A Concise Review of Reinforcement Learning Methods in the context of LLMs

Anjan Goswami

February 17, 2025

Contents

1 Introduction and Historical Context

Reinforcement Learning (RL) is a subfield of AI that studies how an agent learns to act in an environment to maximize long-term rewards. Historically, RL ideas grew from three major streams:

- **Dynamic Programming (DP):** Bellman [1957] introduced the concept of optimal value functions and the principle of optimality.
- **Stochastic Approximation:** Robbins and Monro [1951] provided iterative methods to solve estimation problems, paving the way for value iteration in RL.

- **Temporal-Difference (TD) Learning and Early Games:** Samuel [1959] built a checkers-playing program with rudimentary TD-like updates. Later, Tesauro [1995] used TD for backgammon (TD-Gammon), achieving near-expert play.

By the 1990s, RL was formalized via *Markov Decision Processes (MDPs)* [Puterman, 1994], and methods like Q-learning [Watkins and Dayan, 1992] became standard for small discrete tasks. Around 2013–2015, combining RL with **deep neural networks**—termed *Deep RL*—enabled tackling large-scale problems (e.g., Atari from pixels [Mnih et al., 2015]). Subsequently, Silver et al. [2017] demonstrated RL's power with self-play and Monte Carlo Tree Search (MCTS) to master complex board games like Go.

In parallel, *Large Language Models* (LLMs) soared in performance via generative pretraining. However, large models often needed additional **alignment** to produce factually correct, helpful, or safe responses. Hence, **Reinforcement Learning from Human Feedback (RLHF)** [Ouyang et al., 2022] emerged as a powerful framework to shape LLM outputs according to user preferences.

2 Classical RL Foundations

2.1 Basic MDP Formulation

An MDP is defined by:

$$(\mathcal{S}, \mathcal{A}, P, r, \gamma),$$

where:

- $\mathcal{S}$ is the set of states,
- $\mathcal{A}$ is the set of actions,
- $P(s' \mid s, a)$ is the transition probability to go from state s to s' under action a,
- $r(s, a)$ is the reward function,
- $\gamma \in [0, 1]$ is the discount factor.

The goal is to find a **policy** $\pi(a \mid s)$ that maximizes expected return:

$$\max_{\pi} \ \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \, r(s_t, a_t)\Big].$$

2.2 Blocks World and Maze Problems

Classic toy examples:

Blocks World:

- **States**: Each unique arrangement of N blocks in stacks.
- **Actions**: Move top block from stack i to top of stack j.
- **Transition**: Deterministic re-arrangement of blocks.
- **Reward**: $+1$ upon achieving a goal arrangement, 0 otherwise.

Maze Navigation:

- **States**: Coordinates (x, y) in a grid.
- **Actions**: $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ if not blocked.
- **Transition**: Move to the next cell with probability 1 (or 0 if blocked).
- **Reward**: $+1$ on reaching the exit cell, 0 otherwise.

Though simple, these exemplify tabular RL updates and the interplay of exploration and exploitation.

2.3 Value-Based Methods: Q-Learning

Q-learning [Watkins and Dayan, 1992] learns the *action-value function* $Q(s, a)$ by bootstrapping:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \Big]. \tag{1}$$

It converges to $Q^*(s, a)$ under standard conditions if each state-action pair is visited infinitely often.

Algorithm 1 Basic Q-learning (Maze or Blocks World)

```
Initialize Q(s, a) = 0 for all states s and actions a
for episode = 1 to N do
    Reset environment to initial state s
    while s not terminal do                      ▷ Epsilon-greedy action selection
        a ← { arg max_a Q(s, a)   with probability (1 − ϵ),
            { random action       otherwise.
        s′, r ← StepEnv(s, a)
        Update Q(s, a) via Eq. (1)
        s ← s′
    end while
end for
```

Tabular Q-learning Pseudocode

2.4 Actor-Critic Methods

Unlike Q-learning, *actor-critic* [Sutton and Barto, 2018] decomposes learning into:

- **Actor**: A policy $\pi_\theta(a \mid s)$,
- **Critic**: A value function $V_\psi(s)$.

The critic reduces variance by providing a baseline. A simple one-step update:

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

$$\psi \leftarrow \psi + \beta_v \, \delta_t \, \nabla_\psi V_\psi(s_t), \tag{2}$$

$$\theta \leftarrow \theta + \beta_\theta \, \delta_t \, \nabla_\theta \log \pi_\theta(a_t \mid s_t). \tag{3}$$

Algorithm 2 Actor-Critic (One-step)

Initialize parameters θ (actor), ψ (critic)
for episode = 1 to M **do**
  Reset environment, obtain s_0
  for $t = 0$ to T-1 **do**
    Sample $a_t \sim \pi_\theta(\cdot \mid s_t)$
    $s_{t+1}, r_t \leftarrow \text{StepEnv}(s_t, a_t)$
    $\delta_t = r_t + \gamma V_\psi(s_{t+1}) - V_\psi(s_t)$
    $\psi \leftarrow \psi + \beta_v \, \delta_t \, \nabla_\psi V_\psi(s_t)$
    $\theta \leftarrow \theta + \beta_\theta \, \delta_t \, \nabla_\theta \log \pi_\theta(a_t \mid s_t)$
    if s_{t+1} is terminal **then**
      break
    end if
  end for
end for

Actor-Critic Pseudocode

3 Deep RL Approaches

When the state or action spaces grow large (e.g., visual inputs or continuous controls), *deep neural networks* serve as function approximators.

3.1 Deep Q-Network (DQN)

Mnih et al. [2015] introduced DQN for playing Atari from raw pixels. A CNN approximates $Q_\psi(\text{image}, a)$; key techniques include:

- **Replay buffer**: store transitions in memory and sample mini-batches to break correlation.
- **Target network**: a slowly updated copy $Q_{\psi'}$ to stabilize training.

3.2 Actor-Critic for Continuous Spaces

Robotics often deals with continuous actions. Methods like *DDPG* [Lillicrap et al., 2015], *TD3* [Fujimoto et al., 2018], and *SAC* [Haarnoja et al., 2018] extend actor-critic with an off-policy approach plus a parametric policy $\mu_\theta(s)$ or $\pi_\theta(a \mid s)$.

3.3 Model-Based RL and Matrix Computations

Instead of directly learning Q or π, model-based RL learns $\hat{P}(s' \mid s, a)$ and $\hat{r}(s, a)$. Then *planning* or partial rollouts can reduce sample complexity. In small MDPs, we might solve linear systems $(I - \gamma P)v = R$ [Puterman, 1994]. But in large-scale tasks, approximate planning or partial expansions are typical.

4 Monte Carlo Tree Search (MCTS) and AlphaZero

Monte Carlo Tree Search is a planning algorithm often used in discrete, perfect-information games:

1. **Selection**: Traverse existing search tree using a selection policy (e.g. UCB).
2. **Expansion**: Expand a leaf node by adding a new child.
3. **Simulation**: Simulate a game outcome via random rollout or a value function approximation.
4. **Backpropagation**: Update the statistics ($Q(s, a)$, visit counts) along the visited path.

Algorithm 3 Monte Carlo Tree Search (MCTS) Outline

---

function MCTS(s_{root})
  for $m = 1$ to M **do**
    $s \leftarrow s_{\text{root}}$ ▷ 1) Selection
    while s is fully expanded **and** not terminal **do**
      $a \leftarrow \arg\max_a \big[Q(s, a) + U(s, a)\big]$
      $s \leftarrow \text{NextState}(s, a)$
    end while ▷ 2) Expansion
    if s not terminal **then**
      Expand a child node for an unvisited action
      $s \leftarrow \text{NextState}(s, a')$
    end if ▷ 3) Simulation
    $z \leftarrow \text{RolloutOrValue}(s)$ ▷ 4) Backpropagation
    Update $Q(\cdot)$ along visited path with z
  end for
  return $\arg\max_a Q(s_{\text{root}}, a)$
end function

MCTS Pseudocode

4.1 AlphaZero and Self-Play

Silver et al. [2017] combined MCTS with deep policy/value networks trained via self-play. This yields superhuman performance in games (Go, Chess, Shogi) without human heuristics. The policy network guides expansions, while the value network replaces random rollouts.

5 RL in Large Language Models (LLMs)

Though many RL tasks assume multi-step interactions, *LLMs* often deal with single-turn or short-turn interactions (prompt $\rightarrow$ response). Nevertheless, RL can be highly effective in aligning LLM outputs to user preferences or correctness signals.

5.1 Importance of RL for Improving LLM Efficacy

- **Alignment and Safety**: LLMs can produce factually incorrect or undesirable text if only trained via maximum likelihood. RL with a well-defined reward can penalize such behaviors.
- **Reducing Hallucinations**: RL encourages *truthful and consistent* answers if the reward model checks for factual correctness.
- **Personalization**: By shaping reward signals (human preference data), RL can produce more user-tailored responses.

5.2 Reward Functions in RLHF & Metric Targets (e.g., NDCG)

In *Reinforcement Learning from Human Feedback (RLHF)* [Ouyang et al., 2022], we typically train a **reward model** $r_\phi(q, o)$ from pairwise comparisons (o^+, o^-) to reflect which answer humans prefer. If one aims to maximize something like *NDCG (Normalized Discounted Cumulative Gain)* for relevance, token-level or segment-level scoring might be used. However, in practice, RLHF often collapses to a single scalar reward for the entire output, due to annotation constraints.

5.3 PPO & FRPO: Why They Have Big Impact

PPO (Proximal Policy Optimization) Schulman et al. [2017] introduced PPO to stabilize policy gradients by clipping the probability ratio. For LLM alignment [Ouyang et al., 2022]:

- We sample outputs from $\pi_{\theta_{\text{old}}}$,
- Score them with reward model r_ϕ,
- Compute advantages (via a learned value function or a baseline),
- Update π_θ using the *clipped objective* plus a KL penalty to avoid deviating too far from the reference model.

This approach *stabilizes* training and ensures the updated language model does not degrade fluency.

FRPO (Fine-tuning with Rejection *or* Relative Policy Optimization) While the acronym "FRPO" is not standard in all literature, there are similar variants:

- **RFT (Rejection Sampling Fine-Tuning)**: Filter outputs above a certain reward threshold and fine-tune on these "accepted" outputs.
- **GRPO (Group Relative Policy Optimization)** [Z. Shao et al., 2024]: Sample a group of outputs for each prompt, compute a *group-based* advantage by subtracting the mean reward of the group. This avoids a separate critic, reducing memory usage.

These methods typically show strong performance gains while simplifying the RL loop (fewer large model components or tricky advantage functions).

5.4 Inference-Time Computation as an RL Problem

One can conceptualize the entire process of LLM *token generation* as a multi-step RL environment:

- **State**: The partial conversation or partial token sequence (plus hidden states in the Transformer).
- **Action**: Generating the next token from the model's distribution.
- **Reward**: Derived from correctness or user satisfaction (possibly known only at the final token).
- **Goal**: Produce a solution matching or exceeding a ground-truth standard.

However, the *massive* action space (tens of thousands of tokens) and the *high cost* of large Transformer forward passes makes repeated planning (e.g., MCTS) computationally prohibitive. Hence, *policy gradient* approaches (PPO, GRPO) are favored, typically run *offline* or in short on-policy cycles.

6 Challenges in RL Training Systems

- **Reward Design**: In LLM alignment, capturing desired traits (factual correctness, style, safety) in a single scalar is non-trivial.
- **Exploration & Sample Efficiency**: Generating large batches of tokens is expensive. RL often needs many samples, which is challenging at LLM scale.
- **Hyperparameter Sensitivity**: Methods like PPO or GRPO rely on carefully tuned learning rates, KL coefficients, or clipping thresholds.
- **Scaling and Distribution**: Distributed RL frameworks (e.g., *Anyscale*) are emerging to handle large models, but the details of synchronous updates, replay buffers, and partial on-policy sampling remain non-trivial.

7 Conclusions and Future Directions

Reinforcement Learning has evolved from tabular MDPs (Sections 2–3) to advanced search-based solutions like AlphaZero (Section 4), and more recently it has become integral in shaping Large Language Models (Section 5). Although LLM tasks appear single-step, RL can still yield significant benefits, especially for alignment, correctness, and user-centric improvements.

Future avenues include:

- **Multi-turn Dialogue as a Real Environment**: Each user–model turn can be a step, potentially enabling deeper RL approaches (even MCTS or model-based planning) if partial expansions can be tested or simulated.
- **Advanced Reward Modeling**: Going beyond scalar preference to incorporate metrics like token-level or segment-level NDCG, integrated into chain-of-thought or solution correctness.
- **Distributed / Scalable RLHF**: Handling huge models and large user data with minimal overhead.

References

R. E. Bellman. *Dynamic Programming.* Princeton University Press, 1957.

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, et al. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, 2018.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, et al. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

X. Ouyang et al. Training language models to follow instructions with human feedback. *arXiv:2203.02155*, 2022.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 1994.

H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

John Schulman, Filip Wolski, Prafulla Dhariwal, et al. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2nd edition, 2018.

Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

C. J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

Z. Shao, P. Wang, Q. Zhu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv:2402.03300*, 2024.