Course Notes On Dynamic Optimization (Fall 2023)
Lecture 9A: Approximate value iteration

Instructor: Daniel Russo

Email: djr2174@gsb.columbia.edu

Graduate Instructor: David Cheikhi
Email: d.cheikhi@columbia.edu

These notes are partly based of scribed notes from a previous edition of the class. I have done some follow up light editing, but there may be typos or errors.

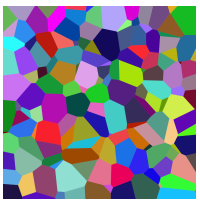
1 Problem setup

Throughout these notes we continue to study indefinite horizon problems under the assumption that all policies eventually reach the terminal state.

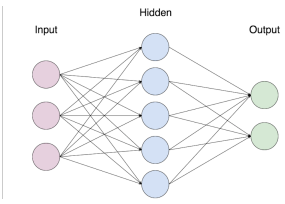
2 Parametric approximations to the value function

In many problems, the number of possible distinct states scales exponentially in some measure variable – the number of queues in a , the length of the lead time in inventory control, etc. Computing or *even storing* the cost-to-go function for each possible state quickly becomes intractable.

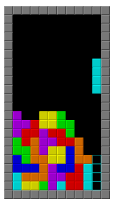
A large set of techniques in the fields of approximate dynamic programming and reinforcement learning attempt to overcome the curse-of-dimensionality by fitting (or 'learning') a parametric approximation of the cost-to-go function J^*.



(a) State aggregated approx.



(b) Neural network approx.



(c) Tetris

2.1 State aggregation

Suppose we believe apriori that some states should be 'similar' to others. State aggregated representations exploit this by ignoring the (hopefully unimportant) distinctions between similar states. See Figure 1a.

For any partition of the states and choice of canonical representatives of the subsets that form the partition, we can define a canonical surjective map $\phi : \mathcal{X} \to \{\bar{x}_1, \ldots, \bar{x}_m\}$ which sends states to the canonical representative of the cluster the state is a member of (conversely, any such ϕ induces a partition). Our general goal is to come up with algorithms whose computational and statistical complexity scales with the number of partitions (as opposed on the number of original states).

The class of state-aggregated cost-to-go functions

$$\mathcal{J}_\phi = \{J \in \mathbb{R}^{\mathcal{X}} \mid J(x) = J(x') \; \forall x, x' \; \phi(x) = \phi(x')\}$$

consists of all rules for assigning real numbers to states that do not distinguish between sates in a common cluster. Any member J of this class can be encoded by a parameter $\theta \in \mathbb{R}^k$ of length equal to the number of clusters by defining $\theta_i = \phi(\bar{x}_i)$ and setting

$$J_\theta(x) = \theta_i \iff \phi(x) = \bar{x}_i.$$

For algorithms that use this approximation to be efficient, we require that $\phi(\cdot)$ is efficient computable.

2.2 Linear function approximation

In a game of Tetris (Fig 1c) we can describe the current state by defining a binary variable for each possible cell in the grid. For a 10×10 grid, we would then have $\approx 2^{100}$ possible states. Instead, one might seek to approximate the optimal cost-to-go as

$$J^*(x) \approx \langle \phi(x), \hat{\theta} \rangle,$$

by searching for the weights θ^*. Successful early applications use handcrafted features vectors defined by

$$\phi(s) = \begin{bmatrix} \text{height of column 1} \\ \vdots \\ \text{height of column } n \\ |\text{intercol. height diff. 1}| \\ \vdots \\ |\text{intercol. height diff. } n-1| \\ \text{Maximum column height} \end{bmatrix},$$

which has only 20 dimensions. The design of these features reflects a lot of human intuition about the game.

2.3 Neural network

Since about 2012, most impressive demonstrating of reinforcement learning have use neural networks (Fig 1c) rather than linear models.

3 Fitted value iteration

Algorithm 1 Fitted value iteration – idealized version

Require: $J_{\theta_0} \in \mathbb{R}^{|\mathcal{X}|}$, and iteration limit N
1: **for** Episode $n = 0, 1, 2, \ldots, N-1$ **do**
2: Fit $J_{\theta_{n+1}} \in \arg\min_{\hat{J} \in \mathcal{J}_\Theta} \|\hat{J} - TJ_\theta\|_{2,w}^2$.
3: **end for**
4: **return** J_{θ_N}

3.1 Practical variants

The idealized fitted value iteration algorithm 1 is a fundamental abstraction for studying issues in reinforcement learning and approximate dynamic programming. Here we describe implementable variants of the algorithm, working up to a method that looks like Q-learning.

Algorithm 2 approximates the idealized Algorithm 1 by sampling a finite number of states from w. This require an ability to a) sample from w and b) compute $TJ_\theta(x)$ for any given state x. Think of tetris, where it is hopeless to loop over all possible states, but you could comptue the the Bellman backup $TJ_\theta(x)$ for a given game state, since the set of possible succesor states to x can be easily enumerated. Batch fitted value iteration requires exactly solving an optimization problem

Algorithm 2 Batch fitted value iteration

Require: $J_{\theta_0} \in \mathbb{R}^{|\mathcal{X}|}$, and iteration limit N, sampling distribution w, sample size M,
1: **for** Episode $n = 0, 1, 2, \ldots, N-1$ **do**
2: **for** $m = 1, \ldots, M$ **do**
3: Sample $x_m \sim w$
4: Compute $J^+(x_m) \leftarrow \min_{u \in U(x_m)} g(x_m, u) + \sum_{x' \in \mathcal{X}} p(x'|x_m, u_m) J_{\theta_n}(x')$.
5: **end for**
6: Fit $\theta_{n+1} = \arg\min_\theta \sum_{m=1}^M (J_\theta(x_m) - J^+(x_m))^2$.
7: **end for**
8: **return** J_{θ_N}

to generate the next paramter θ_{n+1}. It seems wasteful to *exactly* compute the Bellman backup of a cost-to-go function that is anyway . The next algorithm instead makes a single gradient update. (In between these two is a version that makes a fixed number of gradient updates.)

Algorithm 3 still requires the ability to plan locally, i.e. to solve $TJ_\theta(x)$ at a given state x. Moreover, we'd need to continue to do that in an online fashion in order to implement the greedy policy with respect to the final cost-to-go function J. To avoid this, it is common to work with so-called Q functions, which track a cost-to-go for each possible state and each hypothetical control

Algorithm 3 SGD style fitted value-iteration

Require: $J_{\theta_0} \in \mathbb{R}^{|\mathcal{X}|}$, and iteration limit N, distribution w

1: **for** $n = 0, 1, 2, \ldots, N-1$ **do**
2: Sample $x_n \sim w$
3: Compute $J^+(x_n) \leftarrow \min_{u \in U(x_n)} g(x_n, u) + \sum_{x' \in \mathcal{X}} p(x'|x_n, u_n) J_{\theta_n}(x')$.
4: Update $\theta_{n+1} = \theta_n - \nabla_\theta (J_\theta(x_n) - J^+(x_n))^2 \Big|_{\theta=\theta_n}$.
5: **end for**
6: **return** J_{θ_N}

decision. The optimal Q function is the unique solution to the fixed point equation,

$$Q^*(x, u) = g(x, u) + \sum_{x' \in \mathcal{X}} p(x'|x, u) \min_{u' \in U(x')} Q^*(x', u') \qquad \forall x, u \in \mathcal{U}(x).$$

Algorithm 4 modifies Algorithm 3 to work with Q functions. This method is called Q-learning (though practical implementations may differ in terms of how the states at which updates occur are sampled.)

Algorithm 4 SGD style fitted Q-iteration

Require: $Q_{\theta_0} \in \mathbb{R}^{|\mathcal{X}|}$, and iteration limit N, distribution w over $\mathcal{X} \times \mathcal{U}$.

1: **for** $n = 0, 1, 2, \ldots, N-1$ **do**
2: Sample $(x_n, u_n) \sim w$
3: Apply (x_n, u_n) and observe the next state x'_n.
4: Compute $Q^+(x_n, u_n) \leftarrow g(x_n, u_n) + \min_{u \in U(x'_n)} Q_{\theta_n}(x'_n, u)$.
5: Update $\theta_{n+1} = \theta_n - \nabla_\theta (Q_\theta(x_n, u_n) - Q^+(x_n, u_n))^2 \Big|_{\theta=\theta_n}$.
6: **end for**
7: **return** Q_{θ_N}

4 Guarantees for approximate value iteration

We give some guarantees for approximate value iteration. These are, in general, the best one can do. But they do not paint a very promising story.

Recall that T is a contraction in the norm $\|J\|_{\infty,1/V} = \sup_{x \in \mathcal{X}} \frac{|J(x)|}{V(x)}$ with modulus $\alpha = (\|V\|_\infty - 1)/\|V\|_\infty$. That is,

$$\|TJ - TJ'\|_{\infty,1/V} \leqslant \alpha \|J - J'\|_{\infty,1/V},$$

where we defined

$$V(x) = \sup_\pi \mathbb{E}^\pi[\tau \mid x_0 = x].$$

What can we say about approximate value iteration?

Define the approximate bellman operator given by

$$\hat{T}J = \arg\min_{\hat{J}\in\mathcal{J}_\Theta} \|\hat{J} - TJ\|_{2,w}^2.$$

View this as on operator on the space $\mathcal{J}_\Theta$ and define the error measure on that space

Already something is a little weird here – or at least unfortunate – here. We are approximating the Bellman operator in one norm but it is a contraction in a different norm.

The next generic lemma bound performance degredation when you plan with respect to an cost-to-go function.

Lemma 1. *If $\mu \in G(J)$ then $\|J_\mu - J^*\|_{\infty,1/V} \leqslant \frac{1}{(1-\alpha)}\|J - J^*\|_{\infty,1/V}$.*

Proof. Given below. □

How big is the error in the cost-to-go? We bound it here in terms of the horizon and a very severe notion of approximation error.

Lemma 2 (Cost-to-go error)**.** *Under the approximate value iteration scheme where $J_0 \in \mathcal{J}_\Theta$ and*

$$J_n = \hat{T}J_{n-1} \quad n = 1, \ldots,$$

the following error bound is satisfied:

$$\limsup_{n\to\infty} \|J_n - J^*\|_{\infty,1/V} \leqslant \frac{2\alpha}{(1-\alpha)} \cdot \epsilon_\Theta$$

where

$$\epsilon_\Theta = \sup_{J\in\mathcal{J}_\Theta} \|\hat{T}J - TJ\|_{\infty,1/V}.$$

Proof. Your homework! □

If we put these two upper bounds together we accrue two factors of the 'horizon' $1/(1-\alpha)$.

Corollary 1. *Consider the approximate value iteration scheme where $J_0 \in \mathcal{J}_\Theta$ and $J_n = \hat{T}J_{n-1}$ for $n = 1, 2, \ldots$.*

Then,

$$\limsup_{n\to\infty} \|J_{\mu_n} - J^*\|_{\infty,1/V} \leqslant \frac{2\alpha}{(1-\alpha)^2} \cdot \epsilon_\Theta$$

4.1 Unfortunate features of this bound

Horizon dependence In discounted problems, α is simply the discount factor,and $1/(1-\alpha)$ is the effective horizon. In finite horizon problems with N periods, $1/(1-\alpha) = N$ is exactly equal to the horizon. The amplification of error with the square of the horizon is quite unfortunate, suggesting that even on a per-period basis, decision-quality degrades with the length of the horizon.

Th need for uniform approximation. ϵ_Θ depends on a (weighted) infinity norm of error, and so in a sense its is small only if parametric approximation is accurate uniformly across the state space. Moreover, while we might expect that our basis functions reflect well the cost-to-go under 'good' policies, ϵ_Θ seems to require that we can accurately reprent the Bellman backup of an arbitrary input J. The next definition formalizes the scenario in which $\epsilon_\Theta = 0$.

Definition 1. The class of value functions $\mathcal{J}_\Theta$ is *closed* under Bellman operators if $TJ \in \mathcal{J}_\Theta$ for each $J \in J_\Theta$.

Some recent 'RL Theory' literature calls this condition "completeness' rather than closure.

5 Proof of Lemma 1

Proof. Recall that V is a Lyapunov function, meaning that for every possible policy μ'

$$P_{\mu'} V \preceq \alpha V. \tag{1}$$

Now, set $\delta = \|J - J^*\|_{\infty,1/V}$. Then for all x,

$$J(x) - \delta V(x) \leqslant J^*(x) \leqslant J(x) + \delta V(x),$$

i.e.

$$J - \delta V \preceq J^* \preceq J + \delta V.$$

Using monotonicity and the Lyapunov condition yields

$$\begin{aligned}
T_\mu J^* &\preceq T_\mu (J + \delta V) && \text{(Monotonicity)} \\
&= g_\mu + P_\mu (J + \delta V) && \text{Definition} \\
&= T_\mu J + \delta P_\mu V \\
&\preceq T_\mu J + (\delta\alpha) V && \text{(Lyapunov condition)} \\
&= TJ + (\delta\alpha) V && \text{(Since } \mu \in G(J)) \\
&\preceq T(J^* + \delta V) + (\delta\alpha) V && \text{(Monotonicity)} \\
&= \min_{\mu'} (T_{\mu'} J^* + \delta V) + (\delta\alpha) V && \text{(Definition)} \\
&\preceq \min_{\mu'} T_{\mu'} J^* + \delta \max_{\mu''} T_{\mu''} V + (\delta\alpha) V \\
&\preceq \min_{\mu'} T_{\mu'} J^* + 2(\delta\alpha) V && \text{(Lyapunov condition)} \\
&= TJ^* + 2(\delta\alpha) V \\
&= J^* + 2(\delta\alpha) V && (J^* \text{ is a fixed point}).
\end{aligned}$$

The notation $\min_{\mu'} T_{\mu'} J$ refers to element-wise minimization that comes from picking $\mu'(x) = \arg\min_{u \in U(x)} g(x,u) + \sum_{x' \in \mathcal{X}} p(x'|x,u) J(x')$.

We showed $T_\mu J^* \preceq J^* + 2(\delta\alpha)V$. Applying T_μ to both sides and using the Monotinicty property and the Lyapunov property of V gives

$$T_\mu^2 J^* \preceq T_\mu J^* + 2(\delta\alpha)\alpha V \preceq 2\delta(\alpha + \alpha^2) V.$$

Repeating this inductively, we get

$$J_\mu = \lim_{k\to\infty} T_\mu^k J^* \preceq \frac{2\delta}{1-\alpha} V.$$

We get

$$0 \preceq J_\mu - J^* \preceq \frac{2\delta}{1-\alpha} V$$

so $\sup_{x\in\mathcal{X}} \frac{|J_\mu(x)-J^*(x)|}{V(x)} \leqslant \frac{2\delta}{1-\alpha}$ as desired. □