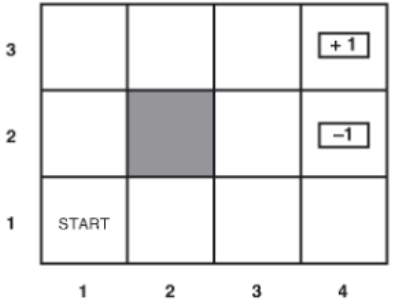
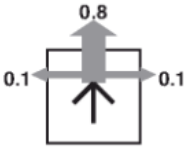

Inteligencia Artificial

Markov Decision Process

- Ambiente secuencial, completamente observable, estocástico
- Modelo de transición Markoviano (la probabilidad de alcanzar s2 desde s1 depende solo de s1 y no de la historia de estados)
- Las recompensas son aditivas (la utilidad es la suma de las recompensas)

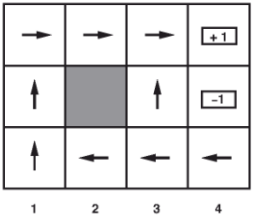


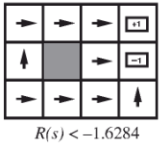
(a)



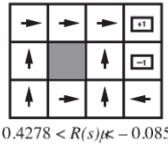
(b)

- Una solución a un MDP es una **política** π
- $\pi(s)$ es la acción recomendada por la política en el estado *s*
- La calidad de una política se mide en términos de la **utilidad esperada**
- Una política **óptima** π^* maximiza la utilidad esperada.

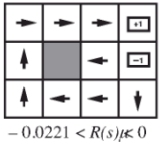




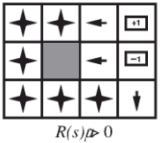
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



$R(s) > 0$

Utilidades en el tiempo.

Las recompensas pueden ser:

- **aditivas**

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots .$$

- **depreciativas**

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

Donde γes el factor de descuento, un valor entre 0 y 1.

Políticas óptimas y utilidad de los estados.

La utilidad esperada ejecutando πen un estado inicial *s* es:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

Una política óptima estaría dada por:

$$\pi_s^* = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(s)$$

El agente selecciona la acción en base al principio de maximización de la utilidad esperada

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' \mid s, a) U(s')$$

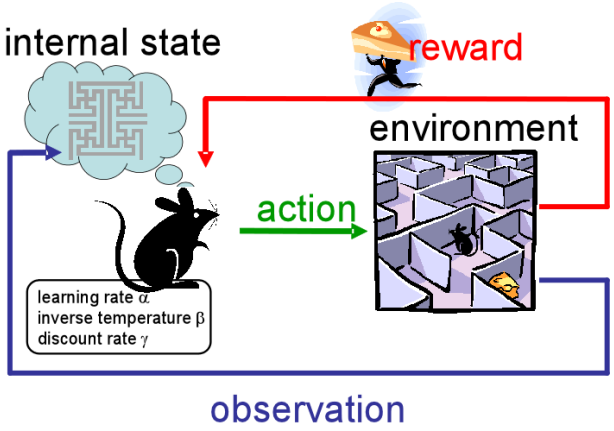
Algoritmos para encontrar políticas óptimas

- Value iteration
- Policy iteration

Ambos métodos hacen pequeños ajustes por cada iteración para calcular las utilidades de cada estado.

Ambos necesitan conocer el modelo de transición y la función **R(s)**

Aprendizaje por refuerzo



La tarea del aprendizaje por refuerzo es usar los refuerzos recibidos para aprender una política óptima para el ambiente.

Vamos a considerar 3 diseños de agente:

- **Agente basado en utilidad:** aprende una función de utilidad basada en los estados. Necesita conocer un **modelo del ambiente**
- **Agente Q-learning:** aprende una función (**Q-function**) de utilidad para las acciones disponibles. No necesita modelo del ambiente
- **Agente reflejo:** aprende una política que mapea estados en acciones

Dos posibles tipos de aprendizaje:

- **Aprendizaje pasivo:** tiene una política fija, solo aprendemos las utilidades
- **Aprendizaje activo:** el agente tiene que aprender que hacer

Hay además varios métodos de resolución:

- Programación dinámica
- Métodos de montecarlo
- **Aprendizaje por diferencia de tiempo (TD)**

TD learning en aprendizaje pasivo

Cuando ocurre una transición desde un estado *s'* a otro estado *s* se aplica la siguiente regla de actualización:

$$U^{\pi}(s) = U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

function PASSIVE-TD-AGENT(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r'

persistent: π, a fixed policy
U, a table of utilities, initially empty
N_s, a table of frequencies for states, initially zero
s, a, r, the previous state, action, and reward, initially null

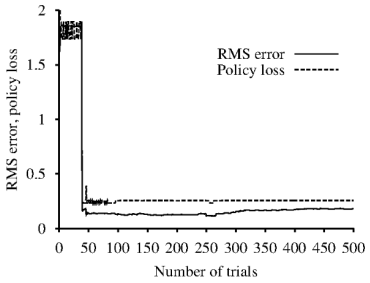
if s' is new **then** $U[s'] \leftarrow r'$
if s is not null **then**
increment $N_s[s]$
$U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
return a

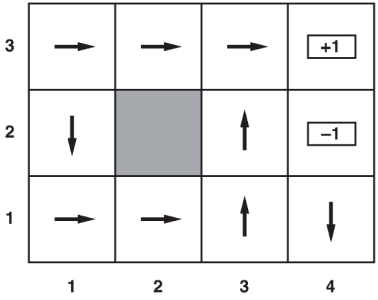
Aprendizaje activo.

- El agente no posee una política fija; debe aprenderla
- Cuando el agente encuentra que determinado comportamiento es bueno ¿debe siempre ejecutarlo?

Exploración vs Explotación

Comportamiento de un agente avaro:





Un agente debiera hacer un balanceo entre **explotar** los conocimientos que posee para maximizar las recompensas y **explorar** para mejorar los conocimientos y maximizar la utilidad esperada a largo plazo.

Función de exploración.

Dadas las acciones disponibles, devuelve que acción tomar dependiendo de las utilidades de las mismas y las frecuencias de aparición.

Debe seguir el principio **GLIE** (Greedy in the Limit of Infinite Exploration)

- Una definición simple:

$$f(u,n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

- Ecuación de Boltzmann:

$$\exp\left(\frac{\mathrm{Q}^{est}(s,a)}{\mathrm{K}_t}\right)$$

Aprendiendo Q-Functions

Q-learning: aprende utilidades por acción. No necesita modelo del ambiente.

function Q-LEARNING-AGENT($percept$) **returns** an action
  inputs: $percept$, a percept indicating the current state s' and reward signal r'
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, r, the previous state, action, and reward, initially null

  if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return a

Un algoritmo similar a Q-Learning es **SARSA** (State-Action-Reward-State-Action). Solo cambia la regla de actualización:

- Regla de Q-Learning:

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma max_{a'} Q(s',a') - Q(s,a))$$

- Regla de SARSA:

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma Q(s',a') - Q(s,a))$$

Bibliografía y enlaces útiles.

- Russell S., Norvig P.: Artificial Intelligence: A modern Approach. Third Edition. Chapter 21.
- Sutton R., Barto A.: Reinforcement Learning: An introduction.