RL and Policy Optimization

James Pustejovsky

CS115B
Brandeis University

April 22, 2025

Reinforcement Learning

Definition: Reinforcement Learning (RL) is a framework where an *agent* interacts with an *environment* in **discrete** or **continuous** time steps.

Key Elements:

- **States** ($s \in \mathcal{S}$): The agent's observation of the environment.
- **Actions** ($a \in \mathcal{A}$): Decisions the agent makes at each step.
- **Reward** ($r \in \mathbb{R}$): A scalar feedback signal for each action-state pair.
- **Policy** ($\pi_\theta(a \mid s)$): Mapping from states to actions (can be stochastic).

Reinforcement Learning

Objective: Maximize cumulative rewards.
Typically formalized as a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$:

- $P(s' \mid s, a)$: Transition probabilities.
- $R(s, a)$: Immediate reward.
- $\gamma \in [0, 1]$: Discount factor for future rewards.

Return:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}.$$

Goal:

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)].$$

Trajectories and Interaction

Trajectory (or episode) τ:

$$(s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T).$$

Agent-Environment Loop:

1. Agent observes s_t.
2. Agent samples action a_t from $\pi_\theta(a_t \mid s_t)$.
3. Environment transitions to s_{t+1}, yields reward r_t.

In Practice:

- Episodic tasks have a terminal state (s_T).
- Continuous tasks can go on indefinitely (with discount $\gamma < 1$).

Policy Gradient Approach

Policy Gradient Theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, Q_{\pi_\theta}(s_t, a_t) \Big].$$

Why Policy Gradients?

- Directly optimize the policy π_θ.
- Good for **continuous action spaces** or when discrete action spaces are large.
- Flexible: We can incorporate function approximators (neural networks).

Components:

- $\log \pi_\theta(a_t \mid s_t)$: Increases probability of chosen actions.
- $Q_{\pi_\theta}(s_t, a_t)$: Guides which actions are valuable.

The Advantage Function

Value Function: $V_{\pi_\theta}(s) = \mathbb{E}[G_t \mid s]$.
Action-Value (Q) Function:

$$Q_{\pi_\theta}(s, a) = \mathbb{E}[G_t \mid s, a].$$

Advantage Function:

$$A_{\pi_\theta}(s, a) = Q_{\pi_\theta}(s, a) \; - \; V_{\pi_\theta}(s).$$

- Tells us how much better (or worse) a specific action a is compared to the average action in state s.
- Lower variance policy gradients often estimate A_{π_θ} to reduce noise.

Generalized Advantage Estimation (GAE) – Derivation

Temporal Difference Error:

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

GAE Formula:

$$\hat{A}_t = \sum_{k=0}^{\infty} (\gamma\lambda)^k \, \delta_{t+k}$$

- λ trades off bias and variance.
- When $\lambda = 1$, high variance; when $\lambda = 0$, high bias.

Fine-Tuning LLMs

Motivation:

- Large Language Models (LLMs) are pretrained on massive corpora in a *general-purpose* fashion.
- Fine-tuning tailors them to:
  - Specific **tasks** (e.g., summarization, QA).
  - Desired **style** or **persona**.
  - **Safety** and **alignment** constraints.

Key Questions:

- Should we fine-tune *all* parameters of the LLM?
- Are there *parameter-efficient* methods to reduce cost and memory usage?
- How do we incorporate **alignment** objectives (e.g., from human feedback)?

Common Fine-Tuning Approaches

1. **Full Model Fine-Tuning**:
   - **All parameters** of the LLM are updated.
   - Pros: Maximum flexibility, can adapt model deeply.
   - Cons: Very large memory footprint and computational cost; can overfit small datasets.
2. **Instruction Tuning / SFT**:
   - **Supervised Fine-Tuning** on curated instructions or examples.
   - Often used to make LLMs follow prompts more effectively (**e.g.**, InstructGPT).
   - Pros: Relatively straightforward (supervised approach).
   - Cons: Might still be expensive if done on full parameters; can be data-hungry.

Common Fine-Tuning Approaches

3. **Parameter-Efficient Methods**:
   - **LoRA** (Low-Rank Adapters), **Prefix Tuning**, **P-Tuning**, **Adapters**.
   - Insert small "adapter" modules or "trainable prompts" into the LLM, *freezing* most main weights.
   - Pros: Greatly reduced parameter updates, often robust performance on specialized tasks.
   - Cons: Less capacity to drastically alter the model if needed; some complexities in combining multiple tasks.

Integration with RLHF:

- RLHF can be done *on top* of these methods (e.g., use LoRA + PPO).
- Reduces GPU memory usage, enabling practical large-scale alignment.

Defining Alignment in AI

General Concept of Alignment:

- **Definition:** Alignment refers to how well an AI system's behaviors, outputs, or objectives match the **intended goals** or **values** specified by human designers or users.
- **Why Important?**
  - An unaligned system can produce *undesired*, *harmful*, or *unsafe* behavior.
  - In LLMs, misalignment risks spreading misinformation, hate speech, or otherwise violating user trust.

In Practice:

- We often rely on **human feedback**, policy guidelines, or domain knowledge to shape the system's behavior.
- Ensuring thorough alignment is an *ongoing challenge* across AI fields.

Alignment Challenges for LLMs

Large Language Models (LLMs):

- Trained on massive text corpora with **no direct supervision** of "right" vs. "wrong" uses.
- Might produce biased, offensive, or factually incorrect outputs if not aligned.

Challenges:

- **Value Misalignment**: The model could generate content that conflicts with user or societal values (e.g., hateful or disallowed content).
- **Hallucination / Fabrication**: LLMs can confidently output false information.
- **Context Sensitivity**: Hard to define alignment in an *open-ended* domain (e.g., creative writing vs. factual QA).
- **Scalability**: Aligning giant models that generate billions of tokens or have billions of parameters is non-trivial.

Alignment Approaches

High-Level Methods:

- **Supervised Fine-Tuning:** Train on curated datasets of "good" vs. "bad" outputs.
- **Reinforcement Learning from Human Feedback (RLHF)**: Use a *reward model* or preference-based signals to shape the model's policy.
- **Direct Preference Optimization (DPO)**: Treat preference data as supervised signals for pairwise ranking.
- **Constrained Decoding / Rules**: Impose constraints on the generation process (filter out disallowed content).

Aligning LLMs with Human Preferences

Key Goal: Ensure Large Language Models (LLMs) produce:

- **Helpful** responses (accurate, relevant, complete).
- **Safe** content (no harmful, biased, or offensive outputs).
- **Aligned** with ethical, legal, and policy guidelines.

Human Feedback Loop:

- Humans (or expert annotators) rank or label model outputs.
- Model updates its parameters to better match these preferences.

Two Notable Methods:

- **PPO (Proximal Policy Optimization):** A standard RLHF approach.
- **DPO (Direct Preference Optimization):** A newer, more direct preference-based method.

The RLHF Pipeline (High-Level)

1. **Pretrained Model:** Start with an LLM M_{pre} trained on large-scale corpora.
2. **Gather Human Feedback:**
   - Show model outputs for prompts to human annotators.
   - Collect preference labels or rankings (e.g., which output is best).
3. **Reward Model (RM):**
   - Convert human preference data into a *reward function* or *scoring model*.
   - The RM approximates how "good" a response is.
4. **Policy Optimization:**
   - Fine-tune M_{pre} (now called π_θ) using RL or direct preference optimization.
   - Output: A final policy π_{θ^*} that yields higher reward (i.e., better alignment with preferences).

Policy Gradient Concept (Mini-Refresher)

Policy: $\pi_\theta(a \mid s)$ is a probability distribution over actions a given state s.
Goal: Maximize expected return (or reward),

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} r(s_t, a_t) \right].$$

where τ is a trajectory $(s_0, a_0, \ldots, s_T, a_T)$ generated by policy π_θ.
Policy Gradient Theorem:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \, Q_{\pi_\theta}(s_t, a_t) \right].$$

In language modeling:

- **States** s_t can be the text context so far.
- **Actions** a_t are tokens or text chunks to generate.

TRPO: Overview

Trust Region Policy Optimization (TRPO) aims to:

- Prevent overly large, destabilizing updates.
- Constrain the change between $\pi_{\theta_{\text{new}}}$ and $\pi_{\theta_{\text{old}}}$ using KL divergence.

Application: Stabilizes fine-tuning of LLMs.

TRPO Basics

Objective:

$$\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right]$$

Subject to:

$$\hat{\mathbb{E}}_t \left[\text{KL}\left(\pi_{\theta_{\text{old}}}(\cdot|s_t) \,\|\, \pi_\theta(\cdot|s_t) \right) \right] \leq \delta$$

δ controls how "close" the new policy must remain.

TRPO: Detailed Derivation

Lagrangian Formulation:

$$L(\theta, \lambda) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t \right] - \lambda \left(\hat{\mathbb{E}}_t[\text{KL}(\pi_{\theta_{\text{old}}} || \pi_\theta)] - \delta \right)$$

- Solved using conjugate gradient methods.
- λ is the Lagrange multiplier enforcing the KL constraint.

TRPO: Example Discussion

Example: Fine-tuning a summarization model.

- **State:** The current summary and context.
- **Action:** Next word/token generated.
- **Reward:** Derived from a reward model assessing summary quality.
- **KL Constraint:** Prevents the model from "forgetting" its fluency learned during pretraining.

TRPO: Pros and Cons

Pros:

- Theoretical guarantees on improvement.
- Effective in safety-critical tasks.

Cons:

- Requires second-order optimization (computationally intensive).
- More complex to implement.

Policy Gradients?

- In Reinforcement Learning (RL), the agent aims to maximize cumulative reward.
- Policy gradients are a direct way to optimize the parameters of a policy $\pi_\theta(a \mid s)$.
- Avoid the complexities of value function approximation (although hybrids exist).
- REINFORCE is the most straightforward policy gradient method.

REINFORCE Algorithm (High-Level)

- Also known as the Monte Carlo Policy Gradient.
- Key idea: update policy parameters θ in the direction of better returns.
- Simple update rule:

$$\Delta\theta \propto \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t$$

- G_t is the return following time t.

Return and Trajectories

- For an episode $\tau = (s_0, a_0, r_1, s_1, \ldots, s_{T-1}, a_{T-1}, r_T)$,

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$$

- $\gamma \in [0, 1]$ is the discount factor.
- REINFORCE uses full-episode returns to update parameters.

Derivation of the Gradient

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[G(\tau)]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[G(\tau) \nabla_\theta \log \pi_\theta(\tau)\right]$$

- $\pi_\theta(\tau) = \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t)$.
- By taking $\nabla_\theta \log \pi_\theta(\tau)$, we get the sum of logs over actions.

Single-Step Example

- For a single-step environment (bandit), $G(\tau) = r(a)$.
- Update:

$$\Delta\theta \propto \mathbb{E}\big[r(a)\nabla_\theta \log \pi_\theta(a)\big].$$

- This can be high variance if $r(a)$ is noisy.

Baselines and Variance Reduction

- REINFORCE often uses a baseline $b(s)$ to reduce variance:

$$\Delta\theta \propto \sum_t (G_t - b(s_t)) \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

- If $b(s)$ is a good approximation of the value function $V^\pi(s)$, variance is greatly reduced.
- Does not introduce bias if $b(s)$ is state-dependent but not action-dependent.

Algorithmic Steps

1. Initialize policy parameters θ.
2. Repeat (for each episode):
   - Generate an episode using π_θ.
   - For each time step t:
     - Compute return G_t.
     - Update:

$$\theta \leftarrow \theta + \alpha \left(G_t - b(s_t)\right) \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

The Objective $J(\theta)$

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\big[G(\tau)\big]$$

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

- Typically in discrete action spaces, $\pi_\theta(a \mid s)$ is parameterized via a softmax of logits.
- In continuous action spaces, π_θ could be Gaussian.

Gradient Trick

$$\nabla_\theta J(\theta) = \sum_\tau \left(\nabla_\theta \prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t) \right) P(\tau) G(\tau).$$

$$= \sum_\tau \left(\prod_{t=0}^{T-1} \pi_\theta(a_t \mid s_t) \right) \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right) G(\tau).$$

$$= \mathbb{E}_{\tau \sim \pi_\theta} \left[G(\tau) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \right].$$

Incremental Updates

- In practice, we sample a batch of episodes and approximate the expectation.
- Single-trajectory gradient estimate:

$$\Delta\theta = \alpha\, G(\tau) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

- This is unbiased but can have high variance.

Pros and Cons of REINFORCE

Pros

- Conceptually simple and easy to implement.
- Does not require value function approximation.
- Can be used in any differentiable policy setting (discrete or continuous).

Cons

- High variance updates.
- Slower convergence compared to methods that use critics (e.g., Actor-Critic).
- Full returns needed (episode-based), which can be expensive for long horizons.

RL for Language Models

- In large language models (LLMs), the "environment" can be seen as the text generation process.
- Actions are tokens, states are partial sequences.
- Rewards come from preference models, user feedback, or automated metrics.
- REINFORCE can directly optimize these rewards, but naive application may be unstable.

Typical RLHF (PPO)

- Proximal Policy Optimization (PPO) is more common for RLHF.
- Clipped updates reduce catastrophic changes to the language model.
- Still a policy gradient approach, but with more stability than pure REINFORCE.

Direct Preference Optimization (DPO)

- A new approach that "removes the RL" from RLHF by deriving a closed-form solution for the updated policy.
- Key idea: Reweight the pretrained policy distribution by $\exp(\mathrm{RM}_\phi(x, y)/\beta)$.
- Conceptually linked to policy gradient ideas—especially the exponential family perspective.

From REINFORCE to DPO

- REINFORCE perspective:

$$\Delta\theta \propto R\nabla_\theta \log \pi_\theta.$$

- If reward is $\mathrm{RM}_\phi(x, y)$, and we have a KL penalty to the pretrained model $\log p^{\mathrm{PT}}$, you can solve for π^* in closed form.
- DPO (and related methods) exploit this closed-form "Boltzmann" distribution.

Connection to Exponential Families

- REINFORCE with reward $R(y)$ and KL to a base distribution leads to:

$$\pi^*(y) \propto p^{\mathrm{PT}}(y) \exp\left(\tfrac{1}{\beta} R(y)\right).$$

- This is the same functional form as a Boltzmann distribution in statistical physics.
- DPO leverages this fact to skip iterative policy gradient loops.

Practical Implications for LLMs

- Efficiency: DPO can be more direct, but still may require sampling to estimate normalization constants.
- Stability: REINFORCE alone can be unstable for large models, so often a mix of ideas (KL control, baselines, etc.) is used.
- Interpretability: The Boltzmann form gives a clear interpretation of how reward and prior combine.

When to Use REINFORCE vs. DPO

- REINFORCE:
  - Good for simple or small-scale tasks.
  - Straightforward to implement, but may need variance reduction.
- DPO:
  - More direct approach if you have a stable reward model.
  - Avoids iterative RL, but normalization can be tricky in large vocabularies.

PPO Overview

Proximal Policy Optimization (PPO) [1]:

- A *trust-region* style algorithm that avoids large destructive updates.
- **Motivation:** Traditional policy gradient methods can produce large policy steps, leading to training instability.
- **Solution:** PPO *clips* the policy update to keep π_θ close to the old policy $\pi_{\theta_{\text{old}}}$.

In RLHF:

- The **reward model** $R_\phi(\cdot)$ is learned from human preferences.
- PPO updates the language model's parameters θ to maximize reward, subject to constraints.

PPO Clipped Objective

Probability Ratio:

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

Clipped Surrogate Objective:

$$L_{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t\big)\Big],$$

where

- $\hat{A}_t$ is an estimator of the **advantage function**,
- ϵ is a small hyperparameter (e.g., 0.1 or 0.2).

Interpretation:

- If $r_t(\theta)$ goes outside $(1-\epsilon, 1+\epsilon)$, the objective stops encouraging that deviation as strongly.
- Prevents overly large policy shifts that could destabilize performance.

Advantage Function & Estimation

Advantage Function:

$$A_{\pi_\theta}(s_t, a_t) = Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t),$$

where

- $Q_{\pi_\theta}(s_t, a_t)$ is the action-value function,
- $V_{\pi_\theta}(s_t)$ is the state-value function.

In practice, we often use GAE (**Generalized Advantage Estimator**) [2]:

$$\hat{A}_t \approx \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k},$$

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

In Language Tasks:

- $\hat{A}_t$ helps measure how good a generated token/sequence is relative to a baseline.
- The reward can be partial (at each token) or only at the end of the generated text.

PPO in Practice for LLMs

1. **Collect samples**: Given prompts, the LLM generates text using the current policy π_θ.
2. **Evaluate with Reward Model**: A separate network R_ϕ assigns a scalar reward r to the generated response.
3. **Compute Advantage**: Estimate $\hat{A}_t$ using the difference between the reward and the baseline (e.g., value function).
4. **Policy Update**:
$$\theta \leftarrow \theta + \alpha \nabla_\theta L_{\text{PPO}}(\theta).$$
5. **Repeat**: Iterate this process with updated θ.

Benefits of PPO:

- Relatively stable training due to clipping.
- Widely adopted in large-scale RLHF (e.g., ChatGPT).

Challenges:

- Requires a well-trained reward model (potential noise or bias).
- Still computationally expensive (multiple passes, large batch sizes).

PPO Example (Hypothetical)

Scenario: Summarization of a news article.

1. **Prompt / State** (s): The article text.
2. **Action** (a): The next token (or chunk) in the summary.
3. **Reward** (r): Higher if the summary is concise, accurate, and well-structured (as judged by R_ϕ).
4. **Policy Update**: PPO adjusts θ to increase probability of better-summarizing tokens.

Illustration of Clipping:

$$r_t(\theta) = \frac{\pi_\theta(\text{“the”} \mid s_t)}{\pi_{\theta_{\text{old}}}(\text{“the”} \mid s_t)}.$$

If $r_t(\theta)$ is too large, the gradient is clipped to prevent big updates.

Strengths and Weaknesses of PPO

Strengths:

- **Robustness**: Clipped objective prevents extreme policy updates.
- **Ease of Use**: Reasonably straightforward hyperparameters (clip range, etc.).
- **Track Record**: Proven in many RL tasks, widely used in LLM RLHF contexts.

Weaknesses:

- **Complexity**: Requires training or maintaining a reward model + value function.
- **Computation-Heavy**: Often large batch sizes and iterative updates.
- **Reward Model Bias**: If the reward model is inaccurate, it can mislead the policy updates.

DPO Overview

Direct Preference Optimization (DPO):

- A more recent technique that directly uses **preference labels** without a traditional RL loop.
- **Idea**: For each pair of responses (y^+, y^-) where y^+ is preferred, increase the likelihood of y^+ and decrease that of y^-.
- Often realized via **pairwise comparisons** and a logistic-like loss.

Advantage:

- Fewer moving parts: No explicit reward or value function is needed.
- Simple training objective: Similar to supervised preference learning.

Potential Limitations:

- Less established for large-scale tasks than PPO-based methods.
- Might not handle complex long-horizon tasks as effectively (ongoing research).

Formal DPO Objective

Pairwise Preference Setting:

$$\mathcal{D} = \{(x, y^+, y^-)\},$$

where x is the prompt or context, and y^+ is the preferred output over y^-.

Loss Function (Logistic):

$$\mathcal{L}_{\text{DPO}}(\theta) = -\sum_{(x,y^+,y^-)\in\mathcal{D}} \log\Big(\sigma(\Delta_\theta(x, y^+, y^-))\Big),$$

where

$$\Delta_\theta(x, y^+, y^-) = \log p_\theta(y^+ \mid x) - \log p_\theta(y^- \mid x),$$

and σ is the logistic sigmoid.

Goal: Minimize $\mathcal{L}_{\text{DPO}}(\theta)$ which *maximizes* the likelihood of the preferred response.

Interpretation of the DPO Loss

Term-by-term Explanation:

- $\log p_\theta(y^+ \mid x)$: Log-likelihood of the preferred response.
- $\log p_\theta(y^- \mid x)$: Log-likelihood of the non-preferred response.
- $\Delta_\theta(x, y^+, y^-)$: If Δ_θ is *large positive*, it means $p_\theta(y^+ \mid x)$ is much higher than $p_\theta(y^- \mid x)$.
- $\sigma(\Delta_\theta)$: Approaches 1 if y^+ is strongly more likely than y^-; approaches 0.5 if they are equally likely.

Loss Minimization:

$$-\log(\sigma(\Delta_\theta)) \quad \text{is low when} \quad \Delta_\theta \gg 0.$$

Hence, we push Δ_θ to be positive and large, favoring y^+ over y^-.

Relates to Binary Classification:

- Just as logistic regression separates classes, DPO separates "preferred" vs. "non-preferred" responses.

DPO Training Procedure

1. **Collect Pairwise Preferences:**
   - For each prompt x, obtain a pair (y^+, y^-) where y^+ is chosen by human raters over y^-.
2. **Compute Gradients:**

$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\theta) = \sum_{(x,y^+,y^-)} \nabla_\theta \Big(-\log \Big(\sigma(\Delta_\theta(x, y^+, y^-)) \Big) \Big).$$

3. **Parameter Update:**

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{\text{DPO}}(\theta).$$

4. **Iterate**: Repeat until convergence or a fixed number of epochs.

No Need for:

- Value function or advantage estimation.
- Trust-region or clipping (as in PPO).

Example of DPO

Scenario: Chatbot Q&A.

- Prompt (x): *"What are some good restaurants in Paris?"*
- Candidate Answers:

$$y^+ = \text{"Le Meurice is famous for fine dining. ..."}$$

$$y^- = \text{"Not sure. I like pizza."}$$

- Humans prefer y^+ over y^- (more informative, relevant).

DPO Step:

$$\Delta_\theta(x, y^+, y^-) = \log p_\theta(\text{"Le Meurice ..."} \mid x) - \log p_\theta(\text{"Not sure ..."} \mid x).$$

- DPO pushes $\log p_\theta(y^+ \mid x)$ higher, $\log p_\theta(y^- \mid x)$ lower.

Strengths and Weaknesses of DPO

Strengths:

- **Simplicity**: Direct pairwise optimization, no need for separate reward or value networks.
- **Efficiency**: Fewer hyperparameters than RL-based methods (no clipping range, advantage function, etc.).
- **Interpretability**: Closer to supervised learning on preferences.

Weaknesses:

- **Limited Evidence at Scale**: Fewer large-scale success stories compared to PPO-based RLHF.
- **Potential Myopia**: Doesn't explicitly account for multi-step or long-horizon dependencies (open question in research).
- **Data Requirements**: Needs sufficient pairwise preference data, which can be expensive to collect.

PPO vs. DPO – The Core Differences

Aspect	PPO	DPO
Core Idea	RL with reward model	Direct optimization of preferences
Training Infrastructure	Reward model + policy + value net	Just preference pairs + model
Objective Type	Clipped surrogate RL objective	Logistic preference loss
Complexity	Higher (advantage, value function, etc.)	Lower (no explicit value function)
Parameter Updates	Constrained by trust region	Unconstrained preference gradient
Stability	Well-studied, stable	Newer, fewer known pitfalls
Use Cases	ChatGPT, large-scale RLHF	Potentially simpler alignment tasks

Practical Considerations

When to Use PPO:

- You already have or plan to build a **reward model**.
- You are comfortable with RL frameworks (value functions, advantage estimation, etc.).
- You want a proven method with a robust track record.

When to Use DPO:

- You have **high-quality pairwise preference data** and want a simpler pipeline.
- You want to avoid some overhead of RL (training a value net, computing advantage).
- You have tasks where direct ranking is sufficient.

Hybrid Approaches?:

- Potential to combine direct preference optimization with partial RL steps.
- Ongoing research on bridging the two approaches.

Example Use Cases

PPO-based Alignments:

- **ChatGPT**: Large-scale RLHF, iterative improvement via user feedback.
- **Content Moderation Systems**: Reward model that heavily penalizes disallowed content, PPO updates policy.
- **Dialogue Agents**: Where the reward is shaped by user satisfaction or correctness signals.

DPO-based Alignments:

- **Simple QA Systems**: If you have pairs of good vs. bad answers.
- **Summaries / Headlines**: If you have pairs of "better summary" vs. "worse summary."
- **Low-Resource Settings**: Quick turn-around with preference pairs from smaller user studies.

Thoughts

- PPO is a **well-established RL method** with strong stability, widely used in RLHF.
- DPO is a **direct preference-based** approach that **removes some RL complexity** but is newer and less tested at scale.
- Both aim for **aligned, human-preferred outputs** in LLMs—core to safe, helpful AI.

Open Questions:

- How to combine the best of both worlds (RL + direct preferences)?
- How to design robust reward models that generalize across many domains?
- How to handle **long-horizon** interactions for chat or multi-step reasoning?

DeepSeek V3: Mixture of Experts (MoE)

- Introduction to the MoE approach in DeepSeek Zero.
- Leverages expert sub-networks with a dynamic gating mechanism.

What is Mixture of Experts (MoE)?

- A neural network design where multiple specialized expert models are combined.
- A gating network determines which experts are activated per input.
- Enables conditional computation: only a subset of experts are used per example.

High-Level Architecture Overview

- DeepSeek V3 integrates a shared backbone with many expert modules.
- The gating network routes each input to the most relevant experts.
- Final output is a weighted aggregation of the selected expert outputs.

Key Components

- **Gating Network:** Processes the input to compute routing probabilities.
- **Expert Networks:** Sub-models that specialize in different aspects of the data.
- Their combined interaction allows for a large effective capacity.

MoE Components

- For an input $\mathbf{x}$, let $E_i(\mathbf{x})$ be the output of expert i.
- The gating network produces weights $g_i(\mathbf{x})$ via a softmax:

$$g_i(\mathbf{x}) = \frac{\exp(W_i \mathbf{x})}{\sum_{j=1}^{N} \exp(W_j \mathbf{x})}$$

- The overall model output is:

$$f(\mathbf{x}) = \sum_{i=1}^{N} g_i(\mathbf{x})\, E_i(\mathbf{x})$$

Sparse Gating Mechanism

- In practice, only the top-k experts (e.g., $k = 2$) are activated per input.
- The gating function is modified to:

$$g_i(\mathbf{x}) = \begin{cases} \frac{\exp(W_i \mathbf{x})}{\sum_{j \in \mathcal{S}(\mathbf{x})} \exp(W_j \mathbf{x})}, & i \in \mathcal{S}(\mathbf{x}) \\ 0, & \text{otherwise} \end{cases}$$

- Here, $\mathcal{S}(\mathbf{x})$ represents the indices of the top-k experts.

Load Balancing and Regularization

- To avoid over-reliance on a few experts, a load balancing loss is added:

$$\mathcal{L}_{balance} = \lambda \sum_{i=1}^{N} \left(\frac{c_i}{\sum_j c_j} - \frac{1}{N} \right)^2$$

- c_i counts the number of times expert i is selected.
- λ is a hyperparameter controlling the regularization strength.

Expert Network Architecture

- Each expert is typically a feed-forward or convolutional sub-network.
- They may include:
  - Dense layers with activation functions (e.g., ReLU).
  - Residual connections for deeper architectures.
- Experts are trained jointly with the gating network.

Integration in DeepSeek Zero

- DeepSeek Zero uses DeepSeek V3's MoE to improve scalability.
- Conditional computation allows a massive increase in capacity without proportional cost.
- Empirical results demonstrate enhanced performance on diverse tasks.

DeepSeek Zero: Overview

- Consists solely of transformer decoder layers.
- Each layer integrates MoE modules to enhance model capacity.
- The MoE allows dynamic routing to expert sub-networks within each decoder layer.

Decoder-Only Transformer Basics

- Uses self-attention to capture contextual dependencies.
- Computes hidden states via:

$$\mathbf{h}^{(l)} = \text{DecoderLayer}(\mathbf{h}^{(l-1)})$$

- The final hidden state is used for next-token prediction.

Integrating Mixture of Experts

- MoE modules are interleaved within selected decoder layers.
- For a given hidden state $\mathbf{h}$, experts provide specialized transformations.
- A gating network determines the contribution of each expert.

Gating Network Details

- For each hidden state $\mathbf{h}$, compute gating scores for N experts:

$$g_i(\mathbf{h}) = \frac{\exp(w_i^\top \mathbf{h})}{\sum_{j=1}^{N} \exp(w_j^\top \mathbf{h})}$$

- These scores determine which experts to activate.
- Typically, only the top-k experts are selected to reduce computation.

Expert Module Processing

- Each expert E_i processes the hidden state:

$$E_i(\mathbf{h}) = \sigma\left(W_{2,i}\,\mathrm{ReLU}(W_{1,i}\mathbf{h})\right)$$

- Experts are typically simple feed-forward networks.
- Their outputs are combined based on the gating scores.

Sparse Activation and Aggregation

- With top-k selection, the gating function becomes:

$$g_i(\mathbf{h}) = \begin{cases} \dfrac{\exp(w_i^\top \mathbf{h})}{\sum_{j \in \mathcal{S}(\mathbf{h})} \exp(w_j^\top \mathbf{h})}, & i \in \mathcal{S}(\mathbf{h}) \\ 0, & \text{otherwise} \end{cases}$$

- The aggregated output for the MoE layer is:

$$f(\mathbf{h}) = \sum_{i \in \mathcal{S}(\mathbf{h})} g_i(\mathbf{h})\, E_i(\mathbf{h})$$

Loss Functions and Regularization

- The primary loss is typically a cross-entropy loss over next-token prediction.
- A load balancing loss is added to ensure even expert utilization:

$$\mathcal{L}_{balance} = \lambda \sum_{i=1}^{N} \left(\frac{c_i}{\sum_{j=1}^{N} c_j} - \frac{1}{N} \right)^2$$

- Here, c_i counts the number of times expert i is selected, and λ is a hyperparameter.

Training and Inference Dynamics

- Training is performed end-to-end using gradient descent (e.g., Adam optimizer):

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- During inference, only the selected experts are computed, enhancing efficiency.
- This conditional computation leads to significant computational savings.

DeepSeek-R1 Overview

- **Objective:** Enhance reasoning capabilities of large language models (LLMs) using reinforcement learning (RL).
- **Models Introduced:**
  - **DeepSeek-R1-Zero:** Trained purely with RL (no supervised fine-tuning at the start).
    - Develops advanced reasoning behaviors (e.g., self-verification, long chain-of-thought).
    - Faces issues like language mixing and poor readability.
  - **DeepSeek-R1:** Improves on R1-Zero by incorporating a small set of cold-start data and multi-stage training.
    - Achieves performance comparable to state-of-the-art models (e.g., OpenAI-o1 series).
    - Produces more coherent and human-friendly outputs.

Group Relative Policy Optimization

- Traditional policy gradient methods (e.g. PPO) rely on a large *critic* network to estimate the value function.
- **GRPO (Group Relative Policy Optimization):**
  - Avoids maintaining a large critic model.
  - Estimates advantage *within a group of sampled outputs*.
  - Reduces computational overhead, making large-scale RL more tractable.
- **DeepSeek Context:**
  - Large Language Models (LLMs) demand huge compute and memory.
  - GRPO serves as a more efficient RL approach to improve *reasoning* in LLMs.

GRPO: Policy Update Objective

Core Objective Function:

$$J_{\text{GRPO}}(\theta) = \mathbb{E}\Big[q \sim P(Q),\ \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(o \mid q)\Big] \Bigg[\frac{1}{G} \sum_{i=1}^{G} \Big(\min\Big(r_i(\theta) \cdot A_i,\ \text{clip}\big(r_i(\theta), 1-\epsilon, 1+\epsilon\big) \cdot A_i \Big) - \beta\, D_{\text{KL}}\big(\pi_\theta \parallel \pi_{\text{ref}}\big) \Big) \Bigg]$$

- q: Query/prompt from distribution $P(Q)$.
- $\{o_i\}$: A *group* of outputs sampled under the old policy $\pi_{\theta_{\text{old}}}$.
- $r_i(\theta) = \dfrac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{\text{old}}}(o_i \mid q)}$: Probability ratio.
- $\text{clip}(\cdot, 1-\epsilon, 1+\epsilon)$: Constrains how far updates can deviate from old policy.
- β: Coefficient for KL penalty to a reference policy π_{ref}.

Group-Based Advantage: A_i

- Rather than using a learned value function, GRPO relies on:

$$A_i = \frac{r_{\text{env},i} - \mu}{\sigma}$$

where:

  - $r_{\text{env},i}$ is the environment reward (or multi-part reward) for the i-th output o_i.
  - $\mu = \text{mean}(\{r_{\text{env},j}\}_{j=1}^{G})$ is the average reward within the group.
  - $\sigma = \text{std}(\{r_{\text{env},j}\}_{j=1}^{G})$ is the standard deviation of rewards within the group.

- **Intuition:**
  - Compare each sample's reward to the average reward of the *same* mini-batch group.
  - No need for a separate critic model to estimate expected returns.
- **Benefit:**
  - Reduces memory footprint and training complexity.
  - Large-scale RL becomes more feasible for LLMs (DeepSeek).

Applying GRPO to LLMs (DeepSeek Example)

- **LLM Setup:**
  - A base language model $\pi_{\theta_{\text{old}}}$.
  - We collect multiple outputs for each prompt q.
- **Group Sampling:**
  1. For each prompt q, sample a group $\{o_1, \ldots, o_G\}$.
  2. Evaluate each output o_i via a reward function:

$$r_{\text{env},i} = r_{\text{acc}} + r_{\text{format}} + \ldots$$

  (e.g., math correctness, chain-of-thought format, language consistency).
- **Update Step:**

$$\theta \leftarrow \theta - \eta \nabla_\theta J_{\text{GRPO}}(\theta)$$

  - Uses the $\text{clip}(\cdot)$ mechanism to bound large deviations in π_θ.
  - Adds a KL term to keep π_θ close to a reference policy (often $\pi_{\theta_{\text{old}}}$ or a baseline).

Multi-Part Rewards in DeepSeek

$$r_{\text{env},i} = r_{\text{accuracy}}(o_i) + r_{\text{format}}(o_i) + r_{\text{lang}}(o_i) + \dots$$

- **Accuracy Reward:** For math or coding tasks, check correctness (e.g., verifying final numeric answer).
- **Format Reward:** Enforce chain-of-thought or designated tags ("<think>, <answer>").
- **Language Consistency Reward:** Penalize code-switching or mixing of multiple languages when undesired.
- **Optional:** Additional sub-rewards for style, helpfulness, alignment, etc.

Group Normalization

Reward signals are normalized within each sampled group, ensuring high or low rewards are relative to other group members.

Why GRPO is Attractive for DeepSeek

- **Reduced Critic Complexity:**
  - Typical PPO includes a critic as large as the policy (huge overhead for 10B+ parameter LLMs).
  - GRPO avoids training/maintaining a large value network.
- **Stabilized Policy Updates:**
  - Clip ratio approach (like PPO) prevents overly large gradient updates.
  - KL penalty ensures the updated policy remains close to a reference, mitigating collapse.
- **Scalable to LLMs:**
  - Group advantage estimation leverages mini-batches of rollouts at scale.
  - Feasible to run large-scale RL on reasoning tasks (math/coding).

Potential Limitations

- **Group-Dependent Signal:**
  - Advantage depends on the distribution of rewards within a group.
  - Highly variable groups could destabilize updates.
- **Reward Engineering:**
  - Multiple sub-rewards may cause *reward hacking* if not carefully balanced.
- **Exploration vs. Exploitation:**
  - With fewer guided signals (compared to a learned critic), exploration might be riskier in early training.
- **Sensitivity to Prompt/Task Distribution:**
  - If the training set does not reflect real usage, the learned policy can overfit to specific reward structures.

Discussion Points

1. **Comparing GRPO vs. PPO:**
   - When might a learned value function still be advantageous?
   - Does group-based advantage scaling degrade with very large group sizes?
2. **Reward Curves and Annealing:**
   - Should the KL penalty β change over time to allow more exploration or exploitation?
   - How to best schedule ϵ-clipping for policy ratio?
3. **Alignment with Human Preferences:**
   - Incorporating user feedback in the RL loop (helpfulness, harmlessness).
   - Balancing correctness and alignment with minimal label overhead.

Methodology and Pipeline

- **Reinforcement Learning Approach:**
  - Utilizes **Group Relative Policy Optimization (GRPO)** to avoid a heavy critic model.
  - Employs a rule-based reward system:
    - **Accuracy Rewards:** Verify correct answers (e.g., math problems, code outputs).
    - **Format Rewards:** Ensure outputs follow a designated chain-of-thought (CoT) structure.
- **Self-Evolution:**
  - The model naturally extends its CoT length and rethinks its approach ("aha moments").

Cold Start & Multi-Stage Training (DeepSeek-R1)

- **Cold Start:** Fine-tune the base model with a curated set of long CoT examples.
- **Multi-Stage Pipeline:**
  - **Stage 1:** Initial RL training on the cold-start fine-tuned model.
  - **Stage 2:** Use rejection sampling to generate additional supervised fine-tuning (SFT) data.
  - **Stage 3:** A final RL stage refines the model with broader prompts and human-friendly rewards.
- **Outcome:** Improved readability, language consistency, and overall reasoning performance.

Distillation to Smaller Models

- Reasoning capabilities learned by DeepSeek-R1 are distilled into smaller models (1.5B to 70B parameters).
- Distillation transfers the advanced reasoning skills efficiently.
- Smaller distilled models sometimes outperform counterparts trained solely with RL.

Experimental Evaluation

- **Benchmarks:**
  - Mathematics: AIME 2024, MATH-500, CNMO.
  - Coding: Codeforces, LiveCodeBench.
  - Knowledge and Long-Context: MMLU, SimpleQA, FRAMES.
- **Metrics:** Pass@1 scores, Elo ratings, accuracy percentages.
- **Results:**
  - DeepSeek-R1 achieves performance comparable to OpenAI-o1.
  - Significant improvements over earlier models (e.g., DeepSeek-V3).

Discussion and Future Directions

- **RL vs. Distillation:**
  - RL directly improves reasoning but can be computationally expensive.
  - Distillation is an economical alternative for deploying reasoning in smaller models.
- **Challenges:**
  - Issues such as language mixing, sensitivity to prompt formulations, and limited performance in non-reasoning domains.
- **Future Research:**
  - Enhancing multi-turn dialogue, structured outputs (e.g., JSON), and long chain-of-thought strategies.
  - Addressing scalability and efficiency in large-scale RL.

Talking Points

- **Reinforcement Learning in LLMs:**
  - How does RL drive advanced reasoning capabilities in LLMs?
  - Benefits of GRPO over traditional actor-critic methods.
- **Supervised Fine-Tuning vs. Pure RL:**
  - Trade-offs between starting with pure RL (DeepSeek-R1-Zero) versus using cold-start data.
- **Reward Engineering:**
  - Discussion on rule-based rewards and the potential of neural reward models.
- **Emergent Behaviors:**
  - Importance of extended chain-of-thought processes and the emergence of "aha moments."
- **Distillation Techniques:**
  - How can reasoning capabilities be effectively transferred to smaller models?

Discussion Questions

1. How does pure RL compare with a cold-start and multi-stage training pipeline in terms of performance and learning efficiency?
2. What challenges arise when using rule-based reward models, and how might neural reward models mitigate these issues?
3. In what ways do extended chain-of-thought processes and "aha moments" enhance a model's reasoning abilities?
4. What are the benefits and limitations of distilling large teacher models into smaller student models, especially under resource constraints?
5. How well do current benchmarks capture the depth of reasoning, and what improvements could be made in future evaluations?
6. Considering the computational demands of large-scale RL, what strategies could improve the scalability and practicality of these models?

References


Schulman, J. et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv preprint arXiv:1707.06347.


Schulman, J. et al. (2015). *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv preprint arXiv:1506.02438.