Planning by
Dynamic Programming

발표자 : 이경임

Review `State-value function`

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- 에이전트가 현재상태 s에서 정책 π를 따랐을때 획득할 총 보상의 기대값
- return `G_t`
  = R_t+1 + γR_t+2 + ...
  = Total **discounted** reward from timestep t

Review `Action-value function`

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid S_t = s, A_t = a]$$

- 어떤 상태 s에서 정책 π에 따라 행동 a를 수행했을때 획득할 총 보상의 기대값
- state-value function에서 행동 a에 대한 조건 추가

" `Bellman equation` "

state-value function과
action-value function의 *관계*를 나타내는 방정식

Review `Bellman expectation equation`

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right]$$

- 현재상태 *S_t에서의 가치는* ***다음상태 S_t+1의 가치에 discount factor γ를 곱해 더한 기대값***이다.
- 현재상태 *S_t와 행동 A_t의 Q-value는* ***다음상태 S_t+1와 행동 A_t+1의 Q-value에 discount factor를 곱해 더한 기대값***이다.

Solution for state-value function

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$$

Solution for action-value function

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Dynamic programming for MDP

- DP : 큰 문제를 작은 문제로 쪼갠다. 작은 문제에 대한 솔루션을 찾아 이를 재귀적으로 반복하여 큰 문제를 풀어낸다.
- MDP는 DP로 풀어낼 수 있는 문제의 조건을 만족한다.
- Model-based 강화학습 문제에서(environment를 모두 알고 있을 때) 최적을 policy를 찾기 위한 방법으로 Dynamic programming을 사용한다.

> Dynamic Programming assumes `full knowledge of the MDP`.
> It is used for `Planning` in an MDP

Policy evaluation

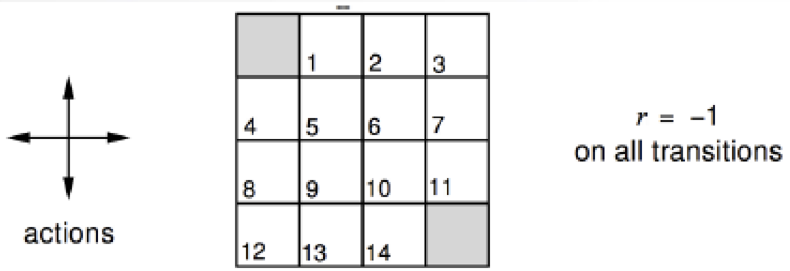
- prediction 문제
- MDP와 어떤 policy가 주어졌을때 해당 정책을 따랐을때 최종적으로 얻게될 v_π()를 계산한다.

> “ Define the value function at the next iteration
> by plugging in the previous iterations' values of the leaves ”

> “ Backup those values to compute
> one single new value for the next iteration of the root ”

- Iterative하게 Bellman expectation equation을 따라 value function을 계산하면 **현재의 policy(uniform random policy)에 대한 true value function으로 수렴**한다
- 이 과정을 policy evaluation이라고 한다.

[Example] Small Gridworld problem



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states 1, ..., 14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

- 2개의 terminal state : 회색지점
- 4가지의 action : 상, 하, 좌, 우
- 모든 칸의 변경 => -1의 reward
- 각 방향으로 움직일 확률은 동일하게 0.25
- uniform random policy에서 시작한다

Review the Lesson 2..

Bellman Expectation Equation?

“

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

”

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

➞ Bellman expectation equation을 통해 계속 모든 state에 대한 value function을 update한다.

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

v_1([2,1]) : (0.25 * (-1 + γ * (0))) * 4

v_2([1,1])

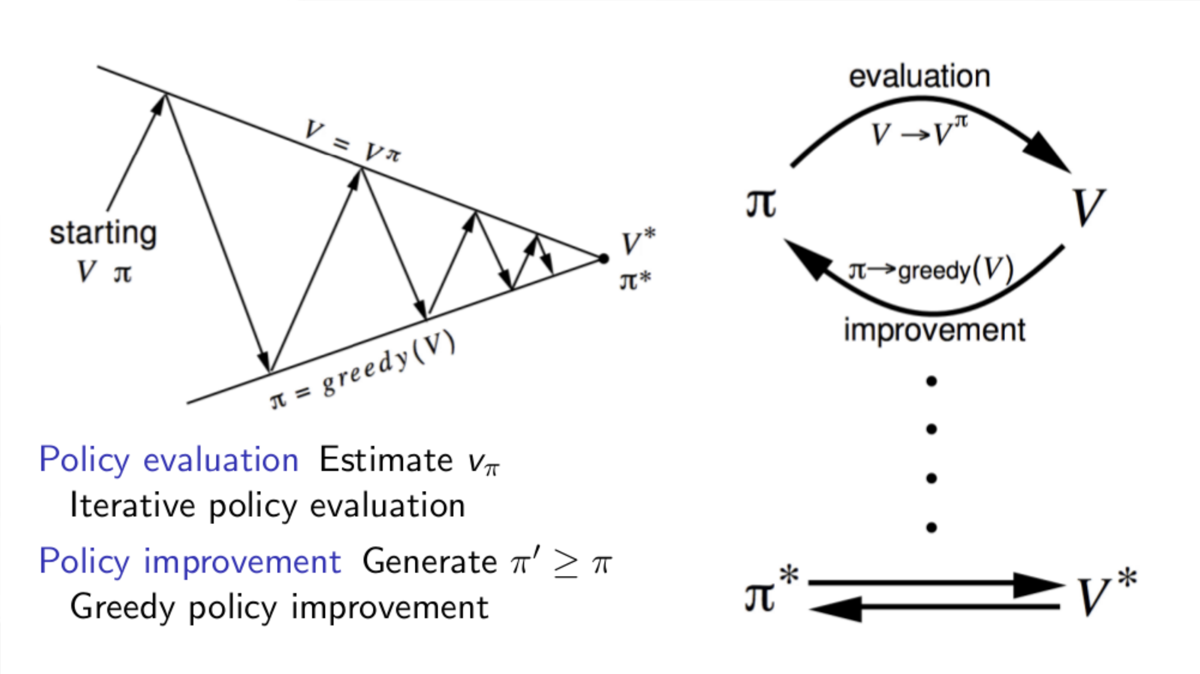
← = 0.25 * (-1 + γ * (0))

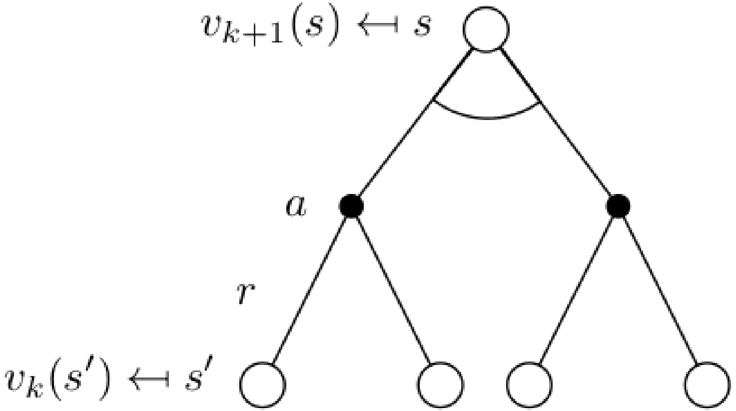
↑(↷), →, ↓ = 0.25 * (-1 + γ * (-1))

= 1.75

Policy iteration

- 다음의 두가지를 반복한다
- Evaluate the policy π
- Improve the policy by `acting greedily with respect to v_π`
- 이를 반복함으로서 optimal policy π를 찾을 수 있다





$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

Value iteration

- **Bellman optimality equation**을 사용한다.
- 특정한 policy 없이 value만으로 value function을 계산해낸다
- value function을 update할 때 *max를 취함으로서* 값을 greedy하게 improve한다.

Value iteration

- Problem : find optimal policy π
- Solution : iterative application of Bellman optimality backup
- Using synchronous backups
  - at each iteration k+1
  - for all states s in S
  - Update v_k+1(s) from v_k(s')
- Converge to v_*
- No explicit policy

[Example] Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

- terminal state에 도달하면 종료. 움직일때마다 -1의 reward.
- `v_2 = max(-1 + v(주변))` = -1

References

- https://dnddnjs.gitbooks.io/rl/
- https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/
- https://towardsdatascience.com/understanding-the-markov-decision-process-mdp-8f838510f150