A Comprehensive Exposition of Modern Reinforcement Learning: From Foundations to Frontiers

By Taha Majlesi

July 12, 2025

Abstract

This document provides a comprehensive exposition of modern Reinforcement Learning (RL), beginning with its foundational principles and progressing to the cutting-edge algorithms that power today's most advanced AI systems. We will dissect the core mathematical framework of Markov Decision Processes (MDPs), explore the significance of Bellman's equations, and trace the historical evolution of Deep RL from the DQN revolution to the practical applications of PPO in systems like ChatGPT. The aim is to offer a structured and explanable journey into the theory and practice of RL, making complex concepts accessible and demonstrating their real-world impact.

Contents

Part I

Foundations of Reinforcement Learning

1 The Reinforcement Learning Paradigm

Reinforcement Learning (RL) represents a distinct and powerful paradigm within machine learning, focused on training intelligent agents to make optimal decisions through interaction and feedback. [15, 16] Unlike supervised learning, which relies on labeled datasets of input-output pairs, or unsupervised learning, which seeks to find structure in unlabeled data, RL is concerned with goal-directed learning from experience. [5] This approach is fundamentally rooted in the concept of trial and error, where an agent learns to map situations to actions to maximize a cumulative numerical reward signal. [12] While foundation models are heavily data-driven, focusing on cleaning and optimizing datasets, RL aims for creative behaviors, requiring optimization during inference time to adapt and go beyond static data patterns.

1.1 Defining the Agent-Environment Interface

The core of reinforcement learning is the **agent-environment interface**. [4, 48] This framework consists of two primary components:

- **The Agent:** The learner or decision-maker. This could be a robot navigating a maze, a program playing a game, or a large language model generating text. [49]
- **The Environment:** The world in which the agent operates. The agent interacts with the environment in a sequence of discrete time steps. [4]

The interaction loop proceeds as follows: At each time step t, the agent observes the current state of the environment, s_t. Based on this state, the agent selects an action, a_t. The environment, in response, transitions to a new state, s_{t+1}, and provides the agent with a scalar reward, r_{t+1}. The agent's ultimate objective is to learn a strategy, known as a **policy (π)**, that maximizes the total accumulated reward over the long run.

1.2 The Modern Imperative: RL in ChatGPT and Beyond

The theoretical framework of RL has recently transitioned into a cornerstone of state-of-the-art AI, most notably in the development of advanced large language models (LLMs) like ChatGPT. [2, 17] The process, often termed **Reinforcement Learning from Human Feedback (RLHF)**, showcases RL's unique capability to align complex models

with nuanced, subjective human preferences. [6, 23] The application of RLHF typically involves a three-step pipeline:

1. **Supervised Fine-Tuning (SFT):** An initial pre-trained model is fine-tuned on a high-quality dataset of demonstrations.
2. **Reward Model (RM) Training:** A separate reward model is trained on human-ranked comparisons of model outputs, learning to predict a scalar score that represents human preference. [19]
3. **Reinforcement Learning Optimization:** The SFT model (now the policy) is further optimized using an RL algorithm, most commonly Proximal Policy Optimization (PPO), to maximize the rewards from the learned preference model.

This process marks a profound evolution where the "reward" itself is a learned model, an internalization of complex human values, enabling the application of RL to domains where objectives are not easily programmable.

1.3 A Fundamental Dichotomy: SFT Memorizes, RL Generalizes

A key motivation for employing RL is its superior ability to generalize compared to standard supervised fine-tuning. [11, 32] While SFT can achieve high performance on tasks similar to its training data (in-distribution), it often struggles with novel scenarios (out-of-distribution) because it tends to memorize specific input-output pairings. [37] In contrast, RL encourages the agent to learn a more robust, underlying decision-making process. By exploring the environment and receiving feedback, an RL-trained agent develops a policy that is less brittle and more adaptable, a critical trait for AI systems operating in the real world. [34]

2 Formalizing the World: The Markov Decision Process (MDP)

To move from intuitive concepts to practical algorithms, RL relies on a formal mathematical framework known as the **Markov Decision Process (MDP)**. [10] An MDP provides a complete specification of the environment and the learning problem.

2.1 The Five Pillars of the MDP: $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$

An MDP is formally defined as a 5-tuple, where each component has a precise meaning:

- $\mathcal{S}$ **(State Space):** The set of all possible states $s \in \mathcal{S}$ that the agent can be in.

- $\mathcal{A}$ **(Action Space):** The set of all possible actions $a \in \mathcal{A}$ the agent can take.
- $\mathcal{P}$ **(Transition Function):** The probability of transitioning from state s to state s' after taking action a, denoted as $\mathcal{P}(s'|s,a) = \Pr(S_{t+1} = s'|S_t = s, A_t = a)$.
- $\mathcal{R}$ **(Reward Function):** The scalar reward r received after transitioning from state s to state s' by taking action a.
- γ **(Discount Factor):** A number $\gamma \in [0,1]$ that balances the importance of immediate versus future rewards. A reward received k steps in the future is discounted by γ^k.

The discount factor is fundamental, shaping the agent's behavior. A γ near 0 creates a "myopic" agent, while a γ near 1 creates a "farsighted" agent. The effective planning horizon can be approximated as $1/(1-\gamma)$.

2.2 The Markov Property: A Memoryless World

The MDP framework is built upon the **Markov Property**, which asserts that the future is independent of the past, given the present. This means the state transition and reward depend only on the current state and action, not the entire history. This assumption makes the RL problem tractable, allowing the policy to be a function of the current state, $\pi(s_t)$, rather than the full history.

2.3 The Agent's Goal: Maximizing Expected Return

The agent's goal is to find a policy that maximizes the expected cumulative discounted reward, known as the **return**. The return from time step t, denoted G_t, is:

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

where H is the horizon. The objective is to learn a policy π that maximizes $\mathbb{E}[G_t|S_t = s]$.

2.4 The Agent's Strategy: Policies (π)

The policy, π, is the agent's "brain," mapping states to actions.

- **Deterministic Policy:** A direct mapping $\pi : \mathcal{S} \to \mathcal{A}$.
- **Stochastic Policy:** A mapping to a probability distribution over actions, $\pi(a|s) = \Pr(A_t = a|S_t = s)$.

In Deep RL, policies are often represented by a neural network with parameters θ, denoted $\pi_\theta(a|s)$.

3 Evaluating Success: Value Functions and Bellman Equations

To find an optimal policy, an agent needs to quantify how "good" a state or action is. This is the role of **value functions**.

3.1 The State-Value Function (V^π)

The state-value function for a policy π, denoted $V^\pi(s)$, is the expected return an agent can achieve by starting in state s and following policy π:

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

3.2 The Action-Value Function (Q^π)

The action-value function (or Q-function), $Q^\pi(s, a)$, is the expected return after taking action a in state s and then following policy π:

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

3.3 The Bellman Expectation Equation

A cornerstone of RL is the **Bellman equation**, which provides a recursive relationship for value functions. The Bellman expectation equation for V^π expresses the value of a state as the expected immediate reward plus the discounted expected value of the next state:

$$V^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s',r} \mathcal{P}(s', r|s, a)[r + \gamma V^\pi(s')]$$

This equation transforms a long-term planning problem into a manageable, one-step recursive calculation.

3.4 Case Study: Calculating Value Functions in Grid World

The Grid World examples illustrate how value functions depend on MDP parameters.

- **Deterministic, $\gamma = 1$:** The value of a state is the undiscounted sum of future rewards.
- **Deterministic, $\gamma = 0.9$:** Rewards further in the future are worth less. The value of a state k steps from a goal with reward $+1$ is γ^{k-1}.

- **Stochastic, $\gamma = 0.9$:** The value must be calculated using the full Bellman equation, averaging over all possible outcomes. Uncertainty reduces the value of states, especially those near hazards.

4 The Pursuit of Perfection: Optimal Policies

The ultimate goal is to find the best possible policy—the **optimal policy**, π^*.

4.1 Optimal Value Functions (V^*, Q^*)

An optimal policy π^* has a value function greater than or equal to that of any other policy for all states. All optimal policies share the same unique **optimal state-value function**, $V^*(s)$, and **optimal action-value function**, $Q^*(s, a)$.

$$V^*(s) = \max_\pi V^\pi(s) \quad \text{and} \quad Q^*(s, a) = \max_\pi Q^\pi(s, a)$$

Once Q^* is known, an optimal policy can be constructed by acting greedily: $\pi^*(s) = \arg\max_a Q^*(s, a)$.

4.2 The Bellman Optimality Equation

The optimal value functions satisfy the **Bellman optimality equation**, which includes a `max` operator reflecting that an optimal policy must select the best action.

- **For V^*:** $V^*(s) = \max_{a \in \mathcal{A}} \sum_{s',r} \mathcal{P}(s', r|s, a)[r + \gamma V^*(s')]$
- **For Q^*:** $Q^*(s, a) = \sum_{s',r} \mathcal{P}(s', r|s, a)[r + \gamma \max_{a'} Q^*(s', a')]$

These equations are the foundation for many RL algorithms, including Value Iteration and Q-learning.

4.3 Case Study: Deriving Optimal Policies in Grid World

The optimal policy is contingent on the MDP parameters (γ and noise).

- **($\gamma = 0.1$, noise=0.5):** Myopic and risky. Prefers the close exit (+1) despite the cliff risk.
- **($\gamma = 0.99$, noise=0):** Farsighted and safe. Prefers the distant exit (+10) and can execute a perfect path.

- **($\gamma = 0.99$, noise=0.5):** Farsighted but risky. Prefers the distant exit (+10) but accepts the risk from high noise.
- **($\gamma = 0.1$, noise=0):** Myopic and safe. Prefers the close exit (+1) and avoids the cliff.

5 Beyond Full Observability: POMDPs

In many real-world applications, the agent does not know the true state of the environment. This is handled by the **Partially Observable Markov Decision Process (POMDP)**.

5.1 The Veil of Perception: States vs. Observations

In a POMDP, the agent receives an **observation** o, which is probabilistically related to the hidden underlying state s. The policy must now be a function of the history of observations.

5.2 The Belief State: A Probabilistic Approach

Since the true state is unknown, the agent maintains a probability distribution over all possible states, known as a **belief state**, $b(s)$. The belief state is a sufficient statistic for the history and is updated at each step using Bayes' rule.

5.3 The POMDP Framework

A POMDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, \mathcal{O}, \gamma)$, where Ω is the set of observations and $\mathcal{O}$ is the observation probability function. Solving a POMDP means finding an optimal policy $\pi^*(b) \to a$ that maps belief states to actions. This transforms the POMDP into a continuous-state MDP, which is significantly more challenging to solve.

Table 1: Comparison of MDPs and POMDPs

Feature	**Markov Decision Process (MDP)**	**Partially Observable MDP (POMDP)**
State Observability	Fully observable; state is known.	Partially observable; state is hidden.
Agent's Input	True state s_t.	Observation o_t.
Core Problem	Find an optimal policy $\pi^*(s) \to a$.	Find an optimal policy $\pi^*(b) \to a$.
Policy Function	Maps states to actions.	Maps belief states to actions.
State Space	The environment's state space $\mathcal{S}$.	The continuous space of belief distributions over $\mathcal{S}$.
Key Challenge	Credit assignment over time.	State estimation (belief tracking) and planning in a continuous belief space.

5.4 Real-World Applications of POMDPs

POMDPs are essential for realistic problems like robotics, medical diagnosis, machine maintenance, and even LLM applications where user intent is a hidden state.

Part II

Deep Reinforcement Learning in Practice

This section traces the historical evolution of Deep RL, showing how the integration of deep learning unlocked the practical application of RL theory to complex problems.

Table 2: Key Milestones in Deep Reinforcement Learning (2013-2019)

Year	**Algorithm/Milestone**	**Institution(s)**	**Key Innovation**
2013	Atari (DQN)	DeepMind	Combining Q-learning with CNNs.
2014	2D Locomotion (TRPO)	Berkeley	Stable policy gradient updates.
2015	AlphaGo	DeepMind	MCTS + deep policy/value networks.
2016	3D Locomotion (TRPO+GAE)	Berkeley	Generalized Advantage Estimation (GAE
2016	Real Robot Manipulation	Berkeley, Google	Guided policy search for real hardware.
2017	Dota2 (PPO)	OpenAI	Scalable first-order policy optimization.
2018	DeepMimic	Berkeley	Learning motor skills from motion captu
2019	AlphaStar	DeepMind	Mastering complex real-time strategy ga
2019	Rubik's Cube (PPO+DR)	OpenAI	Solving manipulation with domain rando

6 Value-Based Learning: The Deep Q-Network (DQN) Revolution (2013)

The 2013 paper "Playing Atari with Deep Reinforcement Learning" by DeepMind presented the **Deep Q-Network (DQN)**, the first algorithm to successfully train a deep neural network to learn control policies directly from high-dimensional, raw sensory input.

6.1 From Q-Tables to Q-Networks

Traditional Q-learning uses a lookup table, which is infeasible for large state spaces like Atari screen images. DQN overcomes this by using a deep convolutional neural network (CNN) as a function approximator to estimate the optimal action-value function, $Q^*(s, a; \theta)$.

6.2 Core Innovations: Experience Replay and Target Network

DQN introduced two key innovations to stabilize training:

- **Experience Replay:** Storing experiences in a replay memory and sampling mini-batches randomly breaks temporal correlations and increases data efficiency.
- **Target Network:** Using a separate, periodically updated target network to compute the target Q-value provides a stable target for the updates, preventing oscillations and divergence.

6.3 Case Study: Conquering Atari Games

DQN learned to play a suite of Atari games at a superhuman level from raw pixels, demonstrating an unprecedented level of generality and end-to-end learning.

7 Stabilizing Policy Updates: Trust Region Policy Optimization (TRPO) (2014-2016)

For continuous action spaces, policy gradient methods are used. **Trust Region Policy Optimization (TRPO)** was a landmark algorithm that solved their inherent instability.

7.1 The Perils of Standard Policy Gradients

Standard policy gradient methods are highly sensitive to the step size (learning rate). A step that is too large can cause a catastrophic drop in performance.

7.2 The TRPO Method

TRPO takes the largest possible step that improves the policy, subject to a constraint that the new policy does not deviate too far from the old one. This constraint is enforced on the policy's output distribution using the **Kullback-Leibler (KL) divergence**, ensuring monotonic performance improvements.

7.3 Case Study: Learning Complex Locomotion

TRPO was able to learn sophisticated behaviors like walking and swimming for simulated agents in continuous control environments. Performance was further improved by **Generalized Advantage Estimation (GAE)**, which reduces variance in the advantage function estimate.

8 The Apex of Search and Learning: AlphaGo (2015)

DeepMind's **AlphaGo** defeated a top professional Go player by masterfully synthesizing deep neural networks with **Monte Carlo Tree Search (MCTS)**.

8.1 A Hybrid Approach: Neural Networks + MCTS

AlphaGo uses two neural networks to guide the MCTS algorithm, pruning the vast search tree to focus on the most promising lines of play.

- **The Policy Network:** Predicts the most promising moves from a given position.
- **The Value Network:** Estimates the probability of winning from a given position.

8.2 Interaction with MCTS

During a game, MCTS builds a search tree. The policy network suggests promising moves to expand the tree (Expansion), and the value network evaluates the new positions (Evaluation). This information is then propagated back up the tree (Backup) to inform the final move selection.

8.3 Case Study: Superhuman Performance in Go

AlphaGo's victory over Lee Sedol was a watershed moment for AI, demonstrating that a hybrid of deep learning and search could master tasks requiring deep strategic intuition. The successor, **AlphaGo Zero**, learned entirely from self-play, surpassing all previous versions.

9 The Era of Practicality: Proximal Policy Optimization (PPO) (2017)

Proximal Policy Optimization (PPO) captures the stability of TRPO while simplifying its implementation, making it a go-to algorithm for many deep RL applications.

9.1 Simplifying TRPO: The Clipped Surrogate Objective

PPO replaces TRPO's complex constrained optimization with a simpler **clipped surrogate objective function**. This mechanism effectively enforces a trust region by clipping the policy update if it would change the policy too drastically, providing stability without TRPO's computational complexity. The objective is:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

where $r_t(\theta)$ is the probability ratio of the new and old policies and $\hat{A}_t$ is the advantage estimate.

9.2 PPO's Role in Modern AI: From Dota2 to RLHF

PPO's robustness and ease of use have made it one of the most widely used RL algorithms. It was the core algorithm behind the OpenAI Five agent that mastered Dota 2 and is the workhorse for the RLHF alignment stage in models like ChatGPT.

9.3 Case Study: The Three-Step Process of RLHF

PPO is crucial in the final step of the RLHF pipeline. It stably updates the LLM's policy based on rewards from the learned reward model, aligning the model's outputs with human preferences without causing performance collapse.

Part III

Synthesis and Future Outlook

10 Concluding Analysis: Unifying Principles of RL

This exploration reveals a field defined by a powerful and generalizable set of principles, structured around the quest to enable autonomous agents to learn optimal behavior through interaction.

10.1 Recapitulation of the Core Theoretical Framework

At its heart, RL is structured around the elegant mathematical framework of the MDP. The Bellman equations provide the master key to solving these problems recursively, and value functions serve as the currency of long-term success, guiding the search for an optimal policy.

10.2 The Symbiotic Relationship Between Theory, Algorithms, and Computation

The history of deep RL is a virtuous cycle between theory, algorithmic innovation, and computational power.

- **Theory enables algorithms** (e.g., Bellman equation $\rightarrow$ Q-learning).
- **Algorithms leverage computation** (e.g., Q-learning + CNNs $\rightarrow$ DQN).
- **Computation enables new frontiers**, which in turn drive new theory and algorithms (e.g., LLM alignment $\rightarrow$ PPO).

10.3 The Path Forward: Open Challenges and the Future of RL

Despite its successes, RL faces significant challenges. **Sample inefficiency** remains a primary concern, as algorithms often require millions of interactions to learn. Developing more efficient exploration strategies and leveraging model-based approaches are active areas of research aimed at overcoming this bottleneck, especially for real-world applications in robotics and beyond.