B9120 Dynamic Programming Lecture 06 - March 2, 2020

Value Iteration and Online Variants

Lecturer: Daniel Russo *Scribe: Marco, Gagan*

So far the course has really emphasized two things: formulation and problem specific solution ideas. The rest of the course will focus on general algorithms that could be applied to a wide variety of problems. The three main algorithmic ideas comprise of Value Iteration, Policy Iteration and an LP formulation of the problem. We will simultaneously consider approaches in the Approximate Dynamic Programming and Reinforcement Learning literature that could be used to

1 Markov Decision Process

We consider algorithms that can solve for a (nearly) optimal policy in a Markov Decision Process (MDP), $\mathcal{M}$, provied to us as a tuple

$$\mathcal{M} = (X, U, P, g, d_0, \alpha),$$

where

- $X = \{1, \ldots, n\}$ is the state space which is assumed to be finite.
- $U(i)$ is the finite set of feasible actions at state i.
- $P_{ij}(u)$ is the transition probability from state i to j when the action u is taken.
- $g(i, u)$ is the expected instantaneous cost which we will assume to be uniformly bounded by M, i.e.,

$$|g(i, u)| \leq M \quad \forall i, u$$

- d_0 is the initial state distribution, viewed as a row vector.
- and $\alpha \in (0, 1)$ is the discount factor.

Remark In many problems, instantaneous costs are random. For example, there may be a known cost $g(i, u, j)$ associated with a transition from state i to j. Working directly with the expected cost

$$g(i, u) = \sum_{j=1}^{n} P_{ij}(u) g(i, u, j), \tag{1}$$

instead leads to the same cost-to-go function under any policy.

Matrix Notation. For an admissible policy, μ, let g_μ be the corresponding per step cost function:

$$g_\mu = \begin{bmatrix} g(1, \mu(1)) \\ \vdots \\ g(n, \mu(n)) \end{bmatrix}$$

where $g(i, \mu(i))$ is the cost associated with state i while following policy $\mu(i)$. For finite states and actions, we will view it as a finite dimensional vector. Similarly, let P_μ be the transition matrix for policy μ. The ij^{th} element of P_μ can be read as $P_{ij}(\mu(i))$.

We denote the Bellman operator as $T_\mu : \mathbb{R}^n \to \mathbb{R}^n$ such that

$$T_\mu J = g_\mu + \alpha P_\mu J \tag{2}$$

Notice that $T_\mu(\cdot)$ is affine function of J. The cost to-go-function $J_\mu \in \mathbb{R}^n$ corresponding to μ is the unique fixed point of T_μ. We have,

$$J_\mu = g_\mu + \alpha P_\mu J_\mu = \cdots = \sum_{t=0}^{\infty} \alpha^t P_\mu^t g_\mu = (I - \alpha P_\mu)^{-1} g_\mu.$$

The fixed point J_μ is just the solution to linear system obtained as a result of the affine Bellman operator.

We will use $G(J)$ to denote the policy class which is greedy with respect to J,

$$G(J) = \{\mu \mid T_\mu J = TJ\} = \operatorname*{argmin}_{\mu'} \left(g_{\mu'} + \alpha P_{\mu'} J\right)$$

1.1 Initial State Distribution

Initial state distribution is a probability simplex over states, $d_0 \in \Delta^{n-1}$, $d_0(i) \geq 0$ and $\sum_{i=1}^n d_0(i) = 1$. The optimal policy is optimal from any starting state, but, especially when studying approximations to an optimal policy, it can be convenient to study the expected cost-to-go starting from a randomly drawn initial state. We overload notation to define $J(d)$ as an expectation over state distribution d,

$$J(d) = \mathbb{E}_{i \sim d}[J(i)] = \sum_{i=1}^{n} J(i)\, d(i).$$

Applying this to J_μ under d_0,

$$J_\mu(d_0) = d_0(I - \alpha P_\mu)^{-1} g_\mu = \left(\sum_{t=0}^{\infty} \alpha^t d_0 P_\mu^t\right) g_\mu$$

Notice that $d_0 P_\mu^t$ is the state distribution at time t. We rewrite the infinite sum as:

$$J_\mu(d_0) = \frac{1}{1-\alpha} \sum_{i=1}^{n} d_\infty^\mu(i) g_\mu(i)$$

where d_∞^μ referred to as the discounted state occupancy measure defined as:

$$d_\infty^\mu = (1-\alpha) \sum_{t=0}^{\infty} \alpha^t d_0 P_\mu^t$$

Here, d_∞^μ can be interpreted to be the time spent in a state weighted by how early it is visited in the Markov chain. This reformulation allows us to think of the problem as minimizing the average cost of visiting the states over the frequency with which they are visited.

2 Value Iteration

Problems with a tractable finite state space can be solved by a number of classical algorithms. Value Iteration produces a convergence sequence of approximations to J^* by repeatedly applying the Bellman operator T. Our earlier analysis shows that $J_k := T^k J_0$ converges linearly to J^*. This analysis produces a fixed but highly conservative upper bound on the number of iterations required to produce an accurate cost-to-go function.

Algorithm 1 Value Iteration (VI)

Input: Initial guess: $J \in \mathbb{R}^n$, tolerance $\epsilon \in \mathbb{R}$.

(a) Bellman update for all states
for $i = 1, \ldots, n$ **do**
  $J_+(i) = \min_{u \in U(i)} g(i,u) + \alpha \sum_j P_{ij}(u) J(j)$
end for
(b) Check progress
if $||J_+ - J||_\infty \leq \epsilon$ **then**
  Stop
else
  Set $J = J_+$ & Goto (a)
end if

In pratice, it is preferable to run the algorithm up until some stopping criteron is met. The implementation described Algorithm 1 stops once successive updates produce similar cost-to-go functions.

By design, the stopping condition in step (b) is met only if $\|TJ - J\|_\infty \leq \epsilon$, and so the current cost-to-go function nearly solves Bellman's fixed point equation. The next result provides some assurances on the quality of approximate solutions to Bellman's equation.

Throughout these notes, we will take $\|\cdot\|$ to mean the infinity norm $\|\cdot\|_\infty$.

Lemma 1. *Let $T : \mathbb{R}^n \to \mathbb{R}^n$ be the Bellman operator with fixed point J^* and T_μ be the Bellman operator corresponding to policy μ. Then for any $J \in \mathbb{R}^n$:*

1. $\|J - J^*\| \leq \frac{1}{1-\alpha} \|J - TJ\|$
2. $\|J - J_\mu\| \leq \frac{1}{1-\alpha} \|J - T_\mu J\| \quad \forall\, \mu$
3. $\|J_\mu - J^*\| \leq \frac{2\alpha}{1-\alpha} \|J - TJ\| \quad \forall\, \mu \in G(J)$

Proof. We have

$$J - J^* = J - TJ + TJ - J^*.$$

By triangle inequality, we get

$$\begin{aligned} \|J - J^*\| &\leq \|J - TJ\| + \|TJ - J^*\| \\ &= \|J - TJ\| + \|TJ - TJ^*\| \\ &= \|J - TJ\| + \alpha \|J - J^*\| \qquad \text{(using contraction)} \end{aligned} \tag{3}$$

which implies,

$$\|J - J^*\| \leq \frac{1}{1-\alpha} \|J - TJ\|.$$

Following the same steps as above, we can show the second part,

$$\|J - J_\mu\| \leq \frac{1}{1-\alpha} \|J - T_\mu J\|.$$

Now we are going to prove the third part. We have

$$\begin{aligned} \|J_\mu - J^*\| &\leq \|J_\mu - T_\mu J\| + \|T_\mu J - J^*\| \\ &= \|T_\mu J_\mu - T_\mu J\| + \|TJ - TJ^*\| && (4) \\ &= \alpha \|J_\mu - J\| + \alpha \|J - J^*\| && (5) \\ &\leq \frac{\alpha}{1-\alpha} (\|J - T_\mu J\| + \|J - TJ\|) && (6) \\ &= \frac{2\alpha}{1-\alpha} \|J - TJ\|. \end{aligned}$$

where the first inequality follows by using the triangle inequality, Equation (4) uses that $T_\mu J = TJ$ since $\mu \in G(J)$, . Equation (5) follows by using that both $T(\cdot)$ and $T_\mu(\cdot)$ are contraction operators with modulus α and finally Equation (6) uses the results in parts (i) and (ii) of this Lemma. □

Convergence rate: Note that simply using contraction properties of $T(\cdot)$, one can deduce that

$$\left\|T^{k+1}J - T^k J\right\| = \alpha^k \left\|TJ - J\right\|.$$

From this, we can show that the number of iterations required by VI scales as:

$$\alpha^k \left\|TJ - J\right\| \leq \epsilon \;\Leftrightarrow\; k \geq \frac{1}{(1-\alpha)} \log\left(\frac{1}{\epsilon}\left\|J - TJ\right\|\right)$$

Note that due to the inverse scaling with $(1-\alpha)$, VI requires a considerably large iterations number of iterations for $\alpha \approx 1$.

Gauss-Seidel VI The Gauss-seidel variant of VI is a natural extension that makes ordered in-place updates. Step (a) of Algorithm 1 is modified, so that, for $i = 1, 2, \cdots n$, we update

$$J(i) = \min_{u \in U(i)} g(i,u) + \alpha \sum_j P_{ij}(u) J(j)$$

Here the update to state 1 is the same as under VI, but updates to subsequent states uses the recently update value of $J(\cdot)$ at previous states. Gauss-Seidel value iteration generally converges faster than VI in practice. It satisfies the same worst-case guarantees as standard VI.

Asynchronous Value Iteration

The synchronous[1] nature of VI is not well suited to distributed computing. We present a template for Asynchronous Value Iteration below. The algorithm updates the cost-to-go function in place at some arbitrary sequence of states. Gauss-Seidel VI is a special case in which the sequence of states is chosen in cyclic order. Asynchronous variants of VI are tolerant to delays in making Bellman updates to particular states, an important feature for distributed computation.

Algorithm 2 Asynchronous VI

Input: $J \in \mathbb{R}^n$, tolerance $\epsilon \in \mathbb{R}$
for $k = 1, \ldots$ **do**
Pick a state $s_k \in \{1, \cdots, n\}$ to update.
$J(s_k) \leftarrow TJ(s_k)$
end for

For the purposes of analysis, let J_k denote the cost-to-go function produced at iteration k. Then Asynchronous VI updates J_k according to

$$J_{k+1}(s) \leftarrow \begin{cases} TJ_k(s) & \text{if } s = s_k \\ J_k(s) & \text{if } s \neq s_k. \end{cases} \tag{7}$$

The next result shows that Asynchronous VI converges if each state is updated infinite often.

Theorem 2. *If J_k follows (7) and $\{k | s_k = s\}$ is infinite for each state $s \in \{1, \cdots, n\}$, then $\|J_k - J^*\|_\infty \to 0$.*

[1] Each iteration requires computing a Bellman update for every state.

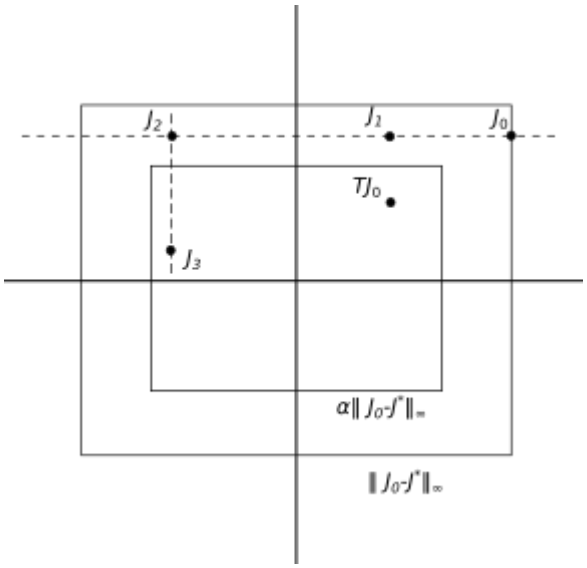

Figure 1: VI updates for $J_0 \in \mathbb{R}^2$

Proof. See homework. □

The idea of the proof is captured in Figure 2. Here $J_0 \in \mathbb{R}^2$ and we recall that T is a contraction in $\|\cdot\| = \|\cdot\|_\infty$. Each box is centered at J^*, the outer box has half-width $\|J_0 - J^*\|$ while the inner box has half-width $\alpha\|J - J^*\|$. Updating along one component gives J_1 which is within the box given by $\alpha \|J_0 - J^*\|_\infty$ along that component. By the monotonicty properties of the Bellman operator, updating again along the same component we get J_2 which is still within the same bounds. Once we perform an update along the other component we get J_3 which now falls completely within the box given by $\alpha \|J_0 - J^*\|_\infty$. Repeating this argument, we can see that once every state is updated a second time, the cost-to-go function will fall within an even smaller box with half-width $\alpha^2\|J_0 - J^*\|$. The shrinking of these boxes implies $J_k \to J^*$.

Waning: One feature of the drawing above is incorrect. Given that J_1 is in the top right quadrant of the picture, J_2 and J_3 must also be in the top right quadrant, by monotonicity properties of the Bellman operator. We'll update the picture soon

3 Error bounds that depend on the state distribution

How should we choose the sequence of states at which we make updates in Asynchronous VI? Roughly speaking, it should be useful to update J at states that matter for control decisions, rather than updating all states equally often as in standard value iteration.

In this pursuit we will take a first look at error bounds independent of the infinity norm. The following result presents the Telescoping formula for value functions. In the reinforcement learning literature, a related expression is sometimes called the the performance difference lemma. Bertsekas calls this a variational form of the Bellman equation $J_\mu = T_\mu J_\mu$.

Lemma 3. *For any $J \in \mathbb{R}^n$ and policy μ',*

$$J - J_{\mu'} = (I - \alpha P_{\mu'})^{-1}(J - TJ)$$
$$J(d_0) - J_{\mu'}(d_0) = \frac{1}{1-\alpha}(J - T_{\mu'}J)(d_\infty^{\mu'}) = \frac{1}{1-\alpha}\sum_{s\in\mathcal{S}}(J - T_{\mu'}J)(s)d_\infty^{\mu'}(s).$$

Proof. We have

$$J - J_{\mu'} = (J - T_{\mu'}J) + (T_{\mu'}J - T_{\mu'}J_{\mu'}) = (J - T_{\mu'}J) + \alpha P_{\mu'}(J - J_{\mu'}) = \sum_{t=0}^{\infty} \alpha^t P_{\mu'}^t (J - T_{\mu'}J)$$

□

We will and discuss the following corrolaries nex time.

Corollary 4. *For any $J \in \mathbb{R}^n, \mu \in G(J)$ and optimal policy μ^*,*

$$0 \preceq J_\mu - J^* \preceq \left[(I - \alpha P_{\mu^*})^{-1} - (I - \alpha P_\mu)^{-1}\right](J - TJ)$$

To interpret this, consider left multiplying each side by the initial distribution d_0. Note that the term $d_0(I - \alpha P_{\mu^*})^{-1}$ is the distribution of states under the optimal policy μ^*) and $d_0(I - \alpha P_\mu)^{-1}$ the distribution of states when following policy μ. Roughly, speaking this shows that a greedy policy μ with respect to J, is near-optimal if we can control:

1. Underestimation of Bellman's equation ($J < TJ$) at states visited by the optimal policy μ^*.
2. Overestimation of Bellman's equation ($TJ < J$) at states visited by μ.

Which term is more problematic? This depends. In a sub-area of reinforcement learning called imitation learning, one observes the control decisions of an expert and attempts to learn from this a policy with similar behavior. There, distribution shift is a major issue: an expert driver never reaches certain dangerous states, but our imitated policy μ may reach those states. Due to lack of training data, the estimated cost-to-go function could be highly inaccurate at those sates and Corollary 4 shows how this contributes to poor decision performance.

In most other settings, the dependence on the state distribution $d_0(I - \gamma P_{\mu^*})$ under the optimal policy is much more problematic, because the optimal policy μ^* is unknown. Much effort has gone into avoiding this issue by producing *optimistic* approximations to the cost-to-go function. As shown by the following corollary, the greedy policies induced by an optimistic cost-to-go function obey error bounds that depend only on the gap in Bellman's equation *at the states this policy actually visits.* We will see that this plays a key role both in the design of exploration algorithms in reinforcement and in the linear programming approach to approximate dynamic programming.

Corollary 5. *$\forall J, \mu \in G(J)$ and optimal policy μ and if $J \preceq J^*$,*

$$0 \preceq J_\mu - J^* \preceq (I - \alpha P_\mu)^{-1}(TJ - J)$$