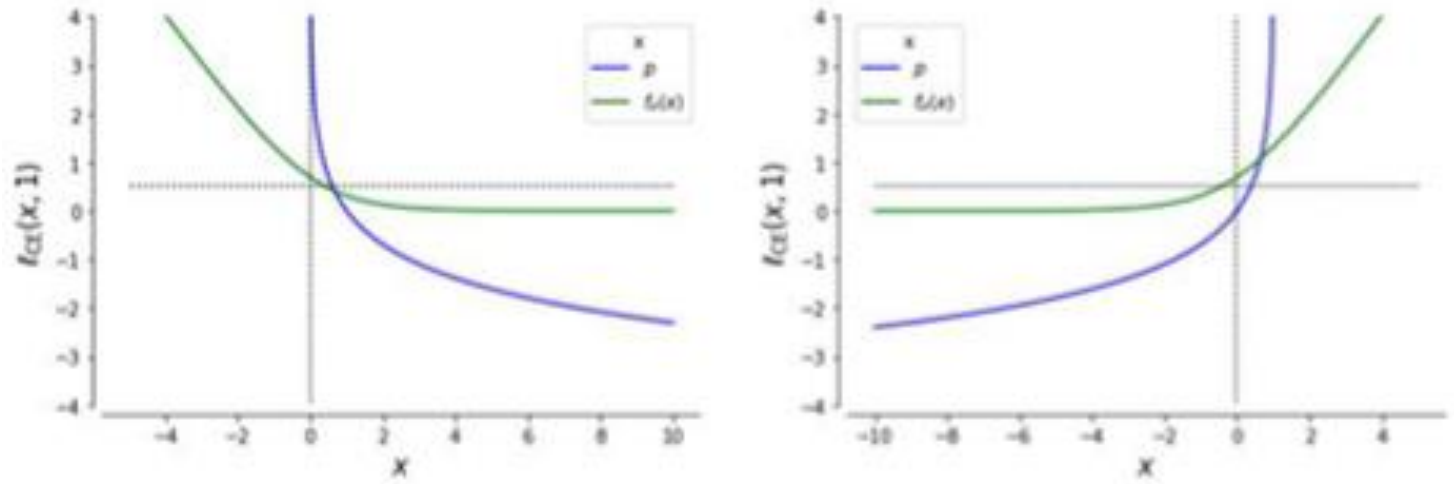
$$f_\sigma(\mathbf{x}) = \frac{e^{\mathbf{x}}}{e^{\mathbf{x}}+1}$$

Red neuronal de una capa



- Las funciones de activación son llamadas no linealidades
- Las funciones de activación se aplican a cada punto
- Producen estimaciones de probabilidad
- Se saturant
- Las derivadas desaparecen

Entropia cruzada



$$\ell_{\mathrm{CE}}(\mathbf{p}, \mathbf{t}) = -[\mathbf{t} \log \mathbf{p} + (1 - \mathbf{t}) \log(1 - \mathbf{p})]$$

- La entropía cruzada también es llamada log verosimilitud negativa o pérdida logística
- Es numéricamente inestable

Softmax

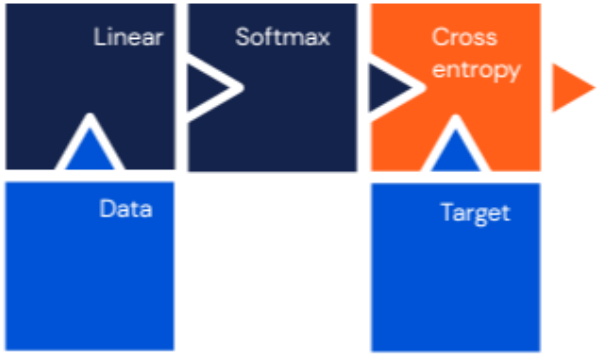
$$f_{\text{sm}}(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_{j=1}^{k} e^{\mathbf{x}_j}}$$

$$f_{\text{sm}}([x, 0]) = \left[\frac{e^{x}}{e^{x} + e^{0}}, \frac{e^{0}}{e^{x} + e^{0}}\right]$$

$$= [f_{\sigma}(x), 1 - f_{\sigma}(x)]$$

- Generalización de la sigmoide
- Produce estimación de probabilidad
- Se satura
- Las derivadas desaparecen

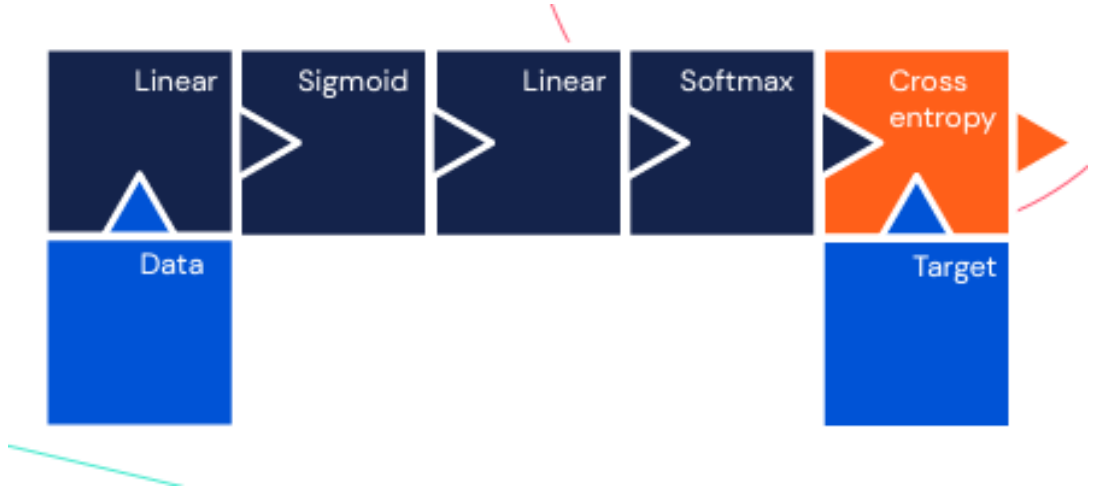
Softmax + cross entropy

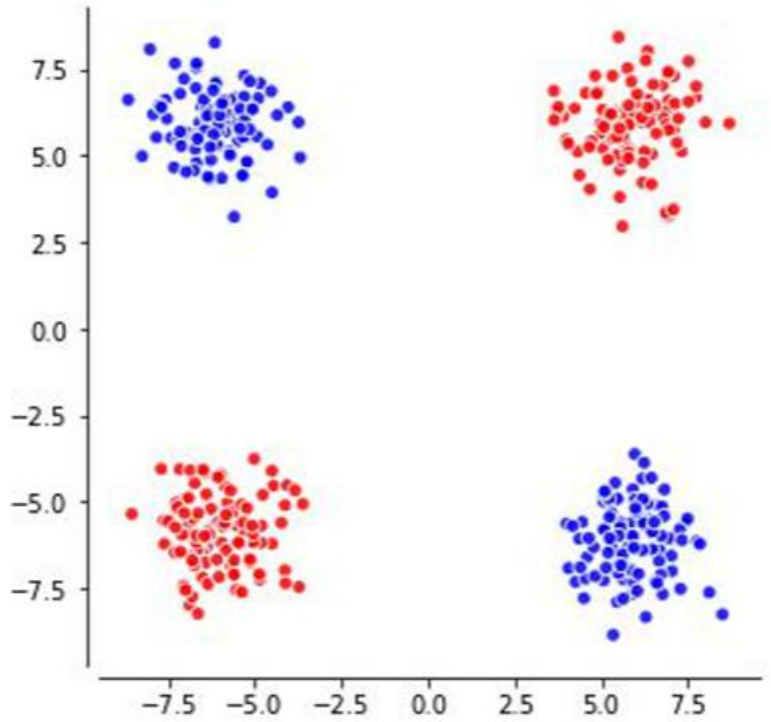


$$= -\sum_{j=1}^{k} \mathbf{t}_j \log[f_{\text{sm}}(\mathbf{x}_j)] = -\sum_{j=1}^{k} \mathbf{t}_j [\mathbf{x}_j - \log \sum_{l=1}^{k} e^{\mathbf{x}_l}]$$

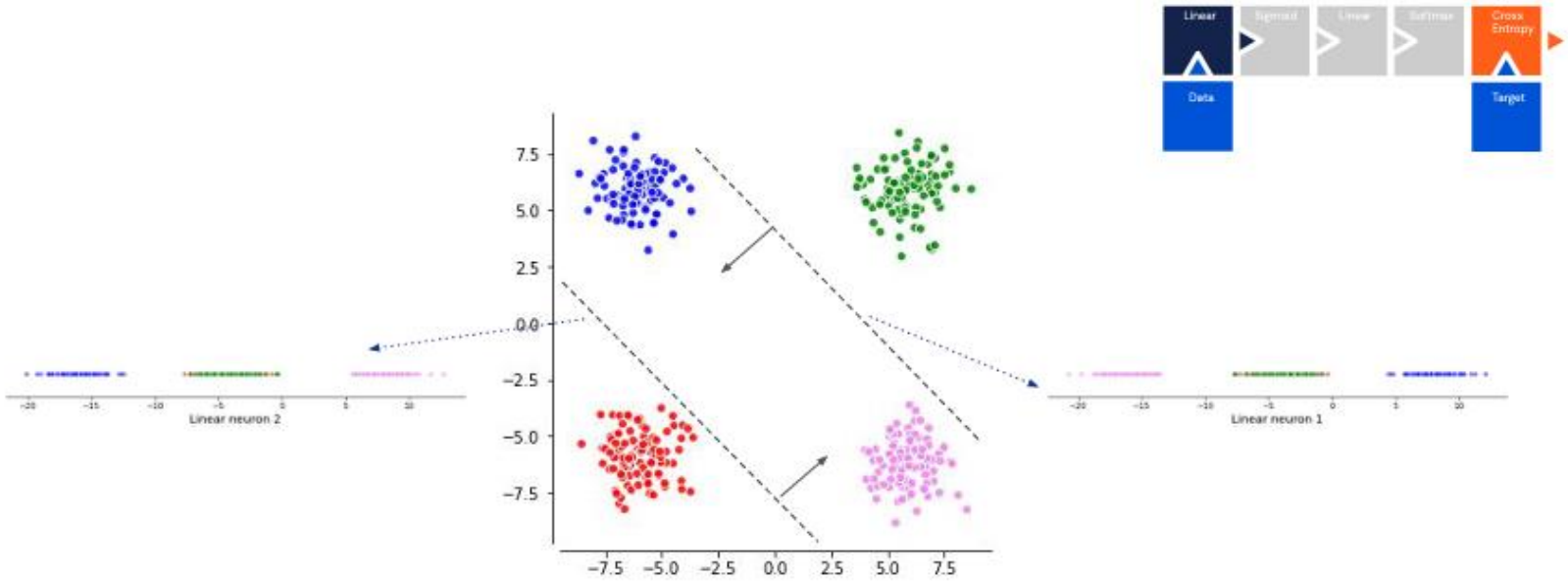
- Numéricamente estables
- Equivalente al modelo de regresión logística multinomial
- Muy utilizada en clasificación y RL

Redes de dos capas



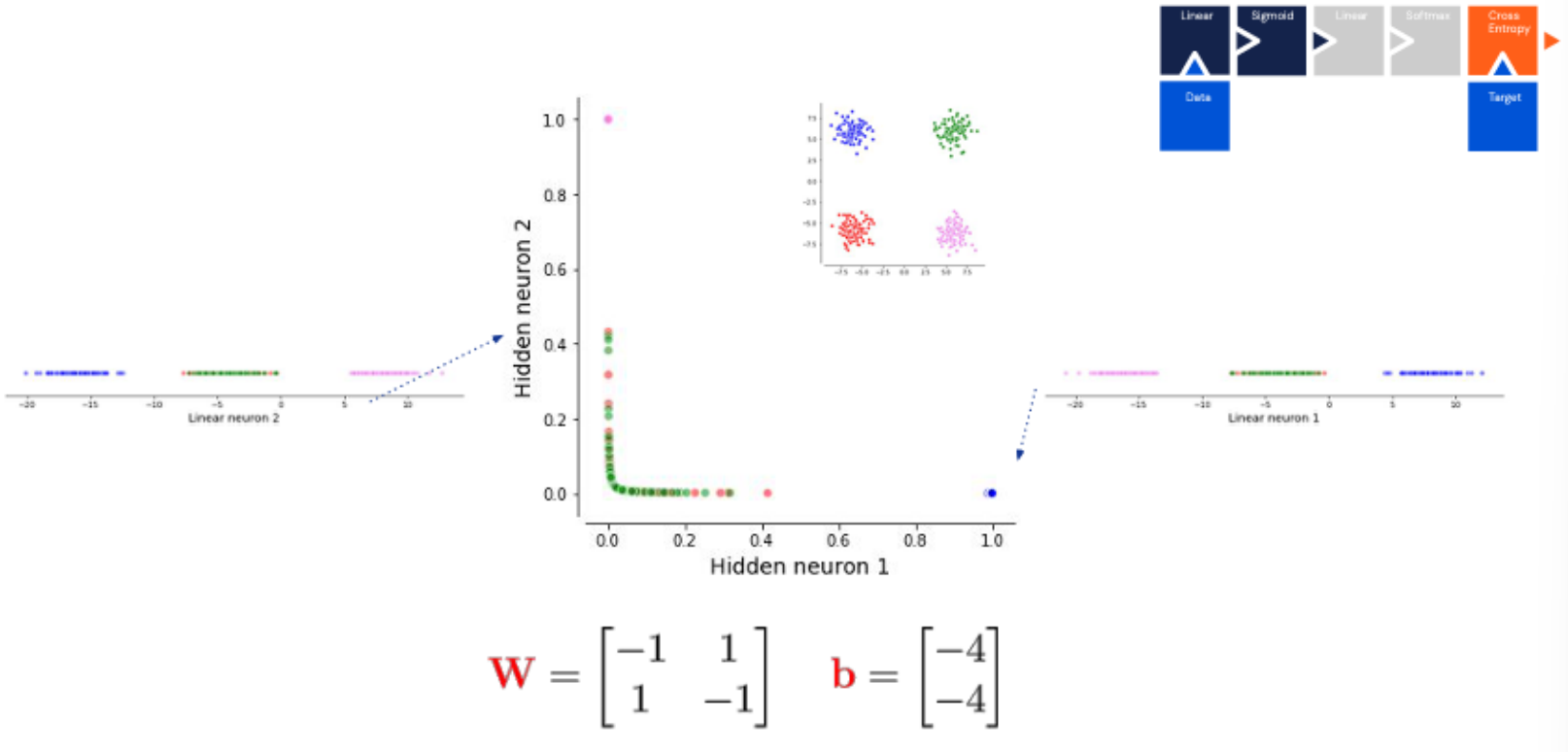


El ejemplo I



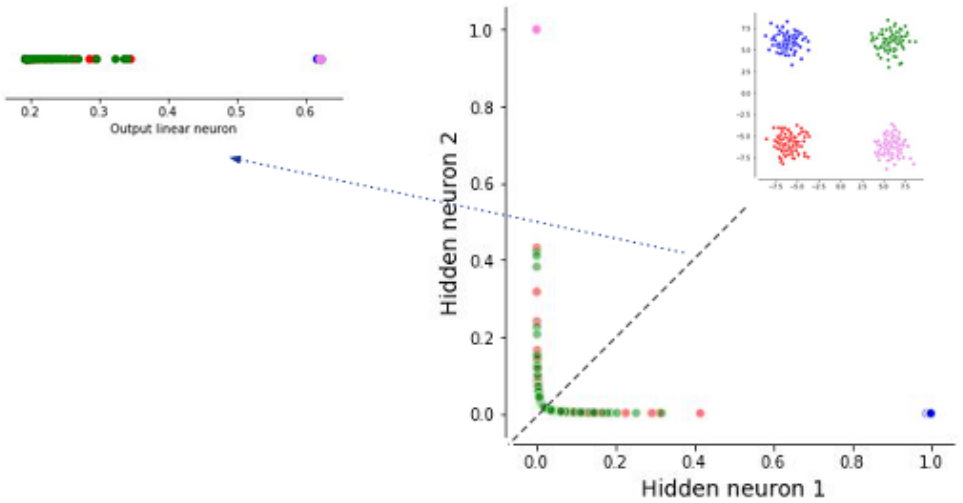
$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

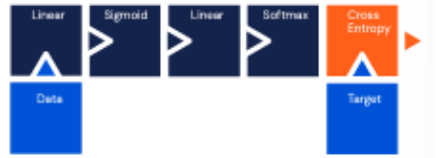
El ejemplo II



$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

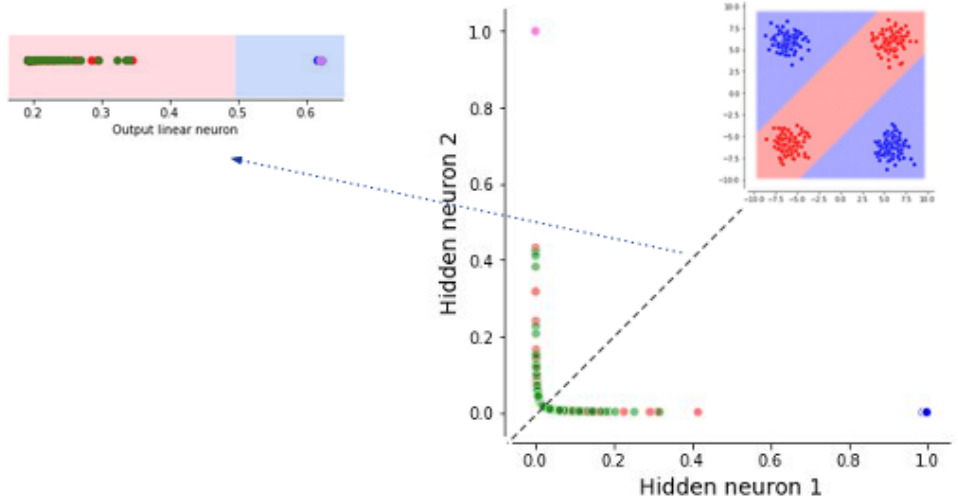
El ejemplo III





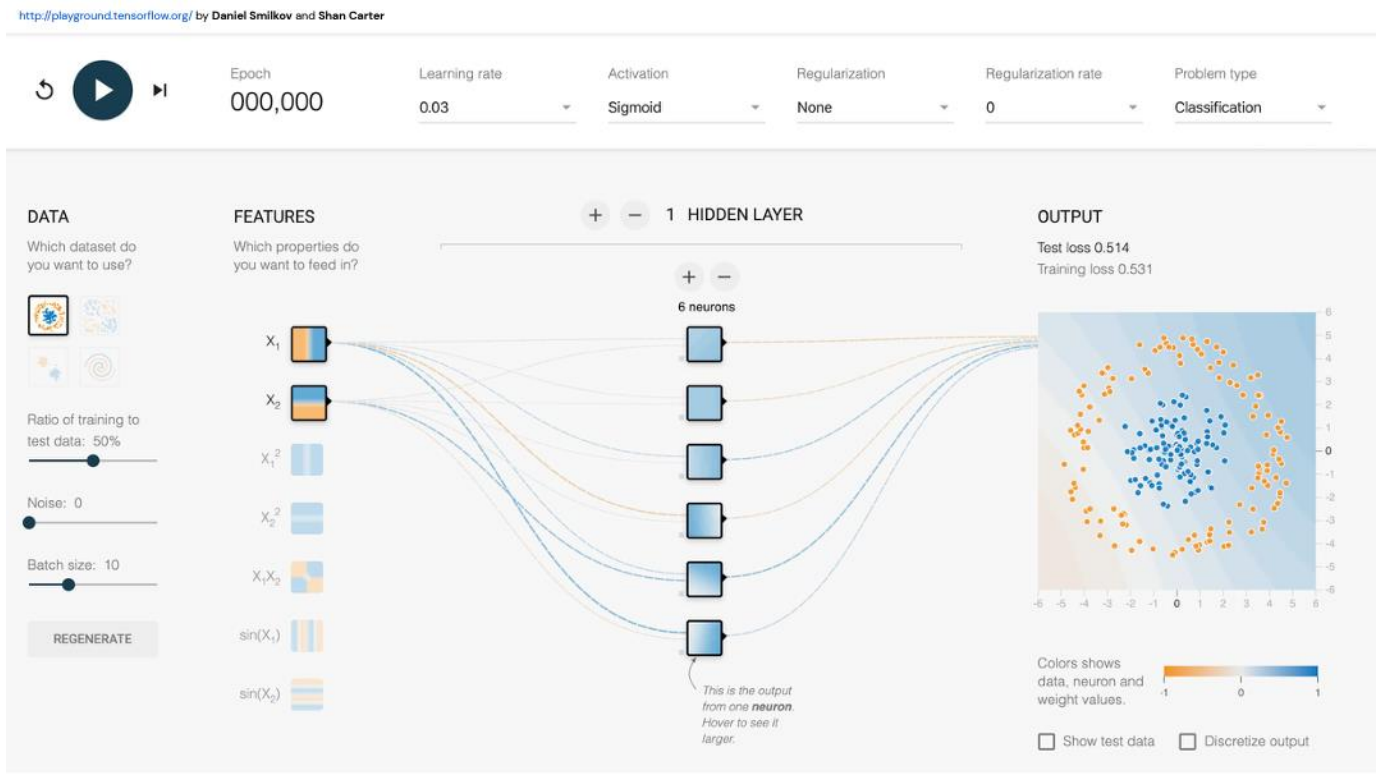
$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

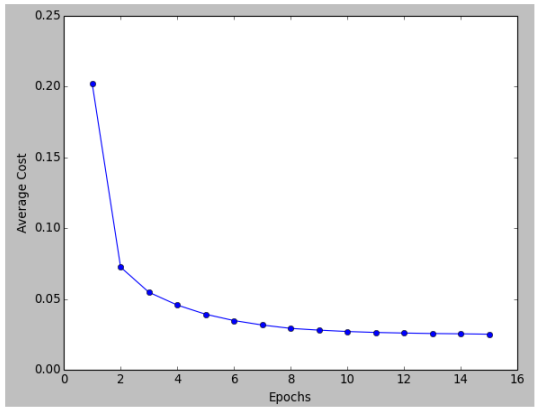
El ejemplo IV

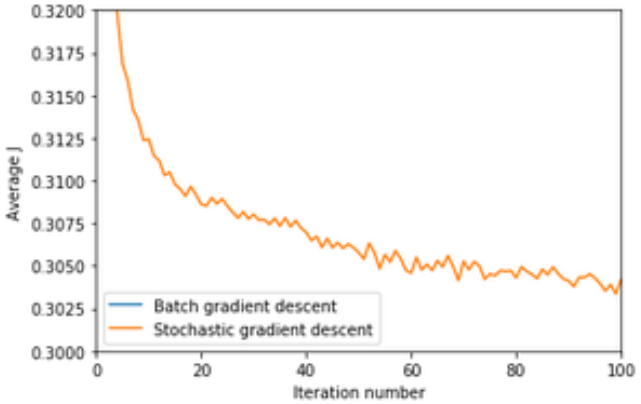


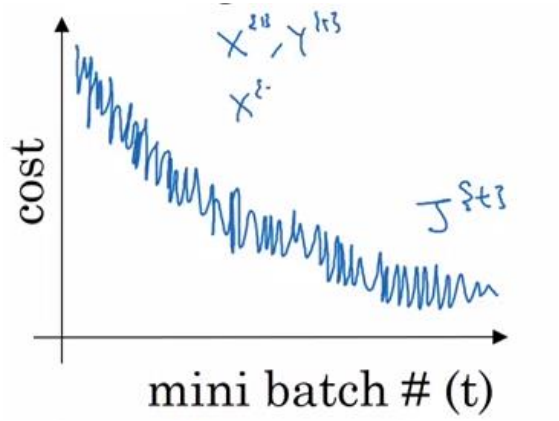
$$\mathbf{W} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} -4 \\ -4 \end{bmatrix}$$

Para ver más


http://playground.tensorflow.org/ by Daniel Smilkov and Shan Carter
Epoch
000,000
Learning rate
0.03
Activation
Sigmoid
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
6 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.514
Training loss 0.531
Colors shows data, neuron and weight values.
Show test data
Discretize output







Aprendizaje en línea

- Batch

$$\theta_{k+1} = \theta_k + \eta \sum_{i=1}^{n} x_i^T (y_i - x_i \theta_k)$$

- Online

$$\theta_{k+1} = \theta_k + \eta x_k^T (y_k - x_k \theta_k)$$

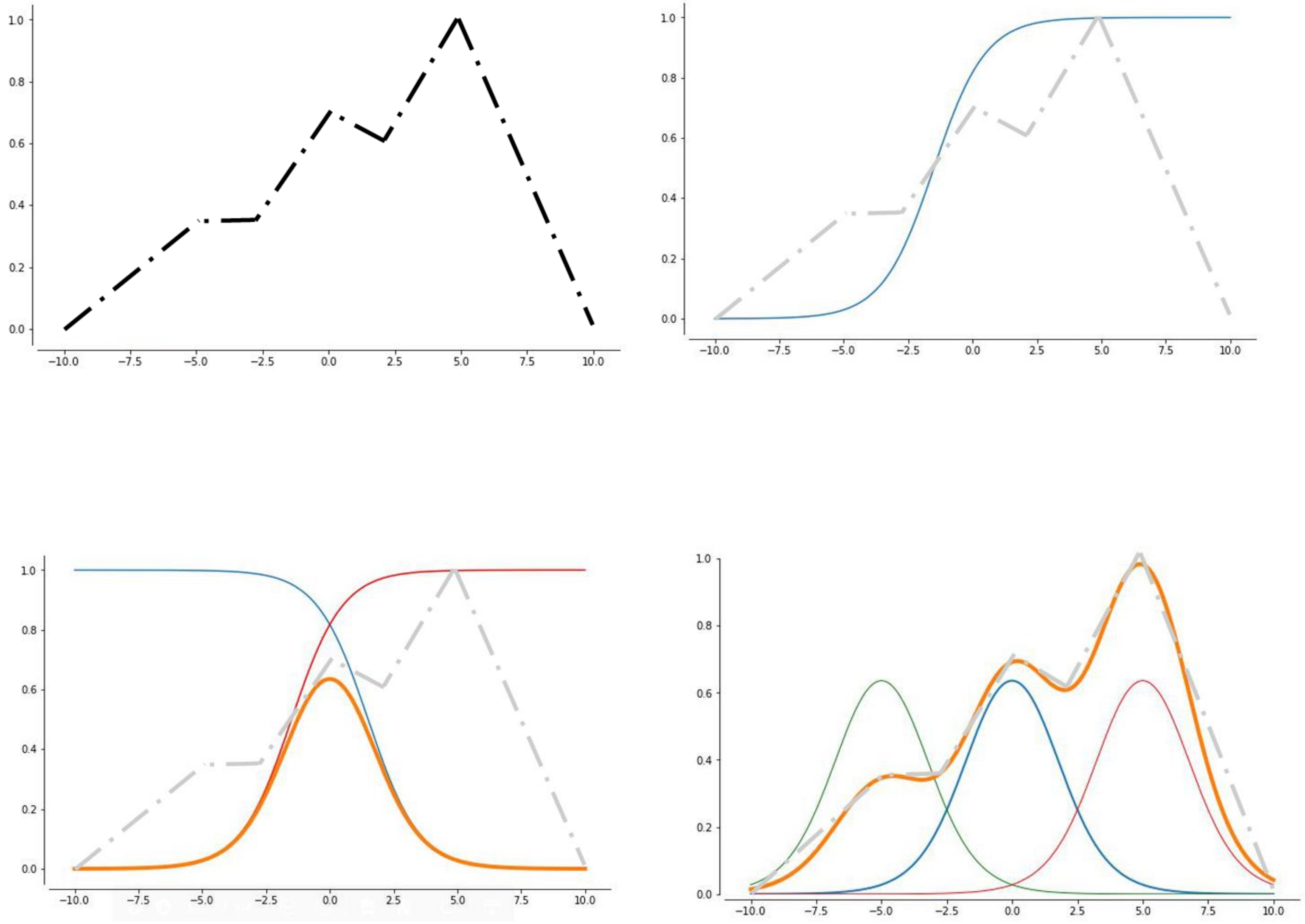
- Mini-batch

$$\theta_{k+1} = \theta_k + \eta \sum_{i=1}^{20} x_i^T (y_i - x_i \theta_k)$$

Teorema de aproximación universal

- Para cualquier función continua desde un hipercubo $[0,1]^d$ a los números reales, y para cualquier épsilon positivo, existe una red neuronal basada en sigmoide con una capa oculta que obtiene a lo más un error épsilon en el espacio de funciones
- Redes grandes puede aproximar pero no representar cualquier función suave

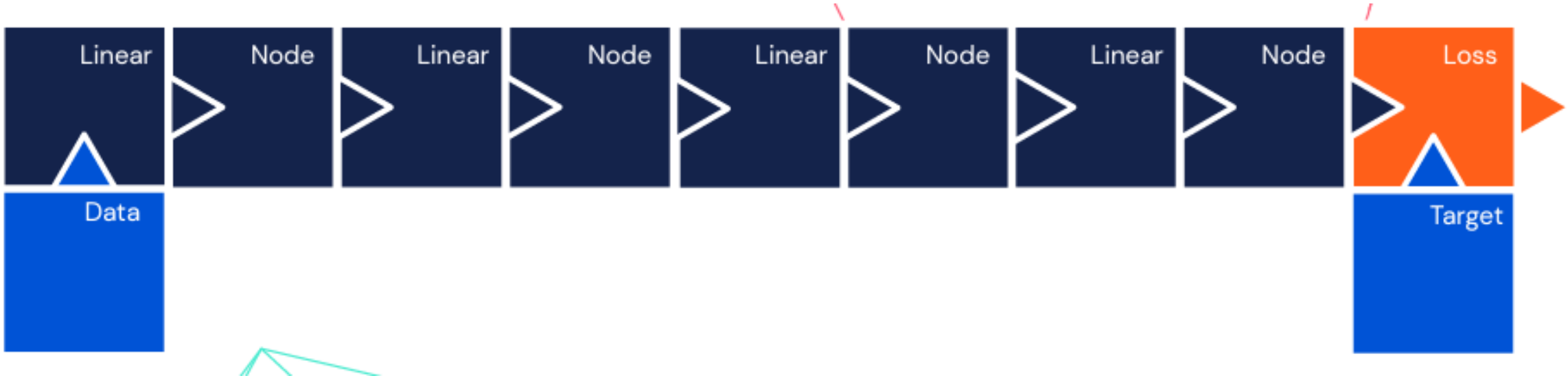
Intuición



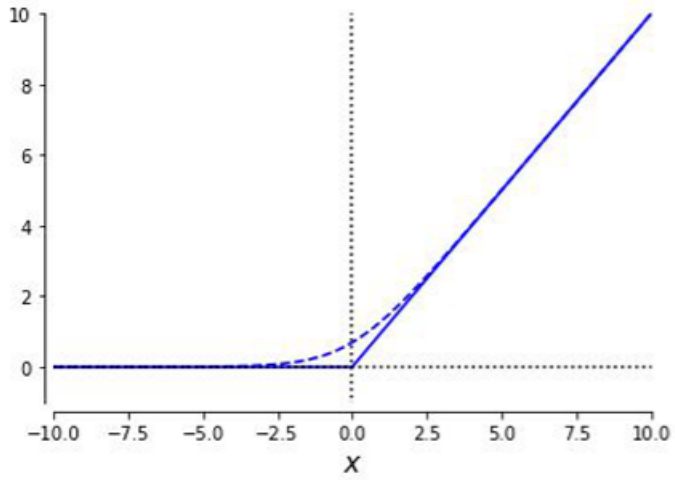
Hace 20 años…

- Solo se utilizaba una capa adicional al modelo de regresión logística
- El enfoque estaba en optimización convexa
- Uso de pocos datos

Redes neuronales profundas



Rectified Linear Unit (ReLU)

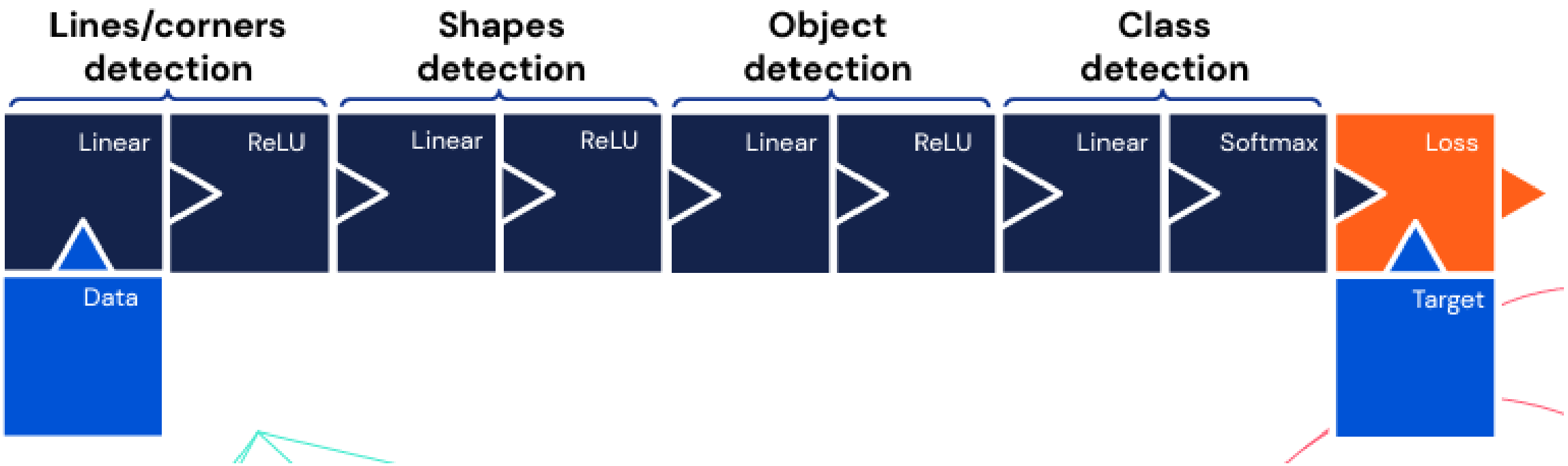


$$f_{\text{relu}}(\mathbf{x}) = \max(0, \mathbf{x})$$

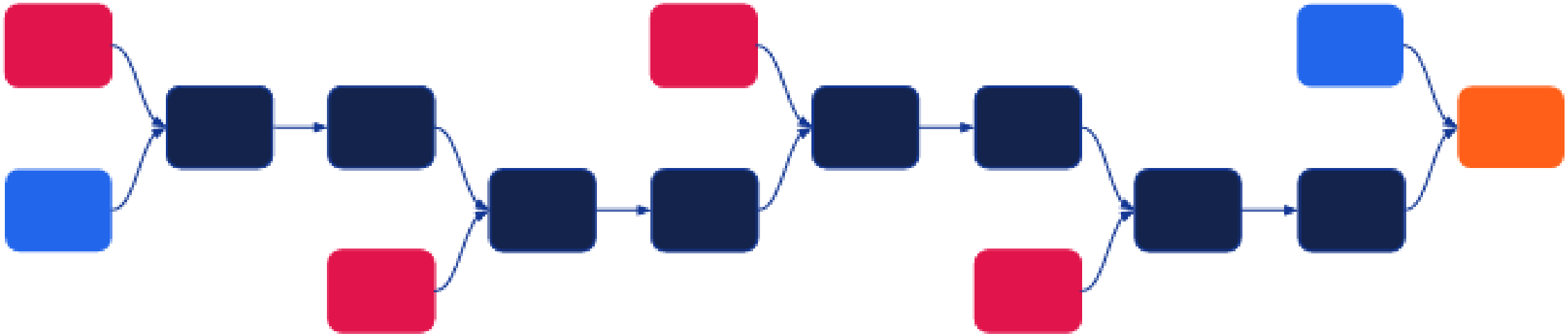
$$f_{\text{sp}}(\mathbf{x}) = \log(1 + e^{\mathbf{x}})$$

- Una de las funciones más utilizadas
- Las derivadas no desaparecen
- Puede ocurrir que neuronas mueran
- Técnicamente no es diferenciable en 0

Un ejemplo


Lines/corners detection
Shapes detection
Object detection
Class detection
Linear
ReLU
Linear
ReLU
Linear
ReLU
Linear
Softmax
Loss
Data
Target

Grafos computacionales



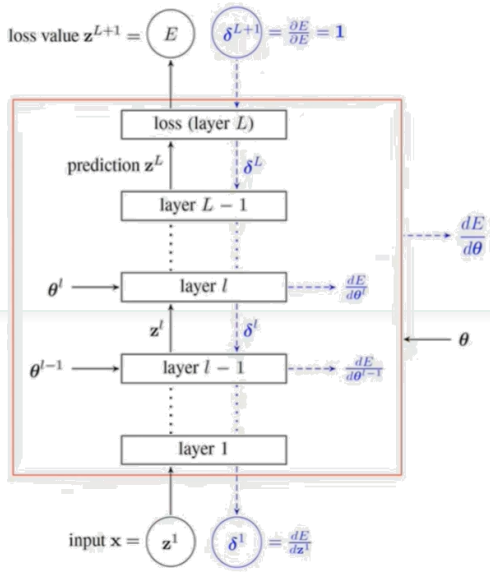
Aprendizaje

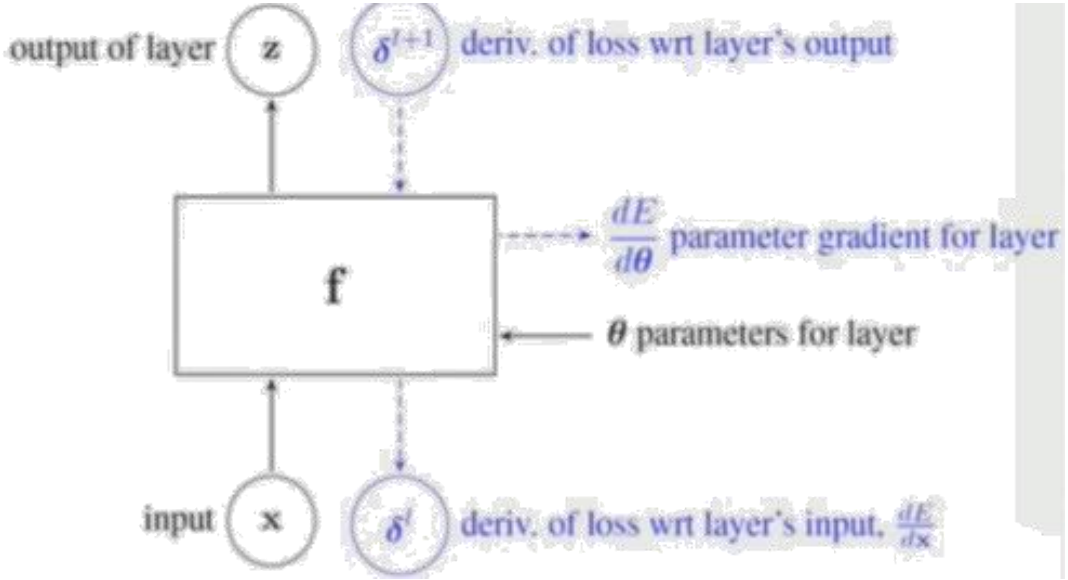
$f(\mathbf{x})$ Forward pass

$\mathbf{J}_{\mathbf{x}} f(\mathbf{x})$ Backward pass

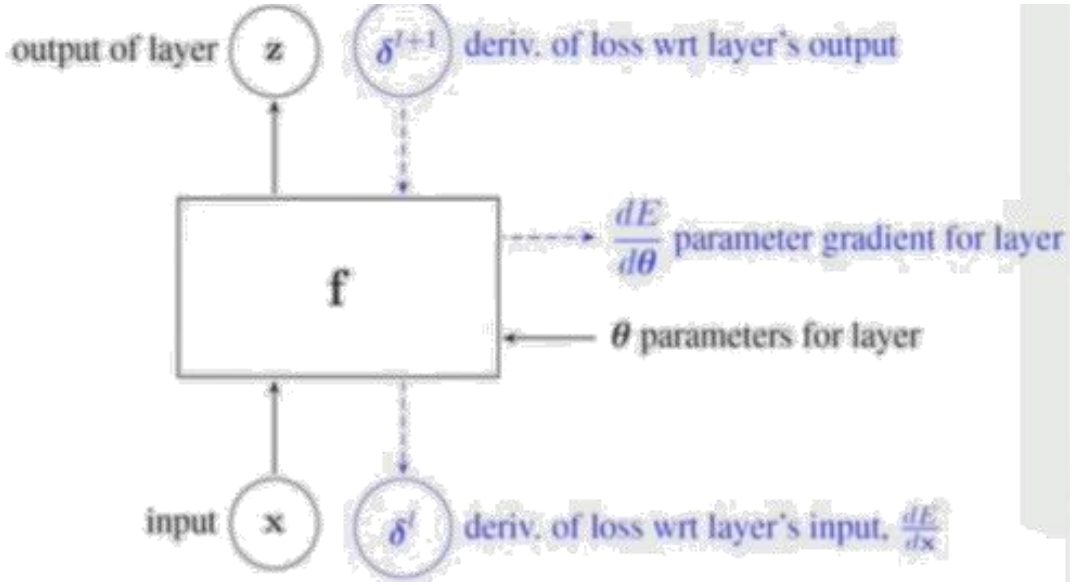
Modelo de composición

- Se necesitan 3 funciones
  - Forward
  - Backward
  - Derivada
- ¿Como componer las capas?
  - Secuencial
  - Recursivas
  - Redes





Capa lineal

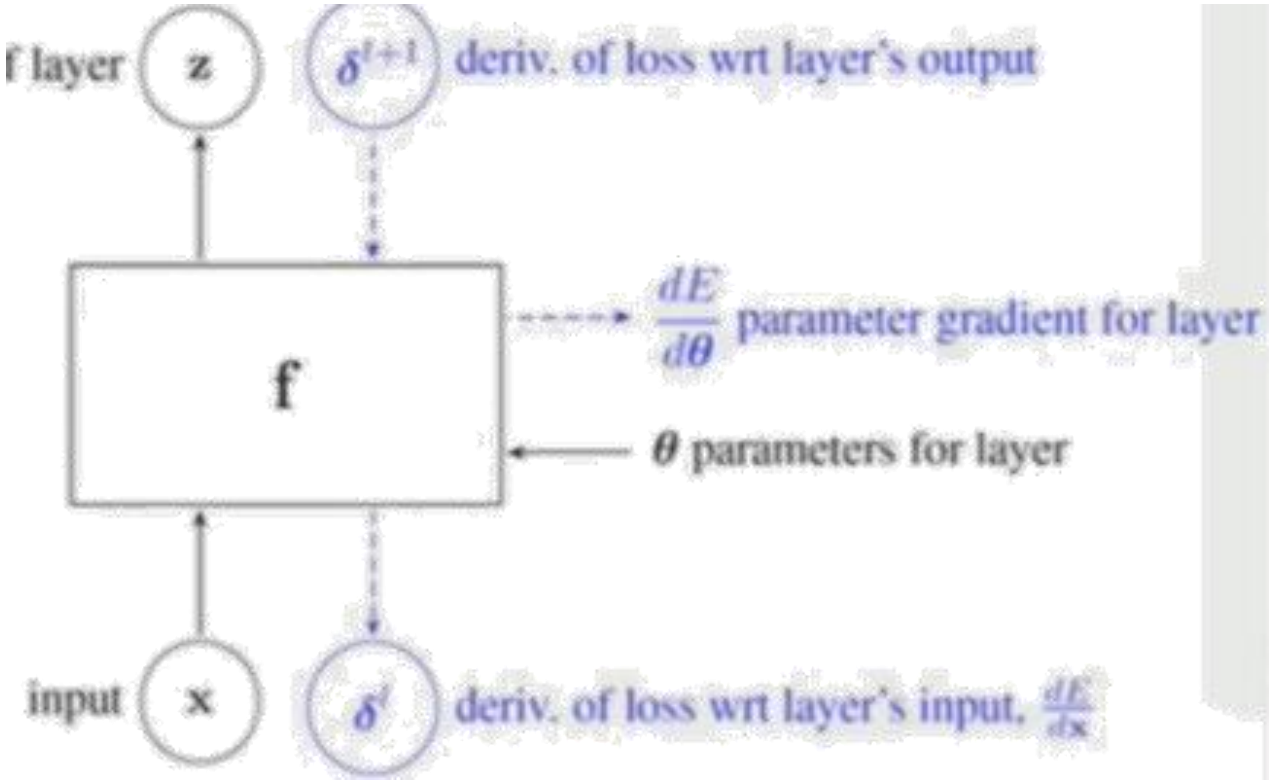


$$z_j = f_j(\mathbf{x};\boldsymbol{\theta}_j) = \sum_i x_i \theta_{ij}$$

$$\delta_i^l = \sum_j \delta_j^{l+1} \frac{\partial f_j(\mathbf{x};\boldsymbol{\theta}_j)}{\partial x_i} = \sum_j \delta_j^{l+1} \theta_{ij}$$

$$\frac{\partial E}{\partial \theta_{ij}} = \sum_j \delta_j^{l+1} \frac{\partial f_j(\mathbf{x};\boldsymbol{\theta}_j)}{\partial \theta_{ij}} = \delta_j^{l+1} x_i$$

Capa ReLU



- $z_j = f_j(x_j) = \max(0, x_j)$
- $\delta_i^l = \sum_j \delta_j^{l+1} \frac{\partial f_j(x_j)}{\partial x_i} = \delta_j^{l+1} I_{[x_i>0]}$

Beneficios de las aproximaciones por redes neuronales profundas

Los aproximadores de función de valor lineal suponen una combinación lineal de las características

Esto funciona si tenemos “buenas características”

Alternativamente, utilizar una clase de funciones más rica sin definir explícitamente las características

Las redes profundas

- distribuidas
- Son aproximadores universales
- Pueden requerir menos nodos
- Pueden aprender con

Redes neuronales convolucionales

Son muy usadas en visión

Nos permite tomar directamente los pixeles de una imagen para tomar decisiones

Red neuronal convolucional

- Considera una estructura local y extracción de características
- No es completamente conectada
- Tiene pesos compartidos para reducción de parámetros

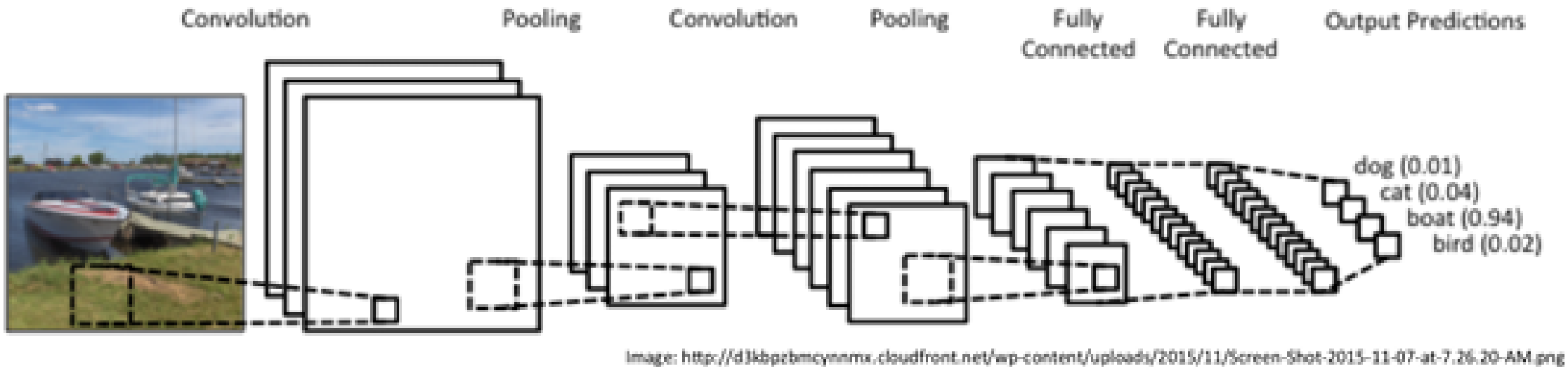
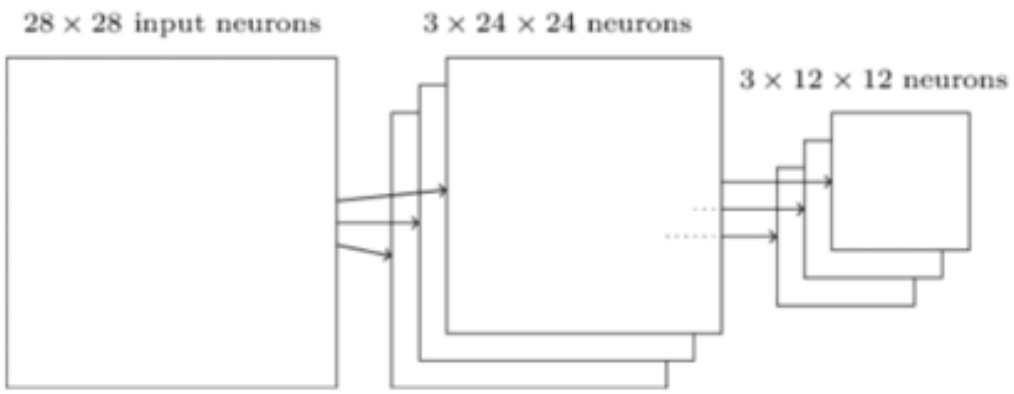
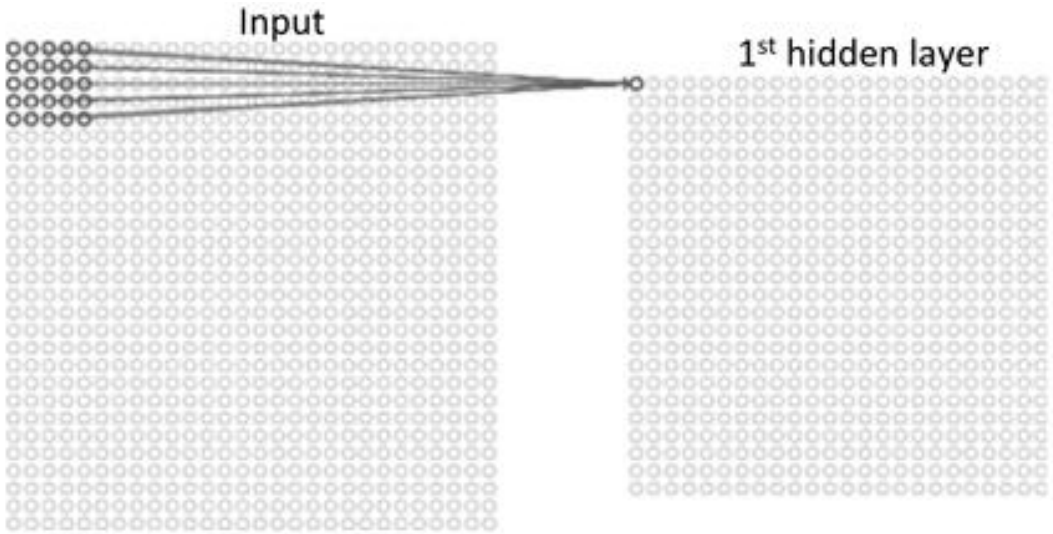


Image: http://d3kbpzbmcynnmx.cloudfront.net/wp-content/uploads/2015/11/Screen-Shot-2015-11-07-at-7.26.20-AM.png

A detalle

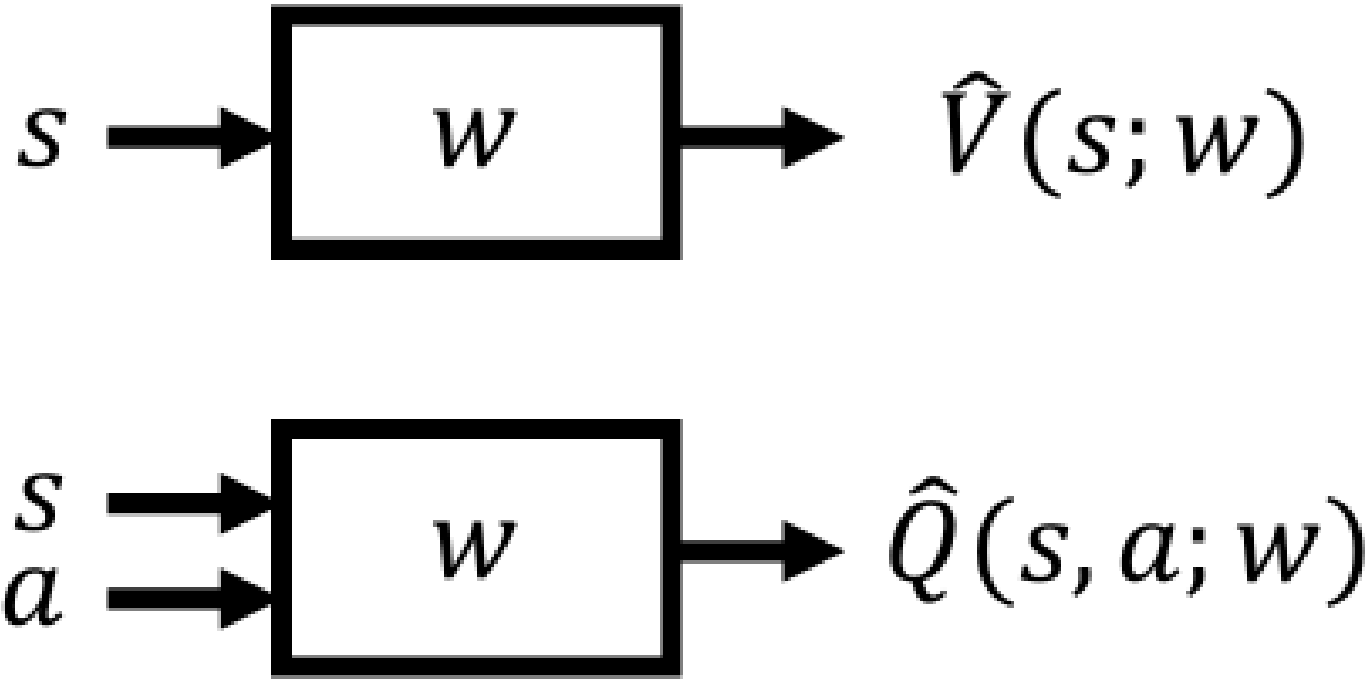

28 × 28 input neurons
3 × 24 × 24 neurons
3 × 12 × 12 neurons


Input
1st hidden layer

Redes Q profundas (DQN)

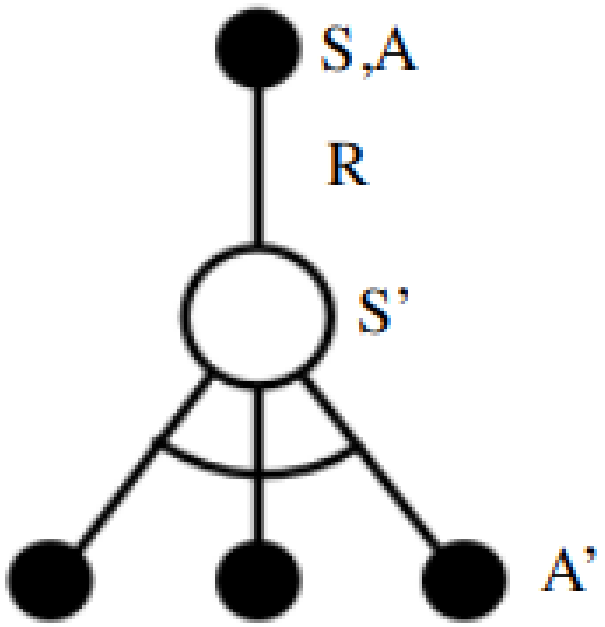
Representan una función estado acción por medio de una red Q con pesos w

$$\hat{Q}(s, a; \boldsymbol{w}) \approx Q(s, a)$$



Algoritmo de control de aprendizaje Q

- $q_{t+1}(S_t, A_t) \leftarrow q_t(S_t, A_t) + \alpha_t(R_{t+1} + \gamma \max_{a'} q_t(S_{t+1}, a') - q_t(S_t, A_t))$
- Teorema
  - Control con aprendizaje Q converge a la función acción valor óptima q → q^* en el limite



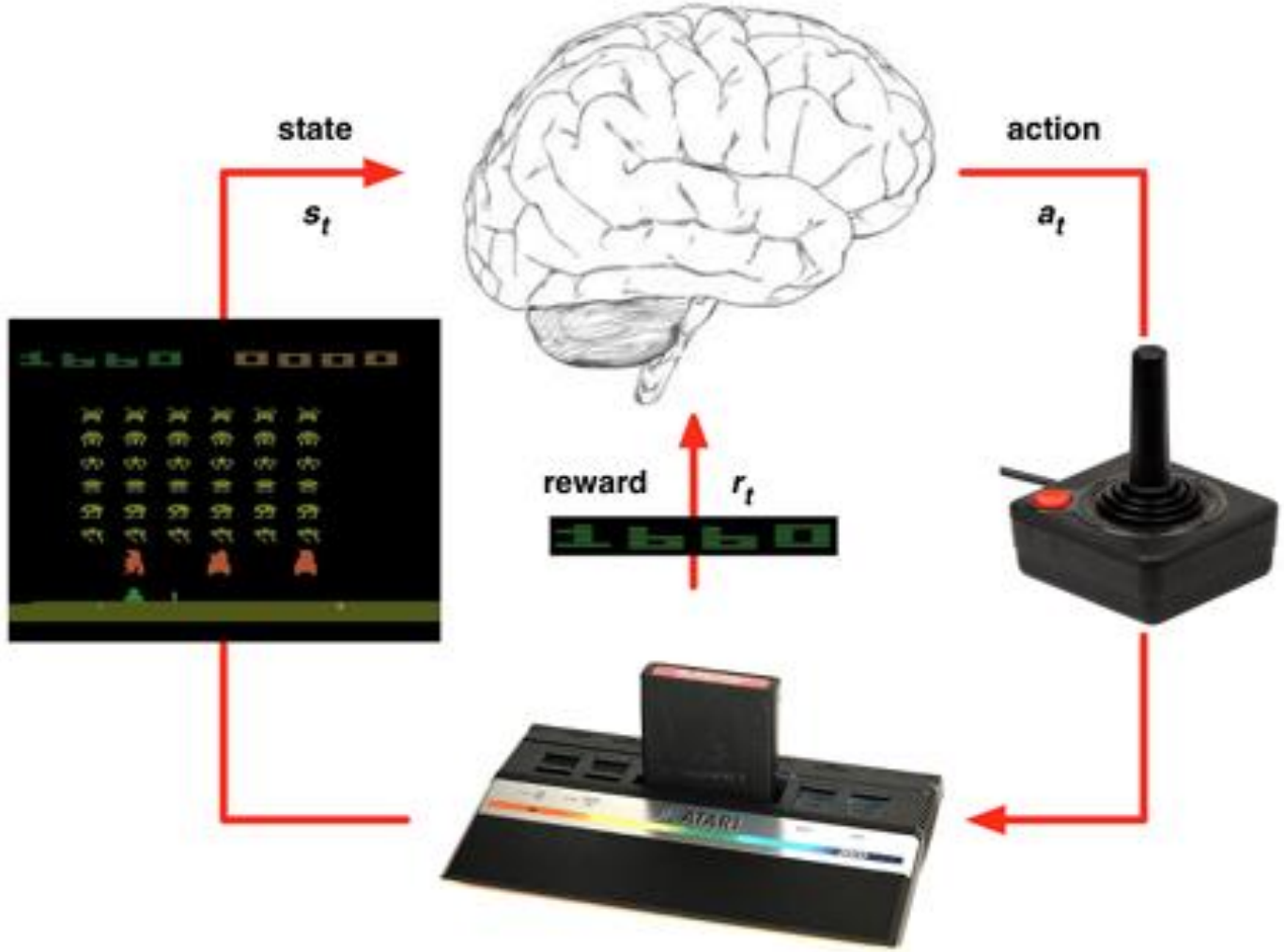
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

DQN en Atari

- Aprender $q(s, a)$ de pixels
- La entrada pixels de los últimos 4 frames
- La salida es $q(s, a)$ para 18 acciones de joystick/botones
- La recompensa cambia en cada paso



Aprendizaje Q profundo

- DQN utiliza repetición de experiencia y Q-targets fijos
- Tomar una acción a_t con respecto a la política ϵ voraz
- Almacenar transiciones $(s_t, a_t, r_{t+1}, s_{t+1})$ en la memoria de repetición D
- Muestrear un mini-batch aleatorio (s, a, r, s') de D
- Calcular los Q-targets con respecto a parámetros viejos w^-
- Optimizar error cuadrático medio entre la red Q y el Q-target
- Utilizar gradiente descendente estocástico para optimizar los parámetros

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i}\left[\left(r + \gamma \max_{a'} q(s', a'; w_i^-) - q(s', a'; w_i)\right)^2\right]$$

Algunos links útiles

- https://colab.research.google.com/drive/1XQu1mUbGtvkQY-D7_YCOZlRzSnjp4u9f?usp=sharing
- Maxim Lapan, Deep Reinforcement Learning Hands-On
  - Cap 3 PyTorch
  - Cap 6 Deep Q-Networks
  - Cap 8 DQN Extensions

Para la otra vez…

- Neuro evolución
- Estrategias evolutivas para aprendizaje por refuerzo