# Aprendizaje y Evolución Orientado al Control

## Introducción al aprendizaje por refuerzo

Nicanor Quijano
Juan Martinez-Piazuelo

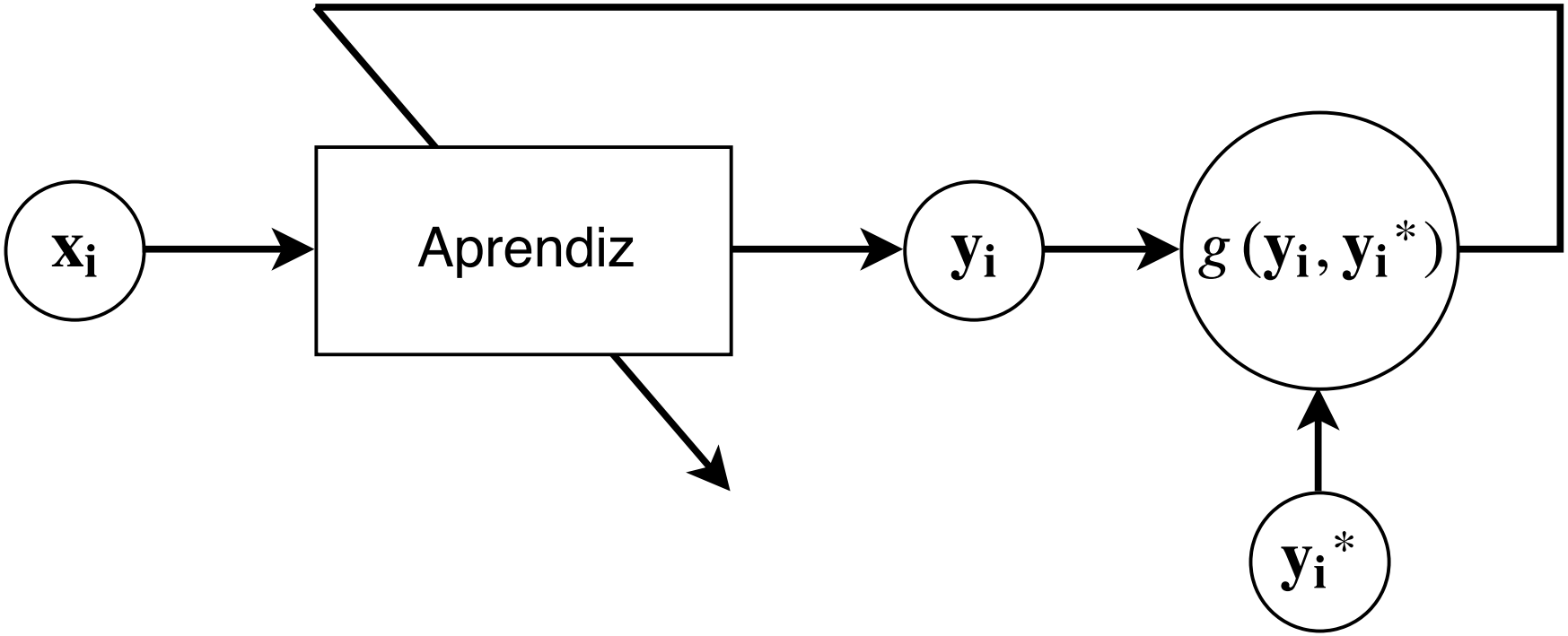
Departamento de Ingeniería Eléctrica y Electrónica
Universidad de los Andes

Semestre 2020-10

# Organización de la clase

1. Aprendizaje por refuerzo vs. Aprendizaje supervisado.
2. Procesos de decisión de Markov.
3. Q-learning tabular.
4. Q-learning con redes neuronales.
5. Q-learning en espacios de acciones continuas.
6. Ejemplos de aplicaciones en control.
7. Taller de programación en Python + Numpy.

# Aprendizaje por refuerzo vs. Aprendizaje supervisado




En aprendizaje supervisado se busca aprender una función desconocida $\mathbf{y}^* = J^*(\mathbf{x})$, a partir de un set de datos $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i^*\}$. El **aprendiz** es una función parámetrizada que busca aproximar a $J^*(\mathbf{x})$ para todo posible $\mathbf{x}$.

# Aprendizaje por refuerzo vs. Aprendizaje supervisado

En aprendizaje supervisado tenemos un **supervisor** que nos da las respuestas correctas.

En aprendizaje por refuerzo tenemos un **crítico** que nos penaliza o recompensa con base en nuestras acciones.

En aprendizaje por refuerzo no conocemos las respuestas correctas, pero podemos medir el desempeño de nuestras decisiones con base en algún criterio. **Se busca aprender a partir de la experiencia**.

**Lectura recomendada:**

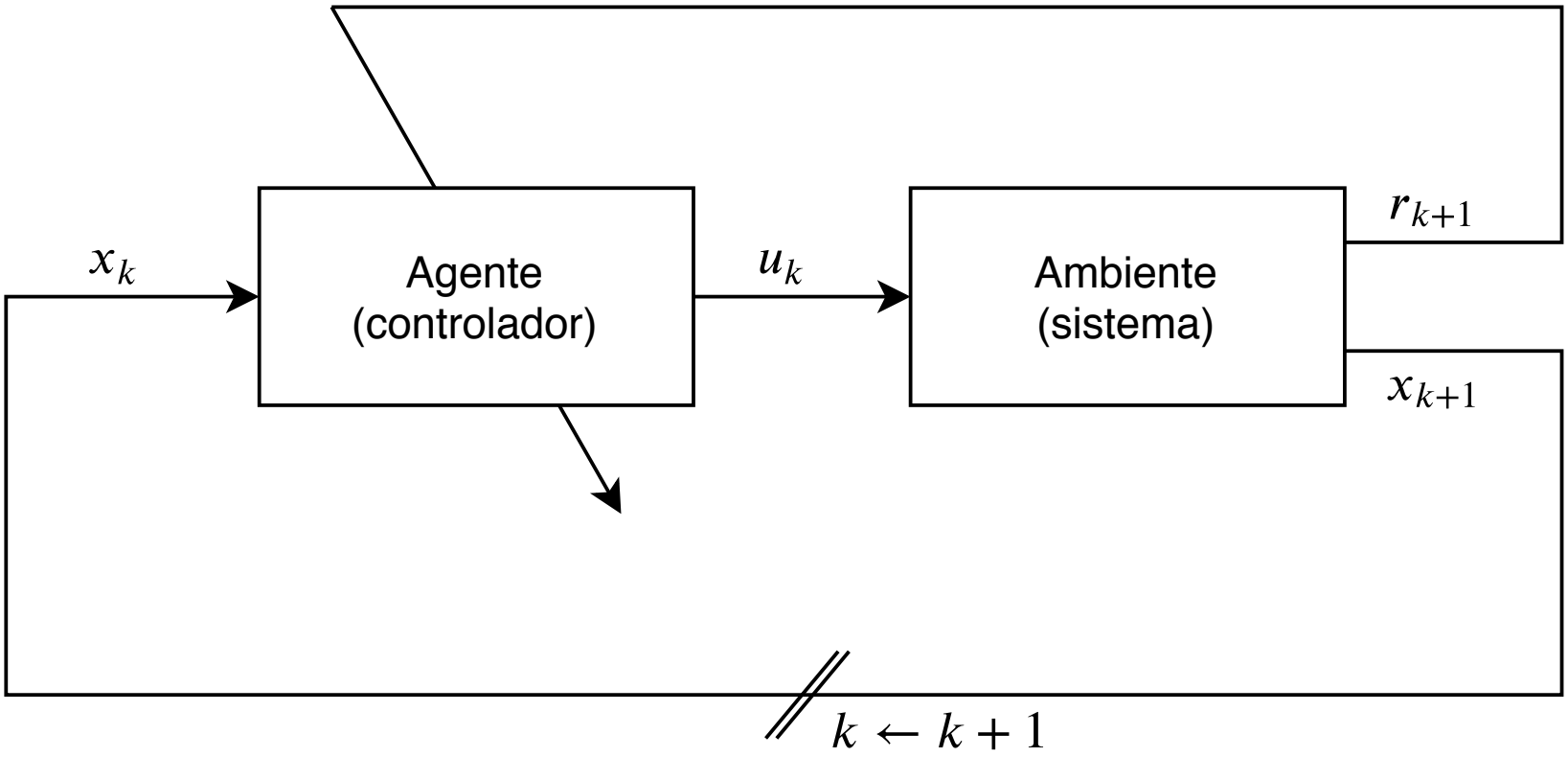
R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Vol. 1, 2nd ed. Cambridge, MA, USA: MIT Press, 2018

# Aprendizaje por refuerzo vs. Aprendizaje supervisado

Imagen tomada de: www.pexels.com/search/dog/

# Aprendizaje por refuerzo y control

**Aprendizaje por refuerzo es un método de control basado en datos (model-free):**




**Al ser model-free tenemos el dilema de exploración/explotación**

# Procesos de decisión de Markov

Un proceso de decisión de Markov (MDP) está compuesto por:

- Un set de estados $\mathcal{X}$.
- Un set de acciones $\mathcal{U}$.
- Una distribución de transición de estado

$$p\left(x_{k+1}\middle|x_k, u_k\right) \doteq \mathsf{Pr}\left(x_{k+1}\middle|x_k, u_k\right),$$

donde $x_{k+1} \in \mathcal{X}$, $x_k \in \mathcal{X}$, $u_k \in \mathcal{U}$, y $p : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$. Por ende:

$$x_{k+1} \sim p\left(x_{k+1}\middle|x_k, u_k\right),$$

- Una señal de recompensa: $r_{k+1} \in \mathbb{R}$ es la recompensa recibida en el estado $x_{k+1}$.

# Procesos de decisión de Markov

Un **episodio** es una secuencia finita de estados, acciones y recompensas[1]:

$$x_0, u_0, r_1, x_1, u_1, r_2, x_2, u_2, \ldots, r_{T-1}, x_{T-1}, u_{T-1}, r_T, x_T$$

donde

$$x_{k+1} \sim p(x_{k+1}|x_k, u_k)$$
$$u_k \sim \pi(u_k|x_k),$$

con $\pi : \mathcal{U} \times \mathcal{X} \to [0, 1]$, de forma que:

$$\pi(u_k|x_k) \doteq \Pr(u_k|x_k).$$

$\pi(u_k|x_k)$ **se denomina la política de control.**

[1] Un episodio termina cuando se completa un numero de pasos $T$, o cuando se alcanza un estado terminal.

# Procesos de decisión de Markov

El **retorno** a partir de un tiempo $k$ hasta un horizonte $T$ está dado por:

$$G(k,T) = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \cdots + \gamma^{T-k-1} r_T,$$

donde $\gamma \in [0,1)$ es un factor de descuento (pesa la importancia de la recompensa en el tiempo).

El retorno puede ser escrito de forma compacta como:

$$G(k,T) = \sum_{i=k}^{T-1} \gamma^{i-k} r_{i+1}.$$

Adicionalmente, note que:

$$G(k,T) = r_{k+1} + \gamma G(k+1,T).$$

# Objetivo de aprendizaje

El objetivo es entonces **aprender** un política de control $\pi^*(u_k|x_k)$, tal que se maximice el valor esperado del retorno:

$$\pi^*(u_k|x_k) = \arg\max_{\pi} \mathbb{E}_{\pi}\left[G(k,T)|x_k\right], \quad \forall x_k \in \mathcal{X}$$

donde $\mathbb{E}_{\pi}[\cdot|x_k]$ denota el valor esperado tomado sobre las distribuciones de las dinámicas del sistema y de la política

**El objetivo es aprender una política de control que en cada estado $x_k \in \mathcal{X}$ tome la acción $u_k \in \mathcal{U}$ que lleve al mayor retorno esperado.**

**Dado que se asume que las dinámicas del sistema son desconocidas, la maximización sobre $\mathbb{E}_{\pi}\left[\cdot|x_k\right]$ no se puede evaluar directamente!**

# Objetivo de aprendizaje

El valor de un par estado-acción $(x_k, u_k)$ bajo una política de control $\pi(u_k|x_k)$ se puede definir como:

$$q_\pi(x_k, u_k) = \mathbb{E}_\pi \left[ G(k, T) | x_k, u_k \right]$$

El objetivo es entonces aprender la **función de valor** $q_*(x_k, u_k)$ tal que para todo par $(x_k, u_k)$ se tenga que:

$$q_*(x_k, u_k) = \max_\pi q_\pi(x_k, u_k).$$

Note que conociendo $q_*(x_k, u_k)$ para todo $(x_k, u_k)$ tendríamos $\pi^*(u_k|x_k)$:

$$u_k = \arg\max_u q_*(x_k, u).$$

Este tipo de política se denomina *greedy*.

# Q-learning

**¿Cómo aprendemos $q_*(x_k, u_k)$ para todo $(x_k, u_k)$?**

**Q-learning** es un algoritmo iterativo que nos permite aprender $q_*(x_k, u_k)$ para todo posible par estado-acción: $(x_k, u_k)$.

- Q-learning tabular
- Q-learning con redes neuronales

Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

# Q-learning tabular

Suponiendo que $|\mathcal{X}|$ y $|\mathcal{U}|$ son cantidades finitas, podemos definir una tabla con $|\mathcal{X}|$ filas y $|\mathcal{U}|$ columnas. Decimos que $Q(x_k, u_k)$ es nuestro estimado de $q_*(x_k, u_k)$.

$Q(x_k, u_k)$ es el elemento de la fila $x_k$ y la columna $u_k$ de la tabla.

**¿Cómo actualizamos $Q(x_k, u_k)$ para aproximarse a $q_*(x_k, u_k)$?**

# Q-learning tabular

Note que las funciones de valor satisfacen la ecuación recursiva de Bellman:

$$q_\pi(x_k, u_k) = r_{k+1} + \gamma q_\pi(x_{k+1}, u_{k+1}).$$

donde $u_{k+1} \sim \pi(u_{k+1}|x_{k+1})$.

En particular, para $q_*(x_k, u_k)$ se tiene que:

$$q_*(x_k, u_k) = r_{k+1} + \gamma \max_u q_*(x_k, u)$$

Por ende, en Q-learning tabular aplicamos actualizaciones de la forma:

$$Q(x_k, u_k) = Q(x_k, u_k) + \alpha \left[ r_{k+1} + \gamma \max_u Q(x_{k+1}, u) - Q(x_k, u_k) \right]$$

La política **greedy** se puede derivar de $Q(x_k, u_k)$:

$$u_k = \arg\max_u Q(x_k, u)$$

# Q-learning tabular

1. Inicializamos la tabla $Q$ de forma arbitraria[2].
2. Inicializamos el estado del MDP y definimos $k = 0$.
3. Escogemos una acción $u_k$ usando una **política de exploración** derivada de $Q(x_k, u_k)$, e.g., $\epsilon$-*greedy*.
4. Ejecutamos $u_k$ y observamos $r_{k+1}$, $x_{k+1}$.
5. Actualizamos $Q(x_k, u_k)$ aplicando:

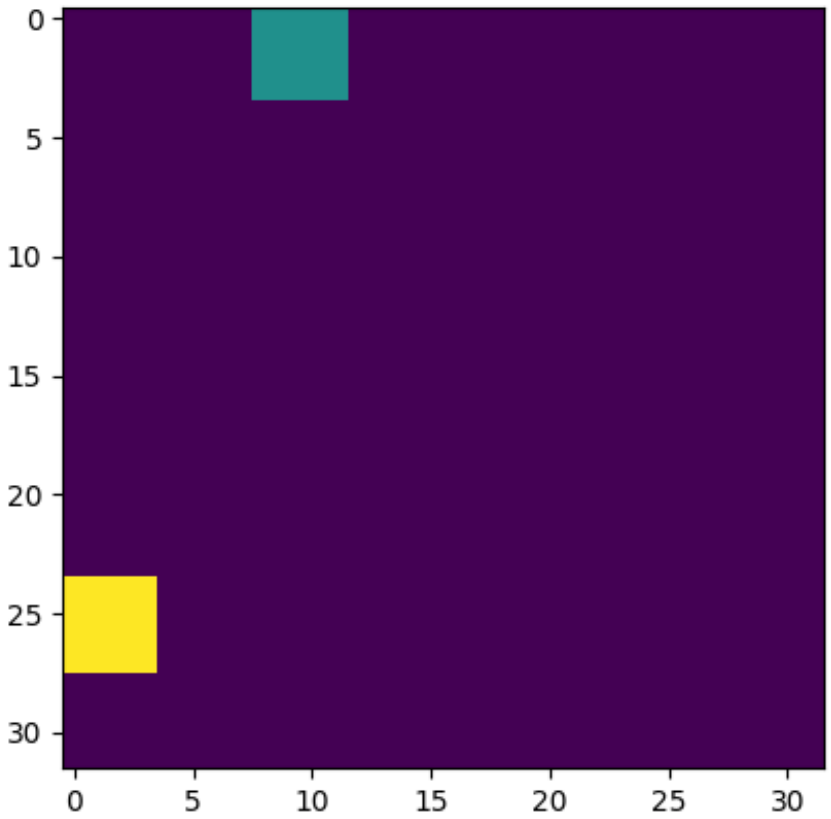
$$Q(x_k, u_k) = Q(x_k, u_k) + \alpha \left[ r_{k+1} + \gamma \max_u Q(x_{k+1}, u) - Q(x_k, u_k) \right]$$

6. Si $k < T$ y $x_{k+1}$ no es un **estado terminal**, actualizamos $k = k + 1$ y regresamos al paso (3). De lo contrario continuamos.
7. Si no se ha cumplido el criterio de parada regresamos al paso (2). De lo contrario terminamos.

[2] Si se tienen estados terminales, el valor $Q$ para esos estados se inicializa en $0$.

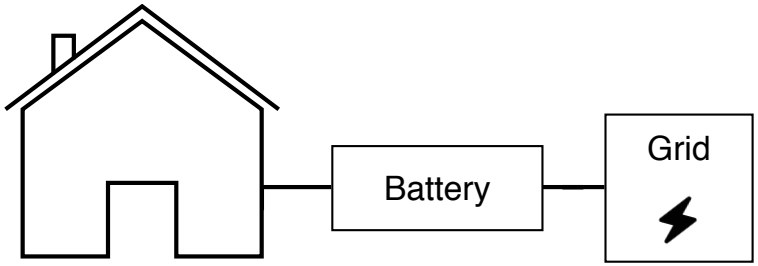
# Q-learning: ejemplos

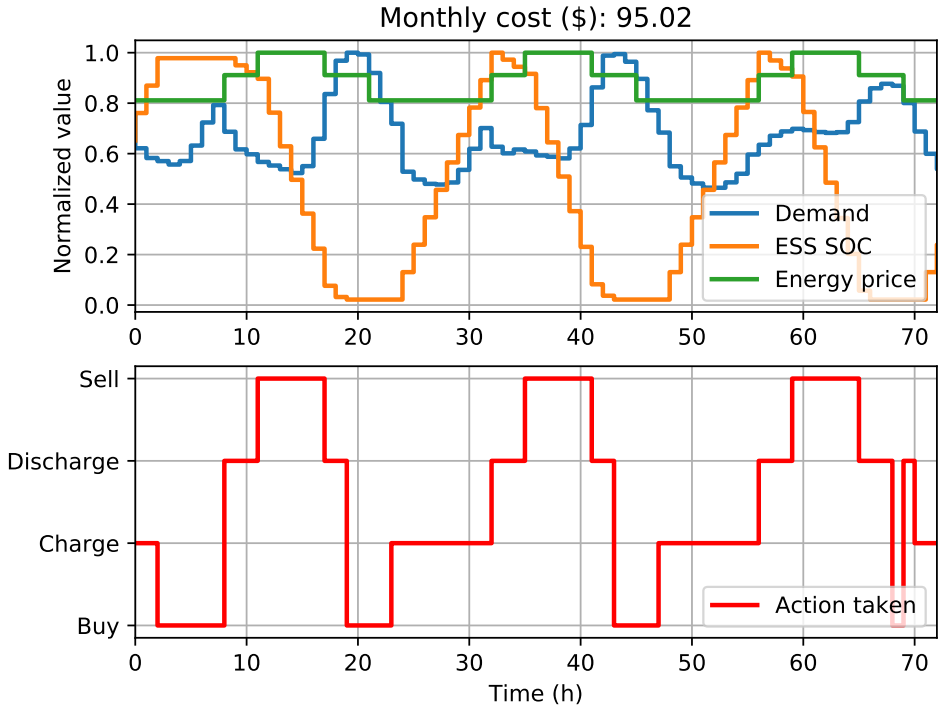
## Ejemplo de control con acciones discretas:



# Q-learning: ejemplos

## Ejemplo de control con acciones discretas:

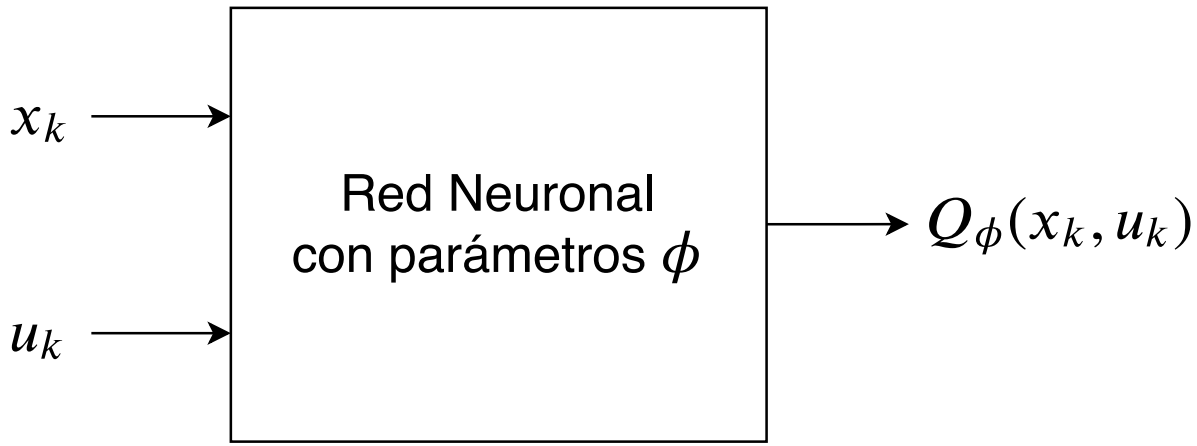


# Q-learning con redes neuronales

En sistemas con estados continuos no podemos implementar $Q(x_k, u_k)$ como una tabla (tendría infinitas filas!).

**Para estos casos implementamos $Q_\phi(x_k, u_k)$ como una red neuronal!**




**¿Cómo entrenamos la red?**

# Q-learning con redes neuronales

La función de costo (loss) para el entrenamiento de la red neuronal la tomamos como el error de Bellman al cuadrado (SBE):

$$SBE = \frac{1}{2}\left(r_{k+1} + \gamma \max_u Q_\phi(x_{k+1}, u) - Q_\phi(x_k, u_k)\right)^2$$

En la práctica se obtienen mejores resultados usando 2 redes neuronales:

$$SBE = \frac{1}{2}\left(r_{k+1} + \gamma \max_u Q_{\hat{\phi}}(x_{k+1}, u) - Q_\phi(x_k, u_k)\right)^2,$$

y los parámetros de $Q_{\hat{\phi}}(x_k, u_k)$ se copian lentamente desde $Q(x_k, u_k)$:

$$\hat{\phi} \leftarrow \tau\phi + (1-\tau)\hat{\phi}$$

con $\tau \in (0, 1]$ (típicamente $\tau << 1$).

# Q-learning con redes neuronales

1. Inicializamos la red $Q_\phi(\cdot,\cdot)$ de forma arbitraria. Inicializamos $Q_{\hat{\phi}}(\cdot,\cdot) = Q_\phi(\cdot,\cdot)$.
2. Inicializamos el estado del MDP y definimos $k = 0$.
3. Escogemos una acción $u_k$ usando una **política de exploración** derivada de $Q_\phi(x_k, u_k)$, e.g., $\epsilon$-*greedy*.
4. Ejecutamos $u_k$, observamos $r_{k+1}$, $x_{k+1}$, y almacenamos $(x_k, u_k, x_{k+1}, r_{k+}$ en memoria[3].
5. Muestreamos un *batch* aleatorio de la memoria de experiencias.
6. Actualizamos los parámetros de $Q_\phi(\cdot,\cdot)$ para minimizar el error cuadrático de Bellman (SBE) en el *batch* de experiencias.
7. Actualizamos $\hat{\phi}$ usando *soft-updates*.
8. Si $k < T$ y $x_{k+1}$ no es un **estado terminal**, actualizamos $k = k + 1$ y regresamos al paso (3). De lo contrario continuamos.
9. Si no se ha cumplido el criterio de parada regresamos al paso (2). De lo contrario terminamos.

[3] $t_{k+1}$ es una bandera booleana que indica si $x_{k+1}$ es un estado terminal.

# Q-learning con redes neuronales

El error cuadrático de Bellman (SBE) lo definimos así:

$$SBE = \frac{1}{2}\left(r_{k+1} + \gamma \max_u Q_{\hat{\phi}}(x_{k+1}, u) - Q_\phi(x_k, u_k)\right)^2,$$

si $x_{k+1}$ no es un estado terminal. Y lo definimos así:

$$SBE = \frac{1}{2}\left(r_{k+1} - Q_\phi(x_k, u_k)\right)^2,$$

si $x_{k+1}$ es un estado terminal.
En cualquier caso, los parámetros de $Q_{\hat{\phi}}(x_k, u_k)$ se copian lentamente desde $Q(x_k, u_k)$:

$$\hat{\phi} \leftarrow \tau\phi + (1-\tau)\hat{\phi}$$

con $\tau \in (0, 1]$ (típicamente $\tau << 1$).

# Q-learning con redes neuronales

Observe que la mecánica de entrenamiento de la red $Q_\phi(\cdot,\cdot)$ es igual a la seguida en **aprendizaje supervisado**.

En este caso la función de *loss*, que habíamos denotado $g(y, y^*)$, está dada por el SBE:

$$SBE = \frac{1}{2}\left(y^* - y\right)^2,$$

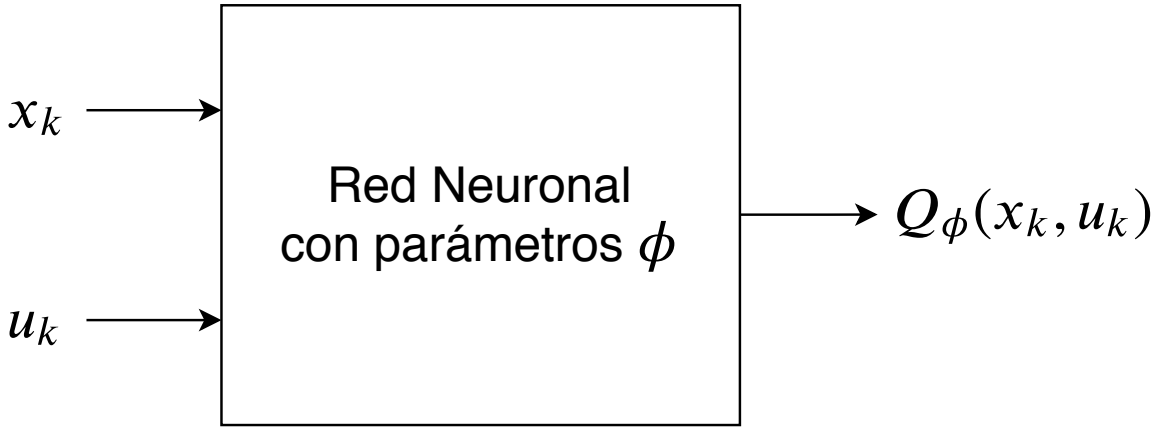
donde:

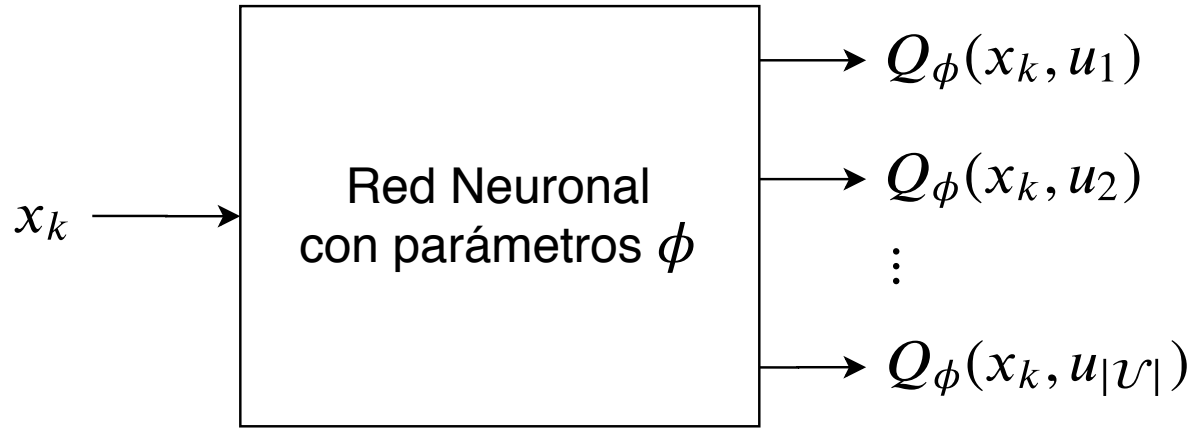
$$\begin{aligned} y &= Q_\phi(x_k, u_k) \\ y^* &= r_{k+1} + \gamma \max_u Q_{\hat{\phi}}(x_{k+1}, u), \quad \text{si } x_{k+1} \text{ no es terminal} \\ y^* &= r_{k+1}, \quad \text{si } x_{k+1} \text{ es terminal} \end{aligned}$$

Finalmente: $\phi \leftarrow \phi - \alpha \nabla_\phi SBE$

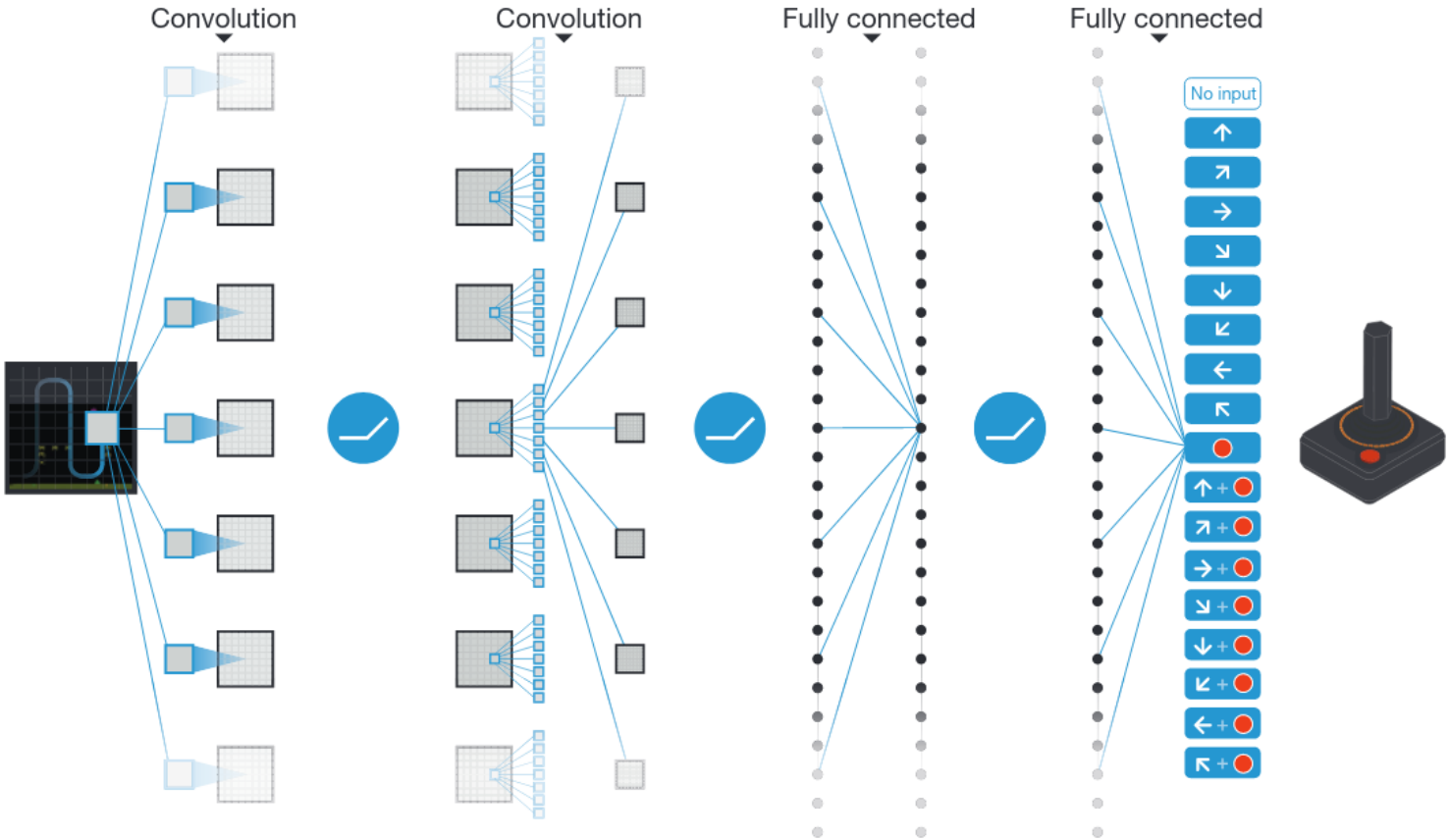
# Q-learning con redes neuronales

**Un truco para eficiencia computacional:**



# Deep Q-learning




Ver: Deep Q-learning for Multi-Agent Cooperation

---

Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

# Q-learning para acciones continuas

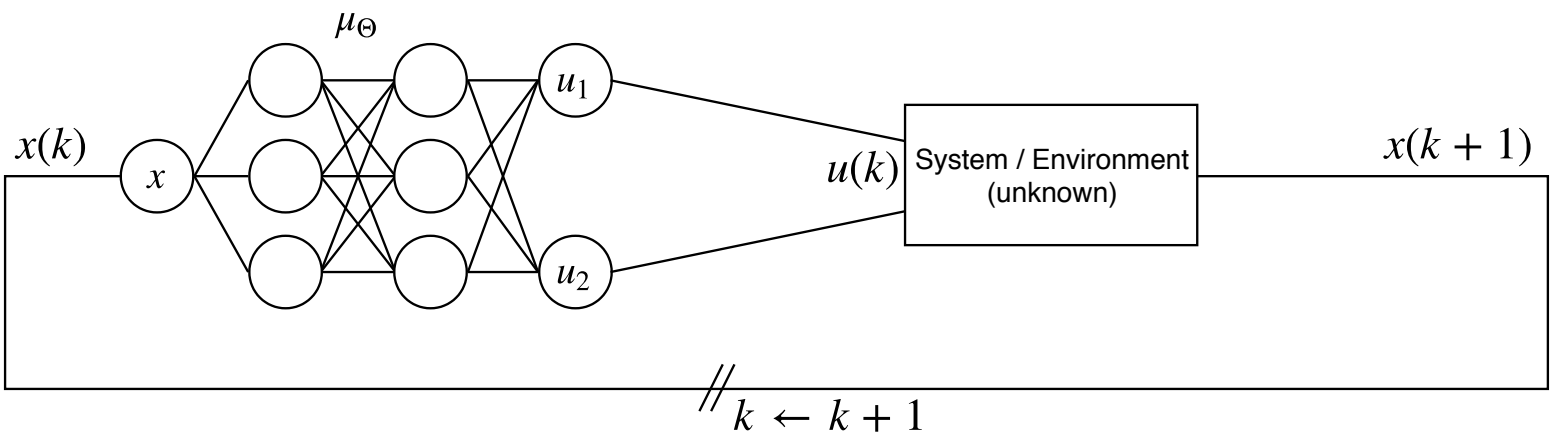
**¿Cómo podemos extender Q-learning para problemas con espacios de acciones continuas?**

Andrea Bonarini, Francesco Montrone, and Marcello Restelli. "Reinforcement Distribution in Continuous State Action Space Fuzzy Q–Learning: A Novel Approach". In: *Fuzzy Logic and Applications*. Ed. by Isabelle Bloch, Alfredo Petrosino, and Andrea G. B. Tettamanzi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 40–45

Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)* (2016)

# Q-learning para acciones continuas

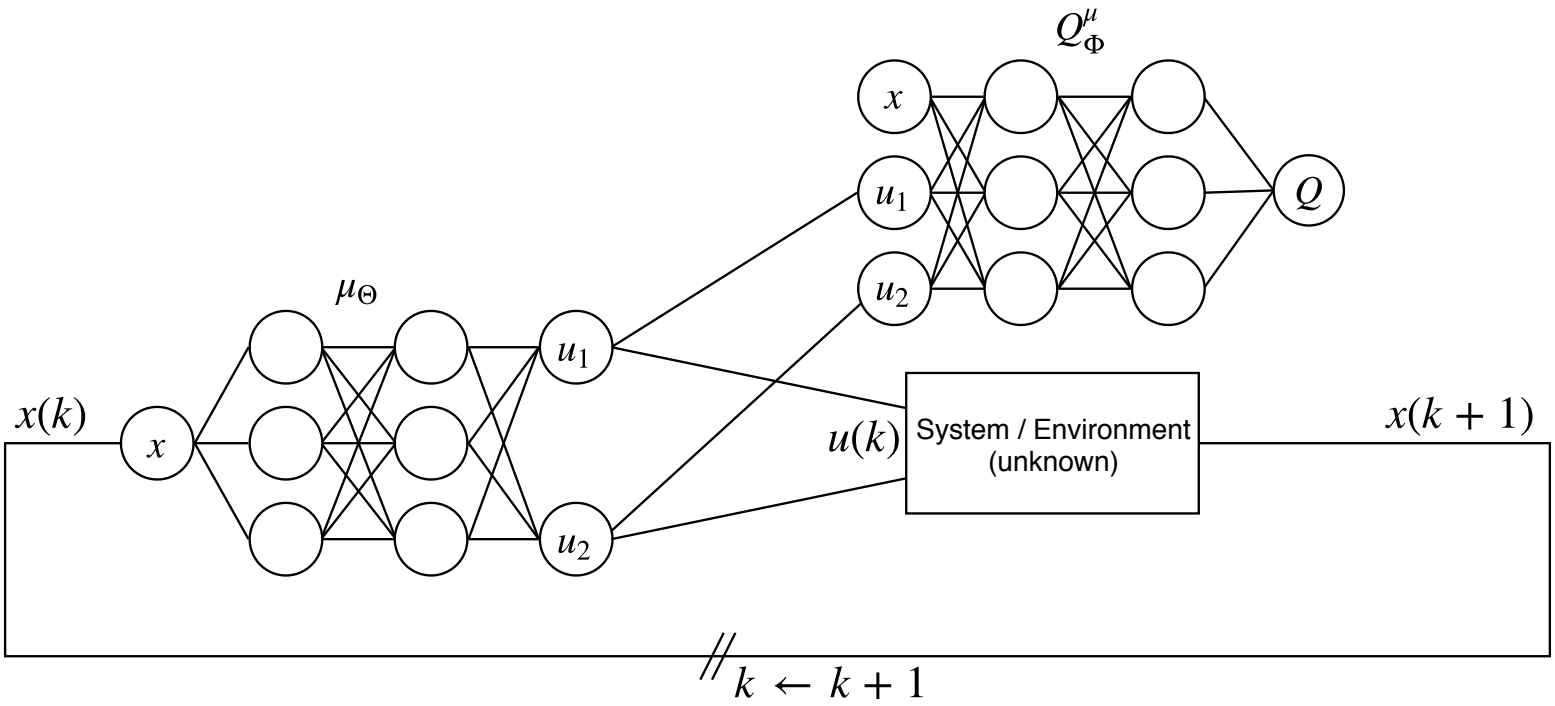
## The DDPG approach:




Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)* (2016).

# Q-learning para acciones continuas

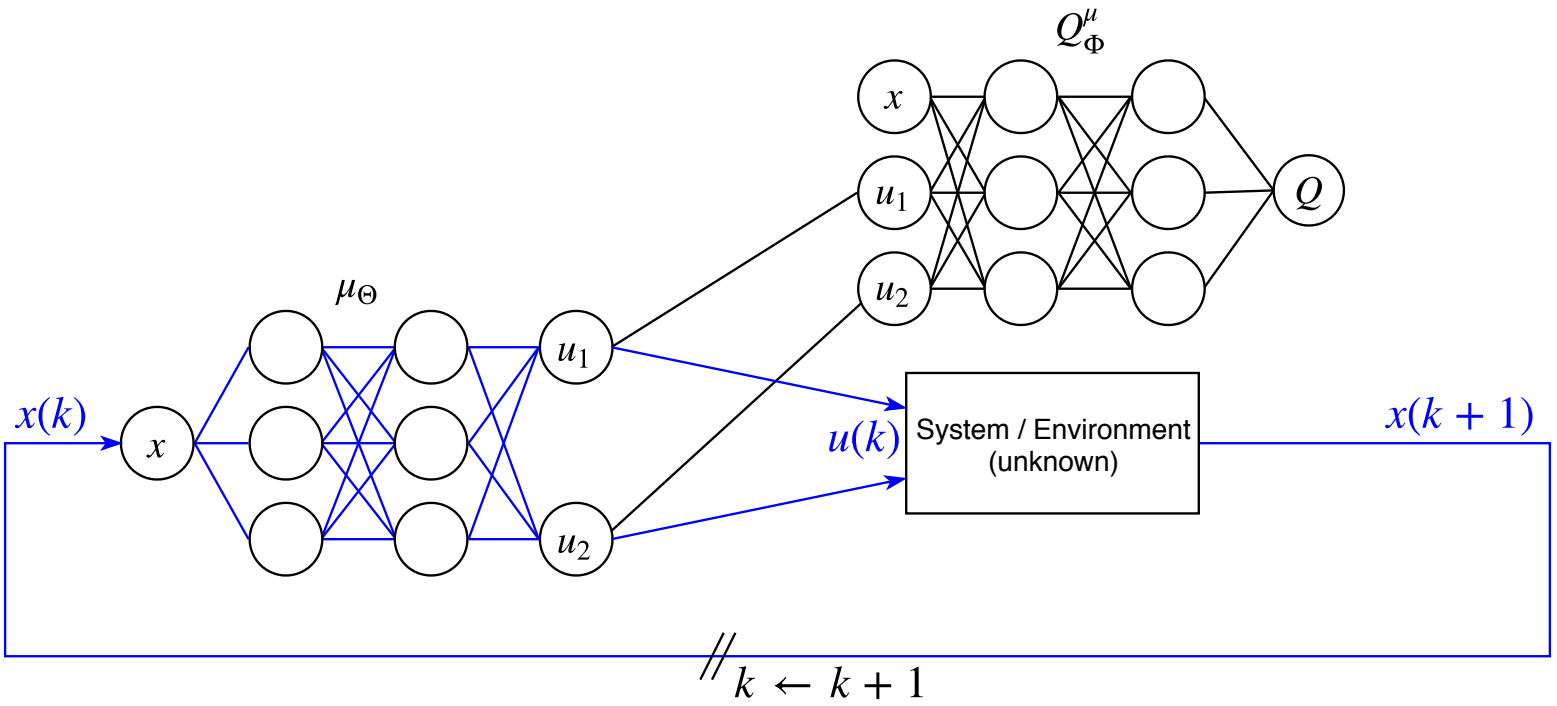
## The DDPG approach:




Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)* (2016).

# Q-learning para acciones continuas

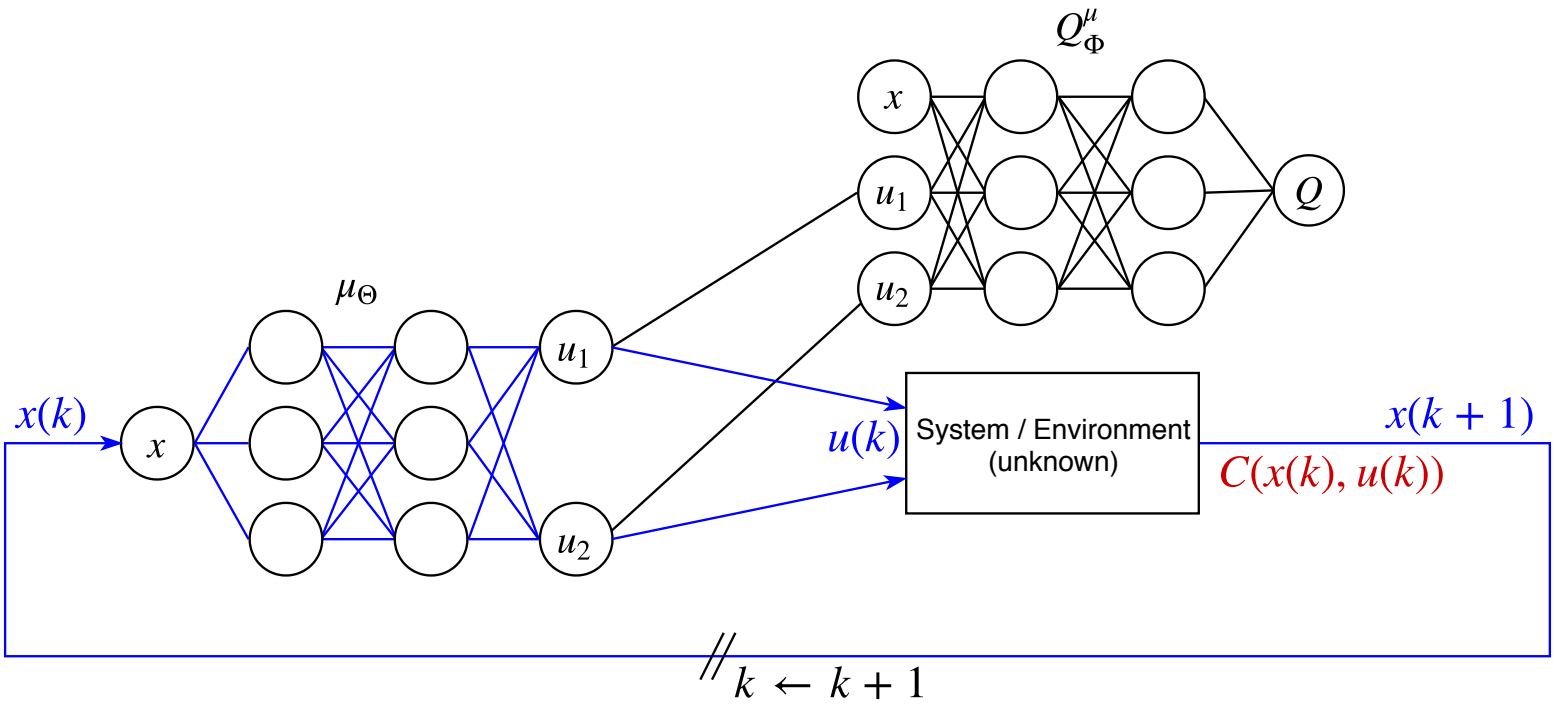
## The DDPG approach:




Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)* (2016).

# Q-learning para acciones continuas

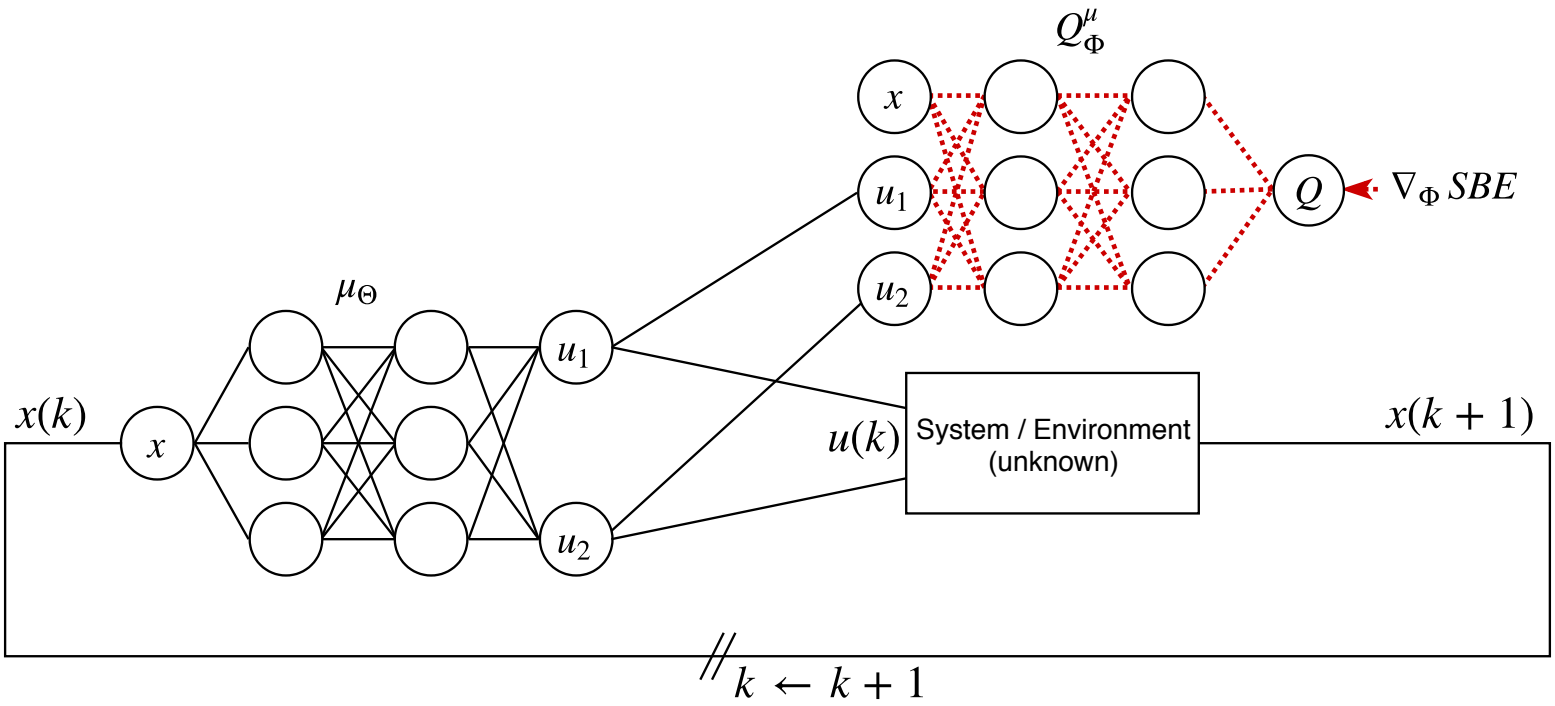
## The DDPG approach:




Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)* (2016).

# Q-learning para acciones continuas

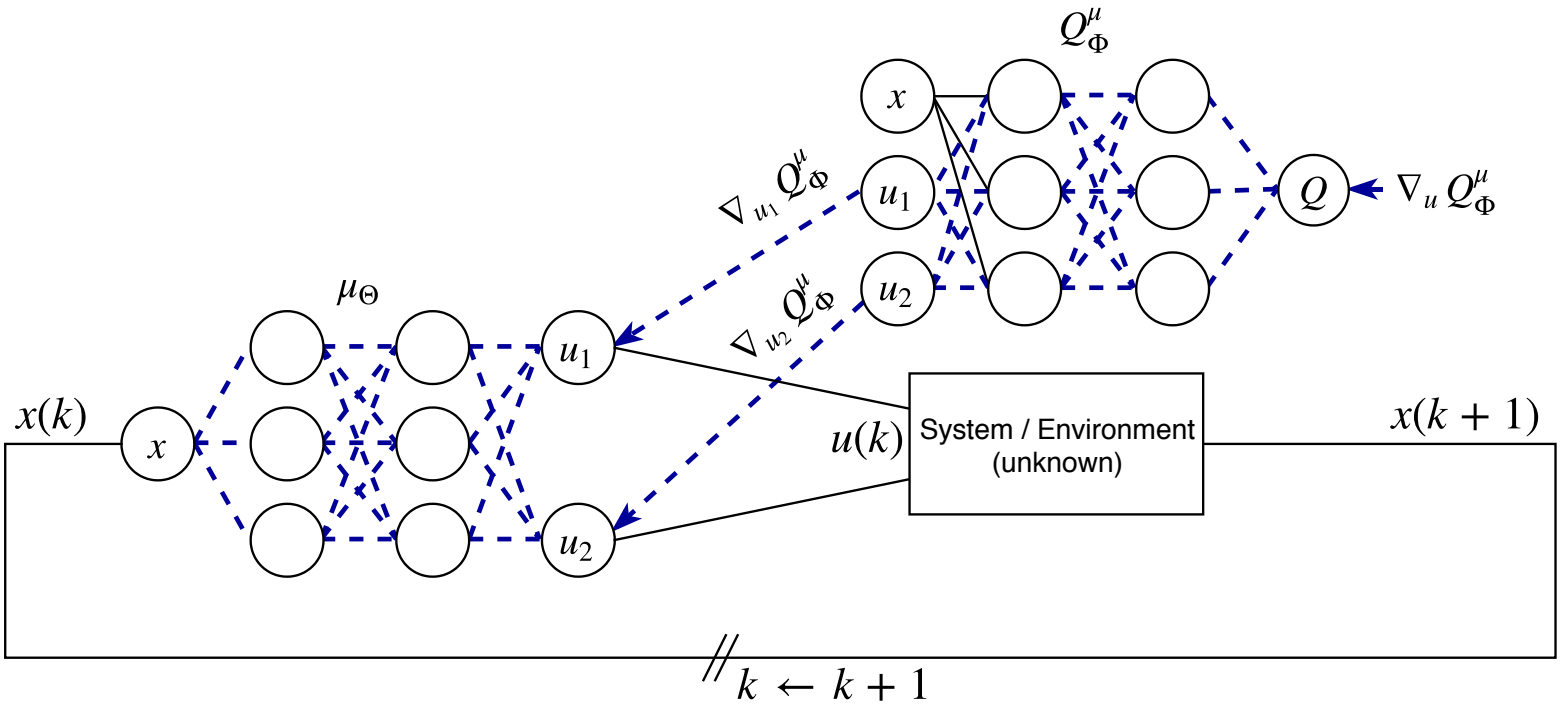
## The DDPG approach:




Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)* (2016).

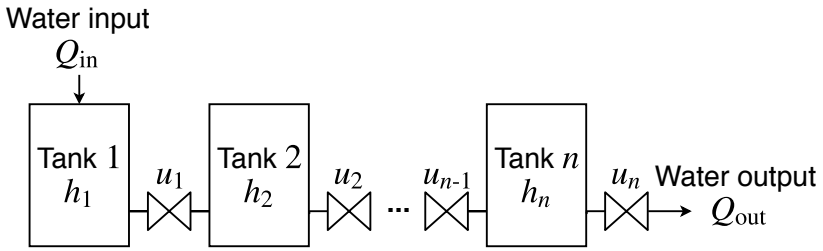
# Q-learning para acciones continuas

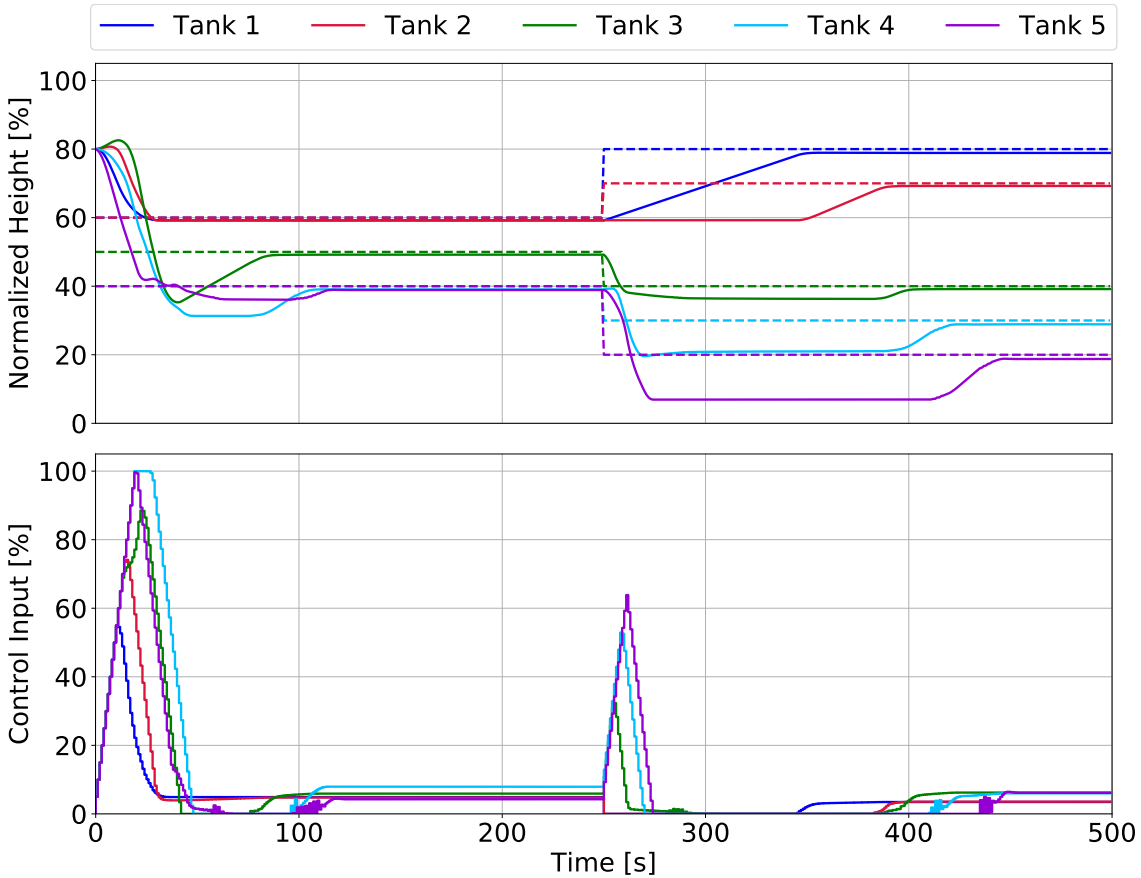
## The DDPG approach:




Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *International Conference on Learning Representations (ICLR)* (2016).

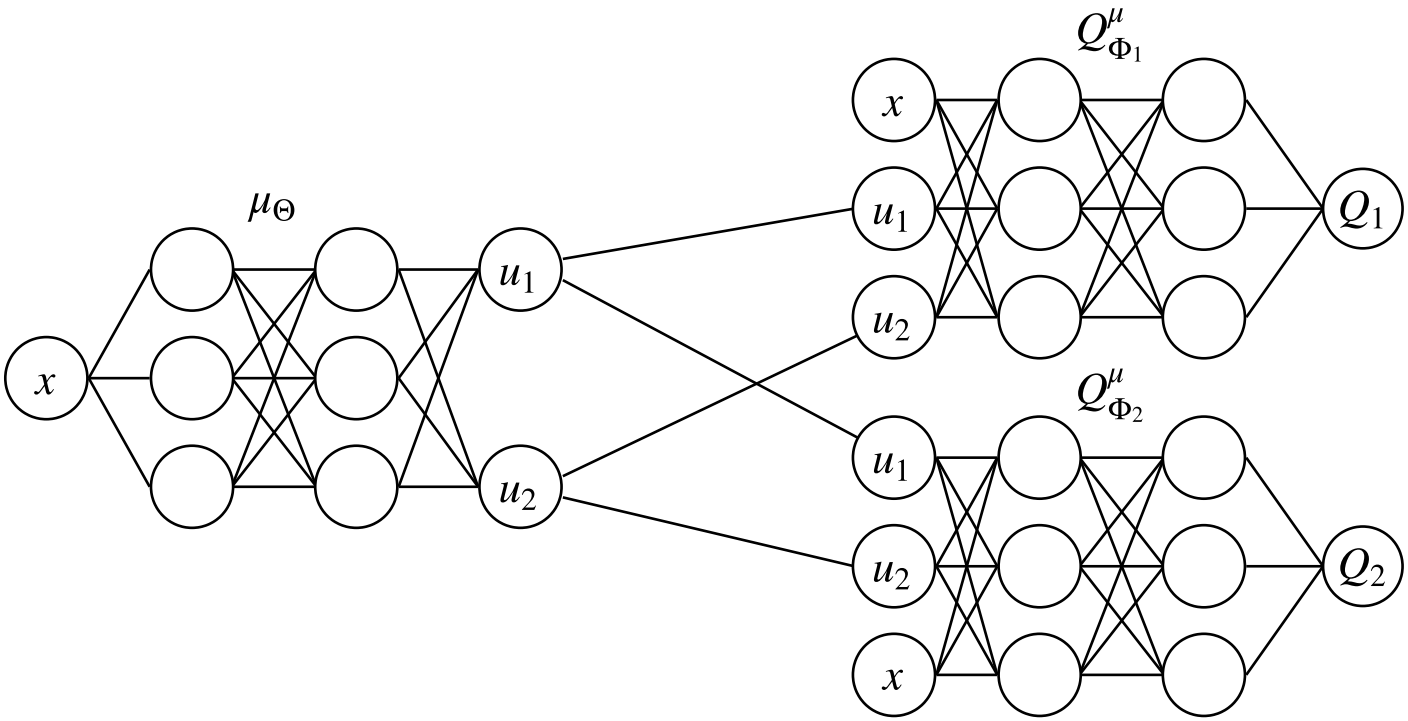
# RL en sistemas de tanques acoplados



# RL en sistemas de tanques acoplados

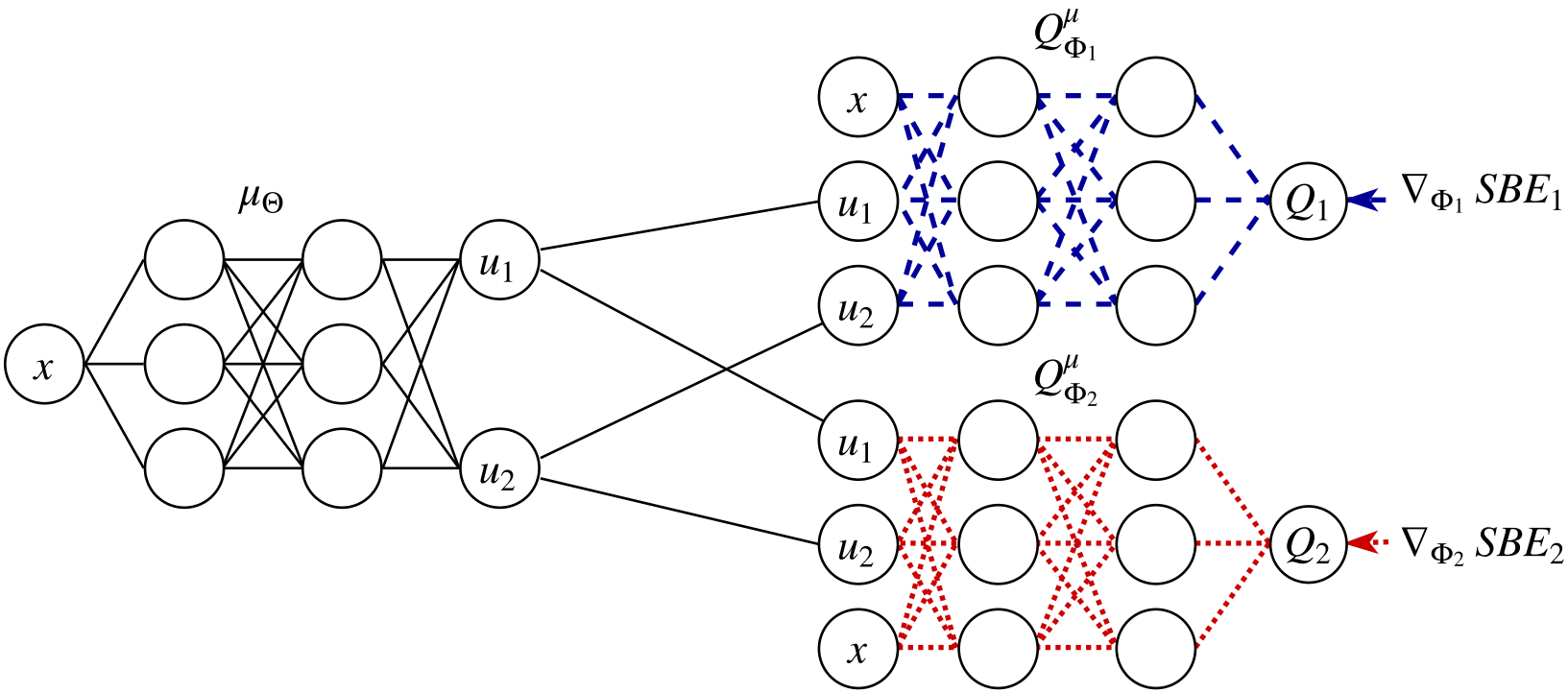
## Arquitectura multi-crítico:




J. Martinez-Piazuelo et al. "A Multi-Critic Reinforcement Learning Method: An Application to Multi-Tank Water Systems". In: *IEEE Access* 8 (2020), pp. 173227–173238. DOI: 10.1109/ACCESS.2020.3025194.

# RL en sistemas de tanques acoplados

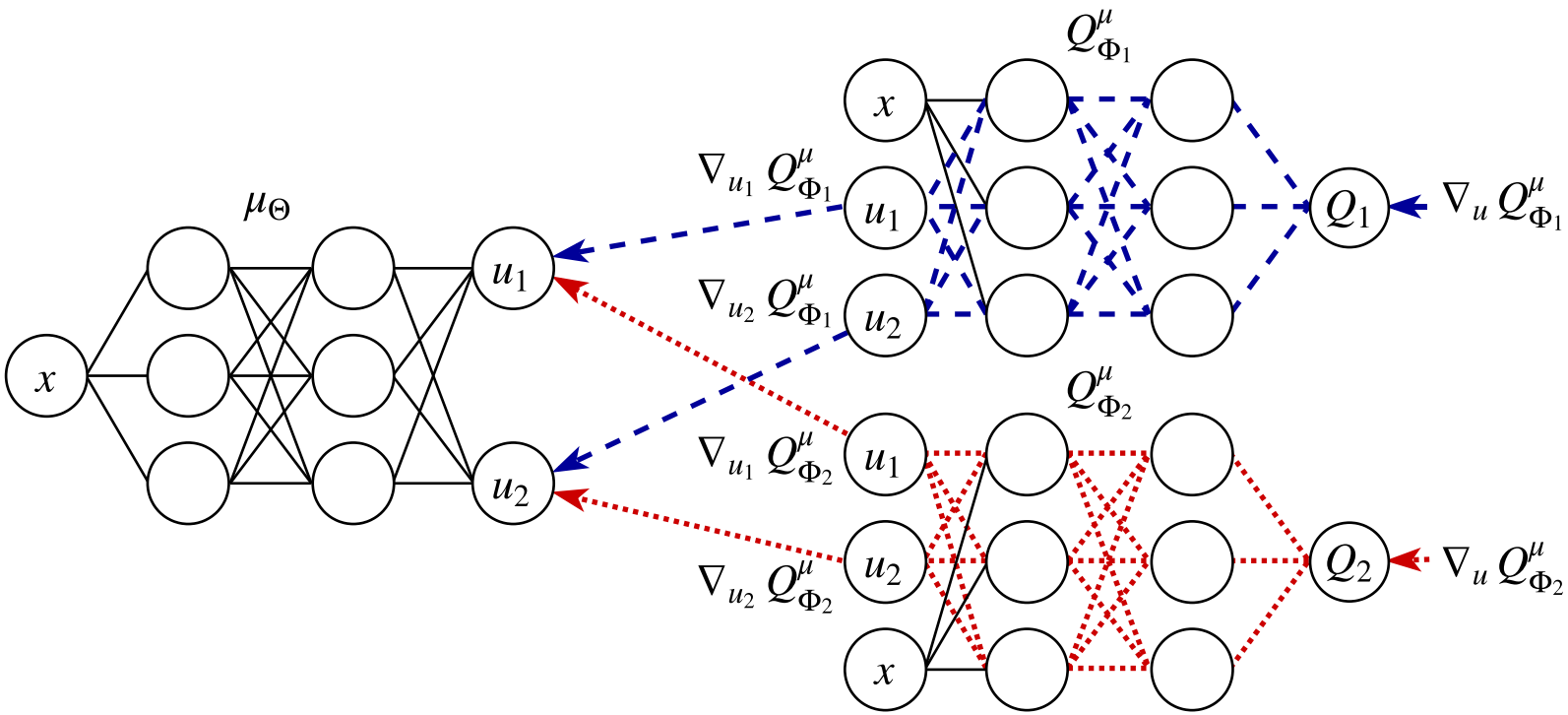
## Arquitectura multi-crítico:




J. Martinez-Piazuelo et al. "A Multi-Critic Reinforcement Learning Method: An Application to Multi-Tank Water Systems". In: *IEEE Access* 8 (2020), pp. 173227–173238. DOI: 10.1109/ACCESS.2020.3025194.

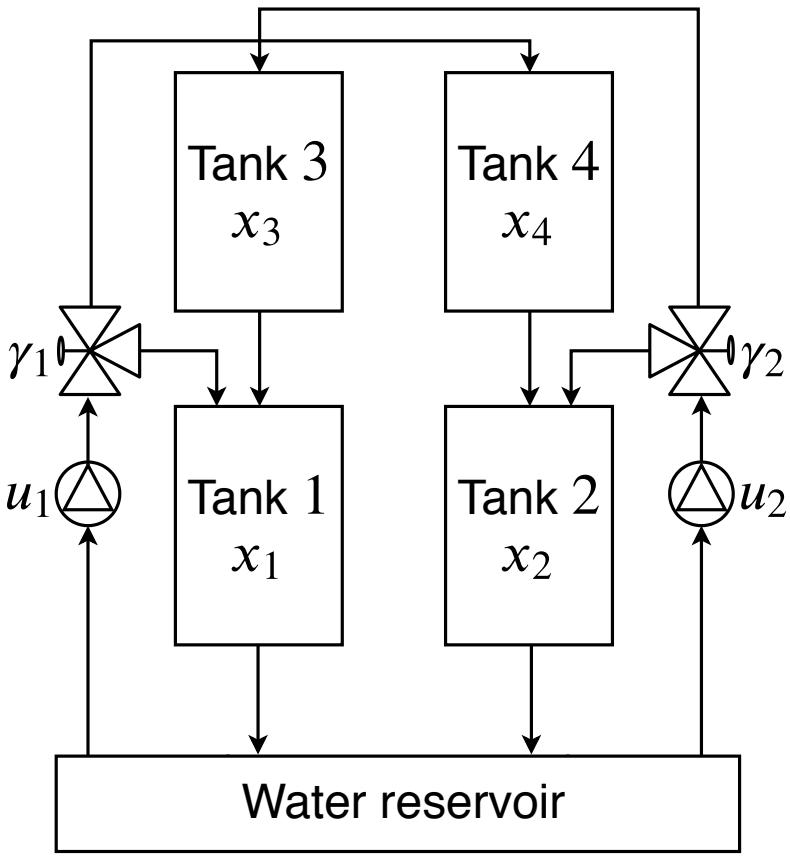
# RL en sistemas de tanques de agua acoplados

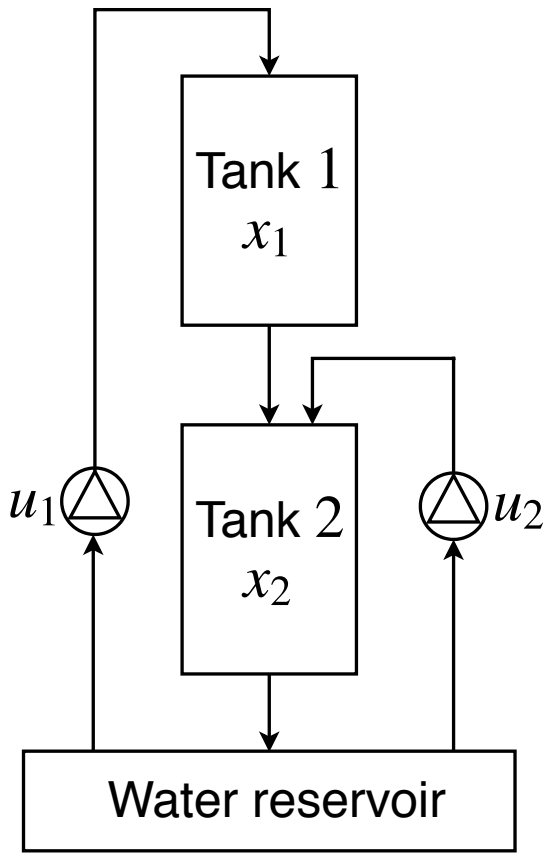
## Arquitectura multi-crítico:




J. Martinez-Piazuelo et al. "A Multi-Critic Reinforcement Learning Method: An Application to Multi-Tank Water Systems". In: *IEEE Access* 8 (2020), pp. 173227–173238. DOI: 10.1109/ACCESS.2020.3025194.

# RL en sistemas de tanques acoplados

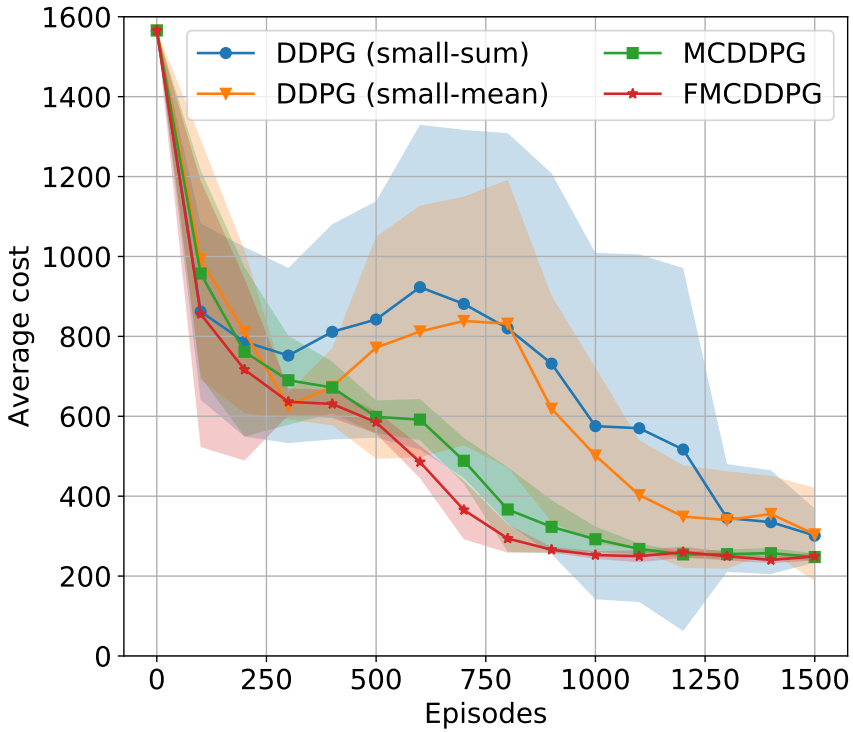


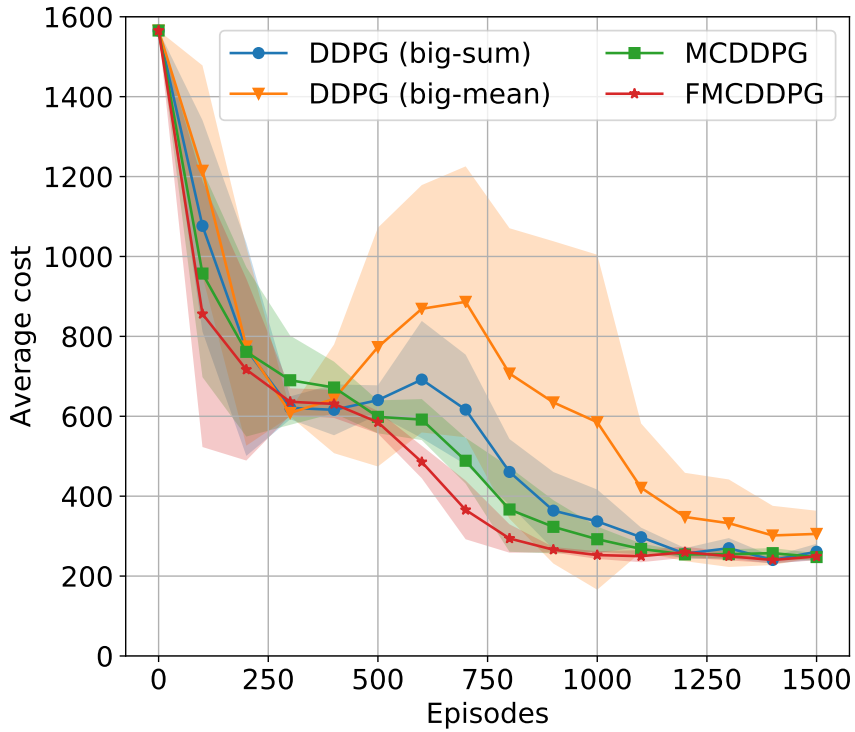




J. Martinez-Piazuelo et al. "A Multi-Critic Reinforcement Learning Method: An Application to Multi-Tank Water Systems". In: *IEEE Access* 8 (2020), pp. 173227–173238. DOI: 10.1109/ACCESS.2020.3025194.

# RL en sistemas de tanques acoplados

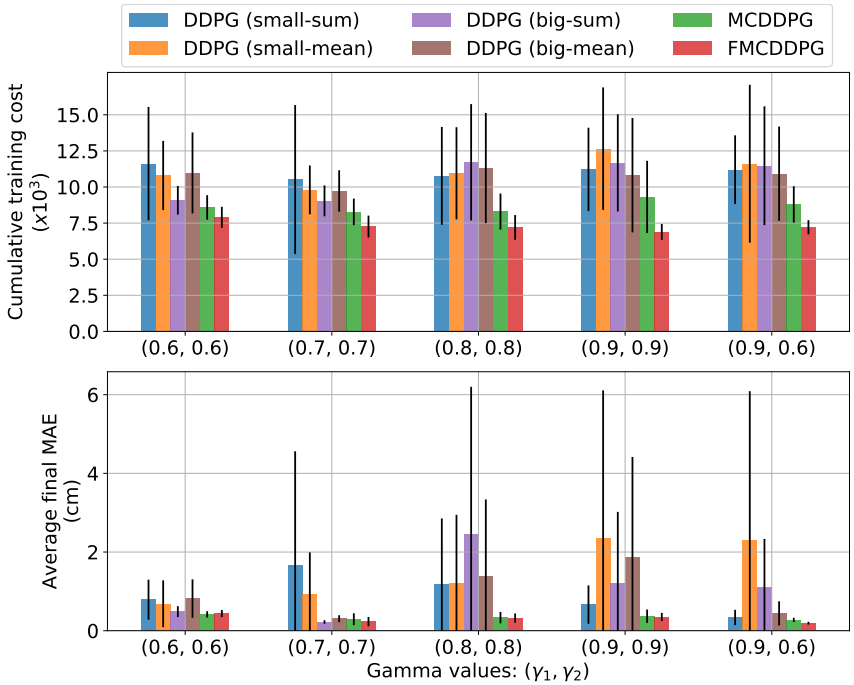
## Desempeño de la arquitectura multi-crítico en quad-tanks:

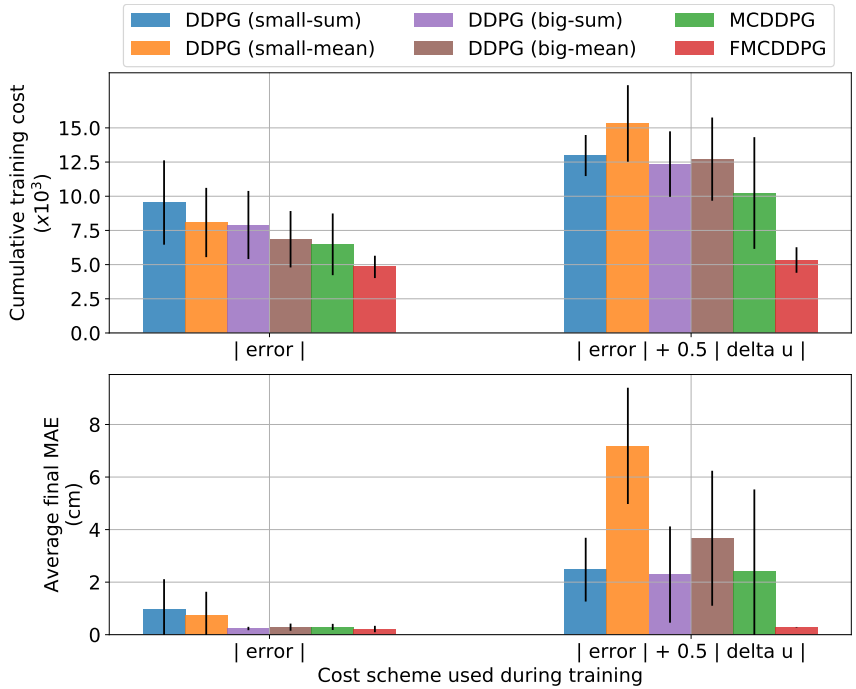



J. Martinez-Piazuelo et al. "A Multi-Critic Reinforcement Learning Method: An Application to Multi-Tank Water Systems". In: *IEEE Access* 8 (2020), pp. 173227–173238. DOI: 10.1109/ACCESS.2020.3025194.

## Desempeño de la arquitectura multi-crítico en quad-tanks y tower-tanks:




J. Martinez-Piazuelo et al. "A Multi-Critic Reinforcement Learning Method: An Application to Multi-Tank Water Systems". In: *IEEE Access* 8 (2020), pp. 173227–173238. DOI: 10.1109/ACCESS.2020.3025194.

# Taller de programación

**Taller de programación en Python + Numpy**

github.com/Martinez-Piazuelo/AEOC-NeuralNetworks-2020