

# APRENDIZAJE REFORZADO Y PROGRAMACIÓN DINÁMICA

*EL7021: Seminario de robótica y sistemas autónomos*

Francisco Leiva<sup>2</sup>    Javier Ruiz-del-Solar<sup>1,2</sup>

<sup>1</sup>Departamento de Ingeniería Eléctrica, Universidad de Chile

<sup>2</sup>Advanced Mining Technology Center (AMTC), Universidad de Chile

Marzo, 2023

# ¿Qué es el aprendizaje reforzado?

## El aprendizaje reforzado es:

- ▶ Un problema,
- ▶ una clase de soluciones para dicho problema,
- ▶ una disciplina que estudia dicho problema y sus soluciones.<sup>1</sup>

---

<sup>1</sup>Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

# ¿Qué es el aprendizaje reforzado?

## El aprendizaje reforzado es:

- ▶ Un problema,
  - ▶ una clase de soluciones para dicho problema,
  - ▶ una disciplina que estudia dicho problema y sus soluciones.<sup>1</sup>
- 
- ▶ El aprendizaje reforzado aborda el problema de aprender a “*como actuar*” para lograr un objetivo.
  - ▶ Al aprendiz no se le dice qué hacer (tiene que aprender a través de experiencias).
  - ▶ Las acciones del aprendiz tienen consecuencias.

## ¿Suena familiar?

---

<sup>1</sup>Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

# El problema que aborda el aprendizaje reforzado

Queremos que un **agente** aprenda como actuar con el fin de **maximizar una señal numérica de recompensa** que este obtiene a partir de sus **interacciones con el ambiente**.

# El problema que aborda el aprendizaje reforzado

Queremos que un **agente** aprenda como actuar con el fin de **maximizar una señal numérica de recompensa** que este obtiene a partir de sus **interacciones con el ambiente**.

- ▶ El agente ejecuta **acciones** de acuerdo a su **política**.
- ▶ La ejecución de estas acciones generan **recompensas** para el agente, y tienen consecuencias (a priori inciertas), dado que podrían afectar el **estado** del ambiente, y así, futuras acciones y recompensas.

# Algunas analogías

- ▶ Los humanos podemos considerarnos como agentes que aprenden.
- ▶ Constantemente interactuamos con el mundo (nuestro ambiente).
- ▶ Nuestras acciones tienen consecuencias.
- ▶ Elegimos acciones de acuerdo a una política (¿nuestra personalidad? ¿nuestro carácter? ¿experiencias pasadas? ¿emociones?)

---

<sup>2</sup>David Silver et al. «Reward is enough». En: *Artificial Intelligence* 299 (2021), pág. 103535.

# Algunas analogías

- ▶ Los humanos podemos considerarnos como agentes que aprenden.
- ▶ Constantemente interactuamos con el mundo (nuestro ambiente).
- ▶ Nuestras acciones tienen consecuencias.
- ▶ Elegimos acciones de acuerdo a una política (¿nuestra personalidad? ¿nuestro carácter? ¿experiencias pasadas? ¿emociones?)
- ▶ Aprendemos cómo actuar para poder conseguir objetivos a través de nuestra experiencia.
- ▶ Lograr nuestros objetivos y propósitos puede pensarse como la maximización de recompensas (“*reward hypothesis*”).
- ▶ No obstante... ¿cuál es la función de recompensa para un humano? ¿cuál es su propósito? (“*reward is enough*” hypothesis?<sup>2</sup>)

---

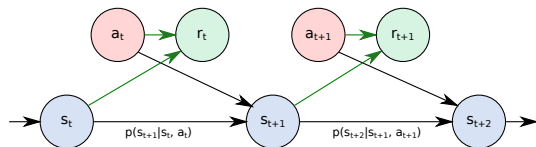
<sup>2</sup>David Silver et al. «Reward is enough». En: *Artificial Intelligence* 299 (2021), pág. 103535.

# Procesos de Decisión de Markov

## Markov Decision Processes (MDPs)

- Formalización clásica para modelar problemas de toma de decisiones secuenciales.
- Un MDP es definido por la tupla  $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$ , donde:

$\mathcal{S}$  : conjunto de estados  
 $\mathcal{A}$  : conjunto de acciones  
 $\mathcal{T}$  : función de transición de estados  
 $r$  : función de recompensa



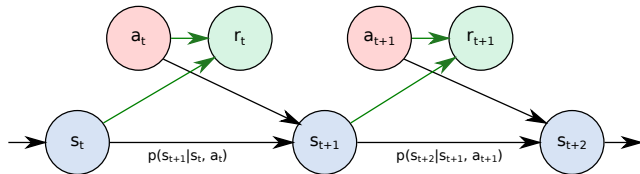


# Procesos de Decisión de Markov

## MDPs

Para  $t = 1, \dots, T$ :

- ▶ El agente se encuentra en un estado  $s_t \in \mathcal{S}$ .
- ▶ El agente elige y ejecuta una acción  $a_t \in \mathcal{A}$  de acuerdo a su política  $\pi(a_t|s_t)$ .
- ▶ El ambiente evoluciona a nuevo estado  $s_{t+1} \in \mathcal{S}$  de acuerdo a la función de transición  $\mathcal{T}(s, a, s') = p(s_{t+1}|s_t, a_t)$  y el agente recibe una recompensa escalar  $r_t$ , de acuerdo a  $\mathcal{R}(s, a)$ .



# La propiedad de Markov

## *The Markov property*

- ▶ En un MDP, la información necesaria para caracterizar interacciones agente-ambiente pasadas está contenida en el estado actual.
- ▶ Dichos estados se dice que poseen la “*propiedad de Markov*”.

***“Dado el presente, el futuro no depende del pasado”.***

# La propiedad de Markov

## *The Markov property*

- ▶ En un MDP, la información necesaria para caracterizar interacciones agente-ambiente pasadas está contenida en el estado actual.
- ▶ Dichos estados se dice que poseen la “*propiedad de Markov*”.

***“Dado el presente, el futuro no depende del pasado”.***

- ▶ ¿Se les ocurre algún ejemplo?
- ▶ ¿Y algún contraejemplo?

# Procesos de Decisión de Markov Parcialmente Observables

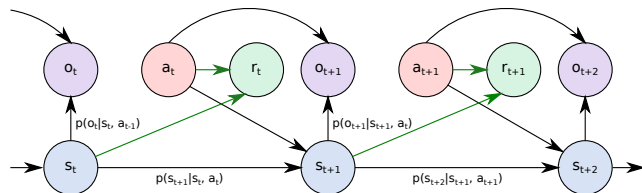
## *Partially Observable Markov Decision Processes (POMDPs)*

- Un POMDP es definido por la tupla  $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \Omega, \mathcal{O})$ , donde:

$(\mathcal{S}, \mathcal{A}, \mathcal{T}, r)$  : define un MDP

$\Omega$  : conjunto de observaciones

$\mathcal{O}$  : función de observación

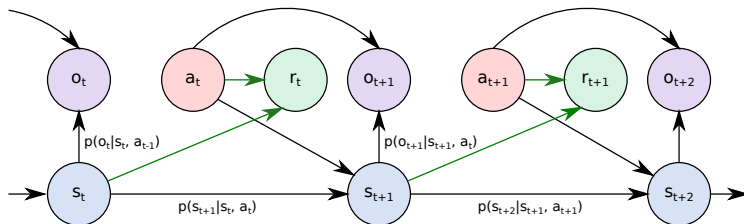


# Procesos de Decisión de Markov Parcialmente Observables

## *Partially Observable Markov Decision Processes (POMDPs)*

**For  $t = 1, \dots, T$ :**

- ▶ El agente se encuentra en un estado  $s_t \in \mathcal{S}$ , y recibe una observación  $o_t \in \Omega$  dado  $\mathcal{O}(s', a, o) = p(o_t | s_t, a_{t-1})$ .
- ▶ El agente selecciona y ejecuta una acción  $a_t \in \mathcal{A}$  de acuerdo a su política  $\pi(a_t | o_t)$ .
- ▶ El ambiente evoluciona a un nuevo estado  $s_{t+1} \in \mathcal{S}$  de acuerdo a su función de transición  $\mathcal{T}(s, a, s') = p(s_{t+1} | s_t, a_t)$  y el agente recibe una recompensa escalar  $r_t$ , de acuerdo a  $\mathcal{R}(s, a)$ .



# El objetivo del aprendizaje reforzado

- La interacción agente-ambiente da lugar a una distribución de probabilidad sobre secuencias de pares estado-acción, a las que llamaremos “*trayectorias*”.

$$\underbrace{p_{\pi}(s_1, a_1, \dots, s_T, a_T)}_{p_{\pi}(\tau)} = p(s_1) \prod_{t=1}^T \pi(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

# El objetivo del aprendizaje reforzado

- La interacción agente-ambiente da lugar a una distribución de probabilidad sobre secuencias de pares estado-acción, a las que llamaremos “trayectorias”.

$$\underbrace{p_{\pi}(s_1, a_1, \dots, s_T, a_T)}_{p_{\pi}(\tau)} = p(s_1) \prod_{t=1}^T \pi(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

- El objetivo del aprendizaje reforzado consiste en encontrar una política que maximice el valor esperado del retorno (suma descontada de recompensas) que recibe el agente con respecto a las trayectorias que visita en su interacción con el ambiente.

$$J_{\text{RL}}(\pi) = \mathbb{E}_{\tau \sim p_{\pi}(\tau)} \left[ \sum_{t=1}^T \gamma^{t-1} r(s_t, a_t) \right]$$

# Funciones de valor

- Para resolver un problema de aprendizaje reforzado, uno debe considerar el retorno que el agente podría recibir de acuerdo a las acciones que este ejecuta.
- Para formalizar esta idea, definamos la función de valor  $V^\pi(s)$  y la función de valor del par estado-acción (o función Q)  $Q^\pi(s, a)$ , como sigue:

$$V^\pi(s) = \mathbb{E}_{\tau_t \sim p_\pi(\tau_t)} \left[ \sum_{k=t}^T \gamma^{k-t} r(s_k, a_k) \middle| s_t = s \right]$$

$$Q^\pi(s, a) = \mathbb{E}_{\tau_t \sim p_\pi(\tau_t)} \left[ \sum_{k=t}^T \gamma^{k-t} r(s_k, a_k) \middle| s_t = s, a_t = a \right]$$



# Value functions

Notando que  $J_{\text{RL}}(\pi) = \mathbb{E}_{s_1 \sim p(s)} V^\pi(s_1)$ , buscamos una política que tenga asociada una función de valor *óptima*.

## Función de valor óptima

$$V^*(s) = \max_{\pi} V^\pi(s)$$

## Función de valor Q óptima

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$$

# Value functions

Notando que  $J_{RL}(\pi) = \mathbb{E}_{s_1 \sim p(s)} V^\pi(s_1)$ , buscamos una política que tenga asociada una función de valor *óptima*.

## Función de valor óptima

$$V^*(s) = \max_{\pi} V^\pi(s)$$

## Función de valor Q óptima

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$$

- ▶ Estas funciones especifican el retorno máximo que el agente puede obtener.
- ▶ Si tuviéramos acceso a una función de valor óptima, podríamos derivar una política “*óptima*”.

# Políticas y funciones de valor

Las funciones de valor establecen un orden parcial sobre las políticas: una política  $\pi$  se dice mejor que  $\pi'$  si y solo si  $V^\pi(s) \geq V^{\pi'}(s)$ , para todo  $s \in \mathcal{S}$ .

Existe al menos una política óptima  $\pi^*$ , tal que  $\pi^* \geq \pi$  para toda política  $\pi$ .  
Todas las políticas óptimas comparten las mismas funciones de valor óptimas:

$$\begin{aligned}V^{\pi^*}(s) &= V^*(s) \\ Q^{\pi^*}(s, a) &= Q^*(s, a)\end{aligned}$$

# Políticas y funciones de valor

Las funciones de valor establecen un orden parcial sobre las políticas: una política  $\pi$  se dice mejor que  $\pi'$  si y solo si  $V^\pi(s) \geq V^{\pi'}(s)$ , para todo  $s \in \mathcal{S}$ .

Existe al menos una política óptima  $\pi^*$ , tal que  $\pi^* \geq \pi$  para toda política  $\pi$ .  
Todas las políticas óptimas comparten las mismas funciones de valor óptimas:

$$\begin{aligned}V^{\pi^*}(s) &= V^*(s) \\ Q^{\pi^*}(s, a) &= Q^*(s, a)\end{aligned}$$

Usando las funciones de valor óptimas, es posible derivar una política óptima.  
Por ejemplo:

$$\pi^*(a|s) = \begin{cases} 1 & \text{si } a = \arg \max_{a \in \mathcal{A}} Q^*(s, a), \\ 0 & \text{si no.} \end{cases}$$

# Ecuaciones de Bellman

Existe una relación directa entre  $V^\pi(s)$  y  $Q^\pi(s, a)$ :

$$\begin{aligned}V^\pi(s) &= \mathbb{E}_{a \sim \pi(a|s)} Q^\pi(s, a) \\Q^\pi(s, a) &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V^\pi(s')\end{aligned}$$

Con lo anterior, se pueden establecer relaciones de recurrencia para  $V^\pi(s)$  y  $Q^\pi(s, a)$ , las cuales se conocen como “ecuaciones de Bellman”:

$$\begin{aligned}V^\pi(s) &= \mathbb{E}_{a \sim \pi(a|s)} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V^\pi(s')] \\Q^\pi(s, a) &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} \mathbb{E}_{a' \sim \pi(a'|s')} Q^\pi(s', a')\end{aligned}$$

# Ecuaciones de optimalidad de Bellman

Ecuaciones de Bellman para  $V^\pi(s)$  y  $Q^\pi(s, a)$ :

$$\begin{aligned}V^\pi(s) &= \mathbb{E}_{a \sim \pi(a|s)} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V^\pi(s')] \\Q^\pi(s, a) &= r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} \mathbb{E}_{a' \sim \pi(a'|s')} Q^\pi(s', a')\end{aligned}$$

Tanto  $V^*(s)$  como  $Q^*(s, a)$  satisfacen su respectiva ecuación de Bellman, no obstante, en este caso pueden ser escritas de manera especial considerando la definición de las funciones de valor óptimas. Estas ecuaciones se conocen como “ecuaciones de optimalidad de Bellman”:

$$\begin{aligned}V^*(s) &= \max_{a \in \mathcal{A}} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V^*(s')] \\Q^*(s, a) &= r(s, a) + \gamma \max_{a' \in \mathcal{A}} \mathbb{E}_{s' \sim p(s'|s, a)} Q^*(s', a')\end{aligned}$$

# Programación dinámica

*Dynamic Programming*

## Programación

- ▶ Optimización.

## Dinámica

- ▶ Naturaleza del problema.

Principales características:

- ▶ Divide el problema en sub-problemas.
- ▶ Combina soluciones de sub-problemas para obtener una solución general.

# Programación dinámica

## *Dynamic Programming*

### Programación

- ▶ Optimización.

### Dinámica

- ▶ Naturaleza del problema.

Principales características:

- ▶ Divide el problema en sub-problemas.
- ▶ Combina soluciones de sub-problemas para obtener una solución general.

¿Cuándo usar programación dinámica?

- ▶ La solución puede ser obtenida dividiendo el problema en sub-problemas.
- ▶ Los sub-problemas son recurrentes (así que sus soluciones pueden ser reutilizadas).



# Programación dinámica

## *Dynamic Programming*

### Programación

- ▶ Optimización.

### Dinámica

- ▶ Naturaleza del problema.

Principales características:

- ▶ Divide el problema en sub-problemas.
- ▶ Combina soluciones de sub-problemas para obtener una solución general.

¿Cuándo usar programación dinámica?

- ▶ La solución puede ser obtenida dividiendo el problema en sub-problemas.
- ▶ Los sub-problemas son recurrentes (así que sus soluciones pueden ser reutilizadas).

Los MDPs cumplen lo anterior:

- ▶ Existen relaciones recursivas (ecuaciones de Bellman).
- ▶ Los valores que toman las funciones de valor pueden ser almacenados y reutilizados.

# Programación dinámica

## *Policy Evaluation*

- ▶ Se busca evaluar una política  $\pi$ .
- ▶ Para ello simplemente se aplica la ecuación de Bellman para la función de valor, como regla de actualización iterativa para estimaciones de  $V^\pi(s)$ .

Ecuación de Bellman para  $V^\pi(s)$ :

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V^\pi(s')]$$

Regla de actualización iterativa:

$$V_{k+1}^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V_k^\pi(s')]$$

# Programación dinámica

## *Policy Evaluation*

---

### Algoritmo 1: Policy Evaluation

---

Dar como entrada una política  $\pi$

Inicializar umbral  $\epsilon > 0$  y  $\Delta v \leftarrow 0$

Inicializar  $V^\pi(s)$  (aleatoriamente), pero con  $V^\pi(s_{\text{terminal}}) \leftarrow 0$

**while**  $\Delta v < \epsilon$  **do**

$\Delta v \leftarrow 0$

**foreach**  $s \in \mathcal{S}$  **do**

$v_s \leftarrow V^\pi(s)$

$V^\pi(s) = \mathbb{E}_{a \sim \pi(a|s)} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V^\pi(s')]$

$\Delta v \leftarrow \max\{\Delta v, |v_s - V^\pi(s)|\}$

**end**

**end**

---

# Programación dinámica

## *Policy Iteration*

- ▶ Queremos encontrar la política óptima.
- ▶ ¿Cómo?
  - ▶ Evaluamos la política (*policy evaluation*).
  - ▶ Mejoramos la política actuando de manera ávara con respecto a  $V^\pi(s)$ .

Este proceso siempre converge a la política óptima  $\pi^*$ .

# Programación dinámica

## Policy Iteration

---

### Algoritmo 2: Policy Iteration

---

Inicializar  $\pi$  y  $V^\pi(s)$  (aleatoriamente), pero con  $V^\pi(s_{\text{terminal}}) \leftarrow 0$

Inicializar umbral  $\epsilon > 0$  y  $\Delta v \leftarrow 0$

Inicializar variable *estable*  $\leftarrow$  *false*

**while**  $\neg$ *estable* **do**

    PolicyEvaluation( $V^\pi(s)$ ,  $\Delta v$ ,  $\epsilon$ )

*estable*  $\leftarrow$  *true*

**foreach**  $s \in \mathcal{S}$  **do**

$a_s \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg \max_{a \in \mathcal{A}} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)} V^\pi(s')]$

**if**  $a_s \neq \pi(s)$  **then**

*estable*  $\leftarrow$  *false*

**end**

**end**

**end**

---

# Programación dinámica

## *Value Iteration*

¿Por qué no actualizar la política tras solo una interacción de la función de valor?

- Estimamos la función de valor  $V^*(s)$  usando la ecuación de Bellman.

Ecuación de Bellman para  $V^*(s)$ :

$$V^*(s) = \max_{a \in \mathcal{A}} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V^*(s')]$$

Regla de actualización iterativa:

$$V_{k+1}^*(s) = \max_{a \in \mathcal{A}} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V_k^*(s')]$$

# Programación dinámica

## Value Iteration

---

### Algoritmo 3: Value Iteration

---

Inicializar  $\pi$

Inicializar umbral  $\epsilon > 0$  y  $\Delta v \leftarrow 0$

Inicializar  $V^\pi(s)$  (aleatoriamente), pero con  $V^\pi(s_{\text{terminal}}) \leftarrow 0$

**while**  $\Delta v < \epsilon$  **do**

$\Delta v \leftarrow 0$

**foreach**  $s \in \mathcal{S}$  **do**

$v_s \leftarrow V^\pi(s)$

$V^\pi(s) = \max_{a \in \mathcal{A}} [r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s, a)} V^\pi(s')]$

$\Delta v \leftarrow \max\{\Delta v, |v_s - V^\pi(s)|\}$

**end**

**end**

---

# Aprendizaje Reforzado

## Algoritmos

### *Model-based vs Model-free*

Hacen o no uso de un modelo del ambiente.

### *Policy gradient*

Buscan  $\pi(a|s)$  a través de la optimización directa de  $J_{RL}(\pi)$ .

### *Value-based*

Aproximan  $V^*(s)$  o  $Q^*(s, a)$  para derivar una política.

### *Actor-Critic*

Aproximan conjuntamente  $V^*(s)$  o  $Q^*(s, a)$  y una política  $\pi(a|s)$ .



# Aprendizaje Reforzado

## *Reinforcement Learning (RL)*

- ▶ ¿Cómo aprender a actuar sin tener acceso a un modelo perfecto del MDP?
  - ▶ Mediante interacciones con el ambiente.
- ▶ ¿Cómo hacer esto?
  - ▶ Métodos de Monte Carlo.
  - ▶ Aprendizaje por diferencia temporal.
  - ▶  $n$ -step Bootstrapping.
  - ▶ Aprender un modelo del ambiente mediante interacciones y luego usarlo para planificar.
  - ▶ Otros...
- ▶ En lo que sigue se va a presentar brevemente una introducción a métodos de Monte Carlo y de aprendizaje por diferencia temporal.

# Métodos de Monte Carlo

## *Monte Carlo methods*

- ▶ La idea general de estos métodos es emplear repetidamente muestreos aleatorios para realizar alguna estimación.
- ▶ En el contexto de RL, se pueden usar para estimar funciones de valor usando las experiencias obtenidas por la interacción agente-ambiente, y usar esto para derivar una política.
- ▶ No requiere conocimiento completo del MDP (función de transición de estados).
- ▶ Como en programación dinámica, los siguientes problemas serán revisados:
  - ▶ *Policy evaluation* (predicción).
  - ▶ *Policy improvement*.
  - ▶ *Policy Iteration* (control).

# Monte Carlo Policy Evaluation

---

## Algoritmo 4: Monte Carlo Policy Evaluation (first-visit)

---

Dar como entrada una política  $\pi$

Inicializar  $V^\pi(s)$  aleatoriamente

Inicializar  $S(s) \leftarrow 0$  y  $N(s) \leftarrow 0$  para cada  $s \in \mathcal{S}$

**for** episodio = 1,  $N$  **do**

    Generar *rollout* siguiendo política  $\pi$ :  $s_1, a_1, r_1, s_2, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$

$R \leftarrow 0$

**for**  $t=T-1, 1$  **do**

$R \leftarrow r_t + \gamma R$

**if**  $s_t \notin \{s_1, \dots, s_{t-1}\}$  **then**

$N(s_t) \leftarrow N(s_t) + 1$

$S(s_t) \leftarrow S(s_t) + R$

**end**

**end**

$V^\pi(s) \leftarrow \frac{S(s)}{N(s)}$  para todo  $s \in \mathcal{S}$

**end**

---

# Monte Carlo Policy Evaluation

- ▶ Sin conocer el modelo del ambiente,  $V^\pi(s)$  no es suficiente para derivar una política.
- ▶ No obstante, sin conocer un modelo del ambiente, es posible derivar  $\pi$  a partir de  $Q^\pi(s, a)$ .
- ▶ Para estimar  $Q^\pi(s, a)$  usando el enfoque Monte Carlo, se sigue el mismo procedimiento que en *Monte Carlo Policy Evaluation*, con las siguientes consideraciones:
  - ▶ Se visitan pares estado-acción  $(s, a)$ .
  - ▶ Pueden existir problemas de exploración.

# Monte Carlo Control

---

## Algoritmo 5: Monte Carlo Control (exploring starts, first-visit)

---

Inicializar una política  $\pi(s)$

Inicializar  $Q^\pi(s, a)$  aleatoriamente

Inicializar  $S(s, a) \leftarrow 0$  y  $N(s, a) \leftarrow 0$  para cada  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}$

**for** episodio = 1,  $N$  **do**

    Seleccionar aleatoriamente  $(s_1, a_1)$

    Generar *rollout* siguiendo política  $\pi$ , desde  $(s_1, a_1)$ :  $s_1, a_1, r_1, s_2, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$

$R \leftarrow 0$

**for**  $t=T-1, 1$  **do**

$R \leftarrow r_t + \gamma R$

**if**  $(s_t, a_t) \notin \{s_1, a_1, \dots, s_{t-1}, a_{t-1}\}$  **then**

$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$

$S(s_t, a_t) \leftarrow S(s_t, a_t) + R$

**end**

**end**

$Q^\pi(s, a) \leftarrow \frac{S(s, a)}{N(s, a)}$  para todo  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}$

$\pi(s_t) \leftarrow \arg \max_{a \in \mathcal{A}} Q^\pi(s_t, a)$

**end**

---

# Aprendizaje por diferencia temporal

## *Temporal-Difference Learning* (TD-Learning)

- ▶ TD-Learning es una combinación de ideas entre programación dinámica y métodos de Monte Carlo:
  - ▶ No requiere el conocimiento de un modelo del ambiente.
  - ▶ Actualiza estimaciones en función de otras estimaciones.
- ▶ Como antes, se revisará el problema de predicción y control.

# Aprendizaje por diferencia temporal

## *Temporal-Difference Learning* (TD-Learning)

Revisión del problema de predicción:

- Actualización incremental Monte Carlo:

$$V_{k+1}^{\pi}(s) = V_k^{\pi}(s_t) + \alpha(R - V_k^{\pi}(s))$$

- Requiere experimentar episodios completos para actualizar.
- Actualización TD-Learning (TD(0)):

$$V_{k+1}^{\pi}(s) = V_k^{\pi}(s_t) + \alpha(r + \gamma V_k^{\pi}(s') - V_k^{\pi}(s))$$

- Puede actualizar tras un paso de tiempo.

# Policy Evaluation usando TD Learning

---

## Algoritmo 6: TD(0) Policy Evaluation

---

Dar como entrada una política  $\pi$

Inicializar  $V^\pi(s)$  aleatoriamente

Inicializar parámetro  $\alpha \in (0, 1]$

**for** *episodio* = 1, *N* **do**

    Obtener  $s_1$

**for**  $t=1, T$  **do**

        Seleccionar  $a_t$  de acuerdo a la política  $\pi(a_t|s_t)$

        Ejecutar acción  $a_t$ , observar  $r_t$  y  $s_{t+1}$

$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha(r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t))$

**end**

**end**

---



# SARSA (On-Policy TD control)

---

## Algoritmo 7: SARSA $\epsilon$ -greedy

---

Inicializar  $Q^\pi(s, a)$  aleatoriamente

Inicializar parámetro  $\alpha \in (0, 1]$  y  $\epsilon > 0$

**for** *episodio* = 1, *N* **do**

    Obtener  $s_1$

    Con probabilidad  $\epsilon$ , elegir  $a_1$  aleatoriamente, si no,  $a_1 = \arg \max_{a \in \mathcal{A}} Q_\theta(s_1, a)$

**for**  $t=1, T$  **do**

        Ejecutar acción  $a_t$ , observar  $r_t$  y  $s_{t+1}$

        Con probabilidad  $\epsilon$ , elegir  $a_{t+1}$  aleatoriamente, si no,  $a_{t+1} = \arg \max_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a)$

$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha(r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t))$

**end**

**end**

---

# Q-Learning (Off-Policy TD Control)

---

## Algoritmo 8: Q-Learning $\epsilon$ -greedy

---

Inicializar  $Q^\pi(s, a)$  aleatoriamente

Inicializar parámetro  $\alpha \in (0, 1]$  y  $\epsilon > 0$

**for** episodio = 1,  $N$  **do**

    Obtener  $s_1$

**for**  $t=1, T$  **do**

        Con probabilidad  $\epsilon$ , elegir  $a_t$  aleatoriamente, si no,  $a_t = \arg \max_{a \in \mathcal{A}} Q_\theta(s_t, a)$

        Ejecutar acción  $a_t$ , observar  $r_t$  y  $s_{t+1}$

$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q^\pi(s_{t+1}, a') - Q^\pi(s_t, a_t))$

**end**

**end**

---

# Aprendizaje Reforzado

## *Reinforcement Learning (RL)*

Preguntas respondidas en las clases anteriores:

- ▶ ¿Qué es el aprendizaje reforzado?
- ▶ ¿Cuál es el objetivo del aprendizaje reforzado?
- ▶ ¿En qué se diferencia el aprendizaje reforzado del aprendizaje supervisado?
- ▶ ¿Qué es un MDP?
- ▶ ¿Qué es la programación dinámica?

Preguntas adicionales:

- ▶ ¿Como aprender a través de interacciones agente-ambiente?
- ▶ ¿Cuáles son las limitaciones del aprendizaje reforzado?
- ▶ ¿Cómo estas limitaciones pueden ser superadas?

# Aprendizaje Reforzado

## *Reinforcement Learning (RL)*

- ▶ ¿Cómo aprender a actuar sin tener acceso a un modelo perfecto del MDP?
  - ▶ Mediante interacciones con el ambiente.
- ▶ ¿Cómo hacer esto?
  - ▶ Métodos de Monte Carlo.
  - ▶ Aprendizaje por diferencia temporal.
  - ▶  $n$ -step Bootstrapping.
  - ▶ Aprender un modelo del ambiente mediante interacciones y luego usarlo para planificar.
  - ▶ Otros...
- ▶ En lo que sigue se va a presentar brevemente una introducción a métodos de Monte Carlo y de aprendizaje por diferencia temporal.

# Métodos de Monte Carlo

## *Monte Carlo methods*

- ▶ La idea general de estos métodos es emplear repetidamente muestreos aleatorios para realizar alguna estimación.
- ▶ En el contexto de RL, se pueden usar para estimar funciones de valor usando las experiencias obtenidas por la interacción agente-ambiente, y usar esto para derivar una política.
- ▶ No requiere conocimiento completo del MDP (función de transición de estados).
- ▶ Como en programación dinámica, los siguientes problemas serán revisados:
  - ▶ *Policy evaluation* (predicción).
  - ▶ *Policy improvement*.
  - ▶ *Policy Iteration* (control).

# Monte Carlo Policy Evaluation

---

## Algoritmo 9: Monte Carlo Policy Evaluation (first-visit)

---

Dar como entrada una política  $\pi$

Inicializar  $V^\pi(s)$  aleatoriamente

Inicializar  $S(s) \leftarrow 0$  y  $N(s) \leftarrow 0$  para cada  $s \in \mathcal{S}$

**for** episodio = 1,  $N$  **do**

    Generar *rollout* siguiendo política  $\pi$ :  $s_1, a_1, r_1, s_2, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$

$R \leftarrow 0$

**for**  $t=T-1, 1$  **do**

$R \leftarrow r_t + \gamma R$

**if**  $s_t \notin \{s_1, \dots, s_{t-1}\}$  **then**

$N(s_t) \leftarrow N(s_t) + 1$

$S(s_t) \leftarrow S(s_t) + R$

**end**

**end**

$V^\pi(s) \leftarrow \frac{S(s)}{N(s)}$  para todo  $s \in \mathcal{S}$

**end**

---

# Monte Carlo Policy Evaluation

- ▶ Sin conocer el modelo del ambiente,  $V^\pi(s)$  no es suficiente para derivar una política.
- ▶ No obstante, sin conocer un modelo del ambiente, es posible derivar  $\pi$  a partir de  $Q^\pi(s, a)$ .
- ▶ Para estimar  $Q^\pi(s, a)$  usando el enfoque Monte Carlo, se sigue el mismo procedimiento que en *Monte Carlo Policy Evaluation*, con las siguientes consideraciones:
  - ▶ Se visitan pares estado-acción  $(s, a)$ .
  - ▶ Pueden existir problemas de exploración.

# Monte Carlo Control

---

## Algoritmo 10: Monte Carlo Control (exploring starts, first-visit)

---

Inicializar una política  $\pi(s)$

Inicializar  $Q^\pi(s, a)$  aleatoriamente

Inicializar  $S(s, a) \leftarrow 0$  y  $N(s, a) \leftarrow 0$  para cada  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}$

**for** episodio = 1,  $N$  **do**

    Seleccionar aleatoriamente  $(s_1, a_1)$

    Generar *rollout* siguiendo política  $\pi$ , desde  $(s_1, a_1)$ :  $s_1, a_1, r_1, s_2, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$

$R \leftarrow 0$

**for**  $t=T-1, 1$  **do**

$R \leftarrow r_t + \gamma R$

**if**  $(s_t, a_t) \notin \{s_1, a_1, \dots, s_{t-1}, a_{t-1}\}$  **then**

$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$

$S(s_t, a_t) \leftarrow S(s_t, a_t) + R$

**end**

**end**

$Q^\pi(s, a) \leftarrow \frac{S(s, a)}{N(s, a)}$  para todo  $s \in \mathcal{S}$ ,  $a \in \mathcal{A}$

$\pi(s_t) \leftarrow \arg \max_{a \in \mathcal{A}} Q^\pi(s_t, a)$

**end**

---



# Aprendizaje por diferencia temporal

## *Temporal-Difference Learning* (TD-Learning)

- ▶ TD-Learning es una combinación de ideas entre programación dinámica y métodos de Monte Carlo:
  - ▶ No requiere el conocimiento de un modelo del ambiente.
  - ▶ Actualiza estimaciones en función de otras estimaciones.
- ▶ Como antes, se revisará el problema de predicción y control.

# Aprendizaje por diferencia temporal

## *Temporal-Difference Learning* (TD-Learning)

Revisión del problema de predicción:

- ▶ Actualización incremental Monte Carlo:

$$V_{k+1}^{\pi}(s) = V_k^{\pi}(s_t) + \alpha(R - V_k^{\pi}(s))$$

- ▶ Requiere experimentar episodios completos para actualizar.
- ▶ Actualización TD-Learning (TD(0)):

$$V_{k+1}^{\pi}(s) = V_k^{\pi}(s_t) + \alpha(r + \gamma V_k^{\pi}(s') - V_k^{\pi}(s))$$

- ▶ Puede actualizar tras un paso de tiempo.

# Policy Evaluation usando TD Learning

---

## Algoritmo 11: TD(0) Policy Evaluation

---

Dar como entrada una política  $\pi$

Inicializar  $V^\pi(s)$  aleatoriamente

Inicializar parámetro  $\alpha \in (0, 1]$

**for** *episodio* = 1, *N* **do**

    Obtener  $s_1$

**for**  $t=1, T$  **do**

        Seleccionar  $a_t$  de acuerdo a la política  $\pi(a_t|s_t)$

        Ejecutar acción  $a_t$ , observar  $r_t$  y  $s_{t+1}$

$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha(r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t))$

**end**

**end**

---

# SARSA (On-Policy TD control)

---

## Algoritmo 12: SARSA $\epsilon$ -greedy

---

Inicializar  $Q^\pi(s, a)$  aleatoriamente

Inicializar parámetro  $\alpha \in (0, 1]$  y  $\epsilon > 0$

**for** *episodio* = 1, *N* **do**

    Obtener  $s_1$

    Con probabilidad  $\epsilon$ , elegir  $a_1$  aleatoriamente, si no,  $a_1 = \arg \max_{a \in \mathcal{A}} Q_\theta(s_1, a)$

**for**  $t=1, T$  **do**

        Ejecutar acción  $a_t$ , observar  $r_t$  y  $s_{t+1}$

        Con probabilidad  $\epsilon$ , elegir  $a_{t+1}$  aleatoriamente, si no,  $a_{t+1} = \arg \max_{a \in \mathcal{A}} Q_\theta(s_{t+1}, a)$

$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha(r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t))$

**end**

**end**

---

# Q-Learning (Off-Policy TD Control)

---

## Algoritmo 13: Q-Learning $\epsilon$ -greedy

---

Inicializar  $Q^\pi(s, a)$  aleatoriamente

Inicializar parámetro  $\alpha \in (0, 1]$  y  $\epsilon > 0$

**for** episodio = 1,  $N$  **do**

    Obtener  $s_1$

**for**  $t=1, T$  **do**

        Con probabilidad  $\epsilon$ , elegir  $a_t$  aleatoriamente, si no,  $a_t = \arg \max_{a \in \mathcal{A}} Q_\theta(s_t, a)$

        Ejecutar acción  $a_t$ , observar  $r_t$  y  $s_{t+1}$

$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q^\pi(s_{t+1}, a') - Q^\pi(s_t, a_t))$

**end**

**end**

---

# Ok... ¿y cómo modelamos/implementamos esto en la práctica?

¿Qué necesitamos?

- ▶ Un problema que queramos resolver, y que no sabemos como resolver.
- ▶ Después:
  - ▶ Un agente.
  - ▶ Un ambiente.
  - ▶ Una función de transición.
  - ▶ Una función de recompensa.
  - ▶ Estados.
  - ▶ Acciones.
  - ▶ Una política.
  - ▶ ...

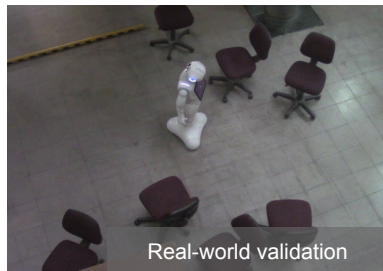
# Ok... ¿y cómo modelamos/implementamos esto en la práctica?

¿Qué necesitamos?

- ▶ Un problema que queramos resolver, y que no sabemos como resolver.
- ▶ Después:
  - ▶ Un agente.
  - ▶ Un ambiente.
  - ▶ Una función de transición.
  - ▶ Una función de recompensa.
  - ▶ Estados.
  - ▶ Acciones.
  - ▶ Una política.
  - ▶ ...
- ▶ Para muchos problemas, es a menudo muy riesgoso y/o caro realizar interacciones agente-ambiente en el mundo real, por lo que es usual utilizar simulaciones.
- ▶ En estos casos, luego de entrenar en simulaciones, podemos intentar desplegar la política en el mundo real (*sim2real*).

# Ejemplo: Navegación robótica

- ▶ El agente sería el robot, la política su controlador (para navegación).
- ▶ El ambiente sería el mundo con el que el robot interactúa (real o simulado).
- ▶ La función de transición sería la dinámica del mundo (leyes físicas, o un motor físico en el caso de las simulaciones).



**Figura:** Ejemplo de sim2real. Basado en Leiva, Lobos-Tsunekawa y Ruiz-del-Solar, «Collision avoidance for indoor service robots through multimodal deep reinforcement learning».



# Ejemplo: Navegación robótica

## Estados/Observaciones: Lo que el agente “ve”.

- ▶ Imágenes, mediciones de rango, otro tipo de datos que entreguen sensores (exteroceptivos o propioceptivos).
- ▶ “Información procesada” (e.g. detecciones de objetos).
- ▶ Etc.

# Ejemplo: Navegación robótica

## **Estados/Observaciones: Lo que el agente “ve”.**

- ▶ Imágenes, mediciones de rango, otro tipo de datos que entreguen sensores (exteroceptivos o propioceptivos).
- ▶ “Información procesada” (e.g. detecciones de objetos).
- ▶ Etc.

## **Acciones: Lo que el agente “hace”.**

- ▶ Control de bajo nivel sobre los actuadores del robot.
- ▶ Control de alto nivel sobre los controladores de bajo nivel (e.g. comandos de velocidad).
- ▶ Control de aún más alto nivel (e.g. selección de comportamientos).
- ▶ Etc.

# Ejemplo: Navegación robótica

## Función de recompensa

Función escalar que define el objetivo del agente.

### Ejemplo: Recompensa “*sparse*”

$$r_t = \begin{cases} 1 & \text{si el objetivo es alcanzado,} \\ 0 & \text{si no} \end{cases}$$

### Ejemplo: Recompensa “*densa*”

$$r_t = \begin{cases} +1 & \text{objetivo alcanzado,} \\ -1 & \text{colisión,} \\ K(d_t - d_{t-1}) & \text{otro caso.} \end{cases}$$

# Ejemplo: Navegación robótica

## Función de recompensa

Función escalar que define el objetivo del agente.

### Ejemplo: Recompensa “*sparse*”

$$r_t = \begin{cases} 1 & \text{si el objetivo es alcanzado,} \\ 0 & \text{si no} \end{cases}$$

### Ejemplo: Recompensa “*densa*”

$$r_t = \begin{cases} +1 & \text{objetivo alcanzado,} \\ -1 & \text{colisión,} \\ K(d_t - d_{t-1}) & \text{otro caso.} \end{cases}$$

## Condiciones episódicas (no siempre)

- ▶ ¿Cómo parte un episodio?
- ▶ ¿Cuándo termina un episodio?

# Pensemos en software

- ▶ ¿Qué necesitamos para modelar interacciones agente-ambiente?

# Pensemos en software

- ▶ ¿Qué necesitamos para modelar interacciones agente-ambiente?

## Pensemos lo anterior en términos de una API:

- ▶ Para reiniciar el ambiente:
  - ▶ `obs_t = env.reset()`

# Pensemos en software

- ▶ ¿Qué necesitamos para modelar interacciones agente-ambiente?

## Pensemos lo anterior en términos de una API:

- ▶ Para reiniciar el ambiente:
  - ▶ `obs_t = env.reset()`
- ▶ Para obtener las acciones que selecciona el agente:
  - ▶ `act_t = agent.select_action(obs_t)`

# Pensemos en software

- ▶ ¿Qué necesitamos para modelar interacciones agente-ambiente?

## Pensemos lo anterior en términos de una API:

- ▶ Para reiniciar el ambiente:
  - ▶ `obs_t = env.reset()`
- ▶ Para obtener las acciones que selecciona el agente:
  - ▶ `act_t = agent.select_action(obs_t)`
- ▶ Para dar lugar a una interacción agente-ambiente:
  - ▶ `obs_t1, rew_t, done_t = env.step(act_t)`



# Pensemos en software

- ▶ ¿Qué necesitamos para modelar interacciones agente-ambiente?

## Pensemos lo anterior en términos de una API:

- ▶ Para reiniciar el ambiente:
  - ▶ `obs_t = env.reset()`
- ▶ Para obtener las acciones que selecciona el agente:
  - ▶ `act_t = agent.select_action(obs_t)`
- ▶ Para dar lugar a una interacción agente-ambiente:
  - ▶ `obs_t1, rew_t, done_t = env.step(act_t)`
- ▶ Y quizás métodos auxiliares que nos permitan implementar a los métodos `reset` y `step`:
  - ▶ `env._set_action(act_t)`
  - ▶ `obs_t = env._get_observation()`
  - ▶ `rew_t = env._evaluate_reward_function(args)`
  - ▶ `done_t = env._check_terminal_state(args)`

# Pensemos en software

## Ejemplo:

```
""" Agent-environment interaction """
obs_t = env.reset()
done = False

for _ in range(nb_steps):

    act_t = agent.select_action(obs_t)
    obs_t1, rew_t, done_t = env.step(act_t)

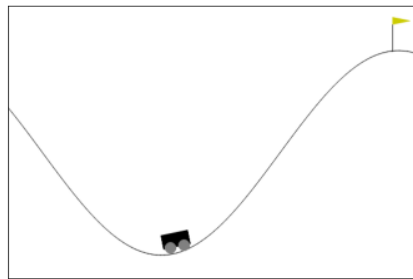
    obs_t = obs_t1

    if done_t:
        obs_t = env.reset()
```

# Pensemos en Q-Learning para un problema simple:

## “*The Mountain Car*”





- ▶ Objetivo: Que el auto llegue al lado de la bandera.
- ▶ Espacio de acciones discreto:  $\{0, 1, 2\}$  (left, do nothing, right).
- ▶ Espacio de estados continuo:
  - ▶ Posición del auto:  $x \in [-1, 2, 0, 6]$ .
  - ▶ Velocidad del auto:  $\dot{x} \in [-0,07, 0,07]$ .
- ▶ Función de recompensa:  $-1$  en cada paso de tiempo (hasta alcanzar la bandera).
- ▶ Condiciones iniciales: La posición es muestreada uniformemente de  $[-0,6, -0,4]$ , y la velocidad se deja en 0.
- ▶ Condición terminal: Alcanzar bandera ( $x > 0,5$ ).



**Figura:** Mountain Car environment de OpenAI Gym. Basado en Moore, *Efficient memory-based learning for robot control*.

**GRACIAS**

# Referencias

-  Leiva, Francisco, Kenzo Lobos-Tsunekawa y Javier Ruiz-del-Solar. «Collision avoidance for indoor service robots through multimodal deep reinforcement learning». En: *Robot World Cup*. Springer. 2019, págs. 140-153.
-  Moore, Andrew William. *Efficient memory-based learning for robot control*. Inf. téc. University of Cambridge, Computer Laboratory, 1990.
-  Silver, David et al. «Reward is enough». En: *Artificial Intelligence* 299 (2021), pág. 103535.
-  Sutton, Richard S y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.