17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 10: Approximate Policy Iteration

Lecturer: Drew Bagnell
Scribe: Benji Li, Riku Arakawa, Vansh Kapoor

Notations In Drew's lectures, we typically think about costs instead of rewards,[1] and use T to denote horizon instead of H.

10.1 Approximate Dynamic Programming

Depending on the level of knowledge about the environment we have access to (transition dynamics, reward function, etc), as well as the amount of privilege we have in our experimental setup (such as the ability to reset to a previous state), we can use different models to sample and learn from our environment.

The following list includes different types of access models in RL.

1. Full probabilistic description of the environment: In this model, the algorithm is given full description of $\{p(s' \mid s, a), T, c(s, a)\}$. In a tabular MDP (with a moderate number of states and actions), we show in a prior lecture that value iteration provides a straightforward way to learn the optimal value functions, given the full probabilistic description of the environment.

2. Deterministic Simulative Model: In its simplest form, a deterministic simulative model provides a function that maps $(x, a) \to x'$ deterministically. More generally, even when the underlying dynamics are stochastic, we may still have access to a fixed random seed within a computer program. This allows us to perfectly recreate trajectories, including all randomness that occurred. Such access is common in computer simulations, where reproducibility is desired.

3. Generative models: In this model, we have programmatic access to state transitions, meaning we can place the system in any desired state and observe its evolution. This enables flexible exploration and controlled experimentation, making it a powerful tool for understanding and optimizing decision processes.

[1] However, he may also switch midway in the lecture.

4. Reset models: In this model, we can execute a policy or simulate rollouts at any time, with the ability to reset the system to a known state or a predefined distribution over states. This feature makes it particularly useful in controlled settings, such as robotics experiments, where a robot can be repeatedly reset to stable configurations for consistent evaluation and testing.

5. Single Trace: This model is the most challenging, where actions are irreversible, and past states cannot be revisited. The trace model captures this fundamental constraint—the inability to ”reset” in real-world decision-making.

10.2 Approximating Value and Q Iteration

In this lecture, we will assume costs are deterministic to simplify notations. Recall the Bellman optimality equations and the Bellman equations in terms of action value functions.

$$
\begin{aligned}
Q^*(s,a,t) &= c(s,a) + \mathbb{E}_{p(s'|s,a)}[\min_{a'} Q^*(s',a',t+1)] \\
Q^*(s,a,t) &= c(s,a) + \text{Total future value of acting optimally} \\
Q^\pi(s,a,t) &= c(s,a) + \text{Total future value of following policy } \pi \\
Q^\pi(s,a,t) &= c(s,a) + \mathbb{E}_{p(s'|s,a)}[Q^\pi(s,\pi(a,t+1),t+1)]
\end{aligned}
$$

Note that one could derive Bellman equations in terms of state value functions as well. There are some pros and cons.

Pros of Action Value Functions Computing the optimal policy from Q^* is simpler than extracting it from V^*. With Q^*, we can obtain the optimal action using a straightforward $\arg\max$, without needing to evaluate expectations or rely on a transition model. Once we have Q^*, we don't need a transition model at all to determine the optimal policy.

Cons of Action Value Functions Action-value functions require more memory than state-value functions. While a value function only needs to store a value for each state ($|\text{States}|$), an action-value function must store values for every state-action pair ($|\text{States}| \times |\text{Actions}|$), leading to a significantly larger space requirement.

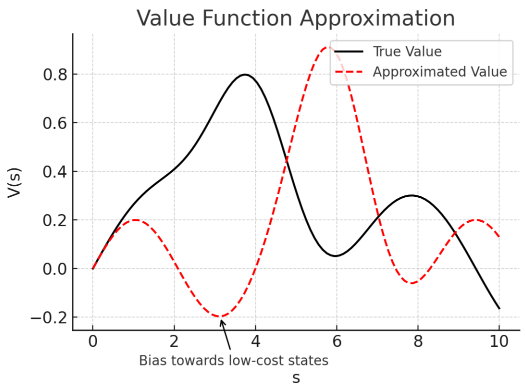
We will depart from the tabular setting in earlier lectures, where we can afford to enumerate all state-action pairs. Instead, we will introduce approximations to algorithms like value iteration and policy iteration. The first algorithm is *Fitted Q-Iteration*, an approximate dynamic programming algorithm that learns approximate action-value functions from data.

Algorithm 1 Fitted Q-Iteration

```
Require: Dataset $\{(s_i, a_i, c_i, s_i')\}_{i=1}^N$, horizon $T$
 1: Initialize: $Q(s, a, T) \leftarrow 0$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
 2: for $t \in [T-1, T-2, \ldots, 0]$ do
 3:     $D^t \leftarrow \varnothing$
 4:     for $i = 1, \ldots, N$ do
 5:         $input \leftarrow (s_i, a_i)$
 6:         $target \leftarrow c_i + \min_{a'} Q\big(s', a', t+1\big)$
 7:         $D^t \leftarrow D^t \cup \{(input, target)\}$
 8:     end for
 9:     $Q(\cdot, \cdot, t) \leftarrow \text{Regression}(D^t)$
10: end for
11: return $Q$
```

Challenges with Fitted Q-Iteration In class, we went through some interesting toy examples presented in [1], in which Fitted Q-Iteration fails even in settings where the true value function lives in the function class we perform regression over. Fitted Q-Iteration and its counterpart, Fitted Value Iteration suffer from bootstrapping issues and sometimes fail to converge. These methods approximate the value function inductively, propagating and even amplifying errors, leading the algorithm to favor suboptimal actions.

The core issue lies in the *minimization step* when generating target values. This step can push the policy toward states where the approximate value function underestimates the true value, making them appear deceptively attractive. This bias is especially severe in sparsely sampled regions of the state space, where poor generalization may result in policies favoring undesirable states. From a learning theory perspective, this violates the i.i.d. assumption on training and test samples.



10.3 Approximate Policy Iteration

One major problem encountered when approximating the optimal Q-function is that of overestimation, which tends to amplify errors during the update process. In addition, there is the issue of covariate shift. As the algorithm updates its policy based on the approximated Q-function, the distribution of state-action pairs encountered during training shifts away from the one initially used to learn the approximation. Consequently, when the improved policy is deployed, it may encounter states that were underrepresented (or even absent) in the training data. This mismatch between the training and deployment distributions further exacerbates the error propagation and amplification issues.

A common remedy to mitigate these issues is to shift from using the optimal $Q^\star$ directly, to instead employing a policy-dependent Q-function, denoted as Q^π. This leads naturally to the use of policy iteration rather than direct optimization of $Q^\star$. Here, there are two fundamental steps: policy evaluation and policy improvement.

10.3.1 Policy Evaluation

The algorithm is shown in Algorithm 2. Instead of $Q^\star$, we are trying to estimate Q^π.

Algorithm 2 Policy Evaluation

```
Require: Dataset $\{(s_i, a_i, c_i, s'_i)\}_{i=1}^{N}$, horizon $T$, discount factor $\gamma$
 1: Initialize: $Q(s, a, T) \leftarrow 0$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
 2: for $t = T-1, T-2, \ldots, 0$ do
 3:     $D^t \leftarrow \varnothing$
 4:     for $i = 1, \ldots, N$ do
 5:         $input \leftarrow (s_i, a_i)$
 6:         $target \leftarrow c_i + \gamma\, Q^{\pi}\big(s'_i,\, \pi(s'_i, t+1),\, t+1\big)$
 7:         $D^t \leftarrow D^t \cup \{(input, target)\}$
 8:     end for
 9:     $Q(\cdot, \cdot, t) \leftarrow \text{Regression}(D^t)$
10: end for
11: return $Q$
```

10.3.2 Policy Improvement

When using an approximated $Q^{\pi}(s, a, t)$ function to derive a policy, a straightforward approach is to select the action that maximizes the estimated value:

$$\pi^{\text{proposed}}(s, t) = \text{argmax}_a Q^{\pi}(s, a, t) \tag{10.1}$$

However, this direct maximization can still suffer from the same issues mentioned earlier (e.g., overestimation or instability due to covariate shift). To ensure a more stable update, we can interpolate between the current policy and the proposed policy. This approach is known as conservative policy iteration. The updated policy is given by:

$$\pi'(s, t) = (1 - \alpha)\pi(s, t) + \alpha\pi^{\text{proposed}}(s, t), \tag{10.2}$$

where $\alpha \in (0, 1]$ is a step-size parameter that controls the degree of change in the policy at each iteration.

10.4 Policy Search by Dynamic Programming

The previous algorithms discussed (Fitted Q-iteration and Conservative P.I.) rely on Bellman back-up and approximation of the action value functions. Unlike value-based methods that derive policies from value estimates, policy search by dynamic programming (PSDP) directly optimizes the policy itself by going backward in time. The intuition is that if we have already obtained the optimal (non-stationary) policies from time step $(t+1)$ and onward, then the optimization problem at time step t becomes much simpler: we can simply choose the action that maximize the total reward given that we have committed to following the policies $\{\pi_{t+1}, \dots \pi_{T-1}\}$ for later steps. By unrolling the policy at every step until termination, we could then avoid the issues of overestimation and compounding errors suffered by the previous algorithms. As we deploy the policy at every time step until termination, we incur a time complexity of $\mathcal{O}(T^2)$.

Algorithm 3 Policy Search by Dynamic Programming

```
 1: Initialize: $Q(s, a, T) \leftarrow 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
 2: for $t = T-1, \dots, 0$ do
 3:     $D_t \leftarrow \varnothing$
 4:     for each state $s_i \in \mathcal{S}$ do
 5:         for each action $a_j \in A$ do
 6:             Compute target as the sum of future rewards:
                target = sum of future rewards executing $\pi_t, \pi_{t+1}, \dots, \pi_T$
 7:             $D_t \leftarrow D_t \cup \{(s_i, a_j, \text{target}_{ij})\}$
 8:         end for
 9:     end for
10:     $\hat{Q}_t \leftarrow \text{Regress}(D_t)$
11:     Update policy: $\pi_t = \arg\max_a \hat{Q}_t(s, a)$
12: end for
13: return $Q$
```

PSDP achieves the following performance bound.

Value Function Bound

$$V_\pi(s_0) \geq V_{\pi_{ref}}(s_0) - \sum_{t=0}^{T-1} \epsilon \left\| \frac{\partial \pi^t_{ref}}{\mu_t} \right\|_\infty$$

where $\partial \pi^t_{ref}(t), \mu(t)$ denote the state distributions induced by the policies π, π_{ref} at time step h, respectively, and ϵ relates to regression loss. We will give the analysis using the performance difference lemma in the next lecture.

10.5 *

References

[1] Justin Boyan and Andrew Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, 7, 1994.