Lecture 8: Simulation-Based Learning

Yasuyuki Sawada, Yaolang Zhong

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

December 10, 2025

Recap: The Computational Challenge

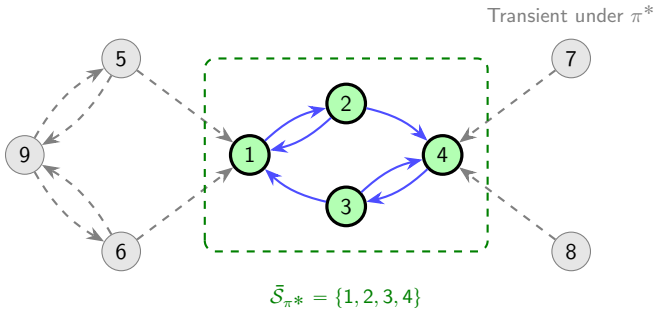
- Recall the Bellman Equation:

$$V(s) = \max_a \left\{ r(s,a) + \beta \int V(s') P(ds' \mid s, a) \right\}$$

- So far, deterministic update:
  - enumerate all states $s \in \mathcal{S}$ (use a fixed grid in the continuous case)
  - update $V(s)$ by enumerating over all states $s' \in \mathcal{S}$ and integrating over P
  - Lecture 7: introduce stochasticity by Monte Carlo integration (but still iterate over all s)
- However: Enumerating all states s means we want accurate $V(s)$ everywhere: the evaluation and update is costly, but the contrubution to gain in accuracy is marginal.
- Ergodic set: the set of states that the agent visits infinitely often under a particular policy. Denote the ergodic set corresponding to a policy π as $\bar{\mathcal{S}}_\pi$.

Example 1: Ergodic Set in a Discrete Markov Chain

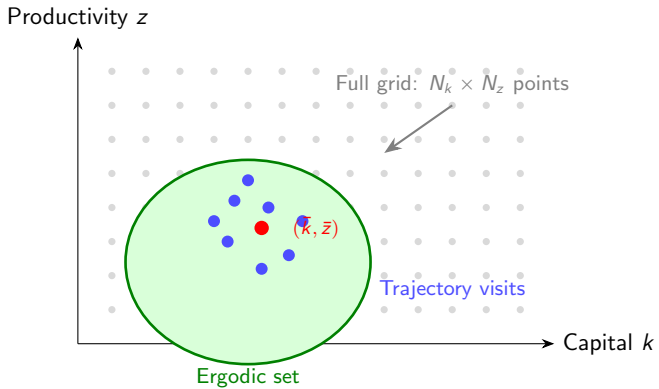
Consider a 9-state MDP. The optimal policy π^* induces transitions that only recurrently visit states $\{1, 2, 3, 4\}$.



- Even for a suboptimal policy π, once it improves sufficiently, we often have $\bar{\mathcal{S}}_\pi = \bar{\mathcal{S}}_{\pi*}$.
- Implication: updating $V(s)$ for $s \notin \bar{\mathcal{S}}_{\pi*}$ wastes computation—these states are never revisited.

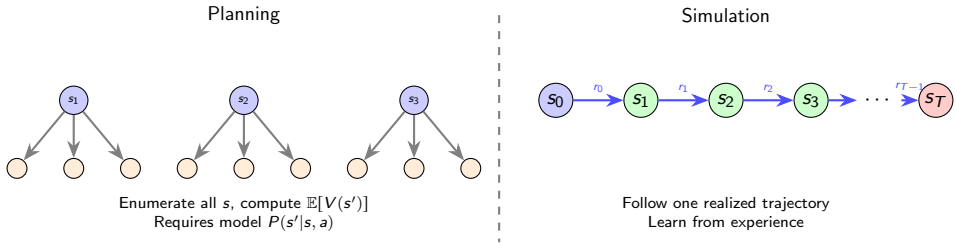
Example 2: Ergodic Set in a Growth Model

In the stochastic growth model $(k_t, z_t) \to (k_{t+1}, z_{t+1})$, the ergodic set is a small region of the state space.



Judd, Maliar, Maliar & Valero (2014): Simulated trajectories visit $< 5\%$ of grid points in high-dimensional models. Updating only visited states achieves similar accuracy at a fraction of the cost.

Planning vs Simulation: illustration



- Trajectory / Episode / Rollout: A sequence of states and rewards $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ generated by following a policy π from initial state s_0 to terminal state s_T.
- The associated return is $G_0 = r_0 + \beta r_1 + \beta^2 r_2 + \cdots + \beta^T r_{T-1}$

Planning: The Bellman Update

- At iteration k, we update $V^{(k)}$ to $V^{(k+1)}$ using the Bellman equation:

$$V^{(k+1)}(s) = r(s) + \beta \int V^{(k)}(s')\, P(ds'|s) = \int \underbrace{(r(s) + \beta V^{(k)}(s'))}_{\text{Target}}\, P(ds'|s)$$

- Planning computes the integral exactly (quadrature) or via MC integration with M samples:

$$V^{(k+1)}(s) = \frac{1}{M} \sum_{m=1}^{M} \left[r(s) + \beta V^{(k)}(s'_m) \right] = \frac{1}{M} \sum_{m=1}^{M} \text{Target}_m^{(k)}$$

- Simulation-based: sample size $M = 1$ and incremental update weighted by α_k (learning rate):

$$\begin{aligned} V^{(k+1)}(s) &= V^{(k)}(s) + \alpha_k \big[\underbrace{r + \beta V^{(k)}(s') - V^{(k)}(s)}_{\text{Bellman residual } \delta_k} \big] \\ &= (1 - \alpha_k) \cdot \underbrace{V^{(k)}(s)}_{\text{old estimate}} + \alpha_k \cdot \underbrace{\big[r(s) + \beta V^{(k)}(s') \big]}_{\text{Target}} \end{aligned}$$

Convergence: Simulation → Planning

- Suppose state s has been visited $n(s)$ times. On the $n(s)$-th visit, choose $\alpha_{n(s)} = \frac{1}{n(s)}$, which satisfies the Robbins–Monro conditions: $\sum_k \alpha_k = \infty$ and $\sum_k \alpha_k^2 < \infty$.
- The simulation update on the $n(s)$-th visit becomes

$$V^{(k+1)}(s) = V^{(k)}(s) + \frac{1}{n(s)}\Big[\underbrace{r(s) + \beta V^{(k)}(s')}_{\text{Target}_{n(s)}} - V^{(k)}(s)\Big] = \frac{1}{n(s)}\,\text{Target}_{n(s)} + \frac{n(s)-1}{n(s)}\,V^{(k)}(s)$$

- Unrolling: substitute $V^{(k)}(s) = \frac{1}{n(s)-1}\text{Target}_{n(s)-1} + \frac{n(s)-2}{n(s)-1}V^{(k-1)}(s)$, and repeat:

$$\begin{aligned} V^{(k+1)}(s) &= \frac{1}{n(s)}\,\text{Target}_{n(s)} + \frac{n(s)-1}{n(s)} \cdot \frac{1}{n(s)-1}\,\text{Target}_{n(s)-1} + \cdots \\ &= \frac{1}{n(s)}\,\text{Target}_{n(s)} + \frac{1}{n(s)}\,\text{Target}_{n(s)-1} + \cdots + \frac{1}{n(s)}\,\text{Target}_1 \\ &= \frac{1}{n(s)} \sum_{i=1}^{n(s)} \text{Target}_i \end{aligned}$$

Temporal Difference (TD) Learning: Formal Introduction

- The simulation-based update we derived uses the estimated value $V(s_{t+1})$ to update $V(s_t)$. This is called *bootstrapping*: updating an estimate based on another estimate. (This differs from the bootstrap in statistics, which is resampling from data to estimate uncertainty.)
- Simulation-based methods with bootstrapping are called Temporal Difference (TD) learning:

$$V(s_t) \leftarrow V(s_t) + \alpha \cdot \big(r_t + \beta V(s_{t+1}) - V(s_t)\big), \quad \text{where } s_t \sim \bar{\mathcal{S}}_\pi$$

- The name "Temporal Difference" refers to the difference between value estimates at two successive time points: $V(s_{t+1})$ vs $V(s_t)$.
- The Bellman residual $\delta_t = r_t + \beta V(s_{t+1}) - V(s_t)$ is also called the TD error.
- This specific update rule is called TD(0). The "0" indicates that we bootstrap immediately after observing just one reward r_t, using zero additional steps before estimating the remaining value via $V(s_{t+1})$.

TD(0) Algorithm for Policy Evaluation

- Step 0 (inputs): Policy π to evaluate; discount $\beta \in (0,1)$; learning rate schedule $\{\alpha_k\}$; number of episodes K; initial state distribution μ_0.
- Step 1 (initialize): For all states $s \in \mathcal{S}$, set $V^{(0)}(s) \leftarrow 0$ (or arbitrary values).
- Step 2 (episode loop): For episode $k = 1, 2, \ldots, K$:
  - Sample initial state $s_0 \sim \mu_0$.
  - For $t = 0, 1, 2, \ldots$ until terminal state:
    1. Take action $a_t = \pi(s_t)$, observe reward r_t and next state s_{t+1}.
    2. Compute TD error: $\delta_t = r_t + \beta\, V(s_{t+1}) - V(s_t)$.
    3. Update: $V(s_t) \leftarrow V(s_t) + \alpha_k \cdot \delta_t$.
- Step 3 (output): Value function estimate $V \approx V^{\pi}$.

The Exploration Problem: Ergodic Set Depends on π

- The ergodic set of states visited under a policy,

$$\bar{\mathcal{S}}_\pi = \{s : \Pr_\pi(s_t = s \text{ for some } t) > 0\},$$

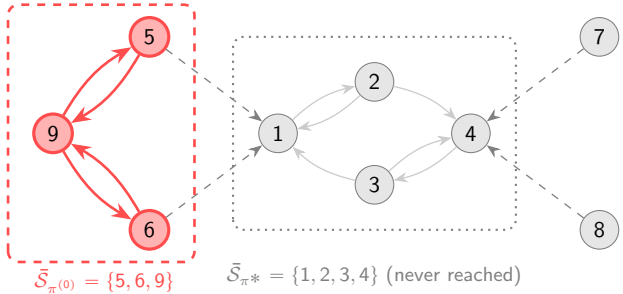
  depends on the policy π itself.
- During learning, the policy evolves:

$$\pi^{(0)} \to \pi^{(1)} \to \pi^{(2)} \to \cdots \to \pi^*.$$

- This creates a fundamental difficulty:
  - Accurate estimation of $V(s)$ requires many visits to state s.
  - But the states we visit depend on the current policy $\pi^{(k)}$.
  - A poor early policy may prevent us from ever reaching states that matter under π^*.
- Consequences of poor initialization:
  - $\bar{\mathcal{S}}_{\pi^{(0)}}$ may omit states in $\bar{\mathcal{S}}_{\pi^*}$.
  - For such states, $V(s)$ is never updated.
  - Policy improvement cannot discover these states, and learning becomes trapped in a suboptimal region.

Example: How Poor Initialization Causes Suboptimal Convergence

Same 9-state MDP, but now consider a poorly initialized policy $\pi^{(0)}$ that gets stuck in $\{5, 6, 9\}$.



- Policy $\pi^{(0)}$ cycles within $\{5, 6, 9\}$ — never transitions to state 1.
- $V(s)$ for $s \in \{1, 2, 3, 4\}$ never gets updated $\Rightarrow$ cannot discover higher rewards there.
- Learning is stuck: $\bar{\mathcal{S}}_{\pi^{(0)}} \cap \bar{\mathcal{S}}_{\pi*} = \varnothing$.

Solutions to the Exploration Problem

- ε-greedy policy:
  - With probability $1 - \varepsilon$: follow the current best policy
  - With probability ε: take a random action
  - Ensures all state-action pairs have positive visitation probability
- Optimistic initialization:
  - Initialize $V(s)$ to high values (optimistic about unexplored states)
  - Agent is "curious" — explores until reality disappoints the optimism
- Exploring starts:
  - Begin episodes from randomly chosen states
  - Guarantees coverage of the full state space over time
- Key insight: Exploration is the price we pay for not knowing the model. Planning (with known P) can evaluate all states without visiting them; simulation cannot.

TD Learning: Advantages and Disadvantages

- Advantages:
  - Model-free: No need to know the transition probabilities P.
  - Scalable: DP must iterate over all states each sweep; TD only updates visited states, making it feasible for large or continuous state spaces.
  - Computationally cheaper: No integration required
- Disadvantages :
  - Noisy updates: using one sample s' instead of integrating over all possible s' introduces variance.
  - Exploration required: Only visited states get updated.
  - Slower convergence: Stochastic approximation converges at $O(1/\sqrt{n})$; DP with known model converges geometrically.
  - Learning rate tuning: Performance depends on α. DP has no such hyperparameter.

Generalizing TD(0): The n-Step Return

- Recall TD(0) bootstraps after observing just one reward:

$$G_t^{(1)} = r_t + \beta V(s_{t+1})$$

- What if we wait longer? Observe n actual rewards before bootstrapping:

$$G_t^{(n)} = r_t + \beta r_{t+1} + \beta^2 r_{t+2} + \cdots + \beta^{n-1} r_{t+n-1} + \beta^n V(s_{t+n})$$

- The n-step return uses n realized rewards and bootstraps from $V(s_{t+n})$.
- Examples:
  - $n = 1$ (TD(0)): $G_t^{(1)} = r_t + \beta V(s_{t+1})$
  - $n = 2$: $G_t^{(2)} = r_t + \beta r_{t+1} + \beta^2 V(s_{t+2})$
  - $n = 3$: $G_t^{(3)} = r_t + \beta r_{t+1} + \beta^2 r_{t+2} + \beta^3 V(s_{t+3})$
- As n increases, we rely more on actual rewards and less on the estimated value.

The Two Extremes: TD(0) and Monte Carlo

- TD(0) ($n = 1$): Bootstrap immediately after one reward.

$$G_t^{(1)} = \underbrace{r_t}_{\text{1 actual reward}} + \underbrace{\beta V(s_{t+1})}_{\text{bootstrap}}$$

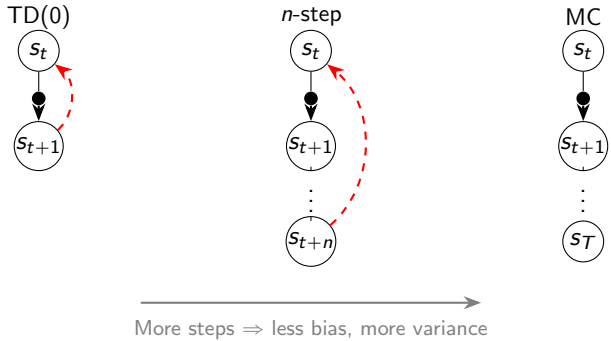
  - ✓ Low variance: only one random transition affects the target.
  - ✗ Biased: relies on current (imperfect) estimate $V(s_{t+1})$.

- Monte Carlo ($n = T - t$, i.e., wait until episode ends): No bootstrapping at all.

$$G_t^{(\infty)} = \underbrace{r_t + \beta r_{t+1} + \beta^2 r_{t+2} + \cdots + \beta^{T-t} r_T}_{\text{all actual rewards, no bootstrap}}$$

  - ✓ Unbiased: $G_t^{(\infty)}$ is a true sample of $V(s_t)$.
  - ✗ High variance: depends on many random transitions.
  - ✗ Must wait for episode to end before updating.

Visualizing the Spectrum: Backup Diagrams



- TD(0): Bootstrap after 1 step (red arrow shows bootstrap).
- n-step: Bootstrap after n steps.
- MC: No bootstrap—use full trajectory to terminal state s_T.

TD(0) vs MC: The Bias-Variance Trade-off

- TD(0) is based and MC is unbias:
  - TD(0): $r_t + \beta V(s_{t+1})$ is based on the estimated value $V(s_{t+1})$, which is our current guess—not the true value. We are "updating a guess towards another guess."
  - MC: base on the actual realized rewards, which is a random variable whose expectation equals the true $V(s_t)$.
- TD(0) has lower variance than MC:
  - TD(0): depends on only one random transition (s_t, r_t, s_{t+1}).
  - MC: accumulates randomness from all transitions in the episode: $(s_t, r_t, s_{t+1}), (s_{t+1}, r_{t+1}, s_{t+2}), \ldots$
- The n-step return interpolates: larger n means less bias but more variance.

TD(λ): A Weighted Blend of All n-Step Returns

- Instead of choosing a single n, why not combine all n-step returns?
- The λ-return takes a weighted average with geometrically decaying weights:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t^{(T-t)}$$

- The parameter $\lambda \in [0,1]$ controls the decay:
  - Weight on $G_t^{(1)}$: $(1-\lambda)$
  - Weight on $G_t^{(2)}$: $(1-\lambda)\lambda$
  - Weight on $G_t^{(3)}$: $(1-\lambda)\lambda^2$
  - $\vdots$
- Special cases:
  - $\lambda = 0$: All weight on $G_t^{(1)} \Rightarrow$ TD(0).
  - $\lambda = 1$: All weight on $G_t^{(\infty)} \Rightarrow$ Monte Carlo.

Numerical Experiment: VFI vs TD Policy Evaluation

Setup: Evaluate the *same* optimal policy π^* (solved by VFI) using different methods.

- Model: Consumption-saving with discrete assets and income shocks.
- Ground truth: V^{VFI} from exact dynamic programming.
- TD evaluation: Run TD(λ) for $\lambda \in \{0, 0.5, 0.9, 1.0\}$.
- Repeat each TD method 5 times with different random seeds.

Metrics:

- MSE: $\frac{1}{|\mathcal{S}|} \sum_s (V^{\text{TD}}(s) - V^{\text{VFI}}(s))^2$ (bias + variance)
- Variance: $\frac{1}{|\mathcal{S}|} \sum_s \text{Var}[V^{\text{TD}}(s)]$ across runs

Key insight: All methods evaluate the *same* policy—differences are purely due to estimation method.

Code: Comparing VFI and TD(λ)

```
from model import create_default_model
from algos import compare_vfi_td

model = create_default_model()
results = compare_vfi_td(
    model,
    lambdas=[0.0, 0.5, 0.9, 1.0],
    n_runs=5,
    n_episodes=5000,
    alpha=0.5
)

# Results contain:
# - V_vfi: exact value function from VFI
# - td_results[lambda]: V_mean, V_std, mse, avg_variance
```

See `Lab_8_Simulation_Based_Learning/` for full implementation.

Expected Results: Bias-Variance Trade-off

Method	MSE vs VFI	Avg Variance	Interpretation
TD(0), $\lambda = 0$	Medium	Low	High bias, low variance
TD(0.5)	Lower	Medium	Balanced
TD(0.9)	Lower	Higher	Less bias, more variance
MC, $\lambda = 1$	Lowest bias	Highest	Unbiased but noisy

Observations:

- TD(0) converges fast but may have residual bias (depends on α schedule).
- MC is unbiased in the limit but requires many episodes to reduce variance.
- Intermediate λ (e.g., 0.9) often achieves the best MSE in practice.
- All TD methods are slower than VFI when the model is known—but TD works without knowing P.