Capítulo 4: Dynamic Programming.

Exercise 4.1: If π is the equiprob. random policy, what is $q_\pi(11, \downarrow)$? And $q_\pi(7, \downarrow)$

$$q_\pi(s, a) = \sum_{s', r} p(s', r | s, a)[r + \gamma V_\pi(s')]$$

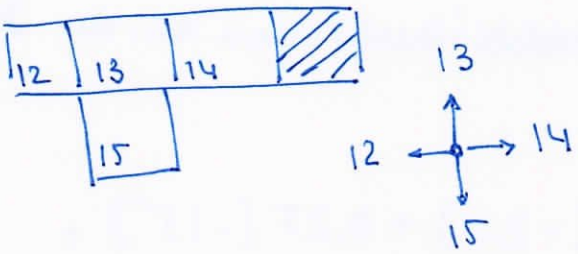
$$q_\pi(11, \downarrow) = p(\text{terminal}, -1 | 11, \downarrow)[-1 + \gamma \cdot V_\pi(\text{terminal})] = -1$$

($p = 1.0$; $\gamma = 1.0$; $V_\pi(\text{terminal}) = 0.0$)

$$q_\pi(7, \downarrow) = p(11, -1 | 7, \downarrow)[-1 + \gamma \cdot V_\pi(11)] = -15$$

($p = 1.0$; $\gamma = 1.0$; $V_\pi(11) = -14$)

Exercise 4.2. We add a state "15" below "13"



¿$V_\pi(15)$ para $\pi(a|s) = 0.25$ $\forall s, \forall a \in A(s)$?

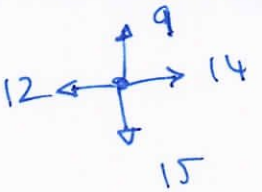
$$V_\pi(15) = \sum_a \pi(a|s) \sum_{s', r} p(r, s' | s, a)[r + \gamma V_\pi(s')] =$$

$$= 0.25[-1 + V_\pi(12)] + 0.25[-1 + V_\pi(13)] + 0.25[-1 + V_\pi(14)] +$$

($V_\pi(12) = -22$; $V_\pi(13) = -20$; $V_\pi(14) = -14$)

$$+ 0.25[-1 + V_\pi(15)] \neq . \rightarrow$$ Hay que resolver la ecuación, y queda $V_\pi(15) = -20$

b). Supongamos ahora que las dinámicas en el c.d.o $s=13$ cambian



¿$V_n(15)$ ahora?

→ En este caso, $V_n(13)$ cambia! y por tanto, $V_n(15)$ también! Empiezo con $V_n(15) = -20$, ¿después mejoro $V_n(13)$ y luego resuelvo $V_n(15)$? O lo escribo todo en términos de $V_n(15)$

$$V_n(13) = 0.25\,[-1 + \underbrace{V_n(12)}_{=-22}] + 0.25\,[-1 + \underbrace{V_n(9)}_{=-20}] + 0.25\cdot$$
$$[-1 + \underbrace{V_n(11)}_{-14}] + 0.25\,[-1 + V_n(15)] \quad (*)$$

$$V_n(15) = 0.25\,[-1 + V_n(13)] + 0.25\,[-1 + \underbrace{V_n(12)}_{-22}] +$$
$$+ 0.25\,[-1 + \underbrace{V_n(14)}_{-14}] + 0.25\,[-1 + V_n(15)] \quad (**)$$

Sustituyendo $(*)$ en $(**)$, obtendríamos una ec. de 2º grado:

$$V_n(13) = 0.25 \cdot (-23) + 0.25\,[-21] + 0.25\,[-15] +$$
$$0.25\,[-1 + V_n(15)] = -5.75 + (-5.25) + (-3.75) +$$
$$+ 0.25\,[-1 + V_n(15)] =$$
$$= -15 + 0.25\,V_n(15)$$

$$V_n(15) = 0.25\,[-16 + 0.25\,V_n(15)] + (-5.75) +$$
$$+ \underbrace{(-3.75)}_{-3.75} + 0.25\,[-1 + V_n(15)] =$$

$$= -4 + 0.0625\, V_\pi(15) \neq -11 - 0.25 + 0.25\, V_\pi(15) =$$

-13.75

$$= \quad + 0.3125\, V_\pi(15)$$

$$0.6875\, V_\pi(15) = \quad \Rightarrow V_\pi(15) = 20$$

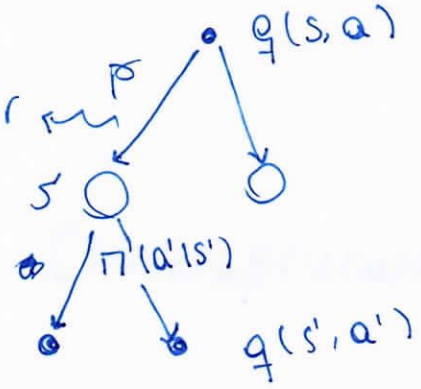
Ex. 4.3 Ecs. análogas a 4.3, 4.4 y 4.5 para $q_\pi(s,a)$? ?

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_0^\infty \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] =$$

$$= \mathbb{E}_\pi\left[R_t + \gamma \sum_0^\infty \gamma^k R_{t+k+2} \mid S_t = s, A_t = a\right] =$$

$$= \mathbb{E}_\pi[R_t \mid S_t = s, A_t = a] + \gamma\, \mathbb{E}_\pi[G_{t+1} \mid S_t = s, A_t = a] =$$

$$= \sum_{s',r} r\, p(s', r \mid s, a) + \gamma \sum_{s',r,a'} \pi(a' \mid s')\, p(s', r \mid s, a)\, q_\pi(s', a')$$



y entonces:

$$q_{k+1}(s,a) = \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \sum_{a'} \pi(a' \mid s')\, q_k(s', a')\right]$$

Ejercicio 4.4. El alg. de policy iteration en la pág. 80 tiene un bug...

~~Tuo~~ Imaginemos varias π que son todas igual de buenas.

if $V_\pi(s) \neq V_{\pi'}(s)$ then policy-estable $\leftarrow$ false

Ejercicio 4.5. ¿Cómo se definiría la policy iteration para $q_\pi(s,a)$?

~~$q(s,a) \in \mathbb{R}$ y $\pi(s)$~~

$V(s) \in \mathbb{R}$ arbitrario } $\forall s \in S$.

$\pi(s) \in A(s)$ " } Inicialización *

$q_{\pi}(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')]$ $\forall s, \forall a$

Policy Evaluation:

Loop:

$\Delta \leftarrow 0$

Loop for each $s \in S$, a:

$q \leftarrow q_k(s,a)$

$q_{k+1}(s,a) \leftarrow \sum_{s',r} p(s',r|s,a)\left[r + \gamma \sum_{a'} \pi(a'|s') q_k(s',a')\right]$

$\Delta \leftarrow \max(\Delta, |q - q(s,a)|)$

until $\Delta < \theta$

3. Policy Improvement:

policy-stable ← true

$\forall s \in \mathcal{S}$:

old-action ← $\pi(s)$

$\pi(s) \leftarrow \arg\max_a \left[\sum_{s',r} p(s', r | s, a) [r + \gamma q(s', a')] \right]$

if $q(s, \text{old-action}) \neq q(s, \pi(s))$ then policy-stable ← false

If policy-stable, then stop and return $q \approx q_*$ y $\pi \approx \pi_*$ else 2.

* otra posible inicialización:

$\pi \leftarrow$ arbitrario

$Q \leftarrow$ arbitrario $\mathcal{S} \times A(s) \mapsto \mathbb{R}$

Ej. 4.6. políticas ε-soft: la prob. de escoger cada acción en cada estado es $\frac{\varepsilon}{A(s)}$. ¿Cambios necesarios a introducir en los pasos 3.2 y 1 del algoritmo de iteración de política para cumplir con esto?

Paso 3:

$\forall s$ ~~y $\forall a$~~:

if $\pi(a|s) = \frac{\varepsilon}{|A(s)|}$ if $a \neq \arg\max$

~~$p+= \frac{\varepsilon}{|A(s)|}$~~

En este caso, la política pasaría a ser estocástica.
En el paso 3, tendríamos que darle una prob. $\pi(a|s) = \frac{\varepsilon}{|A(s)|}$ a los acciones que no son óptimas y el resto de la masa

$$1 - \left[|A(s)| - 1\right]\frac{\varepsilon}{|A(s)|}$$ a la acción óptima.

En el paso 2, tendríamos

$$V(s) \leftarrow \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$$

En el paso 1, $\pi(s)$ es ahora una distribución, no es una función:

$$\pi: A \times S \longrightarrow \mathbb{R}.$$

Ejercicio 4.8. ¿Por qué la política óptima del problema del jugador tiene esa forma? ¿~~Qué otras~~ ...

Estados: capital en cada momento. $S = \{1, 2, \dots, 99\}$

$a \in \{0, 1, \dots, \min(s, 100-s)\}$

Ej.: $s = 54$.

$a \in \{0, 1, \dots, \min(54, 100-54)\} = \{0, 1, \dots, 46\}$

¿Por qué es mín? Para no ganar más de 100:

Tengo 54. Apuesto 54 $\longrightarrow$ Gano $54+54 = 108$.

Máximo

En lugar de eso, lo que gano es lo que me falta para llegar a 100:

$100 - 54 = 46$.

~~Si lo que apuesto es 33, que es menor que 50, entonces gano 33+33 =~~

~~No tengo que~~

Si $s = 33$ y apuesto 33, entonces no hay que hacer el artificio anterior, ya que $33+33 < 100$.

En general, cuando $s > 50$ y lo apuesto todo, lo que gano es

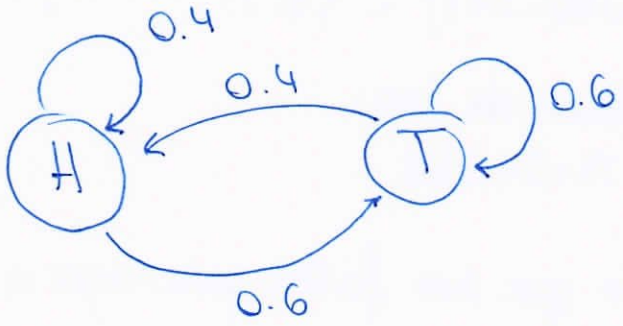
~~100~~ $s + (100-s) = 100$.

$P_h = 0.4$; $P_t = 1 - 0.4 = 0.6$.

Value iteration:

$$V_{k+1}(s) = \max_a \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma V_k(s') \right]$$

~~Sabemos que, en este caso, hallar la misma $p(s', r \mid s, a)$, esto se puede descomponer: $p(s', r \mid s, a) = p(s') \cdot p(r \mid s, a)$?~~



~~$p(s'=1, r=1 \mid s=5, a=4) = \ldots$~~

Imaginemos que tenemos q estados: $\{1, \ldots, q\}$ y

$a \in \{0, 1, \ldots, \min(s, 10-s)\}$

$p(s'=1, r=0 \mid s=1, a=0) = 1.0$

$p(s'=2, r=0 \mid s=1, a=1) = 0.4$

$p(s'=0, r=0 \mid s=1, a=1) = 0.6$

~~$p(s'=3, r=0 \mid s=1, a=2) =$~~

~~$p(s'=0)$~~

$p(s'=3, r=0 \mid s=2, a=1) = 0.4$

$p(s'=1, r=0 \mid s=2, a=1) = 0.6$

$p(s'=4, r=0 \mid s=2, a=2) = 0.4$

$p(s'=\emptyset, r=0 \mid s=2, a=2) = 0.6$

según Sutton, la lógica detrás es: dado que la moneda está inclinada en contra del agente, lo mejor es apostar lo mínimo posible. Cuando llega a un 50% del capital, entonces lo mejor es apostarlo todo. Pero si tiene 51%, puede hacerlo mejor: puede apostar un euro y así ponerse en 52€, con lo que seguirá teniendo la opción de apostar 50 y ganarlo todo si hubiera necesidad. Lo mismo para 53, 54, 55, ... Cuando llega a 75, puede apostar 25 y ganará ~~todo~~ en una jugada. Pasados los 75, de nuevo se aplica la lógica anterior: si apuesta 1, se pone en 76, con lo que ~~puede~~ va ganando mientras mantiene los 25 o 4 salvo para una jugada en un golpe.

~~Con respecto a los~~

Esto explica los cambios bruscos. Con respecto a las primas de sierra, diría que cualquier política con la misma integral bajo esas porciones de curva funciona?

14.10. ¿Cuál es el análogo de value iter. update usando $q_{k+1}(s,a)$?

$$q_{k+1}(s) = \max_a \sum_{s',r} p(s',r|s,a)\,[r + \gamma \max_{a'} q_k(s',a')]$$

Ejercicio 4.9: Programming. Implement Value Iteration for gambler's problem...

~~la cosa~~

El pseudocódigo de Value iteration es:

Parámetro (θ).

Inicializa $V(s)$, $\forall s \in \mathcal{S}^+$, arbitrariamente excepto $V(\text{terminal}) = 0$.

loop:

$\Delta \leftarrow 0$

loop for each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$

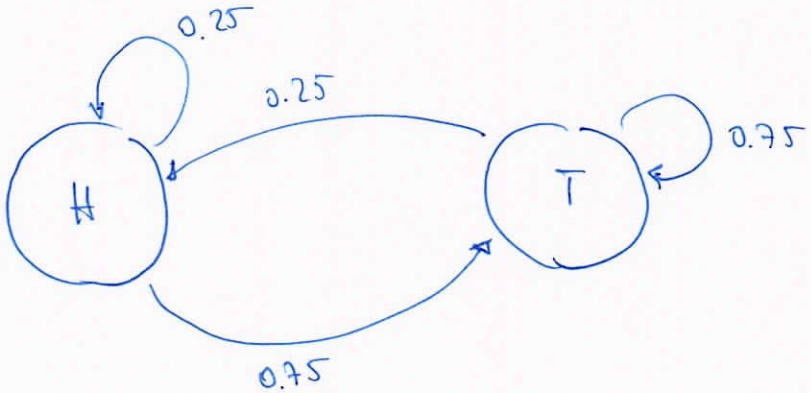
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

output $\pi \approx \pi_*$:

$\pi(s) = \arg\max_a \sum_{s',r} p(s', r | s, a) [r + \gamma V(s')]$

En nuestro caso, $\gamma = 1$. También conocemos las probs. de cara y cruz:



$s' = s + a$

Si sale cara, ~~[illegible]~~. ~~Si~~ y $r=0$. Si sale cruz, $s' = s-a$ y $r=0$.

$r=1$ sólo si $s'=100$.

~~Por lo tanto,~~

Vamos a tratar de especificar esto en términos de probabilidad $p(s', r | s, a)$:

~~Sabemos que $V(s')$ también se puede~~

1) Sabemos que

$$r(s', a, s) = \frac{\sum_r r \cdot p(s', r | s, a)}{p(s' | s, a)}$$

y por lo tanto, $\sum_r r\, p(s', r | s, a) = r(s', a, s) \cdot p(s' | s, a)$

2) También sabemos que $r(s', a, s)$ en realidad sólo es función de s': ~~$r(s', a, s) \equiv r(s') =$~~

$$\begin{cases} 0 & \text{if } s' \neq 100 \\ 1 & \text{else} \end{cases}$$

3) Por otra parte, sabemos que $p(s' | s, a) = 0.25$ si ~~$s' = s+a$~~ y $s' = s-a$.

O sea, que dado s, sabemos que $a \in [0, \ldots, \min(s, 100-s)]$, y dados s y a, tenemos:

$$p(s' = s+a \mid s, a) = 0.25$$
$$p(s' = s-a \mid s, a) = 0.75$$

Finalmente,

~~4)~~ También sabemos que ~~$\sum_{r,s'} p(s', r | s, a) V(s') = \sum_{s'} p(s' | s, a) V(s')$~~

Por lo tanto, podemos reescribir la ec. de actualización para $V(s)$ como

$$V(s) \leftarrow \max_a \left[\sum_{s'} r(s', a, s)\, p(s' | s, a) + \sum_{s'} p(s' | s, a) V(s') \right]$$

o lo que es lo mismo:

$$V(s) \leftarrow \max_a \left[\sum_{s'} p(s'|s,a)\left(r(s') + V(s')\right) \right]$$

Dados s y a, la suma anterior es de sólo dos sumandos:

~~$V(s) \leftarrow \max_a \left[0.25 \left[r(s+a) + V(s+a) \right] + 0.75 \left[r(s-a) - V(s-a) \right] \right]$~~

$$\max_a \left[0.25\left(r(s+a) + V(s+a)\right) + 0.75\left(r(s-a) + V(s-a)\right) \right]$$

Valor calculado para "a".

Dado s,

- Hay entonces que almacenar estos valores de candidato a V(s), cada uno asociado a las acciones posibles, y seleccionar el máximo como V(s).

otra forma de hacer esto es ir guardando, en cada estado, el máximo y ir sustituyéndolo. En cada estado, este valor se renueva:

~~max a~~ V_max_a = -1000

A = 0.25 * ( ) + 0.75 · ( )

~~V_max_a = máx (V_max_a, A)~~

if A > V_max_a

V_max_a ← A

~~max_a ← a~~

Por tanto, el algoritmo completo queda:

```
def value_iteration(θ):
    S = [0, ..., 190]
    r = [0, ..., 1]
    V(S) = [0, ..., 0]

    do
        Δ ← 0
        for s_0 in S:
            v ← V(s_0)
            V_max_a ← -1000
            for a in A(S):
                A ← 0.25 f(s+a) + 0.75 f(s-a)
                if A > V_max_a
                    V_max_a ← A
            V(s_0) ← V_max_a
            Δ ← max(Δ, |V - V(s_0)|)
    while Δ ≥ θ
```

$s_0 \equiv s$

$\vec{S} \equiv$ states

$\Delta =$ delta

$\theta =$ theta

$A(s) =$ get_action_space(S)

Bellmann update

```
def A(s):
    return [i for i in (0, min(s, 190-s))]
```

Podemos también encapsular la parte del bucle en A(s) como:

$V(s_0) \leftarrow$ Bellmann_update(s, a), and return V_max_a.