Qiang Liu's Research Diary – Contents

September 28

1 Intro: Policy Gradient

For simplicity, let us first consider a one-step decision-making process (also known as a contextual bandit).

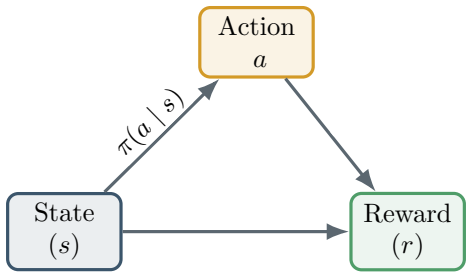
Problem Setup Let $s \in \mathcal{S}$ be the state vector of the environment, and let $a \in \mathcal{A}$ be the action taken by the agent. The reward function is denoted by $r(a, s)$. The agent's behavior is characterized by a policy, which is conveniently represented as a conditional probability distribution, $\pi(a \mid s)$, specifying the probability of selecting each action a given a state s. The objective is to find the optimal policy π^* that maximizes the expected reward:

$$R(\pi) = \mathbb{E}_{s\sim p_0}\mathbb{E}_{a\sim\pi(\cdot|s)}[r(a,s)] = \sum_{s,a} p_\pi(a,s)r(a,s),$$

where p_0 denotes the distribution of states, and we have $p_\pi(a, s)$ denotes the joint distribution of (a, s) when policy π is used:

$$p_\pi(a, s) = \pi(a \mid s)p_0(s).$$

Typically, p_0 is unknown to us, but observed through a collection of data $\{s^{(i)}\}$ drawn from p_0.



Policy Gradient In practice, the policy $\pi(a \mid s) = \pi_\theta(a \mid s)$ is parameterized by a function with parameter θ. Assume the state-action space $\mathcal{S} \times \mathcal{A}$ is finite or accountable, the gradient of the expected reward with respect to θ is

$$\begin{aligned}
\nabla_\theta R(\pi_\theta) &= \nabla_\theta \left(\sum_{s,a} p_0(s)\pi_\theta(a \mid s)r(a,s) \right) \\
&= \sum_{s,a} p_0(s)\nabla_\theta\pi_\theta(a \mid s)r(a,s) \qquad \text{//moving } \nabla_\theta \text{ into sum.} \\
&= \sum_{s,a} p_0(s)\pi_\theta(a \mid s)\frac{\nabla_\theta\pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}r(a,s) \\
&= \mathbb{E}_{p_{\pi_\theta}}\left[r(a,s)\nabla_\theta \log \pi_\theta(a \mid s)\right], \qquad //p_{\pi_\theta}(a,s) = \pi_\theta(a|s)p_0(s),
\end{aligned}$$

where the parts depending on θ are highlighted, and we use the log-derivative trick $\nabla_\theta \log \pi = \nabla_\theta\pi_\theta/\pi_\theta$. This allows us to express the derivative of an expectation with respect to a distribution as the expectation of the reward weighted by $\nabla_\theta \log \pi_\theta(a \mid s)$.

Reward Baseline For any policy π^θ, it is easy to prove the following identity:

$$\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a \mid s)] = 0, \qquad \forall s,\ \theta.$$

Hence, we can generalize the gradient formula above to

$$\nabla_\theta R(\pi_\theta) = \mathbb{E}_{p_{\pi_\theta}}\left[(r(a,s) - v(s))\nabla_\theta \log \pi_\theta(a \mid s)\right], \tag{1}$$

where $v(s)$ is *any* function that does not depend on the action a. The choice of v does not alter the expectation of the formula, but a proper choice of v can reduce the variance of the mean estimation given empirical observations.

One common choice of baseline is $v(s) = \mathbb{E}_{a\sim\pi_\theta(\cdot|s)}[r(a,s)]$, in which case the difference $r(a,s) - v(s)$ is known as the *advantage function*:

$$A(a,s) = r(a,s) - v(s).$$

A positive value $A(a,s) > 0$ (respectively, negative < 0) indicates that the action a performs above (respectively, below) the average.

On-Policy Estimation In practice, given observations $\{s^{(i)}\}$ drawn from p_0 and $a^{(i)} \sim \pi^\theta(\cdot|s^{(i)})$ drawn from the ongoing policy π^θ, the gradient can be estimated by

$$\nabla_\theta R(\pi_\theta) \approx \frac{1}{n}\sum_{i=1}^n A^{(i)} \nabla_\theta \log \pi_\theta(a^{(i)} \mid s^{(i)}), \qquad A^{(i)} \stackrel{\text{def}}{=} A(a^{(i)}, s^{(i)}). \tag{2}$$

The gradient ascent update, also known as REINFORCE,

$$\theta \leftarrow \theta + \epsilon \nabla_\theta R(\pi_\theta).$$

Intuitively, the gradient $\nabla_\theta \log \pi_\theta(a^{(i)} \mid s^{(i)})$ for data points $(a^{(i)}, s^{(i)})$ with positive advantages $A^{(i)} > 0$ is positively weighted, increasing the probability $\pi_\theta(a \mid s)$ during gradient ascent. Conversely, for data points with negative advantages, the gradient is negatively weighted, decreasing the probability.

Connection to Weighted MLE Assuming the data $\{a^{(i)}, s^{(i)}\}$ are fixed, the policy gradient $\nabla_\theta R(\pi_\theta)$ coincides with the gradient of the $A^{(i)}$-weighted log-likelihood function:

$$\ell(\pi_\theta) = \frac{1}{n}\sum_{i=1}^n A^{(i)} \log \pi_\theta(a^{(i)} \mid s^{(i)}).$$

This aligns with the intuition of maximizing $\log \pi(a \mid s)$ for data points with large (and positive) $A^{(i)}$, while minimizing the log-likelihood for negative $A^{(i)}$.

The key difference, however, is that the data $\{a^{(i)}, s^{(i)}\}$ are drawn from the policy π_θ and thus depend on the current parameter θ. As θ is updated during policy optimization, the data must either be regenerated or reweighted to reflect changes in π_θ. In contrast, maximum likelihood estimation assumes the data are fixed.

Hence, policy optimization can be viewed as an adaptively weighted maximum likelihood estimation (MLE), where the weights $(A^{(i)})$ are adjusted across iterations using newly generated data.

Off-Policy Estimation via Importance Sampling The gradient estimation in (2) is *on-policy*, because the actions a must be drawn from the current policy $\pi_\theta(\cdot|s)$. This does not yield efficient use of data as new actions must be rolled out once the policy is updated, and data from different policies cannot be used. In comparison, *off-policy* methods allow us to leverage actions from different policies.

One common approach to off-policy estimation is importance sampling. Assume that the data $\{a^{(i)}, s^{(i)}\}$ are drawn from a *behavior policy* $\pi^{\texttt{data}}$. Using importance sampling, we have

$$\begin{aligned}\nabla_\theta R(\pi_\theta) &= \mathbb{E}_{p_{\pi^{\texttt{data}}}}\left[\frac{\pi_\theta(a|s)}{\pi^{\texttt{data}}(a|s)} A(a,s)\nabla_\theta \log \pi_\theta(a \mid s)\right] \\ &\approx \frac{1}{z}\sum_{i=1}^n w^{(i)} A^{(i)} \nabla_\theta \log \pi_\theta(a^{(i)} \mid s^{(i)}),\end{aligned}$$

where $w^{(i)}$ is the importance weight:

$$w^{(i)} = \frac{\pi_\theta(a^{(i)}|s^{(i)})}{\pi^{\texttt{data}}(a^{(i)}|s^{(i)})},$$

and the normalization constant z is often taken as $z = \sum_i w^{(i)}$.

Remark 1. In RL, *on-policy methods* update the policy using data collected from the current policy itself. *Off-policy methods* learn from data collected under a different policy, allowing reuse of past experiences and improving sample efficiency, but at the cost of a distribution mismatch between the behavior policy π^{data} and the target policy π_θ. *Offline RL* is the case of off-policy learning where training relies entirely on a fixed dataset with no further interaction.

September 28

2 Intro: Proximal Policy Optimization

Proximal Point Method Going beyond gradient descent, proximal point methods offer a more general framework for optimization. Starting from an initialization θ^0, they perform iterative updates of the form:

$$\theta^{k+1} = \arg\max_{\theta} \left\{ R(\theta) - \beta D(\theta, \theta^k) \right\},$$

where, at iteration k, the next point θ^{k+1} is obtained by maximizing the objective while penalizing deviation from θ^k. Here, $D(\theta, \theta')$ is a discrepancy or distance measure between θ and θ'. The default choice is $D(\theta, \theta') = \frac{1}{2} \|\theta - \theta'\|^2$, though it is common to use KL divergence in policy optimization as we show below.

Each update from θ^k to θ^{k+1} requires solving an optimization problem, making the proximal point method a double-loop algorithm. Since this inner problem is often intractable, it is typically approximated using practical methods such as gradient descent or proximal gradient descent.

Different approximations lead to different algorithmic variants.

Proximal Gradient Descent Applying a first-order Taylor approximation to the objective function $R(\theta)$ gives:

$$\theta^{k+1} = \arg\max_{\theta} \left\{ \nabla R(\theta^k)^\top (\theta - \theta^k) - \beta D(\theta, \theta^k) \right\}.$$

This yields the *proximal gradient descent* method.

Preconditioned Gradient Descent Consider a quadratic proximal penalty:

$$D(\theta, \theta^k) = \frac{1}{2} (\theta - \theta^k)^\top J(\theta^k)(\theta - \theta^k),$$

where $J(\theta^k)$ is a positive definite preconditioning matrix chosen by the user. In this case, proximal gradient descent admits a closed-form solution:

$$\theta^{k+1} = \theta^k + J(\theta^k)^{-1} \nabla R(\theta^k).$$

This is known as *preconditioned gradient descent*.

Remark Assume we use gradient descent to solve the inner loops of proximal methods. The difference from vanilla gradient descent then lies in how the *reference point* is updated. To see this, let us consider the following generic update rule:

$$\theta^{k+1} = \arg\max_{\theta} \left\{ \nabla R(\theta^k)^\top (\theta - \theta^k) - \beta D(\theta, \theta^{\mathrm{ref},k}) \right\},$$

where $\theta^{\mathrm{ref},k}$ is the reference point used to anchor the update at iteration k. The choice of $\theta^{\mathrm{ref},k}$ governs the behavior of the algorithm:

- If $\theta^{\text{ref},k} = \theta^k$, the update reduces to *proximal gradient descent*, where the reference point is updated as fast as the iterate itself.
- If the reference point is updated only once every m iterations, i.e. $\theta^{\text{ref},k} = \theta^{m\lfloor k/m \rfloor}$, then each outer iteration corresponds to running m steps of a proximal gradient algorithm to approximately solve the proximal point update.
- If $\theta^{\text{ref},k}$ evolves more slowly, for example as an exponential moving average of past iterates, the algorithm takes more conservative steps, effectively stabilizing the optimization.

In short, the proximal point framework interpolates between fast-updating methods like gradient descent and slower, more implicit schemes, depending on how quickly the reference point is advanced.

Proximal Policy Optimization (PPO) Applying the proximal point method and using importance sampling for reward estimation, we have

$$\theta^{k+1} = \arg\max_\theta \left\{ \mathbb{E}_{\mathcal{D}} \left[\frac{\pi^\theta(a|s)}{\pi^{\texttt{data}}(a|s)} r(a,s) \right] - \beta D(\pi^\theta \;||\; \pi^{\theta^k}) \right\}.$$

We discuss two key design choices:

- *Penalty Function.* Typically, the discrepancy D is taken to be the KL divergence:

$$D(\pi^\theta \;||\; \pi^{\theta^k}) = \mathbb{E}_{s\sim p_0}[\mathrm{KL}(\pi^\theta(\cdot|s) \;||\; \pi^{\theta^k}(\cdot|s))].$$

However, various penalty functions have been used. We review these in the sequel.

- *Clipping Function.* Perhaps the most important trick in PPO is the clip function:

$$\theta^{k+1} = \arg\max_\theta \left\{ \mathbb{E}_{\mathcal{D}} \left[\mathcal{C}(w_\theta(a,s), r(a,s)) \right] - \beta D(\pi^\theta \;||\; \pi^{\theta^k}) \right\},$$

where $w_\theta(a,s) = \frac{\pi^\theta(a|s)}{\pi^{\texttt{data}}(a|s)}$, and $\mathcal{C}$ is defined as

$$\mathcal{C}(w,r) = \min(wr,\ \texttt{clip}(w, [1-\epsilon, 1+\epsilon])\, r).$$

In the following, we discuss the implications of the design choices for the penalty and clip functions, respectively.

References