Dynamic Programming: Reinforcement Learning

Sanyam Singhal

April 5, 2022

1 Summary

1. Dynamic programming refers to a collection of algorithms that can compute optimal policies given a perfect model of the environment as an MDP. We assume the MDP to be finite. DP algorithms turn Bellman equations into iterative procedures to obtain the optimal solution atleast in the theoretical limit of infinite iterations.

2. Policy evaluation or the prediction problem refers to estimation of v_π for al states for a given policy π. We can solve the Bellman equations iteratively:

$$v_{k+1} = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{1}$$

   The solution v_π is a fixed point of this update rule because it satisfies the Bellman equation. The procedure converges to v_π in the limit $k \to \infty$.

   Here, we may implement it by storing v_k and v_{k+1} in two different arrays or we can perform the update in-place by using a single array to store v. This would imply that for some states we use the latest estimate, and for some we use one-step old estimate. Moreover, since we are summing over all possible states and rewards in the RHS, it is called an expected update method. In general, the class of iteratively solving for v_π is called iterative policy evaluation and in practice we perform iterations till a certain distance measure between v_k and v_{k+1} is less than a certain threshold.

3. Say, our policy π is deterministic. To improve the current policy, we need to evaluate $q_\pi(s, a)$ and see if it is greater than $v_\pi(s)$ for an action $a \neq \pi(s)$. Formally, the policy improvement theorem states that for a pair of deterministic policies π and π', if for all $s \in S$:

$$q_\pi(s, \pi'(s)) \geq v_\pi(s) \tag{2}$$

   Then the policy π' is as good as or better than π:

$$v_{\pi'}(s) \geq v_\pi(s) \ \forall \ s \in S \tag{3}$$

   So, to obtain a better policy π' from the current policy π, we can look for the action corresponding to which the $q_\pi(s, a)$ is the highest:

$$\begin{align} \pi'(s) &= arg \max_a q_\pi(s, a) \ \forall \ s \in S \tag{4} \\ &= arg \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \ \forall \ s \in S \tag{5} \end{align}$$

   This process of generating a better policy from the current policy is called policy improvement. The policy improvement theorem also holds for stochastic policies as well. If there are multiple actions that maximize the action-value then we can assign any probability distribution to their selection as long as long as we assign 0 probability to sub-optimal actions.

If for a policy π' and π, $v_{\pi'} = v_\pi\ \forall s \in S$ then π and π' are the optimal policies because the value update equation becomes the Bellman optimality equation.

Finding the best policy is also called the control problem.

4. We can now combine the policy evaluation and improvement into a combined iterative scheme to find the optimal policy i.e.:

$$\pi_0 \xrightarrow[E]{} v_0 \xrightarrow[I]{} \pi_1 \xrightarrow[E]{} v_1 \xrightarrow[I]{} \dots \pi_* \xrightarrow[E]{} v_* \tag{6}$$

Here, E refers to policy evaluation and I refers to policy improvement. This procedure is called the policy iteration algorithm.

5. $\boxed{\textbf{Policy Iteration Algorithm for estimating } \pi_*\textbf{:}}$

   (a) **Initialization**: Arbitrarily initialize v and π for all $s \in S$.

   (b) **Policy Evaluation**: Perform policy evaluation using iterative policy evaluation method and go to policy improvement.

   (c) **Policy Improvement**:

```
i. policy-stable ← true
ii. For each $s \in S$
        old-action ← $\pi(s)$
        $\pi(s) \leftarrow arg\max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v(s')]$
        If old-action $\neq \pi(s)$, then policy-stable ← false
iii. If policy-stable, then stop and return $v \approx v_*$ and $\pi \approx \pi_*$; else go to 5b.
```

---

6. In policy iteration, we have to perform complete policy evaluation in each iteration, which too is exact only in the limit of infinite iterations. So, we can modify the policy iteration by doing just one sweep of update (one change per state) in state values in policy evaluatation per iteration of policy iteration. This modified algorithm is called **value iteration**. Another way to look at is that we use Bellman optimality equation to evaluate v_* and then after coming satisfactorily close to it, we exit the loop and evaluate π_* from v_*. Note that since we are using Bellman equation for value estimation here, we need to assign 0 value to terminal states to get a unique solution.

7. $\boxed{\textbf{Value Iteration Algorithm for estimating } \pi_*\textbf{:}}$

   (a) **Initialization:** Initialize $v(s)$ for all $s \in S^+$ arbitrarily except that $v(terminal) = 0$.

   Also, a small accuracy threshold θ for terminating the value estimation loop.

   (b) **Loop**:

```
i. $\Delta \leftarrow 0$
ii. Loop for each $s \in S$:
        $v \leftarrow v(s)$
        $v(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v(s')]$
        $\Delta \leftarrow \max(\Delta, |v - v(s)|)$
    until $\Delta < \theta$
```

   (c) Output a deterministic policy $\pi \approx \pi_*$ such that:

$$\pi(s) = arg\max_a \sum_{s',r} p(s',r|s,a)[r+\gamma v(s')] \tag{7}$$

---

8. In general, we can have an intermediate between policy iteration and value iteration by having some sweeps of policy evaluation between iterations of policy improvement. Any algorithm that oscillates between policy evaluation and improvement and eventually converges to π_* (and v_*) is called **generalized policy iteration**.

9. A major drawback of both the major algorithms: PI or VI or of their intermediate is that we update some function of all states atleast once in each iteration. This is computationally prohibitive for tasks with exponentially large state spaces. Moreover, all states need not be equally important. So, we can go about updating states as we encounter them or update a subset of important states. Theoretically, in whatever way we update states, we need to update each state infinitely often to get exact answers, but in practice we can get satisfactory results by focusing only on a subset of states, decided in a problem specific manner. Algorithms of this kind are called asynchronous dynamic programming algorithms. Application of RL in problems like learning to play chess requires such strategies as the number of possible states (pieces configuration) is about 10^{45}.