

# Teoría del Cerebro y Neuroinformática

Ganglios Basales y Aprendizaje por refuerzo

Maestría en Ciencias en  
Computación



# Aprendizaje por refuerzo

---

- El estudio de la toma de decisiones mediante prueba y error para maximizar la recompensa o minimizar el castigo
- Un área principal de estudio en inteligencia artificial
- Requiere aprender de la experiencia para predecir las consecuencias gratificantes o punitivas de las acciones candidatas a fin de elegir la mejor acción

# Promedio incremental

---

- La media  $\mu_1, \mu_2, \dots$  de una secuencia  $x_1, x_2, \dots$   
Puede ser calculada incrementalmente,

$$\begin{aligned}\mu_k &= \frac{1}{k} \sum_{j=1}^k x_j \\ &= \frac{1}{k} \left( x_k + \sum_{j=1}^{k-1} x_j \right) \\ &= \frac{1}{k} (x_k + (k-1)\mu_{k-1}) \\ &= \mu_{k-1} + \frac{1}{k} (x_k - \mu_{k-1})\end{aligned}$$

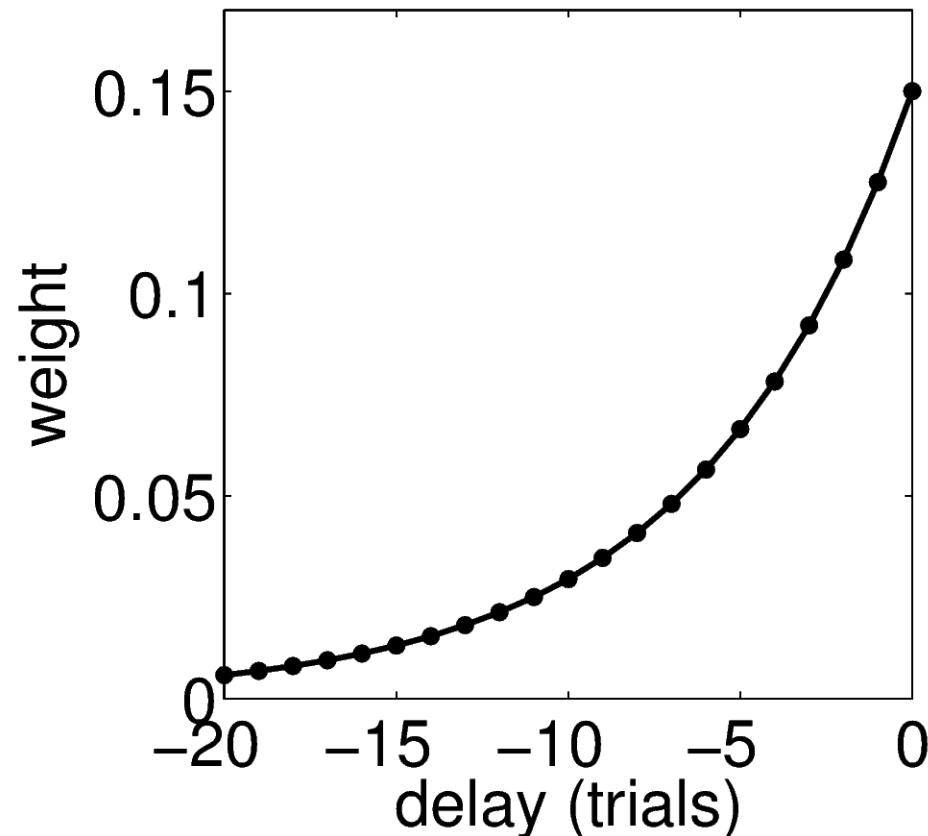
# Promedio exponencial

---

Uso  $\alpha$  con valores entre 0 y 1 en vez de  $n$

$\alpha$  cercana a 1 toma en cuenta más los datos recientes

$\alpha$  cercana a 0 toma en cuenta más los datos pasados



# Condicionamiento pavloviano

---

- No es aprendizaje por refuerzo, pero ayuda a comprender la predicción de recompensa



# Modelo Rescorla-Wagner

---

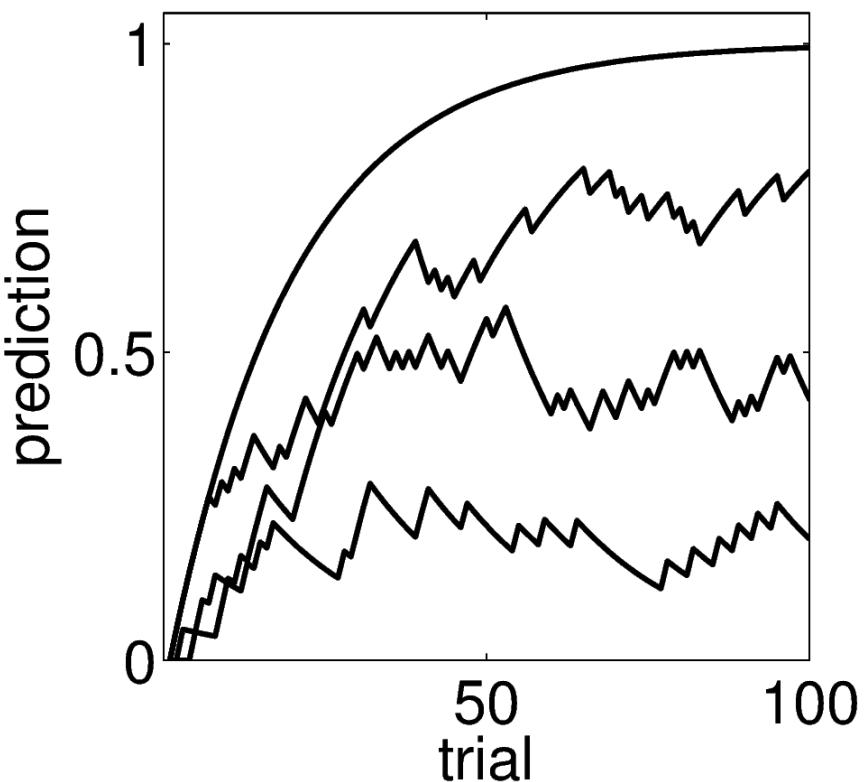
- En prueba t
  - escuchas la campana,  $s_t$
  - obtienes comida de cantidad  $r_t$  (puede ser cero)
- Digamos que  $V_t(s_t)$  mide tu creencia sobre la fuerza de asociación entre la campana y la comida. ¿Cómo deberías actualizar su creencia dada esta experiencia?
- Rescorla y Wagner, 1972:
$$V_{t+1}(s_t) = (1 - \alpha) V_t(s_t) + \alpha r_t$$
donde  $\alpha$  es la tasa de aprendizaje (promedio ponderado)
- Equivalente a:

$$V_{t+1}(s_t) = V_t(s_t) + \alpha \delta_t$$

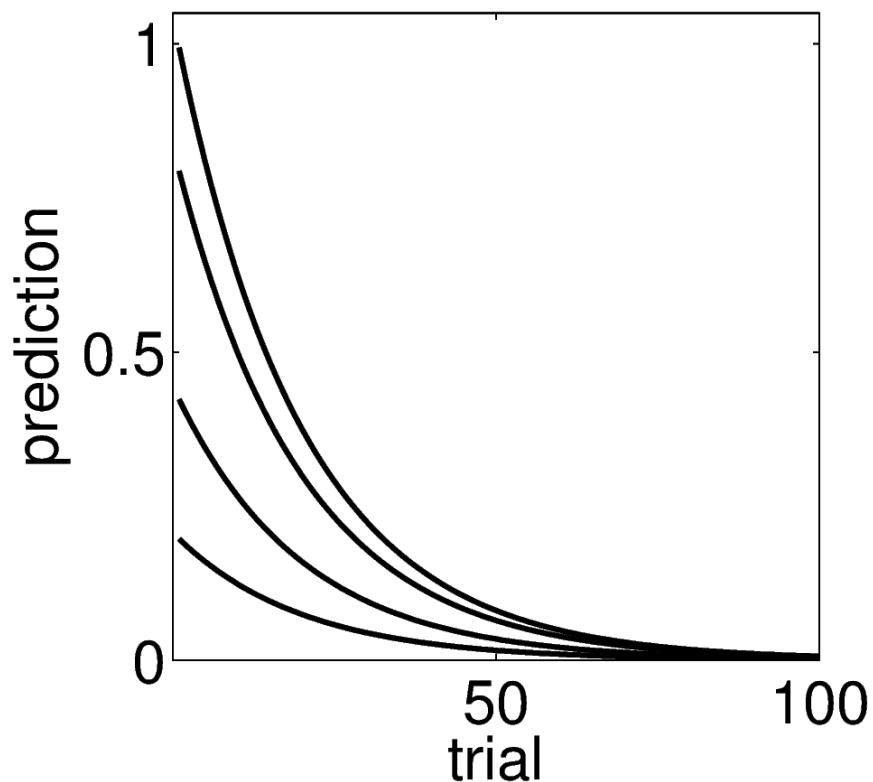
donde  $\delta_t$  es error de predicción  $r_t - V_t(s_t)$  "Aprendizaje impulsado por error"

# Predicción de recompensa

Adquisición (recompensa presente 25%, 50%, 75% y 100% de las veces)

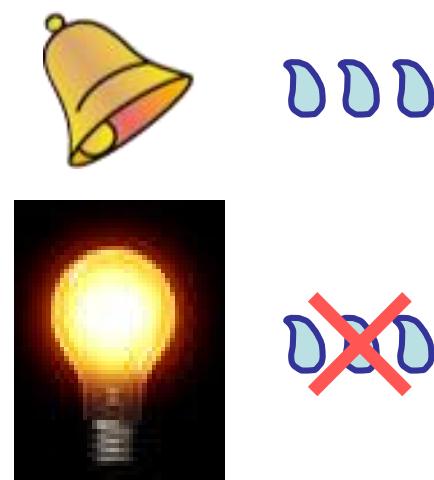


Extinción (estímulo presente sin recompensa)



# Bloqueo

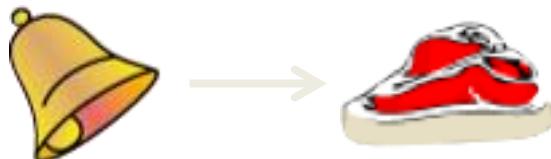
---



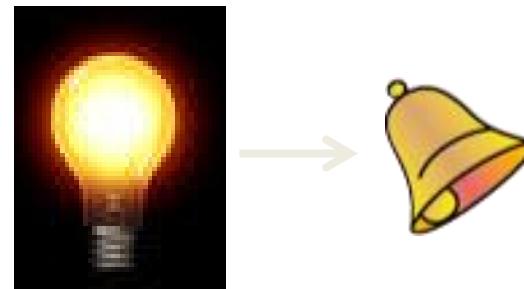
Evidencia de que el aprendizaje es impulsado por el error

# Condicionamiento de segundo orden

## Fase 1



## Fase 2



## Entrena:

## Resultado:



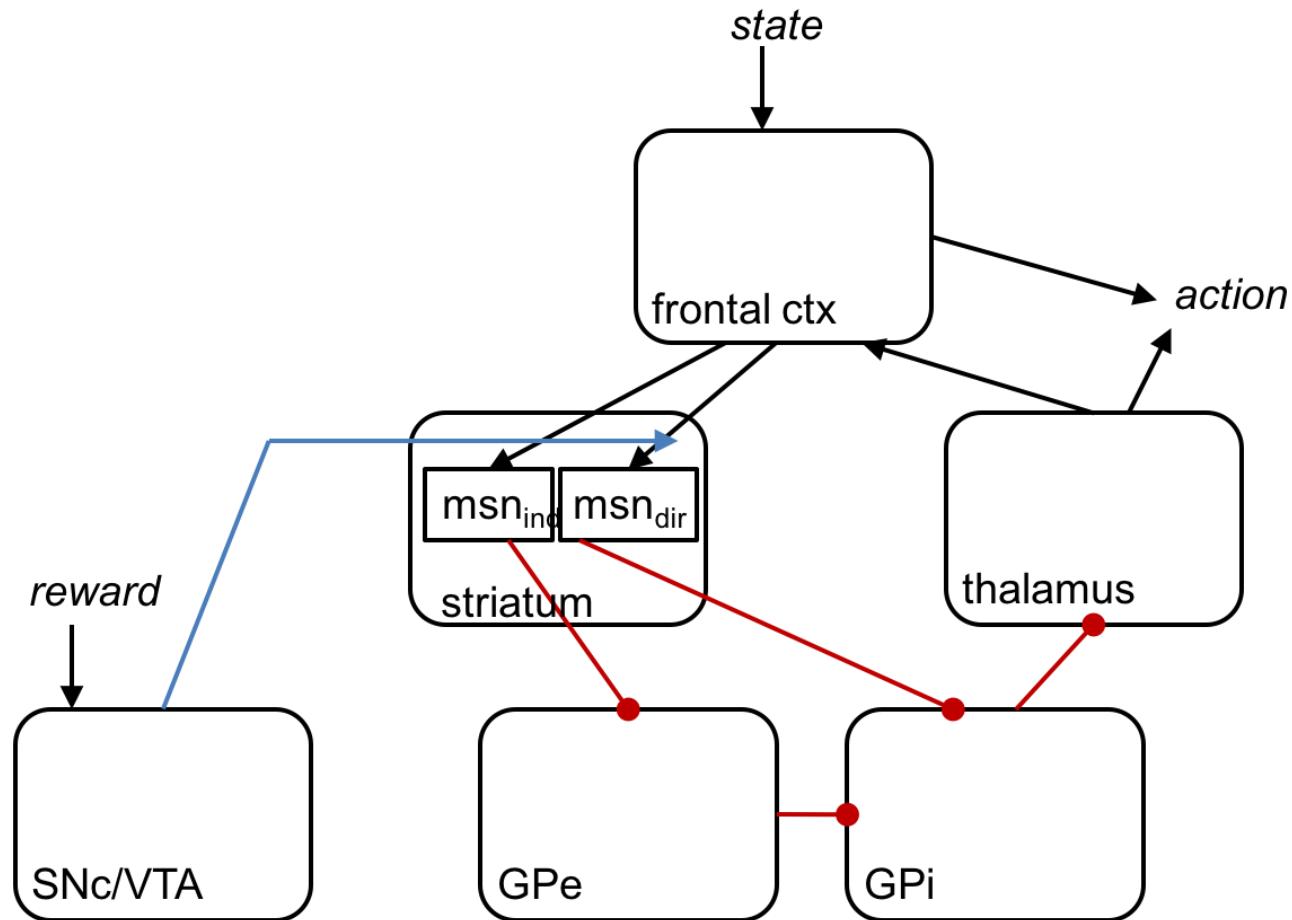
# ¿En el cerebro hay esta señal? Dopamina

---

- Neuromodulador administrado al cerebro anterior (especialmente ganglios basales) a través de proyecciones ascendentes de núcleos en el cerebro medio
- Sustancia negra parre compacta (SNc) y área ventral tegmental (VTA)
- Indicación de dos funciones
  - Recompensa (motivación, adicción)
  - Movimiento (enfermedad de Parkinson)
- ¿Cómo conciliar?

# Modelo de los Ganglios Basales

---



# Aprendizaje por refuerzo

---

- Aprendemos a través de la interacción con nuestro entorno
- Cuando aprendemos, muchas veces no hay supervisor explícito, pero sí hay una conexión **sensoriomotora** directa con el entorno.
- El ejercicio de esta conexión produce una gran cantidad de información acerca de la causa y el efecto, sobre las consecuencias de las acciones, y acerca de qué hacer con el fin de lograr los objetivos.

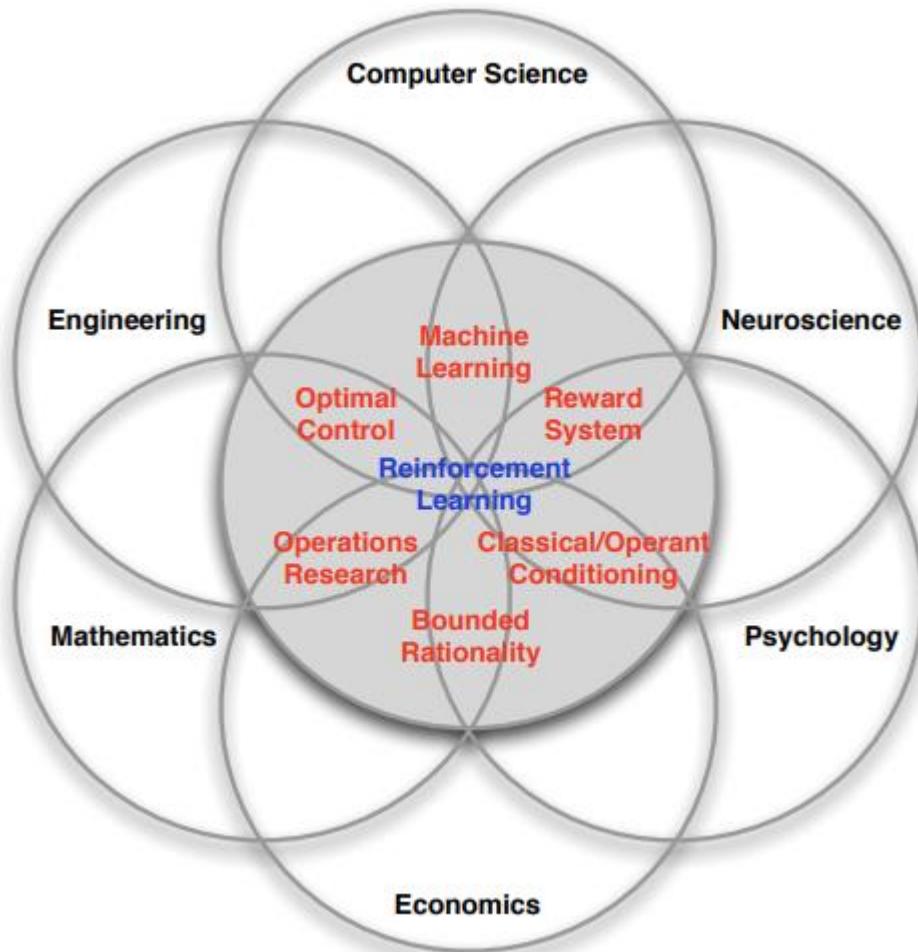
# Aprendizaje por refuerzo

---

- ¿Qué hace que el aprendizaje por refuerzo sea diferente de otros paradigmas de aprendizaje de máquina?
  - No hay ningún supervisor, solamente una señal de recompensa
  - La retroalimentación no es instantánea
  - El tiempo realmente importa (secuencial, observaciones no son independientes e idénticamente distribuidas)
    - Las acciones del agente afectan a los datos posteriores que recibe

# Aprendizaje por refuerzo

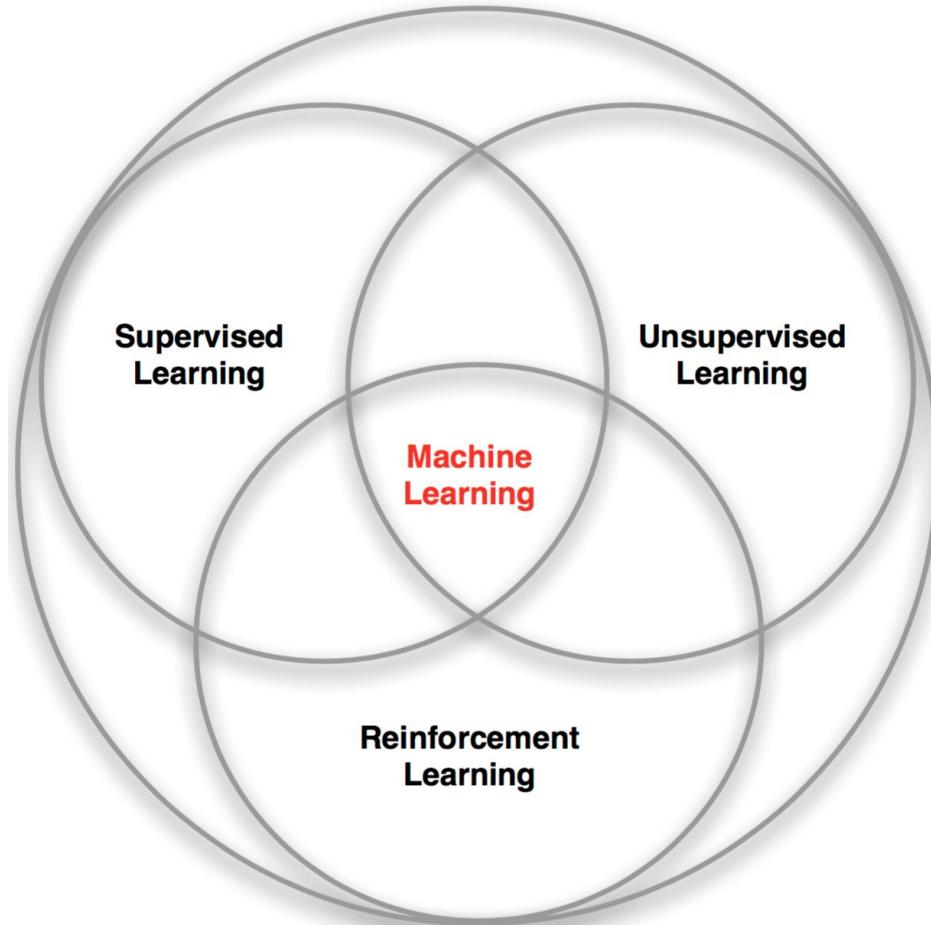
---



- Con el aprendizaje por refuerzo estudiamos la forma óptima de tomar decisiones

# Tipos de aprendizaje máquina

---



# Aprendizaje por refuerzo

---

- Es el aprendizaje de qué hacer - cómo mapear situaciones a acciones - con el fin de maximizar una señal de recompensa numérica.
- Al aprendiz no se le dice qué acciones tomar, como en la mayoría de las formas de aprendizaje de la máquina, sino que debe descubrir qué acciones producen la mayor recompensa probándolas.

# Ejemplos

---

- Realizar maniobras acrobáticas en un helicóptero
- Derrotar al campeón del mundo en Backgammon
- Administrar una cartera de inversión
- Controlar una central eléctrica
- Hacer que un robot humanoide camine
- Jugar muchos juegos de Atari diferentes mejor que los seres humanos

# Deep reinforcement learning

---



Ejemplo de deep learning

<https://www.youtube.com/watch?v=V1eYniJ0Rnk>

# Juegos resueltos con inteligencia artificial

---

Juego	Nivel de juego	Programa
Damas Americanas	Perfecto	Chinook
Ajedrez	Súper-humano	Deep Blue
Otelo	Súper-humano	Logistello
Backgammon	Súper-humano	TD-Gammon
Scrabble	Súper-humano	Maven
Go	Gran maestro	MoGo <sup>1</sup> , Crazy Stone <sup>2</sup> , Zen y AlphaGo <sup>3</sup>
Poker <sup>4</sup>	Súper-humano	Polaris

1 9 x 9

2 9 x 9 y 19 x 19

3 19 x 19

4 Heads-up Limit Texas Hold'em

# Juegos resueltos con aprendizaje por refuerzo

---

Juego	Nivel de juego	Programa
Damas Americanas	Perfecto	Chinook
Ajedrez	Maestro internacional	KnightCap / Meep
Otelo	Súper-humano	Logistello
Backgammon	Súper-humano	TD-Gammon
Scrabble	Súper-humano	Maven
Go	Gran maestro	MoGo <sup>1</sup> , Crazy Stone <sup>2</sup> , Zen y AlphaGo <sup>3</sup>
Poker <sup>4</sup>	Súper-humano	SmooCT

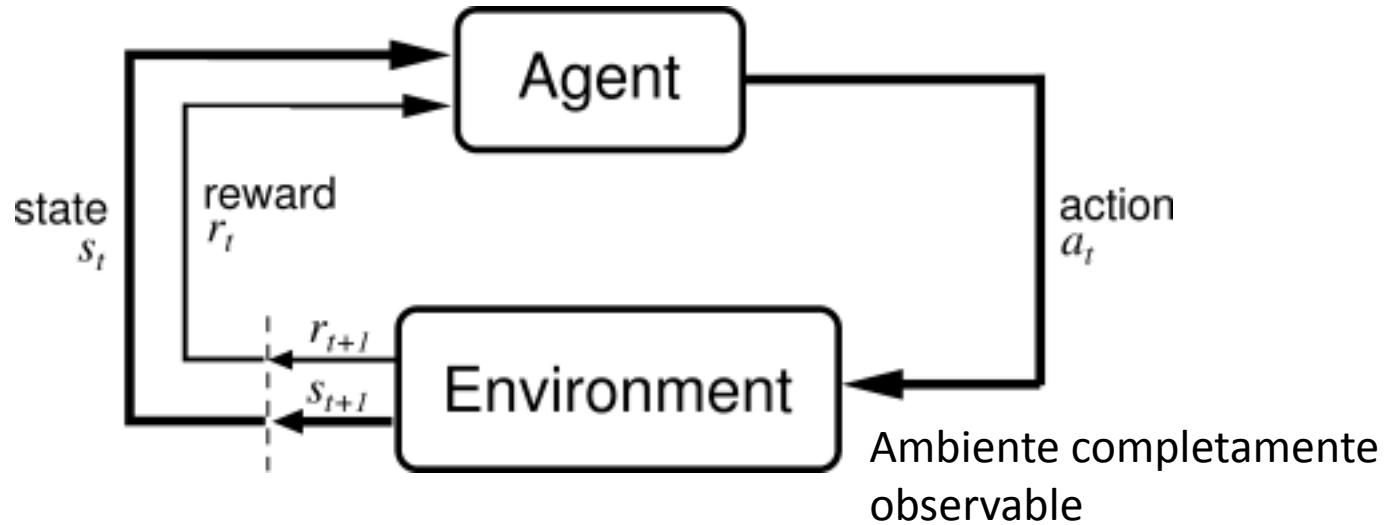
1 9 x 9

2 9 x 9 y 19 x 19

3 19 x 19

4 Heads-up Limit Texas Hold'em

# Agente



$r_{t+1} \in \mathbb{R}$  Recompensa numérica

$s_t \in \mathcal{S}$  Posibles estados

$a_t \in \mathcal{A}(s_t)$  Posibles acciones

$t = 0, 1, 2, 3, \dots$

$\pi_t(s, a)$  Política: probabilidad de seleccionar la acción  $a$  si el estado es  $s$  en el tiempo  $t$

El objetivo del agente es maximizar el monto del refuerzo que recibe a largo plazo

# Recompensa

---

- Una recompensa  $R_t$  es una señal de realimentación escalar
- Indica que tan bien se ha comportado el agente en el tiempo  $t$
- El objetivo del agente es maximizar la acumulación de recompensas
- El aprendizaje de refuerzo se basa en la hipótesis de recompensa
  - Todos los objetivos se pueden describir a través de la maximización de la recompensa acumulada esperada

# Ejemplos de recompensas

---

- Realizar maniobras acrobáticas en un helicóptero
  - Recompensa positiva por seguir la trayectoria deseada
  - Recompensa negativa por desplomarse
- Derrotar al campeón del mundo en Backgammon
  - Recompensa positiva por ganar
  - Recompensa negativa por perder
- Administrar una cartera de inversión
  - Recompensa positiva por cada \$ ganado
- Controlar una central eléctrica
  - Recompensa positiva por la generación de energía
  - Recompensa negativa por exceder los umbrales de seguridad
- Hacer que un robot humanoide camine
  - Recompensa positiva por avanzar
  - Recompensa negativa por caerse
- Jugar muchos juegos de Atari diferentes mejor que los seres humanos
  - Recompensa positiva por incrementar la puntuación
  - Recompensa negativa por disminuir la puntuación

# Secuencia de toma de decisiones

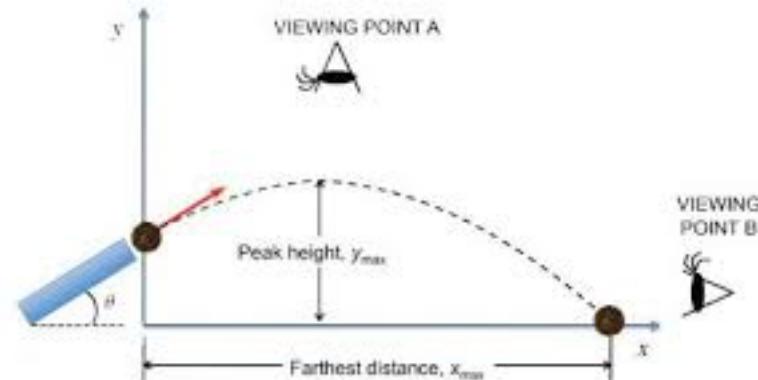
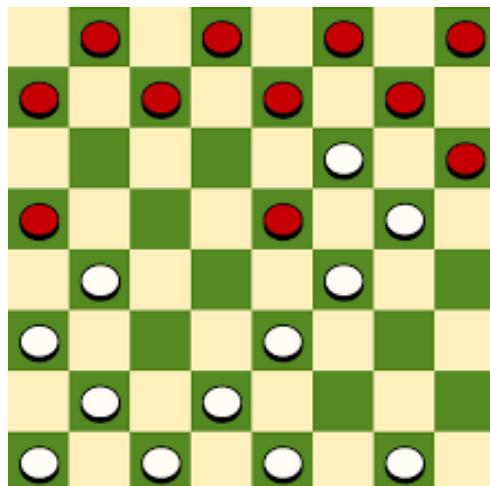
---

- **Objetivo:** seleccionar acciones para maximizar la recompensa futura total
- Las acciones pueden tener consecuencias a largo plazo
- La recompensa puede retrasarse
- Puede ser mejor sacrificar recompensa inmediata para ganar más recompensas a largo plazo
- Ejemplos:
  - Una inversión financiera (puede tardar meses para madurar)
  - Cargar combustible a un helicóptero (podría prevenir un accidente en varias horas)
  - Bloquear los movimientos de un oponente (podría mejorar las posibilidades de ganar dentro de muchos movimientos a partir de ahora)

# Estados

---

- El agente toma sus decisiones en función de una señal del entorno que le llamamos estado
- Una señal de estado que logra retener toda la información relevante se dice que es Markov



# Propiedad de Markov

---

- Asumiendo un número finito de estados y valores de refuerzo, el caso más general, el nuevo estado depende de lo todos lo que ha pasado hasta el tiempo t:

$$Pr \{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\},$$

- Si la señal de estado tiene la propiedad de Markov, sólo depende del estado y la acción en el tiempo t

$$Pr \{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\},$$

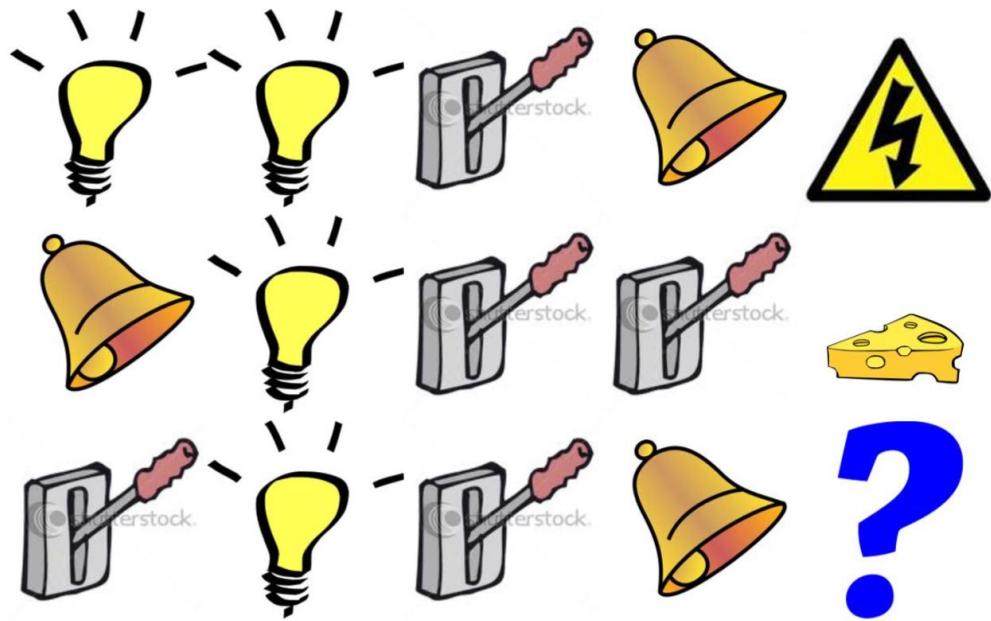
# Propiedad de Markov

---

- El futuro es independiente del pasado dado el presente
- Una vez que el estado es conocido, el pasado puede descartarse

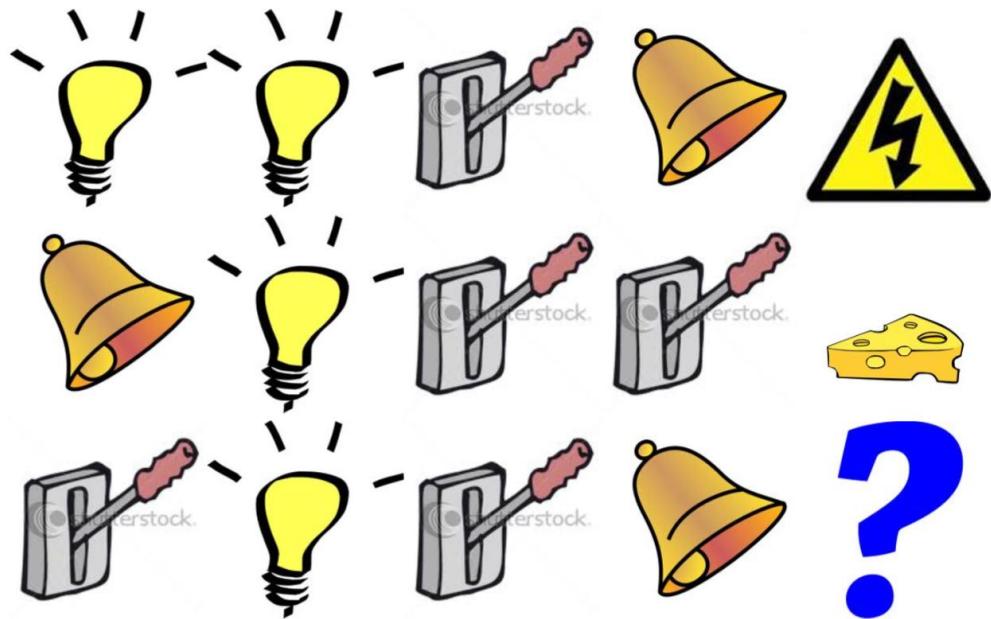
# Ejemplo de la rata

---



- ¿qué pasa si el estado lo definen los últimos 3 elementos de la secuencia?

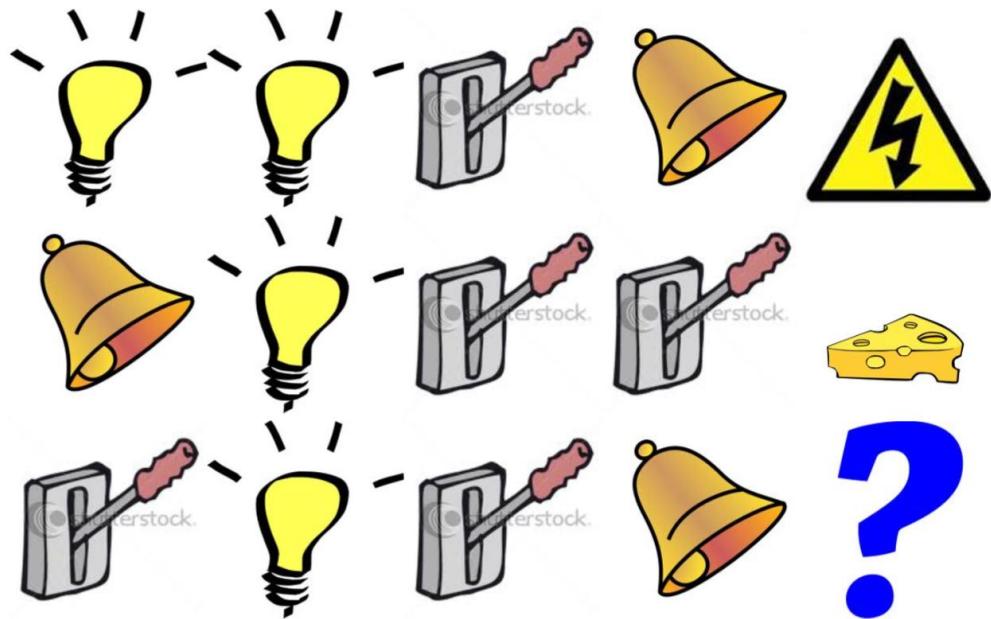
# Ejemplo de la rata



- ¿qué pasa si el estado lo definen el número de focos, palancas y campanas?

# Ejemplo de la rata

---



- ¿qué pasa si el estado lo definen todos los elementos de la secuencia?

# Principales componentes de un agente con aprendizaje por refuerzo

---

- Política: función de comportamiento del agente
- Función de valor: que tan bueno es cada estado/acción
- Modelo: representación del ambiente en el agente

# Política

---

- Es el comportamiento del agente
- Es un mapa de los estados a las acciones
  - Política determinista:  $a=\pi(s)$
  - Política estocástica:  $\pi(a|s)=P\{A_t=a|S_t=s\}$

# Función de valor

---

- Es una predicción de la recompensa futura
- Usado para evaluar que tan buenos o malos son los estados
- Por lo tanto para seleccionar entre posibles acciones

$$V_{\pi}(s) = E_{\pi}\{R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s\}$$

# Modelo

---

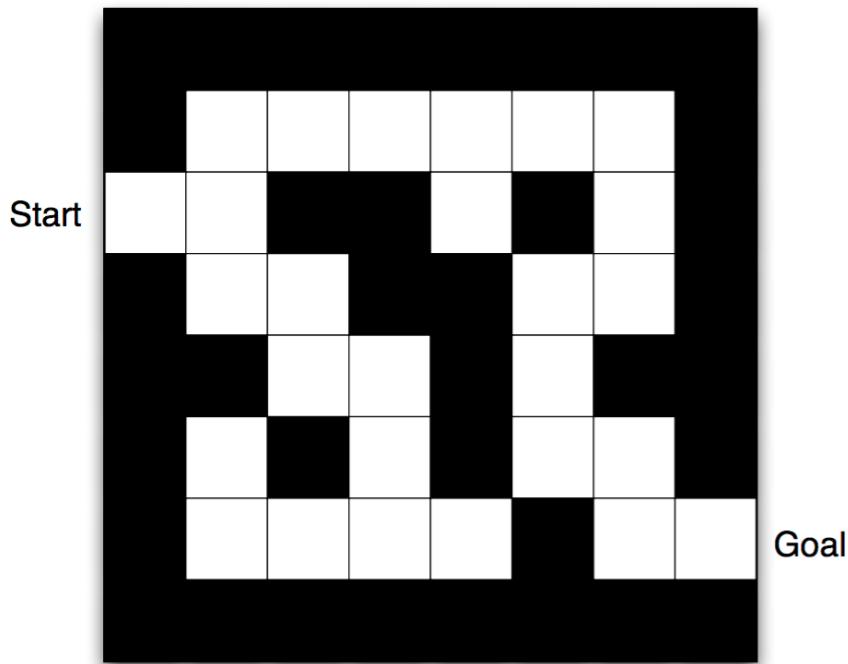
- Predice lo que el ambiente hará a continuación
- P predice el siguiente estado
- R predice la siguiente recompensa (inmediata)

$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

# Ejemplo del laberinto

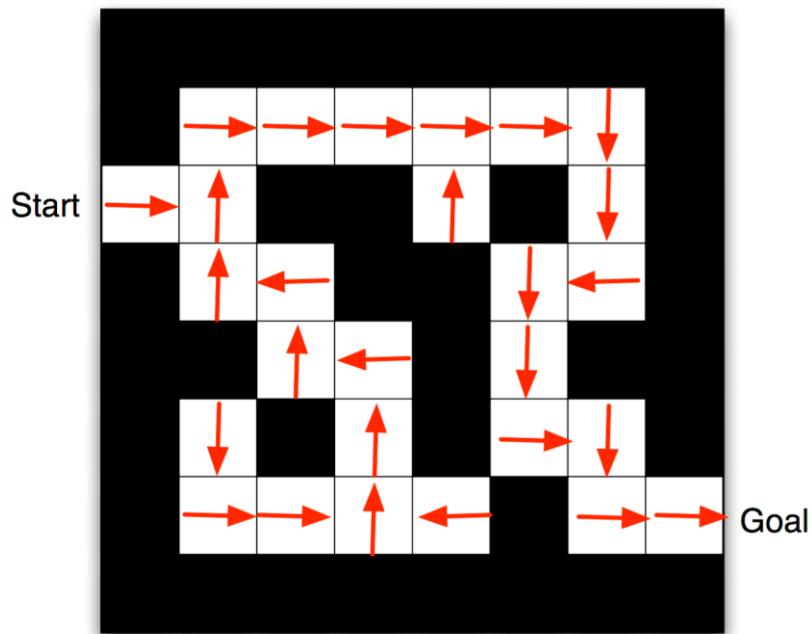
---



- **Recompensa:** -1 por cada paso dado
- **Acciones:** N, S, E, O
- **Estados:** Ubicación del agente

# Ejemplo del laberinto: Política

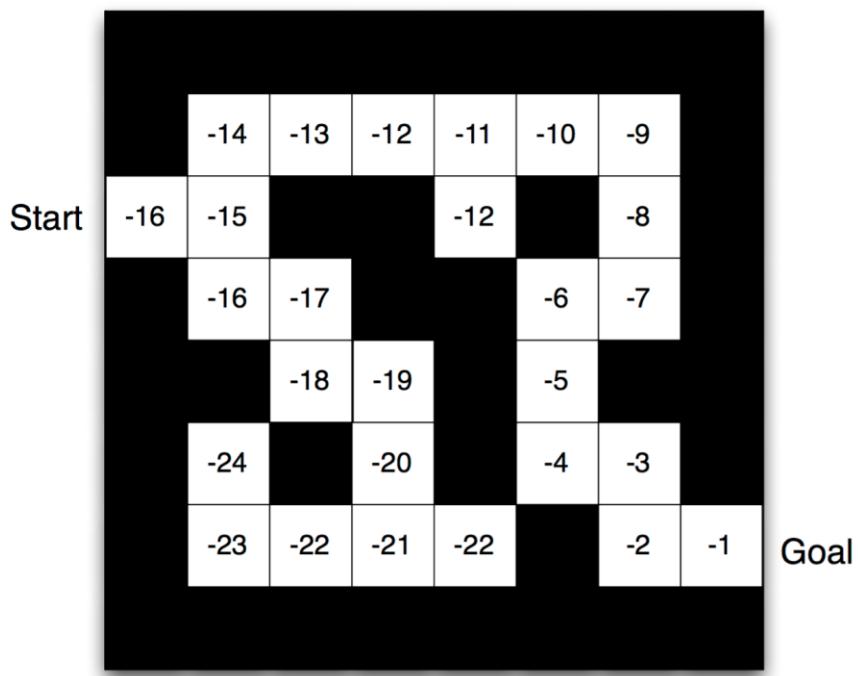
---



- Las flechas representan la política  $\pi(s)$  para cada estado  $s$

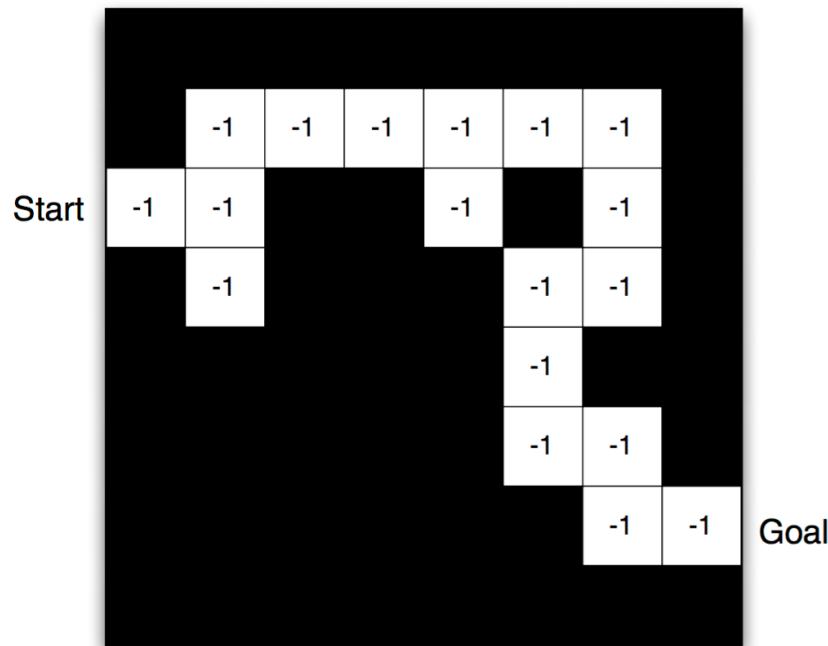
# Ejemplo del laberinto: Función de valor

---



- Número representa el valor  $V_\pi(s)$  de cada estado  $s$

# Ejemplo del laberinto: Modelo



## $\mathcal{P}_{ss'}^a$ Cuadrícula

# $\mathcal{R}_s^a$ Números

- Los agentes pueden tener un modelo del ambiente
  - Dinámica: como las acciones cambian el ambiente
  - Recompensas: cuanta por cada estado
  - El modelo puede ser imperfecto

# Tipo de agentes AR

---

- Basados en Valores
  - Sin política (implícita)
  - Funciones de valor
- Basados en Política
  - Política
  - Sin función de valor
- Actor - Crítico
  - Política
  - Funciones de valor

# Tipo de agentes AR

---

- Libre de modelos
  - Política y/o funciones de valor
  - Sin modelo
- Basados en modelos
  - Política y/o funciones de valor
  - Modelo

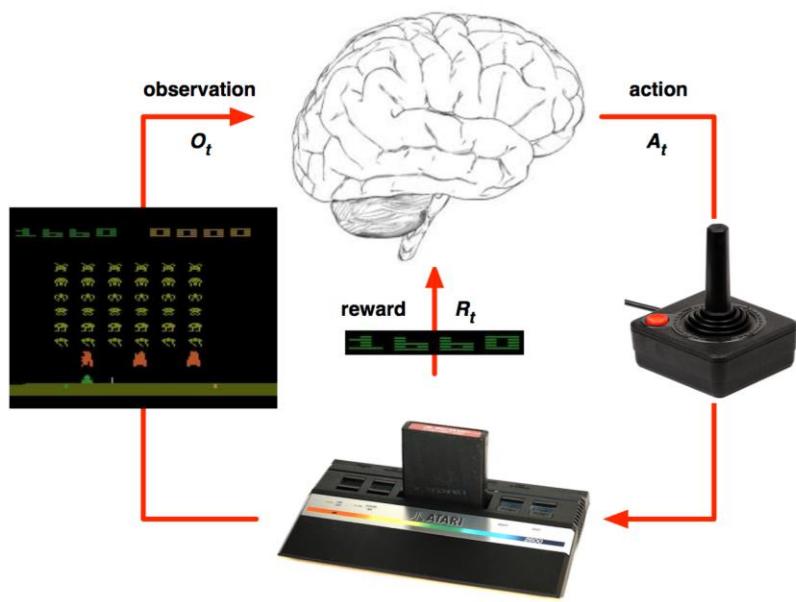
# Aprendizaje y planificación

---

- Dos problemas fundamentales en la secuencia de toma de decisiones
  - Aprendizaje por refuerzo
    - El ambiente es inicialmente desconocido
    - El agente interactúa con el ambiente
    - El agente mejora su política
  - Planificación
    - Un modelo del ambiente es conocido
    - El agente realiza cálculos con su ambiente (sin interacción externa)
    - El agente mejora su política
    - Conocido como deliberación, razonamiento, introspección, reflexión, pensamiento, búsqueda

# Atari: Aprendizaje por refuerzo

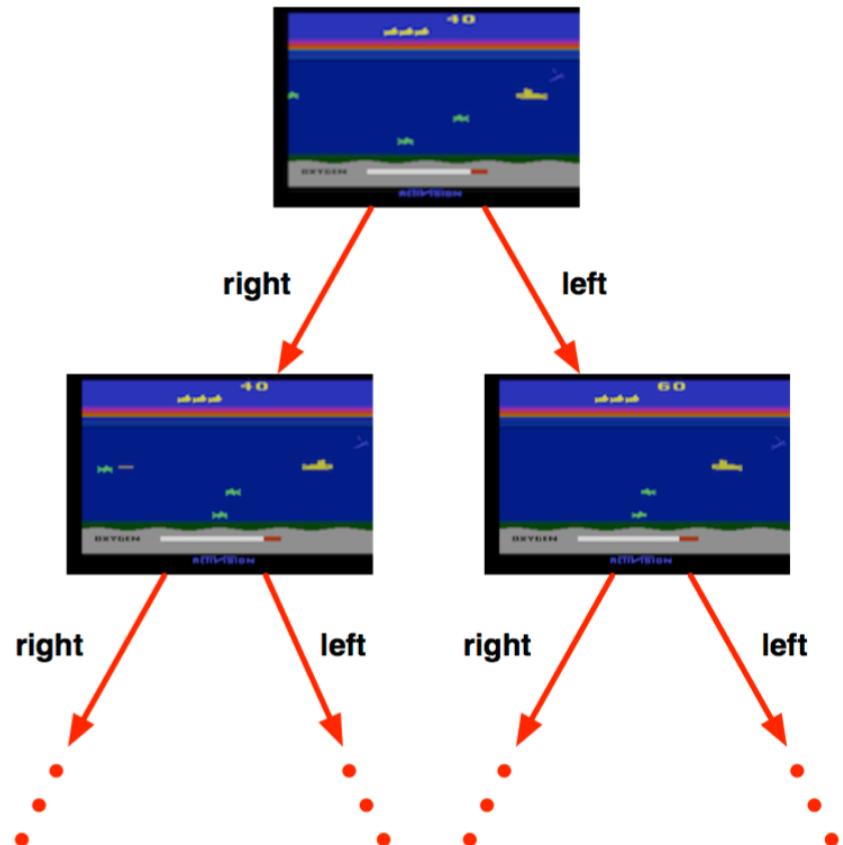
---



- Reglas de los juegos son desconocidos
- Se aprende directamente jugando
- Se seleccionan acciones a realizar con el joystick, se observa la imagen y la puntuación

# Atari: Planificación

- Reglas del juego son conocidas
- Podemos acceder al emulador
  - Modelo perfecto dentro del agente
  - Puedo saber el estado y puntuación que obtendré seleccionando una acción
- Planifico con antelación para obtener política optima
  - Un árbol de búsqueda



# Exploración y explotación

---

- Uno de los retos es el mantener el equilibrio entre la **exploración** y **explotación** en este método.
  - Para maximizar la recompensa, un agente de aprendizaje por refuerzo debe preferir las acciones que ha intentado en el pasado y han sido efectivas en la obtención de recompensa.
  - Para descubrir este tipo de acciones, tiene que intentar acciones que no se ha seleccionado antes.

# Proceso de decisión de Markov

---

- Los procesos de decisión de Markov describen formalmente un entorno para el aprendizaje por refuerzo
  - Donde el ambiente es completamente observable
    - Es decir, el estado actual caracteriza por completo el proceso
- Casi todos los problemas de AR se pueden formalizar como un PDM

# Propiedad de Markov

---

- El futuro es independiente del pasado dado el presente
- Una vez que el estado es conocido, el pasado puede descartarse

# Matriz de transición de estados

---

- Para un estado de Markov  $s$  y sucesor del estado  $s'$ , la probabilidad de transición de estado se define por

$$\mathcal{P}_{ss'} = \mathbb{P} [S_{t+1} = s' | S_t = s]$$

- La matriz de transición de estado  $P$  define las probabilidades de transición de todos los estados  $s$  a todos los estados sucesores  $s'$ ,

$$\mathcal{P} = \text{from} \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

- Donde cada renglón de la matriz suma 1

# Proceso de Markov

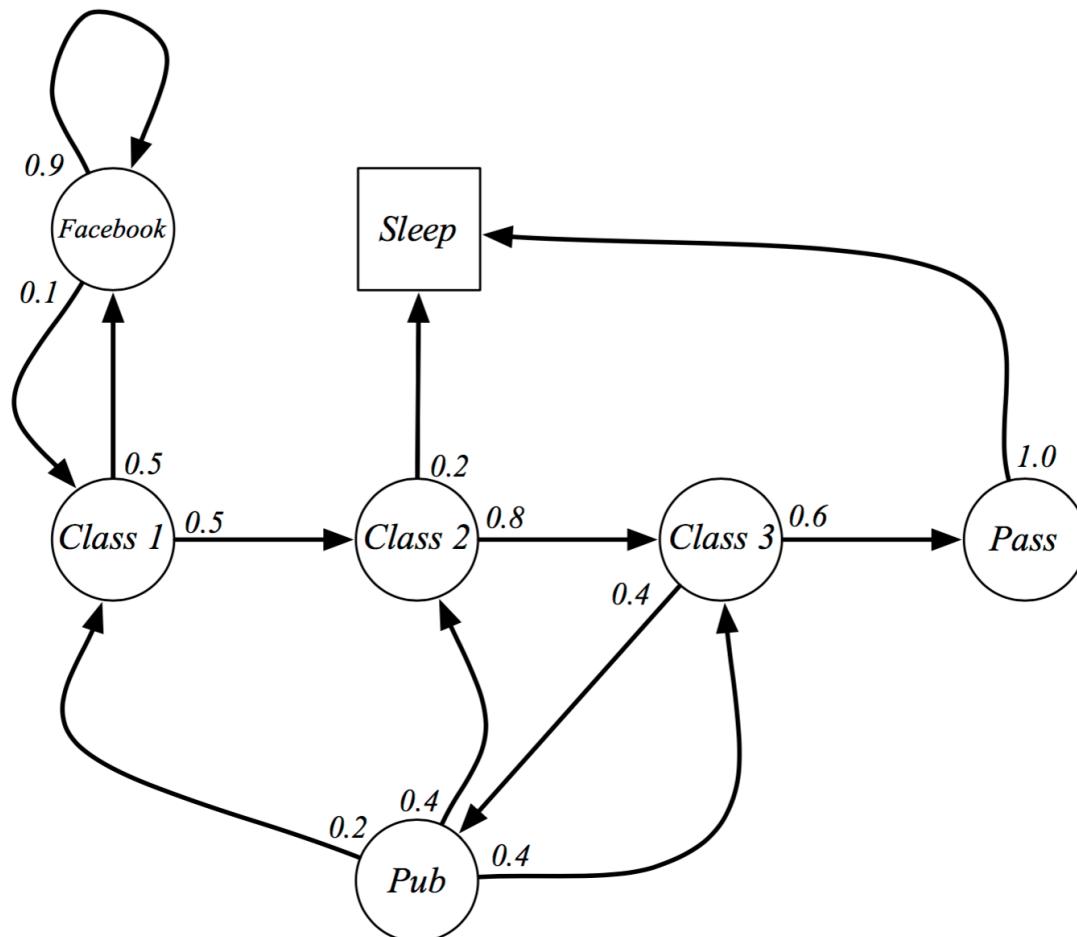
---

- Un proceso de Markov es un proceso aleatorio sin memoria, es decir una secuencia de estados aleatorios  $S_1, S_2, \dots$  con la propiedad de Markov
- Un proceso de Markov (o cadena de Markov) es una tupla  $\langle S, P \rangle$ 
  - $S$  es un conjunto finito de estados
  - $P$  es una matriz de transición de estados

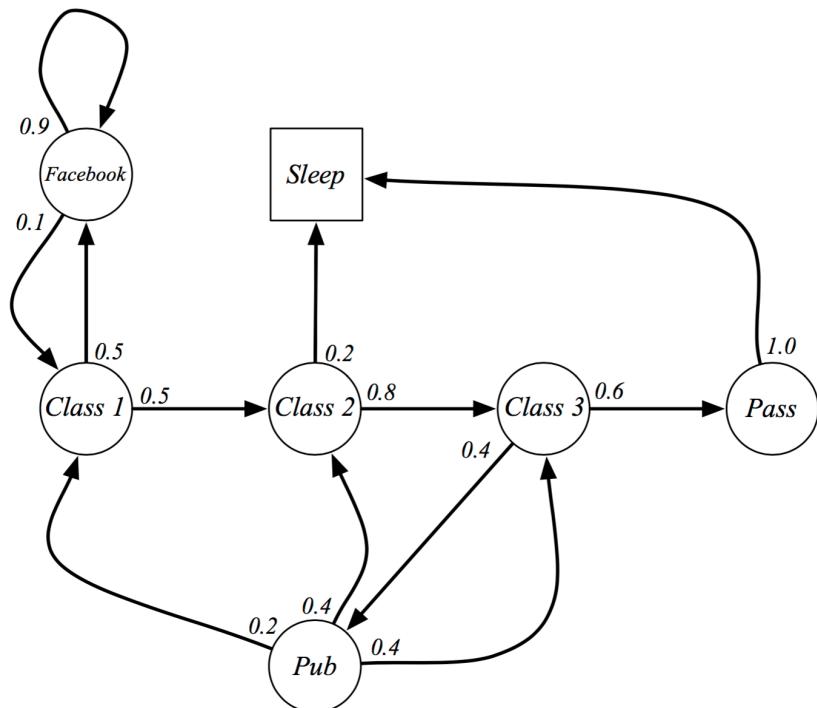
$$\mathcal{P}_{ss'} = \mathbb{P} [S_{t+1} = s' \mid S_t = s]$$

# Ejemplo: Cadena de Markov de un Estudiante

---

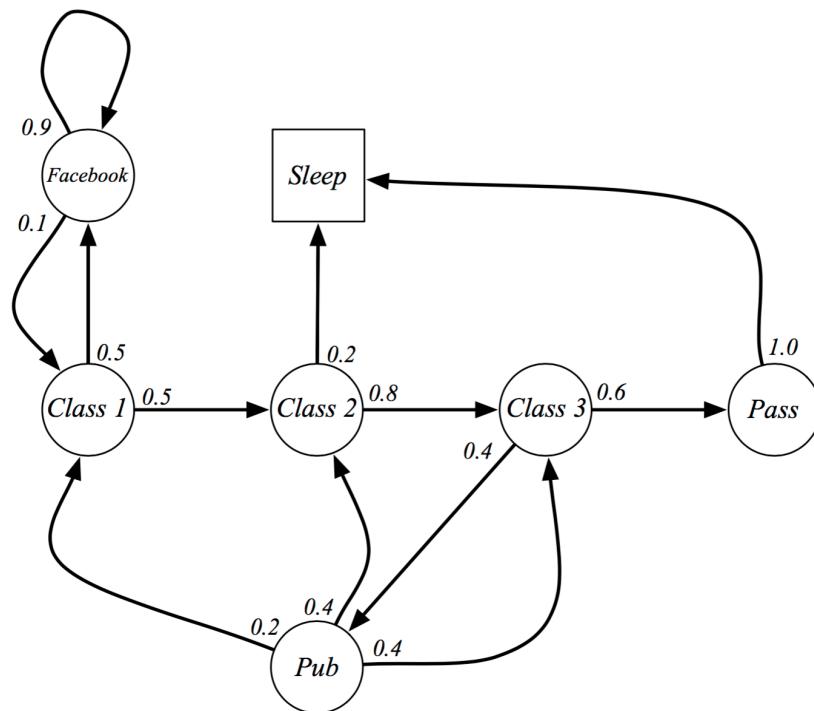


# Ejemplo: Episodios de la cadena de Markov de un Estudiante



- Ejemplo de episodios de la cadena de Markov de un estudiante comenzando por  $S_1 = C1$ 
  - $S_1, S_2, \dots, S_T$
- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

# Ejemplo: Matriz de transición de la cadena de Markov de un Estudiante



$$\mathcal{P} = \begin{bmatrix} C1 & C2 & C3 & Pass & Pub & FB & Sleep \\ C1 & 0.5 & 0.8 & 0.6 & 0.4 & 0.5 & 0.2 \\ C2 & 0.2 & 0.4 & 0.4 & 0.6 & 0.4 & 1.0 \\ C3 & 0.4 & 0.1 & 0.4 & 0.2 & 0.4 & 0.9 \\ Pass & 0.6 & 0.2 & 0.4 & 0.5 & 0.4 & 0.1 \\ Pub & 0.4 & 0.4 & 0.4 & 0.6 & 0.4 & 0.2 \\ FB & 0.1 & 0.1 & 0.1 & 0.1 & 0.5 & 0.9 \\ Sleep & 0.2 & 0.9 & 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}$$

# Proceso de recompensa de Markov

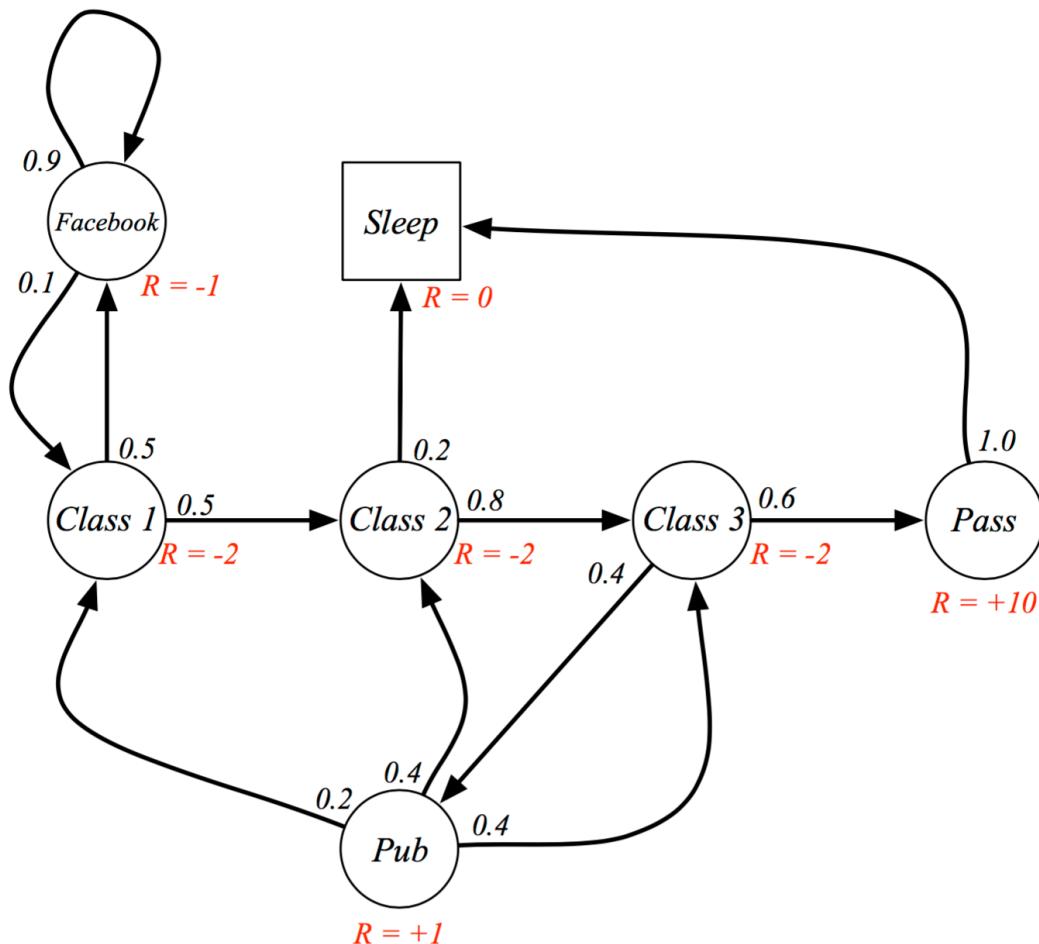
---

- Un proceso de recompensa de Markov es una cadena de Markov con valores
- Un proceso de recompensa de Markov es una tupla  $\langle S, P, R, \gamma \rangle$ 
  - $S$  es un conjunto finito de estados
  - $P$  es una matriz de transición de estados

$$P_{ss'} = \mathbb{P} [S_{t+1} = s' | S_t = s]$$

- $R$  es una función de recompensa,  $R_s = E[R_{t+1} | S_t = s]$
- $\gamma$  es un factor de descuento,  $\gamma \in [0,1]$

# Ejemplo: Cadena de recompensa de Markov de un Estudiante



# Retorno

---

- Si la secuencia de recompensas después del tiempo  $t$  están descritas por:

$$r_{t+1}, r_{t+2}, r_{t+3}, \dots$$

- El retorno esperado está dado por la suma de las recompensas esperadas

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \dots + r_T,$$

- Dónde  $T$  es el tiempo final.
  - La interacción con el ambiente se puede romper en episodios

# Retorno

---

- El retorno esperado para  $T = \infty$  podría ser infinito.
- Para evitar esto se utiliza el retorno descontado
  - se introduce una tasa de descuento  $0 \leq \gamma \leq 1$

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \end{aligned}$$

0 miope (recompensa inmediata)

1 todas las recompensas

# Función de valor

---

- $v(s)$  da el valor a largo plazo del estado  $s$
- La función de valor del estado  $v(s)$  de un PRM es el retorno esperado a partir del estado  $s$

$$v(s) = E[G_t | S_t=s]$$

# Retornos del PRM de un estudiante

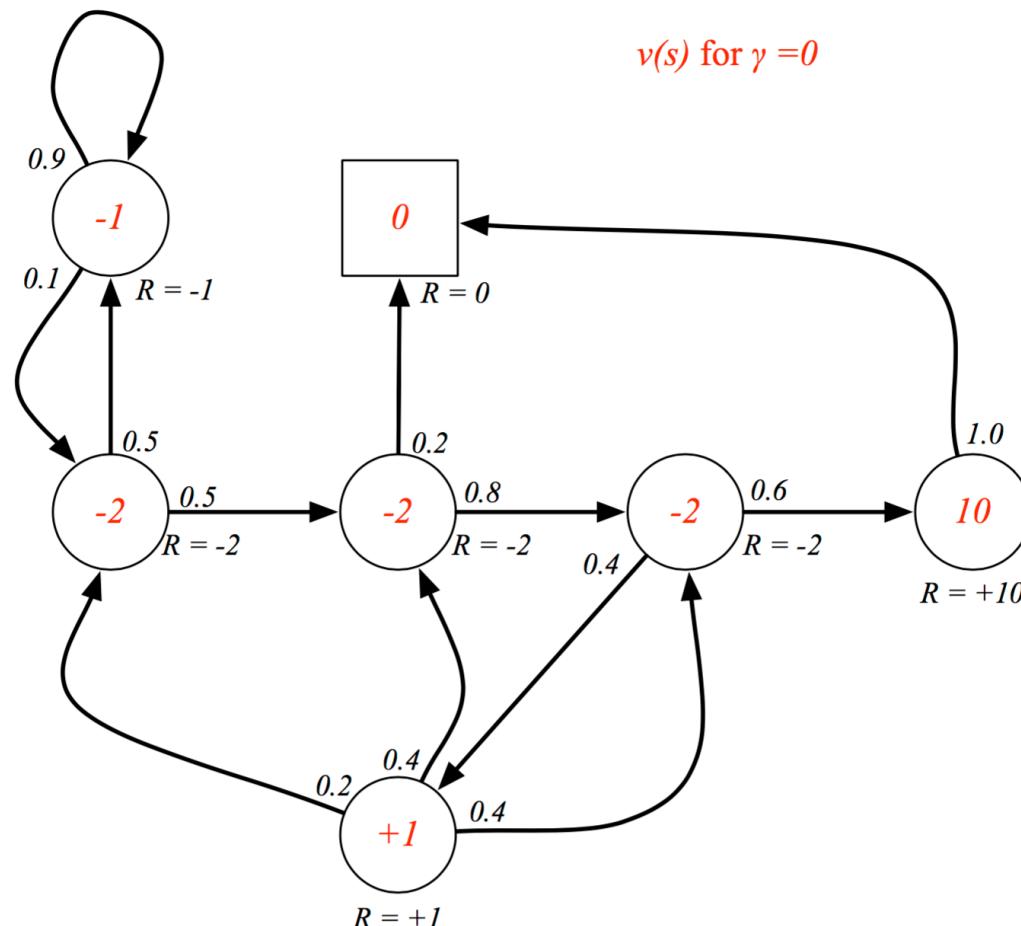
---

- Ejemplo de los retornos del PRM de un estudiante:
- Comenzando desde  $S_1 = C_1$  con  $\gamma=1/2$

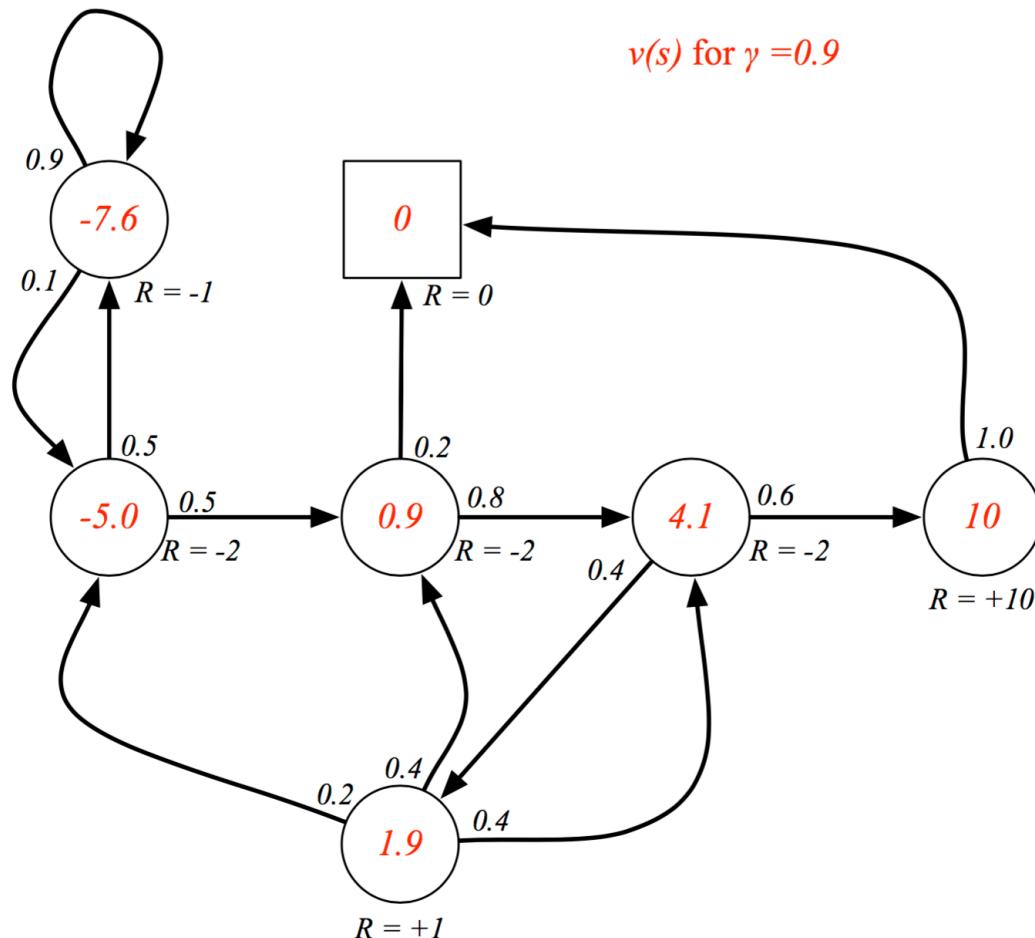
$$G_1 = R_2 + \gamma R_3 + \dots + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	=	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	=	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	=	-3.41
C1 FB FB C1 C2 C3 Pub C1 ...	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	=	-3.20
FB FB FB C1 C2 C3 Pub C2 Sleep			

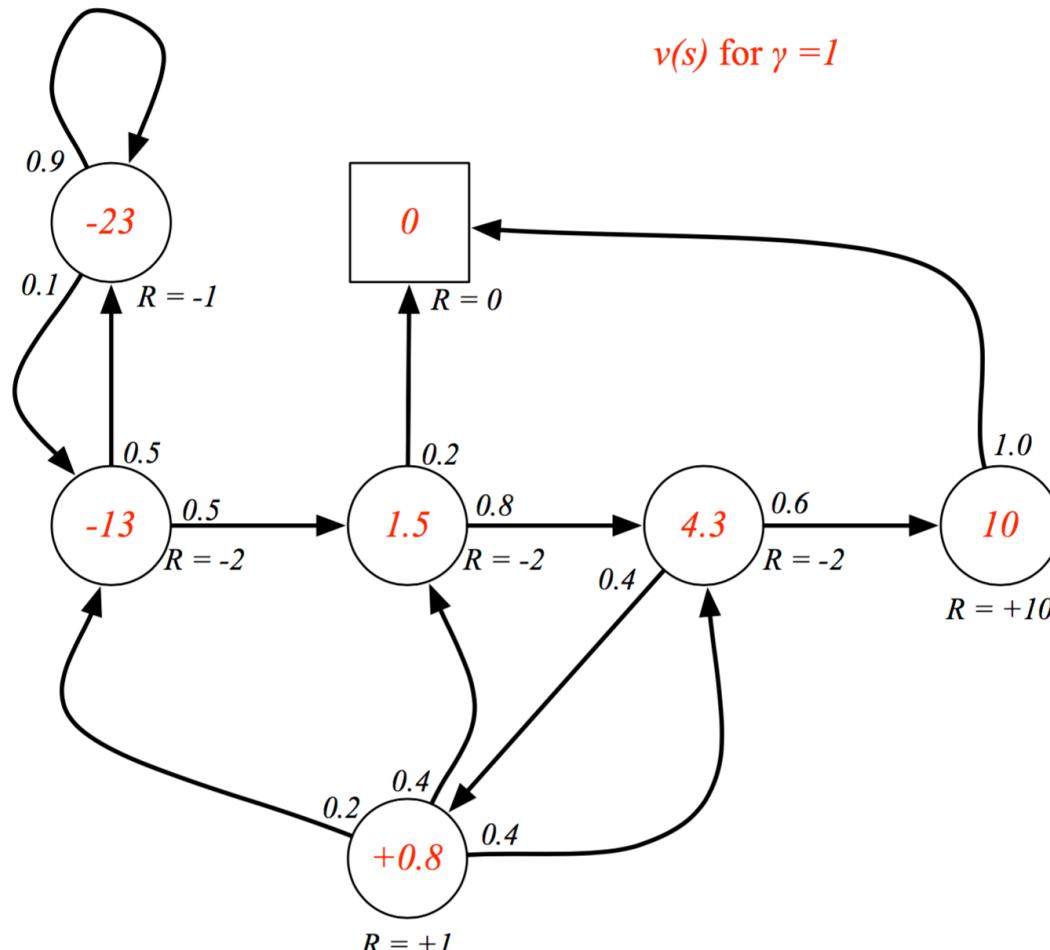
# Ejemplo: Función del valor del estado del PRM de un Estudiante



# Ejemplo: Función del valor del estado del PRM de un Estudiante



# Ejemplo: Función del valor del estado del PRM de un Estudiante



# Ecuación de Bellman para PRMs

---

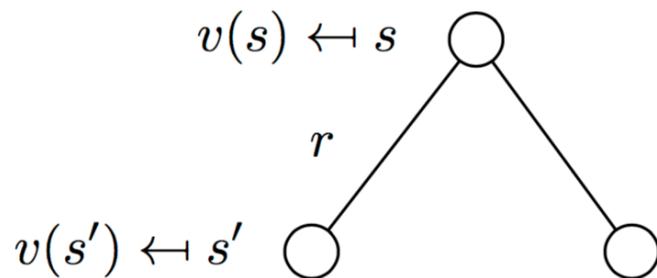
- La función de valor se puede dividir en dos partes:
  - una recompensa inmediata  $R_{t+1}$
  - valor descontado del estado sucesor  $\gamma v(S_{t+1})$

$$\begin{aligned}v(s) &= \mathbb{E}[G_t \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]\end{aligned}$$

# Ecuación de Bellman para PRMs

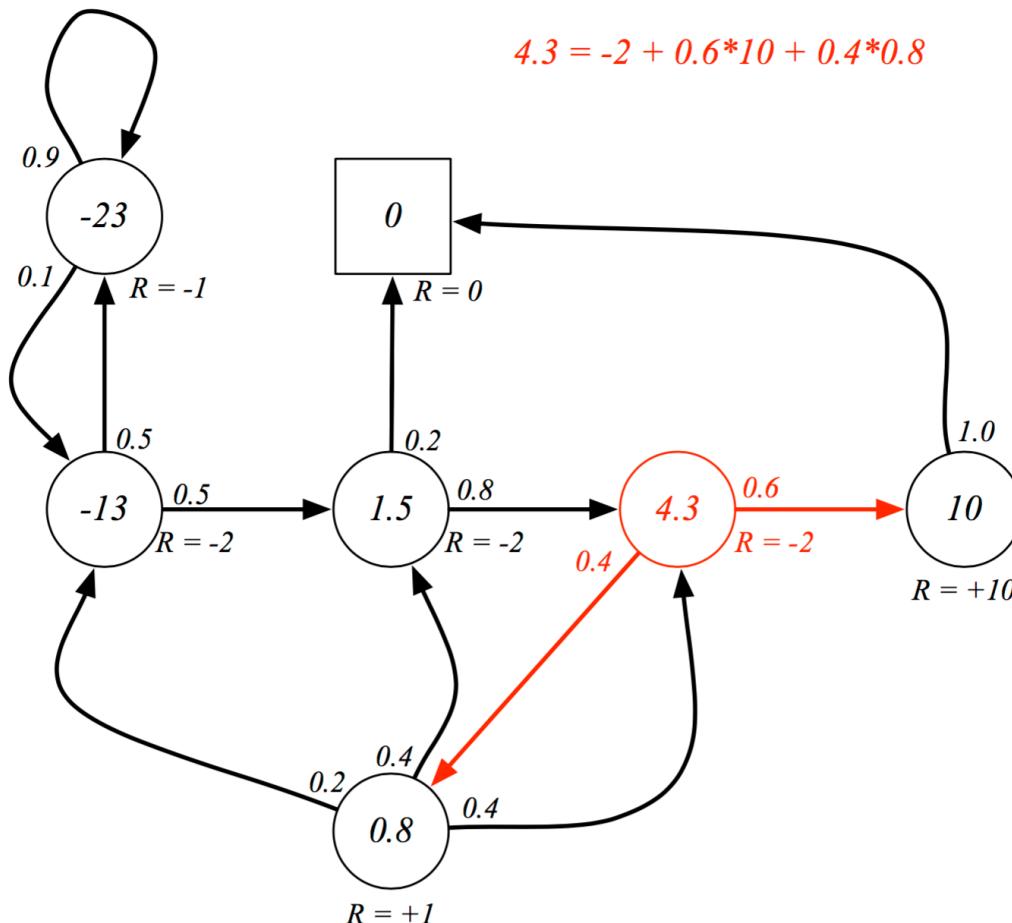
---

$$v(s) = \mathbb{E} [R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]$$



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

# Ejemplo: Ecuación de Bellman para el PRM de un Estudiante



# Ecuación de Bellman en forma matricial

---

- La ecuación de Bellman puede ser expresada concisamente usando matrices

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

- Donde  $v$  es una vector columna con una entrada por estado

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

# Resolviendo la ecuación de Bellman

---

- La ecuación de Bellman es una ecuación lineal
- Puede ser resuelta directamente:

$$v = \mathcal{R} + \gamma \mathcal{P}v$$

$$(I - \gamma \mathcal{P}) v = \mathcal{R}$$

$$v = (I - \gamma \mathcal{P})^{-1} \mathcal{R}$$

- Orden del algoritmo es  $O(n^3)$  para  $n$  estados
- Solución directa sólo es posible para PRMs pequeños
- Hay muchos métodos iterativos para PRMs grandes:
  - Programación dinámica
  - Evaluación Monte-Carlo
  - Aprendizaje de la Diferencia-Temporal

# Proceso de decisión de Markov

---

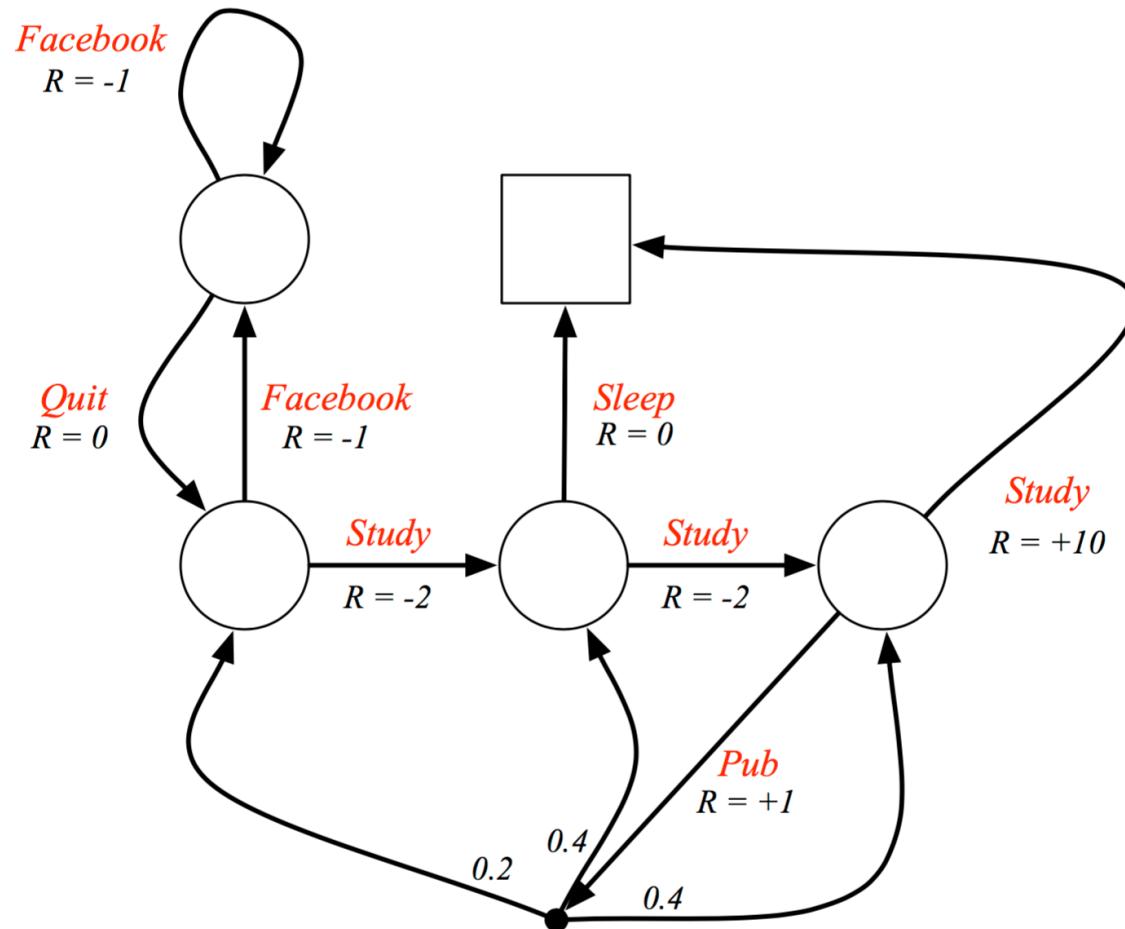
- Un proceso de decisión de Markov es una proceso de recompensa de Markov con decisiones. Es un ambiente en donde todos los estados son Markov.
- Un proceso de decisión de Markov es una tupla  $\langle S, A, P, R, \gamma \rangle$ 
  - $S$  es un conjunto finito de estados
  - $A$  es un conjunto finito de acciones
  - $P$  es una matriz de transición de estados
  - $R$  es una función de recompensa

$$P_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$R_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

- $\gamma$  es un factor de descuento,  $\gamma \in [0,1]$

# Ejemplo: Proceso de decisión de Markov de un Estudiante



# Política

---

- Una política  $\pi$  es una distribución sobre acciones dados estados
$$\pi(a|s) = P[A_t=a | S_t=s]$$
- Una política define completamente el comportamiento de un agente
- Las políticas PDM dependen del estado actual (no la historia)
  - es decir, las políticas son estacionarias (independientes del tiempo)

Sin R, Las recompensas futuras son las que importan



La política  $\pi$  determina el comportamiento del agente

# Función de valor

---

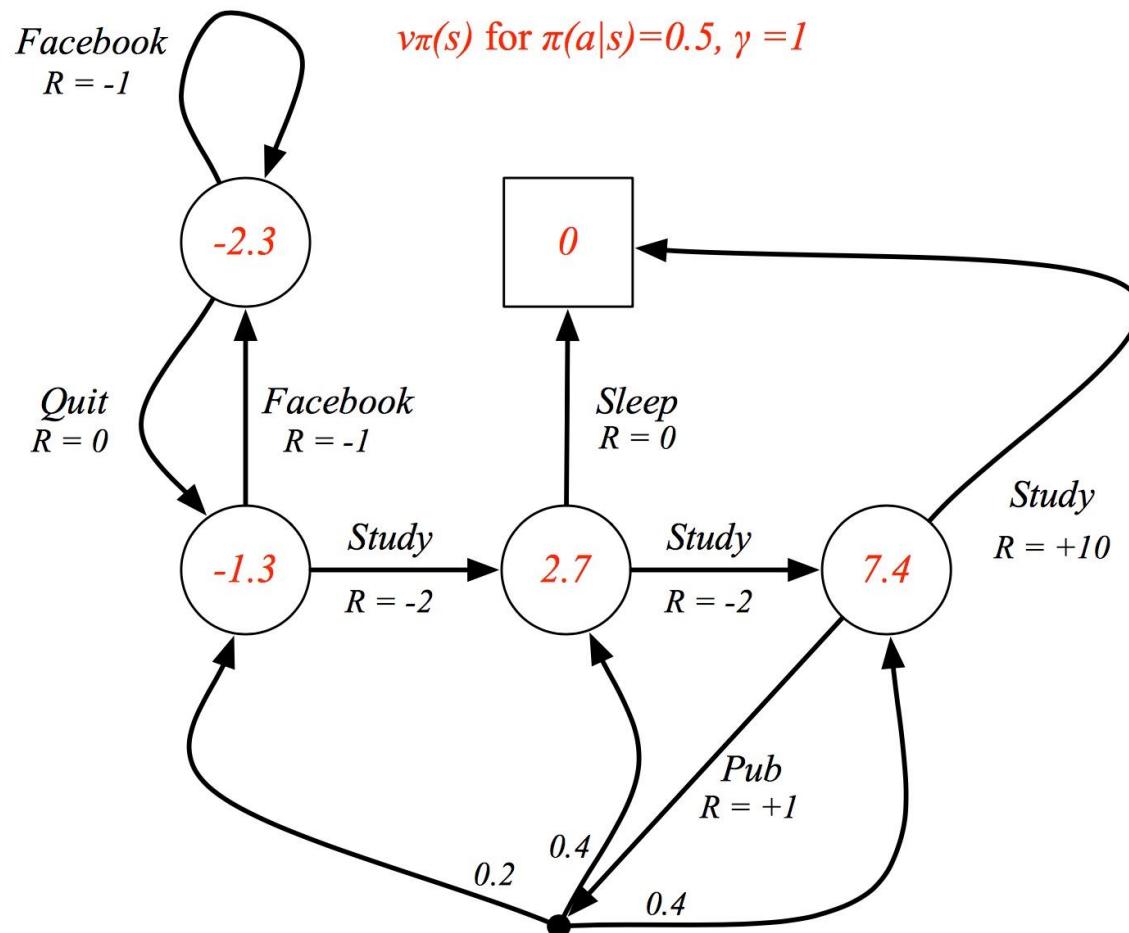
- La función de estado-valor  $v_\pi(s)$  de un PDM es el retorno esperado comenzando desde el estado  $s$  y siguiendo la política  $\pi$

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

- La función de acción-valor  $q_\pi(s)$  de un PDM es el retorno esperado comenzando desde el estado  $s$ , tomando la acción  $a$  y siguiendo la política  $\pi$

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

# Ejemplo: Función estado - valor de un PDM de un Estudiante



# Ecuación de esperanza de Bellman

---

- La función de estado-valor también se puede dividir en dos partes:
  - una recompensa inmediata  $R_{t+1}$
  - valor descontado del estado sucesor  $\gamma v_\pi(S_{t+1})$

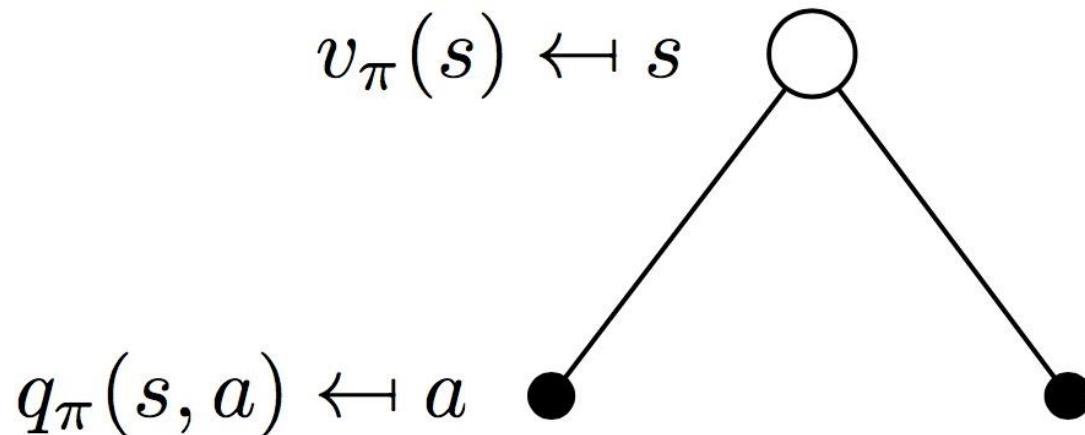
$$v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

- La función de acción-valor puede dividirse de la misma forma

$$q_\pi(s, a) = \mathbb{E}_\pi [R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

# Ecuación de esperanza de Bellman para $v^\pi$

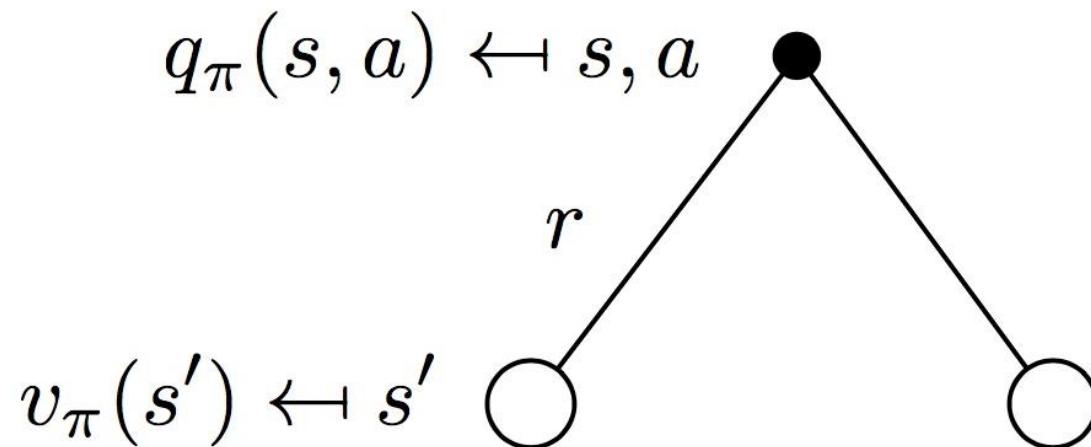
---



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$$

# Ecuación de esperanza de Bellman para $q^\pi$

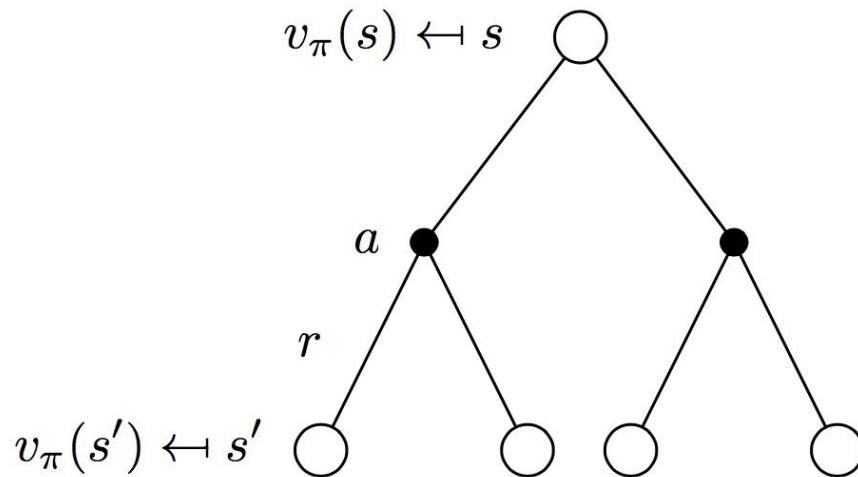
---



$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

# Ecuación de esperanza de Bellman para $v^\pi$

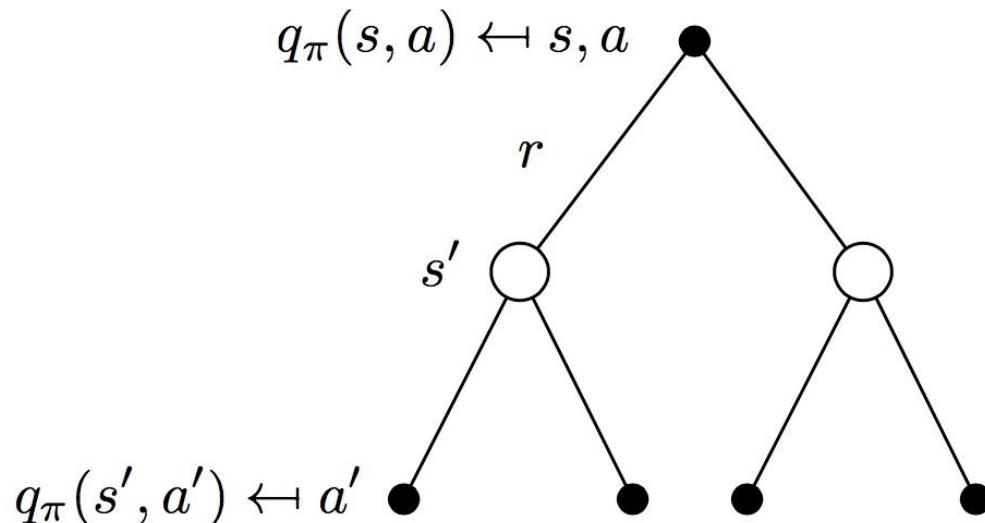
---



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

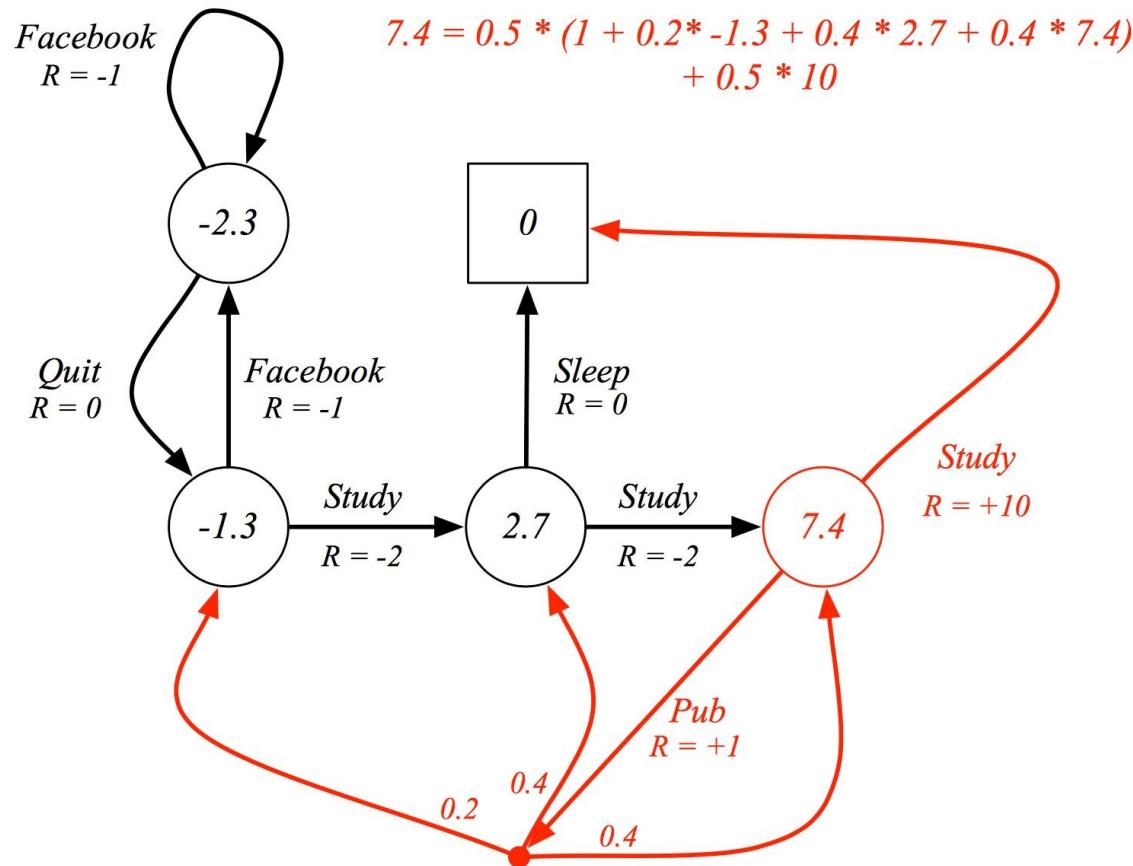
# Ecuación de esperanza de Bellman para $q^\pi$

---



$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a')$$

# Ejemplo: Ecuación de Bellman para el PDM de un Estudiante



# Función de valor óptima

---

- La función de estado-valor óptima  $v_*(s)$  es la función de estado-valor máxima sobre todas las políticas

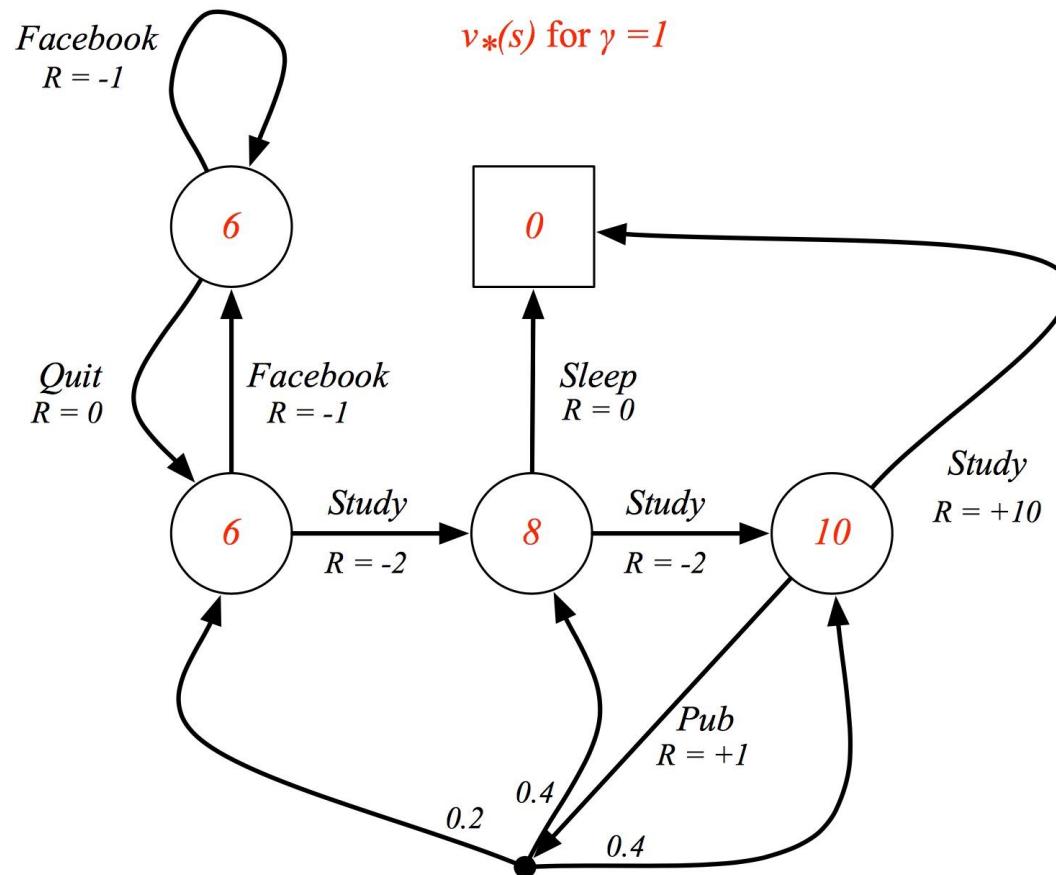
$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

- La función de acción-valor óptima  $q_*(s)$  es la función de acción-valor máxima sobre todas las políticas

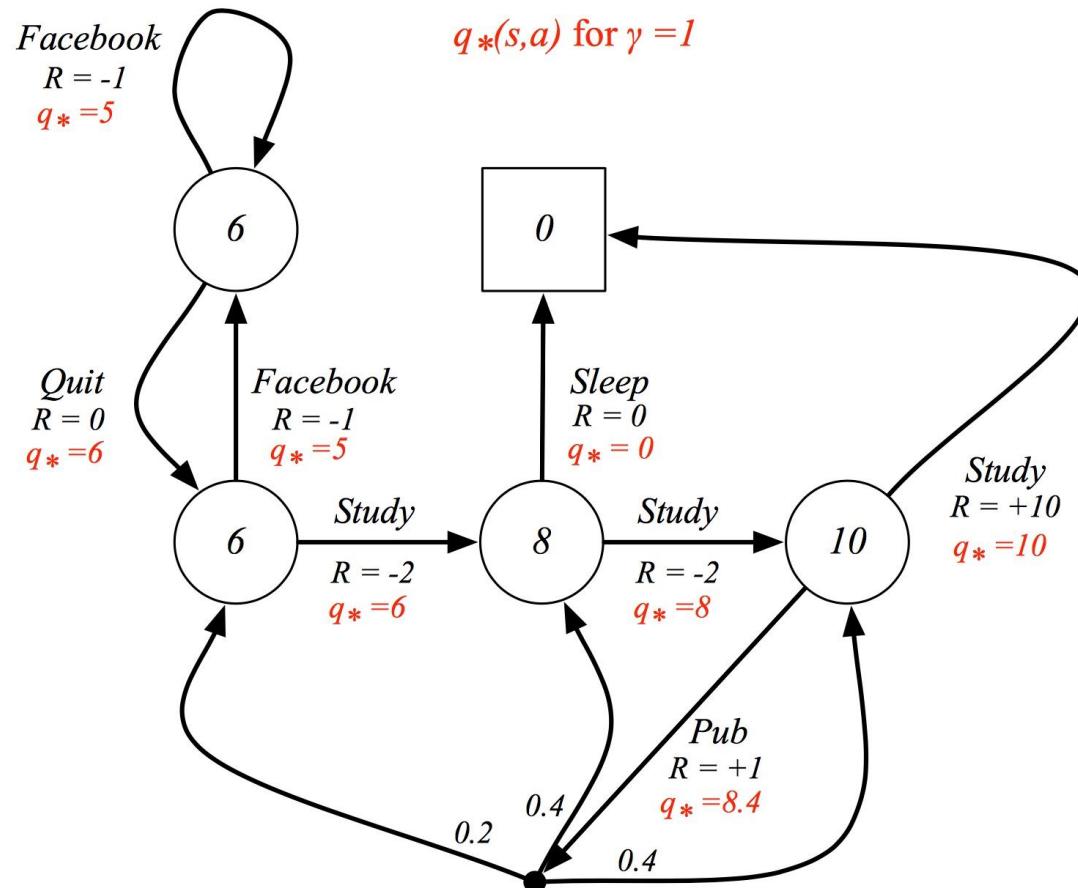
$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$

- La función de valor óptima especifica el mejor desempeño posible en el PDM
- Un PDM es “resuelto” cuando conocemos la función de valor óptima

# Ejemplo: Función de valor óptima para el PDM de un Estudiante



# Ejemplo: Función de valor óptima para el PDM de un Estudiante



# Política óptima

---

- Definimos un orden parcial sobre las políticas

$$\pi \geq \pi' \text{ si } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

- Para cualquier PDM
  - Existe una política óptima  $\pi_*$  que es mejor o igual a todas las otras políticas  $\pi_* \geq \pi, \forall \pi$
  - Todas las políticas óptimas obtienen la función de estado-valor óptima  $v_{\pi_*}(s) = v_*(s)$
  - Todas las políticas óptimas obtienen la función de acción-valor óptima  $q_{\pi_*}(s, a) = q_*(s, a)$

# Encontrando la política óptima

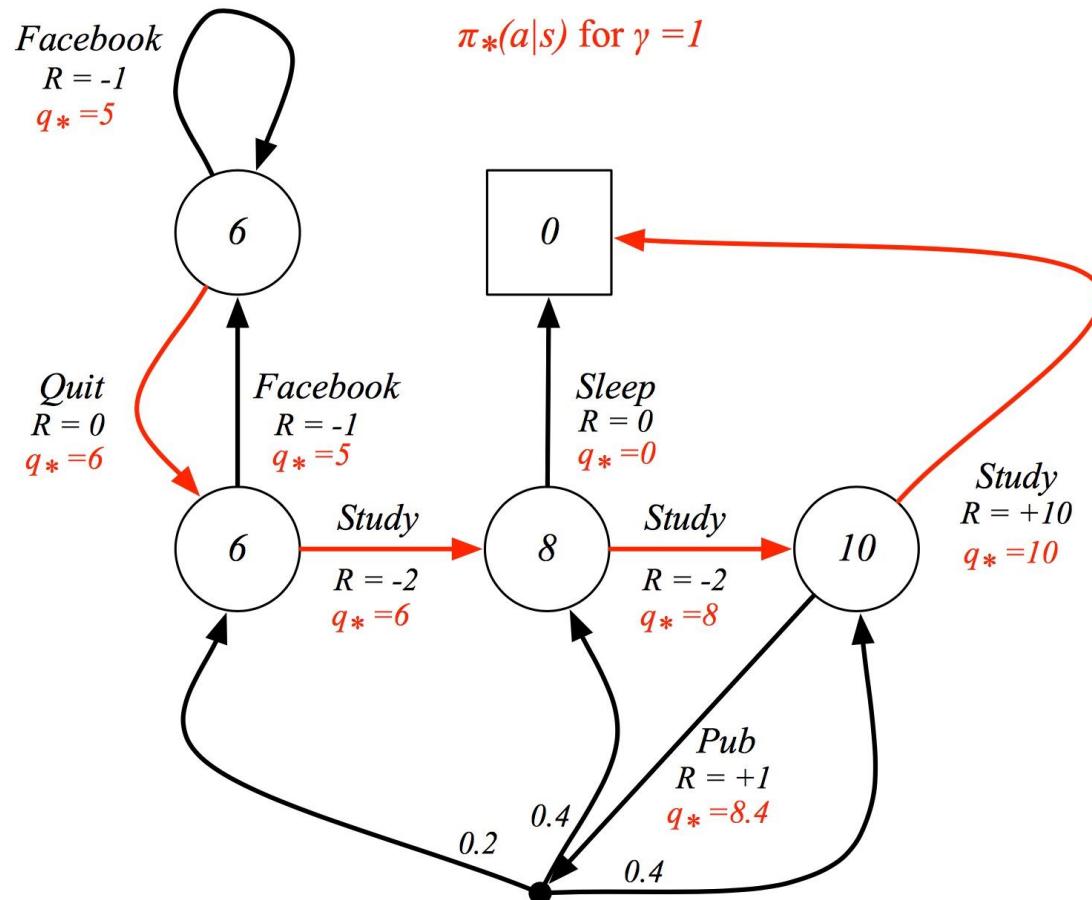
---

- Una política óptima puede ser encontrada maximizando sobre  $q_*(s, a)$

$$\pi_*(a|s) = \begin{cases} 1 & \text{si } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}} q_*(s, a) \\ 0 & \text{de otra forma} \end{cases}$$

- Siempre hay una política óptima determinista para cualquier PDM
- Si conocemos  $q_*(s, a)$ , de inmediato tenemos la política óptima

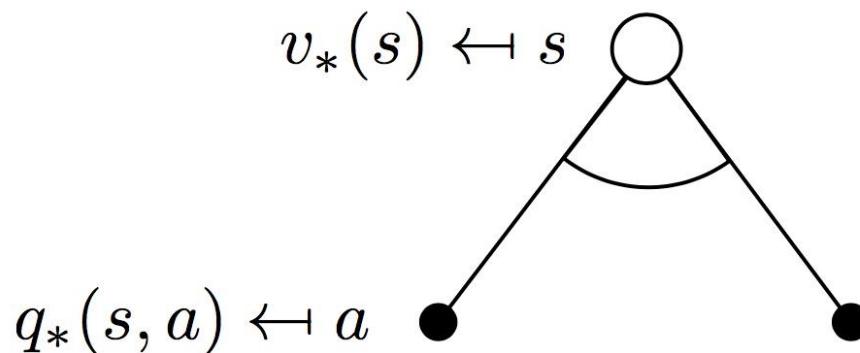
# Ejemplo: Política óptima para el PDM de un Estudiante



# Ecuación de optimalidad de Bellman para $v_*$

---

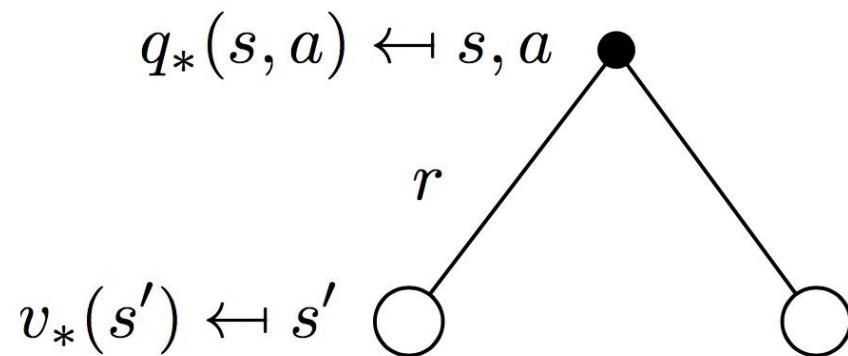
- Las funciones de estado-valor óptimas están recursivamente relacionadas por la ecuación de optimalidad de Bellman



$$v_*(s) = \max_a q_*(s, a)$$

# Ecuación de optimalidad de Bellman para $q_*$

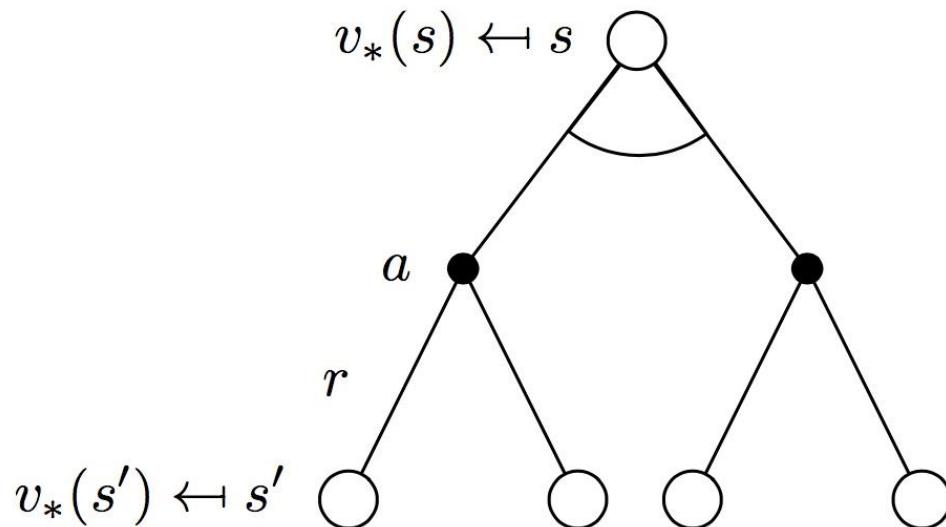
---



$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

# Ecuación de optimalidad de Bellman para $v_*$

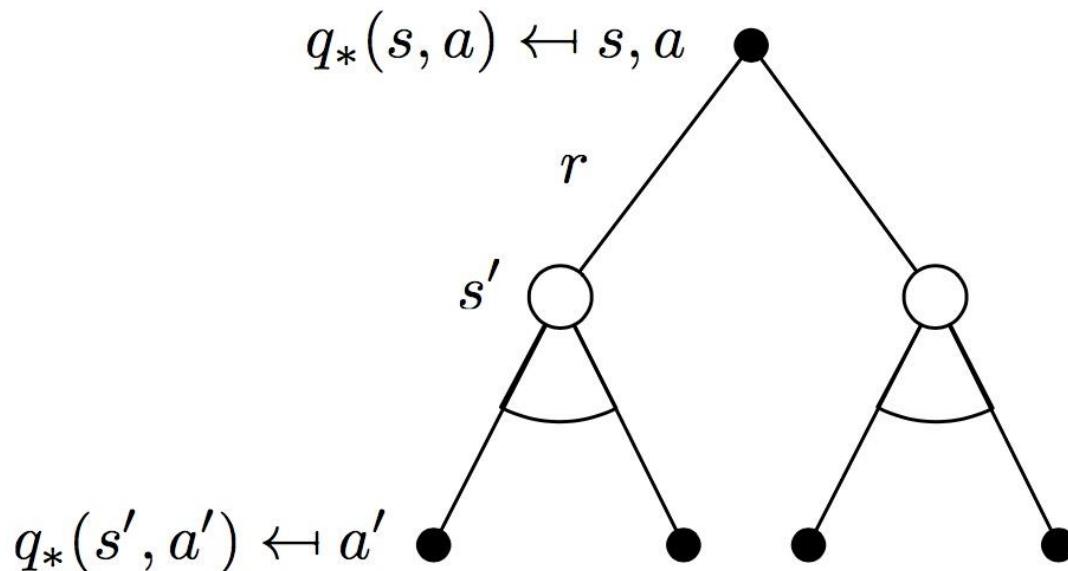
---



$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

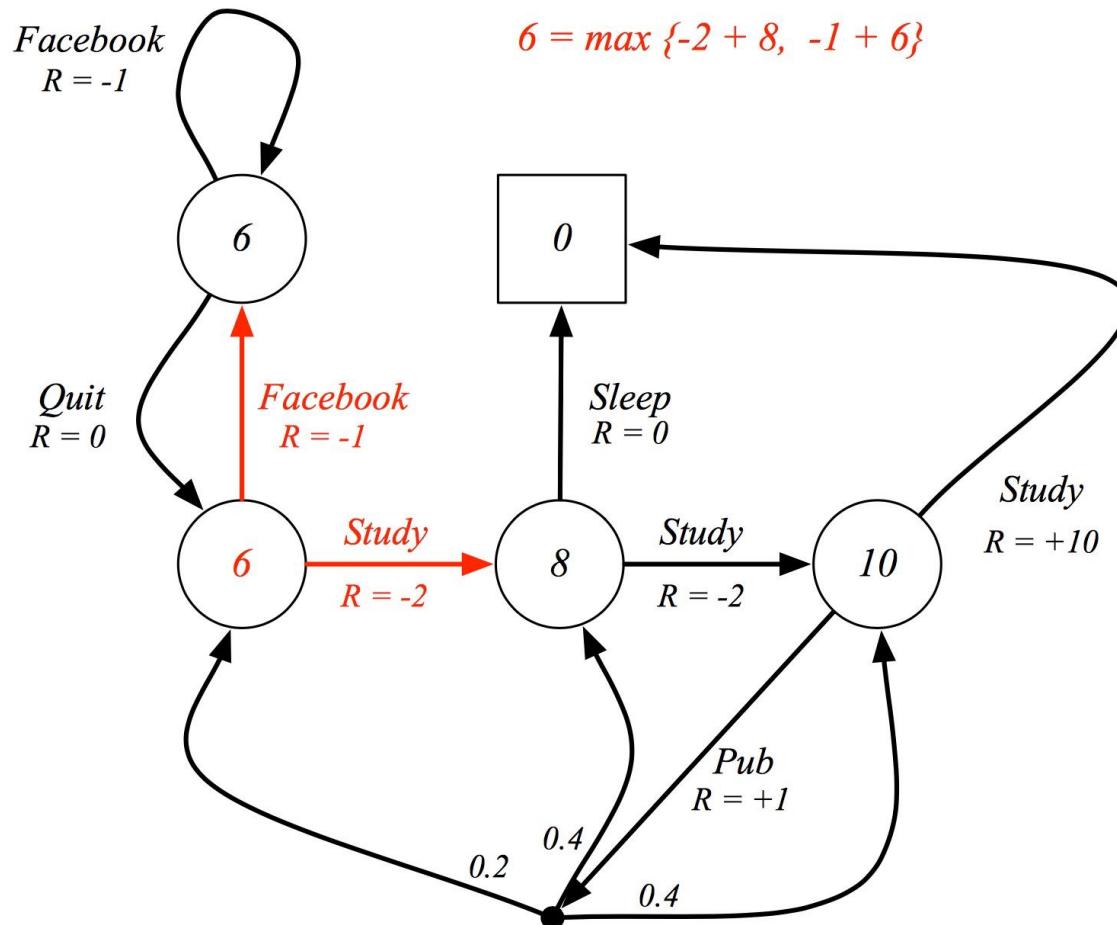
# Ecuación de optimalidad de Bellman para $q_*$

---



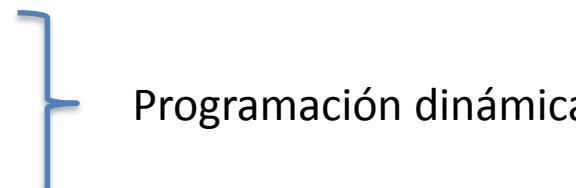
$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a')$$

# Ejemplo: Ecuación de optimalidad de Bellman para el PDM de un Estudiante



# Resolviendo la ecuación de optimalidad de Bellman

---

- La ecuación de optimalidad de Bellman no es lineal
  - No hay solución cerrada (en general)
  - Hay métodos de solución iterativos
    - Iteración de valor
    - Iteración de política
    - Aprendizaje Q
    - Sarsa
- 
- Programación dinámica

# Programación dinámica

---

- Programación: optimización de una política
- Dinámica: problema con un componente secuencial o temporal
  - Un método para resolver problemas complejos dividiéndolos en subproblemas
    - Resolver los subproblemas
    - Combinar las soluciones de los subproblemas

# Requerimientos para la programación dinámica

---

- La programación Dinámica es un método de solución general para problemas que tienen dos propiedades:
  - Subestructura óptima
    - Aplica el principio de optimalidad (cualkiera que sea el estado inicial, las decisiones restantes deben ser óptimas en relación con el estado que le sigue a la primera decisión)
    - Solución óptima se puede dividir en subproblemas (divide y vencerás)
  - Subproblemas superpuestos
    - Subproblemas se repiten muchas veces
    - Soluciones se pueden almacenar en caché y volver a utilizar

# Requerimientos para la programación dinámica

---

- Los Procesos de Decisión de Markov satisfacen ambas propiedades
  - Ecuación de Bellman provee una descomposición recursiva
  - La función de valor almacena y reutiliza soluciones

# Planificación con programación dinámica

---

- La programación dinámica asume pleno conocimiento del PDM
- Se utiliza para la planificación en un PDM
- Para predicción:
  - Entrada: PDM  $\langle S, A, P, R, \gamma \rangle$  y política  $\pi$
  - o: PRM  $\langle S, P^\pi, R^\pi, \gamma \rangle$
  - Salida: valor de la función  $v_\pi$
- o para control:
  - Entrada: MDP  $\langle S, A, P, R, \gamma \rangle$
  - Salida: función de valor óptimo  $v_*$
  - y: política óptima  $\pi_*$

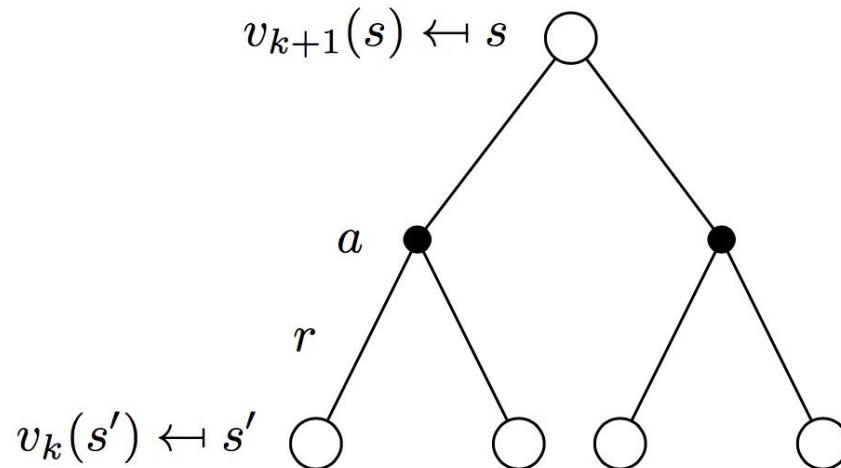
# Evaluación iterativa de la política

---

- Problema: evaluar una política determinada  $\pi$
- Solución: aplicación iterativa del respaldo de la esperanza de Bellman
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_\pi$
- El uso de respaldos síncronos,
- En cada iteración  $k + 1$ 
  - Para todos los estados  $s \in S$
  - actualizar  $v_{k+1}(s)$  a partir de  $v_k(s')$
  - donde  $s'$  es un estado sucesor de  $s$

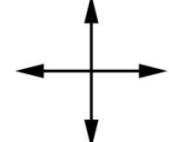
# Evaluación iterativa de la política

---



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$
$$\mathbf{v}^{k+1} = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \mathbf{v}^k$$

# Evaluando una política aleatoria en el pequeño mundo cuadriculado



acciones

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

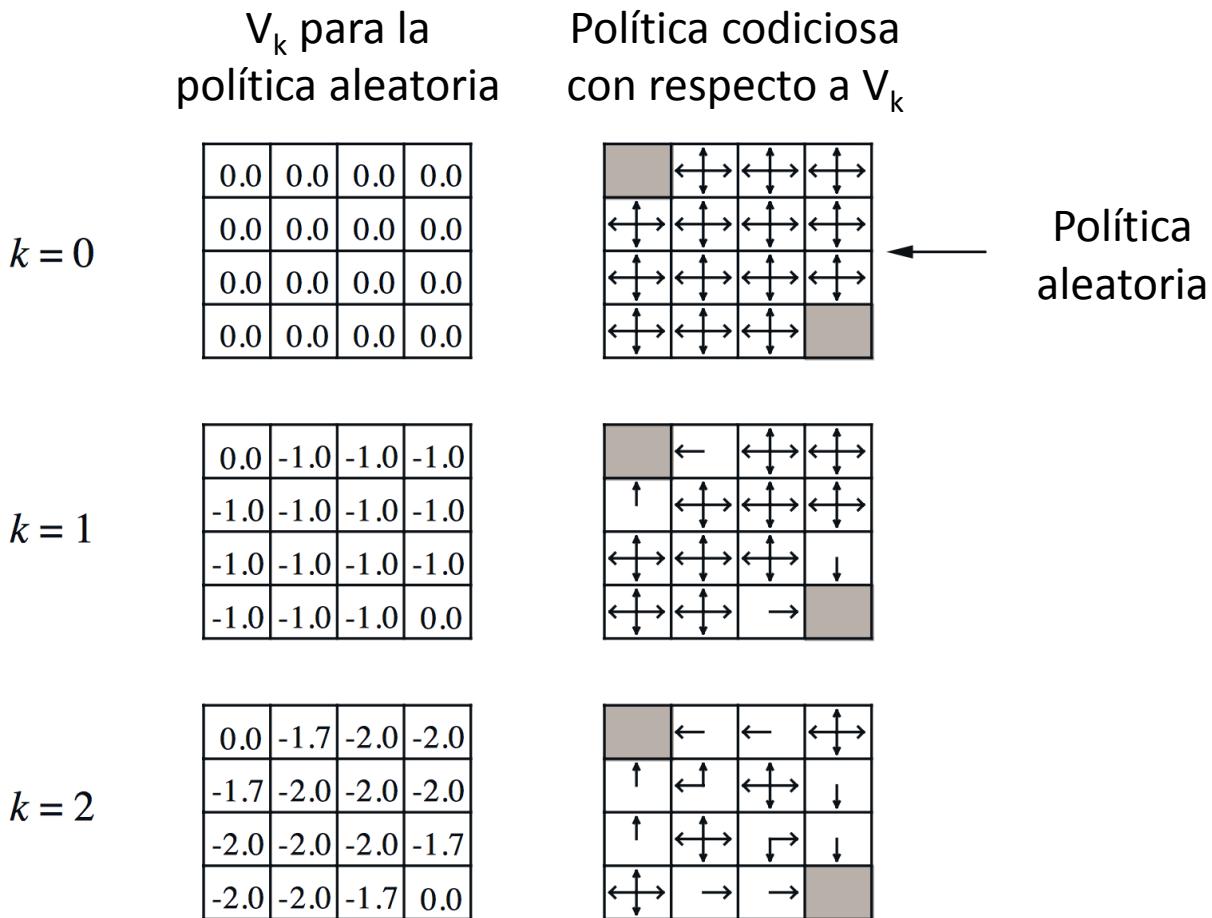
$$r = -1$$

En todas las  
transiciones

- PDM episódico sin descuento ( $\gamma = 1$ )
- Estados no terminales 1, ..., 14
- Un estado terminal (cuadrados grises)
- Acciones que se salen de la cuadrícula no cambian el estado
- Recompensa es -1 hasta que se llega al estado terminal
- La política sigue una política aleatoria uniforme

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

# Evaluación iterativa de la política en el pequeño mundo cuadriculado



# Evaluación iterativa de la política en el pequeño mundo cuadriculado

$k = 3$

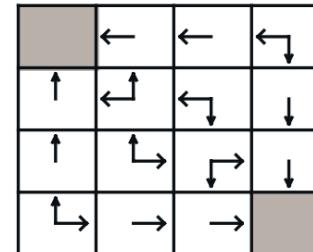
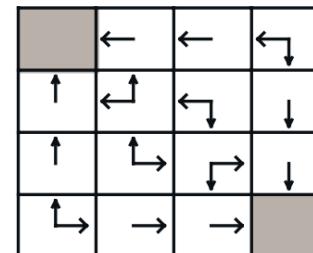
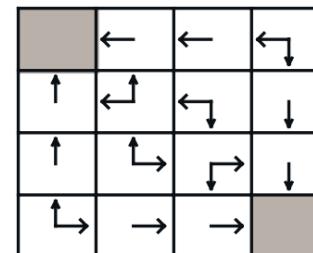
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



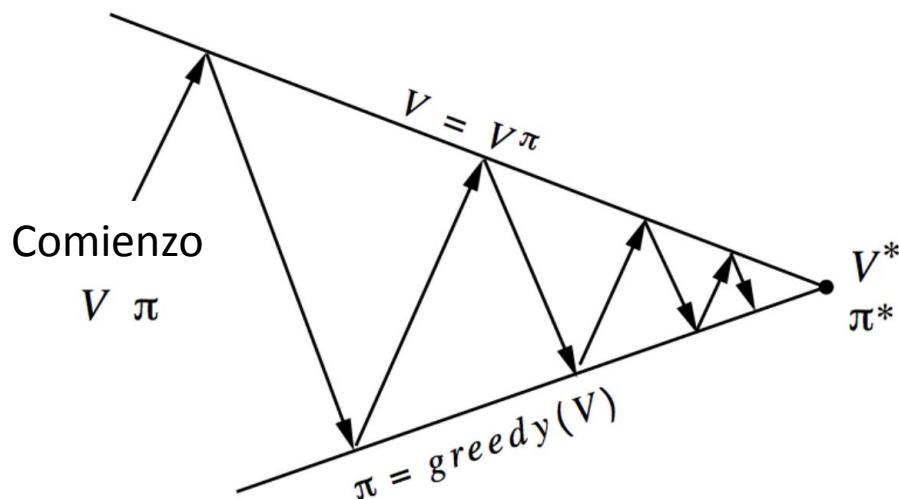
Política  
óptima

# Cómo mejoramos la política

---

- Dada la política  $\pi$ 
  - **Evaluar** la política  $\pi$ 
$$v_\pi(s) = E[R_{t+1} + \gamma R_{t+2} + \dots | S_t = s]$$
  - **Mejorar** la política actuando codiciosamente (greedily )con respecto a  $v_\pi$ 
$$\pi' = \text{greedy}(v_\pi)$$
- En el Pequeño Mundo Cuadriculado la política mejorada era óptima,  $\pi' = \pi^*$
- En general, se necesitan más iteraciones de mejora / evaluación
- Pero este proceso de **iteración de política** siempre converge a  $\pi^*$

# Iteración de la política

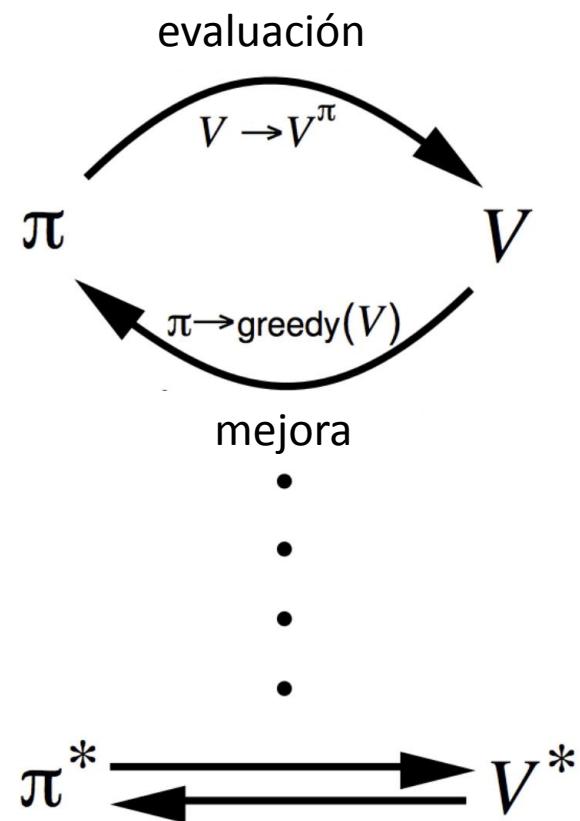


Estimación de la política Estimar  $v_\pi$

Evaluación iterativa de la política

Mejora de la política Generar  $\pi' \geq \pi$

Mejora codiciosa de la política



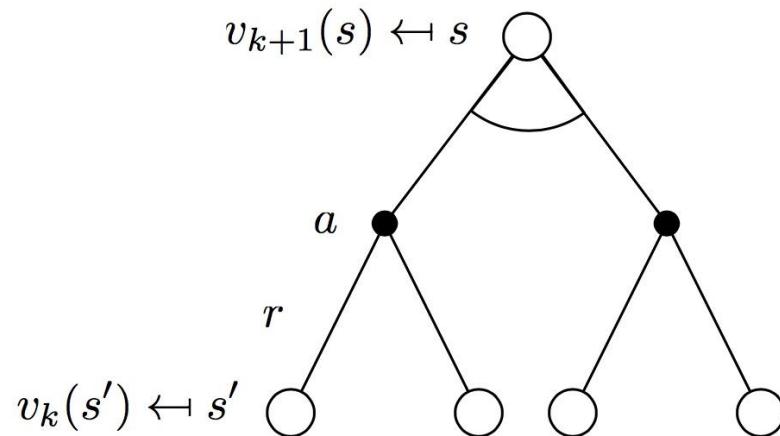
# Iteración de valor

---

- Problema: evaluar la política óptima  $\pi$
- Solución: aplicación iterativa del respaldo de la optimalidad de Bellman
- $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_*$
- El uso de respaldos síncronos,
- En cada iteración  $k + 1$ 
  - Para todos los estados  $s \in S$
  - actualizar  $v_{k+1}(s)$  a partir de  $v_k(s')$
- Funciones de valor intermedias pueden no corresponder a ninguna política

# Iteración de valor

---



$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left( \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a \mathbf{v}_k$$

# Ejemplo: La trayectoria más corta

g				

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$v_1$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

$v_2$

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

$v_3$

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

$v_4$

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

$v_5$

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

$v_6$

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

$v_7$

# Algoritmos de Programación Dinámica Síncrona

Problema	Ecuación de Bellman	Algoritmo
Predicción	Esperanza	Evaluación iterativa de la política
Control	Esperanza + Mejora codiciosa de la política	Iteración de la política
Control	Optimalidad	Iteración del valor

Los algoritmos están basados en funciones estado-valor  $v_\pi(s)$  o  $v_*(s)$

Complejidad  $O(mn^2)$  por iteración para m acciones y n estados

Se pueden aplicar tambien para la función acción-valor  $q_\pi(s,a)$  o  $q_*(s,a)$

Complejidad  $O(m^2n^2)$  por iteración

# Programación dinámica asíncrona

---

- Los métodos de PD descritos usan respaldos síncronos
  - Todos los estados son respaldados en paralelo
- PD asíncrona respalda los estados individualmente, en cualquier orden
- Para cada estado se aplica el respaldo apropiado
- Puede reducir el cómputo significativamente
- Garantizada la convergencia si todos los estados siguen siendo seleccionados

# Aprendizaje por refuerzo sin modelo

---

- Normalmente no tenemos un modelo de la dinámica del ambiente.
- Estimaremos los problemas de predicción y de control sin tener un modelo.

# Aprendizaje por refuerzo Monte-Carlo

---

- Los métodos MC aprenden directamente de los episodios de experiencia
- MC es libre de modelo: ningún conocimiento de las transiciones de PDM ni recompensas,
- MC aprende de episodios completos: no bootstrapping
- MC utiliza la idea más simple posible: valor = retorno promedio
- Advertencia: sólo se puede aplicar MC para PDMS episódicos
  - Todos los episodios deben terminar

# Evaluación de la política Monte-Carlo

---

- Objetivo: aprender  $v_\pi$  de episodios de experiencia bajo la política  $\pi$

$$S_1, A_1, R_2, \dots, S_k \sim \pi$$

- Recordemos que el retorno es la recompensa total descontada:

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Recordemos que la función de valor es el retorno esperado:

$$V_\pi(s) = E_\pi[G_t \mid S_t = s]$$

- La evaluación de las política Monte-Carlo utiliza la media empírica del retorno en lugar del retorno esperado

# Evaluación de la política Monte-Carlo tipo primera visita

---

- Para evaluar el estado  $S$
- El primer paso de tiempo  $t$  que el estado  $s$  es visitado en un episodio
- Se incrementa el contador  $N(s) \leftarrow N(s) + 1$
- Se incrementa el retorno total  $S(s) \leftarrow S(s) + G_t$
- El valor es estimado con el retorno promedio  $V(s) \leftarrow S(s)/N(s)$
- Según la ley de los grandes números,  $V(s) \rightarrow v_\pi(s)$  conforme  $N(s) \rightarrow \infty$

# Evaluación de la política Monte-Carlo tipo cada visita

---

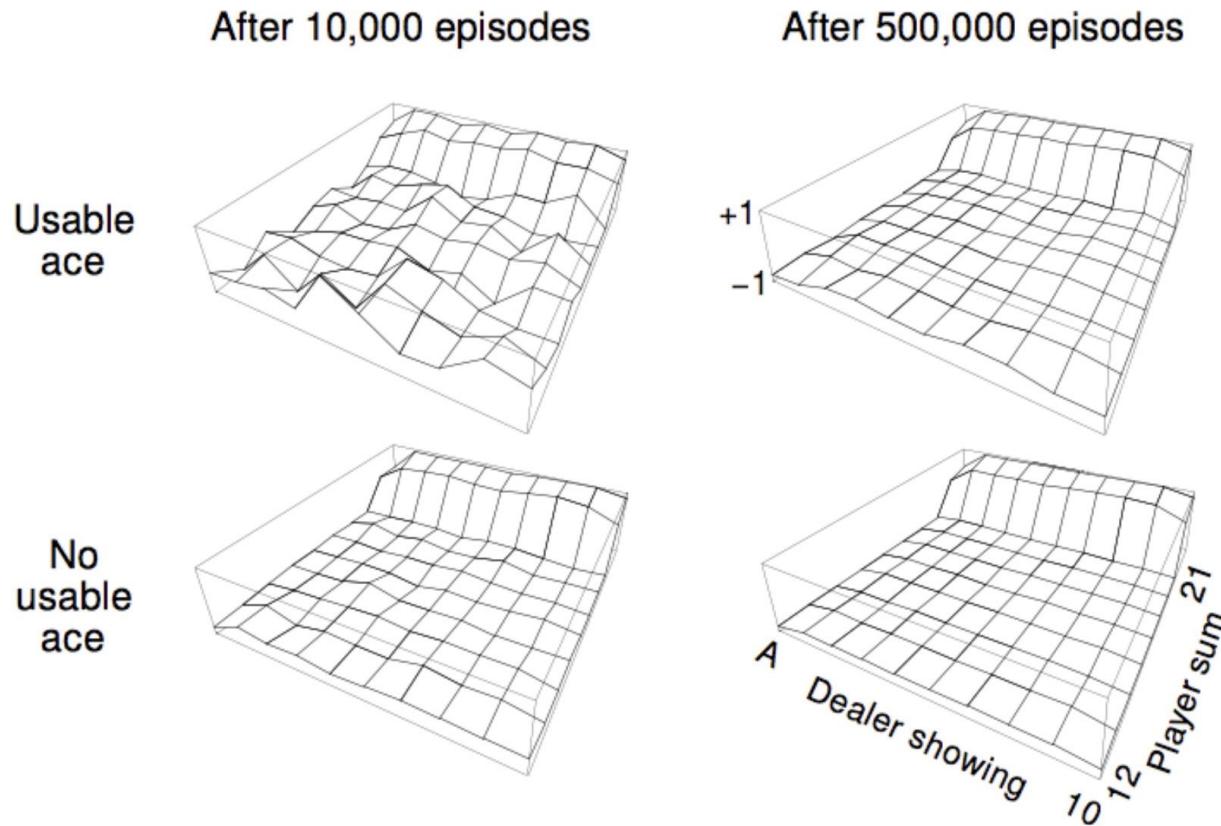
- Para evaluar el estado  $S$
- Cada paso de tiempo  $t$  que el estado  $s$  es visitado en un episodio
- Se incrementa el contador  $N(s) \leftarrow N(s) + 1$
- Se incrementa el retorno total  $S(s) \leftarrow S(s) + G_t$
- El valor es estimado con el retorno promedio  $V(s) \leftarrow S(s)/N(s)$
- Según la ley de los grandes números,  $V(s) \rightarrow v_\pi(s)$  conforme  $N(s) \rightarrow \infty$

# Ejemplo: Black Jack

---

- Estados (200 de ellos):
  - Suma actual (12-21)
  - Repartidor de cartas (as-10)
  - ¿Tengo un as usable (que valga 11)?
- Acción “me quedo”: Paro de recibir cartas (y termino)
- Acción “quiero otra”: Pido otra carta (sin reemplazo)
- Recompensa por “me quedo”:
  - +1 si la suma de las cartas > suma de las cartas del repartidor
  - 0 si la suma de las cartas = suma de las cartas del repartidor
  - -1 si la suma de las cartas < suma de las cartas del repartidor
- Recompensa por “quiero otra”:
  - -1 si la suma de las cartas > 21
  - 0 de otra manera
- Transiciones: automáticamente “quiero otra” si la suma de las cartas es < 12

# Función de valor Black Jack (Aprendizaje Monte-Carlo)



Política: “me quedo” si la suma de cartas es  $\geq 20$ , si no “quiero otra”

# Actualizaciones incrementales en Monte-Carlo

---

- Actualizar  $V(s)$  incrementalmente después de cada episodio  $S_1, A_1, R_2, \dots, S_T$
- Para cada estado  $S_t$  con retorno  $G_t$

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

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)} (G_t - V(S_t))$$

- En problemas no estacionarios, puede ser útil tener una media móvil, con el fin de olvidar episodios antiguos.

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$

# Temporal-Difference Learning

---

- Los métodos TD aprenden directamente de episodios de experiencia
- TD es libre de modelo: no hay conocimiento de las transiciones, ni de la función de recompensa del PDM
- TD Aprende de episodios incompletos, a través de bootstrapping
- TD actualiza una adivinanza hacia una adivinanza

# MC y TD

---

- Objetivo: aprender  $v_\pi$  en línea de la experiencia bajo la política  $\pi$
- Monte-Carlo tipo cada visita incremental
  - Actualización del valor  $V(S_t)$  hacia el retorno real  $G_t$ 
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$
- El algoritmo de aprendizaje temporal-difference más simple: TD(0)
  - Actualizar el valor  $V(S_t)$  hacia el retorno estimado  $R_{t+1} + \gamma V(S_{t+1})$ 
$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$
  - $R_{t+1} + \gamma V(S_{t+1})$  se llama el objetivo TD
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$  es llamado el error TD

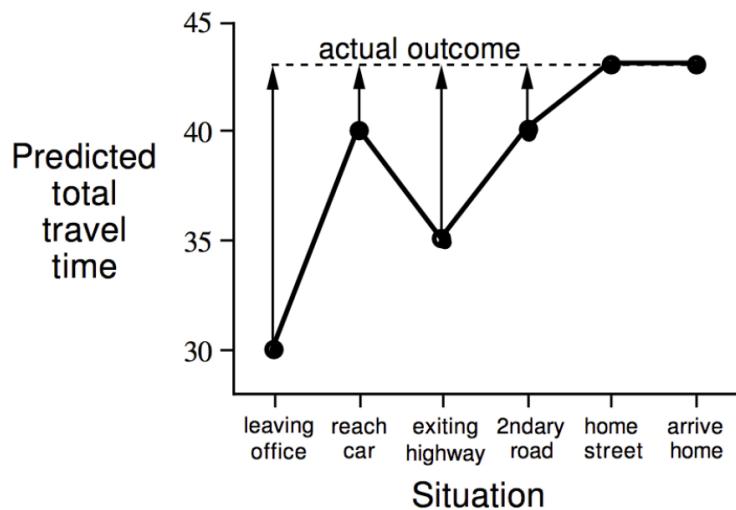
# Ejemplo: manejando a casa

Estado	Tiempo transcurrido	Tiempo restante predicho	Tiempo total predicho
Saliendo del trabajo	0	30	30
Ir al coche (lluvia)	5	35	40
Salir del periférico	20	15	35
Atrás de un camión	30	10	40
Calle de la casa	40	3	43
Llegando a casa	43	0	43

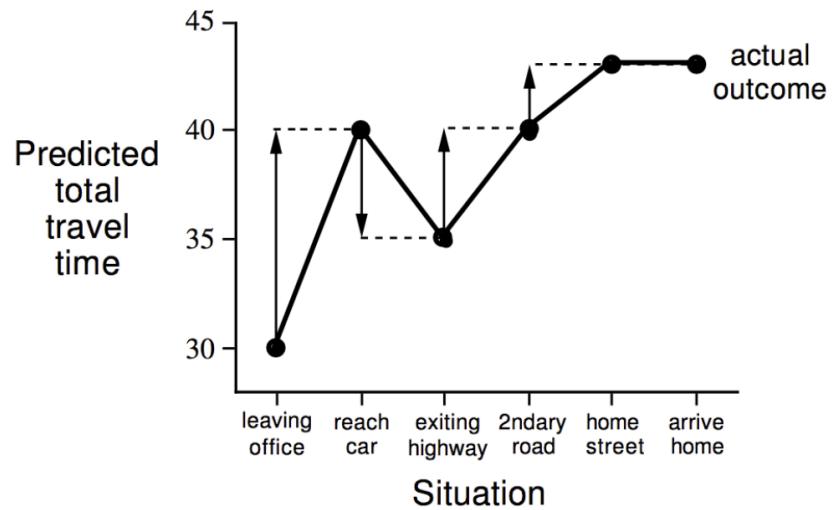
# Ejemplo: manejando a casa (MC vs. TD)

---

Cambios recomendados por el método Monte Carlo ( $\alpha=1$ )



Cambios recomendados por el método TD ( $\alpha=1$ )



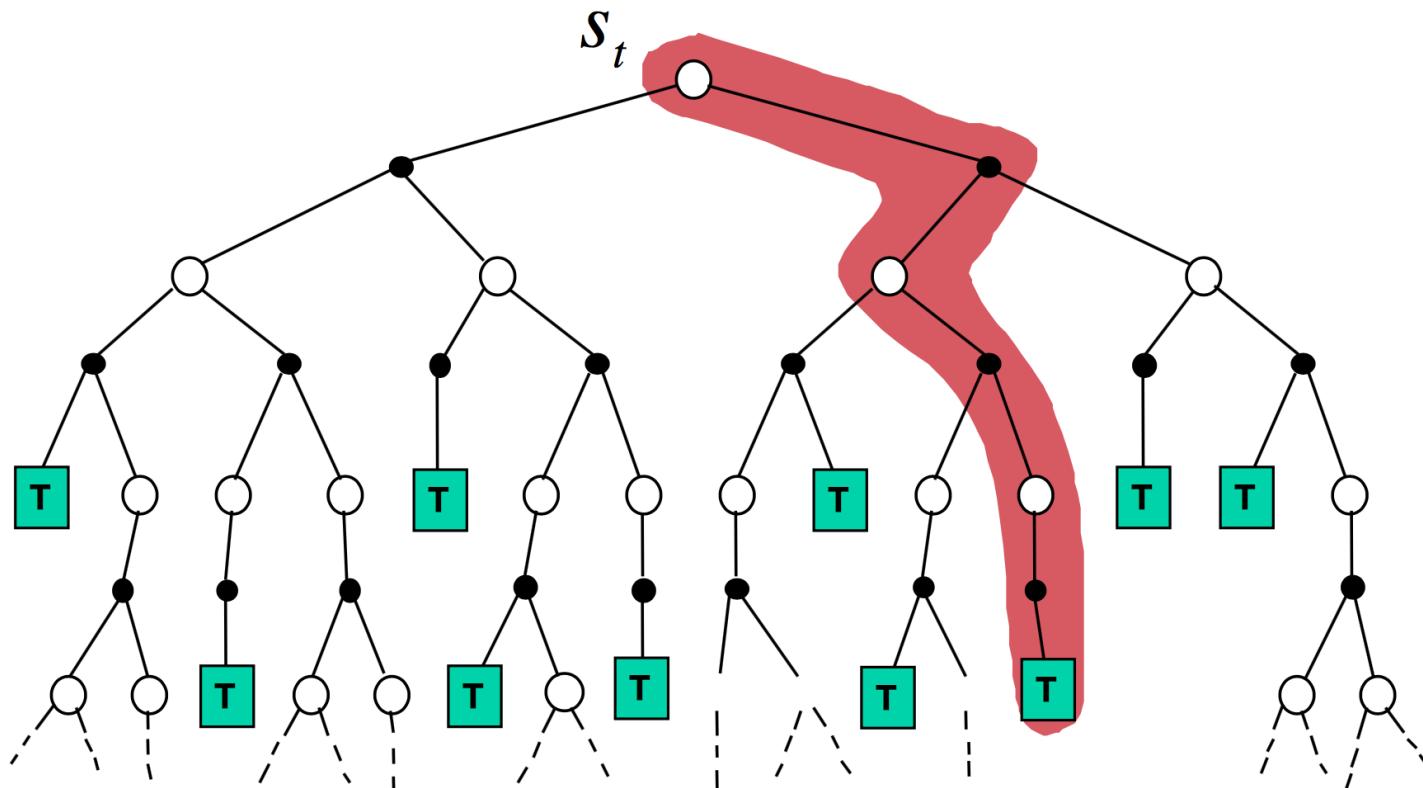
# Ventajas y desventajas MC vs. TD

---

- TD puede aprender antes de conocer el resultado final
  - TD puede aprender en línea después de cada paso
  - MC debe esperar hasta el final del episodio antes de que se conozca el retorno
- TD puede aprender sin el resultado final
  - TD puede aprender de secuencias incompletas
  - MC sólo puede aprender de secuencias completas
  - TD trabaja en entornos continuos (sin termino)
  - MC sólo funciona en entornos episódicos (con termino)

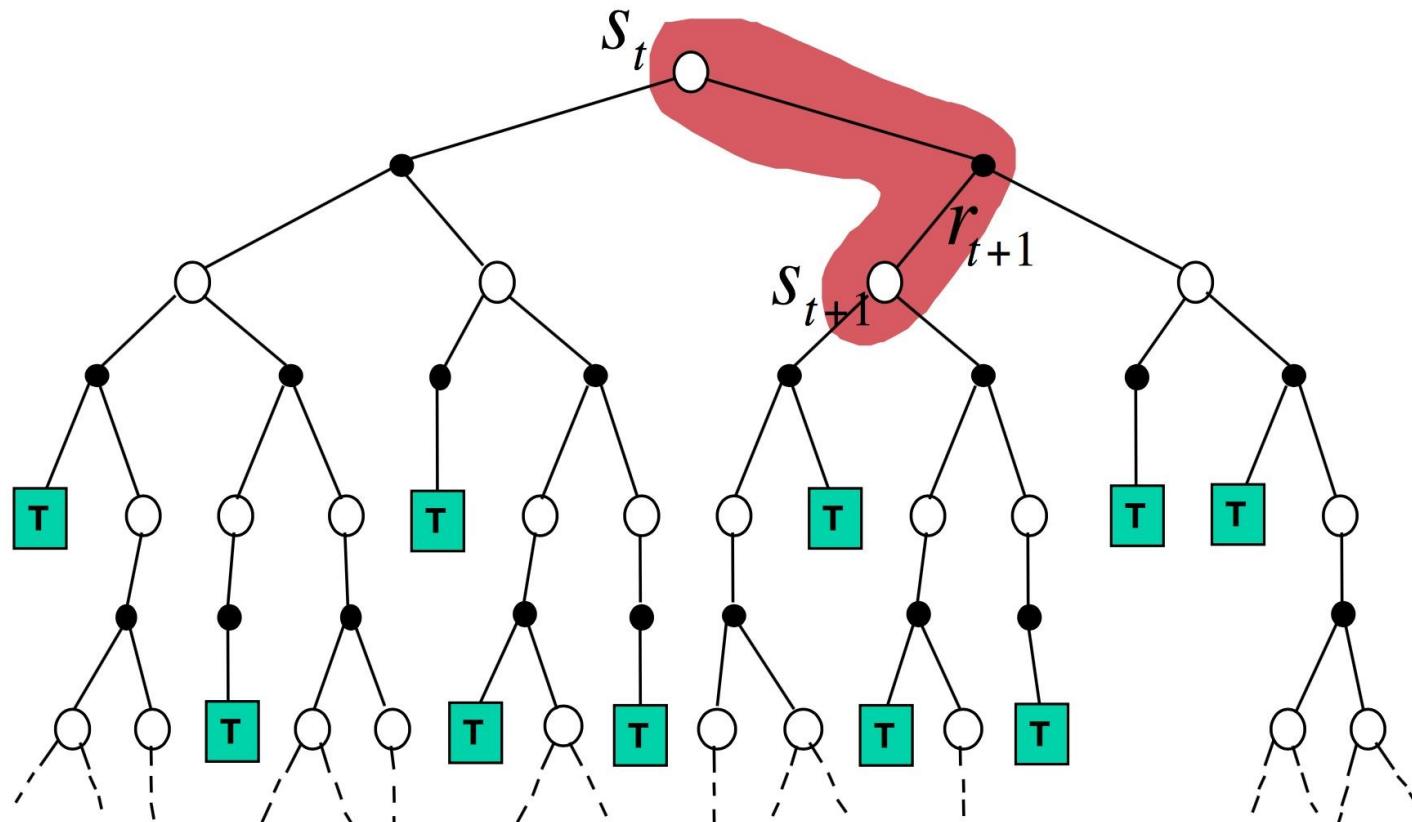
# Respaldo Monte - Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t - V(S_t))$$



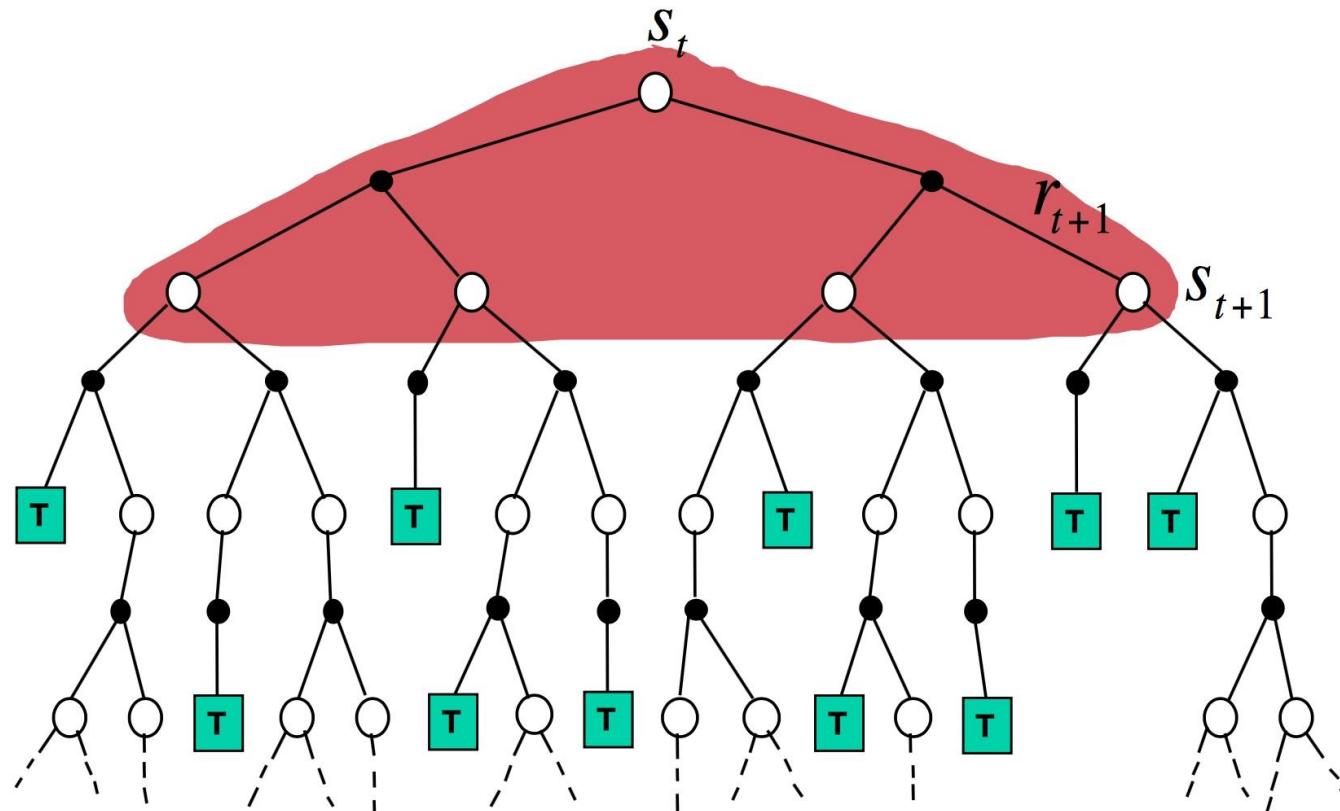
# Respaldo Temporal - Difference

$$V(S_t) \leftarrow V(S_t) + \alpha (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$



# Respaldo Programación Dinámica

$$V(S_t) \leftarrow \mathbb{E}_\pi [R_{t+1} + \gamma V(S_{t+1})]$$

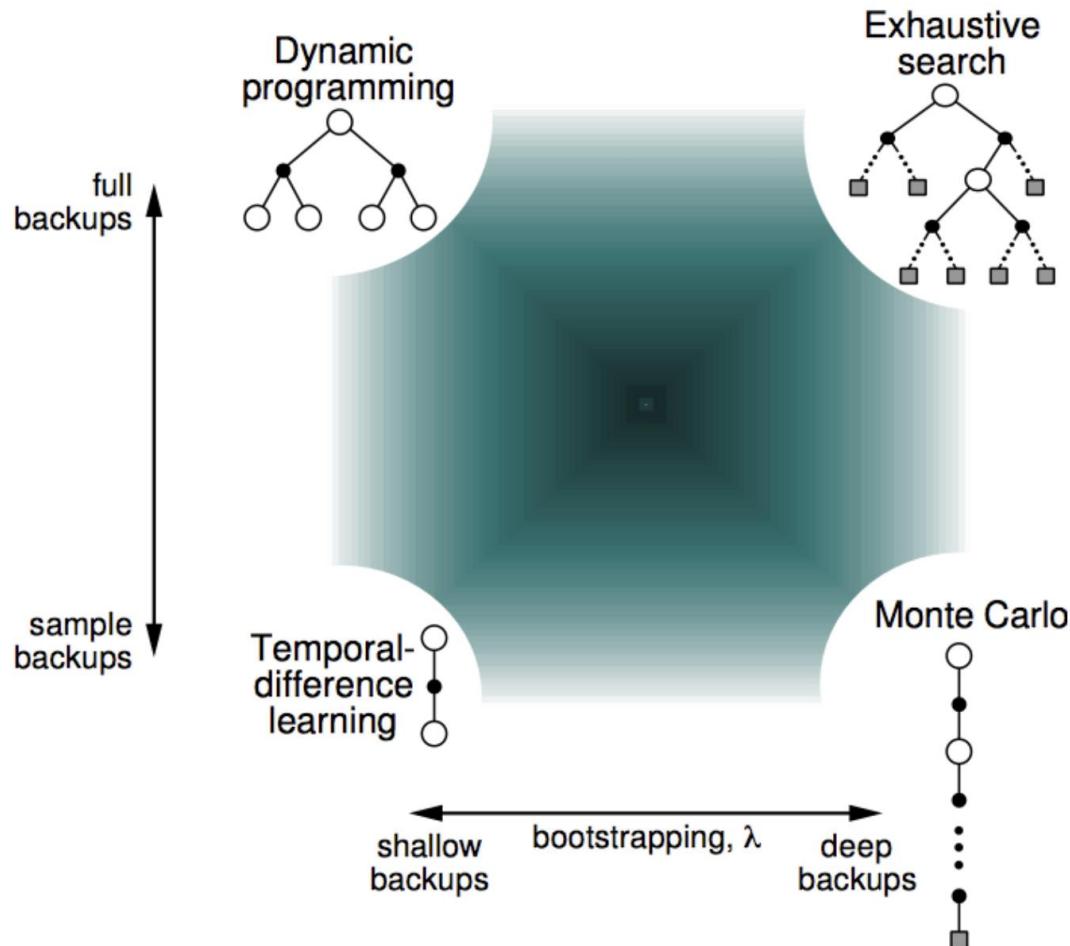


# Bootstrapping y muestreo

---

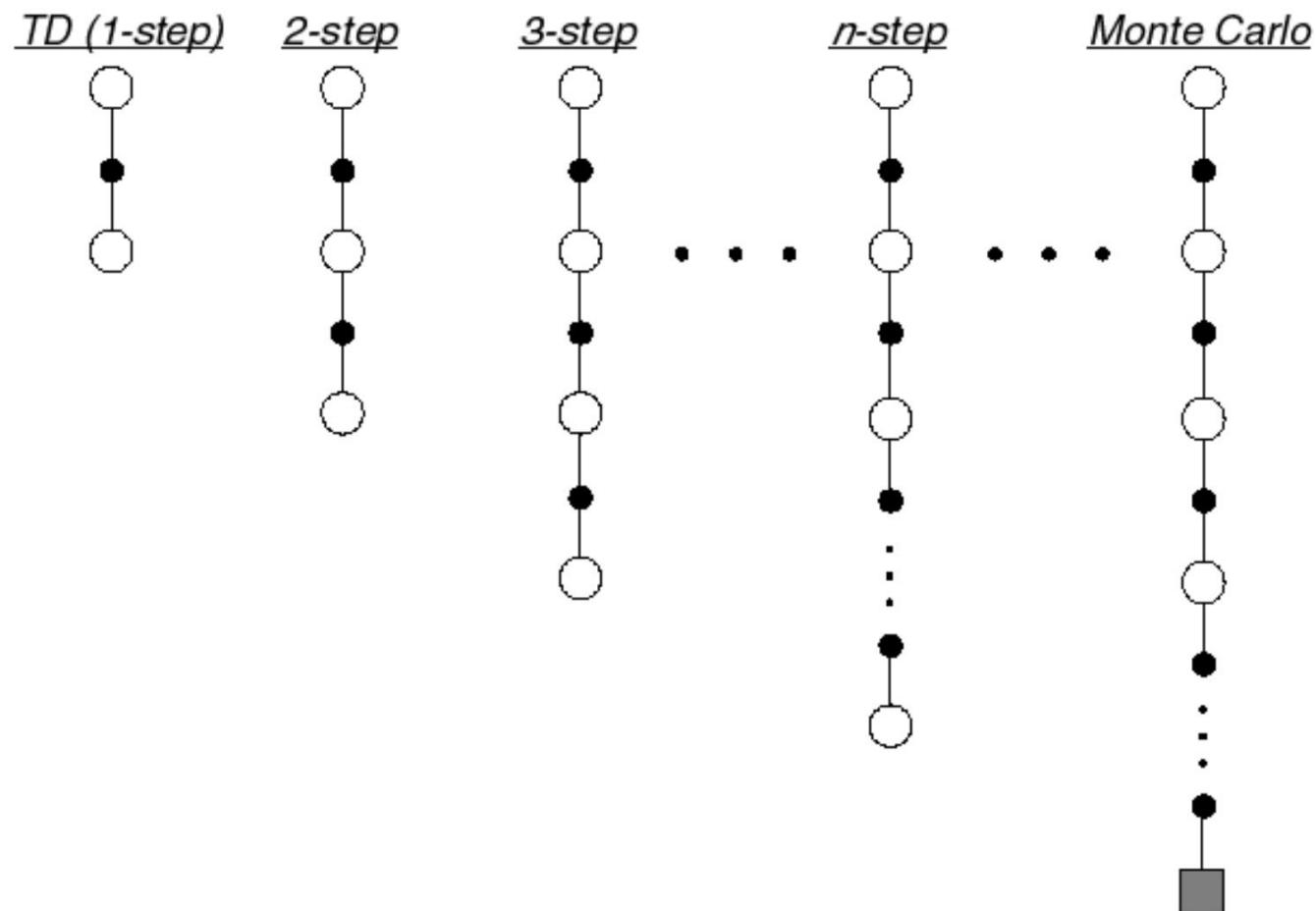
- Bootstrapping: actualización involucra una estimación
  - MC sin bootstrap
  - DP con bootstrap
  - TD con bootstrap
- Muestreo: actualización involucra muestrear una expectativa
  - MC muestrea
  - DP no muestrea
  - TD muestrea

# Vista unificada de aprendizaje por refuerzo



# Predictión de n-pasos

---



# Predicción de n-pasos

---

- Considere las siguientes respuestas de n-pasos para  $n=1, 2, \infty$

$$n = 1 \quad (TD) \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$

$$n = 2 \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$$

⋮

⋮

$$n = \infty \quad (MC) \quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T$$

- Definir el retorno de n-pasos

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- Aprendizaje temporal difference de n-pasos

$$V(S_t) \leftarrow V(S_t) + \alpha \left( G_t^{(n)} - V(S_t) \right)$$

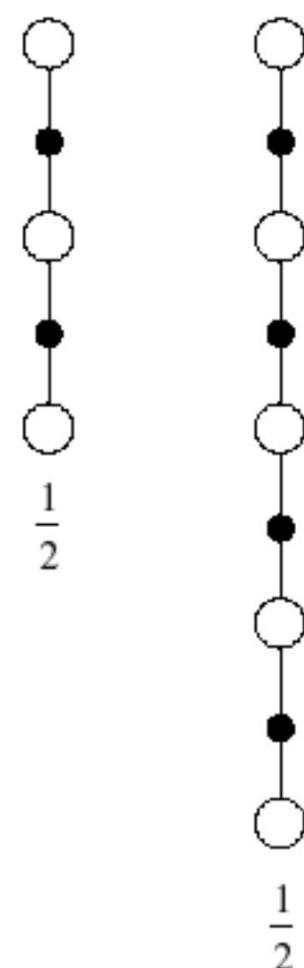
# Promediando retornos de n-pasos

---

- Podemos promediar retornos de n-pasos sobre diferentes n's
- Por ejemplo promediar los retornos de 2 y 4 pasos

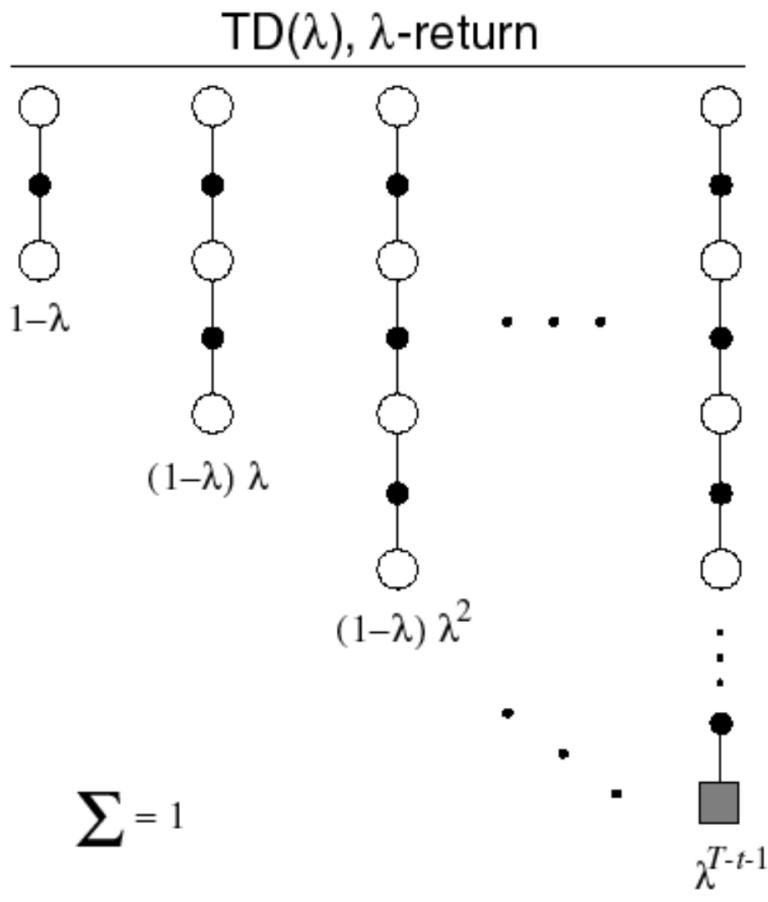
$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(4)}$$

- Combina la información de dos pasos de tiempo diferentes
- ¿Podemos combinar eficientemente la información de todos los pasos de tiempo?



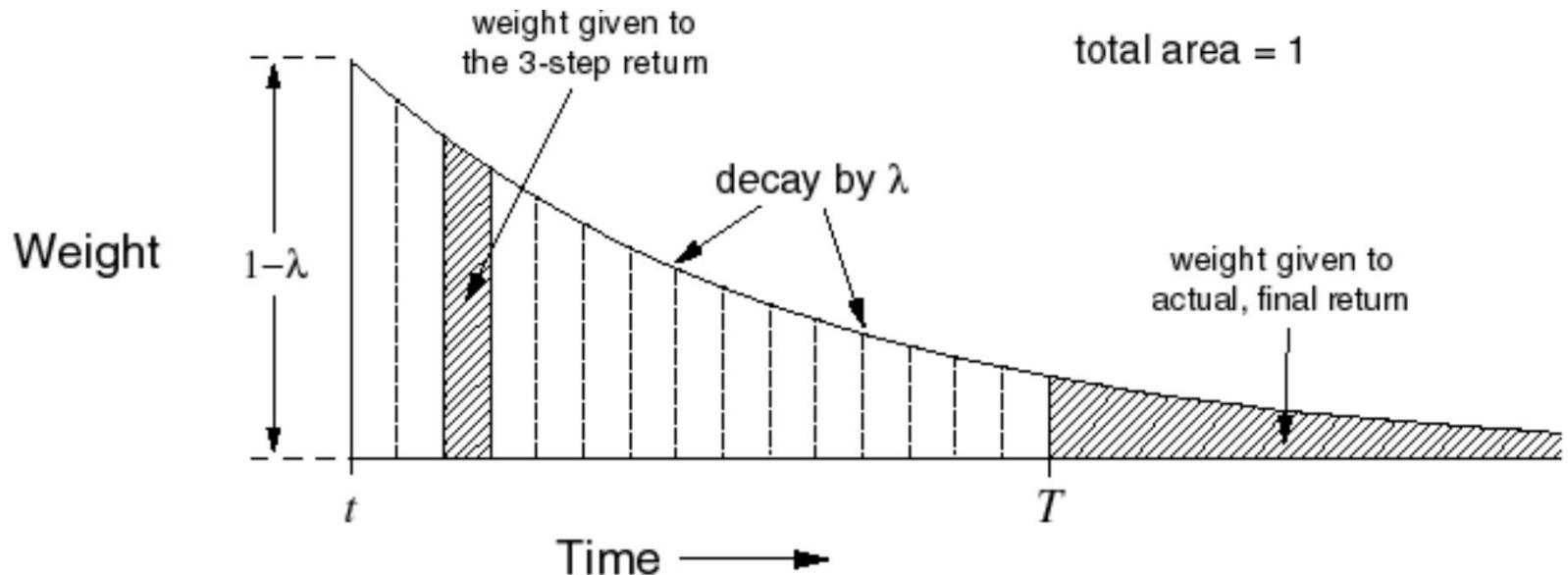
# Retorno $\lambda$

---



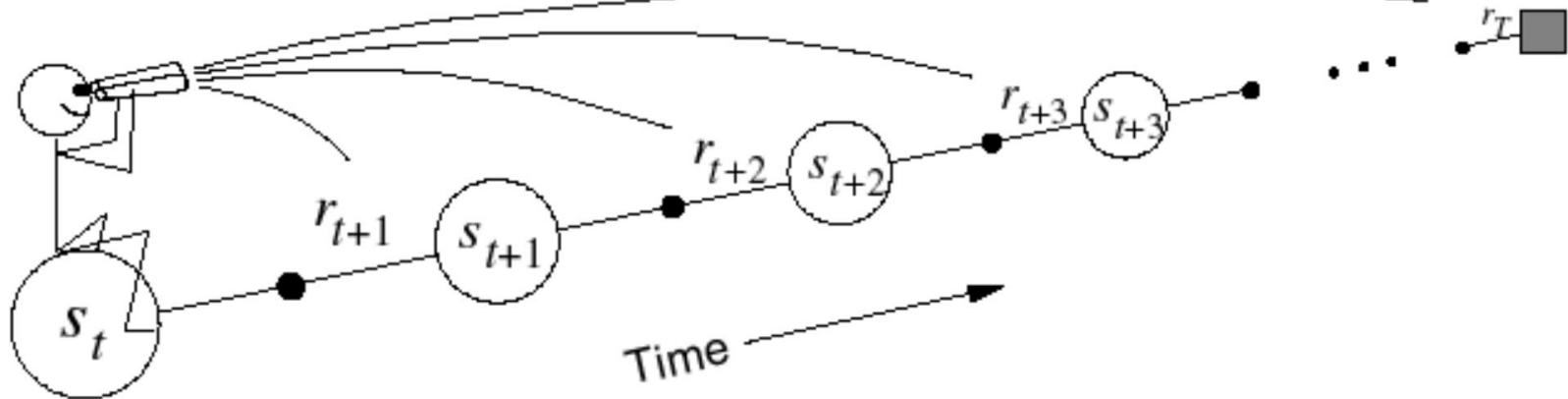
- El retorno  $\lambda$   $G_t^\lambda$  combina todos los retornos de  $n$ -pasos  $G_t^{(n)}$
- Usando el peso  $(1 - \lambda)\lambda^{n-1}$ 
$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$
- Vista hacia delante
$$V(S_t) \leftarrow V(S_t) + \alpha \left( G_t^\lambda - V(S_t) \right)$$

# Función de pesos TD( $\lambda$ )



$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

# Vista hacia adelante TD( $\lambda$ )



- Actualiza la función de valor hacia el retorno  $\lambda$
- La vista hacia delante ve hacia el futuro para calcular  $G_t^\lambda$
- Como en MC, sólo puede ser calculada en episodios completos

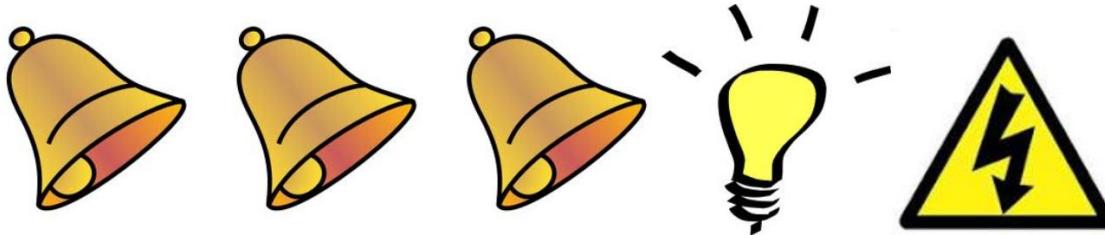
# Vista hacia atrás TD( $\lambda$ )

---

- Vista hacia adelante provee la teoría
- Vista hacia atrás provee el mecanismo
- Actualiza en línea, cada paso, de secuencias incompletas

# Rastros de elegibilidad

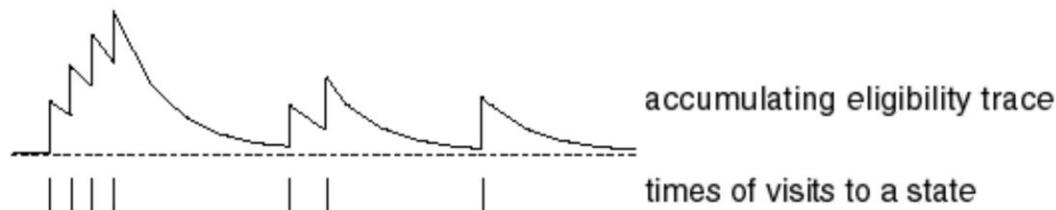
---



- Problema de asignación del crédito: ¿fue la campana o el foco lo que provocó la descarga?
- Heurística de frecuencia: asigna el crédito a los estados más frecuentes
- Heurística de novedad: asigna el crédito a los estados más recientes
- Rastreos de elegibilidad utiliza ambas heurísticas

$$E_0(s) = 0$$

$$E_t(s) = \gamma \lambda E_{t-1}(s) + \mathbf{1}(S_t = s)$$

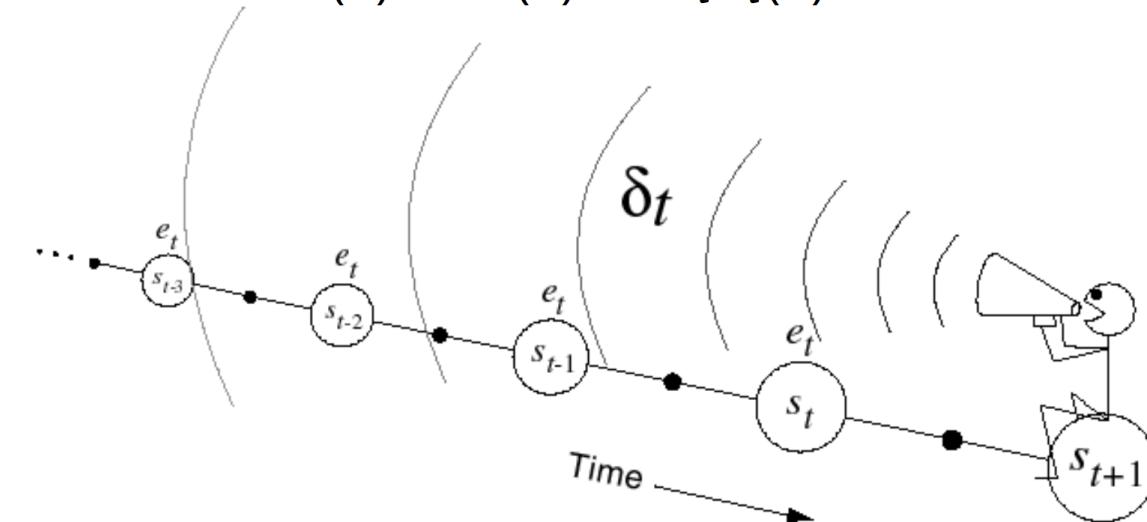


# Vista hacia atrás TD( $\lambda$ )

- Se mantiene un rastro de elegibilidad para cada estado
- Actualiza el valor de  $V(s)$  para cada estado  $s$
- En proporción al error de TD  $\delta_t$  y el rastro de elegibilidad  $E_t(s)$

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$



# TD( $\lambda$ ) y TD(0)

---

- Cuando  $\lambda=0$ , sólo el estado actual se actualiza

$$E_t(s) = \mathbf{1}(S_t = s)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

- Esto es exactamente equivalente a la actualización TD(0)

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t$$

# TD( $\lambda$ ) y MC

---

- Cuando  $\lambda=1$ , la acreditación se pospone hasta el final del episodio
- Considerar ambientes episódicos con actualizaciones fuera de línea
- A través del curso de un episodio, la actualización total para TD(1) es lo mismo que la actualización total para MC
- La suma de actualizaciones fuera de línea es idéntica para la vista hacia atrás y hacia adelante de TD( $\lambda$ )

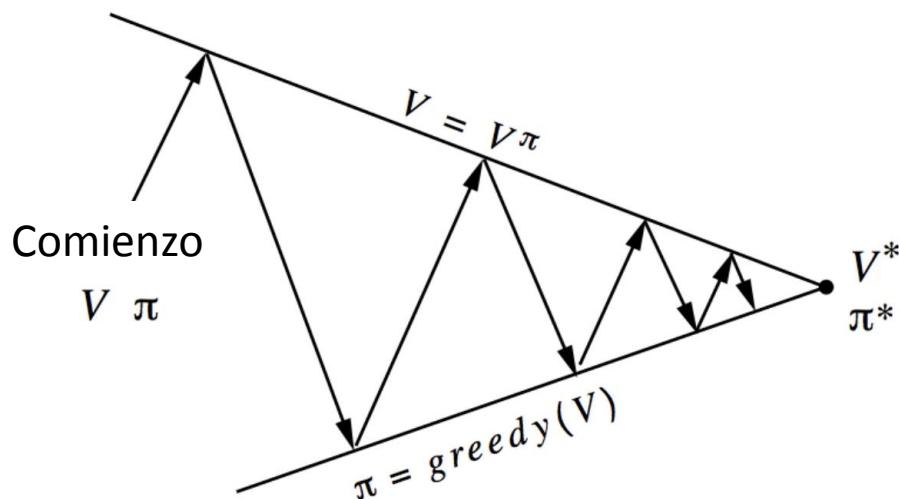
$$\sum_{t=1}^T \alpha \delta_t E_t(s) = \sum_{t=1}^T \alpha \left( G_t^\lambda - V(S_t) \right) \mathbf{1}(S_t = s)$$

# Control sin modelo

---

- Para la mayoría de los problemas
  - PDM no es conocido, pero podemos tomar muestras a través de experiencia
  - PDM es conocido, pero es demasiado grande para ser utilizado, excepto a través de tomar muestras

# Iteración de la política

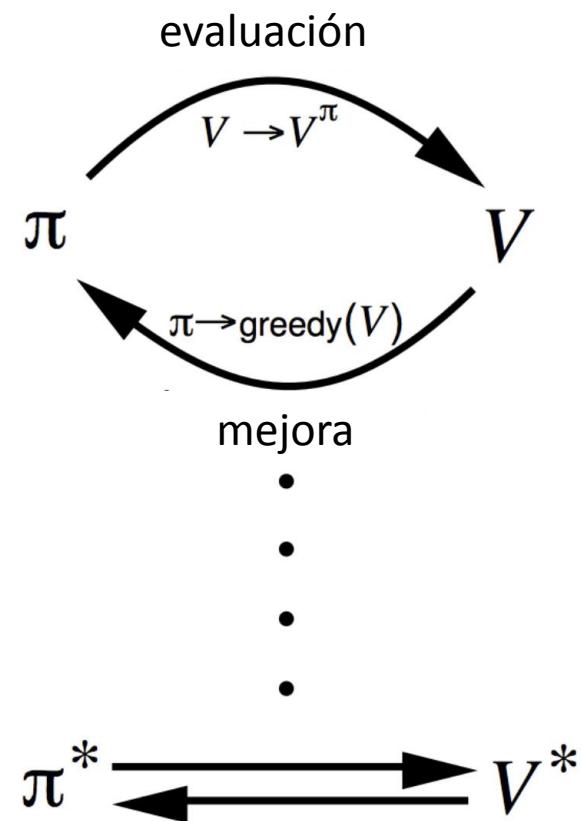


Estimación de la política Estimar  $v_\pi$

Evaluación iterativa de la política

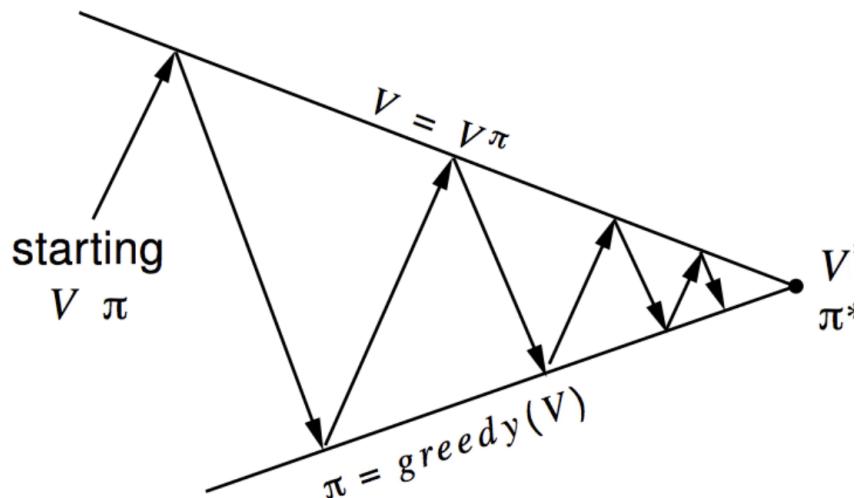
Mejora de la política Generar  $\pi' \geq \pi$

Mejora codiciosa de la política



# Iteración de la política con MC

---



- Evaluación de la política: MC necesitaría muchos episodios para que  $V = v_\pi$
- Mejora de la política: Mejora codiciosa requiere de un PDM y provocaría no explorar todo el espacio

# Iteración de la política sin modelo usando la función de acción-valor

---

- La actualización codiciosa de la política sobre  $V(s)$  requiere el modelo del PDM

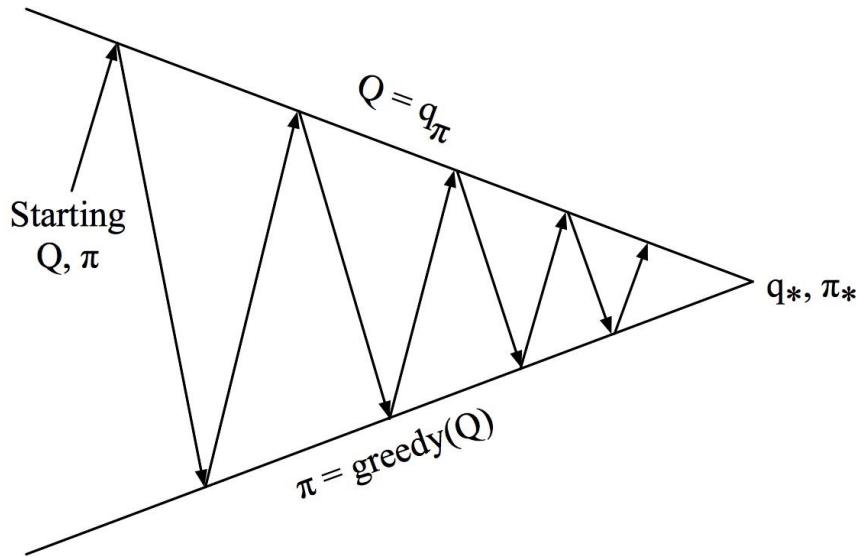
$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- La actualización codiciosa de la política sobre  $Q(s, a)$  no requiere del modelo

$$\pi'(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$$

# Iteración de la política usando la función de acción-valor

---



- Evaluación de la política:  $Q = q_\pi$
- Mejora de la política: Mejora codiciosa provocaría no explorar todo el espacio

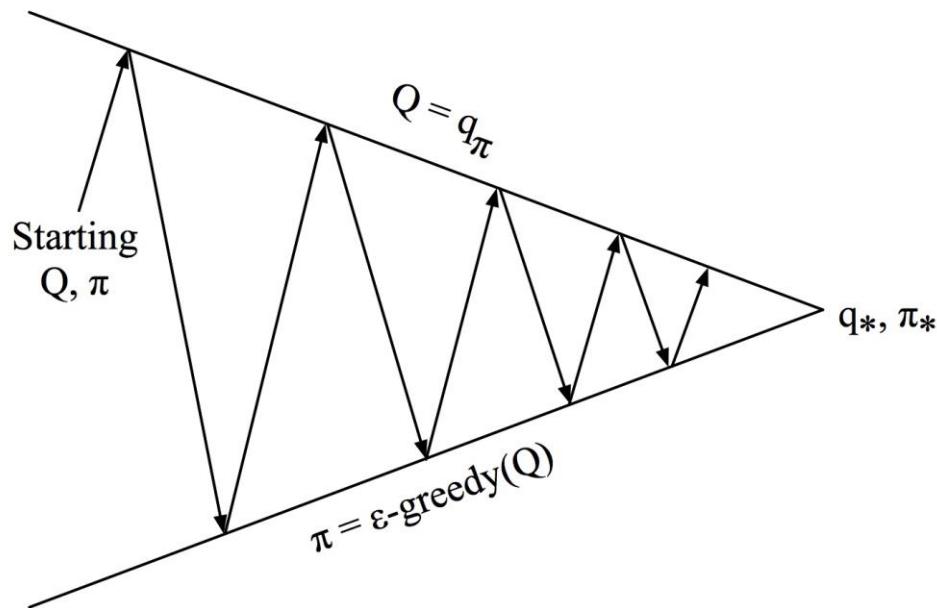
# Exploración $\epsilon$ -greedy

---

- La idea más simple para asegurar exploración continua
- Todas las acciones son probadas con probabilidad distinta de cero
- Con probabilidad  $1 - \epsilon$  escogemos la acción codiciosa
- Con probabilidad  $\epsilon$  escogemos una acción al azar

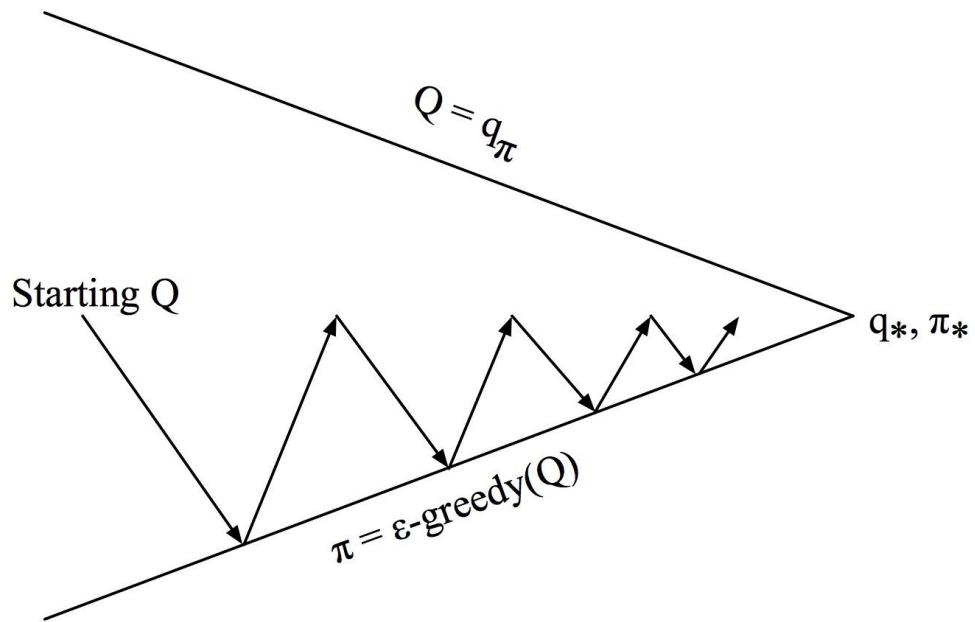
$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{si } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}} Q(s, a) \\ \epsilon/m & \text{en caso contrario} \end{cases}$$

# Iteración de la política usando la función de acción-valor y $\epsilon$ -greedy



- Evaluación de la política:  $Q = q_\pi$
- Mejora de la política: Mejora codiciosa  $\epsilon$ -greedy

# Iteración de la política usando la función de acción-valor y $\epsilon$ -greedy



- Evaluación de la política:  $Q \approx q_\pi$
- Mejora de la política: Mejora codiciosa  $\epsilon$ -greedy

# Control GLIE MC

---

- GLIE: Greedy in the Limit with Infinite Exploration
- Toma una muestra del episodio k usando  $\pi$ :  $\{S_1, A_1, R_2, \dots, S_T\} \sim \pi$
- Para cada estado y  $S_t$  acción  $A_t$  en el episodio,

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$

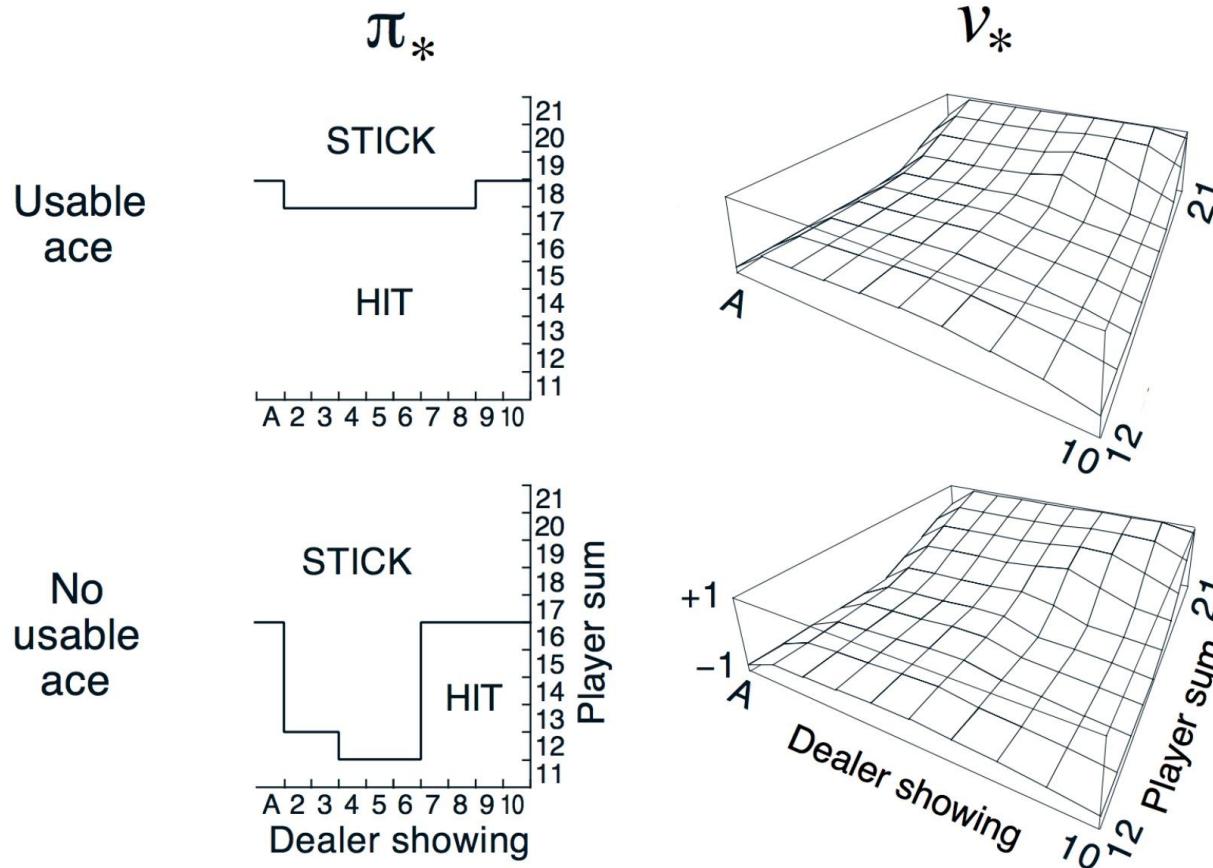
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t))$$

- Mejora la política con base en la nueva función acción-valor

$$\epsilon \leftarrow 1/k$$

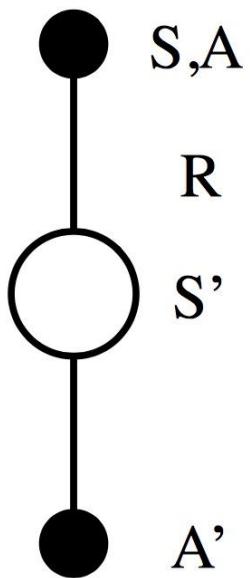
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

# MC control en Blackjack



# Actualizar la función acción-valor con Sarsa

---



$$Q(S, A) \leftarrow Q(S, A) + \alpha (R + \gamma Q(S', A') - Q(S, A))$$

# Algoritmo Sarsa

---

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$

Repeat (for each episode):

    Initialize  $S$

    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

    Repeat (for each step of episode):

        Take action  $A$ , observe  $R, S'$

        Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$S \leftarrow S'; A \leftarrow A'$ ;

    until  $S$  is terminal

# Algoritmo Sarsa( $\lambda$ )

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

Repeat (for each episode):

$E(s, a) = 0$ , for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$

Initialize  $S, A$

Repeat (for each step of episode):

Take action  $A$ , observe  $R, S'$

Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$

$E(S, A) \leftarrow E(S, A) + 1$

For all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$ :

$Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$

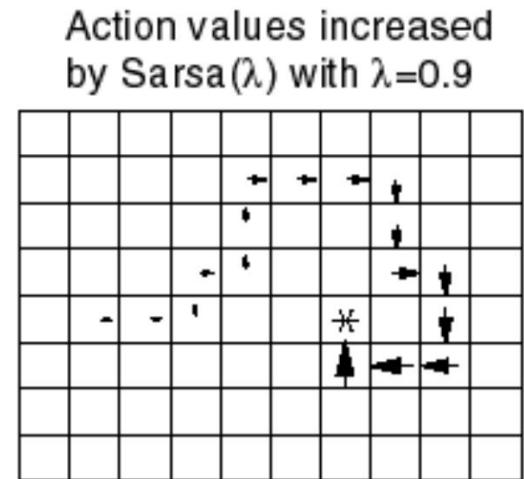
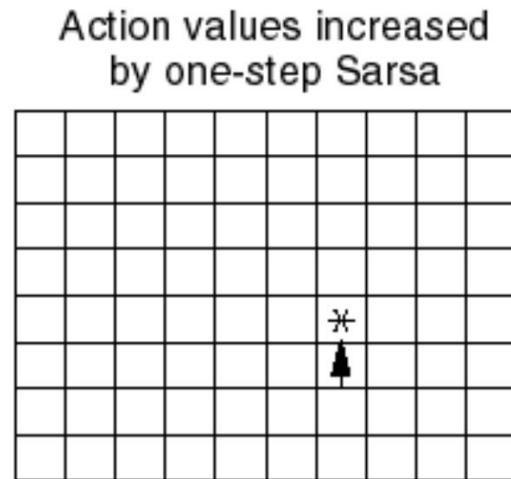
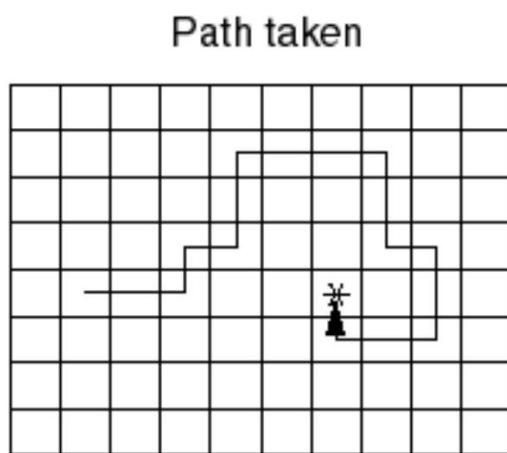
$E(s, a) \leftarrow \gamma \lambda E(s, a)$

$S \leftarrow S'; A \leftarrow A'$

until  $S$  is terminal

# Ejemplo mundo cuadriculado Sarsa( $\lambda$ )

---



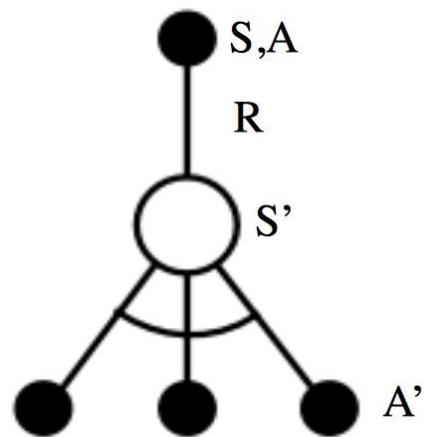
# Aprendizaje fuera de política

---

- Evaluar la política de destino  $\pi(a|s)$  para calcular  $v_\pi(s)$  o  $q_\pi(s, a)$
- Mientras se sigue la política de comportamiento  $\mu(a|s)$   
 $\{S_1, A_1, R_2, \dots, S_T\} \sim \mu$
- ¿Porque es esto importante?
  - Aprender de la observación de seres humanos u otros agentes
  - Reutilizar la experiencia generada a partir de políticas antiguas  $\pi_1, \pi_2, \dots, \pi_{t-1}$
  - Aprender la política óptima mientras sigue una política exploratoria
  - Aprender varias políticas mientras sigue una política

# Control Q learning

---



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left( R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

# Algoritmo Q learning

---

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$   
Repeat (for each episode):

    Initialize  $S$

    Repeat (for each step of episode):

        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)

        Take action  $A$ , observe  $R, S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$S \leftarrow S'$ ;

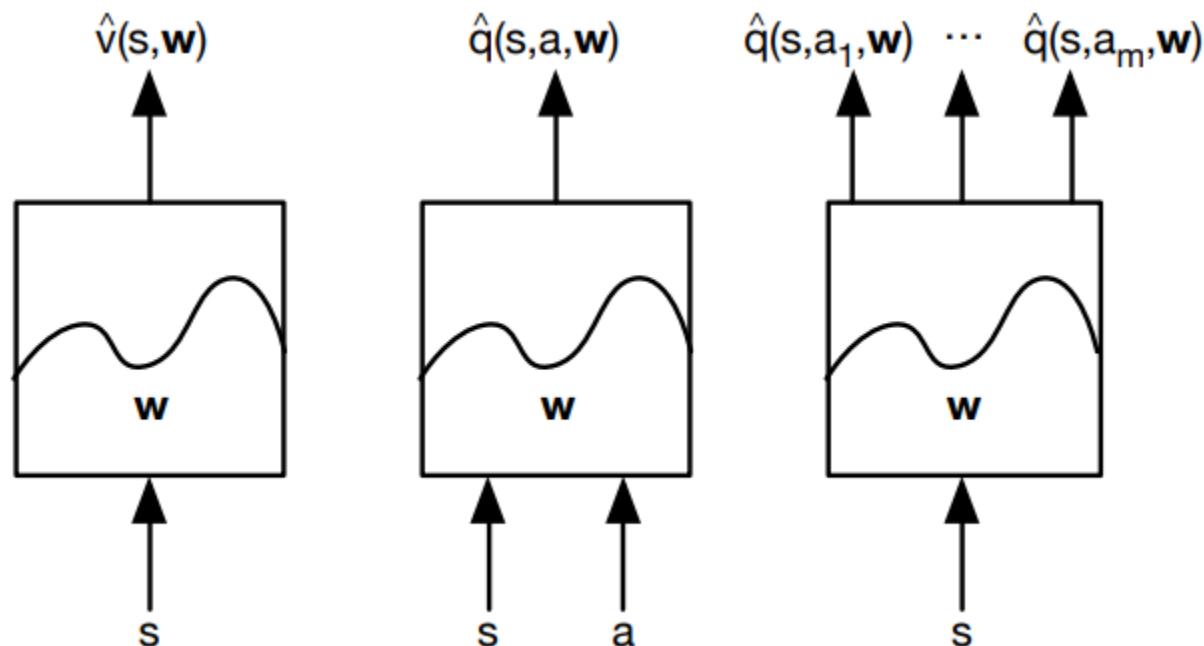
    until  $S$  is terminal

# Relación entre PD y TD

	<i>Full Backup (DP)</i>	<i>Sample Backup (TD)</i>
Bellman Expectation Equation for $v_\pi(s)$	$v_\pi(s) \leftarrow s$ <p>A tree diagram illustrating the Bellman Expectation Equation for <math>v_\pi(s)</math>. The root node is labeled <math>v_\pi(s) \leftarrow s</math>. It branches down to a node labeled <math>a</math>, which further branches to two nodes. The left node is labeled <math>r</math> and the right node is labeled <math>s'</math>. Below this, the tree continues to branch into four nodes, with the bottom-right node being a solid black circle.</p> <p>Iterative Policy Evaluation</p>	$TD\ Learning$ <p>A vertical sequence of three nodes: a solid black circle at the top, followed by a solid black dot in the middle, and a solid white circle at the bottom.</p>
Bellman Expectation Equation for $q_\pi(s, a)$	$q_\pi(s, a) \leftarrow s, a$ <p>A tree diagram illustrating the Bellman Expectation Equation for <math>q_\pi(s, a)</math>. The root node is labeled <math>q_\pi(s, a) \leftarrow s, a</math>. It branches down to a node labeled <math>r</math>, which further branches to two nodes. The left node is labeled <math>s'</math> and the right node is labeled <math>a'</math>. Below this, the tree continues to branch into four nodes, with the bottom-right node being a solid black circle.</p> <p>Q-Policy Iteration</p>	$Sarsa$ <p>A vertical sequence of four nodes: a solid black circle at the top, followed by a solid white circle in the middle, then a solid black dot below it, and finally a solid black circle at the bottom.</p>
Bellman Optimality Equation for $q_*(s, a)$	$q_*(s, a) \leftarrow s, a$ <p>A tree diagram illustrating the Bellman Optimality Equation for <math>q_*(s, a)</math>. The root node is labeled <math>q_*(s, a) \leftarrow s, a</math>. It branches down to a node labeled <math>r</math>, which further branches to two nodes. The left node is labeled <math>s'</math> and the right node is labeled <math>a'</math>. Below this, the tree continues to branch into four nodes, with all bottom nodes being solid black circles.</p> <p>Q-Value Iteration</p>	$Q-Learning$ <p>A tree diagram illustrating Q-Learning. The root node is a solid black circle. It branches down to three solid black circles at the bottom level.</p>

# Aproximación de la función de valor

---



# Algoritmos de predicción incremental

---

- Para MC, el objetivo es  $G_t$

$$\Delta \mathbf{w} = \alpha(G_t - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

- Para TD(0), el objetivo es  $R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}) - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

- Para TD( $\lambda$ ), el objetivo es  $G_t^\lambda$

$$\Delta \mathbf{w} = \alpha(G_t^\lambda - \hat{v}(S_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{v}(S_t, \mathbf{w})$$

# Algoritmos de control incremental

---

- Para MC, el objetivo es  $G_t$

$$\Delta \mathbf{w} = \alpha(G_t - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- Para TD(0), el objetivo es  $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

$$\Delta \mathbf{w} = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- Para TD( $\lambda$ ) hacia adelante, el objetivo es  $q_t^\lambda$

$$\Delta \mathbf{w} = \alpha(q_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

- Para TD( $\lambda$ ) hacia atrás

$$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})$$

$$E_t = \gamma \lambda E_{t-1} + \nabla_{\mathbf{w}} \hat{q}(S_t, A_t, \mathbf{w})$$

$$\Delta \mathbf{w} = \alpha \delta_t E_t$$

# Deep Q-Networks (DQN)

---

DQN uses **experience replay** and **fixed Q-targets**

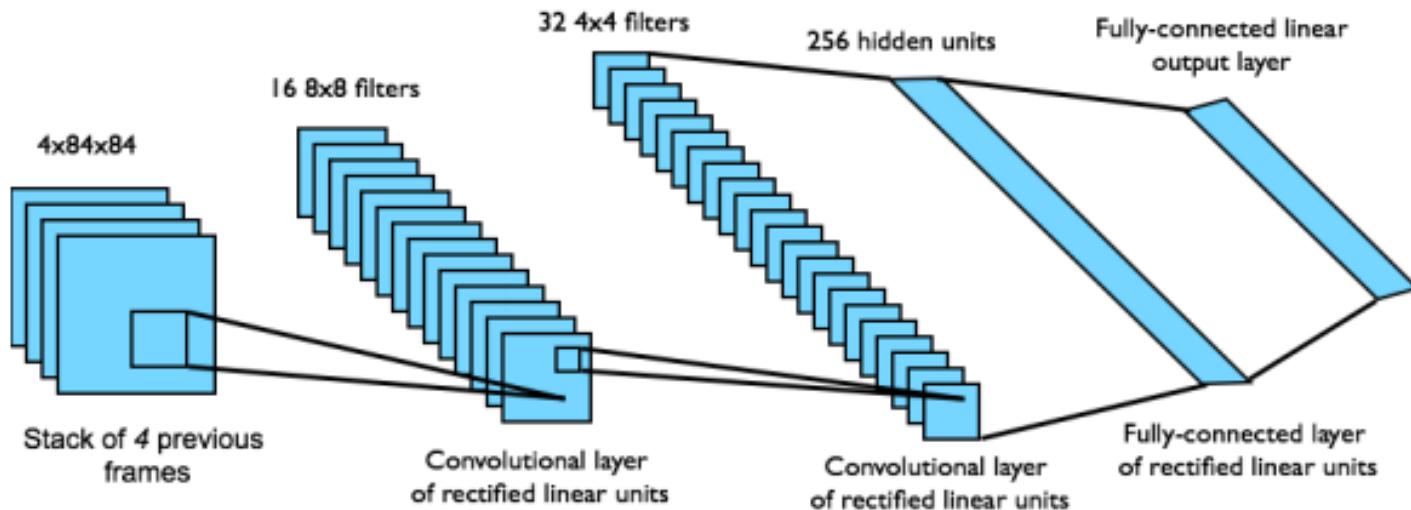
- Take action  $a_t$  according to  $\epsilon$ -greedy policy
- Store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in replay memory  $\mathcal{D}$
- Sample random mini-batch of transitions  $(s, a, r, s')$  from  $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters  $w^-$
- Optimise MSE between Q-network and Q-learning targets

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

- Using variant of stochastic gradient descent

# DQN en Atari

- End-to-end learning of values  $Q(s, a)$  from pixels  $s$
- Input state  $s$  is stack of raw pixels from last 4 frames
- Output is  $Q(s, a)$  for 18 joystick/button positions
- Reward is change in score for that step



Network architecture and hyperparameters fixed across all games

# Resultados de DQN en Atari

