

Personalized Education in the Era of AI I

Recap on Control, Reinforcement Learning, Game Theory

Ruixiang Wu & Costas Courcoubetis

October 30, 2025

School of Data Science

Table of contents

1. Recap: Linear Control
2. Recap: Reinforcement Learning
3. Recap: Game Theory

Recap: Linear Control

Classic Linear Quadratic Control: Problem Formulation

Consider a system with state $x_t \in \mathbb{R}^d$ and control $u_t \in \mathbb{R}^k$. The dynamics are given by:

$$x_{t+1} = Ax_t + Bu_t,$$

The incurred cost is:

$$x_t^\top Qx_t + u_t^\top Ru_t,$$

for positive definite matrices Q and R .

A policy $\pi : \mathbb{R}^d \mapsto \mathbb{R}^k$ maps the current state to a control action. The cost of a policy over T steps is:

$$J_T(\pi) = \sum_{t=1}^{T-1} x_t^\top Qx_t + u_t^\top Ru_t + x_T^\top Q_T x_T,$$

The solution is a linear feedback policy:

$$u_t = -Kx_t$$

Linear Feedback Policy: An Informal Proof

We see the problem through the lens of Dynamic Programming:

$$J_T = x_T^\top Q_T x_T \quad (1a)$$

$$J_t(x_t) = \min_{u_t} \{x_t^\top Q x_t + u_t^\top R u_t + J_{t+1}(A x_t + B u_t)\} \quad (1b)$$

We write Eq. 1b for $t = T - 1$.

$$\begin{aligned} J_{T-1}(x_{T-1}) = \min_{u_{T-1}} \{ & x_{T-1}^\top Q x_{T-1} + u_{T-1}^\top R u_{T-1} \\ & + (A x_{T-1} + B u_{T-1})^\top Q_T (A x_{T-1} + B u_{T-1}) \} \end{aligned} \quad (2)$$

By differentiating with respect to u_{T-1} and by setting the derivative equal to zero, we obtain:

$$u_{T-1}^* = -(B^\top Q_T B + R)^{-1} B^\top Q_T A x_{T-1}$$

Plug it into Eq. 2, we have:

$$J_{T-1}(x_{T-1}) = x_{T-1}^\top K_{T-1} x_{T-1},$$

where:

$$K_{T-1} = A^\top (Q_T - Q_T B (B^\top Q_T B + R)^{-1} B^\top Q_T) A + Q$$

Classic Model Predictive Control: Problem Formulation

MPC uses a model to predict the system's future evolution and solve an online optimization problem at each time step.

Core Idea: Receding Horizon Control At each time step t , given the current state x_t :

1. **Optimize:** Solve a finite-horizon optimal control problem over a prediction horizon H :

$$\begin{aligned} \min_{u_{t|t}, \dots, u_{t+H-1|t}} \quad & \sum_{k=0}^{H-1} (x_{t+k|t}^\top Q x_{t+k|t} + u_{t+k|t}^\top R u_{t+k|t}) \\ \text{s.t.} \quad & x_{t+k+1|t} = A x_{t+k|t} + B u_{t+k|t} \\ & x_{t|t} = x_t \quad (\text{current state}) \end{aligned}$$

2. **Apply:** Apply **only the first** optimal control action:

$$u_t = u_{t|t}^*$$

Key Advantage: Unlike the LQR's pre-computed (offline) policy, MPC can explicitly handle state and control constraints.

- **Video Series:**

- Steve Brunton's "Control Bootcamp" series on YouTube.
- Click: [Link to the playlist](#)

- **Textbook:**

- "Dynamic programming and optimal control" [1]
- **Chap 1:** Dynamic Programming Algorithms.
- **Chap 3:** Continuous-Time Optimal Control.
- **Chap 4:** Discrete-Time Optimal Control.
- **Chap 5:** State Estimation

Recap: Reinforcement Learning

Introduction to Markov Decision Processes (MDPs)

An MDP is formally defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- \mathcal{S} : **State Space**: A finite set of states representing the different situations an agent can be in. (e.g., current position on a board game, specific image on a screen.)
- \mathcal{A} : **Action Space**: A finite set of actions available to the agent in each state. (e.g., move left, jump, attack.)
- $P(s'|s, a)$: **Transition Probability Function**: The probability of transitioning to state s' from state s after taking action a . (Defines the dynamics of the environment.)
- $R(s, a, s')$: **Reward Function**: The immediate scalar reward received after transitioning from state s to s' by taking action a .
- γ : **Discount Factor**: A value between 0 and 1 that determines the present value of future rewards. A higher γ means future rewards are considered more important.

A policy π is a mapping from \mathcal{S} to \mathcal{A} . An optimal policy π^* minimize the discounted sum over (possibly) infinite horizon: $\mathbb{E} [\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})]$

Model Based Reinforcement Learning i

Key point of “Model Based”: Know $P(s', s, a)$, and $R(s, a)$. (i.e. Assume that the MDP model is known.)

A Model Based RL Algorithm: Policy Iteration: Policy Iteration is a classic model-based algorithm in reinforcement learning used to find an optimal policy π^* .

The Core Idea: It finds the optimal policy by iteratively repeating two simple steps:

1. **Policy Evaluation:** Determine how effective the current policy is.
2. **Policy Improvement:** Use that knowledge to improve the policy.

This process is repeated until the policy no longer improves, at which point it has converged to the optimal policy.

Model Based Reinforcement Learning ii

Step 1: Policy Evaluation

Goal: Given a fixed, deterministic policy π , calculate the state-value function $V^\pi(s)$ for all states s .

This step answers the question: "What is the expected long-term reward if we follow policy π from each state?"

To find V^π , we solve the **Bellman Equation** for the policy π . This equation expresses the value of a state as the expected immediate reward plus the discounted value of the next state.

$$V^\pi(s) = \sum_{s'} P(s'|s, \pi(s)) (R(s, \pi(s), s') + \gamma V^\pi(s'))$$

For a finite number of states, this is a system of linear equations that can be solved for V^π .

Model Based Reinforcement Learning iii

Step 2: Policy Improvement

Goal: Given the value function V^π for a policy π , find a better policy π' .

We improve the policy by acting greedily with respect to the value function V^π .

For each state s , we update the policy by choosing the action that maximizes the expected value, looking one step ahead:

$$\pi'(s) \leftarrow \arg \max_{a \in \mathcal{A}} \left\{ \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V^\pi(s')) \right\}$$

This new greedy policy π' is guaranteed to be an improvement over (or equal to) the old policy π . That is, for all states s :

$$V^{\pi'}(s) \geq V^\pi(s)$$

Policy Iteration Algorithm

1. **Initialization:** Start with an arbitrary policy π_0 .

2. **Iteration:** For $k = 0, 1, 2, \dots$

2.1 **Policy Evaluation:** Compute the value function for the current policy π_k .

$$V_k \leftarrow V^{\pi_k}$$

2.2 **Policy Improvement:** Create a new greedy policy π_{k+1} using V_k .

$$\pi_{k+1}(s) \leftarrow \arg \max_a \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V_k(s'))$$

2.3 **Check for Convergence:** If $\pi_{k+1}(s) = \pi_k(s)$ for all states s , then stop.

3. **Termination:** Return the final policy π_{k+1} and value function V_k .

Model Based Reinforcement Learning v

Another Model Based RL Algorithm: Value Iteration.:

The Core Idea: Instead of iterating on the policy, Value Iteration iteratively improves the **value function** until it converges to the optimal value function, V^* . The optimal policy is then extracted just once at the very end.

The algorithm finds the optimal value function by repeatedly applying the **Bellman Optimality Equation** as an update rule:

The Update Rule

Starting with an initial value function V_0 (e.g., all zeros), each iteration $k + 1$ updates the value of every state s using the values from the previous iteration V_k :

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma V_k(s'))$$

Model Free Reinforcement Learning i

“Model Free”: MDP is unknown, rely on sampling strategies to learn.

A simple model free RL algorithm is Temporal Difference learning, where the value function is updated using the estimate from one step ahead.

TD Target (R): The value we are trying to move our estimate towards. It's the immediate reward plus the discounted value of the *next* state.

$$R = r_k + \gamma V_{old}(s_{k+1})$$

TD Error (δ_k): The difference between the TD Target and our old estimate.

$$\delta_k = R - V_{old}(s_k) = [r_k + \gamma V_{old}(s_{k+1}) - V_{old}(s_k)]$$

TD(0) Update Rule: We update the old value by moving it a small step (α) in the direction of the TD Error.

$$V_{new}(s_k) \leftarrow V_{old}(s_k) + \alpha \cdot \delta_k$$

Model Free Reinforcement Learning ii

TD(n): n-Step Look Ahead

Generalizes TD(0) by looking $n + 1$ steps into the future to form the target.

- **TD(1) Target:** $R^{(1)} = r_k + \gamma r_{k+1} + \gamma^2 V(s_{k+2})$
- **TD(2) Target:** $R^{(2)} = r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + \gamma^3 V(s_{k+3})$

General n-Step TD Target ($R^{(n)}$)

$$R^{(n)} = r_k + \gamma r_{k+1} + \cdots + \gamma^n r_{k+n} + \gamma^{n+1} V_{old}(s_{k+n+1})$$

$$R^{(n)} = \sum_{j=0}^n \gamma^j r_{k+j} + \gamma^{n+1} V_{old}(s_{k+n+1})$$

TD(n) Update Rule

$$V_{new}(s_k) \leftarrow V_{old}(s_k) + \alpha [R^{(n)} - V_{old}(s_k)]$$

Model Free Reinforcement Learning iii

A representative example of Model Free RL is Q Learning, which directly learns the **optimal action-value function**, denoted q_* , regardless of the policy being followed for exploration.

The update rule is defined as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[\underbrace{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)}_{\text{TD Target: Best possible next value}} - \underbrace{Q(S_t, A_t)}_{\text{Old value}} \right]$$

Difference between on-policy and off-policy?

Deep Reinforcement Learning i

Key point of “Deep”: Using a Deep Neural Network to approximate the value function or quality function.

Policy Gradient Methods

The Core Idea

Policy Gradient methods optimize a **parameterized policy** $\pi_{\theta}(s, a)$ directly. The parameters θ could be the weights of a neural network.

- The goal is to adjust θ to maximize the total expected reward, $J(\theta)$.
- We do this by performing **gradient ascent** on the policy parameters.
- This approach is often more effective in high-dimensional or continuous action spaces.

Deep Reinforcement Learning ii

The Policy Gradient Update The update is guided by the **Policy Gradient Theorem**, which provides an expression for the gradient of the expected reward.

Policy Gradient Theorem

The gradient of the objective function $J(\theta)$ is given by the expectation:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} [Q^{\pi_{\theta}}(s, a) \nabla_{\theta} \log \pi_{\theta}(s, a)]$$

Want to see the proof? $J(\theta) = \sum_{s \in S} P(s) \sum_{a \in A} Q(s, a) \pi_{\theta}(s, a)$

The Update Rule

The policy parameters are then updated via gradient ascent:

$$\theta_{\text{new}} \leftarrow \theta_{\text{old}} + \alpha \nabla_{\theta} J(\theta)$$

where α is the learning rate.

The Challenge for Classic RL: Representation

Classic RL methods often use tables to store values for each state (e.g., Q-tables). This is impossible for problems with huge state spaces, like video games, where a single screen has more states than atoms in the universe!

The Solution: Deep RL

Combine the power of **deep learning** for function approximation with the decision-making framework of **reinforcement learning**.

We use deep neural networks to approximate the key RL functions:

- **Policy:** $\pi(s, a) \approx \pi(s, a, \theta)$
- **Value Function:** $V(s) \approx V(s, \theta)$
- **Q-Function:** $Q(s, a) \approx Q(s, a, \theta)$

Deep Reinforcement Learning iv

Deep Q-Networks (DQN): DQN uses a **deep neural network** to approximate the Q-function: $Q(s, a) \approx Q(s, a, \theta)$.

How It Works

Instead of updating a table, DQN trains the network weights θ by minimizing a loss function based on the TD error from Q-learning:

- The network takes the state s (e.g., screen pixels) as input and outputs Q-values for all possible actions.
- The loss function to minimize is the squared difference between the TD target and the network's prediction.

DQN Loss Function

$$L(\theta) = \mathbb{E} \left[\left(\underbrace{R + \gamma \max_{a'} Q(S', a', \theta_{\text{target}})}_{\text{TD Target}} - \underbrace{Q(S, a, \theta)}_{\text{Prediction}} \right)^2 \right]$$

Reinforcement Learning Overview

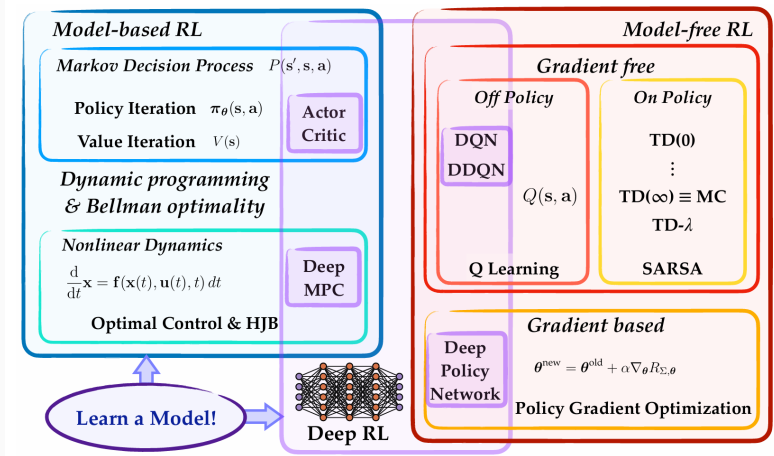


Figure 1: Overview of the World of RL.

Reinforcement Learning Resources

- **Video Series:**

- Steve Brunton's "Reinforcement Learning" series on YouTube.
- Click: [Link to the playlist](#)

- **Textbook:**

- "Reinforcement Learning: An Introduction" [3]
- **Chap 3:** Basic MDP formulation.
- **Chap 4:** DP formulation.
- **Chap 6:** Classic RL algorithms (TD learning, Q learning).
- **Chap 13:** Policy gradient method.

- **Survey Paper:**

- *Comprehensive Survey of Reinforcement Learning: From Algorithms to Practical Challenges* [2]

Recap: Game Theory

Key Concepts in Game Theory

- **Players:** The decision makers within the game.
- **Strategies:** The plans or actions that players can take.
- **Payoffs:** The outcomes or rewards players receive as a result of the strategies chosen.
- **Nash Equilibrium:**
 - A Nash Equilibrium occurs when each player's strategy is optimal given the strategies of all other players.
 - Formally, a strategy profile $(s_1^*, s_2^*, \dots, s_n^*)$ is a Nash Equilibrium if no unilateral deviation is beneficial for any player, i.e., for each player i ,



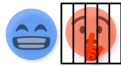

$$u_i(s_i^*, s_{-i}^*) \geq u_i(s_i, s_{-i}^*), \forall s_i \in S_i,$$

where u_i is the payoff function for player i , s_i is player i 's strategy, S_i is the strategy set for player i , and s_{-i}^* represents the strategy profile of all players except for player i .

The Prisoner's Dilemma

Scenario:

- Two members of a criminal gang are arrested and imprisoned.
- Each prisoner is in solitary confinement with no means of communicating with the other.
- The prosecutors lack sufficient evidence to convict on the principal charge, but they have enough to convict on a lesser charge.
- Prisoners are given the opportunity to betray each other or to remain silent.

A \ B	B	
	B stays silent	B testifies
A stays silent	 R:-1 R:-1	 S:-3 T:0
A testifies	 T:0 S:-3	 P:-2 P:-2

The Prisoner's Dilemma (Continued)

Nash Equilibrium:

- Each prisoner has two strategies: to “testify” or to “stay silent”.
- The Nash Equilibrium occurs when each player’s strategy is optimal, given the other’s strategy.
- In the Prisoner’s Dilemma, the Nash Equilibrium is for both prisoners to betray each other, as betrayal provides a better outcome regardless of the other’s decision.
- Despite mutual cooperation yielding a better collective outcome, the individual incentive to defect leads to a worse overall outcome.

- **Video Series:**

- Game theory video series on YouTube.
- Click: [Link to the playlist](#)



D. Bertsekas.

Dynamic programming and optimal control: Volume I, volume 4.

Athena scientific, 2012.



M. Ghasemi, A. H. Moosavi, and D. Ebrahimi.

A comprehensive survey of reinforcement learning: From algorithms to practical challenges, 2025.



R. S. Sutton and A. G. Barto.

Reinforcement Learning: An Introduction.

A Bradford Book, Cambridge, MA, USA, 2018.