Maths for Policy Gradient Methods

Li Ju
li-ju666@outlook.com

1. Policy Gradient Theorem

For a given environment, we define trajectories $\tau \in \mathrm{T}$, which is a sequence of state s and action a in the form of $(s_0, a_0, s_1, a_1, \ldots)$. The environment may assign rewards for each trajectory $R(\tau)$. Now assume that we have a policy model π_θ parameterised by θ. We are interested in the finding the parameter θ^* that maximizes the expected return $J(\theta)$, defined by

$$J(\theta) = \sum_{\tau \in \mathrm{T}} P(\tau \mid \theta) R(\tau).$$

Here $P(\tau \mid \theta)$ is a pretty complicated distribution, which involves in the distribution of initial states s_0 from the **environment**, action distributions from the **policy model** π_θ for state s_t, and state distributions of s_{t+1} from the **environment** if action a_t is taken. Note here τ is not a function of θ: it is a trajectory.

Nonetheless, we only care about its gradient w.r.t. the parameter θ, which tells us which direction to follow to maximize $J(\theta)$:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{\tau \in \mathrm{T}} P(\tau \mid \theta) R(\tau) \\
&= \sum_{\tau \in \mathrm{T}} \nabla_\theta P(\tau \mid \theta) R(\tau)
\end{aligned}$$

We know that $\nabla_x \ln f(x) = \frac{\nabla f(x)}{f(x)}$. By replacing $\nabla_\theta P(\tau \mid \theta) = \nabla_\theta \ln P(\tau \mid \theta) P(\tau \mid \theta)$, we get

$$\nabla_\theta J(\theta) = \sum_{\tau \in \mathrm{T}} \nabla_\theta \ln P(\tau \mid \theta) \cdot P(\tau \mid \theta) R(\tau).$$

The focus here is $\nabla_\theta P(\tau \mid \theta)$. Let's decompose $P(\tau \mid \theta)$ in this term:

$$\begin{aligned}
\nabla_\theta \ln P(\tau \mid \theta) &= \nabla_\theta \ln \left[P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t) \prod_{t=0} P(s_{t+1} \mid a_t, s_t) \right], \\
&= \nabla_\theta \ln \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t) + \mathbf{0}
\end{aligned}$$

since both $P(s_0)$ and $P(s_{t+1} \mid a_t, s_t)$ are not functions of parameter θ. (Here we assume Markovianity of both policy model and the environment, though the latter is not necessary for the derivation: The environment part has zero gradient anyway.)

Then back to $\nabla_\theta J(\theta)$, we have

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{\tau \in \mathrm{T}} P(\tau \mid \theta) \cdot R(\tau) \nabla_\theta \ln \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t) \\
&= \sum_{\tau \in \mathrm{T}} P(\tau \mid \theta) \cdot R(\tau) \nabla_\theta \sum_{t=0}^{T} \ln \pi_\theta(a_t \mid s_t). \\
&= \mathbb{E}_{\tau \sim P_\theta} \left[\left(\nabla_\theta \sum_{t=0}^{T} \ln \pi_\theta(a_t \mid s_t) \right) R(\tau) \right]
\end{aligned}
$$

Notes:

1. The beauty of the formula is that the complexity of the world $P(s_0)$ and $P(s_{t+1} \mid a_t, s_t)$ is not presented, making it powerful for very complex problems.
2. The expectation $\mathbb{E}_{\tau \sim P_\theta}$ can be naturally estimated by Monte-Carlo sampling once we run the experiment: Continuously taking actions a_t and interacting with the environment s_{t+1} is exactly sampling from the distribution of the trajectories defined by the policy model π_θ.

2. From theorem to implementation

Now we have the reciepe to train a policy model from the reward $R(\tau)$ directly. To train it, given a policy $\pi(\theta)$, we simply simulate the entire episode (trajectory), obtain the reward of the trajectory $R(\tau)$ and do backpropagation. Easy-peasy.

But this is extremely inefficient mainly due to two problems:

- The reward $R(\tau)$ is of very large variance w.r.t. trajectory τ. Just imagine the space of the trajectory, which is exponential to the space of states and actions w.r.t the number of steps.
- A trajectory might be extremely long, with which only one sinle gradient can be obtained.

2.1 REINFORCE

REINFORCE is the simplest algorithm that actually implements Policy Gradient in practice. It reduces the variance of the problem by replacing $R(\tau)$ to be "return-to-go" G_t at step t.

The policy gradient for REINFORCE is given as follows:

$$
\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P_\theta} \left[\nabla_\theta \sum_{t=0}^{T} (\ln \pi_\theta(a_t \mid s_t) G_t) \right].
$$

This is very similar to the original formula while the last time independent reward $R(\tau)$ is replaced by time-dependent $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, the weighted sum of future rewards. This is done by the assumption of Markovianity of the **environment**: Future actions do not affect the reward of the past actions.

Here is the formal derivation:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim P_\theta}\left[\left(\nabla_\theta \sum_{t=0}^{T} \ln \pi_\theta(a_t \mid s_t)\right) R(\tau)\right] \\
&= \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) \cdot \sum_{k=0}^{T} r_k(\tau)\right] \\
&= \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{t=0}^{T} \sum_{k=0}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) r_k(\tau)\right]
\end{aligned}
$$

For cases where $t > k$, we have

$$
\begin{aligned}
&\mathbb{E}_{\tau\sim P_\theta}\left[\nabla_\theta \ln \pi_\theta(a_t \mid s_t) r_k(\tau)\right] \\
=&\mathbb{E}_{s_{0:t}, a_{0:t-1}}\left[r_k \cdot \mathbb{E}_{a_t}\left[\nabla_\theta \ln \pi_\theta(a_t \mid s_t) \mid s_t\right]\right] \\
=&r_k \mathbb{E}_{a_t}\left[\frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} \mid s_t\right] \\
=&r_k \left[\sum_{a_t \in \mathcal{A}} \pi_\theta(a_t \mid s_t) \frac{\nabla \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)}\right] \\
=&r_k \nabla_\theta \sum_{a_t \in \mathcal{A}} \pi_\theta(a_t \mid s_t) \\
=&r_k \nabla_\theta(1) = 0
\end{aligned}
$$

Then we have

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{t=0}^{T} \sum_{k=0}^{t-1} \underbrace{\nabla_\theta \ln \pi_\theta(a_t \mid s_t)}_{\text{const. w.r.t } k} r_k(\tau)\right] \\
&= \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{t=0}^{T} \left[\nabla_\theta \ln \pi_\theta(a_t \mid s_t) \sum_{k=0}^{t-1} r_k(\tau)\right]\right] \\
&= \mathbb{E}_{\tau\sim P_\theta}\left[\nabla_\theta \sum_{t=0}^{T} \left[\ln \pi_\theta(a_t \mid s_t) \sum_{k=0}^{t-1} r_k(\tau)\right]\right]
\end{aligned}
$$

Compared to the given form which contains $G_t(s_t, a_t)$, we lack a discount factor γ_t. Our derivation is an unbiased estimator of the true gradient. However, the variance is still large. By adding the discount factor, we essentially value immediate rewards while reducing far-future rewards. This is a biased estimator but the variance is greatly reduced.

2.2 Before we move to "advanced" methods

We then show that, actually, adding any action independent functions in the return-to-go term G_t will not affect the policy gradient:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim P_\theta}\left[\nabla_\theta \sum_{t=0}^{T} \left[\ln \pi_\theta(a_t \mid s_t) \sum_{k=0}^{t-1} r_k(\tau)\right]\right] \\
&= \mathbb{E}_{\tau\sim P_\theta}\left[\nabla_\theta \sum_{t=0}^{T} \left[\ln \pi_\theta(a_t \mid s_t) \left(\sum_{k=0}^{t-1} r_k(\tau) + f(s_t)\right)\right]\right].
\end{aligned}
$$

This is equivalent to say

$$\mathbb{E}_{\tau\sim P_\theta}\left[\nabla_\theta \sum_{t=0}^{T} \ln \pi_\theta(a_t \mid s_t)\cdot f(s_t)\right] = 0.$$

This can be proved as follows:

$$\begin{aligned}
&\mathbb{E}_{\tau\sim P_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t)\cdot f(s_t)\right] \\
=&\sum_{t=0}^{T} \mathbb{E}_{\tau\sim P_\theta}\left[\nabla_\theta \ln \pi_\theta(a_t \mid s_t)\cdot f(s_t)\right] \\
=&\sum_{t=0}^{T} \mathbb{E}_{s_t}\left[\mathbb{E}_{a_t\sim\pi_\theta(a_t|s_t)}\left[\nabla_\theta \ln \pi_\theta(a_t \mid s_t)\cdot f(s_t)\right]\right] \\
=&\sum_{t=0}^{T} \mathbb{E}_{s_t}\left[\sum_{a_t\in\mathcal{A}} \pi_\theta(a_t \mid s_t)\nabla_\theta \ln \pi_\theta(a_t \mid s_t)\cdot f(s_t)\right] \\
=&\sum_{t=0}^{T} \mathbb{E}_{s_t}\left[\sum_{a_t\in\mathcal{A}} \cancel{\pi_\theta(a_t \mid s_t)} \frac{\nabla_\theta \pi_\theta(a_t \mid s_t)}{\cancel{\pi_\theta(a_t \mid s_t)}}\cdot f(s_t)\right] \\
=&\sum_{t=0}^{T} \mathbb{E}_{s_t}\left[\nabla_\theta \sum_{a_t\in\mathcal{A}} \pi_\theta(a_t \mid s_t)\cdot f(s_t)\right] \\
=&\sum_{t=0}^{T} \mathbb{E}_{s_t}\left[\nabla_\theta(1)\cdot f(s_t)\right] = 0.
\end{aligned}$$

2.3 REINFORCE with baseline

Now we have seen that REINFORCE can be equipped with any function independent of the action, we can use a value function acting as the baseline $V_\phi(s)$. This is useful since it greatly reduces the variance of the targets. To implement it, basically we need two neural networks: the policy $\pi_\theta(s) : s \to a$ and the value function $V_\phi(s) : s \to v$. The training algorithm is as follows:

1. Generate an episode: run the policy π_θ till the end T. This is essentially continuous inference using the policy model π_θ,
2. Calculate the returns $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$.
3. Update the policy:
    - Calculate the advantage $A_t = G_t - V_\phi(s_t)$.
    - Calculate $\ln \pi_\theta(a_t \mid s_t) A_t$: choose a_t from the output distribution and you get a scalar.
    - Backpropagate to get the gradient and update θ
4. Update the value function / critic: Perform gradient descent steps on the MSE loss using collected G_t as a target.

Here the value function we are approximating using V_ϕ is essentialy defined by $V^\pi(s) = \mathbb{E}[G_t \mid s_t]$, while the calculated return is one realization of this distribution of G_t.

2.4 Trust Region Policy Optimization

A problem in REINFORCE with baseline is that, the generation of the training data relies on the policy model itself. Any small noise in π_θ will further cause unreasonable episodes, which further mess π_θ up. For each update, the new model should not be too far away from the old one. This leads to the constrained optimization problem.
The objective TRPO is defined as

$$L_{\text{TRPO}}(\theta) = \mathbb{E}_{a.s\sim d^{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A(s,a)\right] \text{ s.t. } \text{KL}\left(\pi_{\theta_{\text{old}}} \| \pi_\theta\right) \leq \delta.$$

This does not look like $\nabla_\theta J(\theta)$ we have been using based on expectation on trajectory distribution dependent on the policy $\tau \sim P_\theta$.

2.4.1 Make sense of the objective of TRPO

1. From trajectory distribution P_θ to state-action distribution d^θ:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) A(a_t, s_t)\right] \\
&= \sum_{t=0}^{T} \mathbb{E}_{s_t, a_t \sim P_\theta} \nabla_\theta \ln \pi_\theta(a_t \mid s_t) A(a_t, s_t) \\
&= \mathbb{E}_{a,s\sim d^\theta} \nabla_\theta \ln \pi_\theta(a \mid s) A(a, s)
\end{aligned}$$

Essentially, the trajectory distribution is the distribution defined by

$$P_\theta(\tau) = P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t).$$

Distribution $s_t, a_t \sim P_\theta$ is the marginal distribution of action and state at step t.
The state-action distribution is defined as the frequent of state s and action a occur under policy π_θ, which is the distribution that ignores the time information:

$$d^\theta(s,a) = \frac{1}{T+1} \sum_{t=0}^{T} P(s_t = s, a_t = a \mid \pi_\theta).$$

Or even more, we can decompose this state-action distribution into action distribution and state distribution $s \sim d^\theta$ and $a \sim \pi_\theta(\cdot \mid s)$.

From this perspective, for policy π_θ, we collect a bunch of states, actions and corresponding rewards, update the model once, drop the old data and move on. This is called **on-policy** method.

However, this is very inefficient: Collecting data/episodes are costly but we only use it once. It would be nice to make use of data collected from an old policy model $\pi_{\theta_{\text{old}}}$.

REINFORCE + baseline still only looks at local information, while TRPO want to decide where to go based on its old model: To be precise, the performance gap between π_θ and $\pi_{\theta_{\text{old}}}$:

$$J(\theta) - J(\theta_{\text{old}}) = \mathbb{E}_{s,a\sim d^\theta}\left[A^{\theta_{\text{old}}}(s,a)\right].$$

This basically says, the performance gap is: under the new distribution, how much advantage would the new states and action gain from the old baseline ($V^{\theta_{\text{old}}}(s)$). Based on the old policy, we want to maximize the performance gap:

$$
\begin{aligned}
L_{\text{TRPO}}(\theta) = J(\theta) - J(\theta_{\text{old}}) &= \sum_s d^{\theta}(s) \sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) A^{\theta_{\text{old}}}(s,a) \\
&\approx \sum_s d^{\theta_{\text{old}}}(s) \sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) A^{\theta_{\text{old}}}(s,a) \\
&= \sum_s d^{\theta_{\text{old}}}(s) \sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) \frac{\pi_{\theta_{\text{old}}}(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A^{\theta_{\text{old}}}(s,a) \\
&= \sum_s d^{\theta_{\text{old}}}(s) \sum_{a \in \mathcal{A}} \pi_{\theta_{\text{old}}}(a \mid s) \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A^{\theta_{\text{old}}}(s,a) \\
&= \mathbb{E}_{a,s \sim d^{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A^{\theta_{\text{old}}}(s,a) \right]
\end{aligned}
$$

The approximation is done by replacing the state distribution under the new policy π_θ withe the state distribution under the old policy $\pi_{\theta_{\text{old}}}$. This approximation only holds when π_θ and $\pi_{\theta_{\text{old}}}$ is close. This also the root why the optimization is formulated under constraint $\text{KL}\left(\pi_{\theta_{\text{old}}} \| \pi_\theta\right) \leq \delta$.

2.4.2 Training algorithm

We initialize two neural networks, one for policy π_θ and one for value function V_ϕ.

1. We always start the training from simulating trajectories. For each trajectory, we save a sequence of $s_t, a_t, r_t, \pi_{\theta_{\text{old}}}(a_t \mid s_t)$ for $t \in [T]$. Now $\pi_{\theta_{\text{old}}} = \pi_\theta$.
2. Then we compute advantages of each tuple using the value function $A_t^{\theta_{\text{old}}} = G_t - V_\phi(s_t)$ and reorganize the date into a collection of tuples $(s, a, A, \ln \pi_{\theta_{\text{old}}}(a \mid s))$.
3. Train the policy π_θ:
    - Computing $\pi_\theta(a \mid s)$ using the policy model (note for the first step, $\pi_\theta = \pi_{\theta_{\text{old}}}$)
    - Compute the gradient $\nabla_\theta L(\theta)$ where $L(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\theta_{\text{old}}}(s,a)$.
    - Calculate the Fisher Information Matrix of the KL divergence term, search for the step using conjugate gradient method and perform linear search for the step size (basically constrained optimization here)
    - Repeat step 2 a few times until the new policy is too far from $\pi_{\theta_{\text{old}}}$
4. Update the critic model V_ϕ using the collected rewards r_t for $t \in [T]$.

2.5 Proximal Policy Optimization

Although TRPO is mathematically simple and elegant, the optimization itself is hard: One needs to compute the Fisher Information Matrix, a second order gradient, with additional conjugate gradient and linear search for step size for each optimization step. Is it possible to simplify the optimization?

Proximal Policy Optimization (PPO) aims to solve this problem with a mathematically ugly but empirically working approach by heuristically reformulate the optimization problem. Intuitively speaking, REINFORCE and REINFORCE+baseline methods are purely based on current θ, while TRPO tries to find the next θ on the basis of an old model $\pi_{\theta_{\text{old}}}$ with in a fixed range defined by the KL divergence between the old and the new policy. Instead of using *hard* KL constraints, PPO proposes to

uses clipped advantages:

$$L_{\text{PPO}}(\theta) = \mathbb{E}_{a,s\sim d^{\theta_{\text{old}}}} \min(\rho(\theta,\theta_{\text{old}})A^{\theta_{\text{old}}}, \text{clip}(\rho(\theta,\theta_{\text{old}}), 1-\epsilon, 1+\epsilon)\cdot A^{\theta_{\text{old}}})$$

where $\rho(\theta,\theta_{\text{old}}) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$, the probability mass/density ratio used on TRPO.

Compared to TRPO, the core component is the same: the probability mass/density ratio scaled by the advantage. However, two additional ingredients are added:

- clip operation: clip operation is used to approximate the trust region: It is forced to to within the predefined range limit $[1-\epsilon, 1+\epsilon]$ to avoid large update.
- min operation: This is the mechanism to prevent a case where when $A^{\theta_{\text{old}}}$ is negative (i.e. $r_t(\theta,\theta_{\text{old}})$ **should** be updated to be smaller) but after an update step $\rho(\theta,\theta_{\text{old}})$ gets greater somehow (due to the stochastic nature of optimization): If such **mistake** happens, ignore the range limit and revert the error immediately.

The training algorithm of PPO is essentially the same as TRPO only with a simpler optimization process.

2.6 Group Relative Policy Optimization

Till now, rethinking the training algorithm, we realize that in total we need three models to be loaded in the memory: the policy to be updated (π_θ), the old policy $\pi_{\theta_{\text{old}}}$ and the critic (the value function approximator) $V_\phi(\theta)$. The first two are clearly have the same numbers of the parameters while the critic generally needs to be similar to the policy models too in both architecture and number of parameters. This is inefficient for training. Instead instead of using the real advantage $A^\theta = G_t - V_\phi^\theta(s)$, GRPO propose to use the relative advantages with in a group of actions:

$$L_{\text{GRPO}}(\theta) = \mathbb{E}_{s\sim d^{\theta_{\text{old}}},\{a_i\}^G\sim\pi_{\theta_{\text{old}}}(\cdot|s)} \left[\frac{1}{G}\sum_{i=1}^{G}\left(\min(\rho_i A_i^r, \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon)\cdot A_i^r) - \beta\,\text{KL}(\pi_\theta \mid \pi_{\text{ref}})\right)\right]$$

where

$$\rho_i = \frac{\pi_\theta(a_i \mid s)}{\pi_{\theta_{\text{old}}}(a_i \mid s)},$$

$$A_i^r = \frac{r_i - \text{mean}(r_1,\ldots,r_G)}{\text{std}(r_1,\ldots,r_G)},$$

and

$$\text{KL}(\pi_\theta \mid \theta_{\text{ref}}) = \mathbb{E}_{a\sim\pi_\theta} \ln \frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{ref}}}(a \mid s)}$$

which can be approximated by one-sample Monte-Carol using $\ln \frac{\pi_\theta(a|s)}{\pi_{\text{ref}}(a|s)}$.

Here the key changes include:

- Relative advantages: This requires no training for the critic model.

- KL term: Without the critic model, an additional term involving the KL divergence between the model to be updated and a reference model is added.

In the context of LLM post-training, the objective is generally expressed as

$$L_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim P(q),\{o_i\}^G\sim\pi_{\theta_{\text{old}}}} \left[\frac{1}{G}\sum_{i=1}^{G} \left(\min(\rho_i A_i^r, \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon)\cdot A_i^r) - \beta\,\text{KL}(\pi_\theta \mid \pi_{\text{ref}}) \right) \right]$$

and

$$\rho_i = \frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)}$$

where o_i is the output tokens and q is the query tokens of the LLM. For numerical stability, $\rho_i(\theta, \theta_{\text{old}})$ is generally computed as

$$\rho_i(\theta, \theta_{\text{old}}) = \frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)} = \exp\left[\sum_{j=1}^{J} \ln \pi_\theta(a_{i,j} \mid s_{i,j}) - \ln_{\theta_{\text{old}}}(a_{i,j} \mid s_{i,j}) \right]$$

where $a_{i,j}$ is the j-th output token and $s_{i,j}$ is the query + first $j-1$ output tokens.

3. Other References

- DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning (https://arxiv.org/abs/2501.12948)
- Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? (https://arxiv.org/abs/2504.13837)
- ProRL: Prolonged Reinforcement Learning Expands Reasoning Boundaries in Large Language Models (https://arxiv.org/abs/2505.24864)
- SFT Memorizes, RL Generalizes: A Comparative Study of Foundation Model Post-training (https://arxiv.org/abs/2501.17161)