AI for Economic Research: Dynamic Models, Language, and Agents
Lecture 5.2: Monte Carlo, Exploration, and Temporal-Difference Learning

Zhigang Feng

2026-07-08 19:04:44

Lecture 5 in One Map — Act 5.2

Act 5.1 ended with three departures on the table. This act executes #1 — replace the full-expectation backup, which needs a known model (P, R), with a sample backup: Monte Carlo, TD, Q-learning — and pays sampling's price, #3: exploration.

The internal flow of this act: a concrete economic gridworld → **Monte Carlo** *(learn from whole episodes) →* ***the exploration–exploitation problem*** *→* **temporal-difference** *and* **Q-learning** *(update every step). One backbone underneath all of them: Generalized Policy Iteration.*

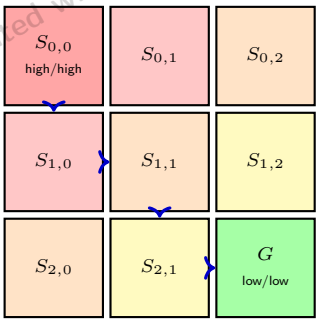
A Motivating Example

A small economic gridworld to make “learning from experience” concrete

A 3×3 Economic Grid

- A **policymaker** navigates macroeconomic conditions (inflation × unemployment).
- Goal: reach and maintain the optimal state (low inflation, low unemployment).

- Top-left ($S_{0,0}$): High inflation & unemployment (worst).
- Bottom-right ($G \equiv S_{2,2}$): Low inflation & low unemployment (**goal**).
- Episode ends upon reaching the goal.
- Optimal trajectory (arrows) shows one greedy path; many equivalent paths exist.



Macro analogue of textbook gridworld.

What Makes This RL, Not DP?

Known (the MDP skeleton): the states $\mathcal{S}$ (9 cells), the actions $\mathcal{A}$ (Up/Down/Left/Right), and the discount γ.

Unknown — and this is what forces *learning*:

- **Transition dynamics $P(s' \mid s, a)$.** The policymaker does *not* know where an action lands them. Does "Down" (expansionary policy) reliably cut unemployment, or overshoot into higher inflation? The *economy's response to an intervention* is exactly the unknown.
- **Reward function $R(s, a)$.** The agent doesn't know *where* the $+1$ lives — it learns "this was the goal" only by *reaching* G.

Known model → DP; unknown model → RL. With P, R unknown you *cannot* sweep the Bellman operator (value iteration). The agent must *sample* trajectories and learn $\widehat{Q}$ from experienced returns (MC) or bootstrapped transitions (TD/Q-learning) — the topic of the rest of this lecture.

Same Framework, Very Different Worlds

Two questions decide how hard an RL problem is: **(a)** is the model (P, R) known? **(b)** can you *explore freely* (reset, simulate, fail cheaply)?

	Dynamics P	**Reward R**	**Free exploration?**
Game of Go	rules known & deterministic; *opponent* unknown	known (win/lose)	**yes** — perfect simulator, self-play
Autonomous driving	unknown, stochastic	hard to *specify*	limited — sim + logged data; unsafe to fail
Real economy	unknown, *non-stationary*	mostly known (mandate loss)	**no** — one history, no resets

Go is the "easy" corner: the *rules* are a perfect free simulator, so the real challenge is not unknown dynamics but the *opponent* and sheer *size* ($\sim 10^{170}$ states) — solved by self-play + look-ahead search (AlphaZero).

Why the Economy Is the Hardest Case

The gridworld teaches the *mechanics* of learning an unknown P. The real economy adds three difficulties the toy problem — and even Go and driving — do not have:

- P **is reflexive (Lucas critique).** The economy's response *changes when the policy rule changes*, because agents are forward-looking. You learn a target that moves *because* you are learning it — not a fixed table.
- **You cannot explore.** No resets, no parallel worlds: a central bank gets *one* history, and a bad "exploratory" move costs real jobs. Algorithms that assume millions of episodes simply do not apply.
- **Partial observability.** The true state (output gap, natural rate) is never observed directly — only estimated with noise. Really a **POMDP** (*partially observable* MDP): the agent sees noisy signals, not the state, so it must act on a *belief* — a distribution over where the economy truly is.

Takeaway. It is not that the *math* is deeper in macro — it is that you *cannot run the experiment*. This is why macro leans so hard on *models* (structure, theory) to substitute for the exploration it can never do.

Actions and Rewards on the Grid

Actions (Policy Interventions):

- **Up**: Contractionary policy (reduces inflation, may raise unemployment).
- **Down**: Expansionary policy (reduces unemployment, may raise inflation).
- **Left/Right**: Fiscal or structural interventions.

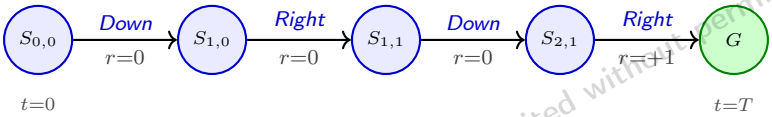
Reward Structure:

$$R(s,a) = \begin{cases} +1 & \text{if next state} = \text{Goal state } G, \\ 0 & \text{otherwise.} \end{cases}$$

MDP on the grid: $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ where $\mathcal{S} = \{S_{i,j}\}$, $\mathcal{A} = \{\text{Up, Down, Left, Right}\}$.

Illustrative Episode

Example rollout ($r_t = 0$ until goal; $r_T = +1$):



- Cumulative return $G_0 = \sum_{t=0}^{T-1} \gamma^t r_{t+1} = \gamma^{T-1}$.
- Update $Q(s, a)$ by averaging returns over many such episodes.
- Repeated simulation $\rightarrow$ stable policy guiding the economy toward G.

How do we compute updates? Three families:

1. **Monte Carlo** — wait for the full episode, then average.
2. **Temporal Difference** — update after every single step.
3. **Actor-Critic** — combine policy learning with value estimation.

Monte Carlo Methods

Learn from complete simulated episodes — no model required

Model-Based DP vs. Monte Carlo

Model-Based Policy Iteration (Classical DP):

1. **Policy evaluation:** Solve the Bellman equation:
   $V^\pi(s) = \sum_a \pi(a|s)\big[R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^\pi(s')\big]$.
2. **Policy improvement:** $\pi'(s) = \arg\max_a Q^\pi(s,a)$.

Requirement: Must know $P(s'|s,a)$ and $R(s,a)$.

Monte Carlo (MC):

- Learn through **simulated episodes** without requiring explicit transition models.
- Estimate $Q(s,a)$ from **observed returns** instead of computing expectations.
- Like running a randomized policy experiment many times and averaging the outcomes.

Basic Monte Carlo: Key Idea

Simulate full episodes, then average the returns.

- Simulate multiple **full episodes** from each (s, a).
- Collect total discounted return:

$$G_t = \sum_{k=0}^{T-1} \gamma^k r_{t+k+1}.$$

- Update rule:

$$Q(s,a) \leftarrow \frac{1}{N} \sum_{i=1}^{N} G_0^{(i)},$$

where N = number of episodes, $G_0^{(i)}$ = return from episode i.

Why full episodes?

- Captures **long-run consequences** of actions.
- Avoids dependence on explicit transition models.
- By the Law of Large Numbers, estimates converge.

Monte Carlo: Two Senses of "Converge"

When we say Monte Carlo "converges," it helps to keep **three distinct quantities** straight:

- G — **not iterated.** One episode gives *one* return, computed directly by summing its discounted rewards. Each episode $\rightarrow$ one sample $G_0^{(i)}$.
- $\widehat{Q}$ **for a fixed** π — **converges statistically.** Average *more* episodes and $\widehat{Q} \rightarrow Q^\pi$ by the Law of Large Numbers (variance shrinks) — *not* by iterating a Bellman fixed point the way DP does. There is no bootstrapping: each G is already unbiased; more samples just cut the noise.
- Q^*, π^* **(optimal) — the outer loop.** Reaching the *optimum* needs the GPI "ping-pong": evaluate ($\widehat{Q}$ by MC) $\rightarrow$ improve (π greedy) $\rightarrow$ re-evaluate $\rightarrow \cdots$ until $(\widehat{Q}, \pi)$ stops moving.

This slide (and the previous one) is MC **evaluation** of a *given* π; wrapping it in the improvement loop turns it into MC **control** for π^*.

Monte Carlo Applied to the Growth Model

Environment: State (k_t, z_t), action c_t, reward $u(c_t)$, transition $k_{t+1} = F(k_t, z_t) - c_t$.

MC Episode:

1. Initialize k_0, z_0.
2. For $t = 0, 1, \ldots, T-1$:
   - Choose c_t from current (or exploratory) policy.
   - Observe $r_t = u(c_t)$, transition to (k_{t+1}, z_{t+1}).
3. Compute $G_0 = \sum_{t=0}^{T-1} \gamma^t r_t$.

Key difference from VFI:

- VFI: evaluate V at **every grid point**, using known $P(z'|z)$.
- MC: evaluate V along **sampled trajectories**, no need for explicit P.

Challenges of Basic Monte Carlo

Full-Episode Dependence:

- Updates only after **entire episodes** are completed.
- Inefficient for long-horizon or continuing (non-terminating) problems.

Exploration Dilemma:

- Infrequent visits to rare but important states (e.g., financial crises).
- Theory requires every (s, a) visited infinitely often.

High Variance:

- Returns from full episodes vary significantly.
- Slow convergence $\Rightarrow$ many episodes needed.

These motivate **exploring starts / subepisodes** (coverage), **explicit exploration** (ϵ-greedy, Boltzmann, UCB), and **incremental TD** updates.

Subepisodes and Exploring Starts

Subepisodes (First-Visit vs. Every-Visit MC):

- **First-Visit MC**: Update $Q(s, a)$ only on the **first** occurrence per episode.
- **Every-Visit MC**: Update $Q(s, a)$ on **every** occurrence.
- Both speed up convergence by leveraging all data within an episode.

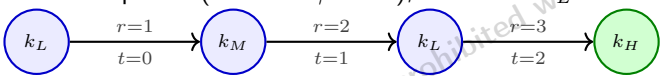
Exploring Starts:

- Each episode starts from a randomly chosen (s_0, a_0) with nonzero probability.
- Ensures every (s, a) pair receives enough visits.
- **Problem:** often unrealistic in practice (can't arbitrarily reset the economy).

$\Rightarrow$ We need a more practical exploration mechanism: ϵ-**greedy**.

First-Visit vs. Every-Visit: a Worked Trajectory

One simulated episode (discount $\gamma = 0.9$), with state k_L visited **twice**:



- Return from t=0: $G_0 = 1 + 0.9(2) + 0.9^2(3) = 5.23$. From t=2: $G_2 = 3$.
- **First-visit MC:** use G_0 only $\Rightarrow \widehat{V}(k_L) = 5.23$ (one clean, independent sample).
- **Every-visit MC:** use *both* $\Rightarrow \widehat{V}(k_L) = \frac{5.23+3}{2} = 4.12$ (more data, but correlated samples).

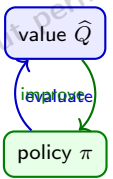
Trade-off: every-visit squeezes more from each episode; first-visit keeps samples independent.

Generalized Policy Iteration (GPI): the RL Backbone

Two steps, alternated until stable:

1. **Evaluate** — estimate $\widehat{Q}^{\pi}$ (MC: average sampled returns; DP: full backup).
2. **Improve** — act greedily: $\pi'(s) = \arg\max_a \widehat{Q}^{\pi}(s, a)$.

MC control = MC prediction inside GPI.



One backbone, every algorithm. DP, MC, TD, SARSA, Q-learning, and (in T3) actor-critic are *all* GPI — they differ only in *how evaluation approximates the expectation* and *how improvement explores*.

GPI in Action: a Two-State "Ping-Pong"

Toy growth model: capital $k \in \{k_L, k_H\}$; action = next capital k'. Current estimate $\widehat{Q}$:

$\widehat{Q}(k,k')$	$k'=k_L$	$k'=k_H$
$k=k_L$	3	**5**
$k=k_H$	**8**	6

- **Improve** (greedy): $\pi(k_L)=k_H$ $(5>3)$, $\pi(k_H)=k_L$ $(8>6)$. *Low capital → invest, high capital → consume* — the sensible rule.
- **Re-evaluate** $\widehat{Q}$ under the new π, then improve again — "ping-pong" until $(\widehat{Q}, \pi)$ stops moving: a fixed point $\approx (Q^*, \pi^*)$.

When noise fools the $\arg\max$**.** True values at k_L: $Q(k_L,k_L)=3$, $Q(k_L,k_H)=5$, so "invest" (k_H) is correct. But with few sampled returns, one lucky episode inflates $\widehat{Q}(k_L,k_L)=5.4$ vs. $\widehat{Q}(k_H,k_H)=5.0$: greedy now flips to k_L (stay low) — the *wrong* action, a *worse* policy. Monotone improvement holds only with *exact* Q — hence the need for **exploration** (next).

Exploration vs. Exploitation

To learn a good policy you must sometimes act sub-optimally on purpose

The Exploration–Exploitation Tradeoff

Key Challenge in RL:

- **Exploitation:** Choose the best-known action to maximize *immediate* reward.
- **Exploration:** Try actions that may seem suboptimal now but could yield better *long-term* rewards.

Economic Analogies:

- **Dynamic Pricing:** Use the current best-known price, or test new prices?
- **Portfolio Management:** Stick to blue-chip stocks, or diversify into unknowns?
- **R&D Investment:** Exploit existing technology, or explore new innovations?
- **Monetary Policy:** Follow the Taylor rule, or experiment with novel instruments?

ϵ-Greedy Exploration

- With probability $1-\epsilon$: select the **best-known action** (greedy).
- With probability ϵ: select an action **at random** (explore).

$$\pi(a|s) = \begin{cases} 1-\epsilon+\frac{\epsilon}{|\mathcal{A}(s)|}, & \text{if } a = \arg\max_{a'} Q(s,a') \\ \frac{\epsilon}{|\mathcal{A}(s)|}, & \text{otherwise.} \end{cases}$$

- Ensures all actions have **nonzero probability** $\Rightarrow$ every (s,a) visited eventually.
- Simple, practical, widely used.
- Limitation: explores **uniformly** among non-greedy actions (doesn't favor promising ones).
- **GLIE** — *Greedy in the Limit with Infinite Exploration* (convergence condition): decay $\epsilon_n \to 0$ slowly (e.g. $\epsilon_n = 1/n$) so the policy becomes greedy in the limit, yet every (s,a) is still visited infinitely often — then ϵ-greedy GPI converges to π^*.

Boltzmann (Softmax) Exploration

Idea: Actions chosen **probabilistically**, weighted by estimated value.

$$\Pr(a \mid s) = \frac{\exp\big(Q(s,a)/\tau\big)}{\sum_{a'} \exp\big(Q(s,a')/\tau\big)},$$

where $\tau > 0$ is the **temperature**:

- τ **high** $\Rightarrow$ near-uniform exploration.
- τ **low** $\Rightarrow$ near-greedy selection.

Connection to LLMs: Large Language Models use the same "temperature" parameter to balance creativity (exploration) and consistency (exploitation) in text generation.

Economic example: Online advertising allocates budget to ads $\propto \exp(\text{estimated ROI}/\tau)$.

ϵ-Greedy vs. Boltzmann: a Numerical Look

Three actions with current values $\widehat{Q} = (10,\ 9,\ 1)$:

	a_1 (Q=10)	a_2 (Q=9)	a_3 (Q=1)
ϵ-greedy (ϵ=0.1)	0.93	0.03	0.03
Boltzmann (τ=2)	0.62	0.37	0.01

Where the numbers come from: each row is the action-selection probability.

ϵ-greedy: best action gets $1 - \epsilon + \frac{\epsilon}{3} = 0.9 + 0.033 = 0.93$; each other gets $\frac{\epsilon}{3} = 0.033$.

Boltzmann: $\Pr(a) = \dfrac{e^{Q(a)/\tau}}{\sum_{a'} e^{Q(a')/\tau}}$, e.g. $a_1 = \dfrac{e^{10/2}}{e^{5} + e^{4.5} + e^{0.5}} = \dfrac{148.4}{240.1} = 0.62$.

- ϵ-greedy spreads exploration **uniformly**: the near-optimal a_2 and the clearly-bad a_3 get the *same* probability (0.03).
- Boltzmann respects the **value ordering**: it explores the promising a_2 (0.37) far more than the inferior a_3 (0.01).

The same softmax/temperature structure economists know from discrete-choice (logit) models — and from LLM decoding.

Upper Confidence Bounds (UCB)

Key idea: Give an **exploration bonus** to less-tried actions.

$$a^* = \arg\max_a \left[\hat{Q}(a) + \alpha \sqrt{\frac{\ln t}{N(a)}} \right],$$

where $\hat{Q}(a)$ = average reward of a, $N(a)$ = number of times a has been tried, and t = **total** number of action selections so far (so $t = \sum_a N(a)$).

- **Intuition:** the bonus is an *optimism* term — an upper edge of a confidence interval. $\ln t$ grows with total experience, $N(a)$ in the denominator shrinks the bonus for actions already tried a lot.
- As $N(a)$ grows, the bonus shrinks ⇒ converges to exploitation. Under-tried actions ($N(a)$ small relative to t) get a large bonus ⇒ principled "optimism in the face of uncertainty."

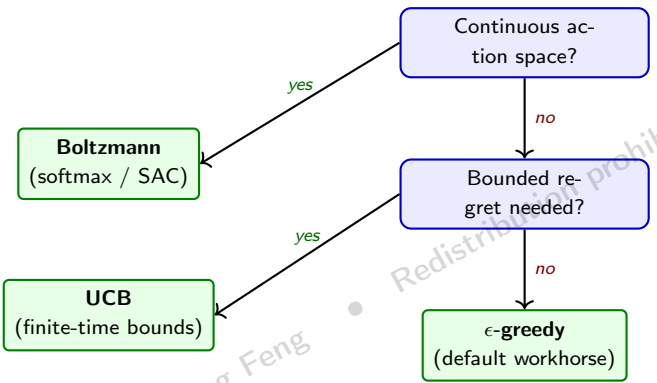
Economic example: A firm evaluating new markets. Early data is sparse, so a confidence bonus encourages trying each region enough to reduce uncertainty.

Comparison Table — ϵ-Greedy / Boltzmann / UCB

Method	Hyperparameter	Exploration mechanism	Sample complexity / when best
ϵ-Greedy	$\epsilon \in [0, 1]$	Uniform random with prob. ϵ; greedy with prob. $1 - \epsilon$	$\tilde{\mathcal{O}}(\lvert\mathcal{A}\rvert/\epsilon^2)$ visits; default workhorse for tabular and deep RL
Boltzmann	Temperature $\tau > 0$	Softmax over Q, $\Pr(a\lvert s) \propto \exp(Q/\tau)$	Smooth; favors near-optimal actions; natural for continuous $\mathcal{A}$
UCB	Exploration weight α	Bonus $\alpha\sqrt{\ln t / N(a)}$ on under-tried actions	$\tilde{\mathcal{O}}(\sqrt{\lvert\mathcal{A}\rvert t \ln t})$ regret; principled but assumes stationarity

Optimistic initialization is a complementary trick: start $Q(s, a)$ large so every action is tried at least once. Often combined with ϵ-greedy in early training.

When to Use Which: Decision Rule by Problem Class



portfolio weight, policy rate, continuous control

online ads, A/B testing on a budget

tabular RL, deep-RL warm-up

Choice also depends on: cost of a bad action, size of $\mathcal{A}$, whether $Q(s, a)$-uncertainty can be quantified.

Temporal-Difference Learning and Q-Learning

Update every step: sampling (like MC) + bootstrapping (like DP)

From MC to TD: The Key Innovation

Monte Carlo: waits until the **end of an episode** to update.

- Analogy: estimate a mean by collecting a **full batch** of data, then averaging.

Temporal Difference (TD): updates **after every single step**.

- Analogy: **incremental mean update** after each new data point.

From statistics:

- Non-incremental: $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} X_i \quad \rightarrow$ collect n samples, then compute.
- Incremental: $\hat{\mu}_{k+1} = \hat{\mu}_k + \alpha_{k+1}(X_{k+1} - \hat{\mu}_k) \quad \rightarrow$ update after each sample.

TD applies this incremental logic to **value function estimation**.

TD(0): The Simplest TD Method

MC update:

$$V(s_t) \leftarrow V(s_t) + \alpha\big(\underbrace{G_t}_{\text{full return}} - V(s_t)\big).$$

TD(0) update:

$$V(s_t) \leftarrow V(s_t) + \alpha\Big(\underbrace{r_{t+1} + \gamma\, V(s_{t+1})}_{\text{one-step target}} - V(s_t)\Big).$$

Key difference:

- MC uses the **actual full return** G_t (unbiased, high variance).
- TD uses a **bootstrapped target** $r_{t+1} + \gamma V(s_{t+1})$ (biased, lower variance), and learns **online**.

TD Error: $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ (the "surprise" signal).

Biased, but self-correcting: the target aims at the Bellman image $T^\pi V$, not V^π. While V is wrong it is biased — but T^π is a contraction, so as $V \to V^\pi$ the bias vanishes.

TD(0) Applied to the Growth Model

Setup: State $s_t = (k_t, z_t)$, policy π chooses c_t, reward $r_{t+1} = u(c_t)$.

Each time step:

1. Observe state (k_t, z_t), choose c_t via policy π.
2. Receive reward $r_{t+1} = u(c_t)$.
3. Observe next state (k_{t+1}, z_{t+1}).
4. Update:

$$V(k_t, z_t) \leftarrow V(k_t, z_t) + \alpha \Big[u(c_t) + \beta\, V(k_{t+1}, z_{t+1}) - V(k_t, z_t) \Big].$$

Advantage over VFI: no need to sum over all possible z'. Just use the **realized** transition.

Advantage over MC: no need to simulate an entire lifetime. Update **every period**.

MC vs. TD(0) vs. DP: Comparison

Dimension	Monte Carlo	TD(0)	Dynamic Prog.
Bias / variance	Unbiased / high	Biased / low	Exact (no sampling)
Bootstrapping	No (full return)	Yes (1-step target)	Yes (full backup)
Requires model?	No	No	Yes (P, R)
On-/off-policy	Either	Either	N/A (planning)
Sample complexity	Episodes	Steps	Sweeps (no samples)
Convergence	Asymptotic (LLN)	Asymptotic	Finite (contraction)

Key insight: TD sits between MC and DP. It uses **sampling** (like MC) and **bootstrapping** (like DP), combining the best of both worlds.

On-Policy vs. Off-Policy Learning

- **On-Policy (SARSA):**
  - The *behavior policy* (collects data) = the *target policy* (being improved).
  - Updates based on the **action actually taken**:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big].$$

- **Off-Policy (Q-Learning):**
  - The behavior policy can **differ** from the target policy.
  - Updates assume **greedy** next action, regardless of what was actually done:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\big].$$

Economic insight: Off-policy = learn about the **optimal** strategy while exploring with a different behavior. Like studying the best possible portfolio allocation while actually holding a diversified "safe" portfolio.

Comparing SARSA and Q-Learning

Feature	SARSA (On-Policy)	Q-Learning (Off-Policy)
Update target	$r + \gamma Q(s', a')$	$r + \gamma \max_{a'} Q(s', a')$
Behavior = target?	Yes	No (can differ)
Converges to	Q^π (of *actual* behavior)	Q^* (optimal)
Risk awareness	Reflects real actions	Plans for "ideal" actions

Which to choose?

- **SARSA**: when **safety** matters — learning matches actual behavior (e.g., autonomous driving).
- **Q-Learning**: when you want the **optimal** policy directly and can tolerate off-policy mismatch.

Off-Policy Evaluation: Learning a New Rule from Old Data

Goal: evaluate a *target* policy π using trajectories generated by a *different behavior* policy μ (e.g. historical data).

$$\mathbb{E}_\pi[G_t] = \mathbb{E}_\mu[\rho_{t:T-1}\, G_t]\,, \qquad \rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(a_k|s_k)}{\mu(a_k|s_k)}$$

the **importance-sampling** (reweighting) ratio.

Volcker as a "behavior policy." Pre-1979 vs. post-1979 Fed behavior $\approx$ two different Taylor rules (Clarida–Galí–Gertler 2000). IS lets us estimate the value of a *new* candidate rule π from historical data — **without re-running the policy experiment**. This is *counterfactual policy evaluation*.

Two cautions. (i) **Coverage / overlap:** need $\mu(a|s) > 0$ wherever $\pi(a|s) > 0$ — the RL version of the *common-support* condition in causal inference (the ratio is dynamic IPW). (ii) The *product* of ratios can make the variance **explode** over long horizons.

Q-Learning in the Growth Model

Watkins & Dayan (1992), *Machine Learning*, "Q-learning"

- **Action-value function:** $Q(k, c) = u(c) + \beta\, \mathbb{E}[V(k') \mid k, c]$.
- **Optimal policy:** $\pi^*(k) = \arg\max_c Q^*(k, c)$.
- **Q-Learning update (online, model-free):**

$$Q_{m+1}(k_t, c_t) = Q_m(k_t, c_t) + \alpha \Big[u(c_t) + \beta \max_{c'} Q_m(k_{t+1}, c') - Q_m(k_t, c_t) \Big].$$

- **Advantages:**
  - Works with stochastic or **unknown** transitions (sample-based).
  - Can approximate $Q(\cdot)$ with neural networks for continuous state/action spaces.
  - Converges to Q^* under standard conditions.

Multi-Step Methods: TD(λ)

Idea: Combine partial returns over n steps with bootstrapping.

- $n = 1$: recovers TD(0).
- $n = T - t$: recovers MC.
- Intermediate n: balances bias and variance.

TD(λ) — eligibility traces:

- Blends all n-step returns using exponential weighting parameter $\lambda \in [0, 1]$.
- $\lambda = 0$: pure TD(0). $\lambda = 1$: pure MC.
- Often improves convergence in practice.

One dial, two endpoints: λ slides continuously between **TD(0)** (λ=0: one-step, low variance) and **Monte Carlo** (λ=1: full return, unbiased). Every method in Act 5.2 is a choice along this single bias–variance dial.

Bridge: From Tables to Function Approximation

Act 5.2 delivered the model-free core. MC, TD(0), SARSA, and Q-learning all learn values and policies from sampled experience — tied together by one backbone, Generalized Policy Iteration.

- But every method so far stores one number per state (or state–action pair) — a **table**. That is exactly the wall Act 5.1 warned about: economic states (capital, prices, wealth distributions) are **continuous and high-dimensional**, so the table explodes.
- **Next (5.3):** replace the table with a **neural network**. Function approximation turns tabular Q-learning into **deep RL** (DQN); the policy-gradient / **actor–critic** family then handles continuous actions — and we see where the recipe has actually worked (AlphaGo, RLHF, AlphaManager, Krusell–Smith).

Arc: 5.1 MODEL (framework + DP) → **5.2 SAMPLE** *(MC / TD / Q-learning)* → *5.3 SCALE (actor–critic + deep RL + applications).*