Reinforcement learning Intro: From Policy Gradient to Proximal Policy Optimization (PPO)

Zhe Li

Reinforcement learning plays a significant role in fine-tuning large language models. However, it is a mathematically complex technique and can be challenging to grasp. This introduction aims to convey the main concepts of reinforcement learning and the intuition behind the algorithm. To achieve this, we will include some essential mathematical derivations while omitting rigorous mathematical proofs.

Let's walk through one complete game step by step. At the start of the game (timestamp $t = 1$), environment randomly generates an initial state $\mathbf{s}_1$; Based on the state $\mathbf{s}_1$, the agent predicts the first action $\mathbf{a}_1$ that the agent should take; The environment, in response to the state $\mathbf{s}_1$ and the action $\mathbf{a}_1$, provides a reward r_1 to the agent; This procedure is repeated at each step: the environment generates a new state, the agent takes an action, and the environment responds with a reward util to the end of the game. We use the following notation to illustrate the entire procedure of the one game run.

t	1	2	$\cdots$	$T-1$	T
$\mathbf{s}$	$\mathbf{s}_1$	$\mathbf{s}_2$	$\cdots$	$\mathbf{s}_{T-1}$	$\mathbf{s}_T$
$\mathbf{a}$	$\mathbf{a}_1$	$\mathbf{a}_2$	$\cdots$	$\mathbf{a}_{T-1}$	$\mathbf{a}_T$
r	r_1	r_2	$\cdots$	r_{T-1}	r_T

The total reward after the above game is defined as

$$R = \Sigma_{t=1}^{T} r_t \tag{1}$$

The agent's goal/motivation is to maximize the total reward from the actions he takes or strategies he chooses in the game. The actions or the strategies the agent takes or chooses may give the best reward in one game run but may give poor reward in another game run due to the randomness in environment generating states and agent choosing the actions. Think in this way, even though the agent takes exactly same actions in two game, it could have different reward. Formally speaking, R is a random variable. Before we move forward. Let's clear out some notations. In order to differentiate different run of game, we introduce **trajectory** τ, which represents one run of game (**Episode**), mathematically

$$\tau : \{\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \mathbf{a}_2, \cdots, \mathbf{s}_T, \mathbf{a}_T\} \tag{2}$$

Back to the goal/motivation of the agent, it is ineffective to directly optimize R because R is random variable. Intuitively, we need to optimize the **expectation** of R for the agent parameterized by θ with respect to all runs of game(all different τ).

$$\bar{R}_\theta = \mathbf{E}_{\tau \sim p_\theta(\tau)}[R(\tau)] = \sum_\tau R(\tau) p_\theta(\tau) \tag{3}$$

Simply speaking, the above expectation is the weighted sum of reward $R(\tau)$ among all different episode τ and the weight is the how likely one episode τ can happen in reality, captured by $p_\theta(\tau)$. More intuitively, the above equation can be decomposed as the following

$$\bar{R}_\theta = R(\tau_1)p_\theta(\tau_1) + R(\tau_2)p_\theta(\tau_2) + \cdots + R(\tau_N)p_\theta(\tau_N) + \cdots \tag{4}$$

The reason why we emphasize the expectation of R is that it not only aids in understanding the derivation of policy gradient and their variants, but also guides practical algorithm implementation.

$$p_\theta(\tau) = p(\mathbf{s}_1)p_\theta(\mathbf{a}_1|\mathbf{s}_1)p(\mathbf{s}_2|\mathbf{a}_1, \mathbf{s}_1) \cdots p_\theta(\mathbf{a}_T|\mathbf{s}_T) \tag{5}$$

where $p(\mathbf{s}_1)$ is the probability that the environment generate the initial state $\mathbf{s}_1$ and $p_\theta(\mathbf{a}_t|\mathbf{s}_t)$ is the probability that the agent parameterized by θ takes the action $\mathbf{a}_t$ given the state $\mathbf{s}_t$. we can reorganize the above equation by

$$p_\theta(\tau) = p(\mathbf{s}_1) \prod_{t=1}^{T} p_\theta(\mathbf{a}_1|\mathbf{s}_1) \prod_{t=1}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_t) \tag{6}$$

The **trajectory** can be seen from two parts: environment ($p(\mathbf{s}_1)$ and $\prod_{t=1}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_t)$) and the agent ($\prod_{t=1}^{T} p_\theta(\mathbf{a}_1|\mathbf{s}_1)$).

1 Policy Gradeint

From calculus we know, in order to maximize $f(\theta)$, we use the gradient accent algorithm by updating θ

$$\theta_{k+1} = \theta_k + \eta \nabla f(\theta_k) \tag{7}$$

where η is the learning rate and $\nabla f(\theta_k)$ is the gradient of function $f(\theta)$ at θ_k. Let's use this frame to maximize the expectation reward $\bar{R}_\theta$. Just replace the gradient of $\bar{R}_\theta$ at θ_k in formular 7, then we have

$$\theta_{k+1} = \theta_k + \eta \nabla \bar{R}_\theta \tag{8}$$

The challenging part is to compute $\nabla \bar{R}_\theta$.

$$\nabla \bar{R}_\theta = \nabla \sum_\tau R(\tau) p_\theta(\tau) \tag{9}$$

$$= \sum_\tau R(\tau) \nabla p_\theta(\tau) \tag{10}$$

If we continue to plug in the definition of $p_\theta(\tau)$ in equation 6 to the above, ending up an massive formular. The trick that we get rid of product is $\nabla f(x) = f(x)\nabla \log f(x)$ and log of prod gives summation.

$$\begin{aligned}
\nabla \bar{R}_\theta &= \sum_\tau R(\tau) p_\theta(\tau) \nabla \log p_\theta(\tau) \\
&= \mathbf{E}_{\tau \sim p_\theta(\tau)} \Big[R(\tau) \nabla \log p_\theta(\tau) \Big] \\
&= \mathbf{E}_{\tau \sim p_\theta(\tau)} \Big[R(\tau) \Big(\sum_{t=1}^{T} \nabla \log p_\theta(\mathbf{a}_t | \mathbf{s}_t) \Big) \Big] \\
&= \sum_\tau R(\tau) p_\theta(\tau) \Big(\sum_{t=1}^{T} \nabla \log p_\theta(\mathbf{a}_t | \mathbf{s}_t) \Big)
\end{aligned} \tag{11}$$

Note that in the above equation $\mathbf{a}_t, \mathbf{s}_t$ refers to the action and state at timestamp t for specific trajectory τ. To make the above equation more clear, assume that we have N trajectories (we have to abuse some subscriptions to make the idea more clear)

$$\begin{aligned}
&\tau_1 : \{\mathbf{s}_1^1, \mathbf{a}_1^1, \mathbf{s}_2^1, \mathbf{a}_2^1, \cdots, \mathbf{s}_T^1, \mathbf{a}_T^1\} \\
&\tau_2 : \{\mathbf{s}_1^2, \mathbf{a}_1^2, \mathbf{s}_2^2, \mathbf{a}_2^2, \cdots, \mathbf{s}_T^2, \mathbf{a}_T^2\} \\
&\vdots \\
&\tau_N : \{\mathbf{s}_1^N, \mathbf{a}_1^N, \mathbf{s}_2^N, \mathbf{a}_2^N, \cdots, \mathbf{s}_T^N, \mathbf{a}_T^N\}
\end{aligned}$$

Thus, the above can be rewrite as

$$\begin{aligned}
\nabla \bar{R}_\theta &= \mathbf{E}_{\tau \sim p_\theta(\tau)} \Big[R(\tau) \Big(\sum_{t=1}^{T} \nabla \log p_\theta(\mathbf{a}_t | \mathbf{s}_t) \Big) \Big] \\
&\approx \frac{1}{N} \sum_{n=1}^{N} R(\tau_n) \sum_{t=1}^{T} \nabla \log p_\theta(\mathbf{a}_t^n | \mathbf{s}_t^n) \\
&= \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} R(\tau_n) \nabla \log p_\theta(\mathbf{a}_t^n | \mathbf{s}_t^n)
\end{aligned} \tag{12}$$

At this stage, we can formalize the process into an algorithmic framework 1. The key difference between policy gradient in reinforcement learning and traditional gradient descent in supervised learning setting is that data collection occurs within a loop. In each iteration, the algorithm collects trajectory data based on the current parameter θ, making

Algorithm 1 Basic Policy Gradient Algorithm Framework

1: Initialize agent's parameters θ_0 and the learning rate η
2: **for** $k = 0, 1, 2 \cdots$ **do**
3: Collect trajectories $\{\tau_i\}$ based on θ_0
4: Compute $\nabla \bar{R}_\theta$

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} R(\tau_n) \nabla \log p_\theta(\mathbf{a}_t^n | \mathbf{s}_t^n)$$

5: Update θ_{k+1} by

$$\theta_{k+1} = \theta_k + \eta \nabla \bar{R}_\theta$$

6: **end for**
7: Return the learned policy

the policy gradient time-consuming. Let's make the step 3 in the Alg 1 more clear. Collect trajectories $\{\tau_i\}$ means collecting $(\mathbf{s}_t^n, \mathbf{a}_t^n)$ and $R(\tau_n)$

$$\begin{aligned}
\text{In } \tau_1 \text{ trajectory} &: (\mathbf{s}_1^1, \mathbf{a}_1^1), (\mathbf{s}_2^1, \mathbf{a}_2^1), \cdots, (\mathbf{s}_T^1, \mathbf{a}_T^1), \text{accumulate } R(\tau_1) \\
\tau_2 &: (\mathbf{s}_1^2, \mathbf{a}_1^2), (\mathbf{s}_2^2, \mathbf{a}_2^2), \cdots, (\mathbf{s}_T^2, \mathbf{a}_T^2), \quad R(\tau_2) \\
&\vdots \\
\tau_N &: (\mathbf{s}_1^N, \mathbf{a}_1^N), (\mathbf{s}_2^N, \mathbf{a}_2^N), \cdots, (\mathbf{s}_T^N, \mathbf{a}_T^N), \quad R(\tau_N)
\end{aligned}$$

In Equation 12, for the entire trajectory $(\mathbf{s}_1^n, \mathbf{a}_1^n), (\mathbf{s}_2^n, \mathbf{a}_2^n), \cdots, (\mathbf{s}_T^n, \mathbf{a}_T^n)$, we use a single $R(\tau_n)$ as reward term corresponding to for all those state-action pairs. However, this approach has several issues. To address them, we replace $R(\tau_n)$ to a more generalized form $A^\theta(\mathbf{s}_t, \mathbf{a}_t)$, which allows us to discuss these problems and how to resolve them.

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} A^\theta(\mathbf{s}_t, \mathbf{a}_t) \nabla \log p_\theta(\mathbf{a}_t^n | \mathbf{s}_t^n) \tag{13}$$

In the current setting $A^\theta(\mathbf{s}_t, \mathbf{a}_t) = R(\tau_n)$, which not depends on $\mathbf{s}_t$ and $\mathbf{a}_t$. There are following improvements:

- $R(\tau)$ could always be positive, making every action seem plausible, which fails to differentiate between actions. To address this, we define $A^\theta(\mathbf{s}_t, \mathbf{a}_t) = R(\tau_n) - b$, where b is the baseline, typically $b = E[R(\tau)]$, this adjustment ensures that the term in front of log includes both positive and negative values, improving action differentiation.
- In a single trajectory, every action shares the same reward. Intuitively, each action only influences the rewards from subsequent actions and has no effect on rewards from

previous actions. To address this, we define $A^\theta(\mathbf{s}_t, \mathbf{a}_t) = \Sigma_{t'=t}^T r_{t'}^n - b$. Furthermore, each action only influences the rewards from subsequent actions, with its impact diminishing over time. To model this, we define $A^\theta(\mathbf{s}_t, \mathbf{a}_t) = \Sigma_{t'=t}^T \gamma^{(t-t')} r_{t'}^n - b$, where $\gamma < 1$, is hype-parameter.

At this point, we can define $A^\theta(\mathbf{s}_t, \mathbf{a}_t)$ as **advantage function**, which measures how favorable it is to take action $\mathbf{a}_t$ compared to other than other actions at the state $\mathbf{s}_t$.

2 Proximal Policy Optimization (PPO)

In the policy gradient algorithm, we must resample training data after each update of θ, which makes the process time-consuming. The question is can we reuse training data sampled from old θ to update θ. We resort to introduce the importance sampling technique to address that.

2.1 Importance sampling

To compute $E_{x\sim p}[f(x)]$, we typically sample many x_i from distribution p and approximate it as:

$$E_{x\sim p}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{14}$$

if directly sampling x_i from distribution p is not possible, but we can sample x_i from another distribution q, we can still compute $E_{x\sim p}[f(x)]$ by using importance sampling:

$$E_{x\sim p}[f(x)] = E_{x\sim q}[f(x)\frac{p(x)}{q(x)}] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)\frac{p(x_i)}{q(x_i)} \tag{15}$$

Utilizing importance sampling, we can have

$$\nabla \bar{R}_\theta = E_{\tau\sim p_{\theta'}(\tau)}\left[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)} R(\tau)\nabla \log p_\theta(\tau)\right] \tag{16}$$

Note that τ is sampled from $p_{\theta'}$ not p_θ and then use the data to train θ. Let's back to the equation with $A^\theta(\mathbf{s}_t, \mathbf{a}_t)$,

$$\begin{aligned}
\nabla \bar{R}_\theta &= E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta}\left[A^\theta(\mathbf{s}_t, \mathbf{a}_t)\nabla \log p_\theta(\mathbf{a}_t^n|\mathbf{s}_t^n)\right] \\
&= E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_{\theta'}}\left[\frac{p_\theta(\mathbf{s}_t, \mathbf{a}_t)}{p_{\theta'}(\mathbf{s}_t, \mathbf{a}_t)} A^{\theta'}(\mathbf{s}_t, \mathbf{a}_t)\nabla \log p_\theta(\mathbf{a}_t^n|\mathbf{s}_t^n)\right] \\
&= E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_{\theta'}}\left[\frac{p_\theta(\mathbf{a}_t|\mathbf{s}_t)}{p_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)} A^{\theta'}(\mathbf{s}_t, \mathbf{a}_t)\nabla \log p_\theta(\mathbf{a}_t^n|\mathbf{s}_t^n)\right]
\end{aligned}$$

In the above derivation, we cancel some trivial terms. Actually, using the above as gradient, means the objective we are maximizing is

$$J^{\theta'}(\theta) = E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_{\theta'}}\left[\frac{p_\theta(\mathbf{a}_t|\mathbf{s}_t)}{p_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}A^{\theta'}(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{17}$$

In importance sampling, $p(x)$ and $q(x)$ should not differ significantly. Thus, we need to add constraints on θ and θ'. In PPO algorithm, the objective becomes

$$J^{\theta'}_{PPO}(\theta) = J^{\theta'}(\theta) - \beta KL(\theta,\theta') \tag{18}$$

where KL is KL divergence and β is hype-parameter. There is some advanced approach, called adaptive KL divergence. Another variant of PPO objective

$$J^{\theta'}_{PPO}(\theta) = \Sigma_{(\mathbf{s}_t,\mathbf{a}_t)}\min\left(\frac{p_\theta(\mathbf{a}_t|\mathbf{s}_t)}{p_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}A^{\theta'}(\mathbf{s}_t,\mathbf{a}_t), \text{clip}(\frac{p_\theta(\mathbf{a}_t|\mathbf{s}_t)}{p_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}, 1-\epsilon, 1+\epsilon)A^{\theta'}(\mathbf{s}_t,\mathbf{a}_t)\right) \tag{19}$$

This is another version of ensuring new policy θ and old policy θ' not differ significantly.