RL as an Adaptive Optimal Control

The Evolution of Decision-Making Under Uncertainty

Zirui Zhang

Cheng Kar-Shun Robotics Institute
The Hong Kong University of Science and Technology

September 21, 2025

Problem Formulation

Consider the deterministic discrete-time optimal control problem:

$$\begin{aligned} \min_{x_{1:N},\,u_{1:N-1}} \; & \sum_{n=1}^{N-1} l(x_n, u_n) + l_F(x_N) \\ \text{s.t.} \;\; & x_{n+1} = f(x_n, u_n) \\ & u_n \in \mathcal{U} \end{aligned}$$

The first-order necessary conditions for optimality can be derived using:

- The Lagrangian framework (special case of KKT conditions)
- Pontryagin's Minimum Principle (PMP)

Lagrangian Formulation

Form the Lagrangian:

$$L = \sum_{n=1}^{N-1} l(x_n, u_n) + \lambda_{n+1}^\top (f(x_n, u_n) - x_{n+1}) + l_F(x_N)$$

Define the **Hamiltonian**:

$$H(x_n, u_n, \lambda_{n+1}) = l(x_n, u_n) + \lambda_{n+1}^\top f(x_n, u_n)$$

Rewrite the Lagrangian using the Hamiltonian:

$$L = H(x_1, u_1, \lambda_2) + \left[\sum_{n=2}^{N-1} H(x_n, u_n, \lambda_{n+1}) - \lambda_n^\top x_n \right] + l_F(x_N) - \lambda_N^\top x_N$$

Optimality Conditions

Take derivatives with respect to x and λ:

$$\frac{\partial L}{\partial \lambda_n} = \frac{\partial H}{\partial \lambda_n} - x_{n+1} = f(x_n, u_n) - x_{n+1} = 0$$
$$\frac{\partial L}{\partial x_n} = \frac{\partial H}{\partial x_n} - \lambda_n^\top = \frac{\partial l}{\partial x_n} + \lambda_{n+1}^\top \frac{\partial f}{\partial x_n} - \lambda_n^\top = 0$$
$$\frac{\partial L}{\partial x_N} = \frac{\partial l_F}{\partial x_N} - \lambda_N^\top = 0$$

For u, we write the minimization explicitly to handle constraints:

$$u_n = \arg\min_u H(x_n, u, \lambda_{n+1})$$
$$\text{s.t. } u \in \mathcal{U}$$

Summary of Necessary Conditions

The first-order necessary conditions can be summarized as:

$$
\begin{aligned}
x_{n+1} &= \nabla_\lambda H(x_n, u_n, \lambda_{n+1}) \\
\lambda_n &= \nabla_x H(x_n, u_n, \lambda_{n+1}) \\
u_n &= \arg\min_u H(x_n, u, \lambda_{n+1}), \quad \text{s.t. } u \in \mathscr{U} \\
\lambda_N &= \frac{\partial l_F}{\partial x_N}
\end{aligned}
$$

In continuous time, these become:

$$
\begin{aligned}
\dot{x} &= \nabla_\lambda H(x, u, \lambda) \\
-\dot{\lambda} &= \nabla_x H(x, u, \lambda) \\
u &= \arg\min_{\tilde{u}} H(x, \tilde{u}, \lambda), \quad \text{s.t. } \tilde{u} \in \mathscr{U} \\
\lambda(t_F) &= \frac{\partial l_F}{\partial x}
\end{aligned}
$$

Application to LQR Problems

For LQR problems with quadratic cost and linear dynamics:

$$
\begin{aligned}
l(x_n, u_n) &= \frac{1}{2}(x_n^\top Q_n x_n + u_n^\top R_n u_n) \\
l_F(x_N) &= \frac{1}{2} x_N^\top Q_N x_N \\
f(x_n, u_n) &= A_n x_n + B_n u_n
\end{aligned}
$$

The necessary conditions simplify to:

$$
\begin{aligned}
x_{n+1} &= A_n x_n + B_n u_n \\
\lambda_n &= Q_n x_n + A_n^\top \lambda_{n+1} \\
\lambda_N &= Q_N x_N \\
u_n &= -R_n^{-1} B_n^\top \lambda_{n+1}
\end{aligned}
$$

This forms a linear two-point boundary value problem.

Bridging Optimal Control and RL

Markov Chains

- State space $\mathscr{X}$
- Action space $\mathscr{U}$
- System dynamics $f(x_n, u_n)$
- Cost function $l(x,u)$ and $l_F(x)$

Find feedback $\boldsymbol{u} = \boldsymbol{K}(\boldsymbol{x})$ to minimize $J(x_0, u)$

$$J(x_0, u) = \mathbb{E}\left[\sum_{n=0}^{N-1} l(x_n, u_n) + l_F(x_N)\right]$$

subject to $x_{n+1} \sim f(x_n, u_n)$.

Markov (Decision) Process

- State space $\mathscr{S}$
- Action space $\mathscr{A}$
- Transition dynamics $P(s'|s,a)$
- Reward function $R(s,a,s')$

Find policy $\boldsymbol{\pi}(\boldsymbol{a}|\boldsymbol{s})$ to maximize $V(s_0, \pi)$

$$V(s_0, \pi) = \mathbb{E}\left[\sum_{n=0}^{H} \gamma^n R(s_n, a_n, s_{n+1})\right]$$

subject to $s_{n+1} \sim P(\cdot|s_n, a_n)$, $a_n \sim \pi(\cdot|s_n)$, where $\gamma \in [0,1)$ is the discount factor.

*RL is an **adaptive method** to solve MDP in the absence of model knowledge.*

Value Function and Action-Value Function

Optimal Control:

- **Value Function:**

$$V(x) = \min_u \mathbb{E}\left[\sum_{n=0}^{N-1} l(x_n, u_n) + l_F(x_N) \,\middle|\, x_0 = x\right]$$

- **Action-Value Function:**

$$Q(x,u) = \mathbb{E}\left[l(x,u) + V(x') \middle| x' \sim f(x,u)\right]$$

Reinforcement Learning:

- **Value Function:**

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{n=0}^{H} \gamma^n R(s_n, a_n, s_{n+1}) \,\middle|\, s_0 = s, a_n \sim \pi(\cdot|s_n)\right]$$

- **Action-Value Function:**

$$Q^{\pi}(s,a) = \mathbb{E}\left[R(s,a,s') + \gamma V^{\pi}(s') \,\middle|\, s' \sim P(\cdot|s,a)\right]$$

The Scalability Challenge

For discrete, low-dimensional problems with a known model, Optimal Control and Model-based RL can be solved exactly using Dynamic Programming (DP). **But what if...**

- The model $f(x,u)$, $l(x,u)$ or $P(s'|s,a)$, $R(s,a,s')$ is **unknown**?
- The state or action space is too **large** or **continuous** making DP loops intractable?
- The system is **too complex** to model accurately?

$$\Downarrow$$

*We need **model-free**, **stochastic**, and **approximate** methods.*

$$\Downarrow$$

This is the core domain of modern Reinforcement Learning.

(Tabular) Q-Learning

(Tabular) Q-Learning **replace expectation by samples**:

- For an state-action pair (s, a) , receive $s' \sim P(s'|s, a)$
- Consider old estimate $Q_k(s, a)$
- Consider new sample estimate: $\text{target}(s') = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
- Incorporate the new estimate into a running average:

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha [\text{target}(s')]$$

- Q-learning converges to optimal policy even if you're acting suboptimally and is called off-policy learning.
- Requires sufficient exploration and a learning rate α that decays appropriately:

$$\sum_{t=0}^{\infty} \alpha_t(s, a) = \infty \quad \sum_{t=0}^{\infty} \alpha_t^2(s, a) < \infty$$

Courtesy of Pieter Abbeel

Approximate Q-Learning

Instead of a table, we use a **parametrized Q function** $Q_\theta(s,a)$ to approximate:

- Learning rule:

$$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_{\theta_k}(s',a')$$

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2}(Q_\theta(s,a) - \text{target}(s'))^2 \right] \bigg|_{\theta=\theta_k}$$

- Practical details:
  - Use Huber loss instead of squared loss on Bellman error:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \le \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

  - Use RMSProp instead of vanilla SGD.
  - It is beneficial to **anneal** the exploration rate over time.

Courtesy of Pieter Abbeel

RL as an Adaptive Optimal Control

Common Core:

- Value Function
- Bellman Equation
- Sequential Decision Making

The Evolution:

- Expectation → Samples
- Table → Function Approximation
- Exact Solution → Stochastic Optimization

A powerful and adaptive optimal control framework.

Thank you for listening !

Zirui Zhang