Intel·ligència Artificial

Cinc Cèntims de IA

Mario Vilar

6 de febrer de 2024

Universitat de Barcelona

Índex

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) & \text{if } n \geq 2 \end{cases}$$

1

Processos de decisió de Markov (MDP)

The learner and decision maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent.

Definició 1.1 (MDP**).** Un MDP se define por:

1. Un conjunto de estados S, un estado inicial y posiblemente un estado terminal.
2. Un conjunto de acciones A.
3. Una función de transición, $T : A \times S \times S \longrightarrow R$, $T(a, s, s')$ la probabilidad de ir al estado s' cuando estamos en el estado s y realizamos la acción a y $T(a, s, s') = P(s'|s, a)$ (the probability of reaching state s' if action a is done in state s).
4. To complete the definition of the task environment, we must specify the utility function for the agent. Because the decision problem is *sequential*, the utility function will depend on a sequence of states —an environment history—rather than on a single state. Función de recompensa $R(s, a, s')$, en ocasiones $R(s)$[1].

The MDP framework is abstract and flexible and can be applied to many different problems in many different ways. En problemas de búsqueda *deterministas* (el cas d'A^*, el minimax...) en que interviene un único agente, queremos un **plan óptimo**, una secuencia de acciones desde el inicio hasta un objetivo. En un MDP queremos una *política* (veure 1.5) óptima $\pi^* : S \longrightarrow A$.

Exemple 1.2 (Problema no determinista**).** El agente vive en una rejilla. El bloque oscuro impide el paso. Las acciones del agente no siempre tienen el mismo resultado. Si quiere ir hacia el norte: 80% de las veces va al norte, 10% al este, 10% al oeste. Puede recibir una pequeña **recompensa** por sobrevivir, la mayor recompensa llega al final. *Objetivo: maximizar la suma de recompensas*. If the environment were deterministic, a solution would be easy: [*Up, Up, Right, Right, Right*]. Unfortunately, the environment won't always go along with this solution, because *the actions are unreliable*: In such an environment, the sequence [*Up, Up, Right, Right, Right*] goes up around the barrier and reaches the goal state at $(4, 3)$ with probability $0.8^5 = 0.32768$. There is also a small chance of accidentally reaching the goal by going the other way around with probability $0.1^4 \cdot 0.8$, for a grand total of 0.32776.

1. Estados: $S = \{(x, y) \mid x \in \{1, \ldots, 4\}, y \in \{1, 2, 3\}\}$, inicial $s_{ini} = (1, 1)$ y terminales $s_1 = (4, 3)$ y $s_2 = (4, 2)$.
2. Acciones: $A = \{\text{up, down, right, left}\}$.

[1] Some definitions of MDPs allow the reward to depend on the action and outcome too, so the reward function is $R(s, a, s')$. This simplifies the description of some environments but does not change the problem in any fundamental way.

3. Función de transición: $T(s, a, s') = P(s'|s, a)$. Això és, per exemple, si estem en s_1 i triem $a = \{\text{up}\}$, la probabilitat que s' sigui l'estat immediatament superior és de 0.8, que sigui, en canvi, el de l'esquerra o el de la dreta és de 0.1, 0.1, respectivament.
4. Función de recompensa, $R(s, a, s')$, $R(s)$: $R(s_1) = 1$, $R(s_2) = -1$, i $R(s) = -0.04$ quan $s \neq s_1, s_2$.

Observació 1.3 (Mundo rejilla). Política óptima para el mundo rejilla cuando $R(s) = -0.04$ para todos los estados no terminales.

- En $(3, 2)$, en el mejor de los casos $-0.04 + 1 = 0.96$.
- En $(3, 1)$, en el mejor de los casos $-0.04 \cdot 6 + 1 = 0.76$. Si fuéramos hacia el N seria $-0.04 \cdot 2 + 1 = 0.92$ pero nos arriesgamos a tener $-0.04 - 1 = -1.04$.

Pel que fa als possibles valors de $R(s)$:

- When $R(s) \leq -1.6284$, life is so painful that the agent heads straight for the nearest exit, even if the exit is worth -1.
- When $-0.4278 \leq R(s) \leq -0.0850$, life is quite unpleasant; the agent takes the shortest route to the $+1$ state and is willing to risk falling into the -1 state by accident. In particular, the agent takes the shortcut from $(3, 1)$.
- When life is only slightly dreary ($-0.0221 < R(s) < 0$), the optimal policy takes no risks at all. In $(4, 1)$ and $(3, 2)$, the agent heads directly away from the -1 state so that it cannot fall in by accident, even though this means banging its head against the wall quite a few times.
- Finally, if $R(s) > 0$, then life is positively enjoyable and the agent avoids both exits. As long as the actions in $(4, 1)$, $(3, 2)$ and $(3, 3)$ are as shown, every policy is optimal, and the agent obtains infinite total reward because it never enters a terminal state. Surprisingly, it turns out that there are six other optimal policies for various ranges of $R(s)$.

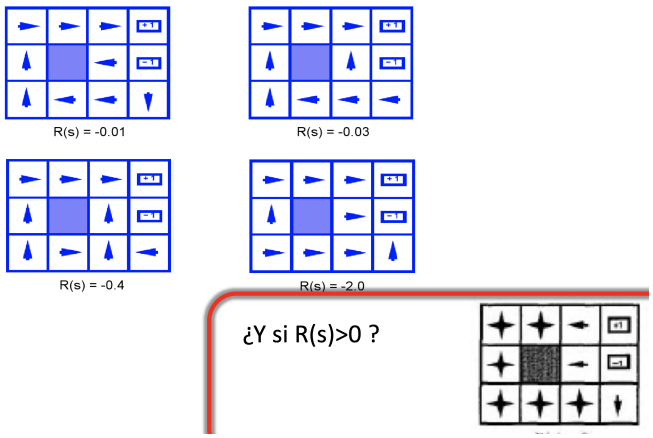

Figura 1: Diferents polítiques per a diferents taules de recompensa.

1. Los procesos de decisión de Markov son una familia de problemas de búsqueda no deterministas.
2. El aprendizaje por refuerzo es un PDM del que desconocemos la función de recompensa o la función de transición.

Observació 1.4.

1. Segons Andréi Markov (1856-1922), el estado actual resume toda la información relevante del pasado. En el caso de los procesos de decisión de Markov, esto quiere decir que:

$$P(S_{t+1}|S_t, a_t, S_{t-1}, a_{t-1}, \ldots, S_0, a_0) = P(S_{t+1}|S_t, a_t).$$

2. ¿Podemos usar expectimax? No, porque en expectimax las recompensas llegan únicamente al final del juego, y en expectimax el juego termina en algún momento. Cada estado del MDP genera un árbol de búsqueda expectimax.

1.1 Polítiques

Definició 1.5 (Política**).** We have seen that any fixed action sequence won't solve the problem, because the agent might end up in a state other than the goal. Therefore, a solution must specify what the agent should do for any state that the agent might reach. A solution of this kind is called a *policy*. Una política asigna una acción a cada estado $\pi(s) = a$. If the agent has a *complete policy*, then no matter what the outcome of any action, the agent will *always* know what to do next.

Each time a given policy is executed starting from the initial state, the stochastic nature of the environment may lead to a different environment history.

Definició 1.6 (Política òptima**).** The quality of a policy is measured by the *expected utility* of the possible environment histories generated by that policy. An optimal policy is a policy that yields the highest expected utility (una política óptima maximiza la utilidad esperada si se sigue). El resultado es un agente reflejo.

1.2 Utilitat d'una seqüència d'accions

Para formalizar la optimalidad de una política, necesitamos entender la optimalidad de una secuencia de acciones.

Definició 1.7 (Preferències estacionàries**).** If two state sequences $[s_0, s_1, s_2, \ldots]$ and $[s'_0, s'_1, s'_2, \ldots]$ begin with the same state (i.e., $s_0 = s'_0$), then the two sequences should be preference-ordered the same way as the sequences $[s_1, s_2, \ldots]$ and $[s'_1, s'_2, \ldots]$. *If you prefer one future to another starting tomorrow, then you should still prefer that future if it were to start today instead*.

Teorema 1.8. *Si las preferencias son estacionarias únicamente existen dos maneras de definir la utilidad de una secuencia de acciones:*

1. *Utilidad aditiva:* $U([r_0, r_1, r_2 \dots]) = r_0 + r_1 + r_2 + \cdots$. *L'exemple 1.2 usa aquesta additivitat.*
2. *Utilidad aditiva con descuento:* $U([r_0, r_1, r_2 \dots]) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$. *The discount factor describes the preference of an agent for current rewards over future rewards. When* γ *is close to 0, rewards in the distant future are viewed as insignificant. When* γ *is 1, discounted rewards are exactly equivalent to additive rewards, so additive rewards are a special case of discounted rewards.*

Problema 1.9. *Las secuencias de acciones infinitas pueden tener una utilidad infinita. En més detall, if the environment does not contain a terminal state, or if the agent never reaches one, then all environment histories will be infinitely long, and utilities with additive, undiscounted rewards will generally be infinite.*

Solució.

1. *Horizonte finito*[2]. Establecemos que sólo se jugará al juego durante T unidades de tiempo. Da lugar a políticas no estacionarias (π^* depende del tiempo que queda).
2. *Estado absorbente* o *proper policy*: If the environment contains terminal state, garantizamos que para cualquier política se alcance siempre un nodo terminal, then we will *never* need to compare infinite sequences. A policy that is guaranteed to reach a terminal state is called a *proper policy*. With proper policies, we can use $\gamma = 1$.
3. *Descontar*: Dado un $0 < \gamma < 1$,

$$U_0([r_0, \dots, r_\infty]) = \sum_{t=0}^{\infty} \gamma^t r_t \leq \frac{R_{max}}{1-\gamma}.$$

Recordemos (cf. 1.8) que un γ más pequeño indica que estamos mucho más interesados en los estados más inmediatos que en los que llegarán más tarde. Normalmente se descuenta por $\gamma < 1$ cada paso de tiempo. Las recompensas más cercanas en el tiempo tienen una utilidad mayor que las que tardarán más en llegar. ■

1.3 Valor óptimo de un estado

Ahora queremos calcular el valor óptimo de cada estado.

1. $V^*(s)$ es el valor que obtendríamos si empezáramos en s y aplicáramos la política óptima.

$$V^*(s) = \max_{a \in A} Q^*(s, a) \tag{1.1}$$

[2] A finite horizon means that there is a fixed time N after which nothing matters—the game is over, so to speak. Thus, $U([r_0, r_1, \dots, s_{N+k}]) = U([r_0, r_1, \dots, s_N])$, for all $k > 0$.

2. $Q^*(s, a)$ es el valor que obtendríamos si empezáramos en s, hiciéramos la acción a y después aplicáramos la política óptima.

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right].$$

Per tant, usant (1.1):

$$V^*(s) = \max_{a \in A} \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right].$$

3. $\pi^*(s)$ es la acción óptima cuando nos encontramos en el estado s.

1.4 Iteració de valors

La definición de «utilidad óptima» nos lleva a pensar que obtenemos la recompensa óptima maximizando sobre la primera acción y siguiendo la política óptima a partir de ahí. Ahora, queremos encontrar una política óptima π^*. Tenim diverses maneres d'afrontar el problema.

1. Podemos hacer una búsqueda expectimax modificada empezando en el estado s. The utility function $U(s)$ allows the agent to select actions by using the principle of maximum expected utility, that is, choose the action that maximizes the expected utility of the subsequent state:

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s'} P(s'|s, a) U(s').$$

2. **Value iteration**, *equacions de Bellman*.

Definició 1.10 (Value iteration**).** The basic idea is to calculate the utility of each state and then use the state utilities to select an optimal action in each state. Computar los valores óptimos para todos los estados al mismo tiempo utilizando aproximaciones sucesivas. Una vez terminamos, no necesitamos replanificar.

There is a direct relationship between the utility of a state and the utility of its *neighbors*: the utility of a state is the *immediate reward* for that state plus the *expected discounted utility of the next state*, assuming that the agent chooses the optimal action. Cuando descontamos, las recompensas que están lejos se vuelven negligibles. Si desde cualquier parte se puede alcanzar un estado terminal, la fracción de episodios que no terminan se convierte en negligible. En otro caso, podríamos tener utilidad infinita y la aproximación no funcionará.

$$U(s) = R(s) + \gamma \max_{a \in A} \sum_{s'} P(s'|s, a) U(s').$$

The **Bellman equation** is the basis of the value iteration algorithm for solving MDPs. If there are n possible states, then there are n Bellman equations, *one for each state*. The n equations contain n unknowns —the utilities of the states. One thing to try is an iterative approach.

Algorisme 1.11. *Para cada estado s calculamos estimaciones $V_k^*(s)$[3]. Empezar con $V_0^*(s) = 0$. Dado V_i^*, calcular el valor de todos los estados a profundidad $i+1$.*

$$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V_i(s')).$$

Ésta es la actualización de Bellman. Repetir hasta que converja.

Teorema 1.12. *El algoritmo converge a valores óptimos únicos. Las políticas pueden converger mucho antes de que lo hagan los valores.*

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Figura 2: Algorisme d'iteració de valors, [RNC22].

1.5 Avaluació i iteració de polítiques

Once a policy, π, has been improved to yield a better policy, π'. We can thus obtain a sequence of monotonically improving policies and value functions:

$$\pi_0 \longrightarrow \pi_1 \longrightarrow \pi_2 \longrightarrow \cdots \longrightarrow \pi^*.$$

Because a finite MDP has only a finite number of policies, this process must converge to an optimal policy and optimal value function in a finite number of iterations.

Problema 1.13 (Evaluación de políticas**).** *Ahora queremos calcular el valor de un estado en una política (en general, no necesariamente óptima) dada.*

Soluciones. We can compare policies by comparing the *expected utilities* obtained when executing them. $V_\pi(s)$ es la suma total de recompensas *esperadas* a partir de s si seguimos la política π. We assume the agent

[3] No son el valor óptimo de s. Son el valor óptimo considerando k recompensas. Cuando $k \to \infty$, se aproxima al valor óptimo.

is in some initial state s and define S_t (a random variable) to be the state the agent reaches at time t when executing a particular policy π:

$$V^\pi(s) = E\left(\sum_{t=0}^{\infty} \gamma^t R(S_t)\right),\ V_0^\pi(s) = 0 \text{ i } V_{i+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V_i^\pi(s'))$$

Hem usat una variació de les equacions de Bellman. Las actualizaciones de Bellman simplificadas nos permiten calcular V para una política preestablecida, necesitamos T y R. ■

Observació 1.14. Remember that π_s^* (política òptima en un estat s) is a policy, so it recommends an action for every state; its connection with s in particular is that it's an optimal policy when s is the starting state. A remarkable consequence of using *discounted utilities* with *infinite* horizons is that *the optimal policy is independent of the starting state*. Of course, the action sequence won't be independent; remember that a policy is a function specifying an action for each state.

Problemas de la iteración de valores: considerar todas las acciones en cada iteración es lento. Tarda $|A|$ veces más tiempo que la evaluación de una política, y en ese tiempo la política no cambia; in practice, it often occurs that π_i becomes optimal long before V_i has converged.

Definició 1.15 (Policy iteration**).** The policy iteration algorithm alternates the following two steps, beginning from some initial policy π_0:

1. *Policy evaluation:* Given a policy π_i, calculate $V_i = V^{\pi_i}$, the utility of each state if π_i were to be executed. Aplicando las ecuaciones de Bellman simplificadas de forma iterativa:

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} T(s, \pi_k(s), s')(R(s, \pi_k(s), s') + \gamma V_i^{\pi_k}(s')).$$

2. *Policy improvement:* Calculate a new policy π_{i+1} using one-step look-ahead based on V_i (a partir de los valores encontrados en el paso anterior).

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} T(s, a, s')(R(s, a, s') + \gamma V^{\pi_k}(s')).$$

The algorithm terminates when the policy improvement step yields no change in the utilities. At this point, we know that the utility function U_i is a solution to the Bellman equations, and π_i must be an optimal policy.

Observació 1.16 (Value iteration vs. Policy iteration**).** En iteración de valores, cada paso modifica tanto los **valores** de cada estado (explícitamente) como la **política**. En cambio, en la iteración de políticas en cada iteración se modifica (únicamente) la política y necesitamos varias iteraciones para calcular las utilidades de una política determinada. If policy evaluation is done iteratively, then convergence exactly to v_π occurs only in the limit.

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      $a \leftarrow \pi(s)$
      $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
      If $a \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return V and π; else go to 2

Figura 3: Algorisme d'iteració de polítiques, [RNC22].

2

Aprenentatge per reforç (RL)

In the absence of feedback from a teacher, an agent can learn a transition model for its own moves and can perhaps learn to predict the opponent's moves, but without some feedback about what is good and what is bad, the agent will have no grounds for deciding which move to make. *This kind of feedback is called a reward, or reinforcement*. In games like chess, the reinforcement is received *only at the end of the game*.

An optimal policy is a policy that maximizes the expected total reward. The task of reinforcement learning is to use observed rewards to learn an optimal (or nearly optimal) policy for the environment.

1. Recibimos información en forma de recompensa.
2. La utilidad del agente viene dada por la función de recompensa.
3. El agente debe aprender a actuar para maximizar la recompensa esperada.

Observació 2.1 (Diferents tipus d'agents que hem vist fins ara**).**

1. A utility-based agent learns a utility function on states and uses it to select actions that maximize the expected outcome utility. It must also have a model of the environment in order to make decisions, because it must know the states to which its actions will lead.
2. A Q-learning agent *learns* an action-utility function, or Q-function, giving the expected utility of

taking a given action in a given state. A Q-learning agent, on the other hand, can compare the expected utilities for its available choices *without needing to know their outcomes*, so it does not need a model of the environment[4].

3. A reflex agent learns a policy that maps directly from states to actions.

Anem a explicar una mica més l'agent Q-learning. Seguimos modelando nuestro mundo mediante un PDM, y buscando una política óptima π^*:

1. S: conjunto de estados.
2. A: conjunto de acciones.
3. $T : S \times A \times S \longrightarrow \mathfrak{R}$: función de transición.
4. $T : S \times A \times S \longrightarrow \mathfrak{R}$: función de recompensa.

Pero ahora no conocemos a priori ni T ni R. Es decir, no sabemos ni qué estados son buenos ni lo que hacen las acciones. En definitiva, tenemos que intentar cosas para aprender sus consecuencias.

2.1 Aprenentatge passiu

Definició 2.2 (Passive learning**).** No conocemos las transiciones $T(s, a, s')$, no conocemos las recompensas R. The agent's policy is fixed and the task is to *learn the utilities of states* (or state–action pairs), $V(s)$.

The principal issue is exploration: an agent must experience as much as possible of its environment in order to learn how to behave in it. We start with the case of a passive learning agent using a state-based representation in a fully observable environment. The agent executes a set of trials in the environment using its policy π. The object is to use the information about rewards to learn the expected utility $V^\pi(s)$ associated with each nonterminal state s.

2.1.1 Estimació directa de la utilitat

A simple method for direct utility estimation was invented, the idea is that the utility of a state is the expected total reward from that state onward (called the expected reward-to-go), and each trial provides a sample of this quantity for each state visited. La *percepción* determina el estado y la recompensa actual. Thus, at the end of each sequence, the algorithm calculates the observed reward-to-go for each state and updates the estimated utility for that state accordingly, just by keeping a running average for each state in a table.

Observació 2.3. Direct utility estimation succeeds in reducing the reinforcement learning problem to an inductive learning problem, about which much is known. Unfortunately, it misses a very important source

[4] On the other hand, because they do not know where their actions lead, Q-learning agents cannot look ahead; this can seriously restrict their ability to learn, as we shall see.

of information, namely, the fact that the utilities of states are not independent! The utility of each state equals its own reward plus the expected utility of its successor states.

$$V^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s').$$

function PASSIVE-ADP-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    mdp, an MDP with model P, rewards R, discount γ
    U, a table of utilities, initially empty
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
    s, a, the previous state and action, initially null

  if s' is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
    for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**
      $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \,/\, N_{sa}[s, a]$
  $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
  if s'.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$
  return a

Figura 4: A passive reinforcement learning agent.

2.1.2 ADP

The process of learning the model itself is easy, because the environment is fully observable, by learning the transition model that connects them and solving the corresponding Markov decision process using a dynamic programming method. Estimar el MDP a partir de las observaciones.

Determinar el valor de cada estado resolviendo el MDP estimado (plugging the learned transition model $P(s'|s, \pi(s))$ and the observed rewards $R(s)$ into the Bellman equations, to calculate the utilities of the states).

Caso más sencillo:

1. Contar los estados a los que se llega para cada s, a.
2. Estimar $T(s, a, s')$ y descubrir $R(s, a, s')$ cada vez que ocurra.

2.1.3 Temporal Difference (TD)

Use the observed transitions to *adjust the utilities* of the observed states so that they agree with the constraint equations. All temporal-difference methods work by adjusting the utility estimates towards the ideal equilibrium that holds locally when the utility estimates are correct. More generally, when a transition occurs from state s to state s', we apply the following update to $V^{\pi}(s)$:

$$\text{sample} = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$
$$V^{\pi}(s) \leftarrow (1 - \alpha) V^{\pi}(s) + \alpha \cdot \text{sample}$$
$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(\text{sample} - V^{\pi}(s)).$$

Here, α is the learning rate parameter. ¿Podemos reemplazar la esperanza con la media? Podemos estimar a partir de las muestras que tenemos sin necesidad de construir un modelo (TD es libre de modelo para aprendizaje pasivo):

$$V^{\pi}_{i+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')(R(s, \pi(s), s') + \gamma V^{\pi}_{i}(s'))$$

En TD no podemos utilizar los valores de los estados obtenidos para generar un política óptima.

Observació 2.4 (ADP vs. TD**).** The ADP approach and the TD approach are actually closely related. Both try to make local adjustments to the utility estimates in order to make each state *agree* with its successors.

1. One difference is that TD adjusts a state to agree with its *observed* successor, whereas ADP adjusts the state to agree with *all* of the successors that might occur, weighted by their probabilities.
2. A more important difference is that whereas TD makes a single adjustment per observed transition, ADP makes as many as it needs to restore consistency between the utility estimates U and the environment model P.
3. TD does not learn quite as fast as the ADP agent and shows much higher variability, but it is much simpler and requires much less computation per observation. Notice that TD does not need a transition model to perform its updates.
4. Once again, it can be shown that the TD algorithm will converge to the same values as ADP as the number of training sequences tends to infinity.

2.2 Aprenentatge per reforç actiu

No conocemos las transiciones $T(s, a, s')$, no conocemos las recompensas $R(s, a, s')$. *A passive learning agent has a fixed policy that determines its behavior. An active agent must decide what actions to take*. Es decir, podemos escoger las acciones que queramos. The utilities it needs to learn are those defined by the optimal policy. Similar a la iteración de valores o de políticas, pero sin conocer T ni R.

Definició 2.5 (Greedy Agent**).** The agent does not learn the true utilities or the true optimal policy. Balance exploration and exploitation by choosing between exploration and exploitation.

Definició 2.6 (Exploitation vs. Exploration**).** An agent therefore must make a tradeoff between *exploitation* to maximize its reward —as reflected in its current utility estimates— and *exploration* to maximize its long-term well-being.

1. Pure *exploitation* risks getting stuck in a rut (podemos ir a parar a acciones que ya sepamos que son malas).
2. Pure *exploration* to improve one's knowledge is of no use if one never puts that knowledge into practice (es refereix a què si ja tenim un camí millor, no l'utilitzarem mai).

Existen diferentes esquemas para forzar la exploración. El más simple, la selección aleatoria de acciones.

Definició 2.7 (ε-greedy). Lanzamos una moneda antes de elegir qué acción realizar. Con probabilidad $1-\varepsilon$ escogemos la mejor opción según los Q-valores actuales. Con probabilidad ε escogemos una acción al azar.

Problema: Exploramos todo el espacio pero lo seguimos haciendo una vez que ya hemos aprendido. Soluciones:

- Disminuir ε con el tiempo.
- Funciones de exploración (intentar acciones que se han probado poco evitando acciones que parezcan tener poca utilidad).

Explorar áreas de las que aún no tenemos información. ¿Cómo?

Definició 2.8 (Exploration function). La denotem per $f(u,n)$. Recibe una estimación del valor del estado y un contador del número de veces que hemos estado. It determines how greed (preference for high values of u) is traded off against curiosity (preference for actions that have not been tried often and have low n). Posem $f(u,n) = u + \frac{k}{n}$[5].

$$Q_{i+1}(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} Q_i(s',a').$$
$$Q_{i+1}(s,a) \leftarrow_\alpha R(s,a,s') + \gamma \max_{a'} f(Q_i(s',a'), N(s',a')).$$

Definició 2.9 (Q-learning). There is an alternative TD method, called Q-learning, which learns an action-utility representation instead of learning utilities. Iteración de Q-valores basada en muestreo.

Definició 2.10 (Q-values, Q-functions). We will use the notation $Q(s,a)$ to denote the value of doing action a in state s. Q-values are directly related to utility values as follows:

$$V(s) = \max_a Q(s,a).$$

Q-functions are a way of storing utility information with a very important property: it does not need a model[6]. Dos *approaches* possibles:

1. When the Q-values are correct, it must hold at equilibrium:

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

This does, however, require that a model also be learned, because the equation uses $P(s'|s,a)$.

[5] The function $f(u,n)$ should be increasing in u and decreasing in n. Obviously, there are many possible functions that fit these conditions.

[6] La cita sencera de [RNC22] és la següent: «A TD agent that learns a Q-function does not need a model of the form $P(s'|s,a)$, either for learning or for action selection.»

2. *TD Q-learning*:

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

This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated.

Observació 2.11. A classe fem una petita reescriptura dels termes per contemplar un nou terme, *sample*, que es refereix a la informació rebuda a cada iteració (s,a):

$$\begin{aligned} \text{sample} &= R(s,a,s') + \gamma \max_{a'} Q(s',a'). \\ Q(s,a) &\leftarrow (1-\alpha)Q(s,a) + \alpha \cdot \text{sample} \end{aligned} \tag{2.1}$$

Observació 2.12. Q-learning *eventually learns the optimal policy* (si exploras bastante, si la ratio de aprendizaje es suficientemente baja, básicamente es idependiente de cómo se seleccionan las acciones), but do so at a much slower rate than the ADP agent. This is because the local updates do not enforce consistency among all the Q-values via the model.

Teorema 2.13. *En el context de l'actualització dels Q-values, (2.1), si posem $\alpha = \frac{1}{N}$, N el nombre d'iteracions, Q_N es calcularà mitjançant la mitjana.*

Demostració. Ho farem per inducció. Prenem $s_n = \sum_{i\in\{1,\dots,N\}} \frac{n_i}{N}$. Calculem Q_1 i Q_2:

$$\begin{aligned} Q_1 &= \left(1 - \frac{1}{1}\right)\cdot Q_0 + \frac{1}{1}n_1 = n_1 \\ Q_2 &= \left(1 - \frac{1}{2}\right)\cdot Q_1 + \frac{1}{1}n_2 = \frac{1}{2}n_1 + \frac{1}{2}n_2 = \frac{1}{2}(n_1+n_2). \end{aligned}$$

Certament, podem suposar Q_N i provar-ho per a Q_{N+1} a partir de:

$$Q_{N+1} = \left(1 - \frac{1}{N+1}\right)\cdot Q_N + \frac{1}{N+1}n_{N+1} = \frac{N}{N+1}\left(\sum_{i=1}^{N}\frac{n_i}{N}\right) + \frac{1}{N+1}n_{N+1} = \sum_{i=1}^{N+1}\frac{n_i}{N+1}. \qquad \blacksquare$$

When the learning agent is responsible for selecting actions while it learns, it must trade off the estimated value of those actions against the potential for learning useful new information. An exact solution of the exploration problem is infeasible, but some simple heuristics do a reasonable job.

Referències

[CDT19] European Commission, Content Directorate-General for Communications Networks i Technology. *Ethics guidelines for trustworthy AI*. Publications Office, 2019. doi: doi/10.2759/346720.

[RNC22] Stuart J. (Stuart Jonathan) Russell, Peter Norvig i Ming-Wei Chang. *Artificial intelligence : a modern approach*. eng. Fourth Edition, Global Edition. Pearsons series in artificial intelligence. Harlow: Pearson Education Limited, 2022. isbn: 9781292401133.

[Viq23] Viquipèdia. *Aprenentatge profund — Viquipèdia, l'Enciclopèdia Lliure*. [Internet; descarregat 31-octubre-2023]. 2023. url: %5Curl%7B//ca.wikipedia.org/w/index.php?title=Aprenentatge_profund&oldid=32624467%7D.

[Wik23a] Wikipedia. *Prueba de Turing — Wikipedia, La enciclopedia libre*. [Internet; descargado 11-octubre-2023]. 2023. url: %5Curl%7Bhttps://es.wikipedia.org/w/index.php?title=Prueba_de_Turing&oldid=154540713%7D.

[Wik23b] Wikipedia contributors. *Evaluation function — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Evaluation_function&oldid=1160711683. [Online; accessed 4-November-2023]. 2023.