

# Lecture 12: Reinforcement Learning

Yasuyuki Sawada, Yaolang Zhong

University of Tokyo  
yaolang.zhong@e.u-tokyo.ac.jp

January 17, 2026

## Recap: Motivation for Machine Learning

- ▶ Dynamic programming faces the curse of dimensionality
  - ▶ Parameterization problem
    - ▶ Classical functional forms (linear, polynomials, splines)
    - ▶ Number of basis terms grows rapidly with state dimension
    - ▶ Strong restrictions and poor capture of interactions
  - ▶ Grid / data problem
    - ▶ Grid-based DP requires  $N^d$  state points
    - ▶ Computation and memory grow exponentially
    - ▶ Sparse grids mitigate but do not eliminate the problem

# How Machine Learning Addresses the Curse of Dimensionality

- ▶ Machine learning relaxes both constraints
  - ▶ Neural networks for parameterization
    - ▶ Flexible, high-dimensional function approximation
    - ▶ Captures nonlinearities and interactions
  - ▶ Stochastic simulation for data collection
    - ▶ Learn from simulated trajectories, not full grids
    - ▶ Scales to high-dimensional state spaces

# Overview

## 1. Network Architecture

- ▶ Actor: policy network  $\sigma(s; \theta)$  maps states to actions
- ▶ Critic: value network  $Q(s, a; \phi)$  evaluates state-action pairs

## 2. Forward Pass

- ▶ Batch of states  $S \in \mathbb{R}^{N \times D_s}$
- ▶ Actor produces actions; critic produces value estimates

## 3. Objective Functions

- ▶ Actor: maximize expected return via policy gradient
- ▶ Critic: minimize Bellman residual (temporal difference error)

## 4. Backward Pass and Parameter Updates

- ▶ Compute gradients  $\nabla_{\theta}\mathcal{L}_{\theta}$  and  $\nabla_{\phi}\mathcal{L}_{\phi}$
- ▶ Update parameters via gradient descent

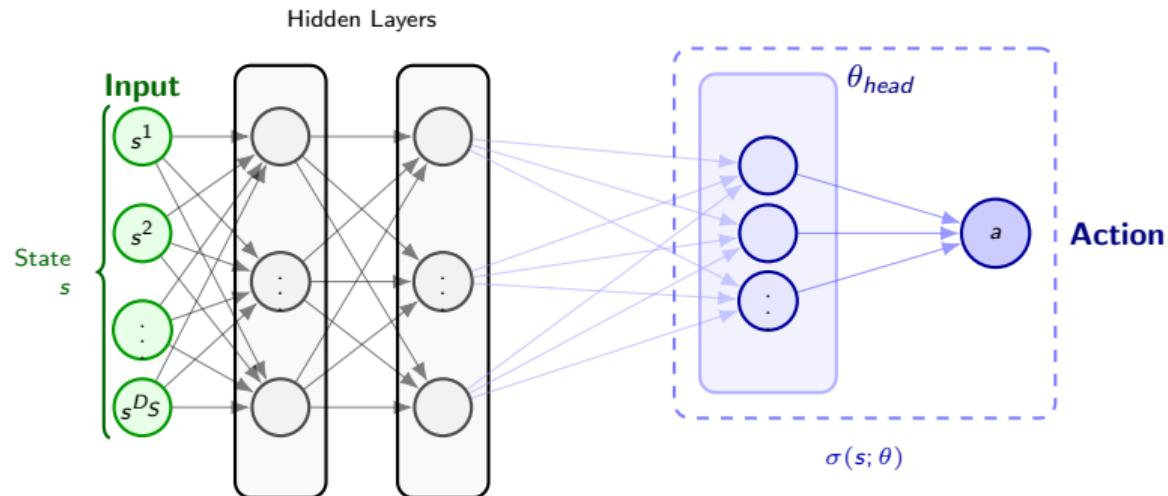
## 5. Implementation Caveats

- ▶ Target networks for training stability
- ▶ Gradient detachment to separate actor and critic updates

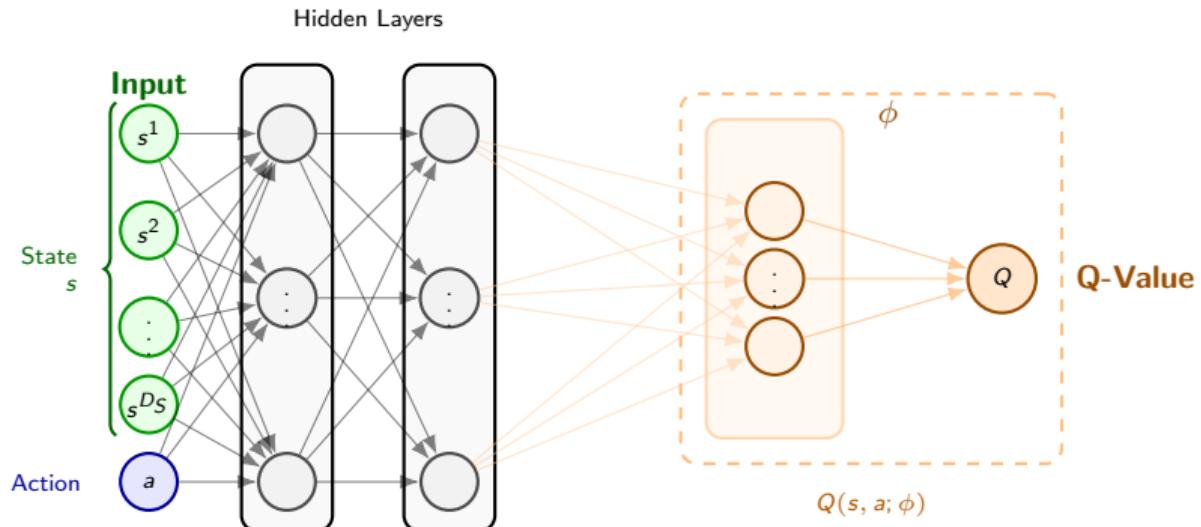
## Actor–Critic Methods: Introduction

- ▶ Actor–critic methods are a general framework for reinforcement learning (RL) problem
- ▶ Particularly suited for Large or continuous state spaces and action spaces
- ▶ The framework decomposes learning into policy learning (actor) and value learning (critic)
  - ▶ Actor: a parameterized policy  $\sigma(a | s; \theta)$  that selects actions
  - ▶ Critic: a parameterized value function  $Q(s, a; \phi)$  that provides an evaluation signal guiding policy improvement
  - ▶ The notation  $\sigma(a | s; \theta)$  denotes a (possibly stochastic) policy: the probability of choosing action  $a$  in state  $s$  given parameters  $\theta$ ; deterministic policies are a special case that assign probability one to a single action (often written simply as  $\sigma(s)$ )

# Actor Network Architecture



# Critic Network Architecture



## Forward Pass

- ▶ Suppose a batch of states available:  $\{s_i\}_{i=1}^N$  where  $s_i \in \mathbb{R}^{D_S}$ , stacked as  $S \in \mathbb{R}^{N \times D_S}$
- ▶ Two neural networks:
  - ▶ Actor:  $\sigma(s; \theta)$
  - ▶ Critic:  $Q(s, a; \phi)$
- ▶ Forward Pass:

$$\begin{array}{ccc} S \in \mathbb{R}^{N \times D_S} & \xrightarrow{\theta} & A \in \mathbb{R}^{N \times D_A} \\ (S, A) \in \mathbb{R}^{N \times (D_S + D_A)} & \xrightarrow{\phi} & Q \in \mathbb{R}^N \end{array}$$

- ▶ Training vs. evaluation:
  - ▶ In *training mode*, the forward pass constructs a computational graph and records gradients with respect to the network parameters
  - ▶ In *evaluation mode*, the same forward mappings are applied, but no gradients are recorded

## Backward Propagation: Intuition

- ▶ In training mode, once a scalar loss  $\mathcal{L}$  is defined, the gradient  $\nabla_{\theta}\mathcal{L}$  is computed automatically by back-propagation in any ML package.
- ▶ Backpropagation applies the chain rule through the computational graph constructed in the forward pass
- ▶ The actor and critic would be updated via gradient descent (although in practice, more stochastic methods are used) as:

$$\theta \leftarrow \theta - \alpha_{\theta} \nabla_{\theta} \mathcal{L}_{\theta}$$

$$\phi \leftarrow \phi - \alpha_{\phi} \nabla_{\phi} \mathcal{L}_{\phi}$$

- ▶ The question is how to construct the loss function  $\mathcal{L}$  for the actor and critic.

## Backward Propagation: Actor Objective (1/2)

- ▶ The actor aims to maximize expected discounted returns:

$$J(\theta) = \mathbb{E}_{\tau \sim P_\theta} [R(\tau)], \quad R(\tau) \equiv \sum_{k=0}^{\infty} \gamma^k r_k$$

- ▶ The policy gradient can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \nabla_\theta \log \sigma(a_t | s_t; \theta) \gamma^t G_t \right], \quad G_t \equiv \sum_{k=t}^{\infty} \gamma^{k-t} r_k$$

- ▶ Realized returns  $G_t$  are noisy and depend on the entire future trajectory
- ▶ For a given policy  $\sigma(a | s; \theta)$ , the state-action value satisfies:

$$Q^\sigma(s_t, a_t) = \mathbb{E}[G_t | s_t, a_t]$$

## Backward Propagation: Actor Objective (2/2)

- ▶ By iterated expectations, replacing  $G_t$  with  $Q^\sigma(s_t, a_t)$  is exact in expectation:

$$\mathbb{E}[\nabla_\theta \log \sigma(a_t | s_t; \theta) G_t] = \mathbb{E}[\nabla_\theta \log \sigma(a_t | s_t; \theta) Q^\sigma(s_t, a_t)]$$

- ▶ In practice, the true  $Q^\sigma$  is unknown; we use the current critic estimate  $Q(s, a; \phi)$
- ▶ Sample-based actor loss (depends on current critic parameters  $\phi$ ):

$$\mathcal{L}_\theta = -\frac{1}{N} \sum_{i=1}^N \sum_{t=1}^{T_i} \log \sigma(a_{i,t} | s_{i,t}; \theta) \gamma^t Q(s_{i,t}, a_{i,t}; \phi)$$

- ▶ The critic  $Q(s, a; \phi)$  provides a lower-variance learning signal compared to realized returns

## Policy Gradient Derivation\* (1/4: Likelihood-Ratio Reformulation)

- ▶ Start from the RL objective in integral form:

$$J(\theta) = \mathbb{E}_{\tau \sim P_\theta}[R(\tau)] = \int R(\tau) P_\theta(\tau) d\tau, \quad \text{where} \quad R(\tau) \equiv \sum_{k=0}^{\infty} \gamma^k r_k$$

- ▶ Differentiate under the integral sign:

$$\nabla_\theta J(\theta) = \int R(\tau) \nabla_\theta P_\theta(\tau) d\tau$$

- ▶ Use  $\nabla_\theta P_\theta(\tau) = P_\theta(\tau) \nabla_\theta \log P_\theta(\tau)$ :

$$\nabla_\theta J(\theta) = \int R(\tau) P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) d\tau$$

- ▶ Convert to expectation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P_\theta} [\nabla_\theta \log P_\theta(\tau) R(\tau)]$$

This is the likelihood-ratio (score-function) trick.

## Policy Gradient Derivation\* (2/4: Trajectory Likelihood Factorization)

- ▶ Trajectory density factorizes as

$$P_\theta(\tau) = p(s_0) \prod_{t=0}^{\infty} \sigma(a_t | s_t; \theta) P(s_{t+1} | s_t, a_t)$$

- ▶ Only the policy depends on  $\theta$ , hence

$$\nabla_\theta \log P_\theta(\tau) = \sum_{t=0}^{\infty} \nabla_\theta \log \sigma(a_t | s_t; \theta)$$

Environment transition terms drop out of the gradient.

## Policy Gradient Derivation\* (3/4: Causality)

- ▶ Substitute the log-policy sum and expand  $R(\tau)$ :

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim P_{\theta}} \left[ \left( \sum_{t=0}^{\infty} \nabla_{\theta} \log \sigma(a_t | s_t; \theta) \right) \left( \sum_{k=0}^{\infty} \gamma^k r_k \right) \right]$$

- ▶ Swap the order of summation:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim P_{\theta}} \left[ \sum_{t=0}^{\infty} \sum_{k=0}^{\infty} \nabla_{\theta} \log \sigma(a_t | s_t; \theta) \gamma^k r_k \right]$$

- ▶ **Causality:** For each  $t$ , terms with  $k < t$  have zero expectation, since rewards before  $t$  do not depend on  $a_t$ . Therefore, restrict inner sum to  $k \geq t$ :

$$= \mathbb{E} \left[ \sum_{t=0}^{\infty} \nabla_{\theta} \log \sigma(a_t | s_t; \theta) \sum_{k=t}^{\infty} \gamma^k r_k \right]$$

- ▶ Define the return from  $t$ :  $G_t \equiv \sum_{k=t}^{\infty} \gamma^{k-t} r_k$ , so  $\sum_{k=t}^{\infty} \gamma^k r_k = \gamma^t G_t$

## Policy Gradient Derivation\* (4/4: Surrogate Objective)

- ▶ Substitute  $G_t$  and collect terms:

$$\nabla_{\theta} J(\theta) = \mathbb{E} \left[ \sum_{t=0}^{\infty} \nabla_{\theta} \log \sigma(a_t | s_t; \theta) \gamma^t G_t \right]$$

- ▶ **Surrogate objective:**

$$\max_{\theta} \mathbb{E} \left[ \sum_{t=0}^{\infty} \log \sigma(a_t | s_t; \theta) \gamma^t G_t \right]$$

- ▶ This surrogate is constructed so that its gradient with respect to  $\theta$  equals the true policy gradient  $\nabla_{\theta} J(\theta)$ .
- ▶ We maximize the surrogate because it is differentiable from sampled data and enables gradient-based optimization of the policy.
- ▶ Sample-based approximation:

$$\hat{J}(\theta) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^{T_i} \log \sigma(a_{i,t} | s_{i,t}; \theta) \gamma^t G_{i,t}$$

## Backward Propagation: Critic Objective

- ▶ The critic aims to approximate the state-action value function  $Q^\sigma(s, a)$
- ▶ The true  $Q^\sigma$  satisfies the Bellman equation:

$$Q^\sigma(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1}}[Q^\sigma(s_{t+1}, a_{t+1})]$$

- ▶ Sample-based critic loss (mean squared Bellman error):

$$\mathcal{L}_\phi = \frac{1}{N} \sum_{i=1}^N (Q(s_i, a_i; \phi) - (r_i + \gamma \mathbb{E}_{s_{i+1}}[Q(s_{i+1}, a_{i+1}; \phi)]))^2$$

## Implementation Caveat 1: Target Networks

- ▶ Problem: The critic loss depends on its own predictions

$$\mathcal{L}_\phi = \frac{1}{N} \sum_{i=1}^N \left( Q(s_i, a_i; \phi) - \underbrace{(r_i + \gamma Q(s_{i+1}, a_{i+1}; \phi))}_{\text{target uses same } \phi} \right)^2$$

- ▶ This creates a *moving target* problem  $\Rightarrow$  training instability
- ▶ Solution: Use separate *target networks*  $\phi_{\text{target}}, \theta_{\text{target}}$

$$y_i = r_i + \gamma Q(s_{i+1}, \sigma(s_{i+1}; \theta_{\text{target}}); \phi_{\text{target}})$$

- ▶ Target networks are updated slowly:
  - ▶ Hard update: Copy parameters every  $K$  steps
  - ▶ Soft update (Polyak averaging):  $\phi_{\text{target}} \leftarrow \tau\phi + (1 - \tau)\phi_{\text{target}}$ , with  $\tau \ll 1$
- ▶ Result: Stable targets  $\Rightarrow$  more stable learning

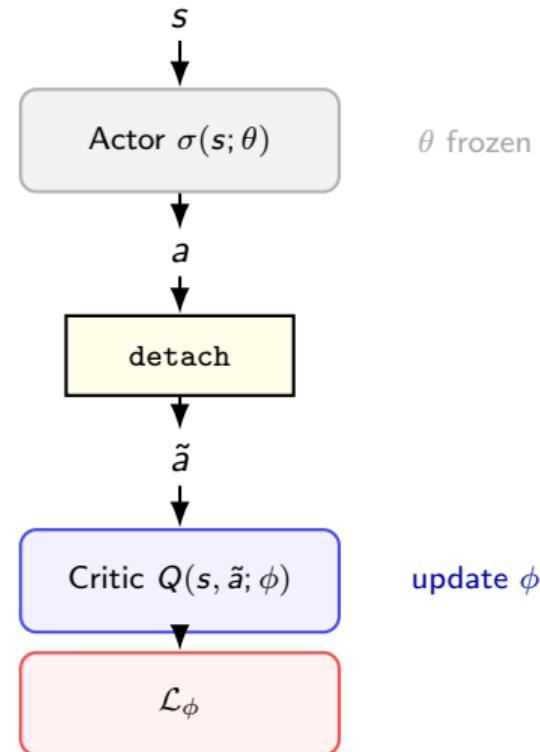
## Implementation Caveat 2: Detaching the Action

The detach operation creates a copy of a tensor that is disconnected from the computational graph, blocking gradient propagation.

- ▶ The action  $a = \sigma(s; \theta)$  is produced by the actor
- ▶ During critic updates, the action should be treated as fixed input data
- ▶ Detaching prevents gradients from flowing back to  $\theta$

Implementation:

- ▶ `a_detached = a.detach()`
- ▶ Pass `a_detached` as input to  $Q(s, a; \phi)$



The critic update optimizes  $\phi$  only; the actor is updated separately via policy gradient.