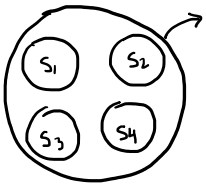
Value Iteration: Frozen Lake

Saturday, April 30, 2022 9:58 AM



→ ∀ trayectory
if we got $S=\{S_1, S_2, S_3, S_4\}$ in our world, all possible trayectories are $V(S_1)$, $V(S_2)$ $V(S_3)$ and $V(S_4)$. There can't be more than that.

In stochastic and deterministic action functions, the optimal trayectory is the same, just one. In Stochastic

the 100 tries build P_{MT} and f_R, then in each ith try, represent each value iteration for solving bellman's equations.

0.85 → 85% percent of playing, it reaches the Goal.

Value Iteration

We saw previously how to solve Bellman's Optimality Equations using the numerical method of Value Iteration. These equations were defined either for $V(s)$ or $Q(s,a)$:

$$V(s) \leftarrow \max_a \left[\sum_{s_f \in S} P_{MT}(s_f \mid s,a) \left[f_R(s,a,s_f) + \gamma V(s_f) \right] \right]$$

$$Q(s,a) \leftarrow \sum_{s_f \in S} P_{MT}(s_f \mid s,a) \left[f_R(s,a,s_f) + \gamma \max_{a_f} \left[Q(s_f, a_f) \right] \right]$$

Known

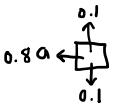
By 'known' we mean that either those were given explicitly or we get them by making the agent experience the world randomly and calculating them.

Therefore, for solving the equations using this method, we need to know the Transition Model and the Reward Function.

↳ To define them, we sent the agent to experience the world randomly 100 times in order to LEARN THE TRANSITION MODEL and REWARD FUNCTION, since we need to know them for solving the Eqs using Value Iteration.

Frozen Lake World: Stochastic result of actions (transition model is P_{MT})

↳ for every action a:



→ 0.8 prob of going correctly
→ 0.1 prob of going left from the expected
→ 0.1 prob of going left from the expected

three
F → free
H → hole (episode ends)
G → Goal (episode ends)

NO REWARD UNTIL GOAL → H, F

↳ is the only way to learn from the world

The whole point is to iterate to solve Bellman's Equations

↳ the info of the reward is implicit in the Bellman Equations we are trying to solve with Value Iteration (learned randomly)

→ If the transition model function is deterministic, the actions in the Optimal Politic are always the same. That is, once we solve Bellman's System of Eqs and thus we get the optimal politic, if we test

→ If the transition model function is deterministic, the actions in the Optimal Politic are always the same. That is, once we solve Bellman's System of Eqs and thus we get the optimal politic, if we test the agent using that politic, every test we will see the same trayectory. But, since frozen lake is stochastic, every test using the optimal Politic will be always different: the Politic is ONE, but transition may diverge.

DETERMINISTIC (TM) / NON DETERMINISTIC (TM) } ONE OPTIMAL POLITIC ⇨ the difference is the trayectories

the difference is the trayectories → World (TM)

DET: one trayectory = optimal politic

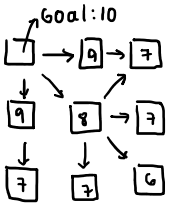
Many trayectories { In NONDET, the action taken results in accordance to P_{MT} even though is following the Optimal Politic

→ In frozen lake program: what is the agent truly experimenting?
↳ the agent moves randomly to know / learn the Transition Model (1) and Reward Function (2)

here, the agent is not really experiencing the world to do this, it's just iterating { (1) and (2) are needed for solving the Bellman's Eq System using valve iteration

valve iteration → implicit info about reward in Goal

Temporary Difference Method: in value iteration, the agent doesn't arrive at the goal and thus solves the politic: the agent knows $f_R(s,a,s_f)$ and with that solves Bellman's Equations which output the politic. Meanwhile, in temporary Difference, the agent does need to arrive to the goal to know there is reward and it has to propagate it backwards to the states near the goal as if it was a ripple.



back propagation of the reward with Temporary Difference

```
In [1]: runfile('C:/Users/José Ab
(1).py', wdir='C:/Users/José Abdó
i= 0  r_prom= 0.0
i= 1  r_prom= 0.0
i= 2  r_prom= 0.0
i= 3  r_prom= 0.0
i= 4  r_prom= 0.0 -> 0.1
i= 5  r_prom= 0.0
i= 6  r_prom= 0.1 -> 0.25
i= 7  r_prom= 0.25 -> 0.3
i= 8  r_prom= 0.15
i= 9  r_prom= 0.3
i= 10  r_prom= 0.3 -> 0.75
i= 11  r_prom= 0.5
i= 12  r_prom= 0.55
i= 13  r_prom= 0.3
i= 14  r_prom= 0.4
```

each i is one value iteration ↓ In this case, the Bellman Eqs are solved in 19 iterations ↓ is like one excel iteration ↓ which updates all V(S)'s ↓

with the update of V(S)'s we are getting avg reward of 0.0 from the 20 episodes

from the 20 episodes, we have avg reward = 0.1
↳ the value iteration is getting close to the solution of the Equations.

→ the 20 episodes: check whether or not V(S) is the final solution to Eqs by computing average reward.
→ the 100 random actions: experience world to calculate and adjust P_{MT} and f_R: needed for Eqs.

which updates all V(S)'s

↓

after this upate, we test the agent in 20 episodes, from which we compute the reward average

→ to know if the V(S) upate is the final one (solved)

```
i= 12  r_prom= 0.55
i= 13  r_prom= 0.3
i= 14  r_prom= 0.4
i= 15  r_prom= 0.7
i= 16  r_prom= 0.75
i= 17  r_prom= 0.65
i= 18  r_prom= 0.65
i= 19  r_prom= 0.75 -> 0.85
¡Resuelto!

Probar agente [1=si|0=no]:
```

world to calculate and adjust P_{MT} and f_R : needed for Eqs.

we specified that when r_{avg} reaches 0.85 or higher, V(S) is the 'correct' solution (or very close)

= 85% of times the agent reaches Goal (G)

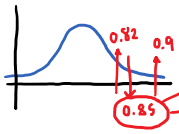
↓

But this average is calculated with 20 episodes, which is our precision for the average: if we compute this 20 episodes many times without updating V(S), the average changes

Why? ← not every time it is

But if we leave 0.85 and increase to 1000 tests, and 0.85 is reached → μ may be close to 0.85

this condition may be wrong



σ big (tests → 20)

This may be μ the real population average reward

σ small (tests → ∞)

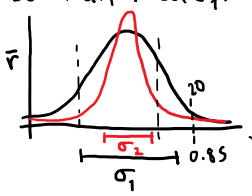
20 is small ← amount of tests to get the population average reward

if we increase 20 to 1,000 the average reward may not change that much = we are close to population's average (μ)

when tests → ∞ : m = μ

Central Limit Theorem

testing 20 many times (σ_1) testing 100 many times (σ_2)



plot of the distribution of average reward

→ When σ/variance is big, we are not giving much importance to the real average reward of the population

↳ but by keeping 20, we can achieve a 'good' solution of the Eqs.

→ We put a stop condition on 0.85 of average which means we stop solving when 85% of times the goal is reached.

↳ it would be 100%. if the world was deterministic. 100% with stochastic may never be reached.

↓

σ = 0

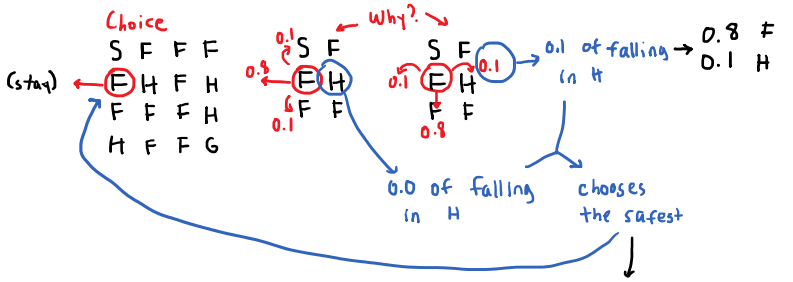
μ = the same every test

Conclusion:

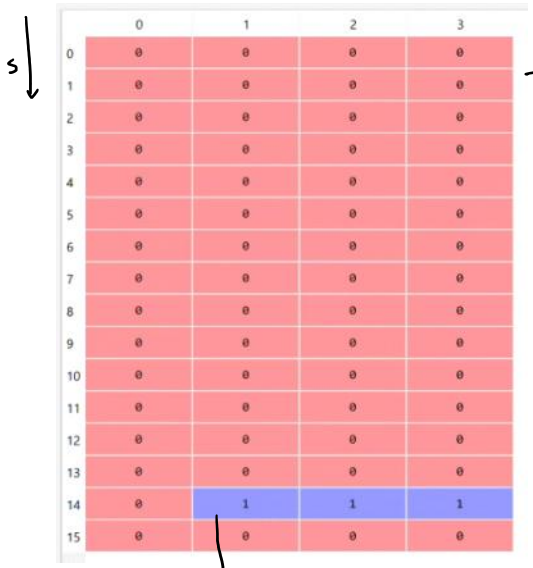
→ Increasing episodes to 1,000 instead of 20 and leaving 0.85 as the stop condition, it means that the 0.85 reached with V(S) solution is closer to the real 85% I want.

↳ 0.85 in 20 is an 85% with less precision

→ Look at the case:



0.8 of F (staying)
0.0 of H



s	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	1	1	1
15	0	0	0	0

r array (3D)

→ r[:, :, 15]
- s
- a
- $s_f = s_{15}$

r[0, :, :]
- $s_0 = s(0)$
- a
- s_f (all)

r[:, 1, :]
- s
- a_1
- s_f

$s = 14$ with a_1, a_2 or a_3
and we arrive at $s_f = 15$, reward of 1

Now with a_f → a

s	0	1	2	3
0	0.077334	0.0743577	0.07552	0.0677858
1	0.0430185	0.0487054	0.0492081	0.0701037
2	0.0823001	0.0738802	0.0766102	0.0658866
3	0.0404025	0.0434737	0.0399304	0.0633655
4	0.100769 (MAX)	0.0746953	0.0653294	0.0586504
5	0	0	0	0
6	0.117726	0.0792994	0.108976	0.022221
7	0	0	0	0
8	0.0782228	0.122758	0.118385	0.155632
9	0.17592	0.255922	0.205472	0.139493
10	0.308186	0.265674	0.261929	0.168142
11	0	0	0	0
12	0	0	0	0
13	0.282138	0.176336	0.390274	0.280115
14	0.404579	0.649728	0.565751	0.518481
15	0	0	0	0

Q(s,a) = the accumulated reward of all trayectories that start in s and execute a.

in s_4, the biggest value of all a's tell us the optimal action which is a_0 :
In s_4, the executed action is a_0

may be never visited s_{11}, and it may be a hole

Thus:
→ V(s): less memory (vector) but slower since it needs to compute a.
→ Q(s,a): more memory (matrix) but faster since it already has a.