Model-Based Reinforcement Learning: A Comprehensive Report on Planning and Optimization

Taha Majlesi

July 17, 2025

Contents

1 The Model-Based Paradigm in Reinforcement Learning

1.1 Defining the Landscape: Model-Free vs. Model-Based RL

Reinforcement Learning (RL) is a domain of machine learning focused on how an intelligent agent should act within an environment to maximize a cumulative reward. Two primary strategies have emerged: model-free and model-based RL. The core difference is how the agent uses its experiences to learn and make decisions.

Model-Free Reinforcement Learning (MFRL) methods learn a policy or value function directly from environmental interactions. The agent perceives the environment as a "black box," learning through trial and error which actions yield high rewards in specific states, without building an explicit understanding of the environment's underlying mechanics. This is like learning to ride a bicycle purely through practice, without studying the physics of balance. Mathematically, MFRL directly optimizes the parameters θ of a policy $\pi_\theta(a|s)$ to maximize the expected total reward over a trajectory τ:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(s_t, a_t) \right] \tag{1}$$

Here, the environment's transition dynamics, $p(s'|s, a)$, are unknown and not explicitly estimated.

Model-Based Reinforcement Learning (MBRL) employs a two-stage approach. First, the agent learns a *model* of the environment, which approximates its dynamics by predicting the next state s' and reward r from the current state s and action a. Second, the agent uses this model to *plan* a sequence of actions that maximizes future rewards. This is analogous to a chess player who first learns the rules of movement and then uses that knowledge to plan several moves ahead.

1.2 The Core Component: The Dynamics Model

The foundation of MBRL is the **dynamics model** (or world model), which represents the agent's understanding of the environment.

- **Known Models:** In many scenarios, like games (Chess, Go) or simulations, the dynamics are perfectly known. The challenge is not learning but pure planning—using the known model to find an optimal strategy.
- **Learned Models:** When dynamics are unknown, the agent must learn them from data. This can be done via:
  - **System Identification:** Fitting parameters to a predefined model structure (e.g., learning a car's wheelbase for a kinematic model).
  - **General-Purpose Function Approximators:** Using complex models like Deep Neural Networks (e.g., RNNs, VAEs) to learn dynamics from high-dimensional data like pixels, without strong prior assumptions.

1.3 The Promise of Model-Based RL: Sample Efficiency and Planning

The main advantage of MBRL is its superior **sample efficiency**. By learning a model, an agent can generate countless "imaginary" experiences through internal simulation. This drastically reduces the need for real-world interactions, which can be expensive, time-consuming, or dangerous. This

is crucial in fields like robotics. Furthermore, a model enables **planning**—the ability to simulate the consequences of actions before execution, leading to more deliberate and far-sighted decisions.

1.4 The Challenge: The Reality Gap and Model Error

The greatest hurdle in MBRL is that the learned model is almost never perfect. This discrepancy, the **"reality gap,"** can lead to catastrophic failures as small inaccuracies compound over long prediction horizons. A planner might exploit a flaw in the model, finding a strategy that seems brilliant in simulation but fails in reality. This creates a fundamental trade-off:

- **MBRL:** Higher sample efficiency (learns faster) but potentially lower final performance due to being limited by the imperfect model (model bias).
- **MFRL:** Lower sample efficiency but potentially higher asymptotic performance, as it learns directly from the true environment and is not constrained by model bias.

Table 1: Comparative Analysis of Model-Based and Model-Free RL

Characteristic	**Model-Based RL (MBRL)**	**Model-Free RL (MFRL)**
Core Idea	Learns a world model, then plans.	Learns a policy/value function direct
Sample Efficiency	High. Learns from fewer real interactions.	Low. Requires many real interaction
Computational Cost	High during planning (simulation).	High during learning (gradient updat
Asymptotic Performance	Often lower, limited by model accuracy.	Potentially higher, no model bias.
Flexibility	High. Adapts to new goals by re-planning.	Low. Requires re-training for new ta
Key Challenge	Compounding model errors.	Inefficient exploration (sparse reward
Example Algorithms	Dyna-Q, PlaNet, Dreamer	Q-Learning, PPO, SAC

2 Planning and Control with a Known Model

Once a model is available, the task is to use it for planning.

2.1 Planning as Inference

Planning is the process of using a model to simulate future outcomes. The goal is to find an action sequence or policy that maximizes the expected cumulative reward over these simulated trajectories.

The Deterministic Case: The model is a simple function $s_{t+1} = f(s_t, a_t)$. The optimization problem is to find the action sequence that maximizes the sum of rewards:

$$\max_{a_1,\dots,a_T} \sum_{t=1}^{T} r(s_t, a_t) \quad \text{subject to} \quad s_{t+1} = f(s_t, a_t) \tag{2}$$

The Stochastic Case: The next state is drawn from a distribution, $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. The planner must optimize for the *expected* cumulative reward, which is often intractable to calculate exactly and requires sampling-based methods.

$$\max_{a_1,\dots,a_T} \mathbb{E}_{s_{t+1}\sim p(\cdot|s_t,a_t)} \left[\sum_{t=1}^{T} r(s_t, a_t) \right] \tag{3}$$

2.2 Open-Loop vs. Closed-Loop Control: A Critical Distinction

Open-Loop (Feedforward) Control determines a complete, fixed sequence of actions at the start and executes it without feedback. This approach is brittle and fails when the environment deviates from the model's predictions.

> *A Persian parable illustrates this weakness: A deaf man pre-plans a conversation with his sick friend. He asks, "How are you?" expecting the reply "Better," to which he will say "Thank God!" When he visits, the friend replies, "I am dying!" Sticking to his plan, the deaf man cheerfully responds, "Thank God!"*

This highlights how open-loop control is only optimal for the single future it was designed for.

Closed-Loop (Feedback) Control uses feedback at each step. It observes the current state s_t and uses this information to select the next action a_t. This makes the system robust and adaptive but also more complex. A thermostat is a classic example.

Model Predictive Control (MPC) bridges this gap by turning an open-loop planner into a closed-loop controller. It follows a simple, iterative "receding horizon" strategy:

1. From the current state s_t, use an open-loop planner to find an optimal action sequence for a finite horizon, $a_t, \ldots, a_{t+H}$.
2. Execute **only the first action**, a_t, in the real environment.
3. Observe the new state, s_{t+1}.
4. Discard the rest of the plan and repeat the process from the new state.

By constantly re-planning, MPC gains the robustness of closed-loop control while using powerful open-loop planning algorithms.

3 Planning as Stochastic Optimization

3.1 Formulating the Objective

The planning problem can be framed as finding an action sequence $A = (a_1, \ldots, a_T)$ that maximizes an objective function $J(A)$, representing the expected cumulative reward:

$$A^* = \arg\max_A J(A) \quad \text{where} \quad J(A) = \mathbb{E}\left[\sum_{t=1}^{T} r(s_t, a_t)\right] \tag{4}$$

Since $J(A)$ cannot be evaluated analytically, we rely on noisy estimates from simulations, turning this into a **stochastic optimization** problem.

3.2 The "Random Shooting" Method: A Simple Baseline

Because we often lack the gradient of $J(A)$, we use derivative-free or "black-box" optimization methods. The simplest is **random shooting**:

1. **Sample:** Generate N candidate action sequences, $A_1, \ldots, A_N$, from a fixed distribution (e.g., Gaussian).

2. **Evaluate:** For each sequence A_i, estimate its value $J(A_i)$ by running one or more simulations (rollouts) and averaging the rewards.

3. **Select:** Choose the action sequence with the highest estimated value.

This "guess and check" method is simple but highly inefficient due to the curse of dimensionality, motivating more intelligent search strategies.

4 The Cross-Entropy Method (CEM) for Planning

The Cross-Entropy Method (CEM) is a powerful stochastic optimization algorithm that improves upon random shooting by iteratively refining a sampling distribution to focus on high-reward regions.

4.1 The Core Idea: Iterative Refinement via Elite Sampling

CEM is an evolutionary algorithm that learns from a population of high-performing samples, the "elites." At each iteration, it fits a new probability distribution to this elite set, guiding the next round of sampling to be "more like" the successful samples from the previous one.

4.2 The CEM Algorithm in Detail

For continuous actions, CEM proceeds as follows:

1. **Initialize:** Define a parameterized probability distribution over action sequences, typically a multivariate Gaussian $p(A|\theta)$ with parameters $\theta = (\mu, \Sigma)$. Initialize μ to zero and Σ to the identity matrix.

2. **Sample:** Draw N candidate action sequences, $A_1, \ldots, A_N$, from the current distribution $p(A|\theta)$.

3. **Evaluate:** Compute the total reward $J(A_i)$ for each sequence.

4. **Select Elites:** Sort the sequences by reward and select the top fraction (e.g., top 10%) as the "elite set."

5. **Refit:** Update the distribution parameters θ to best fit the elite set using Maximum Likelihood Estimation (MLE).

6. **Iterate:** Repeat steps 2-5 until convergence.

4.3 Mathematical Formulation and Update Rules

For a Gaussian distribution, the MLE update for the elite set $\{A_{\text{elite}}\}$ has a simple closed-form solution.

Mean Update: The new mean $\hat{\mu}_{\text{new}}$ is the sample mean of the elite action sequences:

$$\hat{\mu}_{\text{new}} = \frac{1}{M} \sum_{j=1}^{M} A_j \tag{5}$$

where M is the number of elites.

Covariance Update: The new covariance $\hat{\Sigma}_{\text{new}}$ is the sample covariance of the elite action sequences:

$$\hat{\Sigma}_{\text{new}} = \frac{1}{M}\sum_{j=1}^{M}(A_j - \hat{\mu}_{\text{new}})(A_j - \hat{\mu}_{\text{new}})^T \tag{6}$$

For stability, a diagonal covariance is often used, or smoothing is applied to prevent the variance from collapsing too quickly.

5 Monte Carlo Tree Search (MCTS) for Planning

MCTS is a heuristic search algorithm that incrementally builds a search tree, intelligently focusing on the most promising action sequences. It is especially effective for problems with discrete action spaces, like board games.

5.1 The Four Steps of MCTS

Each MCTS iteration involves a four-step loop:

1. **Selection:** Traverse the tree from the root, selecting child nodes based on a policy that balances exploration and exploitation (e.g., UCB1), until a leaf node is reached.
2. **Expansion:** If the leaf node is not terminal, expand it by creating new child nodes for unexplored actions.
3. **Simulation (Rollout):** From a new node, run a simulation (e.g., with a random policy) to a terminal state to get a value estimate.
4. **Backpropagation:** Propagate the simulation result back up the tree, updating the visit count and value of each node along the selection path.

The final best action is typically the one corresponding to the most visited child of the root.

5.2 The Exploration-Exploitation Dilemma in MCTS

The intelligence of MCTS lies in its selection step, often guided by the **Upper Confidence Bound 1 (UCB1)** formula applied to trees (UCT):

$$\text{UCB1}(\text{node}_i) = \underbrace{\frac{V_i}{n_i}}_{\text{Exploitation}} + \underbrace{C\sqrt{\frac{\ln N}{n_i}}}_{\text{Exploration}} \tag{7}$$

where V_i is the total value of node i, n_i is its visit count, N is the visit count of its parent, and C is an exploration constant. The first term favors historically good moves, while the second encourages trying less-visited moves.

5.3 Integrating MCTS with Learned Models: The AlphaGo/MuZero Approach

Modern MCTS (e.g., AlphaZero, MuZero) integrates learned neural networks:

- A **value network** $v_\theta(s)$ provides a better estimate of a state's value than a random rollout.
- A **policy network** $\pi_\theta(a|s)$ provides prior probabilities for promising moves, guiding the search.

This creates a powerful cycle: MCTS search improves the policy and value estimates, which are then used as training targets to improve the networks. The improved networks then conduct an even stronger search.

6 A General Framework: Trajectory Optimization

Trajectory Optimization (TO) is a formal discipline from optimal control for computing a time-indexed history of states and controls that minimizes an objective while satisfying constraints. It finds an open-loop solution that can be implemented in a closed-loop fashion using MPC.

6.1 Key Components of a TO Problem

- **Cost Function:** The objective to be minimized (e.g., energy, time).
- **Dynamics Constraints:** The differential equations governing the system's motion.
- **Path Constraints:** Inequalities that must hold (e.g., joint limits, obstacle avoidance).
- **Boundary Conditions:** Equality constraints on the start and end states.

6.2 Major Approaches: Direct vs. Indirect

- **Direct Methods:** Transcribe the problem into a finite-dimensional nonlinear program (NLP) by discretizing the trajectory.
  - **Shooting Methods:** Optimize control inputs and integrate dynamics forward.
  - **Direct Collocation:** Represent trajectories as splines and enforce dynamics as algebraic constraints at collocation points.
- **Indirect Methods:** Use calculus of variations to derive optimality conditions, transforming the problem into a boundary value problem. These methods are fast and accurate but difficult to use.

7 Synthesis: A Unified View of Planning Algorithms

7.1 Connecting the Dots: A Conceptual Map

The discussed algorithms exist on a spectrum within optimal control:

- **Trajectory Optimization (TO):** The overarching mathematical framework.
- **Direct Methods:** A class of TO algorithms.
- **Shooting Methods:** A specific type of direct method.
- **Random Shooting & CEM:** Basic and advanced iterative shooting methods, respectively.
- **MCTS:** An alternative heuristic tree search approach, well-suited for discrete optimal control problems.

7.2 The Receding Horizon Strategy (MPC): A Unifying Principle

The recurring theme is that core planners like Shooting, CEM, and MCTS are open-loop. **Model Predictive Control (MPC)** is the essential technique for deploying them in the real world, creating a robust, closed-loop system by constantly re-planning from the current state.

Table 2: Comprehensive Comparison of Planning Algorithms

Characteristic	Random Shooting	Cross-Entropy Method (CEM)	Monte Carlo Tree Search (MCTS)	Trajectory Optimization (General)
Basic Idea	Randomly guess and check.	Iteratively fit a distribution to elites.	Build an asymmetric search tree.	Solve a constrained optimization problem.
Control Type	Open-Loop Planner	Open-Loop Planner	Open-Loop Planner	Open-Loop Planner
Action Space	Continuous / Discrete	Primarily Continuous	Primarily Discrete	Primarily Continuous
Key Strength	Extreme simplicity.	Simple, effective, parallelizable.	Principled exploration, "anytime."	Formal, powerful, handles constraints.
Key Weakness	Very inefficient.	Struggles in high dimensions.	Hard for continuous actions.	Complex setup, local minima.
Requires MPC?	Yes	Yes	Yes	Yes
Application	Baselines, simple problems.	Moderate-dim continuous control.	Games, discrete sequential problems.	Robotics, aerospace, process control.

7.3 Future Directions: The Frontier of Model-Based RL

- **Differentiable Trajectory Optimization:** Developing end-to-end differentiable planners to learn model and cost function parameters by backpropagating task loss through the planner.
- **Uncertainty-Aware Planning:** Explicitly accounting for model uncertainty during planning to enable safer and more efficient exploration in unknown environments.
- **Hybrid Approaches:** Combining the strengths of different methods, such as using a global search (CEM) to initialize a powerful local optimizer.