Reinforcement Learning for Economists

Lecture 1

The Basic Idea of RL, Link to Dynamic Programming

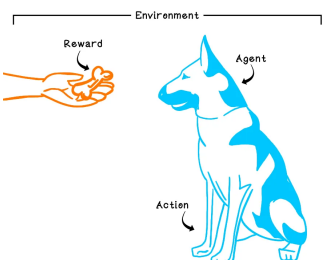
Benjamin Moll

London School of Economics

RL: learning value & policy functions in incompletely-known Markov decision processes from Monte Carlo simulation

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou
Daan Wierstra Martin Riedmiller
DeepMind Technologies
{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

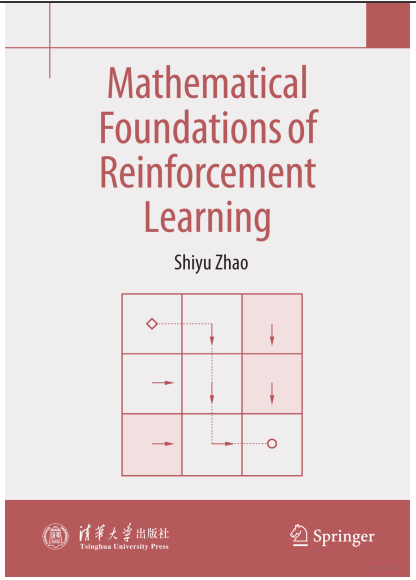


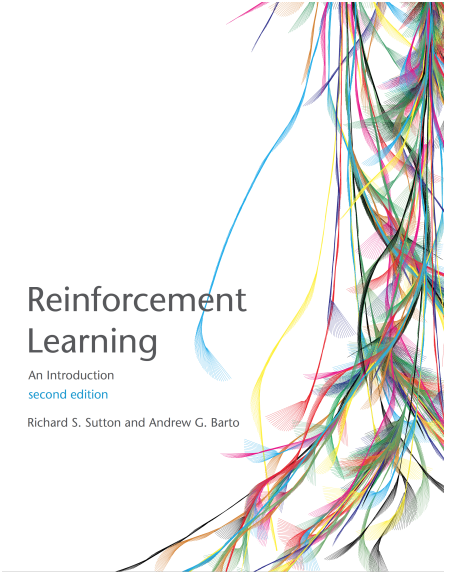
RL in a nutshell for economists...
in terms of two tools you should already know

Reinforcement Learning =

Howard policy iteration + Monte Carlo simulation

References


Mathematical Foundations of Reinforcement Learning
Shiyu Zhao
清華大學出版社
Tsinghua University Press
Springer


Reinforcement Learning
An Introduction
second edition
Richard S. Sutton and Andrew G. Barto

References

Also use Sargent-Stachurski “Dynamic Programming, Volume I: Finite States”

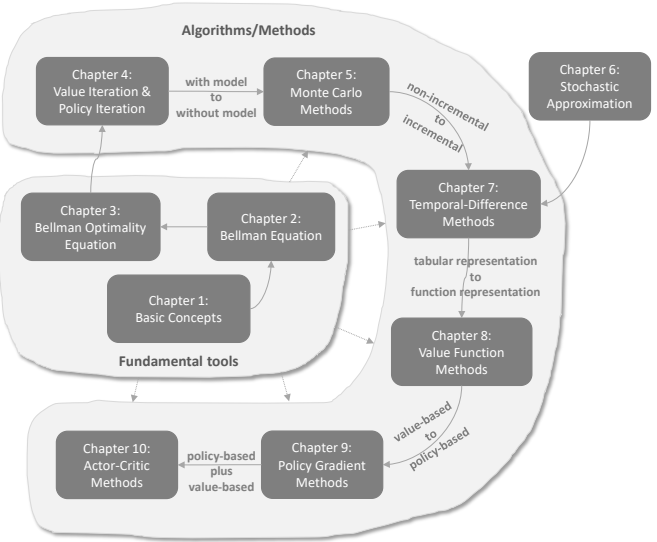
All three books available for free online:

1. https://github.com/MathFoundationRL/Book-Mathematical-Foundation-of-Reinforcement-Learning
2. http://www.incompleteideas.net/book/the-book.html
3. https://dp.quantecon.org/

Other recommended materials:

- Video recordings of David Silver’s 2015 RL course
- Murphy (2025) “Reinforcement Learning: An Overview”, less accessible
- Lilian Weng (2018) on policy gradient methods
- Rawat (2026) “A Survey of Reinforcement Learning For Economics”

Outline of Shiyu Zhao's book



Plan for the four RL lectures

1. The basic idea of RL
2. Dynamic programming revisited
   - Value function iteration
   - Howard policy iteration
3. Monte Carlo methods
4. Stochastic approximation
5. Temporal difference learning, Sarsa, Q-learning
6. Policy gradient and actor-critic methods

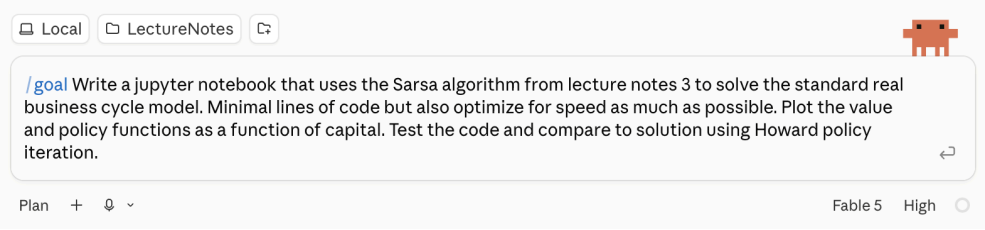
Computational infrastructure

JAX = NumPy on Steroids for GPUs `https://docs.jax.dev/`

Google Colab = cloud-based Jupyter notebooks with free GPU access
`https://colab.google/`

Example codes?

Not including any on purpose – **just ask your favorite coding agent!**



one-shotted this https://github.com/ben-moll/RL_lectures_example_codes/blob/main/sarsa_rbc.ipynb which seems pretty good

But also see all the codes you can find online, e.g.:

- Zhao book github
  https://github.com/MathFoundationRL/Book-Mathematical-Foundation-of-Reinforcement-Learning
- Rawat survey github
  https://github.com/rawatpranjal/survey-of-reinforcement-learning-in-economics

Example codes?

Examples of RL applications with codes from my own work

SRL Tutorial

This repo presents a pedagogical JAX implementation of policy-gradient methods for heterogeneous-agent macro models in general equilibrium, including with aggregate risk.

#	Notebook	Open in Colab	Colab G4 runtime	Free Colab T4 runtime
0	Household problem, in NumPy	Open in Colab	~1 min	~3 min
1	The same problem, in JAX	Open in Colab	~6 s	~15 s
2	The same problem, by policy gradient	Open in Colab	~30 s	~1 min

https://github.com/StructuralRL/SRL

⚡ MFAX: Mean-Field Games in JAX ⚡

https://clarisse-wibault.github.io/rspg/

The basic idea of RL

Computing an expected value

Random variable x

How compute expected value $\mathbb{E}[x]$? Two approaches:

1. **Exact:** know probability distribution $p(x) \Rightarrow$ calculate

$$\mathbb{E}[x] = \int x p(x) dx$$

2. **Monte Carlo:** don't know p but can sample $\{x_1, x_2, \ldots, x_N\}$

$$\mathbb{E}[x] \approx \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Or update incrementally (stochastic approximation method):

$$\bar{x}_k = \frac{1}{k} \sum_{i=1}^{k} x_i \quad \text{satisfies} \quad \bar{x}_k = \bar{x}_{k-1} + \frac{1}{k} \left(x_k - \bar{x}_{k-1} \right), \quad \frac{1}{k} = \text{“learning rate”}$$

Computing a value function

For now: eliminate actions $\Rightarrow$ "Markov reward process" (MRP)

$$v(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t r(s_t)\right], \quad s_t = \text{state} = \text{exogenous Markov process}$$

Two approaches:

1. **Dynamic programming:** know Markov transition probabilities $p(s'|s)$

$$v(s) = r(s) + \beta \int v(s')p(s'|s)ds'$$

Computing a value function

For now: eliminate actions $\Rightarrow$ "Markov reward process" (MRP)

$$v(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t r(s_t)\right], \quad s_t = \text{state} = \text{exogenous Markov process}$$

Two approaches:

1. **Dynamic programming:** know Markov transition probabilities $p(s'|s)$

$$v(s) = r(s) + \beta \int v(s')p(s'|s)ds'$$

2. **Monte Carlo:** don't know p but sample N trajectories $\left\{s_t^i\right\}_{t=0}^{T}$

$$v(s_0) \approx \widehat{v}(s_0) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \beta^t r(s_t^i)$$

Computing a value function

For now: eliminate actions $\Rightarrow$ "Markov reward process" (MRP)

$$v(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t r(s_t)\right], \quad s_t = \text{state} = \text{exogenous Markov process}$$

Two approaches:

1. **Dynamic programming:** know Markov transition probabilities $p(s'|s)$

$$v(s) = r(s) + \beta \int v(s')p(s'|s)ds'$$

2. **Monte Carlo:** don't know p but sample N trajectories $\left\{s_t^i\right\}_{t=0}^{T}$

$$v(s_0) \approx \widehat{v}(s_0) = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \beta^t r(s_t^i)$$

These slides: extend to compute optimal policy (like Howard policy iteration)

Computing a value function

For now: eliminate actions $\Rightarrow$ "Markov reward process" (MRP)

$$v(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t r(s_t)\right], \quad s_t = \text{state} = \text{exogenous Markov process}$$

Two approaches:

1. **Dynamic programming:** know Markov transition probabilities $p(s'|s)$

$$v(s) = r(s) + \beta \int v(s')p(s'|s)ds'$$

2. **Temporal difference learning:** N trajectories, also update v at each t

$$\widehat{v}_{t+1}^k(s_t^k) = \widehat{v}_t^k(s_t^k) + \alpha\left[\left(r(s_t^k) + \beta\widehat{v}_t^k(s_{t+1}^k)\right) - \widehat{v}_t^k(s_t^k)\right]$$

These slides: extend to compute optimal policy (like Howard policy iteration)

Reinforcement learning and deep learning

Sometimes (Lecture 2) we will want to approximate value function

$$\widehat{v}(s, w)$$

where w = parameter vector

- one way: (deep) neural networks, in which case $w =$ "weights" of NN
- DNNs = basic stuff $\Rightarrow$ look it up, if you don't know it already

RL + deep NN = "deep reinforcement learning"

But important: RL and DNNs are orthogonal / completely separate tools

- say like Bellman equations and DNNs

Link to Dynamic Programming

Bellman equation with finite states

Stochastic optimal control problem or “Markov decision process” (MDP)

$$v(s_0) = \max_{\{a_t\}_{t=0}^{\infty}} \mathbb{E}_0 \left[\sum_{t=0}^{\infty} \beta^t r(s_t, a_t) \right] \quad \text{s.t.} \quad s_{t+1} \sim p(\cdot|s_t, a_t), \quad s_0 \text{ given}$$

- $s \in \mathcal{S} = \{s_1, \ldots, s_n\}$ = state
- $a \in \mathcal{A}$ = action (control)
- $\beta \in (0, 1)$ = discount factor
- $r(s, a)$ = reward function
- $p(\cdot|s, a)$ = transition probabilities of s

Goal: find optimal policy $a = \pi(s)$

Same as section 5.1.1 in Sargent-Stachurski but they use x in place of s

Example: neoclassical growth model with finite grid

Growth model but force capital to live on a finite grid $\mathcal{K} = \{k_1, ..., k_n\}$

$$v(k_0) = \max_{\{c_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u(c_t) \quad \text{s.t.} \quad k_{t+1} = f(k_t) + (1-\delta)k_t - c_t, \quad k_{t+1} \in \mathcal{K}$$

Special case of MDP on previous slide

- state $s = k$
- action $a = c$
- reward $r(s, a) = u(a)$
- transition probabilities

$$p(s'|s, a) = \begin{cases} 1, & \text{if } s' = f(s) + (1-\delta)s - a, \quad s' \in \mathcal{K} \\ 0, & \text{otherwise} \end{cases}$$

See Sargent-Stachurski for many other examples, e.g. McCall search model with finite wage distribution

Bellman equation with finite states

$$v(s) = \max_{a \in \mathcal{A}} \left\{ r(s,a) + \beta \sum_{s' \in \mathcal{S}} v(s') p(s'|s,a) \right\}$$

- s = state
- a = action (control)
- $r(s,a)$ = reward function
- $p(s'|s,a)$ = transition probabilities of s

Same as equation (5.2) in Sargent-Stachurski but they use x in place of s

Almost same as equation (3.1) in Zhao / (3.19) in Sutton-Barto but they use γ in place of β and allow for stochastic rewards r and stochastic policy $\pi(a|s)$

$$v(s) = \max_{\pi} \sum_{a \in \mathcal{A}} \pi(a|s) \left(\sum_{r \in \mathcal{R}} p(r|s,a) r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v(s') \right)$$

Useful later: "action-value function" or "Q function"

$$v(s) = \max_{a \in \mathcal{A}} \left\{ r(s,a) + \beta \sum_{s' \in \mathcal{S}} v(s') p(s'|s,a) \right\}$$

Can write this as

$$v(s) = \max_{a \in \mathcal{A}} q(s,a) \quad \text{where} \quad q(s,a) = r(s,a) + \beta \sum_{s' \in \mathcal{S}} v(s') p(s'|s,a)$$

is the "action-value function" or "Q function"

$q(s,a)$ = value of taking action a in state s, following optimal policy π^* thereafter

$$q(s,a) = r(s,a) + \mathbb{E}_{\pi^*} \left[\sum_{\tau=1}^{\infty} \beta^\tau r(s_{t+\tau}, a_{t+\tau}) \middle| s_t = s, a_t = a \right]$$

Given knowledge of $q(s,a)$ can always find optimal value and policy functions

$$v(s) = \max_{a \in \mathcal{A}} q(s,a), \qquad \pi(s) = \arg\max_{a \in \mathcal{A}} q(s,a)$$

Will use this heavily in later lectures!

Matrix-vector form of Bellman equation

$$v(s) = \max_a \left\{ r(s,a) + \beta \sum_{s' \in \mathcal{S}} v(s') p(s'|s,a) \right\}$$

Index states as s_i, $i = 1, \ldots, n$:

$$v(s_i) = \max_a \left\{ r_a(s_i) + \beta \sum_{s_j} v(s_j) p_a(s_j|s_i) \right\}$$

Put all these equations for all states together and rewrite to matrix-vector form:

$$\mathbf{v} = \max_a \{ \mathbf{r}_a + \beta \mathbf{P}_a \mathbf{v} \}$$

where maximization is componentwise, where

$$\mathbf{v} = \begin{bmatrix} v(s_1) \\ \vdots \\ v(s_n) \end{bmatrix}, \qquad \mathbf{r}_a = \begin{bmatrix} r_a(s_1) \\ \vdots \\ r_a(s_n) \end{bmatrix}$$

and where $\mathbf{P}_a$ is $n \times n$ transition matrix with entries $p_a(s_j|s_i)$

Value function iteration and Howard policy iteration

Bellman equation

$$\mathbf{v} = \max_a \{\mathbf{r}_a + \beta \mathbf{P}_a \mathbf{v}\}$$

Next discuss two algorithms for solving it

- Value function iteration
- Howard policy iteration

Value function iteration:

$$\mathbf{v}_{k+1} = \max_a \{\mathbf{r}_a + \beta \mathbf{P}_a \mathbf{v}_k\}, \quad k = 0, 1, 2, ...$$

where v_0 can be arbitrary

Howard policy iteration

Howard policy iteration = two-step procedure

1. “policy evaluation”: compute value function for given (suboptimal) policy
2. “policy improvement”: improve policy

Preview: all RL algorithms start from Howard policy iteration

They replace step 1 = “policy evaluation” by some sort of Monte Carlo method for estimating v

- can do this without knowledge of $\mathbf{P}_\pi$ (“model free”)

All RL algorithms do some version of this, have two-step structure

Howard policy iteration: algorithm description

Given **initial policy** $\boldsymbol{\pi}_0$,

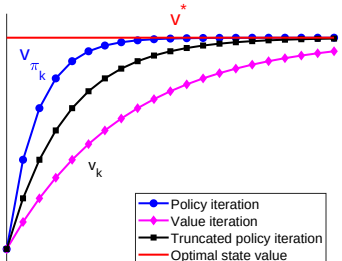
- **Step 1: policy evaluation (PE)** = calculate value of following policy π_k:

$$\mathbf{v}_{\pi_k} = \mathbf{r}_{\pi_k} + \beta \mathbf{P}_{\pi_k} \mathbf{v}_{\pi_k}$$

- **Step 2: policy improvement (PI)** (note: the maximization is componentwise)

$$\boldsymbol{\pi}_{k+1} = \arg\max_{\pi} (\mathbf{r}_{\pi} + \beta \mathbf{P}_{\pi} \mathbf{v}_{\pi_k})$$

Howard policy iteration often yields faster convergence than value iteration



Howard policy iteration algorithm

Question: In step 1 (policy evaluation), how get values $\mathbf{v}_{\pi_k}$ satisfying

$$\mathbf{v}_{\pi_k} = \mathbf{r}_{\pi_k} + \beta \mathbf{P}_{\pi_k} \mathbf{v}_{\pi_k} ?$$

- Closed-form solution:

$$\mathbf{v}_{\pi_k} = (I - \beta \mathbf{P}_{\pi_k})^{-1} \mathbf{r}_{\pi_k}$$

- Iterative solution:

$$\mathbf{v}_{\pi_k}^{(j+1)} = \mathbf{r}_{\pi_k} + \beta \mathbf{P}_{\pi_k} \mathbf{v}_{\pi_k}^{(j)}, \quad j = 0, 1, 2, \ldots$$

Summary: value and policy iteration side by side

- Value iteration: given initial value $\mathbf{v}_0$

$$\mathbf{v}_{k+1} = \max_{\pi}(\mathbf{r}_\pi + \beta \mathbf{P}_\pi \mathbf{v}_k)$$

- Howard policy iteration: given an initial policy $\boldsymbol{\pi}_0$

$$\begin{cases} \text{Policy evaluation: } \mathbf{v}_{\pi_k} = \mathbf{r}_{\pi_k} + \beta \mathbf{P}_{\pi_k} \mathbf{v}_{\pi_k} \\ \text{Policy improvement: } \boldsymbol{\pi}_{k+1} = \arg\max_{\pi}(\mathbf{r}_\pi + \beta \mathbf{P}_\pi \mathbf{v}_{\pi_k}) \end{cases}$$

Policy improvement and action-value function (Q function)

Step 2: policy improvement (PI)

$$\boldsymbol{\pi}_{k+1} = \arg\max_{\pi}(\mathbf{r}_{\pi} + \beta \mathbf{P}_{\pi}\mathbf{v}_{\pi_k})$$

Element-wise version

$$\pi_{k+1}(s) = \arg\max_{\pi}\left(r(s,\pi) + \beta \sum_{s' \in \mathcal{S}} v_{\pi_k}(s')p(s'|s,\pi)\right)$$

or

$$\pi_{k+1}(s) = \arg\max_{\pi} \; q_{\pi_k}(s,\pi) \quad \text{where}$$

$$q_{\pi_k}(s,a) = r(s,a) + \beta \sum_{s' \in \mathcal{S}} v_{\pi_k}(s')p(s'|s,a)$$

is "action-value function" or "Q function" under policy π_k

In matrix-vector form:

$$\boldsymbol{\pi}_{k+1} = \arg\max_{\pi} \; \mathbf{q}_{\pi,\pi_k} \quad \text{where} \quad \mathbf{q}_{a,\pi_k} = \mathbf{r}_a + \beta \mathbf{P}_a \mathbf{v}_{\pi_k}$$

Preview: all RL algorithms

- All RL algorithms start from Howard policy iteration for Bellman equation
- Recall two steps of Howard policy iteration

  1. Policy evaluation: $\mathbf{v}_{\pi_k} = \mathbf{r}_{\pi_k} + \beta \mathbf{P}_{\pi_k} \mathbf{v}_{\pi_k}$
  2. Policy improvement: $\boldsymbol{\pi}_{k+1} = \arg\max_{\pi} (\mathbf{r}_{\pi} + \beta \mathbf{P}_{\pi} \mathbf{v}_{\pi_k})$

- Key idea of RL: replace step 1 = "policy evaluation" by some sort of Monte Carlo method for estimating value function $\mathbf{v}_{\pi_k}$
  - can do this without knowledge of $\mathbf{P}_{\pi}$ ("model free")
- All RL algorithms do some version of this, have two-step structure
  - Sutton-Barto: "Almost all RL methods are well described as generalized policy iteration"
- Rest: implementation details to ensure this actually works in practice

Preview: all RL algorithms

- All RL algorithms start from Howard policy iteration for Bellman equation
- Recall two steps of Howard policy iteration

  1. Policy evaluation: $\mathbf{q}_{a,\pi_k} = \mathbf{r}_a + \beta \mathbf{P}_a \mathbf{v}_{\pi_k}$
  2. Policy improvement: $\boldsymbol{\pi}_{k+1} = \arg\max_{\pi} \mathbf{q}_{\pi,\pi_k}$

- Simplest version of RL: replace step 1 = "policy evaluation" by some sort of Monte Carlo method for estimating Q function $\mathbf{q}_{a,\pi_k}$ – see Lecture 2
  - can do this without knowledge of $\mathbf{P}_\pi$ ("model free")
- All RL algorithms do some version of this, have two-step structure
  - Sutton-Barto: "Almost all RL methods are well described as generalized policy iteration"
- Rest: implementation details to ensure this actually works in practice