Session 3.1

From Dynamic Programming to Reinforcement Learning

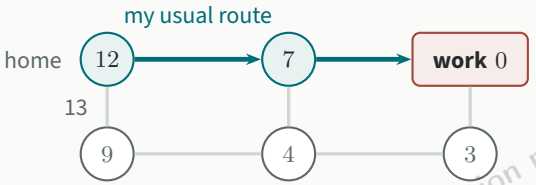


Redistribution prohibited without permission

© Zhigang Feng

Zhigang Feng

The Setup: a Map of Turns, and No Travel Times



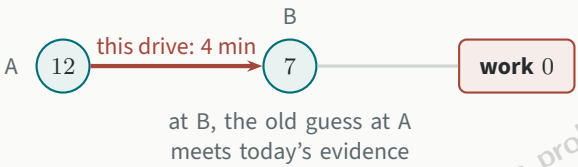
a guess per turn: from home 12 straight on, 13 down; circles show the usual turn's

known: which turns exist
unknown: minutes per street; traffic, which varies by the day
goal: a habit, one turn a corner, fewest minutes to work

© Zhigang Feng. Redistribution prohibited without permission

- New city: the map shows which turns exist; nobody knows the minutes; traffic varies.
- The goal is a habit: a turn at every corner that reaches work in the fewest minutes.
- A guess per turn at every corner: minutes to go if I take it, then drive as usual. The circle shows my usual turn's.
- A drive reveals, corner by corner: where I was, the turn, the minutes, where I landed.
- **Everything the learner will ever use is in those four things. The map only lists the options.**

How We Play: Drive, Compare, Correct a Little

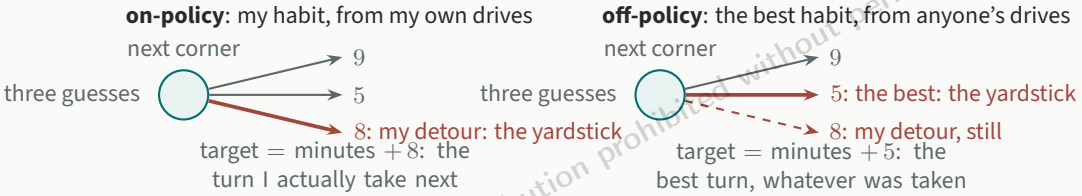


target $= 4 + 7 = 11$
surprise $= 11 - 12 = -1$
new guess $= 12 + \text{step} \times (-1)$
step $\frac{1}{2}$: $11\frac{1}{2}$; step 1: 11; step $1/n$: the average of all n drives

- Drive. At each corner take the turn with the smaller guess; now and then, any turn.
- The target at the next corner: minutes just spent plus the guess there, $4 + 7 = 11$.
- Move the old guess part of the way: the step is how much of one drive you believe; today's 4 minutes may be luck, and B's 7 is itself a guess.
- Step $\frac{1}{2}$: $11\frac{1}{2}$. Step 1 takes today whole: 11. Step $1/n$ on the n-th visit: the average of every drive, Deck 3.3's running mean.
- **Compare, correct a little, keep driving. Work anchors it: zero minutes there, a fact, passed back a corner a day.**

© Zhigang Feng Redistribution prohibited without permission

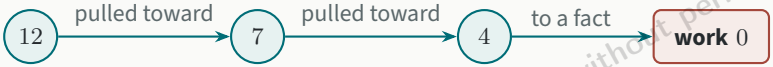
Two Ways to Learn: On-Policy and Off-Policy



- Both drive alike; all three numbers are guesses; only the yardstick differs.
- On-policy: the yardstick is the turn I take next, detour or not. I learn my own habit's value: precise, but a habit that detours is slightly wrong everywhere.
- Off-policy: the yardstick is the best turn there, whatever I take: the best habit's value; but the best of noisy guesses is optimistic, and it is checked only when driven.
- **Same drives, two questions: my habit's value, answered precisely; the best habit's, answered noisily.**

© Zhigang Feng · Redistribution prohibited without permission

Why the Guesses Settle: Every Yardstick Is Itself Corrected

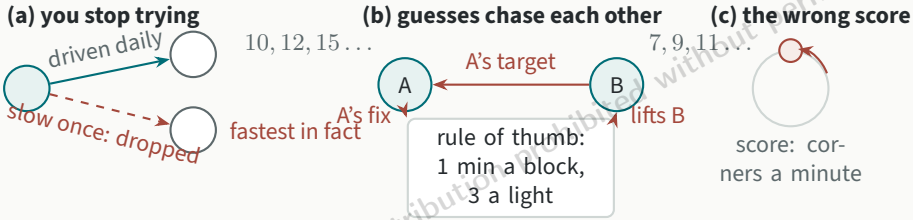


each corner corrected as often as it serves as a yardstick:
the corners you drive through, as often as you drive them

- Each corner is pulled toward the corners it leads to; the last, to the real arrival.
- You correct the corners you drive through, as often as you drive them (on-policy): every yardstick is a corner you also correct.
- So the truth travels back from work one corner at a time: the web tightens like a net drawn in to the shore.
- The classic success, TD-Gammon: backgammon learned by self-play; every position it judged by, it also corrected, and every game ended in a real result.
- **Guesses checked against guesses are safe when they are checked where they are used.**

(C) Shigeng Feng • Redistribution prohibited without permission

Why It Can Fail: Three Ways



- Stop trying: a street slow once is never driven again, and the data stay silent on it. Cure: a worse-looking turn now and then; free in a simulator, costly in an economy.
- Guesses chase each other: a rule of thumb makes one fix move every guess. Correct A against B's guess, never B itself: each fix lifts B, which lifts A's target: a runaway.
- Wrong score: it does what you wrote, not what you meant. Corners a minute? It circles.
- **Three dangers, three remedies to come: keep trying (Deck 3.2); close the loop or freeze the target (Decks 3.3–3.4); grade against a known answer (Deck 3.5).**

© Zhigang Feng • Redistribution prohibited without permission

The Same Story in This Course's Words

the story	the word	the growth model
a corner	state	capital and productivity, (k, z)
a turn	action	how much to save, k'
minutes spent	reward	this period's utility, $u(c)$
minutes to go	value	lifetime utility from here, $V(k, z)$
the city	the environment	the model, stepped one period
the map of turns	what is known	the law of motion of capital
the times and the traffic	what is learned	the shocks; later, the distribution's motion

- The driver knows the map and learns the rest. Here the law of motion is known; the shocks are learned.
- **Session 3 in one line: two habits (Decks 3.2–3.3), a rule of thumb (Deck 3.4), every method on one model (Deck 3.5). Three verdicts next: on or off; what we use RL for; why not in the economy.**

© Zhigang Feng • Redistribution prohibited without permission

On-Policy or Off-Policy: Neither Is Better

	on-policy	off-policy
whose value	the rule you run, detours included	the best rule, or any rule you name
data	its own fresh drives only	anyone's: old records, a replay memory
the best rule	reached as the detours fade	reached while still detouring
noise	an average's	a max or reweighting: optimistic
a rule of thumb	stable: every yardstick corrected	can chase itself (p. 6)
needs	freedom to run the rule and tremble	overlap: data that do what it would

- On wins when you can run the rule and tremble freely; a rule of thumb stays stable.
- Off-policy wins when data are scarce or someone else's, or the best rule is wanted now.
- **An accurate value of a slightly wrong rule, or a noisy value of the right rule from anyone's data (Deck 3.2). In an economy the choice is made for you: off.**

© Jiarang Feng · Redistribution prohibited without permission

How Economists Use Reinforcement Learning: Four Ways

a solver in a written model	**a model of behaviour**	**policy evaluation** from data	**adaptive experiments**
heterogeneous agents, a distribution in the state	agents who learn; pricing algorithms that collude	overlap, inverse probability weights, policy learning	bandits on platforms: tests that adapt

© Zhirang Feng · Redistribution prohibited without permission

- The mathematics was ours first: Bellman, Howard, Monte Carlo, adaptive expectations.
- As a solver the model is the simulator, trembles free, drives unlimited: RL scales a model too big for a grid (Maliar–Maliar–Winant; Azinovic–Gaegauf–Scheidegger).
- As behaviour: learning, experimenting agents; algorithms that collude (Calvano et al.).
- As evaluation: RL's coverage is overlap; same weights, same fixes (Athey–Wager).
- **RL's mathematics is ours; its algorithms are becoming our solvers; its one new idea is used where the environment is a model (Deck 3.5).**

Why Not in the Real Economy: It Is Not a Simulator

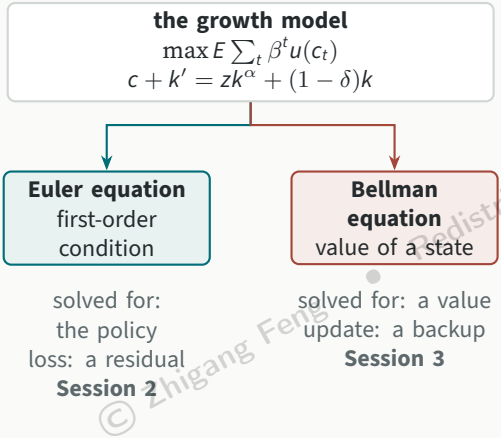
a simulator
a reset button, free trembles, unlimited drives; responses fixed: physics, dice, traffic

the economy
one history, once; a tremble is a policy error; decades for one rule's data; responses move with the rule

- A central bank runs one rule, once, for real: it cannot tremble.
- So any new rule is valued off-policy, with thin overlap where the old rule never did what the new one would; and Lucas: even with overlap, the responses were the old regime's.
- Identification needs a model: the data saw one rule; only an assumption separates what stays from what moves. Estimate it, re-solve, simulate (Clarida–Galí–Gertler).
- Engineering has the data limits, not Lucas: a turbine expects nothing of its controller, so old data and simulators stay faithful; RL's wins are games and simulated robots.
- **RL scales the solution of a model; it does not replace the model. Now the road in from Session 2, and the three challenges.**

© Zhigang Feng Redistribution prohibited without permission

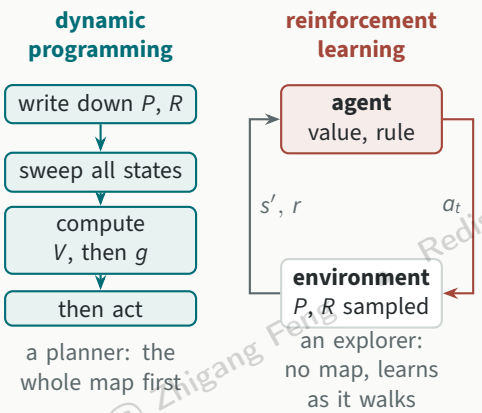
Session 2 Took the Euler Road. The Bellman Equation Is the Other One



© Zhigang Feng — Redistribution prohibited without permission

- Session 2 trained a policy network on Euler residuals, never a value.
- The Bellman equation describes the same optimum through the *value* of each state.
- It needs no derivatives: discrete choices and binding constraints are fine.
- **This deck takes that road and meets three challenges: the state grows, the backup looks further ahead, the law of motion has no closed form.**

Two Ways to Solve the Same Bellman Equation: Model First, or Learn by Acting

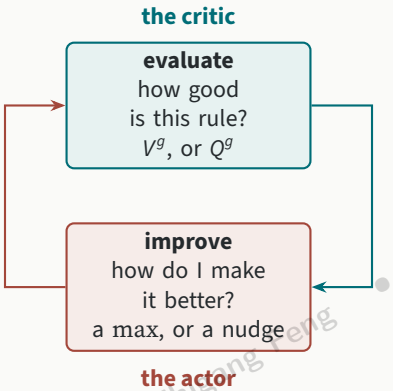


- Dynamic programming writes down P and R, sweeps every state, computes V, then acts.
- **Reinforcement learning** acts, sees a reward and a new state, improves, acts again.
- **RL's one idea**: learn from realised (s, a, r, s'), one at a time, made by its own rule. A learner that reads only these, never u or P_z, is **model-free**.
- **RL fills the expectation with samples: each challenge ahead leaves only states and transitions along walked paths.**

© Zhigang Feng Redistribution prohibited without permission

Source: R. Sutton, A. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., 2018, ch. 1.

Two Questions, Two Learners



- Evaluate to convergence, then improve: Howard's (1960) **policy iteration**. One backup, then improve: value function iteration.
- The **critic** evaluates; the **actor** improves. **A structure, not a source of data.**
- Methods differ in what each learner reads (the model, or experience) and how far ahead it looks (one period, T, the whole path).
- **Every later deck answers both in turn, on different data: Deck 3.2 from whole paths, 3.3 from single steps, 3.4 with the model in hand, then with a network per learner. They return once more with the Third Answer (p. 25).**

© Zhengyang Feng
Redistribution prohibited without permission

Session 3.1

The Benchmark, and What It Requires

The growth model's Bellman equation, and the two things dynamic programming needs to solve it



© Zhigang Feng · Redistribution prohibited without permission

The Growth Model: a Sequence Problem, and Its Recursive Form

sequence problem

$$\max E_0 \sum_t \beta^t \ln c_t$$
$$c_t + k_{t+1} = y(k_t, z_t)$$

one function, not a path

Bellman equation

$$V = \max_{k'} \{\ln c + \beta E[V' \mid z]\}$$

$y(k, z) = z f(k) + (1 - \delta)k$, $f(k) = k^\alpha$: output plus capital left

- Sequential, coupled, uncertain; yet today's utility plus the same problem from tomorrow, since z is Markov: one equation.
- k real, so is k'; or a **saving share** $a \in (0, 1)$: $k' = a\,y$, $c = (1 - a)y$. Two shocks, 0.9 and 1.1, staying w.p. 0.8.
- T^*, the **Bellman operator** (the star: a max inside), contracts with modulus β (Session 1.4): V^* unique, iteration converges, g^* its arg max.

© Zhuang Feng Redistribution prohibited without permission

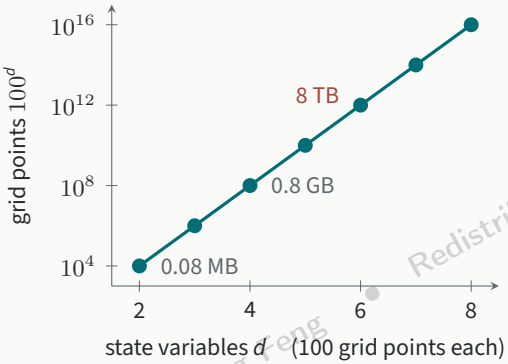
What Dynamic Programming Quietly Requires

Requirement	In value iteration, each round	Where it fails
a table, swept	V kept at every grid point of (k, z), interpolated off it; every point updated, every round	m^d entries with d state variables: too many to store, too many to sweep
the model: score, average	P_z and the law of motion inside $E[V' \mid z]$: each k' scored, tomorrow averaged	a sum too large to take; a law of motion that is not known

- Two requirements: a store that can be swept, and a model to score and average.
- In two dimensions neither binds: DP runs, and V^*, g^* are the **answer key**; the **grade** of a learned rule is the consumption given up for ever to hold it instead.
- The store breaks when the state grows (Challenge 1); the average, when the backup looks further ahead (Challenge 2); the model, when the law of motion has no closed form (Challenge 3).
- **Three challenges, three answers: a network for the table, a walk for the sum, a simulated economy for the model. One at a time, next.**

© Albert Pell. Redistribution prohibited without permission

Challenge 1: the Curse of Dimensionality in the State, Which RL Still Faces

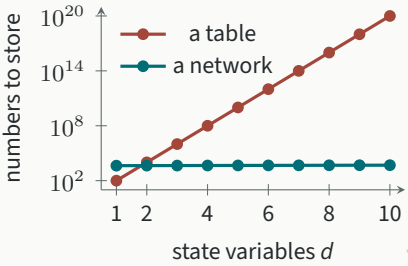


© Zhigang Feng
Redistribution prohibited without permission

- m points per dimension: m^d entries; one more variable multiplies them by m.
- Go: rules on one page, about 10^{170} positions. A known model, and no table.
- *Sampling changes which states get visited, not the table's size.*
- **A table shares nothing between its entries. The first answer: a function, where one parameter serves many states.**

Source: Memory at 8 bytes per entry; the count of legal Go positions: J. Tromp, 2016.

First Answer: a Network for the Table

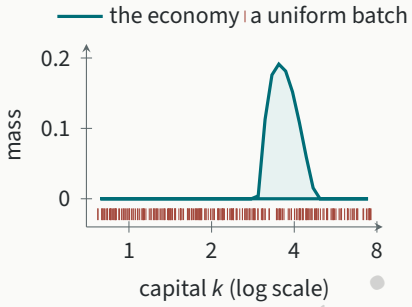


- A table needs a number for every state; a network grows with d only in its first layer.
- Fitted by regression on a sample of states, it needs to be right only where the sample falls: next page.
- **Projection** (a fit on a basis) is economists' version; **deep DP** (by networks) the same tool at larger d. A poor form spreads its errors (Deck 3.4).
- **First answer: a network for the table. Its size grows with d, not m^d, and its fit is paid only where it is needed.**

Source: Table: 100 points a variable, 8 bytes a number. Network: d inputs, two hidden layers of 64 units, one output.

© Zhigang Feng
Redistribution prohibited without permission

The First Answer, Completed: Fit It Where the Economy Goes

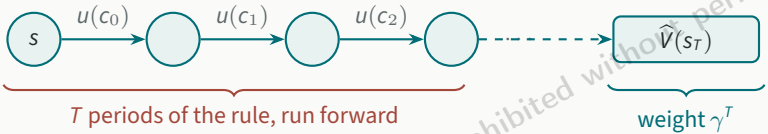


- A uniform box is cursed too: six times wider in $\ln k$ than where the economy spends 98 % of its time; 6^d times the volume in d dimensions.
- **Simulate, burn in, draw**: the rule's path picks the states, as **ergodic-set methods** do.
- Fitted on that region, not the box: the cost of approximation is the region's size.
- Its price: rare states fit worse, and the sample moves with the rule (Deck 3.4).
- **Challenge 1 answered twice: a network, fitted where the economy goes, not the box.**

© Zhigang Feng
Redistribution prohibited without permission

Source: Session 3 lab, block B: the long-run distribution of k under the learned rule, $z = 0.9$; ticks: one uniform batch of 256 states.

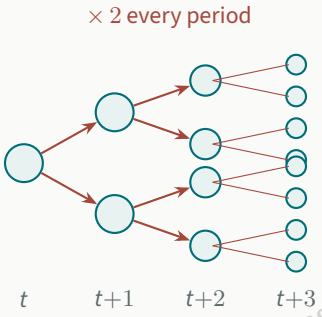
Challenge 2: Why Look Further Ahead



- Value iteration's **one-period backup**, $B(s) = u(c) + \gamma \sum_{z'} P_z(z' \mid z)\, \widehat{V}(k', z')$: today's utility is real; the rest is last round's guess, at weight $\gamma = 0.95$.
- A classical speed-up: between two maximisations, run the current rule T periods forward, $B_T(s) = E\left[\sum_{t<T} \gamma^t u(c_t) + \gamma^T \widehat{V}(s_T) \,\middle|\, s_0 = s\right]$: the guess's weight falls to γ^T, 0.77 at $T = 5$, 0.36 at $T = 20$. This is **modified policy iteration**, Howard's rollout.
- Only the evaluation's target lengthens; the maximisation is untouched. Fewer rounds, each trusting fewer numbers.
- **But the T-period expectation runs over T periods of shocks. From one state, the uncertainty unfolds: a tree.**

© Zhigang Feng • Redistribution prohibited without permission

The Tree: Uncertainty Unfolds, Two Branches a Period

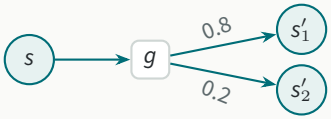


- One period ahead the backup has n_z branches, weighted by P_z: two here.
- T periods ahead, n_z^T shock histories: 2^T here, and $2^{20} \approx 10^6$ at $T = 20$. With d_z shocks of n_z values each the base is $n_z^{d_z}$: every shock added multiplies every level.
- Reading the tree exactly costs $2^{T+1} - 2$ evaluations of the rule and the model per state: 62 at $T = 5$, two million at $T = 20$.
- **A second curse, in time: the lookahead's cost grows as n_z^T, as the table's grew as m^d. The second answer: sample it.**

© Zhigang Feng
Redistribution prohibited without permission

Second Answer: Sample the Tree by Walking It

enumerate: every branch, weighted



run: one branch, drawn



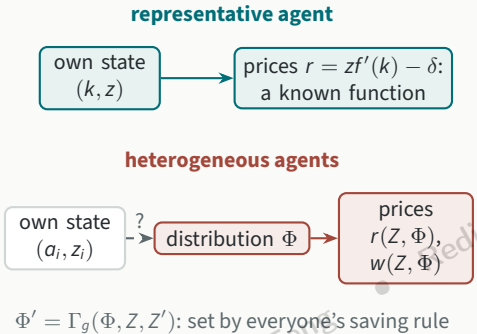
	enumerate	**run**
reads	P_z	realised (r, s')
T ahead	$2^{T+1} - 2$	$N \times T$
error	none	$\sigma_G/\sqrt{N}$

σ_G: one run's spread

- A **sample backup**: run the rule N times for T periods, drawing z', and average: cost $N \cdot T$, error $\sigma_G/\sqrt{N}$, free of n_z^T and of d.
- No P_z is read: the simulator is a black box returning (r, s'); with the model known, it is the model stepped once.
- **The first time the learner acts**: **the walk**, forced by the lookahead (Decks 3.2–3.3 run it).
- **RL's one idea (p. 12): learn from transitions the rule itself makes; with the model known, another path, not a last resort. The price: variance.**

© Zhigang Feng
Redistribution prohibited without permission

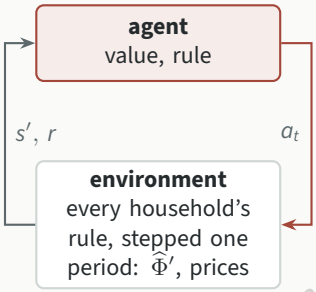
Challenge 3: an Unknown Law of Motion for the Distribution



- Heterogeneous agents, aggregate shocks: prices depend on the distribution Φ; to forecast r' is to forecast Φ', so Φ enters the state.
- Γ_g, its law of motion, has no closed form in most models and depends on the rule solved for; yet a period can always be run (Deck 3.5).
- **The most severe: not a cost, a missing input. Every primitive is written, yet part of the environment, how Φ and prices move, is unknown; in Session 4 usually truly unknown to the household.**

© Zhigang Feng Redistribution prohibited without permission

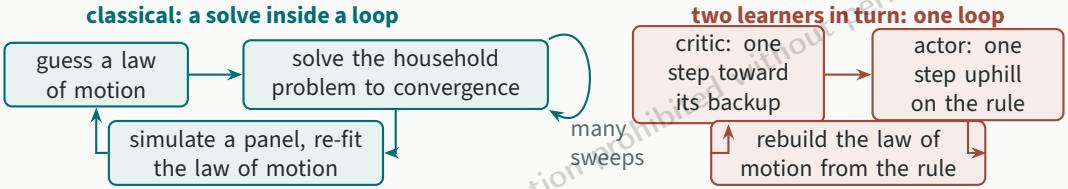
Third Answer: Learn the Environment by Running It



- No closed form, yet it runs: step every rule once, and Φ' and the prices are realised.
- A learner reading only (s, a, r, s') never needs Γ_g: the simulation stands in for it.
- It learns the law of motion where the rule goes, and re-learns it as the rule changes.
- The price: the data move with the rule; each step changes tomorrow's sample (Deck 3.4).
- **Third answer: learn the environment by simulating it; the walk is the only source.**

© Zhigang Feng • Redistribution prohibited without permission

Two Learners in Turn, So the Fixed Point Becomes Training



- Classical Krusell–Smith nests two fixed points: the household problem, solved to convergence, inside a loop that re-fits the law of motion from a simulated panel.
- Two learners in turn take one step each, the actor along a derivative (Deck 3.4); the law of motion is rebuilt by stepping the current rule: no inner solve, no outer loop.
- The three answers compose: a network that takes Φ as input (p. 18), a walk instead of a sum (p. 22), and alternation that turns the fixed point into training.
- **The third answer, made efficient: a step per learner and one simulated period, never a solve.**

© Jesse Perla. Unauthorized distribution prohibited without permission

Session 3.1

One Problem, Two Languages

The Markov decision process, its dictionary, and the two value functions every later deck uses



© Zhigang Feng · Redistribution prohibited without permission

The Markov Decision Process

- A **Markov decision process** (MDP) is a five-tuple
$$\mathcal{M} = \langle \mathcal{S},\ \mathcal{A},\ P,\ R,\ \gamma \rangle.$$
- **States** $\mathcal{S}$; feasible **actions** $\mathcal{A}(s)$; **transition** probabilities $P(s' \mid s, a)$; expected **reward** $R(s, a)$; **discount factor** $\gamma \in [0, 1)$, the economist's β from here on.
- The **Markov property**, $P(s_{t+1} \mid s_t, a_t, s_{t-1}, a_{t-1}, \ldots) = P(s_{t+1} \mid s_t, a_t)$: the current state carries everything needed to forecast.
- P: a law of motion plus shocks, or chance of its own (a job offer). DP assumes P known; RL lets it be unknown.
- **Markov makes a problem recursive, not computable: enlarge the state (habit: add c_{t-1}) and the three challenges remain.**

Source: M. Puterman, *Markov Decision Processes*, 1994.

© Zhaonan Feng · Redistribution prohibited without permission

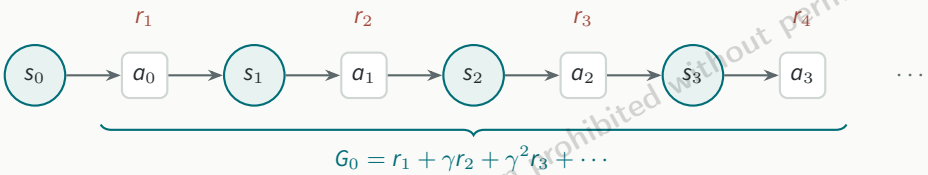
One Dictionary: the MDP and the Dynamic Economic Model

MDP	Economics	Example
state s	state variables	capital k, wealth, productivity z
action a	decision, choice variable	next capital k', the policy rate
transition $P(s' \mid s, a)$	law of motion, shocks	k' chosen, or $a\,y$; $z' \sim P_z(\cdot \mid z)$
reward $R(s, a)$	period utility or profit	$u(c)$
discount γ	discount factor β	0.99 quarterly, 0.95 annual
policy $g(a \mid s)$	decision rule	$k' = g(k, z)$
value $V^g(s)$	value function	lifetime utility $V(k, z)$
episode	a finite life, or a simulated path	a T-period run

© Zhiyang Fang. Redistribution prohibited without permission

- The policy is g (RL's π); *control* here means improving the rule (Deck 3.2), not the action.
- **Rule, policy, decision rule** are one object: the actor is the rule a learner improves, the critic the value it keeps; Session 2's share s is here the action a.
- **The heterogeneous-agent household is this five-tuple with state $(a_i, z_i; Z, \Phi)$ and Γ_g inside P: Markov in form, infinite-dimensional to compute (p. 23).**

Trajectory and Return



- A **trajectory** is one simulated path. The reward r_{t+1} is period t's utility $u(c_t)$; the index says when it is seen.
- The **return** is the discounted utility stream, $G_t = \sum_{j \geq 0} \gamma^j r_{t+j+1}$.
- It is recursive: $G_t = r_{t+1} + \gamma\, G_{t+1}$. Take expectations and the Bellman equation appears.
- Only T periods simulated: the **partial return** $G_{t:t+T} = \sum_{j<T} \gamma^j r_{t+j+1}$ (Decks 3.2–3.3).
- **A return is one random draw of lifetime utility. A value is its expectation.**

Source: Sutton & Barto, *Reinforcement Learning*, 2018, §3.3.

© Zhigang Feng Redistribution prohibited without permission

Two Value Functions: V and Q

- The **state-value function**: expected return from state s when following g,
$$V^g(s) = E_g\left[\, G_t \mid s_t = s \,\right].$$
- The **action-value function**, or **Q function**: take action a first, follow g afterwards; on a grid a table, one row per state (Deck 3.2 prints Q^*),
$$Q^g(s,a) = E_g\left[\, G_t \mid s_t = s,\ a_t = a \,\right].$$
- Linked by $V^g(s) = \sum_a g(a \mid s)\, Q^g(s,a)$. The **Bellman expectation equation** evaluates a *given* rule, with no $\max$. Call its right-hand side $T^g V$:
$$V^g(s) = R(s, g(s)) + \gamma \sum_{s'} P(s' \mid s, g(s))\, V^g(s').$$
- T^g is a contraction of modulus γ, as T^* is (p. 15). With V alone, improving needs the expectation for every action; one sample shows one, so a sampling learner keeps Q, or an actor.
- **The Bellman equation never changes this session; only how its expectation is filled: summed, or sampled from the walk.**

© Zhigang Feng • Redistribution prohibited without permission