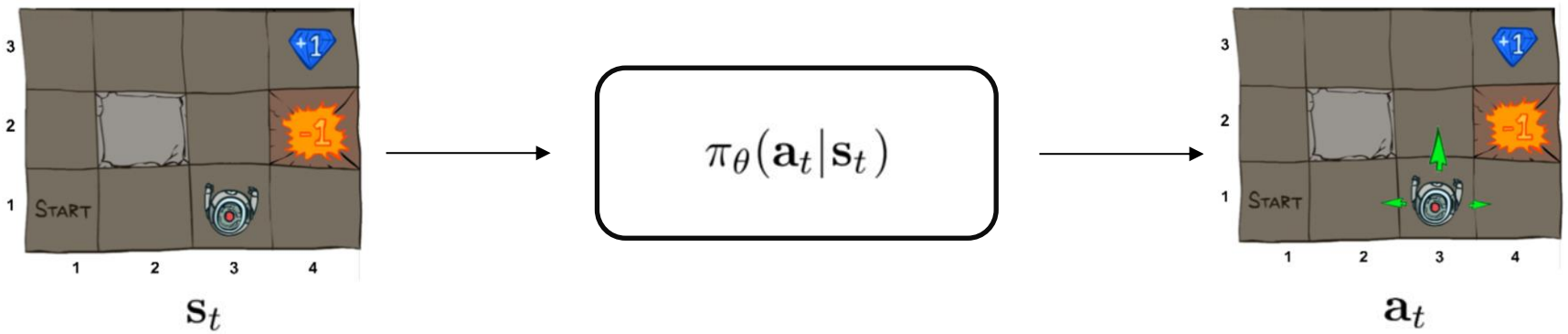
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 - Inteligencia Artificial

Value Iteration y Q-Learning

Hans Löbel
Dpto. Ingeniería de Transporte y Logística
Dpto. Ciencia de la Computación

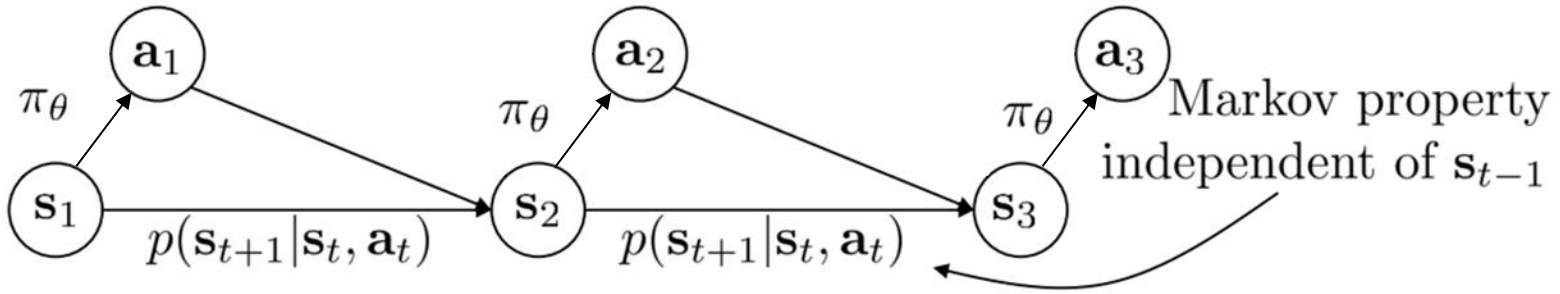
Antes de empezar con las técnicas, un poco de notación



$\mathbf{s}_t$ – state

$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy



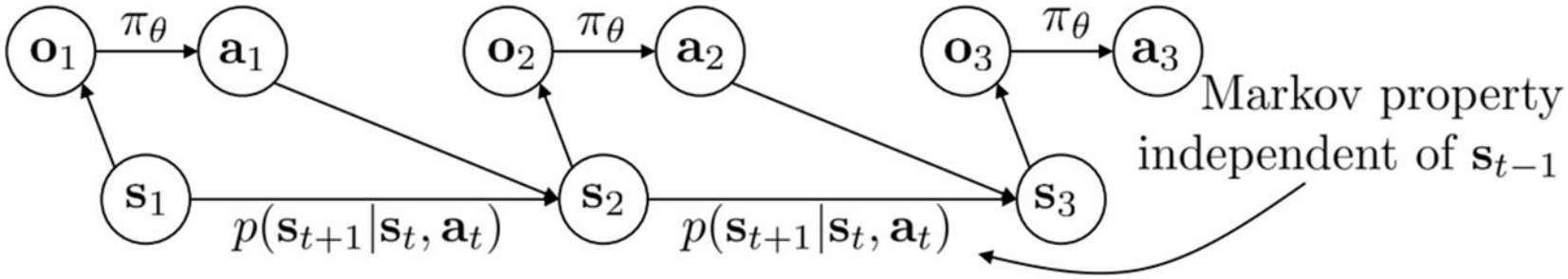
Antes de empezar con las técnicas, un poco de notación

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

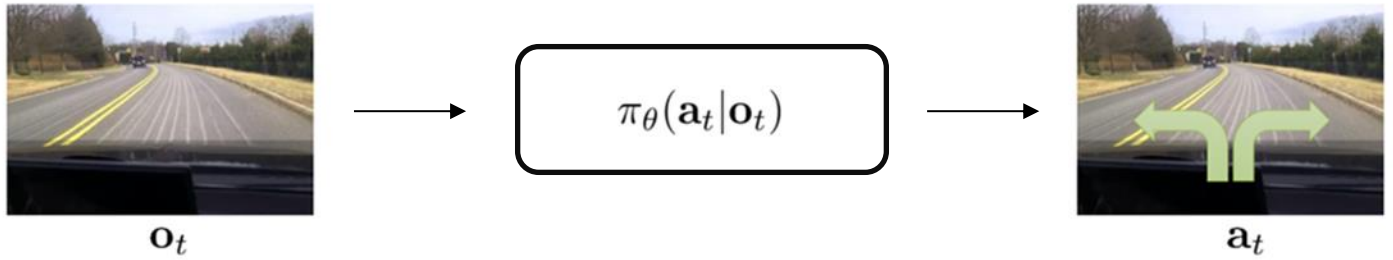
$\mathbf{o}_t$ $\mathbf{a}_t$

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)



La recompensa actúa como una especie de supervisión



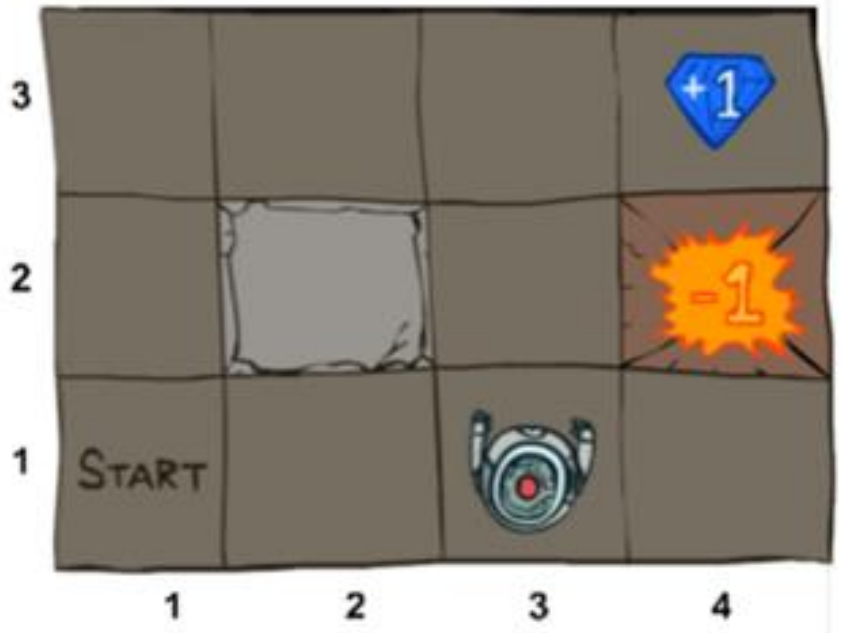
which action is better or worse?

$r(\mathbf{s},\mathbf{a},\mathbf{s}')$: reward function ⟶ tells us which states and actions are better

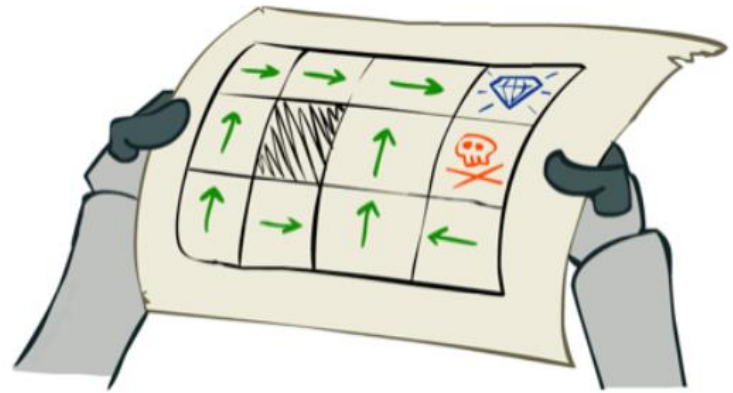
high reward 

 low reward

$S, A, r(s, a, s')$ y $p(s'|s, a)$ definen un proceso de decisión markoviano (MDP)



π:



$$max_\pi \mathrm{E}[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1})|\pi]$$

Comencemos enfrentando esto de manera intuitiva

La idea de “buenos estados” se puede formalizar a través de la función de valor

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \mid \pi, s_0 = s\right]$$

$V^*(s)$= suma de las recompensas con descuento al empezar en el estado s y actuar óptimamente

Supongamos acciones siempre exitosas, $\gamma = 1, H = 100$

$V^*(4,3) =$

$V^*(3,3) =$

$V^*(2,3) =$

$V^*(1,1) =$

$V^*(4,2) =$

La idea de “buenos estados” se puede formalizar a través de la función de valor

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \mid \pi, s_0 = s\right]$$

$V^*(s)$= suma de las recompensas con descuento al empezar en el estado s y actuar óptimamente

Supongamos acciones siempre exitosas, $\gamma = 0{,}9$, $H = 100$

$V^*(4,3) =$

$V^*(3,3) =$

$V^*(2,3) =$

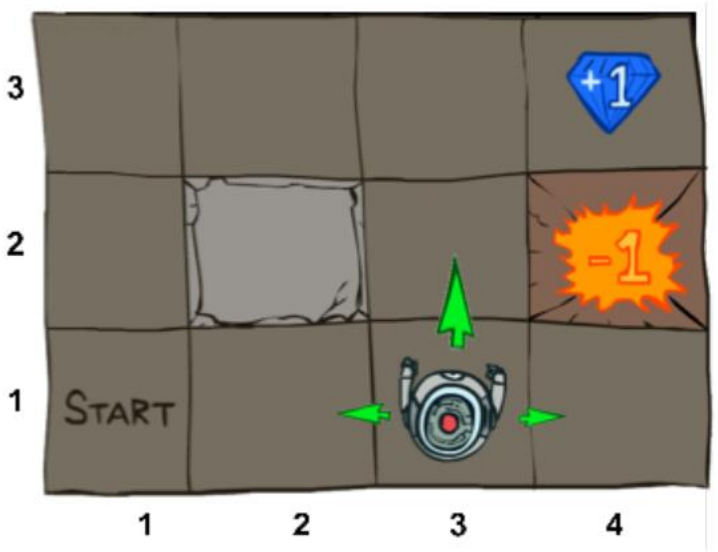
$V^*(1,1) =$

$V^*(4,2) =$

La idea de “buenos estados” se puede formalizar a través de la función de valor

$$V^*(s) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) \mid \pi, s_0 = s\right]$$

$V^*(s)$= suma de las recompensas con descuento al empezar en el estado s y actuar óptimamente



Supongamos acciones con $P = 0{,}8, \gamma = 0{,}9, H = 100$

$V^*(4,3) =$

$V^*(3,3) =$

$V^*(2,3) =$

$V^*(1,1) =$

$V^*(4,2) =$

¿Cómo podemos estimar esta función de valor?

$V_0^*(s)$ = optimal value for state s when H=0

- $V_0^*(s) = 0 \quad \forall s$

$V_1^*(s)$ = optimal value for state s when H=1

- $V_1^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_0^*(s'))$

$V_2^*(s)$ = optimal value for state s when H=2

- $V_2^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_1^*(s'))$

$V_k^*(s)$ = optimal value for state s when H = k

- $V_k^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_{k-1}^*(s'))$

Este simple algoritmo es conocido como *Value Iteration*

Start with $V_0^*(s) = 0$ for all s.

For k = 1, … , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

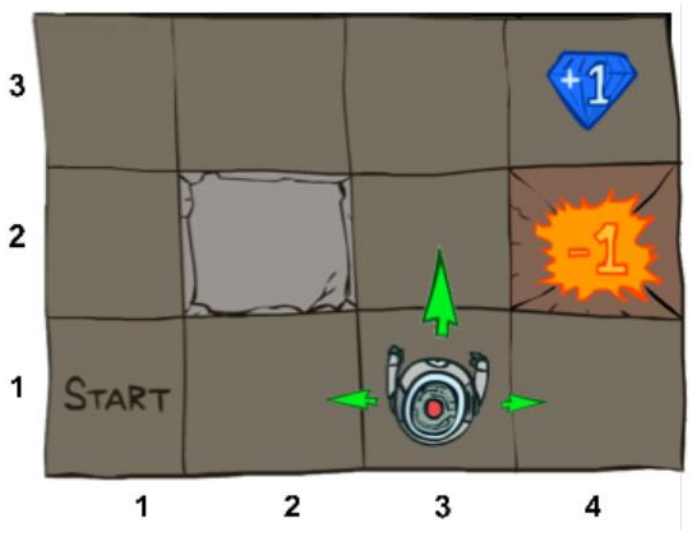
$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

Es posible demostrar que este algoritmo converge a la función de valor óptima $V^*(s)$, la cual satisface las ecuaciones de Bellman:

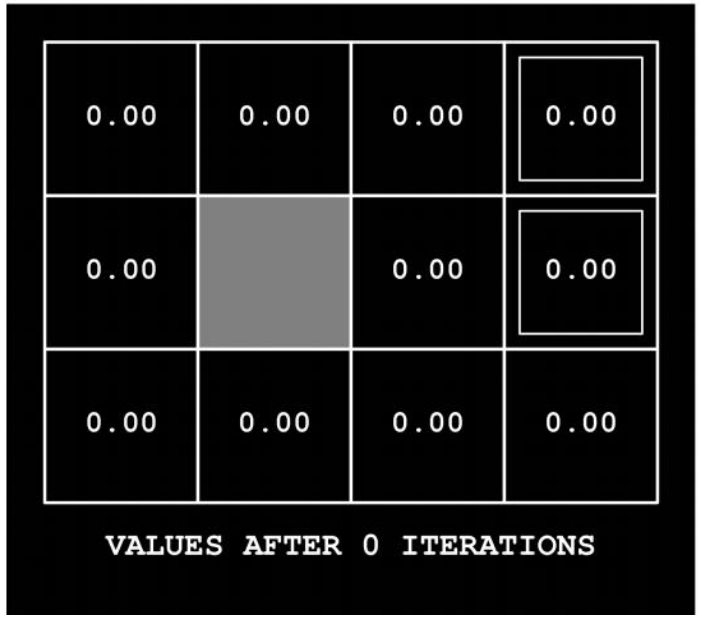
$$\forall\, s \in S: \quad V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

Veamos gráficamente un ejemplo de su ejecución

$$V_0(s) \leftarrow 0$$



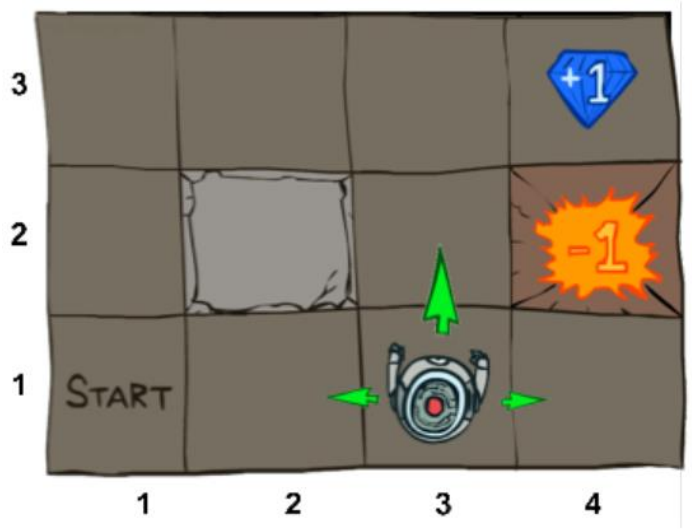
k = 0



Noise = 0.2
Discount = 0.9

Veamos gráficamente un ejemplo de su ejecución

$$V_1(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_0(s'))$$

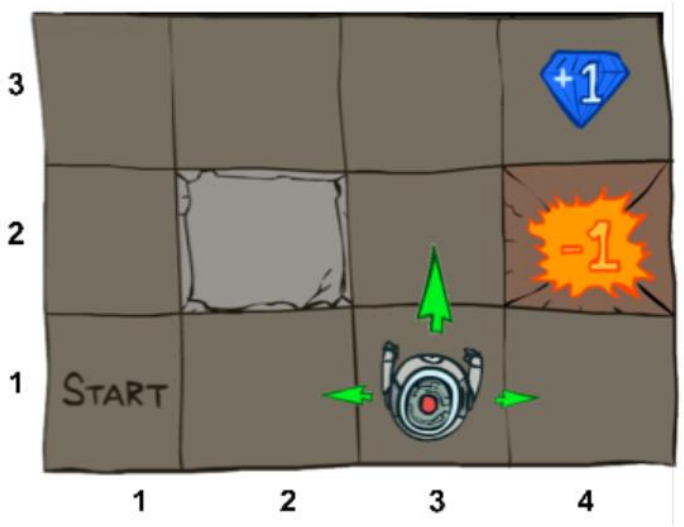


k = 0

Noise = 0.2
Discount = 0.9

Veamos gráficamente un ejemplo de su ejecución

$$V_2(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_1(s'))$$



k = 1

Noise = 0.2
Discount = 0.9

Veamos gráficamente un ejemplo de su ejecución

$$V_2(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_1(s'))$$

k = 2

Noise = 0.2
Discount = 0.9

Veamos gráficamente un ejemplo de su ejecución

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 3

Noise = 0.2
Discount = 0.9

Veamos gráficamente un ejemplo de su ejecución

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 4

Noise = 0.2
Discount = 0.9

Veamos gráficamente un ejemplo de su ejecución

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 5

Noise = 0.2
Discount = 0.9

Veamos gráficamente un ejemplo de su ejecución

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 12

Noise = 0.2
Discount = 0.9

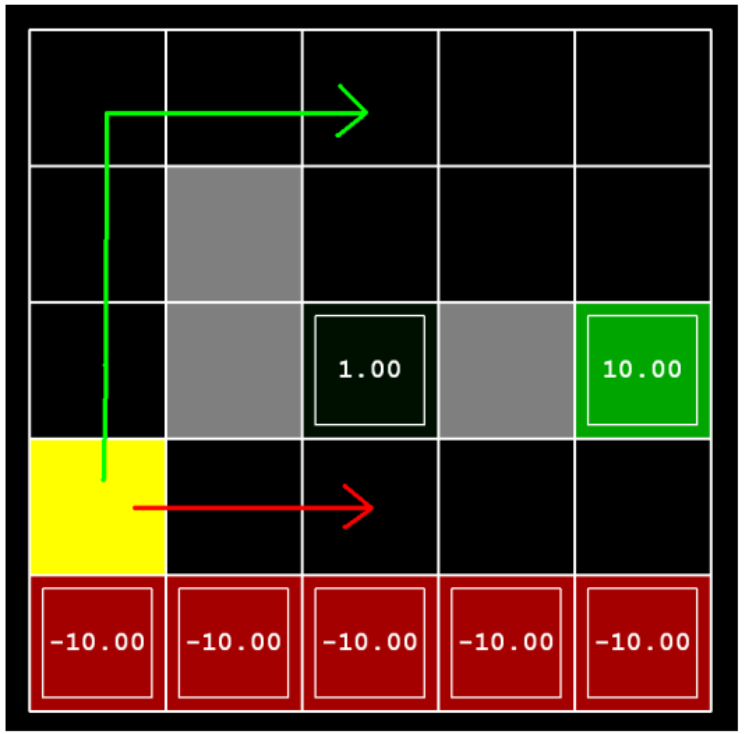
Veamos gráficamente un ejemplo de su ejecución

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_k(s'))$$

k = 100

Noise = 0.2
Discount = 0.9

Antes del ver el código, hagamos un ejercicio: términos pareados



a) Preferir la salida más cercana (+1), pasando cerca del abismo (-10)

b) Preferir la salida más cercana (+1), evitando completamente el abismo (-10)

c) Preferir la salida más lejana (+10), pasando cerca del abismo (-10)

d) Preferir la salida más lejana (+10), evitando completamente el abismo (-10)

1) $\gamma = 0.1,\ p(error) = 0.5$

2) $\gamma = 0.99,\ p(error) = 0$

3) $\gamma = 0.99\ p(error) = 0.5$

4) $\gamma = 0.1,\ p(error) = 0$

Podemos desagregar la función de valor con respecto a las acciones y ahora estimar la función Q

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

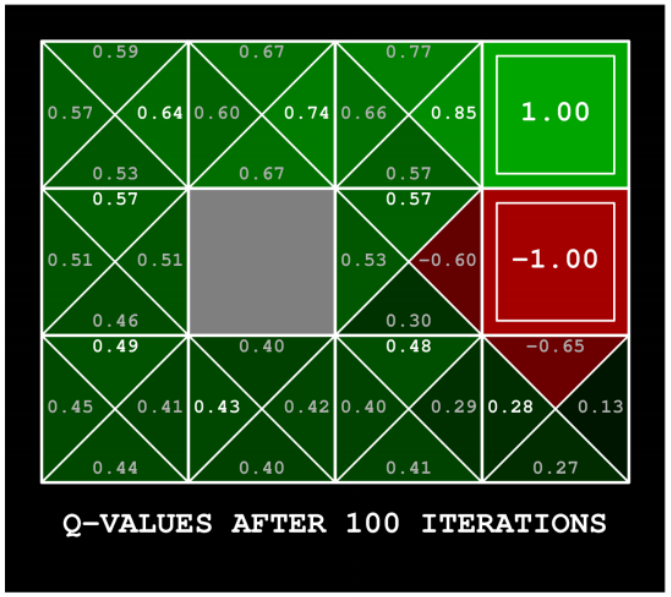
$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$$

- La función $Q^*(s,a)$ estima la utilidad esperada partiendo desde s, tomando la acción a y luego actuando óptimamente
- No es evidente en este momento por qué esto sirve de algo

Podemos refinar la función de valor y estimar la función Q

$$Q^*_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*_k(s',a'))$$

k = 100



Noise = 0.2
Discount = 0.9

¿Qué limitantes tienen estos métodos de estimación de funciones de valor?

- Ecuaciones requieren la existencia de las probabilidades de transición (modelo dinámico del mundo):

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

- Algoritmo requiere visitar y almacenar una gran cantidad de estados, lo que fuerza a tener una cantidad de estados y acciones manejable.

Q-Learning al rescate

- La función Q nos entrega la solución a ambos problemas.
- Al desacoplar la optimalidad del estado y la acción, es posible utilizar un enfoque de muestreo, cambiando las transiciones por un valor esperado:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$$

$$Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$$

Q-Learning al rescate

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \, [\text{target}]$$

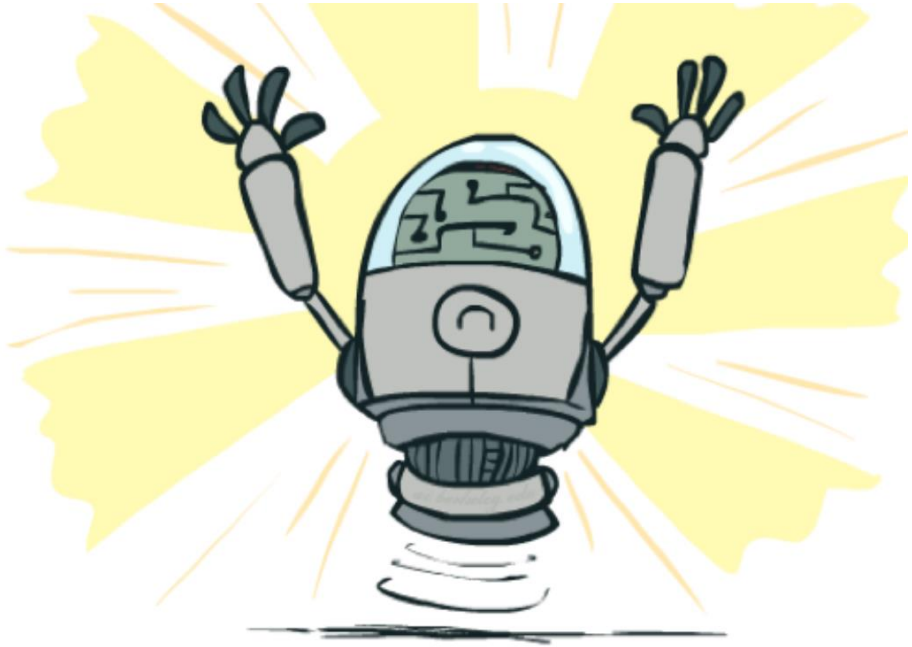
$$s \leftarrow s'$$

La pregunta es ahora cómo muestrear

- Esta es la madre de todas las batallas: exploration vs exploitation
- En otras palabras, ¿elijo la acción optima de acuerdo a $Q_k(s,a)$, o busco nuevas posibilidades?
- En la práctica se utiliza una técnica mixta, ϵ-Greedy: acción al azar con probabilidad ϵ, en otro caso, acción que maximiza $Q_k(s,a)$.

Propiedades de Q-learning

- El principal resultado de Q-Learning es que converge a una política óptima, a pesar de actuar de manera subóptima.
- En otras palabras, desacopla exploración de optimización: off-policy learning.
- Requiere mucha exploración y ajustes cuidadosos del learning rate, pero funciona.



https://youtu.be/AMnW-OsOcl8

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 - Inteligencia Artificial

Value Iteration y Q-Learning

Hans Löbel
Dpto. Ingeniería de Transporte y Logística
Dpto. Ciencia de la Computación