Lecture 25, 26: Post-Training, Test-Time Compute, and RL for LLMs

Contents

1 Introduction to Post-Training

After the Pre-training phase (learning general distributions) and Supervised Fine-Tuning (SFT, primarily for instruction following), the focus shifts to enhancing the model's capabilities in problem-solving and reasoning. This phase is known as **Post-Training**.

The core objective is to move beyond simply mimicking the training data (SFT) to maximizing a reward signal, thereby enabling the model to solve problems better than the average human demonstrator. We explore two primary avenues for this:

1. **Test-Time Compute:** Improving performance by expending more computational resources during inference (generation).
2. **Reinforcement Learning (RL):** Training the model to optimize a reward function (e.g., RLVR, RLHF, DPO).

2 Test-Time Compute

Test-time compute refers to techniques used during inference to improve output quality without changing the model weights.

2.1 Basic Approaches

- **Chain-of-Thought (CoT):** Prompting the model to "think step-by-step."
- **Best-of-N (Rejection Sampling):**
  1. Generate N independent responses to a prompt.
  2. Select the best response based on:
     - **Majority Voting:** Useful for problems with a unique correct answer (e.g., math).
     - **Reward Model Scoring:** Use a separate trained model (verifier) to score each response and pick the $\arg\max$.

2.2 Reasoning with Sampling (MCMC)

Recent research suggests that for reasoning tasks, higher probability sequences under the base model often correlate with correctness, provided the model has some reasoning capability. The goal is to sample from a distribution where probability mass is shifted toward higher likelihood sequences.

We define a target distribution proportional to the model's probability raised to a power $\alpha > 1$:

$$\pi(x_{0:T}) \propto (p_\theta(x_{0:T}))^\alpha \tag{1}$$

where p_θ is the autoregressive probability of the sequence.

2.2.1 Why not Low-Temperature Sampling?

Standard low-temperature sampling adjusts the probability of the *next token* locally:

$$p_{\text{temp}}(x_t|x_{<t}) \propto p(x_t|x_{<t})^{1/\tau} \tag{2}$$

This is **not** equivalent to raising the probability of the *entire sequence* to a power α. The standard sampling method assumes a greedy, local optimization, whereas we wish to find sequences that are globally probable. A "pivotal token" might have low local probability but lead to a high-probability valid solution downstream.

2.2.2 Power Sampling via MCMC

Since we cannot normalize $(p_\theta(x))^\alpha$ directly, we use Markov Chain Monte Carlo (MCMC) methods, specifically a Metropolis-Hastings approach, to sample from the unnormalized target distribution.

Algorithm Concept:

1. Initialize a sequence x.
2. Propose a mutation x' (e.g., resample a block of tokens starting from index m using the base model p_θ).
3. Accept or reject x' based on the Metropolis-Hastings acceptance ratio $A(x', x)$:

$$A(x', x) = \min\left(1, \frac{\pi(x')}{\pi(x)} \frac{q(x|x')}{q(x'|x)}\right) \tag{3}$$

where q is the proposal distribution. If the proposal is simply the autoregressive generation from the base model, the terms largely cancel, leaving a ratio dependent on the sequence probabilities raised to α.

Empirical results show that this method (sampling from p^α via MCMC) can significantly boost performance on math reasoning benchmarks (e.g., MATH500) without any parameter updates.

3 Reinforcement Learning with Verifiable Rewards (RLVR)

RLVR applies when we have a ground-truth mechanism (an "autograder") to evaluate outputs. Examples include:

- **Math problems:** Checking if the final boxed answer matches the known solution.
- **Code generation:** Checking if the code passes unit tests.

This provides a binary (or scalar) reward $r(y) \in \{0, 1\}$ for a response y.

3.1 Scaling Laws for RL

Performance in RL post-training often follows an S-curve (sigmoid-like) with respect to the amount of compute (GPU hours) spent on generation and training.

- **Initial Phase:** Little gain (exploration).
- **Growth Phase:** Rapid improvement.
- **Saturation Phase:** Asymptotic convergence to a maximum accuracy A.

Understanding these curves allows practitioners to extrapolate whether investing compute in RL is worthwhile for a specific model architecture.

3.2 The Training Objective (ScaleRL / GRPO)

The objective is to maximize the expected reward of the generated sequences. However, since the reward is non-differentiable, we use policy gradient estimators.

3.2.1 Notation

- π_θ: The policy (LLM) being trained.
- π_{old}: The policy used to generate the data (often a lagging version of π_θ).
- π_{ref}: The fixed SFT reference model (used for regularization).
- G: Number of generations (rollouts) per prompt.

3.2.2 Advantage Estimation

For a prompt x, we generate G responses $\{y_1, ..., y_G\}$. We calculate the reward r_i for each. To reduce variance, we compute the **Normalized Advantage** $\hat{A}_i$:

$$\hat{A}_i = \frac{r_i - \mu(\{r\})}{\sigma(\{r\}) + \epsilon} \tag{4}$$

where μ and σ are the mean and standard deviation of rewards within the batch (group). This centers the signal: "good" responses get positive advantages, "bad" responses get negative advantages.

3.2.3 Loss Function

We minimize a loss based on the PPO (Proximal Policy Optimization) or GRPO (Group Relative Policy Optimization) structure. The loss typically looks like:

$$\mathcal{L}(\theta) = -\mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_{\text{old}}}\left[\min\left(\rho_t \hat{A}_t, \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\hat{A}_t\right) - \beta D_{KL}(\pi_\theta || \pi_{\text{ref}})\right] \tag{5}$$

Key components:

1. **Importance Sampling Ratio (ρ_t):** Since we generate with π_{old} but optimize π_θ, we must correct for the distribution shift:
$$\rho_t = \frac{\pi_\theta(y_t|x)}{\text{sg}[\pi_{\text{old}}(y_t|x)]} \tag{6}$$
The `sg` (stop gradient) operator ensures gradients do not flow into the denominator.
2. **Clipping:** The min and clip functions prevent the policy from changing too drastically in one step, stabilizing training.
3. **Token-level view:** While rewards are sequence-level, the likelihood is computed as the product of token probabilities. The advantage $\hat{A}$ is "assigned" to every token in the sequence.

3.3 KL Regularization

In RL for LLMs, we almost always include a KL-divergence penalty against a reference model (usually the SFT model).

$$\text{Objective} = \mathbb{E}_{y\sim\pi}[r(x,y)] - \beta D_{KL}(\pi_\theta(\cdot|x)||\pi_{\text{ref}}(\cdot|x)) \tag{7}$$

This prevents "Reward Hacking" (exploiting the reward function in degenerate ways) and "Catastrophic Forgetting" (losing the ability to generate coherent English).

The closed-form optimal policy for this objective is:

$$\pi^*(y|x) \propto \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right) \tag{8}$$

4 Reinforcement Learning from Human Feedback (RLHF)

When verifiable rewards are unavailable (e.g., "Write a poem," "Summarize this text"), we rely on human preference.

4.1 Challenges with Humans

- Humans are expensive and slow.
- Humans are poor at assigning absolute scalar scores (e.g., "Is this a 7.2 or a 7.5?").
- Humans are good at **binary comparisons**: "Is response A better than response B?" ($y_w \succ y_l$).

4.2 The Standard RLHF Pipeline

1. **SFT:** Train a base instruction-following model.
2. **Reward Modeling (RM):**
   - Collect pairs (y_w, y_l) where humans prefer y_w.
   - Train a Reward Model $r_\phi(x, y)$ to maximize the likelihood of the preference data using a Bradley-Terry model:

$$p(y_w \succ y_l|x) = \sigma(r_\phi(x, y_w) - r_\phi(x, y_l)) \tag{9}$$

3. **RL (PPO):** Use the learned r_ϕ as the reward function to train the policy π_θ using the PPO methods described in the RLVR section.

5 Direct Preference Optimization (DPO)

DPO (2023) simplifies the RLHF pipeline by eliminating the separate Reward Model and the PPO loop. It relies on the insight that the language model itself can implicitly represent the reward.

5.1 Derivation

Recall the analytical solution for the optimal policy under KL-regularized RL:

$$\pi^*(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x, y)}{\beta}\right) \tag{10}$$

where $Z(x)$ is the partition function.

We can rearrange this equation to express the reward function $r(x, y)$ in terms of the optimal policy and the reference policy:

$$\exp\left(\frac{r(x, y)}{\beta}\right) = \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} Z(x) \tag{11}$$

$$r(x, y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x) \tag{12}$$

Now, substitute this expression for reward into the Bradley-Terry preference model used for training reward models:

$$p(y_w \succ y_l|x) = \sigma\left(r(x, y_w) - r(x, y_l)\right) \tag{13}$$

Substituting the derived expression for r:

$$r(x, y_w) - r(x, y_l) = \beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} + \underbrace{(\beta \log Z(x) - \beta \log Z(x))}_{0} \tag{14}$$

Crucial Insight: The partition function $Z(x)$ cancels out because it depends only on the prompt x, not the response y.

5.2 The DPO Loss

We can now optimize the policy π_θ directly by maximizing the likelihood of the preference data. The DPO loss is:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right] \tag{15}$$

5.3 Gradient Mechanics

Though the reference policy π_{ref} appears in the loss, gradients do not flow through it (it is frozen). The loss effectively:

- Increases the likelihood of the winner y_w.
- Decreases the likelihood of the loser y_l.
- **Implicitly Dynamic Weights:** The gradient magnitude is scaled by the sigmoid derivative. If the model already confidently predicts the winner (high probability ratio difference), the gradient vanishes (saturation). If the model is wrong, the gradient is strong.

5.4 Iterative DPO

A variation (Iterative DPO) involves updating the reference model π_{ref} periodically. Instead of anchoring to the initial SFT model forever, we reset $\pi_{\text{ref}} \leftarrow \pi_\theta$ every few hundred steps. This allows the model to drift further while maintaining local regularization constraints.