Reinforcement Learning With Hado van Hasselt

Yaro Kazakov

September 15, 2024

Abstract

These are study notes to prepare myself for PhD applications.

1 Lecture 4 - Dynamic Programming

The term dynamic programming (DP) refers to 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).
We usually assume that the environment is a finite MDP. That is, we assume that its state, action, and reward sets, S, A, and R, are finite, and that its dynamics are given by a set of probabilities p(s', r—s, a)
The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies.

1.1 Policy Evaluation

First we consider how to compute the state-value function v_π for an arbitrary policy π. This is called policy evaluation in the DP literature. We also refer to it as the prediction problem. Recall from Chapter 3 that, for all s ∈ S.

$v_*(s) = max_{a \in A(s)} q_{\pi_*}(s,a) = max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')]$

$q_*(s,a) = \sum_{r \in R, s' \in S} p(s',r|s,a)[r + \gamma v_*(s')] = \sum_{r \in R, s' \in S} p(s',r|s,a)[r + \gamma max_{a'} q_*(s',a')]$

$v_\pi(s) = \sum_a \pi(a|s) \sum_{(s',r)} p(s',r|s,a)[r + \gamma v_\pi(s')]$ for all s ∈ S.

This algorithm is called iterative policy evaluation

To write a sequential computer program to implement iterative policy evaluation as given by (4.5) you would have to use two arrays, one for the old values, vk(s), and one for the new values, vk+1(s). With two arrays, the new values can be computed one by one from the old values without the old values being changed.

This algorithm is called iterative policy evaluation.

1.1.1 Exercise 4.1

In Example 4.1, if pi is the equiprobable random policy, what is q(11, down)? What is q(7, down)?

Using the fact that the state under cell 11 is terminal and:

$q_\pi(s,a) = \sum_{r \in R, s' \in S} p(s',r|s,a)[r + \gamma v_\pi(s) = \sum_{r \in R, s' \in S} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s')(q(a',s'))$ We get the answers -1 and (-1 -14) = -15

1.1.2 Exercise 4.2

In Example 4.1, suppose a new state 15 is added to the gridworld just below state 13, and its actions, left, up, right, and down, take the agent to states 12, 13, 14, and 15, respectively. Assume that the transitions from the original states are unchanged. What, then, is v(15) for the equiprobable random policy? -20, solve the equation for v(s) for pi.

1.1.3 Exercise 4.3

4.4 - $q_\pi(a,s) = \sum_{r \in R, s' \in S} p(s',r|s,a)[r + \gamma v(s')] = \sum_{r \in R, s' \in S} p(s',r|s,a)[r + \gamma \sum_a \pi(a'|s') q_\pi(a',s')]$

4.3 - like 4.3 but with conditioning on At=a

4.5 - Like 4.5 but without tje first sum (action averaging) and instead of v_k use the equation above (4.4)

1.2 Policy Improvement

Policy Improvment Theorem:

$q_\pi(s,\pi'(s)) => v_\pi(s)$ (must be greated or equal for ALL States) then $v_{\pi'}(s) => v_\pi(s)$

1.2.1 Exercise 4.4

The policy iteration algorithm on page 80 has a subtle bug in that it may never terminate if the policy continually switches between two or more policies that are equally good. This is ok for pedagogy, but not for actual use. Modify the pseudocode so that convergence is guaranteed.

There could be a case when there are two optimal actions and we choose randomly between them. Hence, policy changes on every iteration. This will get the code stuck at the policy improvement stage forever. To mitigate that we could store lists of optimal actions, where multiple actions leading to optimal value functions are stored.

1.2.2 Exercise 4.5

Step 1:
Initialize $Q(s,a)$ for each state-action pair.

Step 2:
Loop:
delta $< -$ 0
Loop for each s,a $\in$ S, A:
q(s,a) $< -$ Q(s,a)
Q(s,a) $< - \sum_{r \in R, s' \in S} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s')(q(a',s'))$
$\delta < -max(\delta, |q(s,a) - Q(s,a)|)$
until $\delta < \theta$

Step 3: Do as in Policy Iteration for $V_(s)$