The Complete Math Collection for LLM RL Fine-Tuning

Sherman Wong

May 5, 2025

1 Traditional Reinforcement Learning Problem

- **Input:** A simulator providing a state transition function:

  $$P(s'|s,a)$$

  and a reward function:

  $$R(s,a).$$

- **Output:** A policy:

  $$\pi(a|s)$$

  that maximizes the expected cumulative reward:

  $$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right],$$

  where γ is the discount factor.

1.1 Returns and Episodes

The **discounted return** G_t represents the total accumulated reward starting from time t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

which can be written recursively as:

$$G_t = R_{t+1} + \gamma G_{t+1}.$$

1.2 Policies and Value Functions

The **state-value function** for a policy π, denoted as $V_\pi(s)$, is the expected return when starting from state s and following policy π:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s].$$

Similarly, the **action-value function** $Q_\pi(s,a)$ is the expected return when taking action a in state s, and then following policy π:

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a].$$

1.3 Bellman Equation

From the definition of value functions, we derive the **Bellman equation**:

$$V_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right].$$

Expanding the expectation using the transition probability:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[R(s,a) + \gamma V_\pi(s') \right].$$

Similarly, the Bellman equation for action-value function $Q_\pi(s,a)$ is:

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right].$$

1.4 Optimal Policies and Value Functions

The optimal state-value function $V_*(s)$ is the maximum achievable value across all policies:

$$V_*(s) = \max_\pi V_\pi(s).$$

The optimal action-value function $Q_*(s,a)$ is the maximum achievable value for taking action a in state s:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a).$$

A policy is optimal if and only if it satisfies:

$$\pi_*(a|s) = \arg\max_a Q_*(s,a).$$

1.5 Bellman Optimality Equation

Since an optimal policy must maximize future rewards, we derive the **Bellman optimality equation**:

$$V_*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[R(s,a) + \gamma V_*(s') \right].$$

For the optimal action-value function:

$$Q_*(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a) \left[R(s,a) + \gamma \max_{a'} Q_*(s',a') \right].$$

These equations form the foundation for dynamic programming methods such as **Value Iteration** and **Policy Iteration**.

1.6 Monte Carlo Methods

Monte Carlo methods rely on complete episode data to estimate value functions under the current policy π:

1. Generate multiple trajectories by following policy π.
2. Compute the return for each state:
$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots$$
3. Update the value function using the average return:
$$V(s) = \frac{1}{n} \sum_{i=1}^{n} G_t^i,$$
where n is the number of times state s has been visited, and G_t^i is the return from the i-th visit to s.

1.7 Temporal Difference (TD) Learning

Unlike Monte Carlo methods, Temporal Difference (TD) methods do not require waiting for an entire episode to finish. Instead, TD methods update the value function online using partial episode data.

1.7.1 TD(0)

TD(0) is the simplest form of temporal difference learning. The Bellman equation serves as the theoretical foundation for TD learning, providing a recursive definition of the value function:

$$V(s_t) \leftarrow V(s_t) + \alpha \left(r_t + \gamma V(s_{t+1}) - V(s_t) \right),$$

where the **TD error** is defined as:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

The goal of TD(0) is to iteratively reduce the TD error δ_t, ensuring that $V(S_t)$ converges to the true value function $V_\pi(s_t)$ **under policy** π as $t \to \infty$:

$$\mathbb{E}[V(s_t)] \to V_\pi(s_t).$$

1.7.2 n-step TD

Building upon TD(0), n-step TD leverages multiple future rewards to accelerate the convergence of δ_t. The n-step return is defined as:

$$G_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n}).$$

The update rule for n-step TD is:

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^{(n)} - V(S_t) \right).$$

Special cases:

- When $n = 1$, n-step TD reduces to TD(0).
- When $n = T - t$ (i.e., the remaining steps in an episode), n-step TD becomes equivalent to the Monte Carlo method.

1.7.3 λ-return

To further improve the n-step TD method and achieve a finer balance between bias and variance, λ introduces a decay factor λ to compute a weighted average of different n-step returns:

$$G_t^{(\lambda)} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{(n-1)} G_t^{(n)}.$$

Equivalently, it can be rewritten as:

$$G_t^{(\lambda)} = (1 - \lambda) \left(G_t^{(1)} + \lambda G_t^{(2)} + \lambda^2 G_t^{(3)} + \dots \right).$$

Further derivation gives

$$G_t^(\lambda) = \sum_{n=0}^{\infty} (\gamma\lambda)^n [r_{t+n} + (1 - \lambda)\gamma V(s_{t+n+1})]$$

Special cases:

- When $\lambda = 0$, lambda-return reduces to TD(0).
- When $\lambda \to 1$, lambda-return approximates the Monte Carlo method.

1.8 Policy Gradient Method

Policy gradient methods directly optimize the policy $\pi_\theta(a|s)$ by adjusting the parameters θ to maximize the expected cumulative reward:

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

Using the **policy gradient theorem**, the gradient of the objective function is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) Q(s_t, a_t) \right].$$

Since estimating $Q(s, a)$ directly can be difficult, it is common to use the **advantage function**:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t).$$

Substituting this into the policy gradient equation:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) \right].$$

The advantage function reduces variance by centering the updates around the state value $V(s_t)$, improving training stability.
The policy is then updated using gradient ascent:

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

1.8.1 Monte Carlo Rollouts: High Variance

A full Monte Carlo estimate of the return is given by:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T.$$

REINFORCE is the classic Monte Carlo policy gradient method. It directly optimizes the policy using:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \right].$$

To reduce variance, a baseline $V(s_t)$ is often subtracted (with equal expectation)

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) \right],$$

Note here advantage function A(s,t) is estimated as:

$$A(s_t, a_t) \approx G_t - V(s_t).$$

The value function is updated using the full return:

$$V(s_t) \leftarrow V(s_t) + \alpha(G_t - V(s_t)).$$

Although Monte Carlo estimation is **unbiased**, it suffers from **high variance**, making training unstable.

1.8.2 TD-Based Advantage Estimation: Lower Variance

Instead of waiting for full episode rollouts like Monte-Carlo estimate, the **TD error** estimate is defined as:

$$A(s_t, a_t) \approx \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

The approximation on the left is b/c TD error is an **unbiased estimator of the advantage function**:

$$\mathbb{E}_\pi[\delta_t | S_t = s_t, A_t = a_t] = A(s_t, a_t),$$

it provides a lower-variance alternative to Monte Carlo estimation.
Using TD estimation, the policy gradient theorem gives:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \delta_t \right].$$

The **Actor-Critic (AC) method** uses **bootstrapping** (updating the value estimate of a state from estimated values of subsequent states) to update its critic function, specifically:

- Actor: The policy $\pi_\theta(a|s)$, updated via policy gradients.
- Critic: The value function $V_\phi(s)$, trained to minimize the TD error, which bootstraps from $V(s_{t+1})$

The policy gradient update in Actor-Critic is:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t) \right],$$

where the advantage function A is estimated using TD learning:

$$A(s_t, a_t) \approx \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t).$$

The value function is updated using bootstrapping with the TD error:

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t = V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)).$$

Actor-Critic methods reduce variance compared to REINFORCE but **introduce bias due to bootstrapping**. Example Actor-Critic methods include A2C, PPO, TRPO

1.8.3 Generalized Advantage Estimation

Generalized Advantage Estimation (GAE) is a method used in policy gradient algorithms for estimating the advantage function. GAE introduces λ weighting to balance bias and variance:

$$A^{GAE(\lambda,\gamma)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}.$$

GAE provides a flexible trade-off between variance and bias in advantage estimation, making it widely used in modern deep reinforcement learning algorithms. Example algorithm is PPO with GAE.

1.8.4 Importance Sampling

During RL iteration, we need to learn from the trajectory generated from "old policy", in which case we need to apply importance sampling to rectify the "return" of old policy, imagine we have probability distribution p(x) and q(x), and expectation of f(x) under p equals to:

$$\mathbb{E}_{x\sim p}[f(x)] \;=\; \mathbb{E}_{x\sim q}\left[\tfrac{p(x)}{q(x)}\, f(x)\right]$$

In RL training, let μ be the old policy distribution, and π being the current policy, the expected return of current policy is $R(\tau)$ is:

$$w(\tau) \;=\; \prod_{t=0}^{T} \frac{\pi(a_t \mid s_t)}{\mu(a_t \mid s_t)}$$

$$\mathbb{E}_{\tau\sim\mu}\big[w(\tau)\, R(\tau)\big] \;=\; \mathbb{E}_{\tau\sim\pi}\big[R(\tau)\big]$$

$$R(\tau) \;=\; \sum_{t=0}^{T} \gamma^t\, r(s_t, a_t)$$

Note in this off-policy definition, we assume the "old policy" is known, meaning we have access to its probability distribution $\mu(a|s)$.

1.8.5 PPO-clip

Carrying on from previous section, after importance sampling and GAE, the actor objective becomes:

$$\text{argmax}\mathbb{J}(\pi_\theta) = \mathbb{E}_{\pi_{old}}\Big[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} A^{GAE}(s_t,a_t) log\pi_\theta(a_t|s_t)\Big]$$

when the "old policy" is substantially different from current policy, the ratio π/μ will experience high fluctuation, which would also cause high fluctuation on the objective function above, PPO introduces a "clip" operation and GAE to suppress high variance update, the PPO objective is:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)A^{GAE}(s_t,a_t),\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A^{GAE}(s_t,a_t)\big) log\pi_\theta(a_t|s_t)\Big]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

1.8.6 PPO-KL

In order to prevent π_{old} and π_{new} from large deviation, PPO-KL introduces a KL term to penalize such deviations:

$$\text{argmax}\mathbb{J}(\pi_\theta) = \mathbb{E}_{\pi_{old}}\Big[\big(\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)} A^{GAE}(s_t,a_t) - \beta \cdot \text{KL}(\pi_\theta||\pi_{old})\big) log\pi_\theta(a_t|s_t)\Big]$$

1.8.7 Critric

Critics objective is to minimize the distance between "predicted" value function $V_\phi(s_t)$ and target value V_t^{target}

$$L^{\text{critic}}(\phi) = \mathbb{E}_t\Big[\big(V_\phi(s_t) - V_t^{\text{target}}\big)^2\Big] \quad , \quad V_t^{\text{target}} = V_t^{GAE} + V_{\phi old}(s_t) = (r_t + \gamma V_{\phi old}(s_{t+1})) + \gamma\lambda V_{t+1}^{GAE}$$

2 LLM RL

Let's try to map the RL defintions to LLM scenarios, in LLM scenario

$$\begin{aligned}
s_t &= \text{prompt} + \text{generated tokens up to } t \\
a_t &= \text{generate next token given } s_t \\
\pi(a_t|s_t) &= \text{probability distribution over next tokens given } s_t \\
r_t &= \text{immediate reward for } a_t \\
R &= \sum_{k=t}^{T} r_k \quad \text{(accumulated return)} \\
V(s_t) &= \mathbb{E}[R \mid s_t] \quad \text{(expected cumulative reward)}
\end{aligned}$$

2.1 PPO(Actor-Critic)

Now imagine π is the policy LLM model, q is one of the questions in the training dataset, o is the output, the PPO objective in LLM scenarios is

$$L^{\text{PPO}}(\theta) = \mathbb{E}_{q,o\sim\pi_{\theta_{\text{old}}}}\Big[\min\big(r_\theta(q,o)\,\hat{A}(q,o),\ \text{clip}\big(r_\theta(q,o),\,1-\epsilon,\,1+\epsilon\big)\,\hat{A}(q,o)\big)\Big]$$

$$r_\theta(q,o) = \frac{\pi_\theta(o \mid q, o_{<t})}{\pi_{\theta_{\text{old}}}(o \mid q, o_{<t})},$$

There are 2 drawbacks of LLM-PPO:

- Need to train a value model, which has similar cost to the policy model
- Token level loss, the reward is assigned at the LAST token only, no process reward during generation

2.2 GRPO

GRPO [SWZ+24] proposes a few key changes to the PPO algorithm

2.2.1 Remove Critic

GRPO skips the value function estimation step, instead, it uses a "sample group" to replace the value function

$$L^{\text{GRPO}}(\theta) = \mathbb{E}_{q,o_{i\,i=1}^{G}\sim\pi_{\theta_{\text{old}}}}\frac{1}{G}\Sigma_{i=1}^{G}\frac{1}{|o_i|}\Sigma_{t=1}^{|o_i|}\Big[\min\big(r_\theta(q,o_i)\,\hat{A}_{i,t},\ \text{clip}\big(r_\theta(q,o_i),\,1{-}\epsilon,\,1{+}\epsilon\big)\,\hat{A}_{i,t}\big)\Big]{-}\beta D_{KL}\Big[\pi_\theta||\pi_{ref}\Big]$$

Here $o_{i\,i=1}^{G} \sim \pi_{old}$ means sampling the outputs of the G candidates from the old policy.

2.2.2 Advantage $A_{i,t}$ calculations

1. Outcome supervision
   (a) For each of the output o_i, use reward model to score $r = \{r_1, r_2 ... r_G\}$
   (b) for each token $o_{i,t}$, the advantage is the normalized return:

$$\hat{A}_{i,t} = \frac{r_i - mean(r)}{std(r)}$$

2. Process Supervision At the end of each reasoning "step", a process reward is provided.
   (a) process supervision model will provide a reward according to:

$$\text{R} = \Big\{\Big\{r_1^{index(1)}, ...r_1^{index(K_1)}\Big\}, ... \Big\{r_G^{index(1)}, ...r_G^{index(K_G)}\Big\}\Big\}$$

   here $index(j$ means the $j-th$ **reasoning step** token index, K_i is the number of reasoning steps for the i-th response
   (b) for each **reasoning step**, the reward is normalized by

$$\tilde{r}_i^{index(j} = \frac{r_i^{index(j)} - mean(r)}{std(r)}$$

   (c) the advantage of each token is the sum of all normalized reward from subsequent steps

$$\hat{A}_{i,t} = \Sigma_{index(j)\geq t}\tilde{r}_i^{index(j}$$

2.2.3 Why GRPO

- Significantly reduced memory/compute cost by skipping value model training
- For the same input q, GRPO will produce different advantages across the group G, which will encourage model to differentiate the quality between responses within the same group. This is similar to using inference time scaling to improve response quality.

2.3 DAPO

2.3.1 Dynamic Sampling

There are observations that for some prompt p, all responses are correct and receive the same reward 1, which resulted in an advantage of 0 for all samples in group G. Hence DAPO [YZZ+25] proposed a **oversampling and filtering** control mechanism that leaves all prompts in the batch with effective gradients (accuracy is neither 0 or 1) and maintains a consistent number of prompts.

2.3.2 Token level loss

DAPO does not average token loss per example, instead it averages tokens across the entire group, which encourages long CoT with high-reward samples to learn more effectively, the overall DAPO update is

$$\mathrm{J}_{DAPO} = \mathbb{E}_{q,{o_i}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}}\left[\frac{1}{\Sigma_{i=1}^{G}|o_i|}\Sigma_{i=1}^{G}\Sigma_{t=1}^{|o_i|}\mathrm{J}_\theta\right]$$

2.3.3 Removing KL penalty

During training the long-CoT reasoning model, the distribution of the model can diverge significantly from the initial model; therefore, this restriction is not necessary. Therefore, DAPO excluded the KL term from the gradient equation, which basically removed the need for **reference policy model** which further reduced memory cost. The complete implementation of DAPO can be found in VeRL [SZY+24]

References

[SWZ+24] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[SZY+24] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

[YZZ+25] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.