COS 435/ECE 433 Week 4 Precept Notes

February 26, 2025

1 Dynamic Programming

Dynamic programming is widely applied to learn the *optimal policy* of a sequential decision-making process. Recall that we have the following identities for value function with respect to a Markov policy π, known as the Bellman equations:

$$Q^\pi(s,a) = r(s,a) + \gamma \cdot \mathbb{E}_{s' \sim p(\cdot|s,a)}[V^\pi(s')], \tag{1}$$
$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)}[Q^\pi(s,a)] \tag{2}$$

for every $s \in \mathcal{S}$ and $a \in \mathcal{A}$, where

$$V^\pi(s) = \mathbb{E}_\pi\left[\sum_{h=0}^{\infty} \gamma^h r(s_h,a_h) \middle| s_0 = s\right],$$
$$Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{h=0}^{\infty} \gamma^h r(s_h,a_h) \middle| s_0 = s, a_0 = a\right].$$

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. For a finite state/action MDP, when the reward and transition is known, solving Eq. (1) is equivalent to solving a linear system with $O(|\mathcal{S}|)$ equations. Instead, we can estimate the value function by iterative dynamic programming:

$$V_k^\pi(s) = R(s) + \gamma \cdot \sum_{s'} P^\pi(s'|s) V_{k-1}^\pi(s'), \tag{3}$$

where $R(s) := \sum_a r(s,a)\pi(a|s)$, $P^\pi(s'|s) = \sum_a \pi(a|s)p(s'|s,a)$, and $V_0^\pi(s) = 0$ for all $s \in \mathcal{S}$. The following argument supports such an algorithm:

$$\mathbf{V}^\pi = \mathbf{R} + \gamma \mathbf{P}^\pi \cdot \mathbf{V}^\pi,$$

where $\mathbf{V}^\pi = (V^\pi(s))_{s\in\mathcal{S}}$, $\mathbf{P}^\pi_{\mathbf{s},\mathbf{s}'} = (P^\pi(s'|s))$. Note that the eigenvalue of $\mathbf{P}^\pi$ is always less than 1 (Gershgorin circle theorem), we have

$$\mathbf{V}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1}\mathbf{R} = \sum_{i \geq 0} (\gamma \mathbf{P}^\pi)^i \mathbf{R},$$

which exactly gives us Eq. (3).

Policy Learning. In Reinforcement Learning, we aim to maximize the expected discounted return:

$$\max_{\pi} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, a_t)\Big],$$

where $s_{t+1} \sim P(\cdot \mid s_t, a_t)$, $a_t \sim \pi(\cdot \mid s_t)$, and $0 < \gamma < 1$. If the MDP model (transition probabilities P and reward function r) is known, we can use Dynamic Programming (DP) to find the optimal value function V^* and corresponding optimal policy π^*.

Bellman Optimality Equations

- **Optimal State Value Function:**

$$V^*(s) \;=\; \max_a \Big[r(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V^*(s')\Big].$$

- **Optimal State-Action Value Function:**

$$Q^*(s,a) \;=\; r(s,a) + \gamma \sum_{s'} P(s' \mid s,a) \, \max_{a'} Q^*(s',a').$$

- These equations characterize the Bellman optimality condition and form the basis of DP solutions for MDPs.

Value Iteration

- **Iteration Update:**

$$V_{k+1}(s) \;=\; \max_a \Big[r(s,a) + \gamma \sum_{s'} P(s' \mid s,a)\, V_k(s')\Big].$$

- **Convergence:**
  - Under $\gamma < 1$, the above update is a γ-contraction in the $\|\cdot\|_\infty$ norm.
  - By the Banach fixed-point theorem, $V_k \to V^*$ as $k \to \infty$.
  - Error bound:

$$\|V_k - V^*\|_\infty \;\le\; \gamma^k \, \|V_0 - V^*\|_\infty.$$

Convergence Speed

- The factor γ^k indicates exponential decay of the error in terms of the number of iterations k.
- Another related bound:

$$\|V_{k+1} - V^*\|_\infty \;\le\; \frac{\gamma}{1-\gamma} \|V_{k+1} - V_k\|_\infty.$$

- Both show that once the value function updates become small, V_k is close to V^*.

Key Points

- **Core objective:** Maximize the expected discounted return to find the optimal policy π^*.
- **DP:** When P and r are known, use value iteration or policy iteration to solve for V^* and π^*.
- **Bellman optimality:** Defines the self-consistent equations for V^* and Q^*.
- **Convergence:** Value iteration is a contraction mapping, ensuring exponential convergence to V^*.
- **Practical note:** For large or unknown MDPs, approximate methods or model-free RL algorithms (e.g., Q-learning) are often used, but the DP viewpoint remains foundational.

2 Policy Gradient

Background and Problem Setup We consider a finite-horizon Markov Decision Process (MDP) with horizon T. At each time step $t = 0, 1, \ldots, T-1$, the agent observes a state s_t and selects an action a_t according to a parameterized policy $\pi_\theta(a_t \mid s_t)$, where θ denotes the parameters of the policy. The environment then produces the next state s_{t+1} and a corresponding reward r_t. Thus, we obtain a complete trajectory:

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T).$$

The reinforcement learning problem is then formulated as an optimization problem: find the optimal parameter θ that maximizes the expected cumulative reward

$$J(\theta) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{T-1} r_t \right],$$

where $p_\pi(\tau)$ denotes the probability distribution of the trajectory τ under the policy π_θ.

Our goal is to maximize $J(\theta)$ using gradient ascent (or, equivalently, minimizing the negative objective with gradient descent), which requires us to compute:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{T-1} r_t \right].$$

Policy Gradient Derivation

1. From Expectation Form to Log-Likelihood Gradient

We start with the gradient of the expected cumulative reward:

$$\nabla_\theta \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{T-1} r_t \right] = \int \nabla_\theta p_\pi(\tau) \left(\sum_{t=0}^{T-1} r_t \right) d\tau.$$

Using the identity

$$\nabla_\theta p_\pi(\tau) = p_\pi(\tau) \nabla_\theta \log p_\pi(\tau),$$

we can rewrite the expression as:

$$\int p_\pi(\tau) \nabla_\theta \log p_\pi(\tau) \left(\sum_{t=0}^{T-1} r_t \right) d\tau.$$

This is often called the “log trick”. The probability of a trajectory under the policy is given by:

$$p_\pi(\tau) = p(s_0) \prod_{t=0}^{T-1} \Big[\pi_\theta(a_t \mid s_t)\, p(s_{t+1} \mid s_t, a_t)\Big],$$

where $p(s_0)$ is the initial state distribution, $\pi_\theta(a_t \mid s_t)$ is the policy, and $p(s_{t+1} \mid s_t, a_t)$ is the environment's transition probability (which does not depend on θ).

Thus, we obtain:

$$\nabla_\theta J(\theta) = \int p_\pi(\tau)\, \nabla_\theta \log p_\pi(\tau) \left(\sum_{t=0}^{T-1} r_t\right) d\tau.$$

2. Sampling Form (Monte Carlo Estimate)

Since it is infeasible to sum over all possible trajectories, we approximate the expectation using Monte Carlo sampling. That is, we sample N trajectories $\tau^{(i)}$ from the policy π_θ and approximate the gradient as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T-1} r_t^{(i)}\right) \nabla_\theta \log p_\pi(\tau^{(i)}).$$

Often, the log-probability $\log p_\pi(\tau)$ is decomposed as a sum over time steps:

$$\log p_\pi(\tau) = \log p(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T-1} \log p(s_{t+1} \mid s_t, a_t).$$

Since the environment's transition probabilities do not depend on θ, their gradients vanish. Thus, we have:

$$\nabla_\theta \log p_\pi(\tau) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

This leads to the following gradient estimate:

$$\boxed{\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T-1} r_t^{(i)}\right) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}).}$$

3. Policy Gradient Update

Once we have an estimate for $\nabla_\theta J(\theta)$, we update the parameters using stochastic gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta),$$

where α is the learning rate.

Variance Issues and Common Techniques (Brief summary)

1. **High Variance:** The cumulative reward $\sum_{t=0}^{T-1} r_t^{(i)}$ can vary significantly across trajectories, leading to high variance in the gradient estimate. Moreover, each trajectory must be fully executed to obtain the reward, which can be inefficient for long horizons.

2. **Baseline:** Introducing a baseline function $b(s_t)$ in the gradient estimator can reduce the variance. A common choice is to use a constant baseline or the state value function $V^\pi(s_t)$. Since $\mathbb{E}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)] = 0$, subtracting a function that does not depend on the action does not introduce bias but can significantly reduce variance.

3. **Advantage Function:** Instead of using the total reward, one can use the advantage function, defined as $Q^\pi(s_t, a_t) - V^\pi(s_t)$, which often leads to a lower variance and faster convergence.

4. **Actor-Critic Methods:** In practice, policy gradient methods are frequently combined with a value function estimator (critic) to form an actor-critic architecture. The critic estimates the value or advantage function, while the actor updates the policy parameters. This approach is generally more efficient than using pure Monte Carlo estimates.

Conclusion and Summary

- The core idea of policy gradient methods is:
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\left(\sum_{t=0}^{T-1} r_t \right) \nabla_\theta \log p_\pi(\tau) \right].$$
By sampling trajectories, we obtain an unbiased estimate of the gradient.

- The Monte Carlo estimate of the gradient is:
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{T-1} r_t^{(i)} \right) \nabla_\theta \log p_\pi(\tau^{(i)}).$$

- The parameter update rule is:
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta).$$

- High variance in the gradient estimate is mitigated by techniques such as using baselines, advantage functions, or actor-critic architectures.

- For continuous action spaces, a **Gaussian Policy** is often employed, parameterized as $\pi_\theta(a \mid s) = \mathcal{N}(\mu_\theta(s), \Sigma_\theta(s))$, which allows the computation of $\nabla_\theta \log \pi_\theta(a \mid s)$ using standard calculus.

Drawbacks of Vanilla Policy Gradient methods. (1) The high variance of the policy gradient results in instability in the training process. (2) The on-policy nature of policy gradient: only utilizing $\{(s_h^k, a_h^k)\}_{h\geq 0}$ when estimating the k-th gradient, causing the waste of previous data.

3 Methods to reduce variance.

We talked about the high variance issue in the previous section. Now, let's look at the methods to reduce the variance.

We aim to prove We first prove the following lemma.

Lemma 3.1. ***(Causality).*** *For any time step t and any earlier step $u < t$ in a Markov Decision Process,*
$$\mathbb{E}\Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot r_u\Big] = 0.$$
In other words, the action a_t at time t does not affect any reward r_u obtained at a time step $u < t$.

Proof (Sketch).

1. *Markov property.* The environment's state transitions and rewards depend only on the current state and action, not on future actions. Once a reward r_u at time u has been generated, it cannot be altered by any action taken at a later time $t > u$.

2. *Expectation argument.* Consider the expectation

$$\mathbb{E}\big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, r_u\big].$$

Because $t > u$, r_u is determined before a_t is even chosen, and there is no causal pathway by which a_t can influence r_u. Under the Markov property, this expectation is zero.

3. *Implication.* Whenever we multiply $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ by any function that involves only time steps from t onward, we do not introduce additional bias in the gradient estimate. Conversely, multiplying by rewards from times $u < t$ yields zero in expectation.

Reward to Go Using the Causality Lemma,

we can replace the total return of the entire trajectory with the *reward to go* starting at each time t. Formally,

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Big(\sum_{k=t}^{T} r_k\Big)\Big].$$

Here, $\sum_{k=t}^{T} r_k$ is the cumulative reward from time t through the final step T. By causality, the action a_t has no influence on any reward r_u with $u < t$, so we can safely omit those earlier rewards in the weighting term.

Why it reduces variance.

- When the gradient term $\nabla_\theta \log \pi_\theta(a_t \mid s_t)$ is multiplied by the entire return $\sum_{k=0}^{T} r_k$, it includes rewards r_u for $u < t$ that have no causal dependence on a_t. This unnecessary inclusion increases variance in the gradient estimate.

- By focusing only on $\sum_{k=t}^{T} r_k$, we eliminate those past rewards that do not depend on a_t, thereby reducing noise in the estimate and lowering variance.

In practice, for each sampled trajectory τ^i of length $T + 1$, one can use the following Monte Carlo estimate:

$$g_{\text{PG}} \;=\; \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta\big(a_t^i \mid s_t^i\big)\Big(\sum_{k=t}^{T} r_k^i\Big),$$

where N is the number of sampled trajectories. This approach exploits causality by assigning to each action a_t^i only the rewards from time t onward.

Generalized Advantage function. When applying the causality lemma once more, we can express the policy gradient in terms of the *advantage function* rather than the full or partial returns. Specifically, we have

$$\nabla_\theta \mathbb{E}\Big[\sum_{t=0}^{T} r(s_t, a_t)\Big] \;=\; \mathbb{E}\Big[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, \big(Q^\pi(s_t, a_t) - V^\pi(s_t)\big)\Big].$$

Here,

$$A^{\pi}(s_t, a_t) \;=\; Q^{\pi}(s_t, a_t) \;-\; V^{\pi}(s_t)$$

is called the *advantage function*, where:

- $Q^{\pi}(s_t, a_t)$ denotes the expected return after taking action a_t in state s_t and then following policy π thereafter;
- $V^{\pi}(s_t)$ denotes the expected return when starting from state s_t and following policy π.

Thus, $A^{\pi}(s_t, a_t)$ measures how much better (or worse) action a_t is compared to the baseline value $V^{\pi}(s_t)$ in that state.

Variance Reduction vs. Bias Using the advantage function can reduce variance compared to simply using the reward-to-go (i.e., $\sum_{k=t}^{T} r_k$). By focusing on the "incremental benefit" of an action, we avoid conflating the intrinsic value of a state with the contribution of the action itself. In practice, however, we typically do not know Q^{π} or V^{π} exactly and must approximate them with function approximators (e.g., neural networks). If we maintain an estimate $\hat{V}^{\theta}$, we can write:

$$\hat{A}(s_t, a_t) \;=\; Q^{\pi}(s_t, a_t) - \hat{V}^{\theta}(s_t),$$

or equivalently,

$$\hat{A}(s_t, a_t) \;=\; r(s_t, a_t) + \gamma \hat{V}^{\theta}(s_{t+1}) - \hat{V}^{\theta}(s_t),$$

depending on the method used to approximate Q^{π}. Although such an approach reduces variance, it may introduce bias if the value function estimator $\hat{V}^{\theta}$ is itself biased.

In later lectures or chapters, we will see how this idea of subtracting a learned baseline or advantage is closely related to actor-critic algorithms. The actor updates the policy parameters using gradient estimates that involve $\nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t)\, \hat{A}(s_t, a_t)$, while the critic learns to approximate V^{π} or Q^{π}. This combination often achieves lower variance and faster learning than pure policy gradient methods that rely solely on returns.

4 Question

With someone next to you, write pseudocode to implement policy learning

5 Answer

Algorithm 1 Value Iteration for Policy Learning

```
Require:
  • State space $\mathcal{S}$
  • Action space $\mathcal{A}$ (with $\mathcal{A}(s)$ for each state $s$)
  • Transition probabilities $P(s'|s,a)$
  • Reward function $r(s,a)$
  • Discount factor $\gamma \in (0,1)$
  • Convergence threshold $\theta > 0$
Ensure: Approximate optimal value function $V^*$ and optimal policy $\pi^*$
 1: Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
 2: repeat
 3:     $\Delta \leftarrow 0$
 4:     for each $s \in \mathcal{S}$ do
 5:         $v \leftarrow V(s)$
 6:         $V(s) \leftarrow \max_{a\in\mathcal{A}(s)} \Big[r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)\, V(s')\Big]$
 7:         $\Delta \leftarrow \max(\Delta,\, |v - V(s)|)$
 8: until $\Delta < \theta$
 9: for each $s \in \mathcal{S}$ do
10:     $\pi(s) \leftarrow \arg\max_{a\in\mathcal{A}(s)} \Big[r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a)\, V(s')\Big]$
11: return $\pi, V$
```