Ayudantía 11: Aprendizaje Reforzado

Tomás Couso
Sebastián Guerra

Markov decision process (MDP)

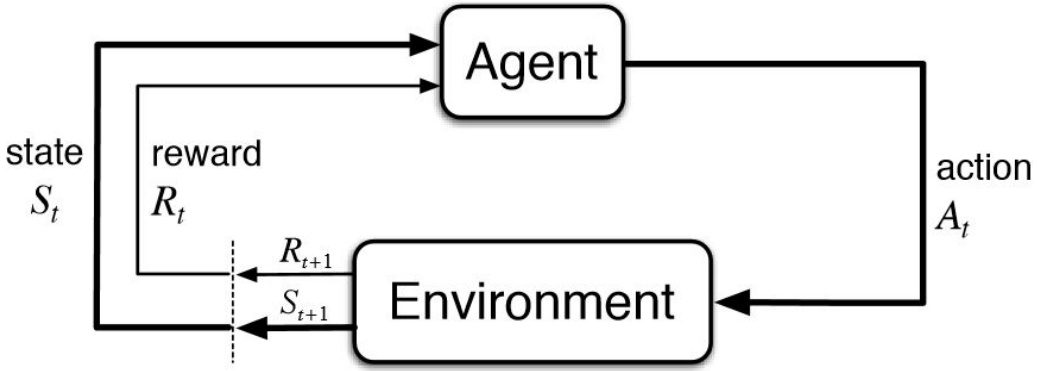
MDP se compone de:

- Set finito de estados: S_1, ...,S_n
- Set de recompensas: r_1, ..., r_n
- Set de acciones: a_1, ..., a_n
- Set de probabilidades de transición: $P_{ij}^{k}=P(s_j|s_i,a_k)$

Idea general

Se busca aprender una política para que el agente actúe.

¿Cómo hacemos que el agente elija la mejor acción?



Value Function

$$V(s) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_t^{\pi}\right\}$$

Valor esperado de recompensas totales del agente al partir en el estado s y siguiendo la política π

Tasa de descuento (0,1]

Recompensas de la política π

Value Function

Buscamos la política π que maximiza la *value function*.

$$V^*(s) = \max_{\pi} E\left\{\sum_{t=0}^{\infty} \gamma^t r_t^{\pi}\right\}$$

Ecuación de Bellman

$$V^*(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right\}, \quad \forall s \in S$$

Recompensa del estado actual

Recompensa futura esperada

Value iteration

Initialize V(s) arbitrarily
loop until policy good enough
    loop for $s \in S$
        loop for $a \in A$
            $Q(s,a) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \hat{V}(s')$
        end loop
        $\hat{V}(s) := \max_a Q(s,a)$
    end loop
end loop
return$\{\hat{V}(s)\}$

Policy iteration

Choose an arbitray policy π'

Loop

$\pi := \pi'$

Compute value function of policy π:

#solve linear equations

$V_\pi(s) := R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$

Improve the policy at each state

$\pi'(s) := \arg\max_a \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s') \right)$

until $\pi = \pi'$

Q-Learning: Generalidades

Algoritmo de aprendizaje reforzado libre de modelo; no disponemos de las probabilidades de transiciones entre estados, de modo que no podemos usar policy/value iteration

El objetivo consiste en calcular una función Q(s, a) que nos entregue q*(s, a) para cada estado s y acción a (donde a debe estar permitida en s)

Como los estados y acciones de las que disponemos son finitos y discretos, Q puede describirse como una tabla

Q-Learning: Intuición

Q-Learning funciona en base a la idea de que, para cada par estado-acción (s, a), es posible ajustar iterativamente los valores q(s, a) hasta hacerlos converger a q*(s, a)

La intuición detrás de esto es que si actualizamos la política de aprendizaje para un par estado-acción (s, a) mediante una **estrategia épsilon-greedy**, la política nueva es levemente mejor a la anterior

Una política mejor que la anterior refiere a que los valores q(s, a) y q'(s, a), asociados respectivamente a la política antigua y a la política nueva, cumplirán con la siguiente desigualdad:

$$q'(s, a) > q(s, a)$$

(Si les interesa esto, en la sección 5.3 del libro de las referencias se explica cómo se llega a esa desigualdad)

Q-Learning: Estrategia épsilon-greedy

La estrategia épsilon-greedy nos permite actualizar la política de nuestro agente sin disponer de un modelo de cómo funciona el mundo

La intuición es que en base a un parámetro de tasa de exploración entre 0 y 1 (épsilon) podemos decidir si en un instante en particular **explotamos** una estrategia conocida, o **exploramos** nuevas opciones

En lo concreto, épsilon es la probabilidad de optar por alguna de las siguientes opciones:

- Elegir una acción aleatoria
- Elegir una acción en base a los que nos indique la tabla Q, donde buscaremos la acción que nos lleve al estado vecino con mayor recompensa posible (de ahí el *greedy*)

Pregunta abierta:

¿Qué creen que es más conveniente, que la tasa de exploración se incremente o disminuya en la medida en que transcurren los episodios de entrenamiento?

Q-Learning: Actualización de la tabla Q

Una vez que elegimos la acción a, la ejecutamos, transitamos de s a s', y registramos la recompensa R

La actualización de valores de la tabla Q para el par (s, a) que acabamos de dejar se lleva a cabo en base a la siguiente expresión

$$q^{new}(s,a) = (1-\alpha)\, q(s,a) + \alpha\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)$$

- alpha: parámetro correspondiente a la tasa de aprendizaje del agente
- gamma: parámetro correspondiente al factor de descuento del agente
- s': estado cualquiera perteneciente al conjunto de vecinos de s
- a': acción cualquiera que puede ser realizada desde s'

Q-Learning: Pseudocódigo

```
# Pseudo codigo Q-Learning

# Q_table incicialmente son solo 0
# epsilon incicialmente es 1

Q-Learning(Q_table, epsilon, alpha, gamma):
    for episode in Episodes:
        for t in Times:
            s = obtener estado actual
            # Seleccion de accion
            if random_float < epsilon:
                # Exploramos
                a = accion seleccionada aleatoriamente

            else:
                # Explotamos
                a = accion asociada al estado vecino de s con mayor valor q

            # Ejecucion de accion
            s_new, r = ejecutar(a,s)

            # Actualizacion de la tabla
            Q_table[s, a] = alpha * Q_table[s, a] + (1 - alpha) * gamma * max(Q_table[s_new, a_new] para todo a_new posible desde s_new)

        # Terminado un episodio, disminuimos la tasa de exploracion
        # f(epsilon, episode) es una funcion que decrementa epsilon en la medida en que transcurren los episodios

        epsilon = f(epsilon, episode)
```

Veamos un ejemplo en código para cerrar!

Referencias y recursos útiles

- Libro: http://incompleteideas.net/book/the-book.html
- Curso online: https://deeplizard.com/learn/video/nyjbcRQ-uQ8