12. RL - Dynamic Programming

Game Engineering & XR Technologies

Prof. HyeongYeop Kang
siamiz@khu.ac.kr
IIIXR LAB

Introduction

Dynamic Programming (DP)

- What is DP?
  - DP = a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP).
  - DP provides an essential foundation for the understanding of the methods used in RL.
    - ✓ Because all of those methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and without assuming a perfect model of the environment.

- Key idea of DP?
  - "The use of value functions to organize and structure the search for good policies."
  - Once we have found the optimal value functions (v_* or q_*) → we can easily obtain optimal policies (π_*).
  - DP algorithms are obtained by turning Bellman equations into update rules for improving approximations of the desired value functions.

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}\,[R_{t+1} + \gamma\, v_*(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \textstyle\sum_{s',r} p(s', r \mid s, a)\,[r + \gamma v_\pi(s')] \end{aligned}$$

$$\begin{aligned} q_*(s, a) &= \mathbb{E}_\pi[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s, A_t = a] \\ &= \textstyle\sum_{s',r} p(s', r \mid s, a)\,[r + \gamma \max_{a'} q_*(s', a')] \end{aligned}$$

Policy Evaluation (Prediction)

Policy Evaluation = prediction problem

- What is policy evaluation?
  - The way of computing the state-value function v_π for an arbitrary policy π.
  - Recall that *Bellman equation* for $v_\pi(s)$ is as follows:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s']\right] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_\pi(s')\right], \qquad \text{(eq.1)} \end{aligned}$$

    where $\pi(a|s)$ is the probability of taking action a in state s under policy π, and the expectations are subscripted by π to indicate that they are conditional on π being followed.
  - If the environment's dynamics are completely known, the value function can be represented as a system of $|S|$ simultaneous linear equations in $|S|$ unknowns (the $v_\pi(s)$, $s \in S$).
  - Solving these linear equations directly for the exact value function can be computationally expensive, especially for large state spaces.
  - Instead, *iterative policy evaluation* is used to estimate the value function for a given policy.

Policy Evaluation (Prediction)

Policy Evaluation = prediction problem

- Iterative policy evaluation
  - Consider a sequence of approximate value functions v_0, v_1, v_2, . . ., each mapping S^+ (set of all states, including the terminal state) to R (the real numbers).
  - The initial approximation, v_0, is chosen arbitrarily (except that the terminal state, if any, must be given value 0), and each successive approximation is obtained by using the Bellman equation for v_π as an update rule:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s')\right], \qquad \text{(eq.2)} \end{aligned}$$

  - The algorithm start with an initial approximation and refines the estimates in each iteration based on the current knowledge of the environment dynamics and the given policy.
  - This process continues until the value function estimates converge, which means that the changes between successive iterations become smaller than a predefined threshold.

Policy Evaluation (Prediction)

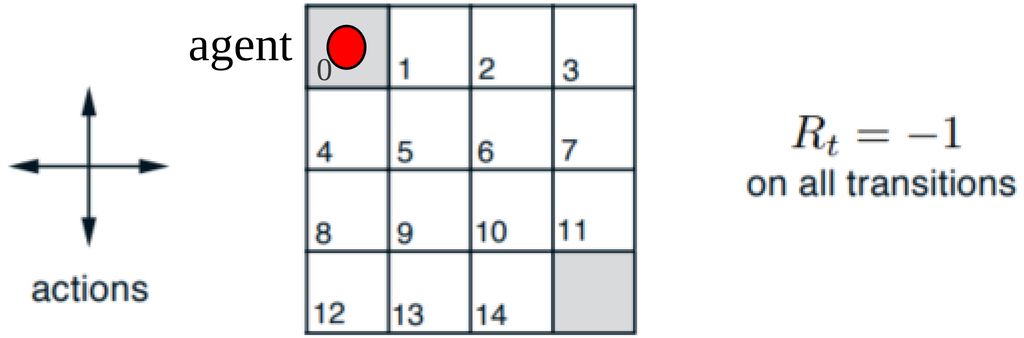
Policy Evaluation = prediction problem

- Expected update
  - To produce each successive approximation, iterative policy evaluation applies the *same operation* to each state *s*:
    → The operation replaces the old value of *s* with a new value obtained from the old values of the successor states of *s*, and the expected immediate rewards, along all the one-step transitions possible under the policy being evaluated.
    → We call this operation *expected update* because it is based on an expectation over all possible next states rather than on a sample next state (recall that $v_{k+1}(s) \doteq \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]$)
  - To implement iterative policy evaluation, we can exploit two arrays: 1) one for the old values - $v_k(s)$, and 2) one for the new values – $v_{k+1}(s)$.
    → With two arrays, the new values can be computed one by one from the old values without the old values being changed.
  - We can also exploit only one array and update the values "in place," that is, with each new value immediately overwriting the old one.
    → This in-place algorithm also converges to v_π; in fact, it usually converges faster than the two-array version.

Policy Evaluation (Prediction)

Policy Evaluation = prediction problem

- Iterative policy evaluation Example
    - Imagine a simple gridworld environment with a 4x4 grid, where the agent can move up, down, left, or right. The agent starts in the top-left corner and aims to reach the bottom-right corner. Moving between cells incurs a reward of -1, and reaching the goal provides a reward of 0. The discount factor (γ) is 0.9.



    - Then, nonterminal states are $S = \{0, 1, ..., 14\}$ and possible actions are $A = \{\text{up, down, right, left}\}$.
    - The actions that would take the agent off the grid leave the state unchanged.
        - $p(5, -1 \mid 4, \text{right}) = 1$, $p(11, -1 \mid 11, \text{right}) = 1$, and $p(6, r \mid 2, \text{right}) = 0$.
    - To calculate the value function, initialize value function v_0 arbitrarily for all states: $V_0(s) = 0$ for all states $s \in S$.
    - For each state $s \in S$, value function is updated based on (eq.2).
        - $v_1(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \, [r + 0.9 * v_0(s')]$
    - Repeat this for all states until the value function estimates converge.

Policy Improvement

Find Better Policies

- The reason of computing the value function for a policy is to help find better polices.
- Suppose we have the value function v_π for an arbitrary deterministic policy π.
  - For some s, we'd like to know whether we should change the policy to deterministically choose an action $a \neq \pi(s)$.
  - One way to answer this question is to consider selecting a in s and thereafter following the existing policy, π:

$$\begin{aligned} q_\pi(s, a) &\doteq \mathbb{E}[R_{t+1} + \gamma\, v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \textstyle\sum_{s',r} p(s', r \mid s, a)\, [r + \gamma v_\pi(s')] \end{aligned}$$

  - If $q_\pi(s, a) \geq v_\pi(s)$, this means that it is better to select a every time s is encountered than just follow π all the time.

Policy Improvement Theorem

- Let π and π' be any pair of deterministic policies such that $q_\pi(s, \pi'(s)) \geq v_\pi(s)$.
  - Then the policy π' is said to be better than π, which means that it must obtain greater or equal expected return from all states: $v_{\pi'}(s) \geq v_\pi(s)$.
- Let's discuss about an extension: changes at all states and to all possible actions, selecting at each state the action that appears best according to $q_\pi(s, a)$.
  - Then, new greedy policy π' is given by $\pi' \doteq \underset{a}{\operatorname{argmax}}\, q_\pi(s, a) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s', r \mid s, a)\, [r + \gamma v_\pi(s')]$
  - *argmax* denotes the value of a at which the expression that follows is maximized.
  - The term "greedy" comes from the fact that this policy always selects the action that appears to be the best choice in the current state, short-term maximum, based on the available information.
  - If $v_{\pi'} = v_\pi$, $v_{\pi'}$ must be v_*, and both π and π' must be *optimal policies*.

Policy and Value Iteration

Policy Iteration

- We can present a sequence of monotonically improving policies and value functions as follows:

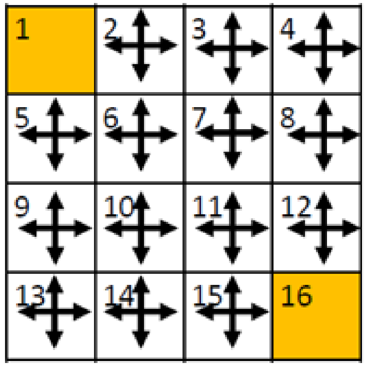
$$\pi_0 \xrightarrow{E} v_{\pi 0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi 1} \xrightarrow{I} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

- The policy evaluation is denoted as $\xrightarrow{E}$: $v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \, [r + \gamma v_k(s')]$
- The policy improvement is denotes as $\xrightarrow{I}$: $\pi' = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s', r \mid s, a) \, [r + \gamma v_\pi(s')]$

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

π_0

terminal state



terminal state

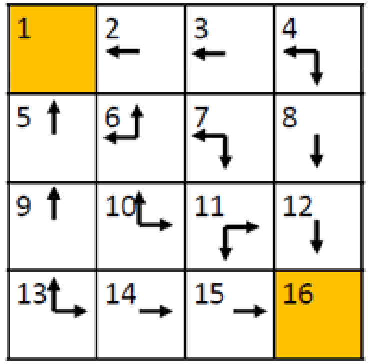
$\xrightarrow{E}$

v_{π_0}

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\xrightarrow{I}$

π_1



Policy and Value Iteration

Value Iteration

- Value iteration combines the policy evaluation and policy improvement steps into a single step:

$$v_{k+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s')\right]$$

- The algorithm iteratively updates the value function estimates until they converge.
- In general, *Policy Iteration* requires fewer iterations to converge, but each iteration can be computationally expensive due to the separate policy evaluation step.
- On the other hand, *Value Iteration* requires more iterations but can have lower computational costs per iteration.

Asynchronous Dynamic Programming

- DP methods that we have discussed so far is that they involve operations over the entire state set of the MDP, that is, they require sweeps of the state set.
- If the state set is very large, then even a single sweep can be prohibitively expensive.
- To overcome this, asynchronous DP algorithms can be used: update the values of states in any order whatsoever, using whatever values of other states happen to be available.
- To converge correctly, an asynchronous algorithm must continue to update the values of all the states: it can't ignore any state after some point in the computation.

Policy and Value Iteration

Generalized Policy Iteration

- We use the term generalized policy iteration (GPI) to refer to the general idea of letting policy evaluation and policy improvement processes interact, independent of the granularity and other details of the two processes.
- Almost all reinforcement learning methods are well described as GPI:
  - If both the evaluation process and the improvement process stabilize, that is, no longer produce changes, then the value function and policy must be optimal.
- The evaluation and improvement processes in GPI can be viewed as both competing and cooperating.
  - Making the policy greedy with respect to the value function typically makes the value function incorrect for the changed policy.
  - Making the value function consistent with the policy typically causes that policy no longer to be greedy.
- In the long run, however, these two processes interact to find a single joint solution: the optimal value function and an optimal policy.

Granularity?
Granularity refers to the *level of detail* or *resolution* at which the policy evaluation and policy improvement processes are performed. Granularity affects the computational complexity of these processes and their convergence rates.

A **fine-grained policy evaluation process** computes the value function estimates more accurately, requiring many iterations and taking longer to converge. In contrast, a **coarse-grained policy evaluation process** estimates the value function less accurately but typically requires fewer iterations, leading to faster convergence.

Reference

[mdpw] https://en.wikipedia.org/wiki/Markov_decision_process
[sutton] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Contributors

List of Contributions
[RL Project Example]