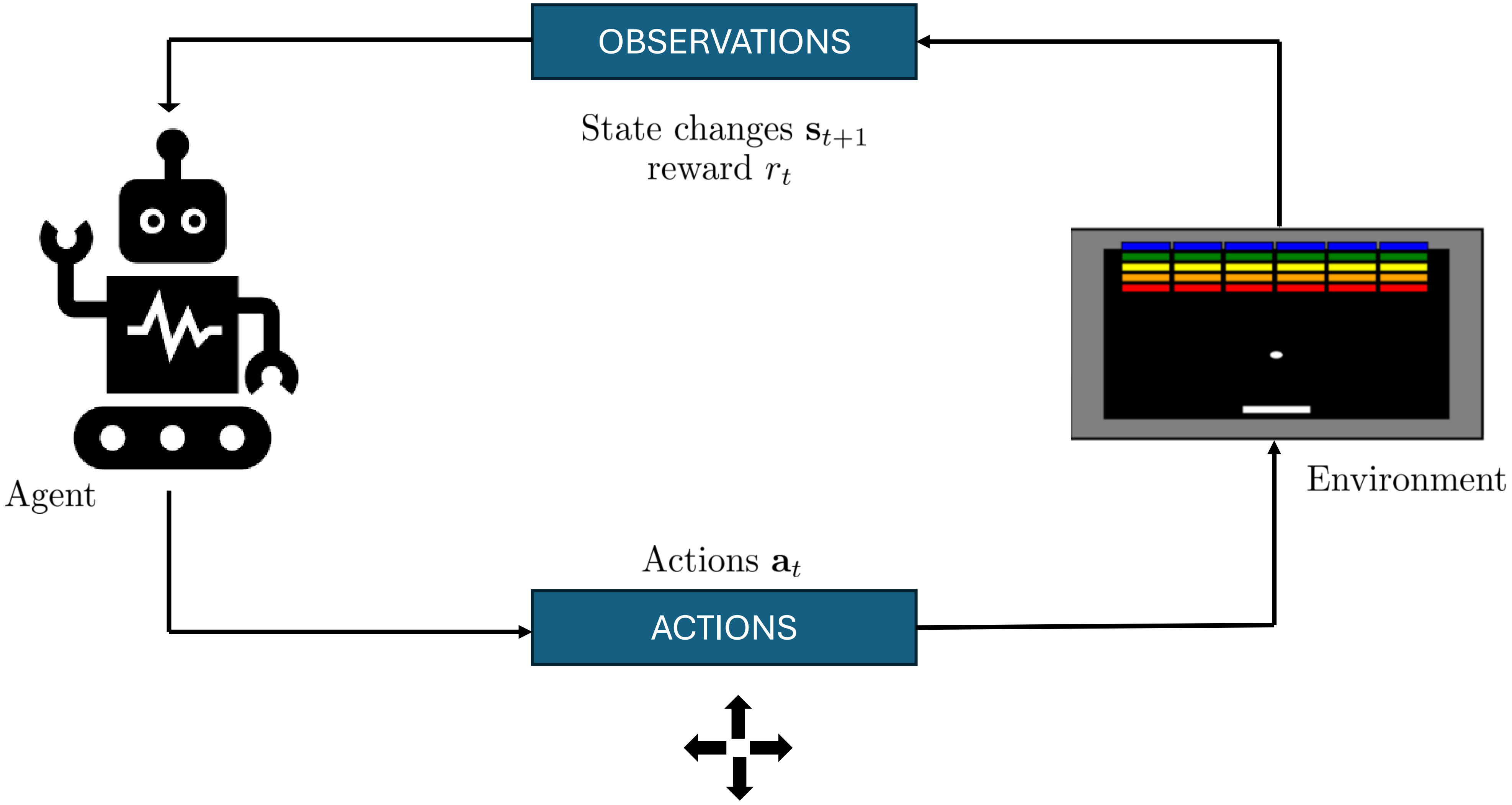
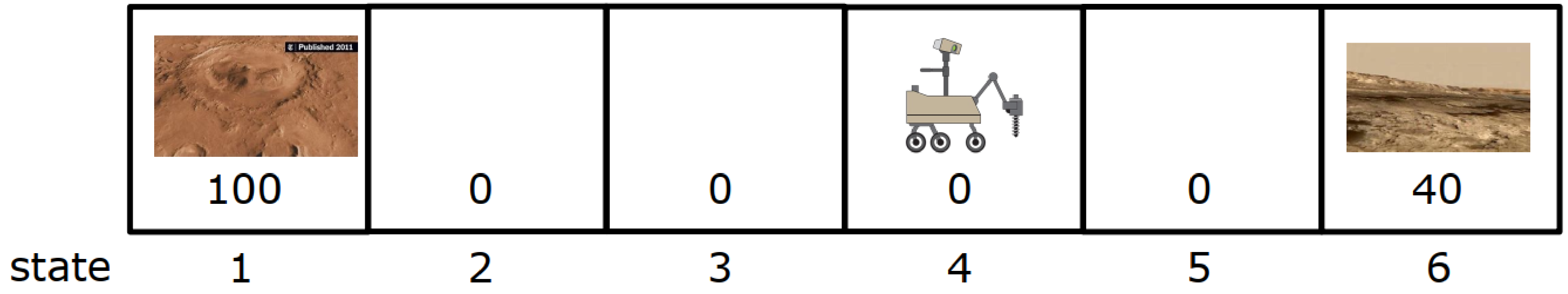
Lecture 12
Aprendizaje por refuerzo

Aprendizaje por refuerzo



Aprendizaje por refuerzo

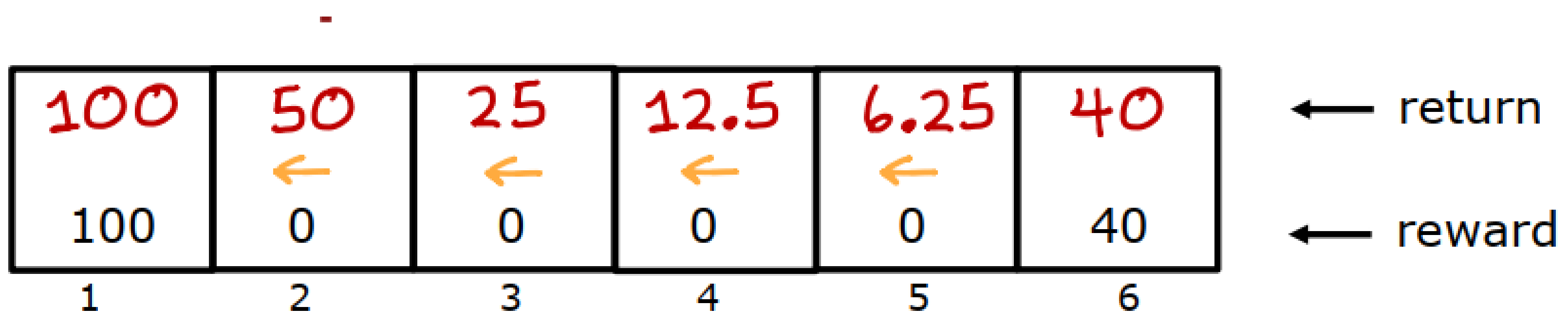
Retorno



$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

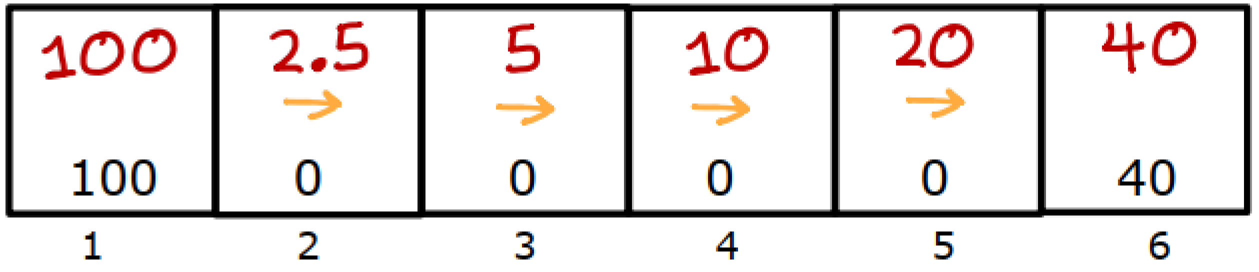
$0 < \gamma < 1$: Factor de descuento

Ejemplo de retorno

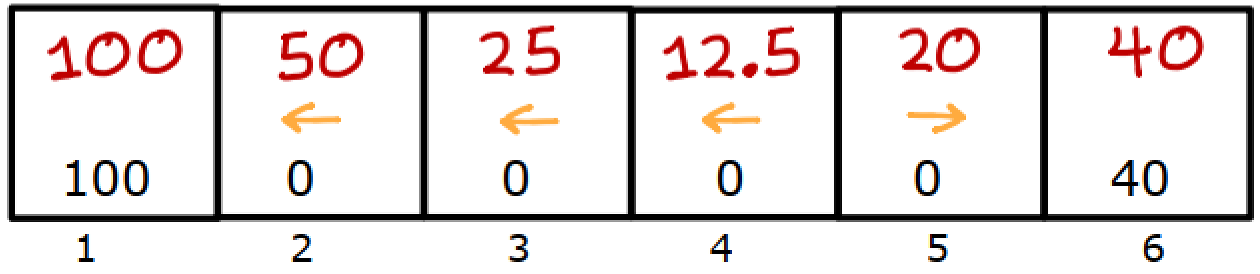


$\gamma = 0.5$

El retorno depende de las acciones que se tomen



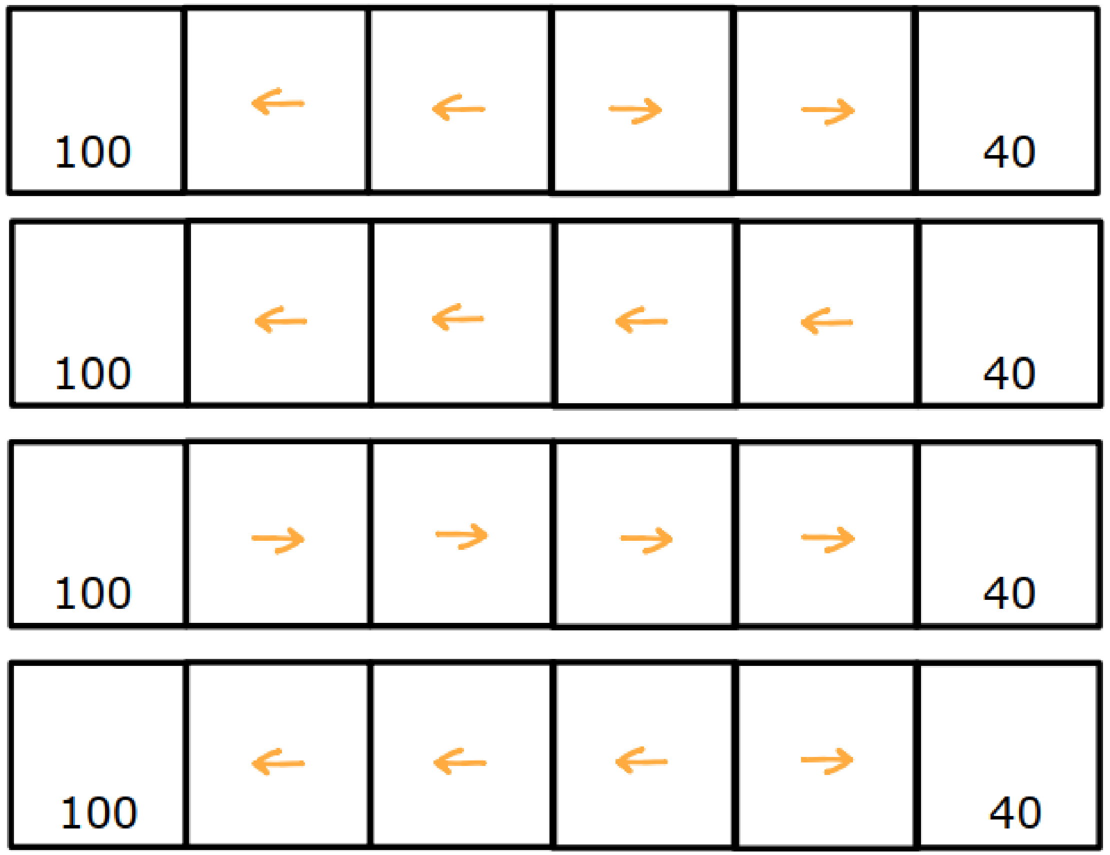
$$\overset{4}{0} + (0.5)0 + (0.5)^2 40 = 10$$



$$0 + (0.5)40 = 20$$

Política (Policy)

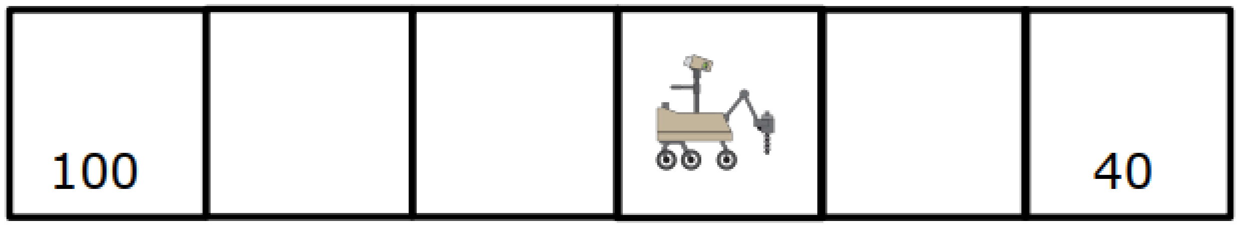
$$\begin{array}{ccc} & \text{policy} & \\ \text{state} & \rightarrow & \text{action} \\ s & \pi & a \end{array}$$



$$\pi(s) = a$$

Una política es una función $\pi(s) = a$ que mapea de estados a acciones, dice qué acción a tomar en un estado s,

Objetivo del aprendizaje por refuerzo



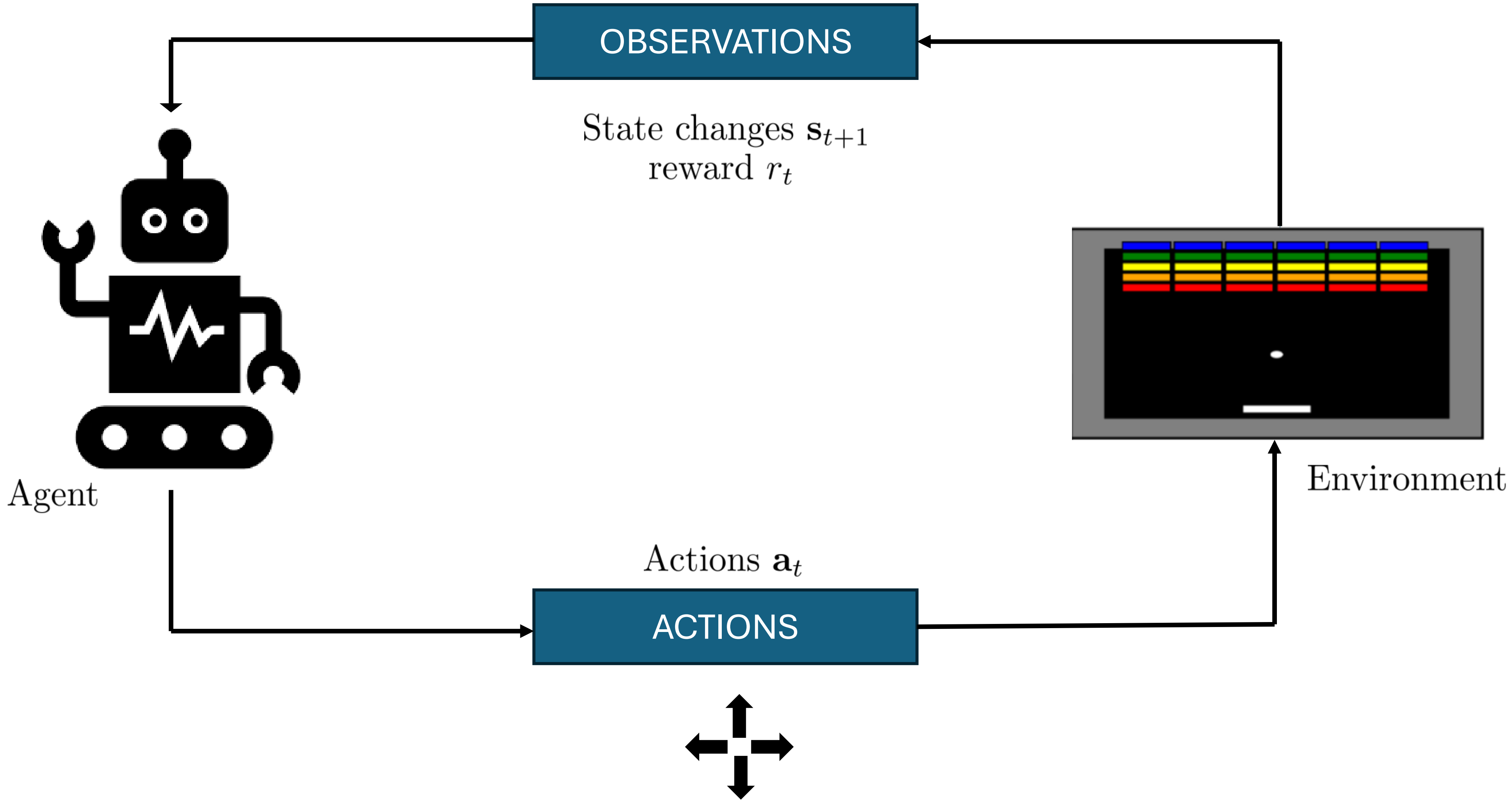
En última instancia, el agente necesita una **política** $\pi(s)$, para inferir la **mejor acción a tomar** en su estado s

Estrategia: la política debe elegir una acción que maximice la recompensa futura

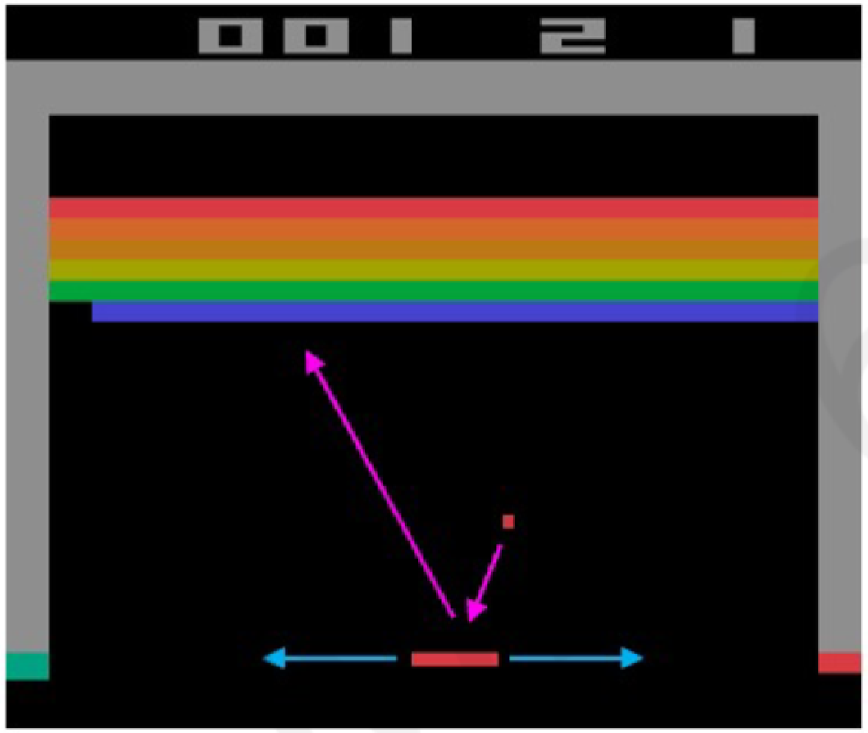
$$\pi^*(s) = \arg\max_a Q(s,a)$$

Hallar una política π que nos diga qué acción tomar $(a = \pi(s))$ para maximizar el retorno

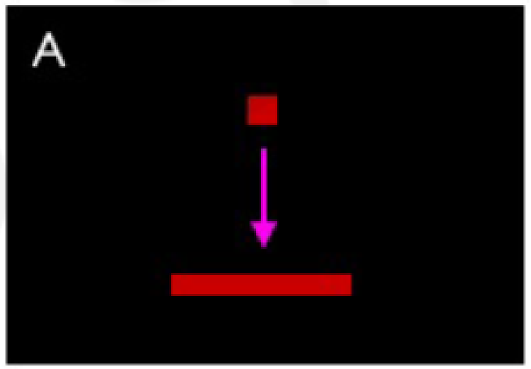
Markov Decision Process (MDP)

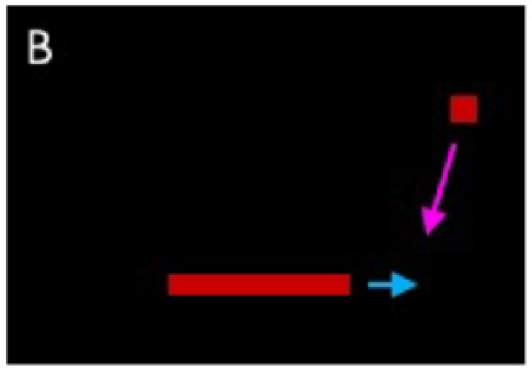


Función Q



¿Cómo estimar los Q-values?





¿Qué par (s,a) genera el mayor Q?

Función Q: Acción-estado

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

$$0 < \gamma < 1 : \text{Factor de descuento}$$

La recompensa total R_t es la suma descontada de todas las recompensas obtenidas desde el tiempo t

$$Q(s_t, a_t) = R(s) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

Retorno si:

- Comienza en el estado s
- Toma la acción a (una vez)
- Se comporta de forma óptima de ahí en adelante

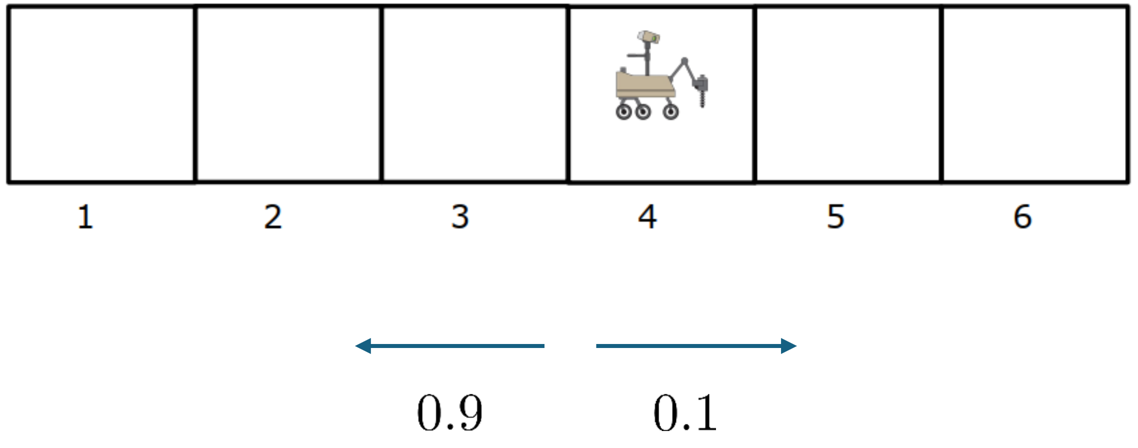
Ecuación de Bellman

$$Q(s_t, a_t) = R(s) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

Retorno si:

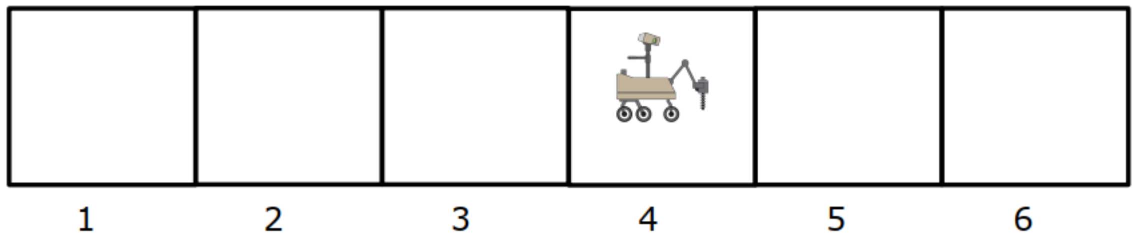
- Comienza en el estado s
- Toma la acción a (una vez)
- Se comporta de forma óptima de ahpi en adelante

Ambiente estocástico



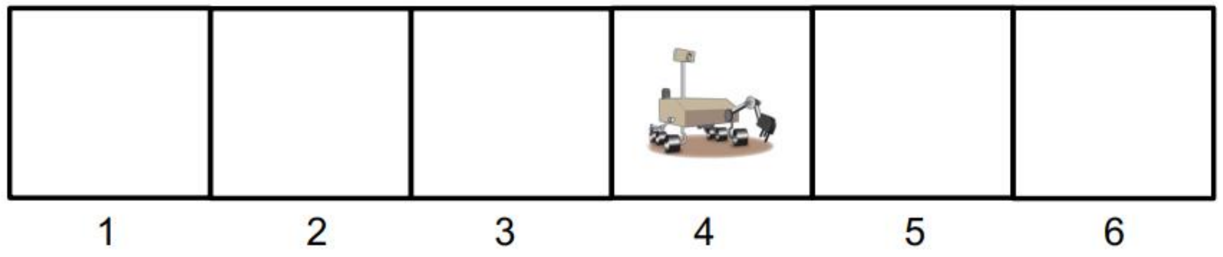
Retorno esperado

$$Q(s_t, a_t) = R_t + \gamma \mathbb{E}[\max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]$$

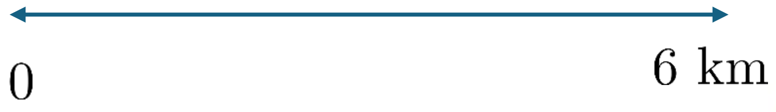


Espacio de estados continuos

Estados discretos



Estados continuos



$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

Algoritmos de aprendizaje por refuerzo

Value learning

Hallar $Q(s, a)$

$a = \arg\max_a Q(s, a)$

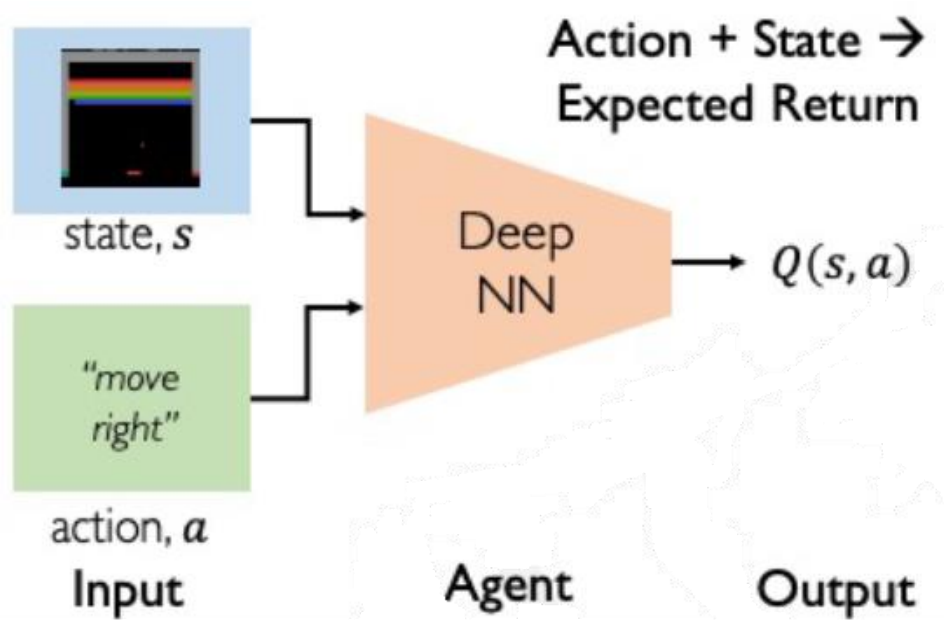
Policy learning

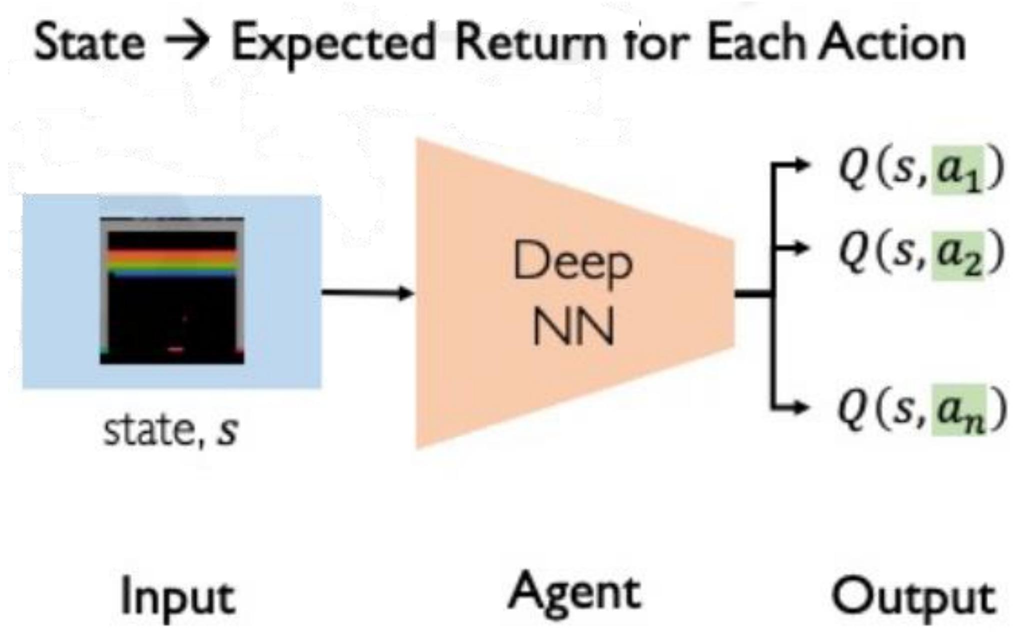
Hallar $\pi(s)$

Muestrea $a \sim \pi(s)$

Deep Q networks (DQN)

¿Cómo utilizar dnns para modelas la función Q?

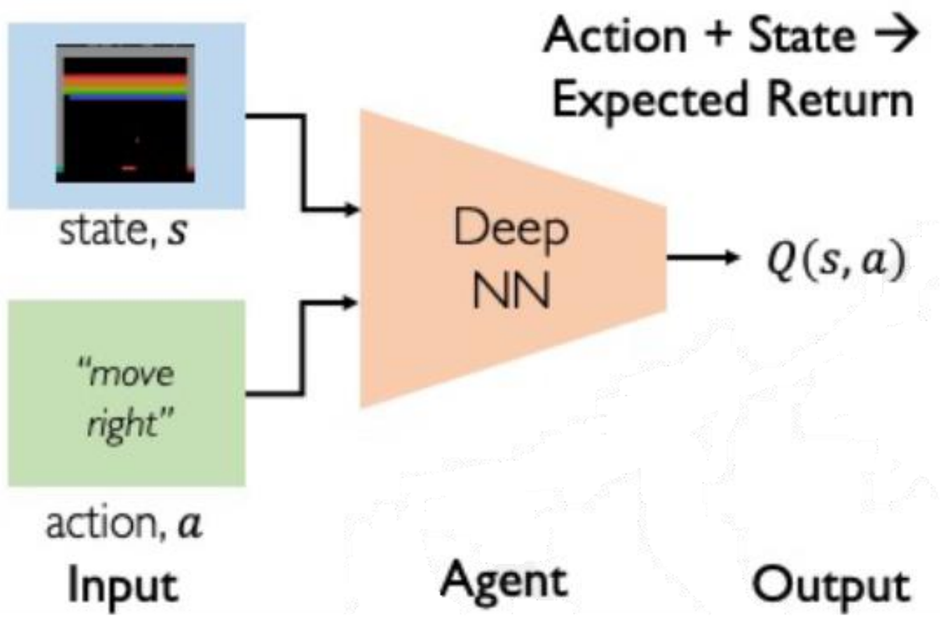


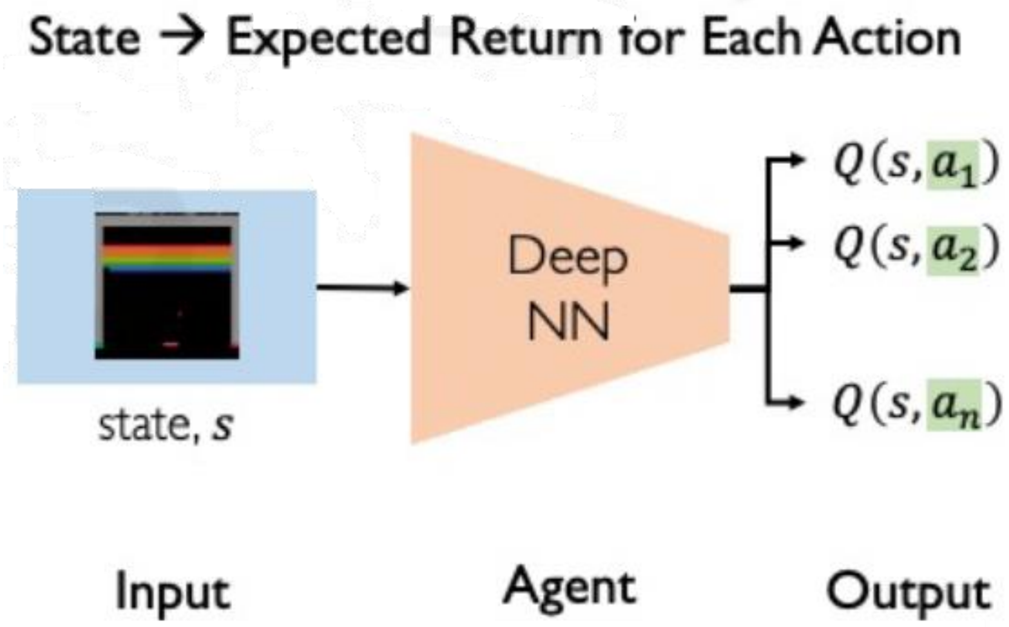


¿Qué pasa si tomamos las mejores acciones?
Maximizamos el retorno → entrenamos el agente

Solución: Deep Q networks (DQN)

¿Cómo utilizar dnns para modelas la función Q?

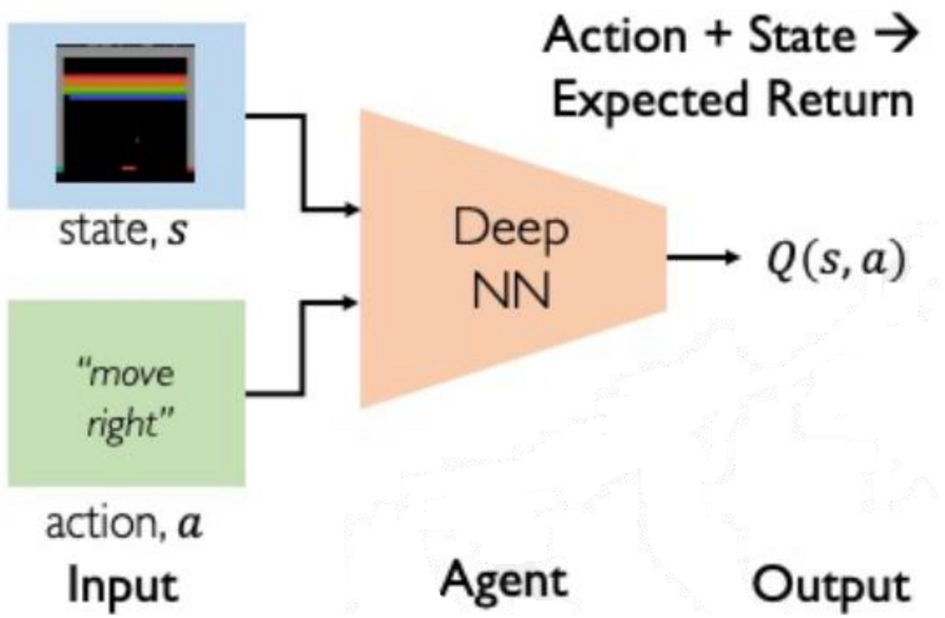


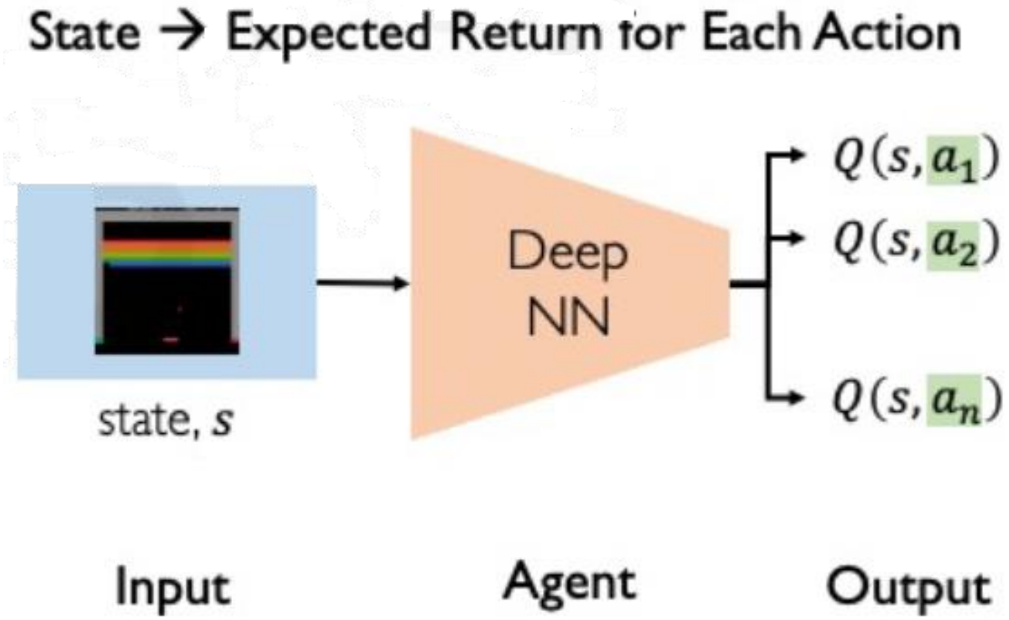


$$\overbrace{Q(s,a)}^{\text{predicted}}$$

Solución: Deep Q networks (DQN)

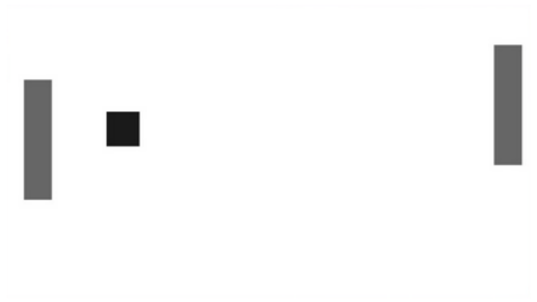
¿Cómo utilizar dnns para modelas la función Q?

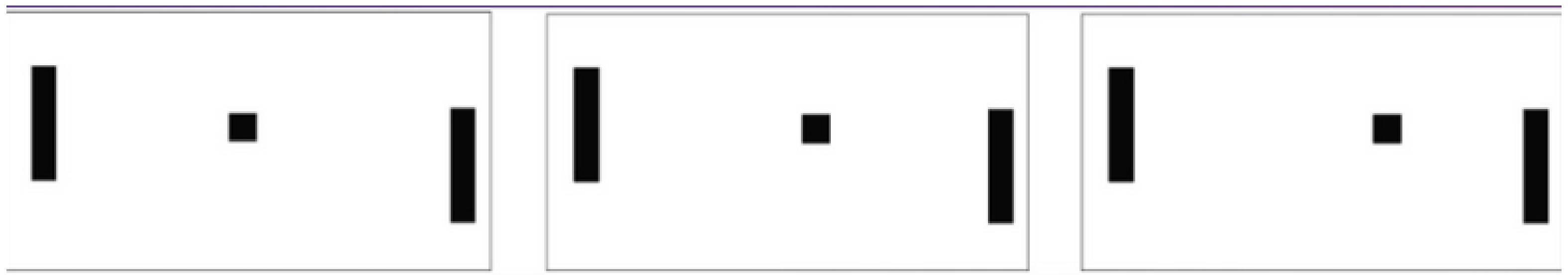




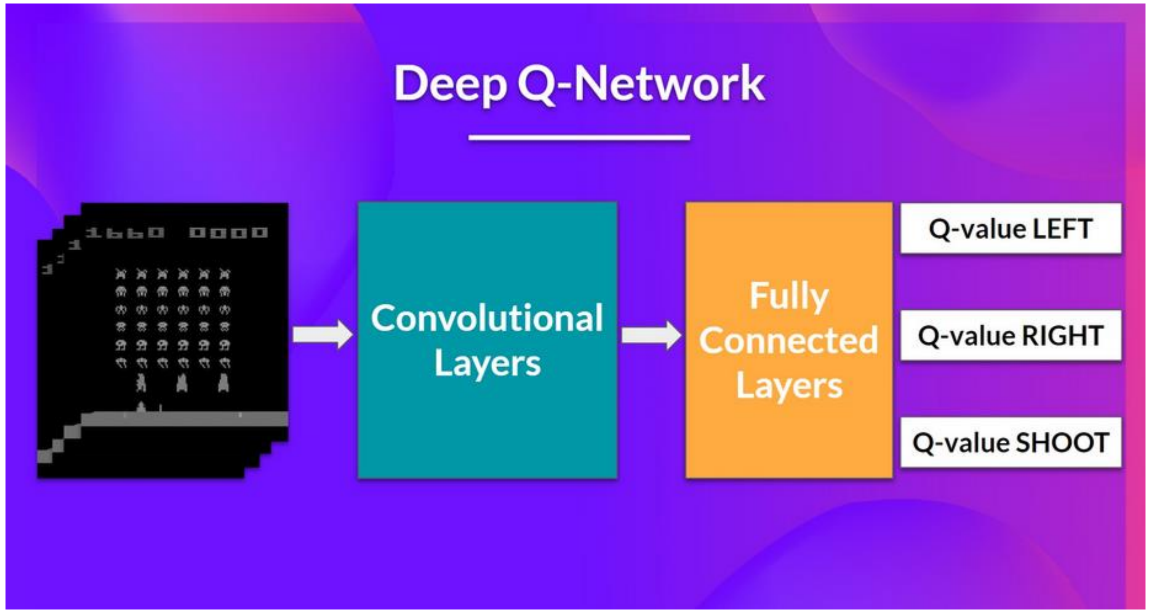
$$\overbrace{\left(r + \gamma \max_{a'} Q(s', a')\right)}^{\text{target}}$$

Limitación temporal





Deep Q Networks



[Submitted on 19 Dec 2013]

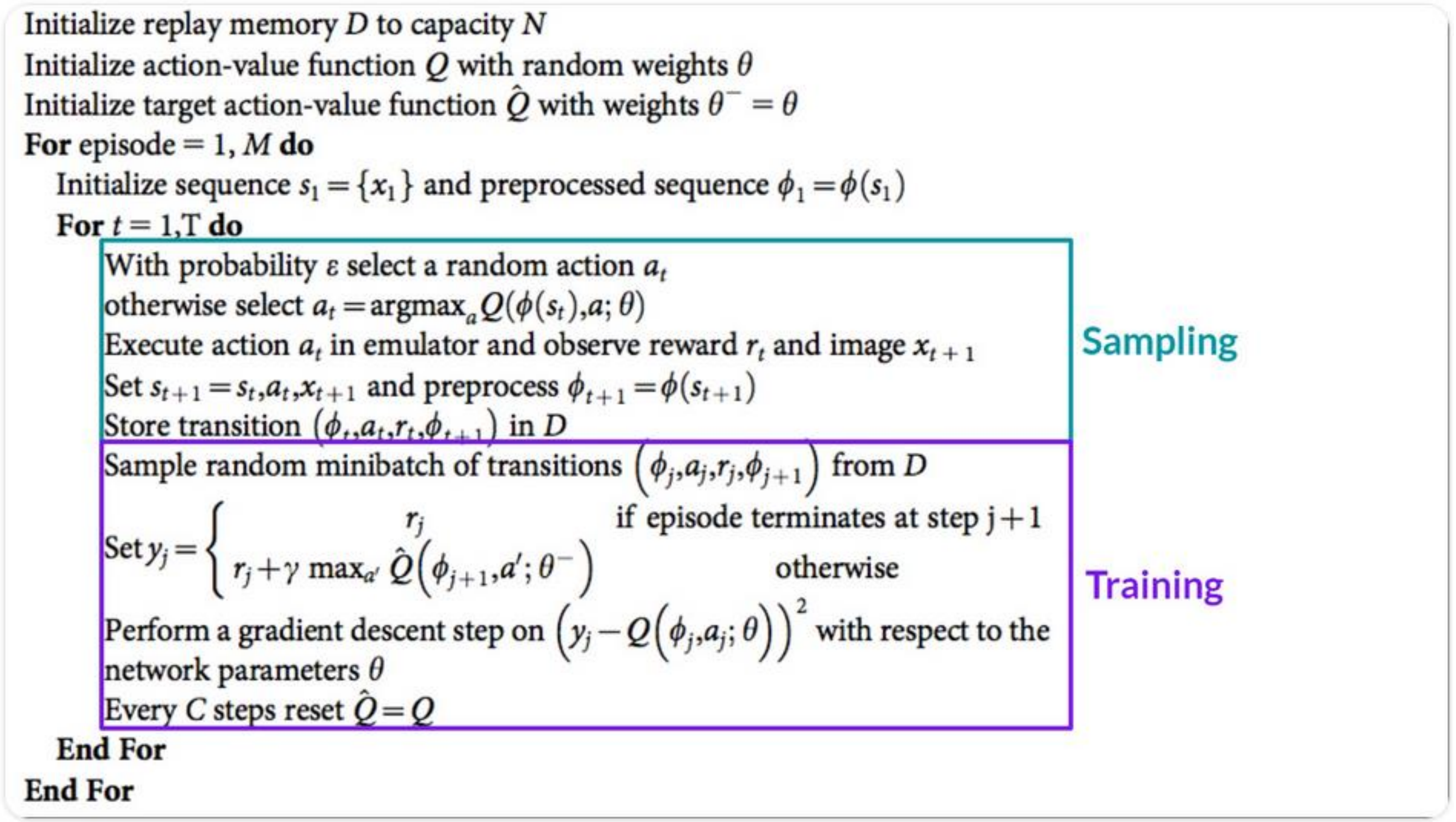
Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

Algoritmo: Deep Q Learning

- **Sampling**: we perform actions and **store the observed experience tuples in a replay memory**.
- **Training**: Select a **small batch of tuples randomly and learn from this batch using a gradient descent update step**.



```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    For $t = 1$,T do
        With probability $\varepsilon$ select a random action $a_t$
        otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the
        network parameters $\theta$
        Every $C$ steps reset $\hat{Q} = Q$
    End For
End For
```

Algoritmo: Deep Q Learning

$$Q(s_t, a_t) = R(s) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

$$f_\theta(\mathbf{x}) \approx y$$

$$(s_t, a_t, R(s_t), s_{t+1})$$

$$\left(s_1^{(1)}, a_1^{(1)}, R(s_1)^{(1)}, s_2^{(1)}\right) \qquad y^{(1)} = R(s_1^{(1)}) + \gamma max_{a_2} Q(s_2^{(1)}, a)$$

$$\left(s_2^{(2)}, a_2^{(2)}, R(s_2)^{(2)}, s_2^{(2)}\right)$$

Algoritmo: Deep Q Learning

Algorithm 1 Deep Q-Learning

```
1: Initialize neural network randomly as a guess for $Q(s, a)$
2: repeat
3:     Take actions in the environment (e.g., lunar lander)
4:     Observe and store the tuple $(s, a, R(s), s')$ in the replay buffer
5:     Store the most recent 10,000 tuples $(s, a, R(s), s')$
6:     Train the neural network:
7:         Sample a batch of 10,000 examples from the replay buffer
8:         For each example, compute the target:
9:         $y = R(s) + \gamma \max_{a'} Q(s', a')$
10:        Train the network to minimize the loss:
11:        $\text{Loss} = (Q_{\text{new}}(s, a) - y)^2$
12:    Update $Q \leftarrow Q_{\text{new}}$
13: until convergence or desired performance
```

Algoritmo: Deep Q Learning

In some state s:

Option 1:

- The agent will always pick the action a that maximizes $Q(s, a)$.
- The agent will never try other actions.

Option 2:

- With probability 0.95, pick the action a that maximizes $Q(s, a)$. (greedy exploitation)
- With probability 0.05, pick an action a randomly. (exploration)

This is known as the ϵ-greedy policy, where $\epsilon = 0.05$:

- The agent acts greedily 95% of the time.
- The agent explores randomly 5% of the time.

To promote exploration early on, start with a high ϵ (e.g., 1.0), and gradually decrease it over time to a lower value (e.g., 0.01).

What happens if $Q(s, \textbf{main})$ is initialized low?

- The agent will never choose actions related to the main thrusters.

Limitaciones de Deep Q Learning

Complexity:

- Can model scenarios where the action space is discrete and small.
- Cannot handle continuous action spaces.

Flexibility:

- Policy is deterministically computed from the Q-function by maximizing the reward.
- Cannot learn stochastic policies.

Limitaciones de Reinforcement Learning

- Much easier to get to work in a simulation than a real robot!
- Far fewer applications than supervised and unsupervised learning.
- But ... exciting research direction with potential for future applications.