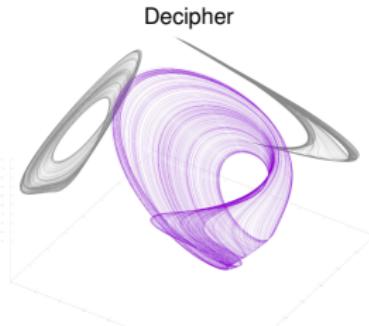


Introduction to Reinforcement Learning

Lionel MATHELIN
LISN-CNRS



Credits

Niao He, Yann Ollivier, Emmanuel Rachelson, Michèle Sebag, Onofrio Semeraro

Illustrations

Wikipedia, Forbes, Shelly Palmer, Discovery Magazine, SoftBank Robotics, Encyclopædia Britannica, Tom Goldstein, Upscaleluxury, H. van Hasselt, John Schulman, Steve Wu



Teaching team

- ▶ Instructor: **Lionel MATHELIN**
 LISN – CNRS
 lionel.mathelin@cnrs.fr



- ▶ Associate instructor: **Rémy Hosseinkhan-BOUCHER**
 LISN – INRIA
 remy.hosseinkhan@universite-paris-saclay.fr



Organization

- ▶ 6×4 hours
- ▶ Mostly focused on fundamentals and methods + some practice
- ▶ Not so much an hands-on class
- ▶ Introduction to core concepts and vocabulary. Understanding of underlying hypotheses and associated limits
 - It is **not** a tutorial on RL but rather a course

Which one is powered by Reinforcement Learning?

Autonomous driving



AlphaGo



ChatGPT



What is Reinforcement Learning?

Definition

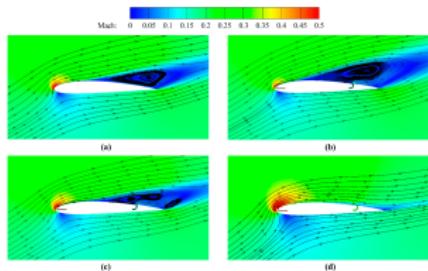
*“Reinforcement learning (RL) is an interdisciplinary area of machine learning and optimal control concerned with how an **intelligent agent** ought to **take actions** in a **dynamic environment** in order to maximize the **cumulative reward**.”*



- Subset of AI
- Stochastic control, optimal control in a stochastic environment
- Approximate dynamic programming

What Reinforcement Learning is about

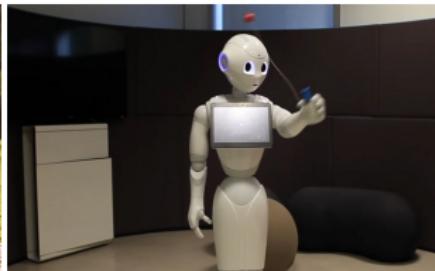
Finding good flow control



Learning to ride a bike

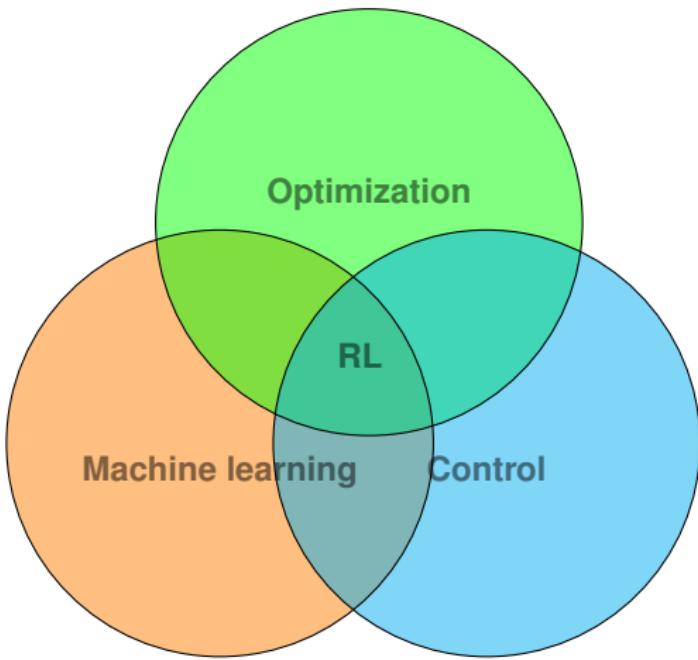


Ball-in-a-cup



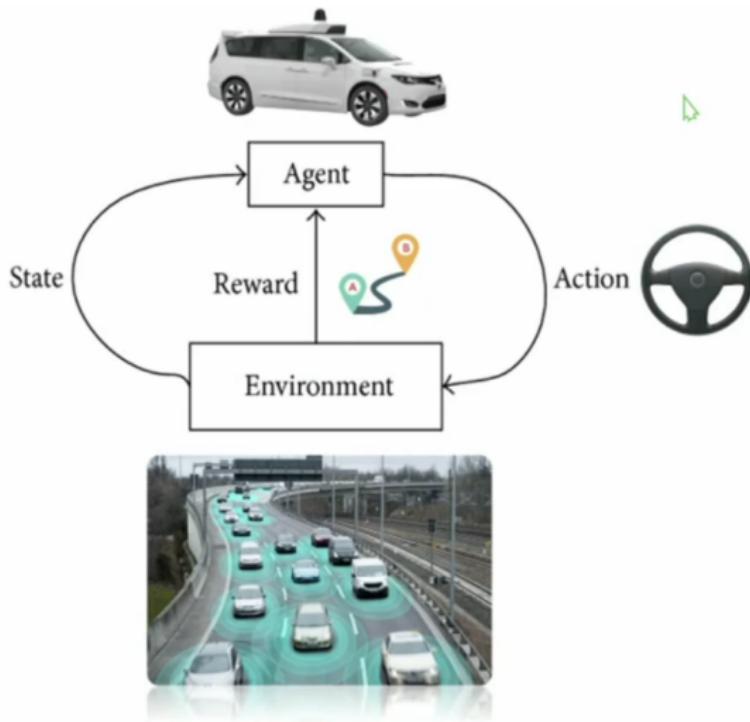
Learning to act by interacting with an uncertain environment

Reinforcement Learning relies on several disciplines



Reinforcement learning

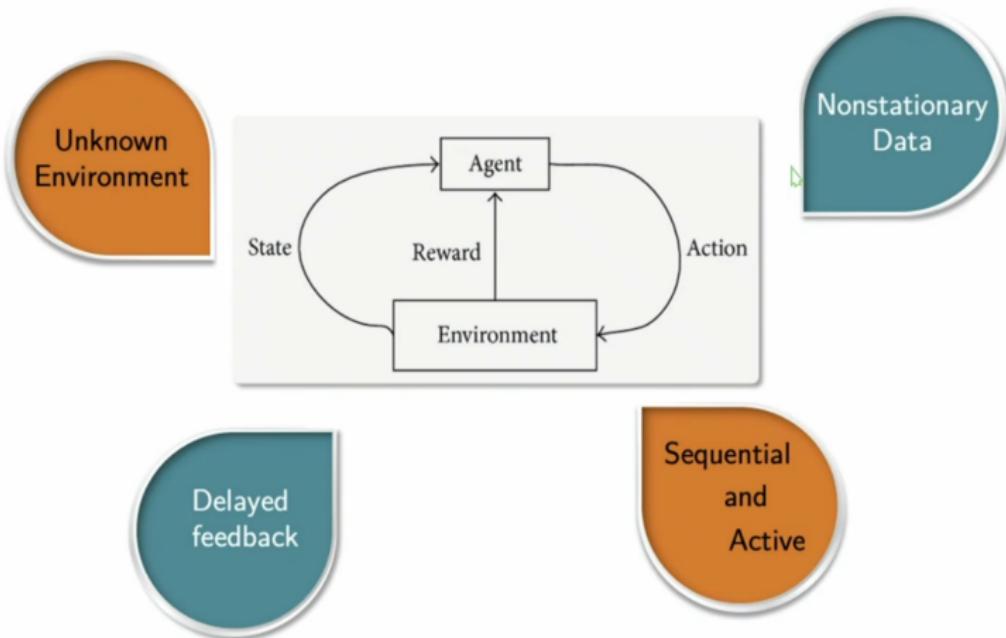
An agent learns to act by interacting with an uncertain environment



Number of states and actions can be finite or infinite

Reinforcement learning

An agent learns to act by interacting with an uncertain environment



Final reward may be only defined once you reach the objective

Nonstationary: different from supervised learning where data are stationary and *iid*

An incomplete list of applications of RL

- Video compression (MuZero)
- Faster matrix multiplication
- Chip design applied to Google TPUs
- Building AC management
- Recommendation systems
- Vision-based autonomous drone racing
- Swarm of self-driving cars (MegaVanderTest) to remove stop-and-go waves, save fuel
- Pick&place robots
- Plasma control (Tokamaks)
- Ocean wave energy harvesting
- Air Traffic Deconfliction

From Prof. Csaba SZEPESVÁRI (U. Alberta)

<https://tinyurl.com/3zw9453p>

Learning objectives

By the end of the course, students will be able to:

- ▶ Define the key features of RL that distinguishes it from standard ML
- ▶ Identify the strengths and limitations of various RL algorithms
- ▶ Recognize the common, connecting boundary of optimization and RL
- ▶ Understand the theoretical properties of RL algorithms
- ▶ Formulate and solve sequential decision-making problems by applying relevant RL tools
- ▶ Generalize or discover “new” algorithms, theories of RL towards conducting innovative research on the topic

Should I take this course? Or not?

This course is right for you if you

- want to understand the theory behind RL
- want to go beyond the mere application of popular algorithms
- have some background in probability, optimization, machine learning

Should I take this course? Or not?

This course is right for you if you

- want to understand the theory behind RL
- want to go beyond the mere application of popular algorithms
- have some background in probability, optimization, machine learning

Or not...

- want a hands-on experience on RL
- focus on a practical application of RL
- want to build robots

Ressources

RL

- *Reinforcement Learning: an introduction*, Richard Sutton & Andrew Barto, 2018
- *Reinforcement Learning and Optimal Control*, Dimitri Bertsekas, 2019
- *Reinforcement Learning: Theory and Algorithms*, Alekh Agarwal, Nan Jiang & Sham Kakade, 2020
- *Control Systems and Reinforcement Learning*, S. Meyn, Cambridge University Press, 2021
- This preliminary version of the course heavily borrows from the material by Prof. Niao He (ETH Zurich)

General purpose AI

- A. Ng: Machine Learning on Coursera
- *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, T. Hastie, R. Tibshirani & J. Friedman, Springer, 2009.
<https://web.stanford.edu/~hastie/ElemStatLearn/download.html>
- *Deep Learning*, I. Goodfellow, Y. Bengio & A. Courville, MIT Press, 2017.
<https://www.deeplearningbook.org/>
- Practical implementation of algorithms: countless online tutorials

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
- 8 Imitation learning
- 9 Going beyond

Sommaire

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
- 8 Imitation learning
- 9 Going beyond

A few illustrative examples

► Q-learning snake

► Locomotion

► Pendulum

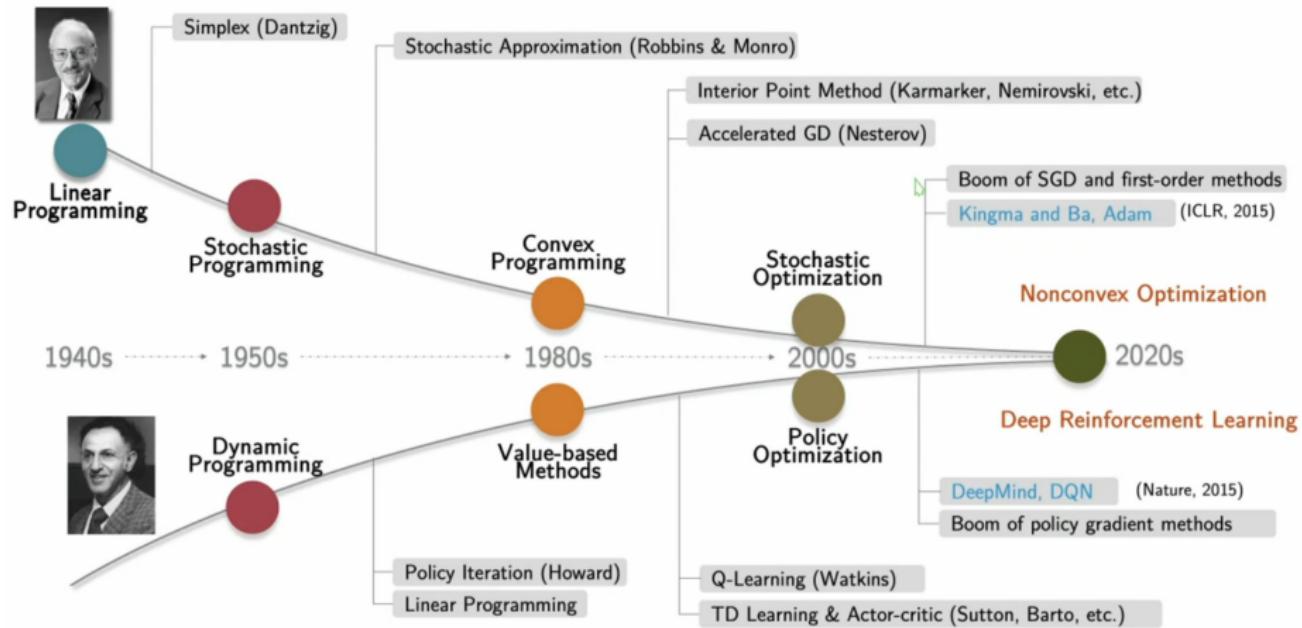
► Imitation learning

► Tennis table

► Rubik's cube

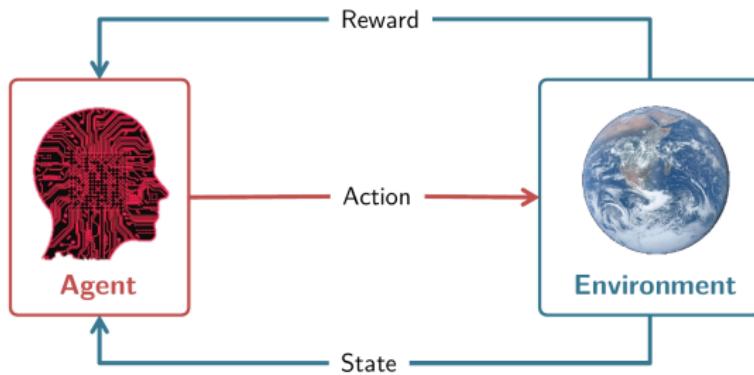
► Coast runner

The crossroad of optimization and RL



RL in an abstract form

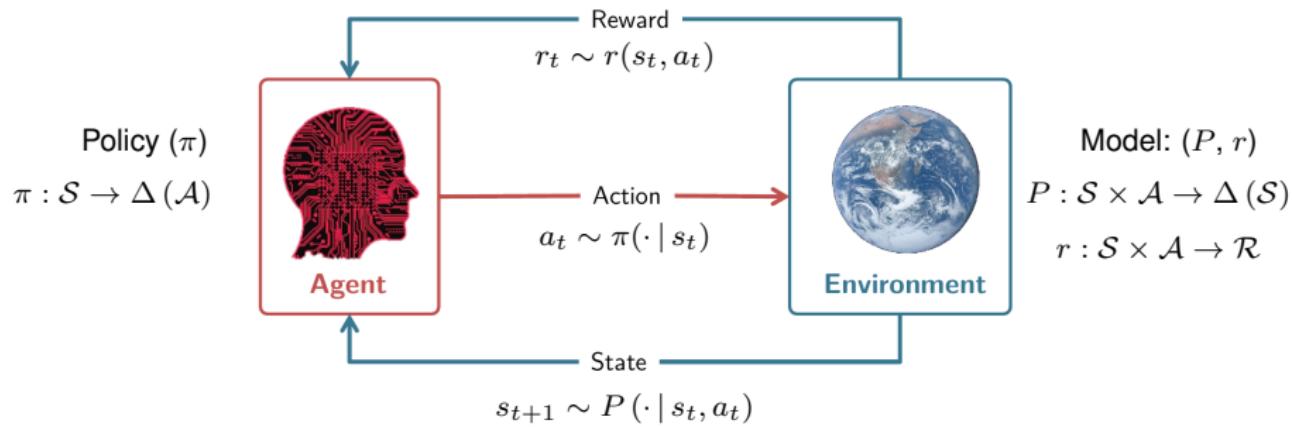
Goal: maximize expected cumulative rewards



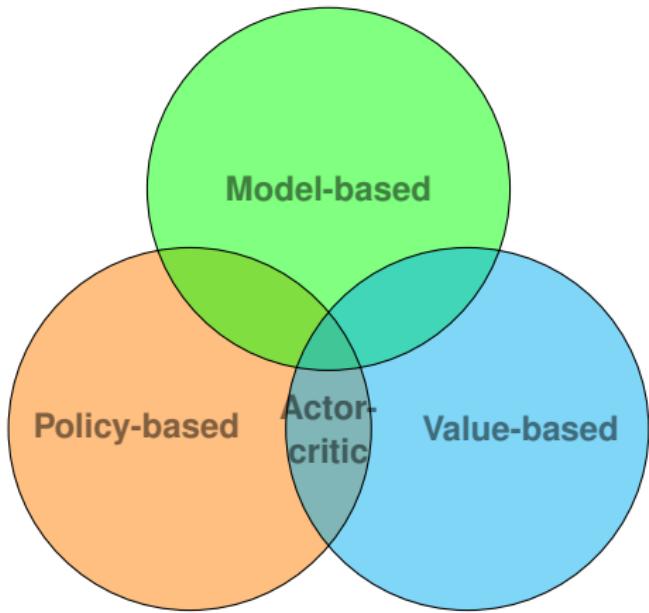
RL in a mathematical form

Goal: $\pi \in \arg \max_{\pi \in \Pi} V^\pi(s)$

Value: $V^\pi(s) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right]$



Overview of RL Approaches



• Value-based RL

- ▶ Learn the optimal value function
- ▶ Example: Monte Carlo, SARSA, Q-learning, DQN, etc.
- ▶ Low variance, not scalable to large action space

• Policy-based RL

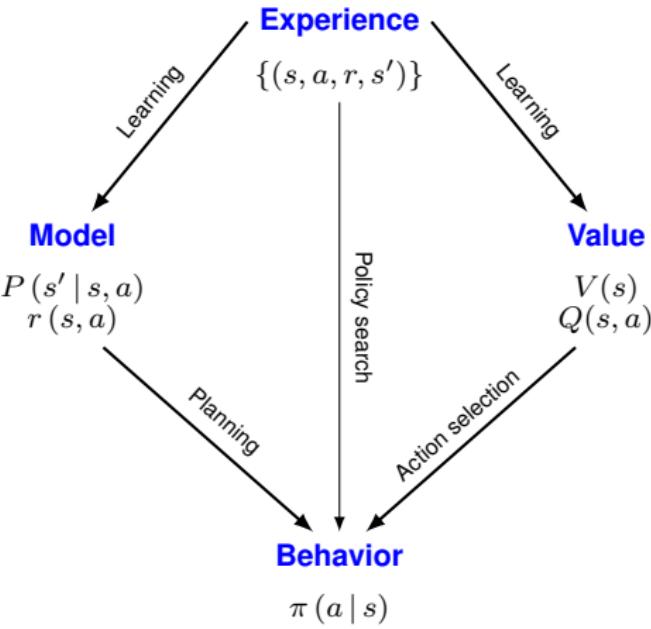
- ▶ Learn directly the optimal policy
- ▶ Example: Policy Gradient, NPG, TRPO, PPO, etc.
- ▶ Scale to large and continuous action space, high variance

• Model-based RL

- ▶ Learn both the model P, r and the optimal policy
- ▶ Example: Dyna, UCRL2, UCB-VI, etc.
- ▶ Computationally expensive, better data efficiency

Model-based vs Model-free RL

- Make full use of experiences
- Can reason about model uncertainty
- Sample efficient for easy dynamics



- Direct and simple
- Not affected by poor model estimation
- Computationally efficient

Online vs. Offline RL

Online RL

Collect online data by interacting with environment

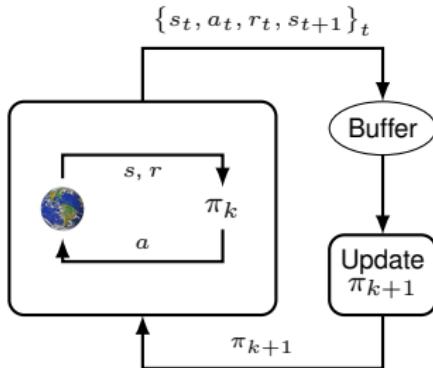
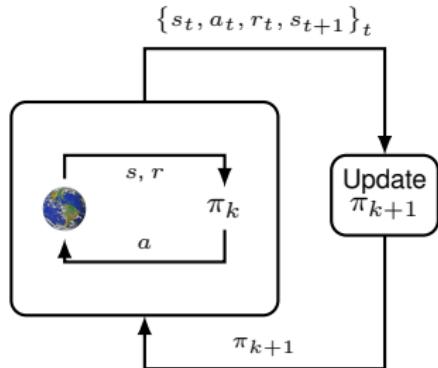
Exploitation-exploration tradeoff

Offline/Batch RL

Use previously collected data (often from behavior policy)

Data is static, no online data collection

On-policy vs Off-policy RL



On-policy RL

Learn based on data collected from current policy

Always online

Off-policy RL

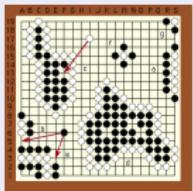
Learn based on data buffer that contains experiences collected from other policies

Can be online or offline

The Grand Challenges in Modern RL Era

Large state and policy space

Representation (state, policy)



Chess: 10^{50}

Go: 10^{172}

The Grand Challenges in Modern RL Era

Large state and policy space

Complex geometric landscape

Representation (state, policy)

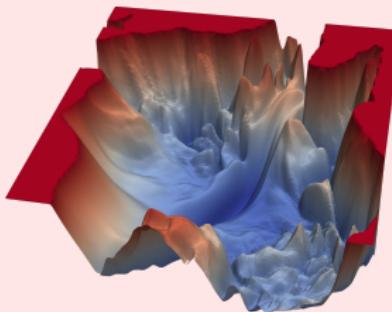


Chess: 10^{50}



Go: 10^{172}

Optimization (Value)



The Grand Challenges in Modern RL Era

Large state and policy space

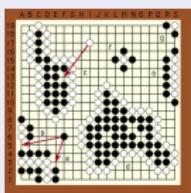
Complex geometric landscape

Exploitation-exploration dilemma

Representation (state, policy)

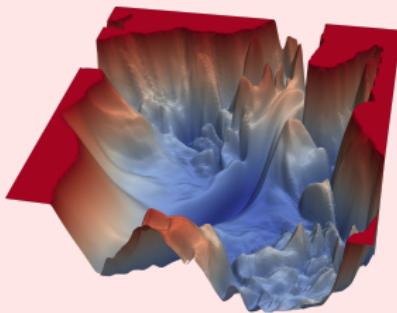


Chess: 10^{50}

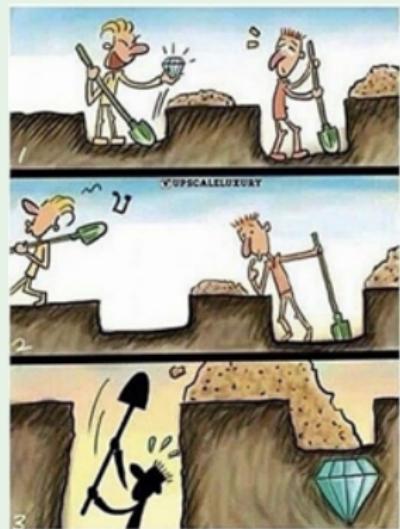


Go: 10^{172}

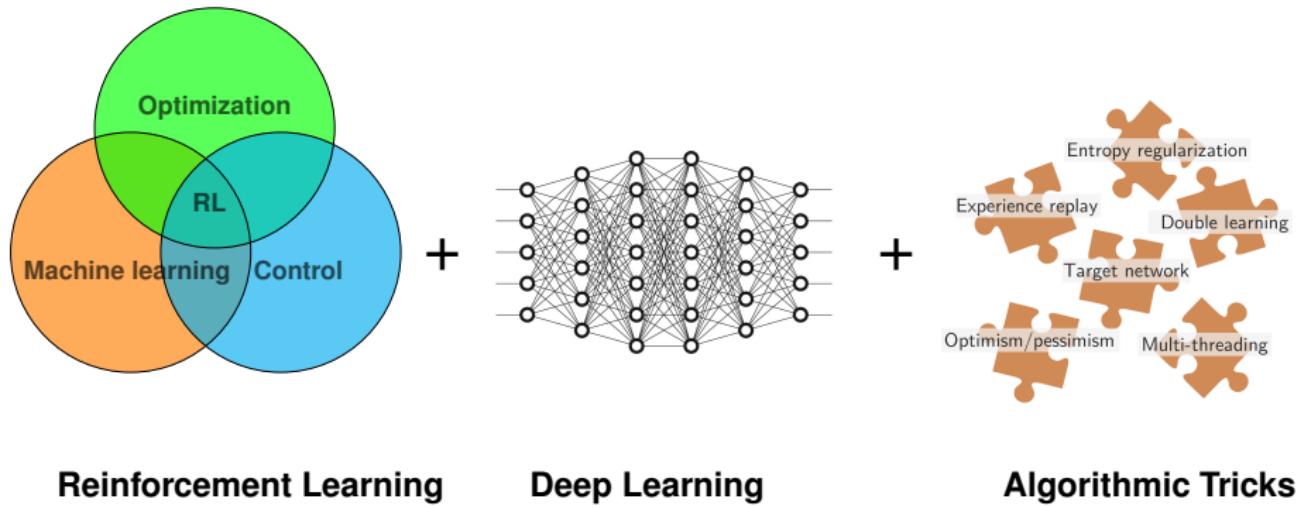
Optimization (Value)



Representation (policy)



The State-of-the-art RL Solutions



The Wide Theory-Practice Gap

RL theory

- Finite (small) state/action space
- Linear function approximation
- Vanilla value- and policy-based algorithms
- MDP models

RL practice

- Large or infinite state/action space
- Deep neural network
- Lots of engineering tricks
- Beyond MDP models

Sommaire

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
- 8 Imitation learning
- 9 Going beyond

Simplified framework: bandits

This part is covered in class on the board.

No slide yet

Sommaire

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems**
- 4 Exact solution methods
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
- 8 Imitation learning
- 9 Going beyond

Minimax problems

This part is covered in class on the board.

No slide yet

Sommaire

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
 - Value Iteration
 - Policy iteration
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
- 8 Imitation learning
- 9 Going beyond

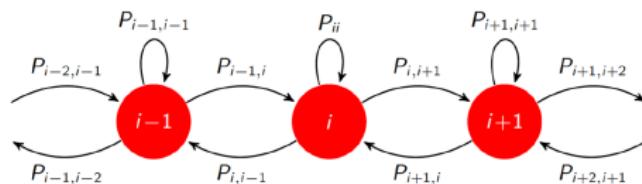
Markov chain

RL is usually formulated in this framework

Definition

A (time-homogeneous) **Markov chain** is a stochastic process $\{s_0, s_1, \dots\}$ (taking value on a countable number of states) satisfying the Markov property, i.e.,

$$P(s_{t+1} = j | s_t = i, s_{t-1}, \dots, s_0) = P(s_{t+1} = j | s_t = i) = P_{ij}$$



- ▶ **Stationary** distribution $d : d = dP$ ($\lambda = 1$)
- ▶ **Ergodicity:** If the Markov chain is irreducible, aperiodic with finite states, then there exists a unique stationary distribution and $\{s_t\}$ converges to it, i.e.,
 $\lim_{t \rightarrow \infty} P_{ij}^t = d_j, \forall i, j.$

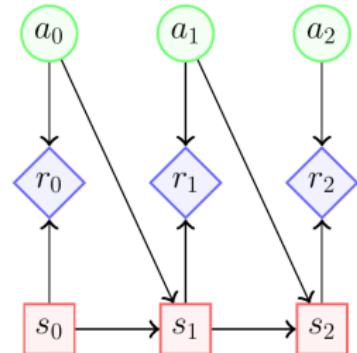
Markov Decision Process (MDP)

MDP \approx controlled Markov Chain + performance objective

Definition

An MDP is a 4-uple $\{\mathcal{S}, \mathcal{A}, P, r\}$ parameterized with μ, γ :

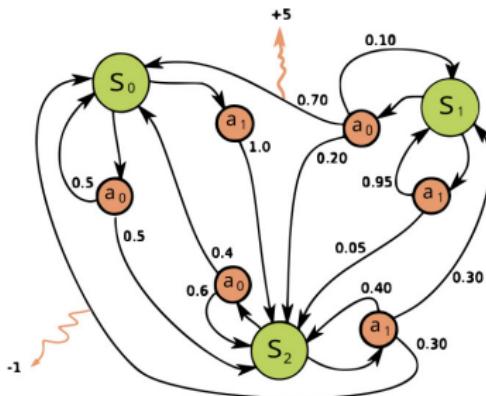
\mathcal{S}	State space, the set of all possible states
\mathcal{A}	Action space, the set of all possible actions
$P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ (simplex)	State transition model
$r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R} \in \mathbb{R}$	Expected reward function
μ	Initial state distribution, $s_0 \sim \mu \in \Delta(\mathcal{S})$
γ	Discount factor, $\gamma \in [0, 1]$



Controlled Markov property

$$P(s_{t+1} = s' | s_t = s, a_t = a, \dots, s_0, a_0) = P(s_{t+1} = s' | s_t = s, a_t = a) = P(s' | s, a)$$

Example of MDP



3 states, 2 actions

Example of MDP (cnt'd)

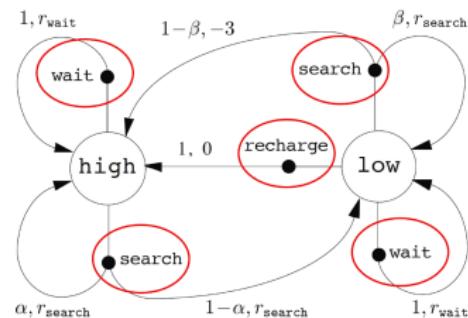
Consider a rechargeable mobile robot collecting empty drink cans

- ▶ State s : high, low (high/low charge level)
- ▶ Action a :
 - ▶ if high, action set {wait, search}
 - ▶ if low, action set {wait, search, recharge}
- ▶ Reward r :
 - ▶ $r(s_t = \text{high}, a_t = \text{search}) = \text{expected number of collected cans}$
 - ▶ $r(s_t = \text{low}, a_t = \text{recharge}) = 0$
 - ▶ ...
- ▶ Transition model P :
 - ▶ $P(s_{t+1} = \text{high} | s_t = \text{high}, a_t = \text{search}) = \alpha$
 - ▶ $P(s_{t+1} = \text{low} | s_t = \text{low}, a_t = \text{wait}) = 1$
 - ▶ ...



Example of MDP (cnt'd)

s	a	s'	$P(s' s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1 - \alpha$	r_{search}
low	search	high	$1 - \beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-



- ▶ Note that here $r(s, a, s')$ is the reward of the state-action-next-state triple
- ▶ $r(s, a) = \mathbb{E}_{s' \sim P(\cdot | s, a)} [r(s, a, s')]$

MDPs: Control Policy

Policy: based on the history $(s_{0:t}, a_{0:t-1})$, what action to select

Deterministic Policy

- ▶ Stationary policy

$$\pi : \mathcal{S} \rightarrow \mathcal{A}, a_t = \pi(s_t)$$

- ▶ Markov policy

$$\pi_t : \mathcal{S} \rightarrow \mathcal{A}, a_t = \pi_t(s_t)$$

- ▶ History-dependent policy

$$\pi_t : \mathcal{H}_{\mathcal{S}} \rightarrow \mathcal{A}$$

- ▶ $\mathcal{H}_{\mathcal{S}}$ is the set of all possible trajectories

- ▶ $a_t = \pi_t(h_t),$

- ▶ $h_t := (s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t)$

MDPs: Control Policy

Policy: based on the history $(s_{0:t}, a_{0:t-1})$, what action to select

Deterministic Policy

- ▶ Stationary policy

$$\pi : \mathcal{S} \rightarrow \mathcal{A}, a_t = \pi(s_t)$$

- ▶ Markov policy

$$\pi_t : \mathcal{S} \rightarrow \mathcal{A}, a_t = \pi_t(s_t)$$

- ▶ History-dependent policy

$$\pi_t : \mathcal{H}_{\mathcal{S}} \rightarrow \mathcal{A}$$

- ▶ $\mathcal{H}_{\mathcal{S}}$ is the set of all possible trajectories

- ▶ $a_t = \pi_t(h_t),$

- ▶ $h_t := (s_0, a_0, \dots, s_{t-1}, a_{t-1}, s_t)$

Stochastic Policy

- ▶ Stationary policy

$$\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A}), a_t \sim \pi(\cdot | s_t)$$

- ▶ Markov policy

$$\pi_t : \mathcal{S} \rightarrow \Delta(\mathcal{A}), a_t \sim \pi_t(\cdot | s_t)$$

- ▶ History-dependent policy

$$\pi_t : \mathcal{H}_{\mathcal{S}} \rightarrow \Delta(\mathcal{A}),$$

$$a_t \sim \pi_t(\cdot | h_t)$$

- We mainly focus on **stationary** policies
- Markov properties \implies no need for history
- Stationarity \implies no need for Markov policy
- Deterministic stationary policy set: $|\mathcal{A}|^{|\mathcal{S}|}$

Finite horizon

- Cumulative reward

$$\mathbb{E} \left[\sum_{t=0}^{T-1} r(s_t, a_t) \right]$$

- Average reward

$$\mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} r(s_t, a_t) \right]$$

Infinite horizon

- Discounted reward

$$\mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

- Average reward

$$\liminf_{T \rightarrow \infty} \mathbb{E} \left[\frac{1}{T} \sum_{t=0}^{T-1} r(s_t, a_t) \right]$$

- ▶ One can also consider risk-sensitive criteria such as variance, percentile, etc.
- ▶ The infinite-horizon objectives can be maximized by a stationary policy; the finite horizon objective needs a Markov policy.

Why discounting in infinite-horizon MDPs?

$$\mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

- ▶ Mathematical perspective
 - ▶ With $\gamma = 1$, the total reward may be infinite due to the infinite horizon
 - ▶ With $\gamma \in [0, 1)$, assuming $|r(\cdot, \cdot)| \leq \bar{r}$, the total reward will always be finite
- ▶ Problem specification
 - ▶ When $\gamma = 0$, the total reward will just be the immediate reward
 - ▶ When $\gamma = 1$, the future reward is as important as the present reward
 - ▶ When $\gamma \in (0, 1)$, it plays a role of “balancing” between the present and the future, while putting more weight to the immediate reward
- ▶ Effective horizon

$$\forall \epsilon > 0, \quad T \geq \mathcal{O} \left(\frac{1}{1-\gamma} \log \left(\frac{\bar{r}}{\epsilon} \right) \right) \implies \sum_{t=T}^{\infty} \gamma^t r(s_t, a_t) \leq \epsilon$$

Value Functions – how good is a policy?

State Value function

$$V^\pi(s) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, \pi \right]$$

Q-function / State-Action Value function

$$Q^\pi(s, a) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a, \pi \right]$$

- ▶ State value function represents the expected return starting at the state under policy π
- ▶ Q-function represents the value of an action at a state
- ▶ For convenience, we may drop the π in RHS when it is clear from the context

Value Functions (cont'd)

By definition, it is clear that for any policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$:

Relations between V^π and Q^π

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^\pi(s')$$

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a | s) Q^\pi(s, a)$$

Proof

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a, \pi \right]$$

Proof

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a, \pi \right] \\ &= \color{blue}{r(s, a)} + \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \end{aligned}$$

Proof

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a, \pi \right] \\ &= r(s, a) + \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_0 = s, s_1 = s', a_0 = a \right] \end{aligned}$$

Proof

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a, \pi \right] \\ &= r(s, a) + \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_0 = s, s_1 = s', a_0 = a \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_1 = s' \right] \quad [\text{Markov hyp.}] \end{aligned}$$

Proof

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a, \pi \right] \\ &= r(s, a) + \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_0 = s, s_1 = s', a_0 = a \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_1 = s' \right] \quad [\text{Markov hyp.}] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s', a_t \sim \pi \right] \end{aligned}$$

Proof

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a, \pi \right] \\ &= r(s, a) + \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_0 = s, s_1 = s', a_0 = a \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_1 = s' \right] \quad [\text{Markov hyp.}] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s', a_t \sim \pi \right] \\ &= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' \mid s, a) V^\pi(s') \end{aligned}$$

Optimal Value functions

Let Π be the set of all (possibly non-stationary and stochastic) policies

Optimal Value Function

$$V^*(s) := \max_{\pi \in \Pi} V^\pi(s)$$

Optimal State Value Function

$$Q^*(s, a) := \max_{\pi \in \Pi} Q^\pi(s, a)$$

Relations between V^* and Q^*

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^*(s')$$

$$V^*(s) = \max_{a \in \mathcal{A}} Q^*(s, a)$$

Solving MDPs: Find the optimal policy

Goal of RL

To find an optimal policy $\pi^* \in \Pi$ such that

$$V^{\pi^*}(s) = V^*(s) := \max_{\pi \in \Pi} V^\pi(s), \quad \forall s \in \mathcal{S}$$

Issues:

- ▶ Does the optimal policy π^* exist?
- ▶ How to evaluate the current policy π , i.e., how to compute $V^\pi(s)$?
- ▶ If π^* exists, how to improve the current policy π , i.e., how to find π^* ?

NB: The optimal policy may not be unique, while V^* and Q^* are unique.

Existence of Optimal Policy

Theorem: Existence of the Optimal Policy

For an infinite horizon MDP $M = (\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \mu, \gamma)$, there exists a stationary and deterministic policy π such that for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$,

$$V^\pi(s) = V^*(s), \quad Q^\pi(s, a) = Q^*(s, a)$$

- ▶ One can directly get a (deterministic and stationary) optimal policy from Q^* :

$$\pi^*(s) \in \arg \max_{a \in \mathcal{A}} Q^*(s, a)$$

- ▶ Finding π^* can be done by first computing V^* or Q^*

$$V^* \text{ or } Q^* \longrightarrow \pi^*$$

Bellman Consistency Equation

Theorem: Bellman consistency equation

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot | s)} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^\pi(s') \right]$$

Obtained by combining relation between V^π and Q^π



Richard Bellman
(1920–1984)

Bellman Consistency Equation

Theorem: Bellman consistency equation

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot | s)} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^\pi(s') \right]$$

Obtained by combining relation between V^π and Q^π



Richard Bellman
(1920–1984)

Matrix form:

$$\mathbf{V}^\pi = \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{V}^\pi$$

where:

- ▶ $\mathbf{V}^\pi \in \mathbb{R}^{|\mathcal{S}|}$: $\mathbf{V}_s^\pi = V^\pi(s)$
- ▶ $\mathbf{r}^\pi \in \mathbb{R}^{|\mathcal{S}|}$: $\mathbf{r}_s^\pi = \sum_{a \in \mathcal{A}} \pi(a | s) r(s, a)$
- ▶ $\mathbf{P}^\pi \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$: $\mathbf{P}_{s,s'}^\pi := \sum_{a \in \mathcal{A}} \pi(a | s) P(s' | s, a)$

Closed-Form Solution for Policy Evaluation

Closed-Form Solution of V^π

$$\mathbf{V}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{r}^\pi$$

\mathbf{P}^π is a stochastic matrix $\implies \lambda \leq 1$

- ▶ This is one of exact solution methods for policy evaluation
- ▶ Note that the matrix $\mathbf{I} - \gamma \mathbf{P}^\pi$ is always invertible for $\gamma \in [0, 1)$
- ▶ The solution of Bellman equation is always unique
- ▶ Computation cost: $\mathcal{O}(|\mathcal{S}|^3 + |\mathcal{S}|^2|\mathcal{A}|)$, which can be expensive for large state space

Bellman Expectation Operator and Fixed-Point Perspective

Bellman expectation operator

Let $\mathcal{T}^\pi : \mathbb{R}^{|S|} \rightarrow \mathbb{R}^{|S|}$ be defined as

$$\mathcal{T}^\pi \mathbf{V} := \mathbf{r}^\pi + \gamma \mathbf{P}^\pi \mathbf{V}$$

- ▶ Bellman equation implies that \mathbf{V}^π is the **unique fixed point** of $\mathcal{T}^\pi : \mathcal{T}^\pi \mathbf{V}^\pi = \mathbf{V}^\pi$
- ▶ \mathcal{T}^π is a linear operator and is a γ -contraction mapping
- ▶ Fixed point iteration: $\mathbf{V}_{t+1} = \mathcal{T}^\pi \mathbf{V}_t, t = 0, 1, \dots$
- ▶ $\lim_{t \rightarrow \infty} (\mathcal{T}^\pi)^t \mathbf{V}_0 = \mathbf{V}^\pi$
- ▶ Computationally efficient since cost of one iteration is $\mathcal{O}(|S|^2 |\mathcal{A}|)$

Bellman Optimality Equations

Bellman optimality equation

$$V^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^*(s') \right]$$

$$Q^*(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) \max_{a' \in \mathcal{A}} [Q^*(s', a')]$$

Obtained directly from the relation between V^* and Q^*

See slide 42

Bellman Optimality Operator

Bellman optimality operator

$$(\mathcal{T}\mathbf{V})(s) := \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) \mathbf{V}(s') \right]$$

- ▶ The optimal value function \mathbf{V}^* is the fixed point of \mathcal{T} , i.e.,

$$\mathcal{T}\mathbf{V}^* = \mathbf{V}^*$$

- ▶ The Bellman optimality operator is a γ -contraction mapping w.r.t. ℓ_∞ -norm
- ▶ The Bellman operator is also monotonic (component-wise):

$$\mathbf{V}_1 \leq \mathbf{V}_2 \implies \mathcal{T}\mathbf{V}_1 \leq \mathcal{T}\mathbf{V}_2$$

- ▶ We can define a similar Bellman operator on the Q -function and show similar properties

Contraction of Bellman Optimality Operator

Contraction property of \mathcal{T}

The Bellman optimality operator \mathcal{T} defined above is a γ -contraction mapping under ℓ_∞ -norm, i.e., for any $\mathbf{V}, \mathbf{V}' \in \mathbb{R}^{|\mathcal{S}|}$:

$$\|\mathcal{T}\mathbf{V}' - \mathcal{T}\mathbf{V}\|_\infty \leq \gamma \|\mathbf{V}' - \mathbf{V}\|_\infty$$

Useful to prove and study the convergence of some algorithms

Contraction of Bellman Optimality Operator

Proof.

For any $\mathbf{V}, \mathbf{V}' \in \mathbb{R}^{|\mathcal{S}|}$ and $s \in \mathcal{S}$, one has:

$$\begin{aligned} |(\mathcal{T}\mathbf{V}') (s) - (\mathcal{T}\mathbf{V}) (s)| &= \left| \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) \mathbf{V}'(s') \right] \right. \\ &\quad \left. - \max_{a' \in \mathcal{A}} \left[r(s, a') + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a') \mathbf{V}(s') \right] \right| \\ &\leq \max_{a \in \mathcal{A}} \left| r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) \mathbf{V}'(s') \right. \\ &\quad \left. - r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) \mathbf{V}(s') \right| \\ &= \max_{a \in \mathcal{A}} \left[\gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) |\mathbf{V}'(s') - \mathbf{V}(s')| \right] \\ &\leq \|\mathbf{V}' - \mathbf{V}\|_\infty \max_{a \in \mathcal{A}} \left[\gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) \right] = \gamma \|\mathbf{V}' - \mathbf{V}\|_\infty \end{aligned}$$



Value Iteration (VI)

- ▶ From the fixed-point perspective, solving MDPs (finding V^* or π^*) can be converted into finding the fixed point of \mathcal{T}
- ▶ A common method is the value iteration, which can be viewed as fixed-point iteration

Value Iteration (VI) for solving MDPs

Start with an arbitrary guess \mathbf{V}_0 (e.g., $V_0(s) = 0$ for any s)

for each iteration t **do**

 Apply the Bellman operator \mathcal{T} at each iteration

$$\mathbf{V}_{t+1} = \mathcal{T}\mathbf{V}_t$$

end for

Discussion on Value Iteration

- ▶ After obtaining V^* by VI, we can obtain an optimal policy from the **greedy policy**:

$$\pi^*(s) = \arg \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^*(s') \right]$$

- ▶ Alternatively, we can run Q -value iteration to obtain Q^* by iteratively applying the corresponding Bellman optimality operator w.r.t. the Q -function, and then directly compute π^* by the greedy policy:

$$\pi^*(s) = \arg \max_{a \in \mathcal{A}} [Q^*(s, a)]$$

- ▶ Per-iteration cost of VI: $\mathcal{O}(|\mathcal{S}|^2 |\mathcal{A}|)$
- ▶ Question: How fast does it converge?

Convergence of Value Iteration

Linear convergence of value iteration

The value iteration algorithm attains a linear convergence rate, *i.e.*,

$$\|\mathbf{V}_t - \mathbf{V}^*\|_\infty \leq \gamma^t \|\mathbf{V}_0 - \mathbf{V}^*\|_\infty$$

Contraction by a γ factor at each iteration (linear convergence)

Proof.

$$\|\mathbf{V}_t - \mathbf{V}^*\|_\infty = \|\mathcal{T}\mathbf{V}_{t-1} - \mathcal{T}\mathbf{V}^*\|_\infty \leq \gamma \|\mathbf{V}_{t-1} - \mathbf{V}^*\|_\infty \leq \dots \leq \gamma^t \|\mathbf{V}_0 - \mathbf{V}^*\|_\infty$$



Very similar to policy evaluation (where one uses Bellman expectation operator)

Directly updating the Policy

- ▶ Value iteration first finds V^* , then computes the optimal policy π^* by the greedy policy
- ▶ Now we directly search for the optimal policy π^*
- ▶ Some intuitions: start from an initial guess π , iteratively do:
 - ▶ Evaluate policy: compute the value function V^π of the current policy
[Policy evaluation]
 - ▶ Improve policy: update the guess by the greedy policy w.r.t. V^π
[Policy improvement]

Policy Improvement Theorem

Theorem: Policy improvement

If a (deterministic) policy π' satisfies,

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s), \quad \forall s \in \mathcal{S},$$

then $V^{\pi'} \geq V^\pi(s)$, $\forall s \in \mathcal{S}$.

- ▶ Improving the current policy by one step everywhere, we can improve the whole policy
- ▶ It suggests a natural way of improving the current policy via,

$$\pi_{t+1}(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q^{\pi_t}(s, a)$$

- ▶ Indeed, $V^{\pi_{t+1}}(s) \geq V^{\pi_t}(s)$, $\forall s \in \mathcal{S}$, and the inequality is strict if π_t is suboptimal.

Policy Improvement Theorem

Theorem: Policy improvement

If a (deterministic) policy π' satisfies,

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s), \quad \forall s \in \mathcal{S},$$

then $V^{\pi'} \geq V^\pi(s), \forall s \in \mathcal{S}$.

Proof.

Denote $s' \sim P(\cdot | s, \pi'(s))$

$$\begin{aligned} V^\pi(s) &\leq Q^\pi(s, \pi'(s)) = \mathbb{E}_{\pi'} [r(s_0, \pi'(s)) + \gamma V^\pi(s_1) | s_0 = s] \\ &\leq \mathbb{E}_{\pi'} [r_0 + \gamma Q^\pi(s_1, \pi'(s_1)) | s_0 = s] \\ &\leq \mathbb{E}_{\pi'} [r_0 + \gamma r_1 + \gamma V^\pi(s_1) | s_0 = s] \\ &\leq \dots \\ &\leq \mathbb{E}_{\pi'} [r_0 + \gamma r_1 + \gamma^2 r_2 + \dots | s_0 = s] \\ &= V^{\pi'}(s) \end{aligned}$$



Policy Iteration

Policy Iteration (PI) for solving MDPs

Start with an arbitrary policy guess π_0

for each iteration t **do**

[Step 1: Policy evaluation] Compute \mathbf{V}^{π_t} :

(Option 1) Iteratively apply policy value iteration, $\mathbf{V}_t \leftarrow \mathcal{T}^{\pi_t} \mathbf{V}_t$, until convergence

(Option 2) Use the closed-form solution: $\mathbf{V}^{\pi_t} = (\mathbf{I} - \gamma \mathbf{P}^{\pi_t})^{-1} \mathbf{r}^{\pi_t}$

[Step 2: Policy improvement] Update the current policy π_t by the greedy policy

$$\pi_{t+1}(s) = \arg \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^{\pi_t}(s') \right]$$

end for

NB: PI converges in finite number of iterations.

Convergence of Policy Iteration

Theorem (Linear Convergence of Policy Iteration)

The policy iteration attains a linear convergence rate:

$$\|\mathbf{V}^{\pi_t} - \mathbf{V}^*\|_\infty \leq \gamma^t \|\mathbf{V}^{\pi_0} - \mathbf{V}^*\|_\infty$$

Proof.

Let $a = \pi_{t+1}(s)$

Since $V^{\pi_{t+1}}(s) \geq V^{\pi_t}(s), \forall s$, we have

$$V^{\pi_{t+1}}(s) = r(s, a) + \mathbb{E}_{s' | s, a} [V^{\pi_{t+1}}(s')] \geq r(s, a) + \mathbb{E}_{s' | s, a} [V^{\pi_t}(s')] = \mathcal{T}V^{\pi_t}(s)$$

Hence, $\mathbf{V}^* \geq \mathbf{V}^{\pi_t} \geq \mathcal{T}\mathbf{V}^{\pi_{t-1}}$

$$\|\mathbf{V}^{\pi_t} - \mathbf{V}^*\|_\infty \leq \|\mathcal{T}\mathbf{V}^{\pi_{t-1}} - \mathcal{T}\mathbf{V}^*\|_\infty \leq \gamma \|\mathbf{V}^{\pi_{t-1}} - \mathbf{V}^*\|_\infty \leq \dots \leq \gamma^t \|\mathbf{V}^{\pi_0} - \mathbf{V}^*\|_\infty$$



Dynamic Programming

Value Iteration (VI)

Initialize \mathbf{V}_0

for each iteration t do

$$\mathbf{V}_{t+1} = \mathcal{T}\mathbf{V}_t$$

end for

$$\pi^*(s) = \arg \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^*(s') \right] \quad [\text{greedy policy}]$$

Policy Iteration (PI)

Initialize π_0

for each iteration t do

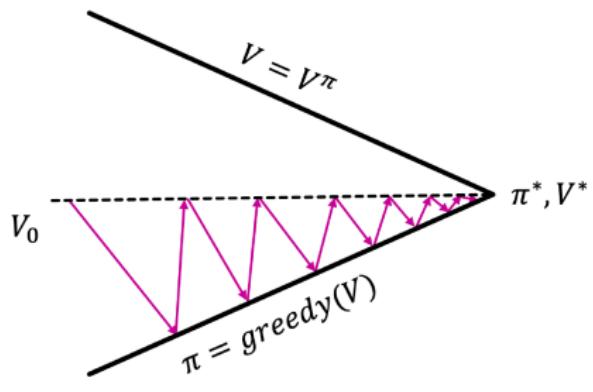
[Policy evaluation] Compute \mathbf{V}^{π_t} or \mathbf{Q}^{π_t}

[Policy improvement] Update π

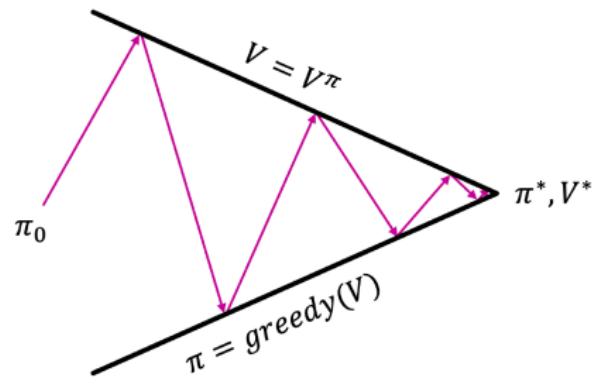
$$\pi_{t+1}(s) = \arg \max_{a \in \mathcal{A}} [Q^{\pi_t}(s, a)]$$

end for

Value Iteration vs. Policy Iteration



(a) Value Iteration



(b) Policy Iteration

Value Iteration vs. Policy Iteration

Algorithm	Component	Output
Value Iteration (VI)	Bellman optimality operator \mathcal{T}	V^*
Policy Iteration (PI)	(multiple) Bellman operator \mathcal{T}^π + greedy policy	π^*

Value Iteration

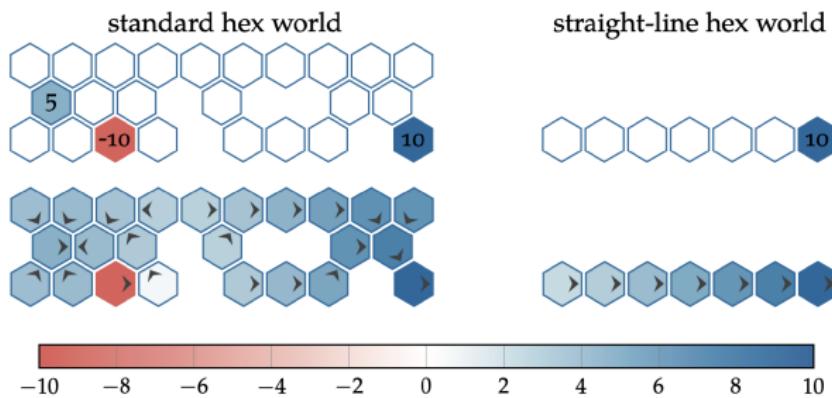
- ▶ One sweep through operator \mathcal{T}
- ▶ Per-iteration cost: $\mathcal{O}(|\mathcal{S}|^2|\mathcal{A}|)$
- ▶ Linear convergence rate
- ▶ Cheap cost, more iterations

Policy Iteration

- ▶ Multiple sweeps through operator \mathcal{T}^π
- ▶ Per-iteration cost:
 - $\mathcal{O}((1 - \gamma)^{-1} |\mathcal{S}|^2 |\mathcal{A}|)$ [policy evaluation]
 - $\mathcal{O}(|\mathcal{S}|^2 |\mathcal{A}| + |\mathcal{S}|^3)$ [closed-form solution]
- ▶ Linear convergence rate
- ▶ Expensive cost, fewer iterations

Example: Hex world¹

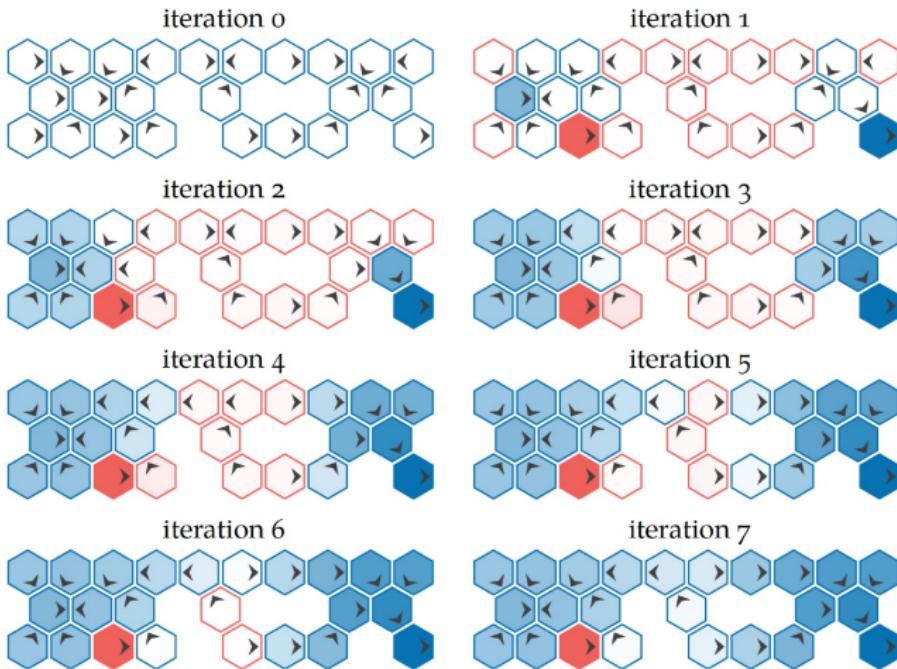
- ▶ Traverse a tile map to reach a goal state
- ▶ Each cell in the tile map represents a state; action is a move in any of the 6 directions
- ▶ Taking any action in certain cells gives a specified reward and transports to a terminal state



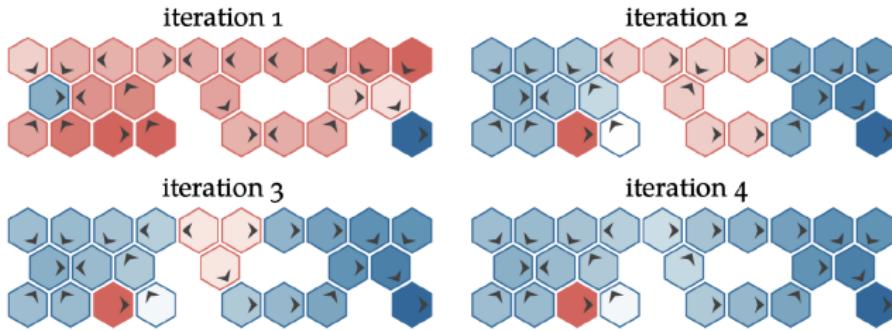
Top row shows the base problem setup and colors hexes with terminal rewards. Bottom row shows an optimal policy for each problem and colors the expected value.

¹ M.J. Kochenderfer, T.A. Wheeler & K.H. Wray, *Algorithms for Decision Making*, MIT press, 2022.

Hex world: Value iteration



Hex world: Policy iteration



Initialized with the **east-moving** policy

An optimal policy is obtained (the algorithm converges) in four iterations

Revisit Bellman Optimality Equation

- ▶ Finding V^* satisfying the Bellman optimality equation can be written as a **feasibility problem**:

minimum V

$$\text{s.t. } V^*(s) = \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^*(s') \right], \quad \forall s \in \mathcal{S}$$

- ▶ Relaxation: The above constraint suggests that V^* is the “least feasible solution” of V satisfying

$$V(s) \geq r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V(s'), \quad \forall s \in \mathcal{S}, a \in \mathcal{A}$$

- ▶ Note that the new constraints above is **linear** in $V \implies$ **Linear Programming (LP)**

Solving MDPs with LP

Let $\mu(s) \geq 0, s \in \mathcal{S}$, be the initial distribution (or any positive weights).

Primal LP for finding V^*

$$\begin{aligned} & \min_V \sum_{s \in \mathcal{S}} \mu(s) V(s) \\ \text{s.t. } & V(s) \geq r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V(s'), \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \end{aligned}$$

- ▶ The optimal value function V^* is the unique solution to the above LP.
- ▶ Number of decision variables: $|\mathcal{S}|$, number of constraints: $|\mathcal{S}| \times |\mathcal{A}|$.
- ▶ For any feasible V , we have $V \geq \mathcal{T}V$ (pointwise)
By monotonicity of \mathcal{T} :

$$V \geq \mathcal{T}V \geq \dots \geq \mathcal{T}^\infty V = V^*$$

$\implies V^*$ is the minimum feasible solution

Linear Programming (LP)

Primal LP:

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \geq b \end{aligned}$$

Dual LP:

$$\begin{aligned} \max \quad & b^\top \lambda \\ \text{s.t.} \quad & A^\top \lambda = c \\ & \lambda \geq 0 \end{aligned}$$

- ▶ **Strong Duality Theorem:** If one of them has an optimal solution, then so does the other one and their optimal values are the same.
- ▶ **Solution Methods:** Interior Point Method (polynomial-time), Simplex Method (more efficient but could be potentially exponential-time).

Dual LP Formulation

$$\begin{aligned} \max_{\lambda} \quad & \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} r(s, a) \lambda(s, a) \\ \text{s.t.} \quad & \sum_{a \in \mathcal{A}} \lambda(s, a) = \mu(s) + \gamma \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} P(s | s', a') \lambda(s', a'), \quad \forall s \in \mathcal{S} \\ & \lambda(s, a) \geq 0, \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \end{aligned}$$

- ▶ Number of decision variables: $|\mathcal{S}| \times |\mathcal{A}|$, number of constraints: $|\mathcal{S}|$.
- ▶ Usually easier to solve than primal because # constraints \leq # decision variables.
- ▶ The solution to the dual LP, λ^* , corresponds to the state-action occupancy of the optimal policy.

State-action Occupancy Measure and Distribution

Occupancy Measure

Given policy π and $s_0 \sim \mu$, the **state-action occupancy measure** of policy π is

$$\lambda^\pi(s, a) := \sum_{t=0}^{\infty} \gamma^t P(s_t = s, a_t = a \mid s_0 \sim \mu, \pi)$$

We can further define **state-action visitation distribution**:

$$d_\mu^\pi(s, a) := (1 - \gamma) \mathbb{E} \left[\sum_{t=0}^T \gamma^t \mathbf{1}(s_t = s, a_t = a) \mid s_0 \sim \mu, \pi \right]$$

where $d_\mu^\pi(s, a) \geq 0$ and $\sum_{s,a} d_\mu^\pi(s, a) = 1$

Dual Interpretation

Value Objective Equivalence

$$\sum_{s \in \mathcal{S}} \mu(s) V^\pi(s) = \sum_{s \in \mathcal{S}, a \in \mathcal{A}} \lambda^\pi(s, a) r(s, a)$$

Indeed

$$\begin{aligned} \sum_{s \in \mathcal{S}} \mu(s) V^\pi(s) &= \mathbb{E}_{s \sim \mu} [V^\pi(s)] && \text{[primal objective]} \\ &= \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu \right] \\ &= \sum_{s \in \mathcal{S}, a \in \mathcal{A}} \sum_{t=0}^{\infty} \gamma^t P(s_t = s, a_t = a \mid \pi) r(s, a) \\ &= \sum_{s \in \mathcal{S}, a \in \mathcal{A}} \lambda^\pi(s, a) r(s, a) && \text{[dual objective]} \end{aligned}$$

Dual Interpretation (contd)

Bellman Equation for State-action Occupancy Measure

$$\lambda^\pi(s, a) = \mu(s)\pi(a | s) + \gamma \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} \pi(a | s)P(s | s', a') \lambda^\pi(s', a')$$

Summing over $a \in \mathcal{A}$ yields the dual constraints in the dual LP:

$$\sum_{a \in \mathcal{A}} \lambda(s, a) = \mu(s) + \gamma \sum_{s' \in \mathcal{S}, a' \in \mathcal{A}} P(s | s', a') \lambda^\pi(s', a'), \quad \forall s \in \mathcal{S}$$

Finding the Optimal Policy

Primal LP approach:

- ▶ Solve primal LP to obtain for the optimal value function V^*
- ▶ Then construct the optimal policy (deterministic) through the greedy policy

$$\pi^*(s) = \arg \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^*(s') \right]$$

Dual LP approach:

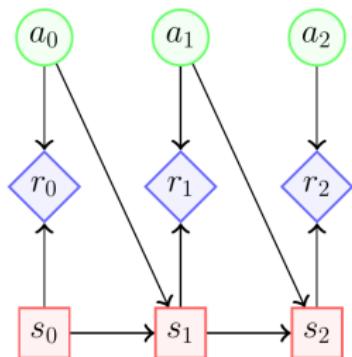
- ▶ Solve the dual LP to obtain the optimal state-action occupancy λ^*
- ▶ Then construct the optimal policy (stochastic) by

$$\pi^*(a | s) = \frac{\lambda^*(s, a)}{\sum_{a \in \mathcal{A}} \lambda^*(s, a)}$$

- ▶ Note that $\lambda^*(s, a) = \lambda^{\pi^*}(s, a)$

Summary I: Markov Decision Processes $M = (\mathcal{S}, \mathcal{A}, P, r, \mu, \gamma)$

state, action	s, a
state space, action space	\mathcal{S}, \mathcal{A}
transition probability	$P(s' s, a)$
reward	$r(s, a)$
discount factor	γ
initial state distribution	$s_0 \sim \mu$
stationary policy (deterministic)	$\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$
stationary policy (stochastic)	$\pi(\cdot s) : \mathcal{S} \rightarrow \Delta(\mathcal{A})$
trajectory	$h = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$
stochastic reward	$r(s, a)$
history at time t	$h_t = (s_{0:t}, a_{0:t-1})$
discount trajectory reward	$R = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$



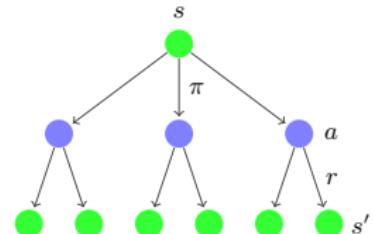
Summary II: Policies and Value functions

state value function	$V^\pi(s) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right]$
state action value function	$Q^\pi(s, a) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right]$
optimal value function	$V^*(s) := \max_{\pi} V^\pi(s)$
optimal state-value function	$Q^*(s, a) = \max_{\pi} Q^\pi(s, a)$
greedy policy	$\pi_Q(s) := \arg \max_{a \in \mathcal{A}} Q(s, a)$
optimal policy	$\pi^* \text{ such that } V^{\pi^*}(s) = V^*(s), \quad \forall s \in \mathcal{S}$
optimal deterministic policy	$\pi^*(s) = \arg \max_{a \in \mathcal{A}} Q^*(s, a)$

Summary III: Bellman Equations

Bellman consistency equation

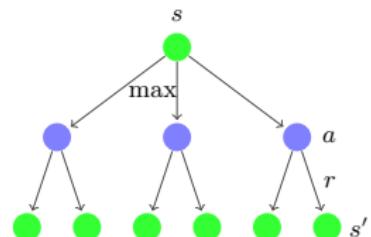
$$V^\pi(s) = \mathcal{T}^\pi V^\pi(s)$$
$$:= \sum_{a \in \mathcal{A}} \pi(a | s) \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^\pi(s') \right]$$



Bellman optimality equation

$$V^*(s) = \mathcal{T}V^*(s)$$
$$:= \max_{a \in \mathcal{A}} \left[r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^*(s') \right]$$

$$Q^*(s, a) = \mathcal{T}_Q Q^*(s, a)$$
$$:= r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) \max_{a' \in \mathcal{A}} Q^*(s', a')$$



Summary IV: Prediction and Control

Policy Evaluation (Prediction)

For a given policy π , estimate

- ▶ state value function $V^\pi(s) : \mathcal{S} \rightarrow \mathbb{R}$
- ▶ state-action value function $Q^\pi(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

Policy Optimization (Control)

Find

- ▶ optimal state value function $V^*(s) : \mathcal{S} \rightarrow \mathbb{R}$
- ▶ optimal state-action value function $Q^*(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
- ▶ optimal policy $\pi^*(s) : \mathcal{S} \rightarrow \mathcal{A}$ (or $\Delta(\mathcal{A})$)

Summary V: Exact Solution Methods of Solving Known MDPs (Planning)

Methods for Policy Evaluation (Prediction)

- ▶ Closed-form solution: $\mathbf{V}^\pi = (\mathbf{I} - \gamma \mathbf{P}^\pi)^{-1} \mathbf{r}^\pi$.
Computation cost: $\mathcal{O}(|\mathcal{S}|^3 + |\mathcal{S}|^2|\mathcal{A}|)$
- ▶ Fixed-point iteration: $\mathbf{V}_{t+1} = \mathcal{T}^\pi \mathbf{V}_t, t = 0, 1, \dots$
Computation cost: $\mathcal{O}((1 - \gamma)^{-1}|\mathcal{S}|^2|\mathcal{A}|)$

Methods for Policy Optimization (Control)

- ▶ **Value Iteration**: first estimate V^* or Q^* , then compute greedy policy
- ▶ **Policy Iteration**: alternate policy evaluation and policy improvement (by greedy approach)
- ▶ **Linear Programming**: solve primal ($\rightarrow V^*$) or dual ($\rightarrow \pi^*$) LP formulation

Summary VI: Dynamic Programming vs Linear Programming

Algorithm	Component	Output
Value Iteration (VI)	Bellman optimality operator \mathcal{T}	V^*
Policy Iteration (PI)	(multiple) Bellman operator \mathcal{T}^π + greedy policy	π^*
Linear Programming (LP)	LP solver (Simplex, Interior point method)	V^*, π^*

Dynamic programming

- ▶ Simple iterative updates
- ▶ Polynomial complexity in $|\mathcal{S}|$ and $|\mathcal{A}|$
- ▶ Works better for large problems

Linear programming

- ▶ Rich library of fast LP solvers
- ▶ Polynomial complexity in $|\mathcal{S}|$ and $|\mathcal{A}|$
- ▶ Works better for small problems

Sommaire

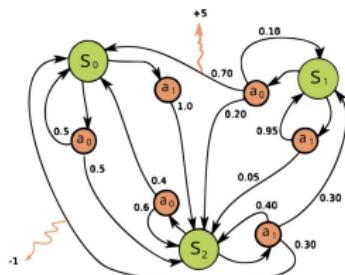
- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
- 5 **Valued-based Reinforcement Learning**
 - Model-free prediction
 - Model-free control
 - Value function approximation
 - Convergence Analysis: Stochastic Approximation and the ODE Method
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
- 8 Imitation learning
- 9 Going beyond

From Planning to Reinforcement Learning

Fundamental challenge #1

The dynamic programming approaches (VI and PI) and linear programming approach all require full knowledge of the transition model P and the reward.

⇒ Learning



From Planning to Reinforcement Learning

Fundamental challenge #1

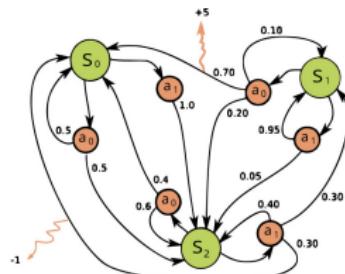
The dynamic programming approaches (VI and PI) and linear programming approach all require full knowledge of the transition model P and the reward.

⇒ Learning

Fundamental challenge #2

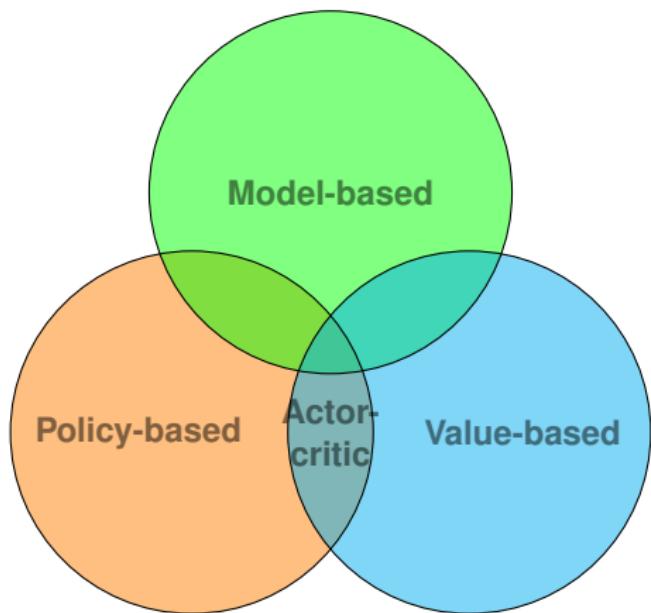
The computation and memory cost can be very expensive for large scale MDP problems.

⇒ Representation



Chess: 10^{50}

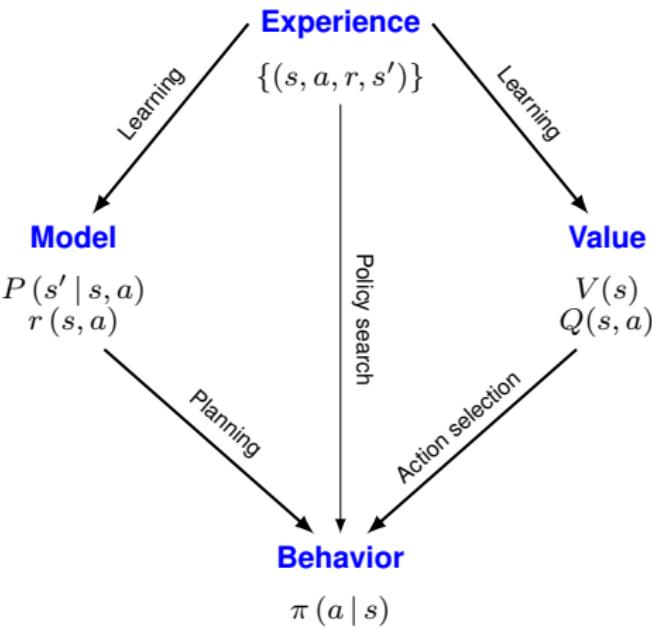
Go: 10^{172}



- **Value-based RL**
 - ▶ Learn the optimal value function V^* , Q^*
 - ▶ Example: Monte Carlo, SARSA, Q-learning, DQN, etc.
 - ▶ Low variance, not scalable to large action space
- **Policy-based RL**
 - ▶ Learn directly the optimal policy π^*
 - ▶ Example: Policy Gradient, NPG, TRPO, PPO, etc.
 - ▶ Scale to large and continuous action space, high variance
- **Model-based RL**
 - ▶ Learn both the model P, r and the optimal policy
 - ▶ Example: Dyna, UCRL2, UCB-VI, etc.
 - ▶ Computationally expensive, better data efficiency

Model-based vs Model-free RL

- Make full use of experiences
- Can reason about model uncertainty
- Sample efficient for easy dynamics



- Direct and simple
- Not affected by poor model estimation
- Computationally efficient

Online vs. Offline RL



Online RL

Collect online data by interacting with environment

Exploitation-exploration tradeoff

Risky to learn from the actual environment



Offline/Batch RL

Use previously collected data (often from behavior policy)

Data is static, no online data collection

Possible delay in the feedback on the learning (batch RL)

Representation Learning



Large (possibly continuous) state and action spaces

Function approximation

$$V(s) \approx V_{\eta}(s)$$

$$Q(s, a) \approx Q_{\eta}(s, a)$$

$$\pi(a | s) \approx \pi_{\theta}(a | s)$$

$$P(s' | s, a) \approx P_{\zeta}(s' | s, a)$$

Representation Learning

Linear function approximation

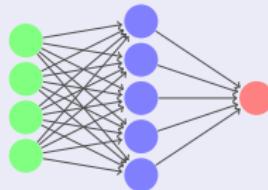
- ▶ Linear combination of basis functions

$$V \approx V_{\eta}(s) = [\phi_1(s), \dots, \phi_d(s)] \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_d \end{bmatrix}$$

- ▶ Reproducing kernel Hilbert space (RKHS) (possibly ∞ -dimensional)

Nonlinear function approximation

- ▶ Fully connected neural networks



- ▶ Convolutional neural networks
- ▶ Residual networks
- ▶ Recurrent networks
- ▶ Self attention
- ▶ Generative adversarial networks

Mean estimation

- ▶ Given a sequence of samples X_1, \dots, X_N , we want to estimate the mean $\mu = \mathbb{E}[X]$.
- ▶ Sample average approximation:

$$\hat{\mu}_N = \frac{1}{N} \sum_{n=1}^N X_n$$

Equivalently

$$\begin{aligned}\hat{\mu}_N &= \frac{1}{N} \left(X_N + (N-1) \frac{1}{N-1} \sum_{n=1}^{N-1} X_n \right) \\ &= \hat{\mu}_{N-1} + \frac{1}{N} (X_N - \hat{\mu}_{N-1})\end{aligned}$$

- ▶ Stochastic approximation:

$$\hat{\mu}_N = \hat{\mu}_{N-1} + \alpha_N (X_N - \hat{\mu}_{N-1})$$

NB: $\hat{\mu}_N \xrightarrow[N \rightarrow \infty]{} \mu$ under Robbins-Monro stepsize, i.e., $\sum_n \alpha_n = \infty$, $\sum_n \alpha_n^2 < \infty$

Model-free Prediction

Goal

Given policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$, estimate $V^\pi(s)$ or $Q^\pi(s, a)$ from episodes of experience under π

$$V^\pi(s) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s, \pi \right]$$

Recall from Bellman consistency equation, we also have

$$V^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot \mid s)} [r(s, a) + \gamma \mathbb{E}_{s' \sim P(s' \mid s, a)} [V^\pi(s')]]$$

Monte Carlo Method

Idea: Estimate $V^\pi(s)$ by the average of returns following all visits to s .

Monte Carlo method

for each episode **do**

 Generate an episode $\{s_0, a_0, r_0, s_1, \dots\}$ following π

for each state s_t **do**

 Compute return $R_t = r_{t+1} + \gamma r_{t+2} + \dots$

 Update counter $n_t(s_t) \leftarrow n_t(s_t) + 1$

 Update $V(s_t) \leftarrow V(s_t) + \frac{1}{n_t} (R_t - V(s_t))$

end for

end for

- ▶ The value estimates are independent and do not build on that of other states (no bootstrap)
- ▶ Learning can be slow when the episodes are long
- ▶ Convergence: MC converges to V^π if each state is visited infinitely often

Temporal difference learning

Idea: Incrementally estimate $V^\pi(s)$ by the intermediate return plus estimated return at next state (sampled by following π)

$$V(s_t) \leftarrow V(s_t) + \alpha_t \underbrace{(r_t + \gamma V(s_{t+1}) - V(s_t))}_{\text{TD error} =: \delta_t} \quad [\text{SGD-like}]$$

Temporal Difference learning (TD(0))

for each step **do**

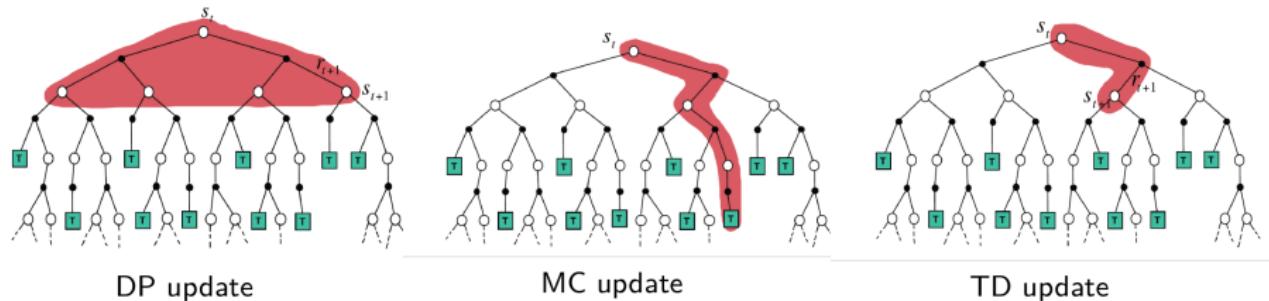
 Observe $\{s_t, a_t, r_t, s_{t+1}\}$ following π

 Update $V(s_t) \leftarrow V(s_t) + \alpha_t (r_t + \gamma V(s_{t+1}) - V(s_t))$

end for

- ▶ Similar to Dynamic Programming: the estimates build on estimates of other states (bootstrap)
- ▶ Similar to Monte Carlo: learn directly from episodes of experiences without knowledge of MDP
- ▶ Unlike MC, can learn from incomplete episodes, applicable to non-terminating environments
- ▶ Convergence: $V \rightarrow V^\pi$ if each state is visited infinitely often and $\alpha_t \rightarrow 0$ at suitable rate

Dynamic Programming vs MC vs TD



- ▶ DynProg: bootstrap, no sampling, exploits Markov property, based on expectation of Values of every next states
- ▶ MC: no bootstrap, sampling, model-free, does not exploit Markov property, 1 trajectory
- ▶ TD: bootstrap, sampling, model-free, online, exploits Markov property

Bias and Variance Trade-off

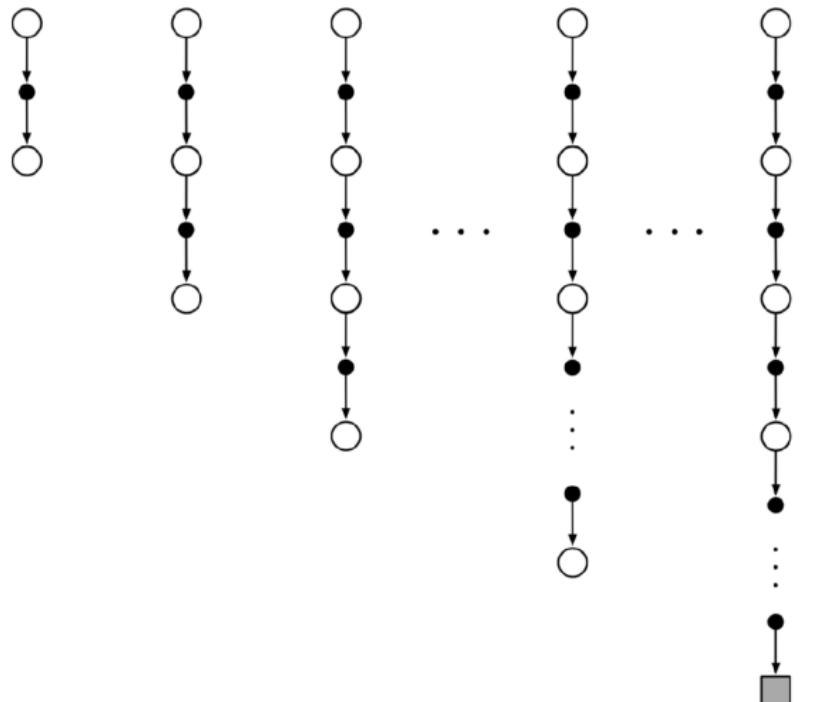
- ▶ MC return is unbiased, but has higher variance since it relies on many random steps
- ▶ TD target is biased, but has lower variance since it only relies on the next step
- ▶ The MC error can be written as a sum of TD errors:

$$\begin{aligned} R_t - V(s_t) &= r_t + \gamma R_{t+1} - V(s_t) + \gamma V(s_{t+1}) - \gamma V(s_{t+1}) \\ &= \delta_t + \gamma (R_{t+1} - V(s_{t+1})) \\ &= \delta_t + \gamma \delta_{t+1} + \gamma^2 (R_{t+2} - V(s_{t+2})) \\ &= \dots \\ &= \sum_{k=t}^{\infty} \gamma^{k-t} \delta_k \end{aligned}$$

Since δ_k are independent → Variance of MC \geq Variance of TD

Bias-Variance Trade-off

1-step TD
and TD(0) 2-step TD 3-step TD n-step TD ∞ -step TD
and Monte Carlo



Biased, low variance

vs

unbiased, high variance

Multiple-step TD Learning

N-step return

Let T be the termination time step in a given episode, $\gamma \in [0, 1]$

$$R_t^{(1)} = r_t + \gamma V(s_{t+1}) \quad \text{TD(0)}$$

$$R_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \quad \text{two-step return}$$

$$R_t^{(N)} = r_t + \gamma r_{t+1} + \dots + \gamma^{N-1} r_{t+N} + \gamma^N V(s_{t+N}) \quad N\text{-step return}$$

$$R_t^{(\infty)} = r_t + \gamma r_{t+1} + \dots + \gamma^{T-t-1} r_T \quad \text{Monte Carlo}$$

NB: $R_t^{(N)} = R_t^\infty$ if $t + N \geq T$

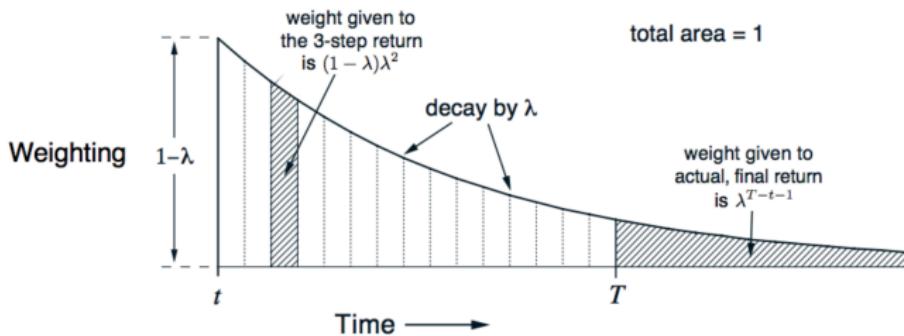
Multi-step TD learning

$$V(s_t) \leftarrow V(s_t) + \alpha_t \underbrace{(R_t^{(N)} - V(s_t))}_{N\text{-step TD error}}$$

- ▶ Unifies and combines TD(0) and MC: $N = 1$ recovers TD(0) and $N \rightarrow \infty$ recovers MC
- ▶ Better balance of bias and variance
- ▶ Need to wait until after seeing r_{t+1}, \dots, r_{t+N}

TD(λ)

TD(λ) unifies and generalizes TD and MC methods.



Weighting given in the λ -return to each of the n -step returns.

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$$

TD(λ)

λ -return (weighted average of all n -step returns)

$$R_t^\lambda = (1 - \gamma) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$$

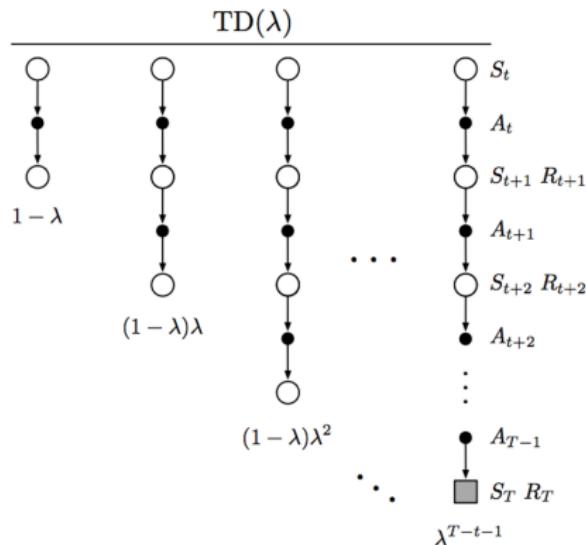
More weight on short horizons since lower variance

TD(λ)

$$V(s_t) \leftarrow V(s_t) + \alpha \left(R_t^\lambda - V(s_t) \right)$$

$\lambda = 0 \longrightarrow \text{TD}(0)$

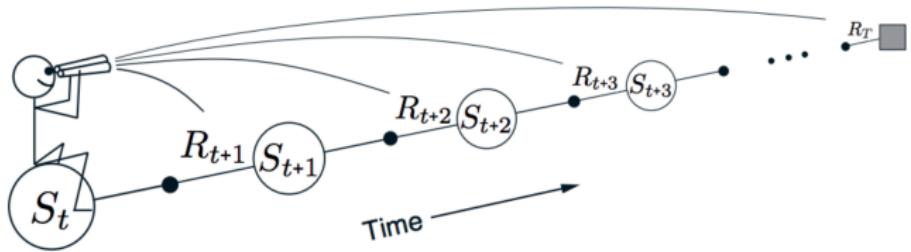
$\lambda = 1 \longrightarrow \text{MC}$



Forward-view TD(λ)

$$V(s_t) \leftarrow V(s_t) + \alpha \left(R_t^\lambda - V(s_t) \right)$$

- ▶ Updates the value function towards the λ -return
- ▶ The forward view looks into the future to compute R_t^λ
- ▶ Like MC, it can only be computed from complete episodes



We decide how to update

each state by looking forward to future rewards and states.

Telescoping in TD(λ)

For general λ , TD errors also telescope to the λ -error $R_t^\lambda - V(s_t)$

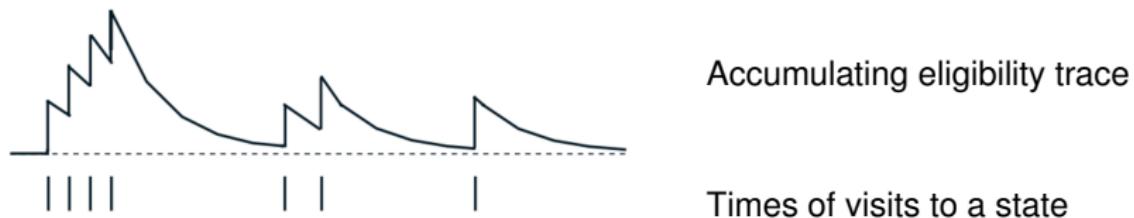
$$\begin{aligned} R_t^\lambda - V(s_t) &= (1 - \lambda) \lambda^0 (r_t + \gamma V(s_{t+1})) \\ &\quad + (1 - \lambda) \lambda^1 (r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})) \\ &\quad + (1 - \lambda) \lambda^2 (r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3})) \\ &\quad + \dots - V(s_t) \\ &= (\gamma \lambda)^0 (r_t + \gamma V(s_{t+1}) - \gamma \lambda V(s_{t+1})) \\ &\quad + (\gamma \lambda)^1 (r_{t+1} + \gamma V(s_{t+2}) - \gamma \lambda V(s_{t+2})) \\ &\quad + (\gamma \lambda)^2 (r_{t+2} + \gamma V(s_{t+3}) - \gamma \lambda V(s_{t+3})) \\ &\quad + \dots - V(s_t) \\ &= (\gamma \lambda)^0 (r_t + \gamma V(s_{t+1}) - V(s_t)) \\ &\quad + (\gamma \lambda)^1 (r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1})) \\ &\quad + (\gamma \lambda)^2 (r_{t+2} + \gamma V(s_{t+3}) - V(s_{t+2})) \\ &\quad + \dots \\ &= \delta_t + \gamma \lambda \delta_{t+1} + (\gamma \lambda)^2 \delta_{t+2} + \dots \end{aligned}$$

Eligibility traces



- Credit assignment problem: did the bell or the light cause the shock?
- Frequency heuristic: assign credit to the most frequent states
- Time heuristic: assign credit to most recent states
- Eligibility traces combine both heuristics

$$e_0(s) = 0, \quad e_t(s) = \gamma \lambda e_{t-1} + \mathbb{1}\{s_t = s\}$$



TD(λ) with Eligibility Trace

TD(λ)

$$V(s_t) \leftarrow V(s_t) + \alpha \left(R_t^\lambda - V(s_t) \right)$$

- ▶ $\lambda = 0$ reduces to TD(0). $\lambda = 1$ reduces to MC
- ▶ Unlike multi-step TD, TD(λ) can be efficiently implemented on the fly:

$$\begin{aligned} V(s) &\leftarrow V(s) + \alpha \delta_t e_t(s) && \text{for all } s \text{ at each } t \\ e_t(s) &= \gamma \lambda e_{t-1}(s) + \mathbb{1}\{s_t = s\} \end{aligned}$$

- ▶ The term $e_t(s) = \sum_{k=0}^t \gamma^{t-k} \mathbb{1}\{s_t = s\}$ is the **eligibility trace**
- ▶ Converge faster than TD(0) when λ is well chosen

Backward-view TD(λ)

- ▶ Keep an eligibility trace $e_t(s)$ for every state s

$$e_0(s) = 0, \quad e_t(s) = \gamma \lambda e_{t-1} + \mathbb{1}\{s_t = s\}$$

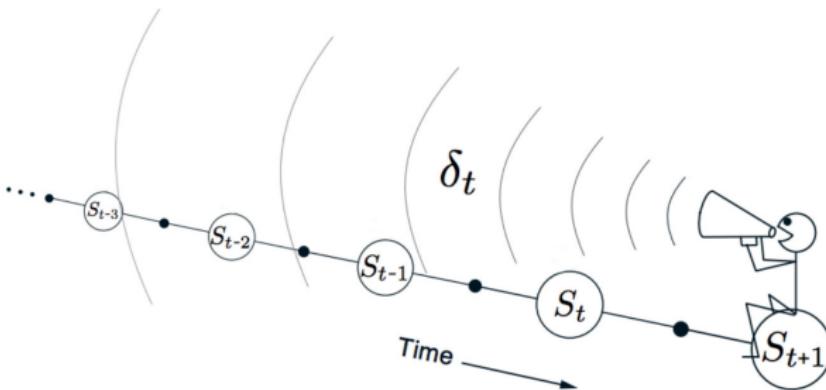
- ▶ The backward view provides a computationally efficient method through eligibility traces.

- ▶ Compute the TD-error δ_t

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

- ▶ Update the value $V(s)$ of **every** state s

$$V(s) \leftarrow V(s) + \alpha \delta_t e_t(s)$$



Each update depends on the current TD error
combined with the current eligibility traces of past events.

Forward- and backward-TD(λ)

- Consider an episode where s is visited only once at time step k
- The eligibility trace of TD(λ) discounts the time since the visit:

$$e_t(s) = \gamma \lambda e_{t-1} + \mathbb{1}_{\{s_t=s\}} = \begin{cases} 0 & \text{if } t < k \\ (\gamma \lambda)^{t-k} & \text{if } t \geq k \end{cases}$$

- Backward-TD(λ) updates accumulate the error online

$$\sum_{t=1}^T \alpha \delta_t e_t(s) = \alpha \sum_{t=k}^T \delta_t (\gamma \lambda)^{t-k} = \alpha \left(R_t^\lambda - V(s_k) \right)$$

- By the end of the episode, they accumulate the total error for the λ -return.

TD(λ) and TD(0)

- ▶ When $\lambda = 0$, only the current state is updated

$$e_t(s) = 1_{\{s_t=s\}}$$
$$V(s) \leftarrow V(s) + \alpha \delta_t e_t(s)$$

- ▶ This is exactly equivalent to the TD(0) update

$$V(s) \leftarrow V(s) + \alpha \delta_t, \quad \text{if } s = s_t$$

- ▶ When $\lambda = 1$, the credit is kept until the end of the episode
- ▶ Consider episodic environments with offline updates
- ▶ Over the course of an episode, the total update for TD(1) is the same as the total update for MC

Theorem

The sum of offline updates is identical for forward-view and backward-view TD(λ)

$$\sum_{t=1}^T \alpha \delta_t e_t(s) = \sum_{t=1}^T \alpha \left(R_t^\lambda - V(s_t) \right) \mathbb{1}_{\{s_t=s\}}$$

Further extension: Sarsa for Q -value estimation

- ▶ In Value Iteration or Policy Iteration, we often require the evaluation of Q -function to compute the greedy policy or optimal policy. Not so easy to derive from V since requires P :

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^\pi(s')$$

- ▶ How do we estimate Q^π directly?

Further extension: Sarsa for Q -value estimation

- ▶ In Value Iteration or Policy Iteration, we often require the evaluation of Q -function to compute the greedy policy or optimal policy. Not so easy to derive from V since requires P :

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s' | s, a) V^\pi(s')$$

- ▶ How do we estimate Q^π directly?

SARSA (Model-free prediction)

for each step **do**

 Observe $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ following π : $a_{t+1} \sim \pi$

 Update $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t (r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$

end for

- ▶ Extends to multi-step SARSA, SARSA(λ), etc.

Summary: model-free prediction

Methods	DynProg	MC	TD(0)
Model knowledge	Need	No need	No need
Bootstrap	Yes	No	Yes
When do updates	After next step	After whole episode	After next step
Use Markov property	Yes	No	Yes
Bias	-	Unbiased	Biased
Variance	-	Large	Small
Convergence rate	Linear rate	-	?

Goal

Find

- ▶ optimal state value function $V^*(s) : \mathcal{S} \rightarrow \mathbb{R}$
- ▶ optimal state-action value function $Q^*(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
- ▶ optimal policy $\pi^*(s) : \mathcal{S} \rightarrow \mathcal{A}$ (or $\Delta(\mathcal{A})$)

Recall: there are three main approaches when dynamics is known:

- ▶ Value Iteration
- ▶ Policy Iteration
- ▶ Linear Programming

Model-free Policy Iteration

Policy Iteration (PI)

Initialize π_0

for each iteration t **do**

[Policy evaluation] **Compute** V^{π_t} or Q^{π_t}

[Policy improvement] Update the policy

$$\pi_{t+1}(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q^{\pi_t}(s, a)$$

end for



Model-free Policy Iteration

Policy Iteration (PI)

```
Initialize  $\pi_0$ 
for each iteration  $t$  do
    [Policy evaluation] Compute  $V^{\pi_t}$  or  $Q^{\pi_t}$ 
    [Policy improvement] Update the policy
```

$$\pi_{t+1}(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q^{\pi_t}(s, a)$$

```
end for
```



Model-free Policy Iteration

```
Initialize  $\pi_0$ 
for each iteration  $t$  do
    [Model-free prediction] Estimate  $V^{\pi_t}$ ,  $Q^{\pi_t}$ 
    [Policy improvement] Update the policy
```

$$\pi_{t+1}(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q^{\pi_t}(s, a)$$

```
end for
```

- ▶ It is better to directly estimate Q^{π_t} instead of V^{π_t} (no required estimation of P as in $V \rightarrow Q$).
- ▶ Model-free prediction: we can use Monte-Carlo method, TD-learning, Sarsa, etc.

Model-free Policy Iteration

Policy Iteration (PI)

```
Initialize  $\pi_0$ 
for each iteration  $t$  do
    [Policy evaluation] Compute  $V^{\pi_t}$  or  $Q^{\pi_t}$ 
    [Policy improvement] Update the policy
```

$$\pi_{t+1}(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q^{\pi_t}(s, a)$$

```
end for
```

Model-free Policy Iteration

```
Initialize  $\pi_0$ 
for each iteration  $t$  do
    [Model-free prediction] Estimate  $V^{\pi_t}$ ,  $Q^{\pi_t}$ 
    [Policy improvement] Update the policy
```

$$\pi_{t+1}(s) \leftarrow \arg \max_{a \in \mathcal{A}} Q^{\pi_t}(s, a)$$

```
end for
```

- ▶ It is better to directly estimate Q^{π_t} instead of V^{π_t} (no required estimation of P as in $V \rightarrow Q$).
- ▶ Model-free prediction: we can use Monte-Carlo method, TD-learning, Sarsa, etc.

Question: what can potentially go wrong with this?

Exploration Scheme

Key Issue:

- ▶ Policy evaluation step requires visiting all state-action pairs infinitely often.
- ▶ The greedy policy lacks exploration; deterministic and cannot sample all (s, a) .

Exploration Scheme

Key Issue:

- ▶ Policy evaluation step requires visiting all state-action pairs infinitely often.
- ▶ The greedy policy lacks exploration; deterministic and cannot sample all (s, a) .

Remedy:

- ▶ ϵ -greedy exploration:

$$\pi_Q^\epsilon(a \mid s) = \begin{cases} \arg \max_{a \in \mathcal{A}} Q(s, a), & \text{with prob. } 1 - \epsilon \\ \mathcal{U}(\mathcal{A}), & \text{with prob. } \epsilon \end{cases}$$

Exploration Scheme

Key Issue:

- ▶ Policy evaluation step requires visiting all state-action pairs infinitely often.
- ▶ The greedy policy lacks exploration; deterministic and cannot sample all (s, a) .

Remedy:

- ▶ ϵ -greedy exploration:

$$\pi_Q^\epsilon(a | s) = \begin{cases} \arg \max_{a \in \mathcal{A}} Q(s, a), & \text{with prob. } 1 - \epsilon \\ \mathcal{U}(\mathcal{A}), & \text{with prob. } \epsilon \end{cases}$$

- ▶ Boltzmann exploration (softmax policy):

$$\pi_Q^\beta(a | s) = \frac{e^{\beta Q(s, a)}}{\sum_{a' \in \mathcal{A}} e^{\beta Q(s, a')}}$$

β is the temperature. $\beta \rightarrow \infty$: greedy ; $\beta \rightarrow 0$: uniform distrib.

The probability of choosing an action increases with the Q -value.

- ▶ Principles of optimism: upper confidence bound (UCB), etc.
Interval estimate of $Q \rightarrow$ pick π based on highest estimate of Q
Also full distrib on V or $Q \longrightarrow$ Truncated Quantile Critics (TQC)

GLIE policy

Desirable property for a policy:

Greedy in the Limit with Infinite Exploration (GLIE)

A GLIE policy satisfies that:

- (i) Each action is visited infinitely often in every state that is visited infinitely often (with prob. 1)
- (ii) The policy converges to a greedy policy (with prob. 1).

- ▶ Example of GLIE: ϵ -greedy policy with $\epsilon \searrow 0$ (not too fast)
- ▶ Example of GLIE: softmax policy with $\beta \nearrow \infty$ (not too fast)
- ▶ Large ϵ leads to exploration; small ϵ leads to exploitation.
- ▶ Small β leads to exploration; large β leads to exploitation.

Examples of GLIE policies

State-specific ϵ -greedy

In ϵ -greedy policy, if we set

$$\epsilon_t(s) = \frac{c}{n_t(s)}, \quad 0 \leq c \leq 1,$$

then it is a GLIE policy.

$n_t(s)$ is the number of times the state s is visited by time t

State-specific Boltzmann exploration

The softmax policy $\pi_t(a | s) = \frac{e^{\beta_t(s)Q(s,a)}}{\sum_{a'} e^{\beta_t(s)Q(s,a')}}$ is GLIE if

$$\beta_t(s) = \frac{\log n_t(s)}{C_t(s)} \quad \text{should not grow too fast}$$

$$C_t(s) \geq \max_{a,a'} |Q_t(s,a) - Q_t(s,a')|.$$

SARSA Algorithm for Model-free Control

- ▶ Model-free policy evaluation: use Sarsa to estimate Q^π
- ▶ Policy improvement: use ϵ -greedy policy or any GLIE policy

SARSA

Repeat for each episode:

 Initialize state s_0

 Choose a_0 from s_0 using a GLIE policy based on Q

 Repeat for each step $t \geq 0$ of episode:

 Take action a_t , observe reward r_t and next state s_{t+1}

 Choose a_{t+1} from s_{t+1} using a GLIE policy based on Q

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t) (r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

 until s_t is terminal

On-policy algorithm

SARSA Algorithm for Model-free Control (cont'd)

Theorem (Convergence of SARSA)

For MDPs with finite states, actions and bounded rewards, if the learning rates satisfy

$$0 \leq \alpha_t(s, a) \leq 1, \quad \sum_t \alpha_t(s, a) = \infty, \quad \sum_t \alpha_t^2(s, a) < \infty$$

where $\alpha_t(s, a) = 0$ unless $(s, a) = (s_t, a_t)$, then SARSA converges to the optimal Q^* .

Extension:

- ▶ Multi-step SARSA
- ▶ SARSA(λ) eligibility traces

Model-free Value Iteration

- ▶ SARSA can be viewed as a model-free policy iteration.
- ▶ What about model-free value iteration?
- ▶ Recall Bellman optimality equation:

$$Q^*(s, a) = \mathcal{T}_Q Q^*(s, a) := r(s, a) + \gamma \mathbb{E}_{s' \sim P(s' | s, a)} \left[\max_{a' \in \mathcal{A}} Q^*(s', a') \right]$$

- ▶ We can estimate $Q^*(s, a)$ similarly as with SARSA and TD-learning from one sample s' .

Q-learning (Watkins, 1989)

A classic algorithm

Q-learning

for each step t **do**

 Observe (s_t, a_t, r_t, s_{t+1})

 Update $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$

end for

- ▶ One of the most popularly used RL algorithm.
- ▶ The observations (s_t, a_t, r_t, s_{t+1}) at time t can come from *any* policy (off-policy).
- ▶ The learned Q -value directly approximates Q^* .

Q-learning (cont'd)

Theorem (Convergence of Q-learning)

If all (s, a) pairs are visited infinitely often and the learning rate $\alpha_t \in (0, 1)$ satisfies

$$\sum_t \alpha_t = \infty, \quad \sum_t \alpha_t^2 < \infty,$$

then the Q-learning algorithm converges to the optimal state-action value function Q^* .

SARSA vs. Q-learning

SARSA:

- ▶ Update rule: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t) (r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$
Based on Bellman consistency equation
- ▶ Action selection: GLIE with respect to Q
- ▶ On-policy control
- ▶ Faster convergence?

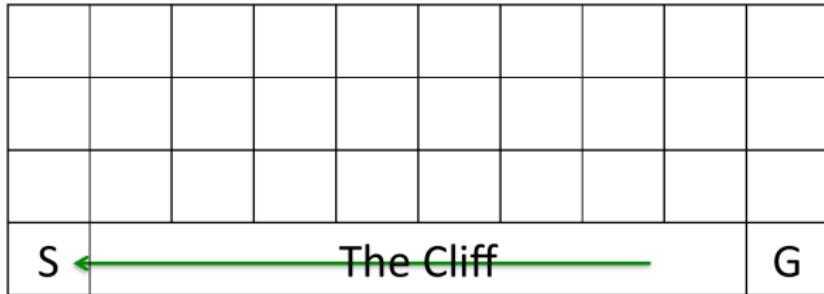
Q-learning:

- ▶ Update rule: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$
Tries to directly estimate the optimal Q^* using Bellman optimality eq.
- ▶ Action selection: taken from any behavior policy (e.g., uniformly randomized policy)
- ▶ Can also apply the standard exploration strategies to generate the observations
- ▶ Off-policy control
- ▶ Slower convergence?

SARSA vs. Q-learning example: Cliff walking

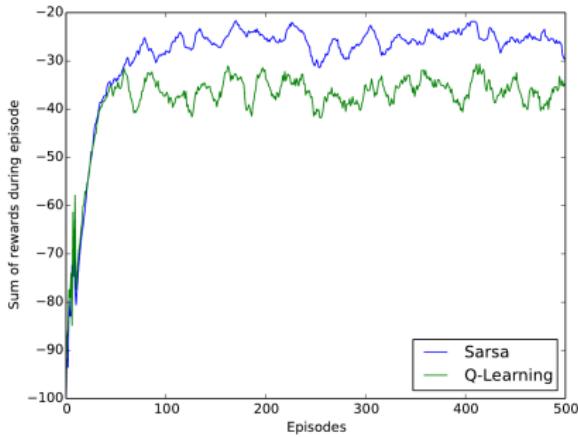
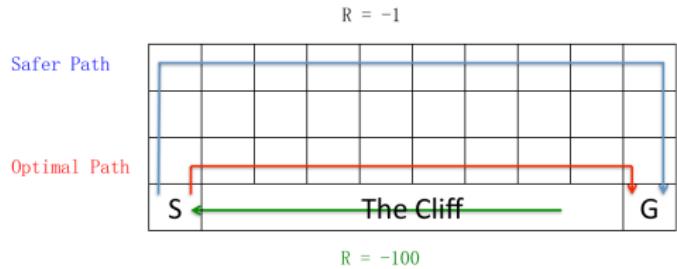
- ▶ Actions $\mathcal{A} = \{\text{up, down, right, left}\}$, starting state S , goal G
- ▶ This is an undiscounted episodic task
- ▶ The reward is -1 on all transitions except the cliff region (-100). Game ends once in G
- ▶ Stepping into the cliff region incurs a reward of -100 and sends the agent instantly back to the start

$$R = -1$$



$$R = -100$$

SARSA vs. Q-learning example: Cliff walking (cont'd)



Here actions are selected based on ϵ -greedy policy for both SARSA and Q-learning algorithms.

Overestimation bias of Q-learning

- ▶ Recall that in Q-learning update

$$\max_{a' \in \mathcal{A}} Q(s_{t+1}, a') = Q(s_{t+1}, \arg \max_{a' \in \mathcal{A}} Q(s_{t+1}, a'))$$

- ▶ The learned Q -values can be noisy.
- ▶ Using the **same** Q -value estimator to select the best action a and evaluate its value can cause upward bias.
- ▶ **Overestimation:** Q-learning tends to overestimate the state-action value function.

NB: If X_1, X_2, \dots, X_N are random variables, then

$$\mathbb{E} \left[\max_n X_n \right] \geq \max_n \mathbb{E} [X_n]$$

From the Jensen's inequality theorem

Double Learning

- ▶ Maximization bias arises because we use the same Q -value to determine the maximization action and estimate its value.
- ▶ Solution: decouple the choice of maximization action and its estimation - learn two independent estimates: $Q_1(s, a)$ and $Q_2(s, a)$
- ▶ Use $Q_1(s, a)$ to select the maximization action
- ▶ Use $Q_2(s, a)$ to estimate the value at the best action

NB: We want to estimate $\max_n \mu_n$, where $\mu_n := \mathbb{E} [X_n]$

- ▶ Single estimator: $\max_n \hat{\mu}_n$
- ▶ Double estimator: First find $n^* = \arg \max_n \hat{\mu}_n^{(1)}$ and then estimate by $\hat{\mu}_{n^*}^{(2)}$.

Double Q-learning (van Hasselt, 2010)

Double Q-learning

for each step t **do**

 Observe (s_t, a_t, r_t, s_{t+1})

 With probability $\beta \in (0, 1)$:

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha_t \left(r_t + \gamma Q_2(s_{t+1}, \arg \max_a Q_1(s_{t+1}, a)) - Q_1(s_t, a_t) \right)$$

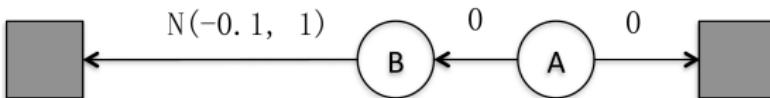
else:

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha_t \left(r_t + \gamma Q_1(s_{t+1}, \arg \max_a Q_2(s_{t+1}, a)) - Q_2(s_t, a_t) \right)$$

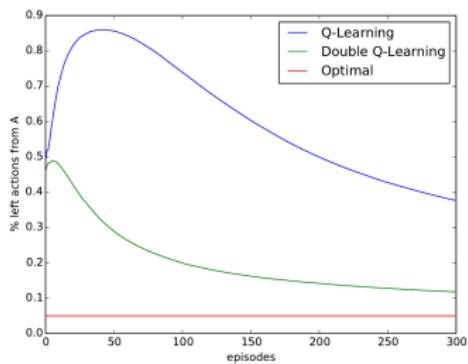
end for

- ▶ Double Q-learning converges to the optimal Q^* under the same conditions as Q-learning.
- ▶ Can output either Q_1 or Q_2 or their average.
- ▶ Can use both or their average to construct ϵ -policy.

Q-Learning vs. Double Q-Learning



- ▶ From state A, a deterministic transition leads to a terminal state assigning reward 0
- ▶ Instead, from state B, the deterministic transition towards the terminal state happens with a stochastic reward sample from a normal distribution with mean -0.1 and variance 1.
- ▶ The optimal action at state A is to take right, which leads to the highest expected return, zero.
- ▶ However, going on the left the agent has the possibility to observe positive rewards.
- ▶ These positive rewards can deceive Q-Learning to favor the left action, which is sub-optimal.



Q-learning initially learns to take the left action much more often than the right action.

Summary: Model-free Control

Methods	DynProg	SARSA	Q-learning / double Q-learning
Model knowledge	Need	No need	No need
Bootstrap	Yes	Yes	Yes
On/off-policy	-	on-policy	off-policy
Convergence rate	Linear rate	?	?

Extensions: SARSA(λ), double SARSA, etc.

Summary: Model-free Control

Methods	DynProg	SARSA	Q-learning / double Q-learning
Model knowledge	Need	No need	No need
Bootstrap	Yes	Yes	Yes
On/off-policy	-	on-policy	off-policy
Convergence rate	Linear rate	?	?

Extensions: SARSA(λ), double SARSA, etc.

What if the state space is very large?

So far, Model-free Prediction works for tabular setting

Monte Carlo method

$$V(s_t) \leftarrow V(s_t) + \frac{1}{n_t} \left(\sum_{i=t}^{\infty} \gamma^{i-t} r_i - V(s_t) \right)$$

TD(0)

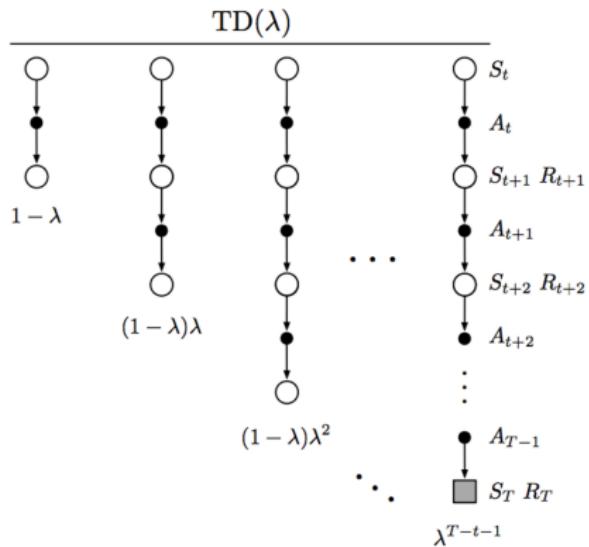
$$V(s_t) \leftarrow V(s_t) + \alpha_t (r_t + \gamma V(s_{t+1}) - V(s_t))$$

TD(λ)

$$V(s_t) \leftarrow V(s_t) + \alpha_t \left(R_t^\lambda - V(s_t) \right)$$

Recall $R_t^\lambda = (1 - \gamma) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$

Balance bias/variance tradeoff between MC and TD(0)



So far, Model-free Control works for tabular setting

SARSA (on-policy)

Select $(s_t, a_t, s_{t+1}, a_{t+1})$ from GLIE policy

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t (r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Q-learning (off-policy)

Select (s_t, a_t, s_{t+1}) from any behavior policy

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right)$$

Double Q-learning (off-policy)

Randomly update

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha_t \left(r_t + \gamma Q_2(s_{t+1}, \arg \max_a Q_1(s_{t+1}, a)) - Q_1(s_t, a_t) \right)$$

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha_t \left(r_t + \gamma Q_1(s_{t+1}, \arg \max_a Q_2(s_{t+1}, a)) - Q_2(s_t, a_t) \right)$$

From Planning to Reinforcement Learning

Fundamental Challenge 1

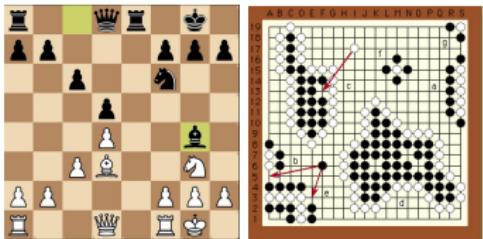
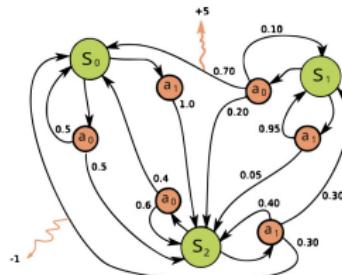
The dynamic programming and linear programming approaches all require **full knowledge** of the transition model P and the reward.

⇒ Learning

Fundamental Challenge 2

The computation and memory cost can be very expensive for large scale MDP problems.

⇒ Representation



The Need from Tabular Setting to Function Approximation

So far, MC and TD learning are done via a lookup table

- ▶ There are too many states and/or actions to store in memory
- ▶ It is slow to learn the value of each state individually
- ▶ We need to generalize to states with rare observations or new states

Value function approximation

$$V(s) \approx V_{\eta}(s)$$

$$Q(s, a) \approx Q_{\eta}(s, a)$$



⇒

Large state and action spaces

- ▶ Linear function approximation

$$V_{\eta}(s) = \phi(s)^T \eta$$

- ▶ Neural networks
- ▶ Decision trees
- ▶ Tensor functions
- ▶ ...

Learn $\eta \in \mathbb{R}^d$

- ▶ MC
- ▶ TD
- ▶ Q-learning
- ▶ ...

$$d \ll |\mathcal{S}|$$

Recall Supervised Learning

Supervised learning seeks to find the predictor $h_{\eta}(\cdot)$ that minimizes the expected risk or empirical risk:

$$\min_{\eta} \mathbb{E}_{(\mathbf{x}, y)} [\ell(h_{\eta}(\mathbf{x}), y)] \approx \frac{1}{N} \sum_{n=1}^N \ell(h_{\eta}(\mathbf{x}_n), y_n)$$

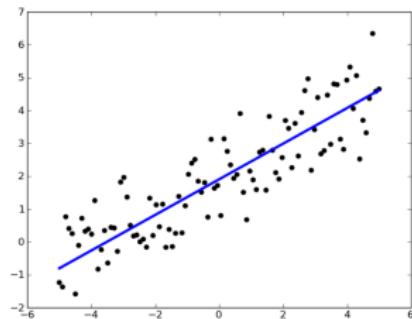
Recall Supervised Learning

Supervised learning seeks to find the predictor $h_{\eta}(\cdot)$ that minimizes the expected risk or empirical risk:

$$\min_{\eta} \mathbb{E}_{(\mathbf{x}, y)} [\ell(h_{\eta}(\mathbf{x}), y)] \approx \frac{1}{N} \sum_{n=1}^N \ell(h_{\eta}(\mathbf{x}_n), y_n)$$

Least squares regression with an affine model:

- ▶ Objective: $\min_{\eta} \mathbb{E}_{(\mathbf{x}, y)} \left[\frac{1}{2} \left(\eta^T (\mathbf{x}, 1) - y \right)^2 \right]$
- ▶ Stochastic Gradient Descent:
$$\eta_{t+1} \leftarrow \eta_t - \alpha_t (\eta_t^T (\mathbf{x}, 1) - y) (\mathbf{x}, 1), \quad t = 0, 1, 2, \dots$$



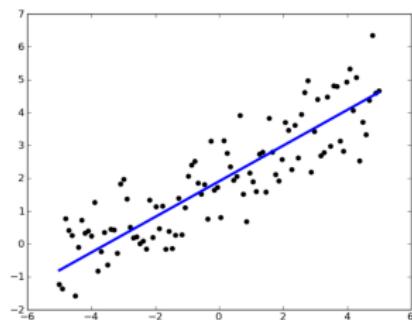
Recall Supervised Learning

Supervised learning seeks to find the predictor $h_{\eta}(\cdot)$ that minimizes the expected risk or empirical risk:

$$\min_{\eta} \mathbb{E}_{(\mathbf{x}, y)} [\ell(h_{\eta}(\mathbf{x}), y)] \approx \frac{1}{N} \sum_{n=1}^N \ell(h_{\eta}(\mathbf{x}_n), y_n)$$

Least squares regression with an affine model:

- ▶ Objective: $\min_{\eta} \mathbb{E}_{(\mathbf{x}, y)} \left[\frac{1}{2} \left(\eta^T (\mathbf{x}, 1) - y \right)^2 \right]$
- ▶ Stochastic Gradient Descent:
$$\eta_{t+1} \leftarrow \eta_t - \alpha_t (\eta_t^T (\mathbf{x}, 1) - y) (\mathbf{x}, 1), \quad t = 0, 1, 2, \dots$$



What are the differences from learning value functions in RL?

In RL, no labels, no data.

Prediction: How to estimate value function V^π given policy π ?

$$V^\pi(s) = \mathbb{E}_{s' \in \mathcal{S}, a \sim \pi(\cdot | s)} [r(s, a) + \gamma V^\pi(s')]$$

Prediction: How to estimate value function V^π given policy π ?

$$V^\pi(s) = \mathbb{E}_{s' \in \mathcal{S}, a \sim \pi(\cdot | s)} [r(s, a) + \gamma V^\pi(s')]$$

One could think of

$$\boldsymbol{\eta} \in \arg \max_{\boldsymbol{\eta}} \frac{1}{2} (V_{\boldsymbol{\eta}}^\pi - \mathbb{E} [r + \gamma V_{\boldsymbol{\eta}}^\pi])^2$$

Prediction: How to estimate value function V^π given policy π ?

$$V^\pi(s) = \mathbb{E}_{s' \in \mathcal{S}, a \sim \pi(\cdot | s)} [r(s, a) + \gamma V^\pi(s')]$$

One could think of

$$\boldsymbol{\eta} \in \arg \max_{\boldsymbol{\eta}} \frac{1}{2} (V_{\boldsymbol{\eta}}^\pi - \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi])^2$$



$$\mathbf{0} = V_{\boldsymbol{\eta}}^\pi \nabla_{\boldsymbol{\eta}} V_{\boldsymbol{\eta}}^\pi - V_{\boldsymbol{\eta}}^\pi \nabla_{\boldsymbol{\eta}} \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi] - \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi] \nabla_{\boldsymbol{\eta}} V_{\boldsymbol{\eta}}^\pi + \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi] \nabla_{\boldsymbol{\eta}} \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi]$$

Prediction: How to estimate value function V^π given policy π ?

$$V^\pi(s) = \mathbb{E}_{s' \in \mathcal{S}, a \sim \pi(\cdot | s)} [r(s, a) + \gamma V^\pi(s')]$$

One could think of

$$\boldsymbol{\eta} \in \arg \max_{\boldsymbol{\eta}} \frac{1}{2} (V_{\boldsymbol{\eta}}^\pi - \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi])^2$$



$$\mathbf{0} = V_{\boldsymbol{\eta}}^\pi \nabla_{\boldsymbol{\eta}} V_{\boldsymbol{\eta}}^\pi - V_{\boldsymbol{\eta}}^\pi \nabla_{\boldsymbol{\eta}} \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi] - \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi] \nabla_{\boldsymbol{\eta}} V_{\boldsymbol{\eta}}^\pi + \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi] \nabla_{\boldsymbol{\eta}} \mathbb{E}[r + \gamma V_{\boldsymbol{\eta}}^\pi]$$

- ▶ $\mathbb{E}[\cdot] \nabla_{\boldsymbol{\eta}} \mathbb{E}[\cdot] \implies$ Requires to sample twice the future for a given action:
impossible
- ▶ → Keep RHS fixed and adjust LHS via $\boldsymbol{\eta}$
- ▶ Not a SGD, no convergence guarantees

Prediction: How to estimate value function V^π given policy π ?

Objective I: Minimize mean-squares error (MSE)

$$\min_{\boldsymbol{\eta}} J_1(\boldsymbol{\eta}) = \frac{1}{2} \mathbb{E}_{s \sim \mathcal{D}_\mu^\pi} \left[(V_{\boldsymbol{\eta}}(s) - V^\pi(s))^2 \right] := \frac{1}{2} \|\mathbf{V}_{\boldsymbol{\eta}} - \mathbf{V}^\pi\|_{D_s}^2$$

- ▶ Here \mathcal{D}_μ^π is the stationary state distribution under policy π , $\|\mathbf{x}\|_{D_s}^2 = \mathbf{x}^\top D_s \mathbf{x}$ with $D_s = \text{diag}(\mathcal{D}_\mu^\pi)$
- ▶ Stationary $\implies \mathcal{D}_\mu^\pi$ does not depend on t [iid samples]
- ▶ The objective is convex under linear function approximation, but nonconvex in general

Prediction: How to estimate value function V^π given policy π ?

Objective I: Minimize mean-squares error (MSE)

$$\min_{\boldsymbol{\eta}} J_1(\boldsymbol{\eta}) = \frac{1}{2} \mathbb{E}_{s \sim \mathcal{D}_\mu^\pi} [(V_{\boldsymbol{\eta}}(s) - V^\pi(s))^2] := \frac{1}{2} \| \mathbf{V}_{\boldsymbol{\eta}} - \mathbf{V}^\pi \|_{D_s}^2$$

- ▶ Here \mathcal{D}_μ^π is the stationary state distribution under policy π , $\|x\|_{D_s}^2 = \mathbf{x}^\top D_s \mathbf{x}$ with $D_s = \text{diag}(\mathcal{D}_\mu^\pi)$
- ▶ Stationary $\implies \mathcal{D}_\mu^\pi$ does not depend on t [iid samples]
- ▶ The objective is convex under linear function approximation, but nonconvex in general
- ▶ Gradient: $\nabla J_1(\boldsymbol{\eta}) = \mathbb{E}_{s \sim \mathcal{D}_\mu^\pi} [(V_{\boldsymbol{\eta}}(s) - V^\pi(s)) \nabla V_{\boldsymbol{\eta}}(s)]$
- ▶ \mathcal{D}_μ^π unknown $\xrightarrow{\text{blue}}$ stochastic gradient: $\nabla J_1(\boldsymbol{\eta}) = (V_{\boldsymbol{\eta}}(s_t) - V^\pi(s_t)) \nabla V_{\boldsymbol{\eta}}(s_t)$ where $s_t \sim \mathcal{D}_\mu^\pi$
- ▶ Estimate $V^\pi(s_t)$:
 - ▶ Monte Carlo return
 - ▶ TD target
 - ▶ λ -return

Model-free Prediction with Function Approximation

Monte Carlo with function approximation (= SGD)

$$\boldsymbol{\eta}_{t+1} \leftarrow \boldsymbol{\eta}_t - \alpha_t (V_{\boldsymbol{\eta}_t}(s_t) - \textcolor{blue}{R}_t) \nabla V_{\boldsymbol{\eta}_t}(s_t)$$

TD with function approximation (\approx SGD)

$$\boldsymbol{\eta}_{t+1} \leftarrow \boldsymbol{\eta}_t - \alpha_t (V_{\boldsymbol{\eta}_t}(s_t) - (r_t + \gamma V_{\boldsymbol{\eta}_t}(s_{t+1}))) \nabla V_{\boldsymbol{\eta}_t}(s_t)$$

TD(λ) with function approximation (\approx SGD)

$$\boldsymbol{\eta}_{t+1} \leftarrow \boldsymbol{\eta}_t - \alpha_t \left(V_{\boldsymbol{\eta}_t}(s_t) - \textcolor{blue}{R}_t^\lambda \right) \nabla V_{\boldsymbol{\eta}_t}(s_t)$$

Remarks

- ▶ MC can be viewed as SGD with unbiased gradient
 - ▶ Unbiased but suffers from high variance when the horizon is long
 - ▶ With linear FA, converges to global optimum (*i.e.*, projection on the linear span of feature vectors)
 - ▶ With nonlinear FA, converges to some stationary point
- ▶ TD is a semi-gradient method
 - ▶ Can also be viewed as regression with moving target (since it depends on η_t)
 - ▶ In general, does not always converge
 - ▶ Converges reliably with linear FA. [If so, where does it converge to? And how well?]

Linear TD

- Consider linear function approximation: $V_{\eta}(s) = \phi(s)^T \eta$
- Denote feature matrix $\Phi \in \mathbb{R}^{|S| \times d}$ such that $V_{\eta} = \Phi \eta$ and assume Φ has full column rank.
- Linear TD-learning update is

$$\eta_{t+1} \leftarrow \eta_t + \alpha_t \underbrace{\left(r_t + \gamma \phi(s_{t+1})^T \eta_t - \phi(s_t)^T \eta_t \right)}_{g(\eta_t)} \phi(s_t)$$

Linear TD

- Consider linear function approximation: $V_{\eta}(s) = \phi(s)^T \eta$
- Denote feature matrix $\Phi \in \mathbb{R}^{|S| \times d}$ such that $V_{\eta} = \Phi \eta$ and assume Φ has full column rank.
- Linear TD-learning update is

$$\eta_{t+1} \leftarrow \eta_t + \alpha_t \underbrace{\left(r_t + \gamma \phi(s_{t+1})^T \eta_t - \phi(s_t)^T \eta_t \right)}_{g(\eta_t)} \phi(s_t)$$

- The limit point η^* of linear TD should satisfy:

$$\bar{g}(\eta^*) := \mathbb{E}_{s,s'} \left[\left(r + \gamma \phi(s')^T \eta^* - \phi(s)^T \eta^* \right) \phi(s) \right] = \Phi^T D_s (\mathcal{T}^\pi \Phi \eta^* - \Phi \eta^*) = 0$$

Here \mathcal{T}^π is Bellman consistency operator under policy π .

The limit point is the unique fixed point to the projected Bellman equation.

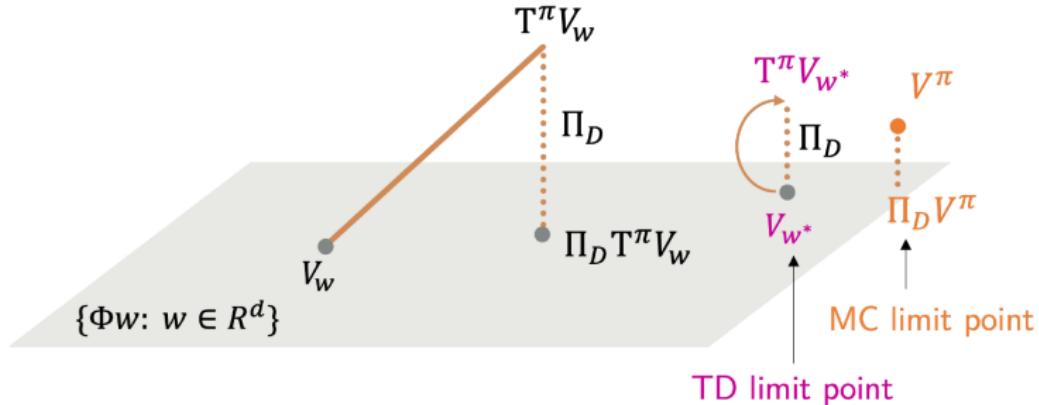
Projected Bellman equation

$$\Phi\boldsymbol{\eta} = \Pi_{D_s} \mathcal{T}^\pi \Phi\boldsymbol{\eta}$$

- ▶ The linear span of feature vectors: $\text{range}(\Phi) = \{\Phi\boldsymbol{\eta} : \boldsymbol{\eta} \in \mathbb{R}^d\}$
- ▶ Projection operator onto the range of
 $\Phi : \Pi_{D_s}(\mathbf{x}) := \arg \min_{\mathbf{x}' \in \text{range}(\Phi)} \|\mathbf{x} - \mathbf{x}'\|_{D_s}^2.$

Proof: Follows immediately from the fact: $\langle \Phi\boldsymbol{\eta}, \mathcal{T}^\pi \Phi\boldsymbol{\eta}^* - \Phi\boldsymbol{\eta}^* \rangle_{D_s} = 0, \forall \boldsymbol{\eta}$

Linear TD vs MC



[The sketch uses w instead of η]

- ▶ The value of MC iterate converges to $\Pi_{D_s} V^\pi$, i.e., the best approximation of V^π in the linear span.
- ▶ The value of TD iterates converges to V_{η^*} such that $V_{\eta^*} = \Pi_{D_s} T^\pi V_{\eta^*}$.

Linear TD Learning (cont'd)

Theorem: Performance bound

Denote $\boldsymbol{\eta}^*$ as the fixed point to the projected Bellman equation and $\mathbf{V}_{\boldsymbol{\eta}^*} = \Phi \boldsymbol{\eta}^*$.

$$\|\mathbf{V}_{\boldsymbol{\eta}^*} - \mathbf{V}^\pi\|_{D_s} \leq \frac{1}{1-\gamma} \|\mathbf{V}^\pi - \Pi_{D_s} \mathbf{V}^\pi\|_{D_s}$$

- ▶ TD solution is not worse than the MC solution with a factor $1/(1-\gamma)$
- ▶ If the true value function \mathbf{V}^π falls within the span of the features, then both MC and TD converge to it.

Alternative Optimization and TD-type Algorithms

Objective I: Minimize mean-square error (MSE)

$$\min_{\boldsymbol{\eta}} J_1(\boldsymbol{\eta}) := \frac{1}{2} \|\mathbf{V}_{\boldsymbol{\eta}} - \mathbf{V}^{\pi}\|_{D_s}^2$$

⇒ MC

Objective II: Minimize mean-square Bellman error

$$\min_{\boldsymbol{\eta}} J_2(\boldsymbol{\eta}) := \frac{1}{2} \|\mathbf{V}_{\boldsymbol{\eta}} - \mathcal{T}^{\pi} \mathbf{V}_{\boldsymbol{\eta}}\|_{D_s}^2$$

⇒ TD learning, GTD

(Gradient descent Temporal Difference)

Objective III: Minimize mean-square projected Bellman error

$$\min_{\boldsymbol{\eta}} J_3(\boldsymbol{\eta}) := \frac{1}{2} \|\mathbf{V}_{\boldsymbol{\eta}} - \Pi_{D_s} \mathcal{T}^{\pi} \mathbf{V}_{\boldsymbol{\eta}}\|_{D_s}^2$$

⇒ GTD2 (gradient

correction), TDC

Control: How to estimate optimal state-action value function Q^* ?

- ▶ Similarly, we can extend Q-learning from tabular settings to function approximation:

Q-learning with function approximation

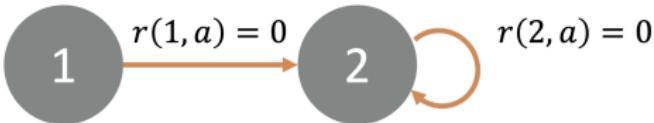
$$\boldsymbol{\eta}_{t+1} \leftarrow \boldsymbol{\eta}_t + \alpha_t \left(r_t + \gamma \max_a [Q_{\boldsymbol{\eta}_t}(s_{t+1}, a)] - Q_{\boldsymbol{\eta}_t}(s_t, a_t) \right) \nabla Q_{\boldsymbol{\eta}_t}(s_t, a_t)$$

- ▶ With linear function approximation $Q_{\boldsymbol{\eta}}(s, a) = \phi(s, a)^T \boldsymbol{\eta}$, we have

$$\boldsymbol{\eta}_{t+1} \leftarrow \boldsymbol{\eta}_t + \alpha_t \left(r_t + \gamma \max_a [\phi(s_{t+1}, a)^T \boldsymbol{\eta}] - \phi(s_t, a_t)^T \boldsymbol{\eta} \right) \phi(s_t, a_t)$$

- ▶ Off-policy Q-learning with linear function approximation does not always converge!

A simple example: Divergence with Off-policy sampling

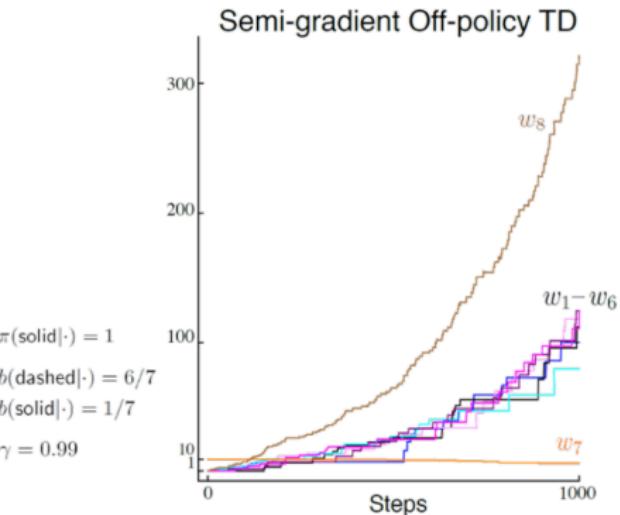
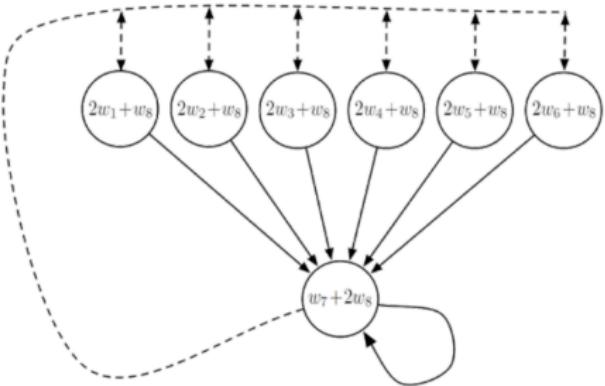


- ▶ Two states, one action, $r(1, a) = r(2, a) = 0$
- ▶ True value function: $V(1) = V(2) = 0$
- ▶ Consider linear function approximation $V_{\eta}(1) = \eta, V_{\eta}(2) = 2\eta$ (namely, $\phi(s) \equiv s$)
- ▶ Off-policy sampling: only observe transitions from state 1 to state 2
$$\eta_{t+1} \leftarrow \eta_t + \alpha (0 + \gamma 2\eta_t - \eta_t) \quad \Rightarrow \quad \eta_{t+1} \leftarrow (1 + \alpha(2\gamma - 1)) \eta_t$$
- ▶ Hence, $\eta_t \rightarrow \infty$ whenever $\gamma > 1/2$ under any stepsize, $\alpha > 0$

Two reasons:

- ▶ parametrization is very poor ($\phi(s) \equiv s$)
- ▶ only partially observe the system: only 1 transition

A richer example: Divergence with Off-policy sampling



7 states, 8 parameters ($d > |\mathcal{S}|$) → Value function is well represented and the true V can be found.

Want to evaluate V^π but data are from b

“Deadly Triad”

Deadly triad

Divergence is likely to happen when we combine bootstrapping, off-policy learning, and function approximations.

- ▶ When using function approximation, updating the parameters from one state creates a risk of inappropriately changing the values of other states.
- ▶ If the agent is learning off-policy, it might not update the bootstrap values sufficiently often.
- ▶ Divergence can happen even with linear function approximation or TD learning.
- ▶ Can be alleviated by using λ -return (MC-like), ϵ -greedy policy, or larger networks.

Stochastic Approximation

- ▶ Suppose we would like to solve a system of equations:

$$h(\boldsymbol{x}) = \mathbf{0}$$

- ▶ Example: $h(\boldsymbol{x}) = -\nabla f(\boldsymbol{x})$ (stationary point) or $h(\boldsymbol{x}) = \mathcal{T}\boldsymbol{x} - \boldsymbol{x}$ (fixed point of \mathcal{T}).

Stochastic Approximation

- ▶ Suppose we would like to solve a system of equations:

$$h(\boldsymbol{x}) = \mathbf{0}$$

- ▶ Example: $h(\boldsymbol{x}) = -\nabla f(\boldsymbol{x})$ (stationary point) or $h(\boldsymbol{x}) = \mathcal{T}\boldsymbol{x} - \boldsymbol{x}$ (fixed point of \mathcal{T}).

Stochastic Approximation

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k (h(\boldsymbol{x}_k) + \boldsymbol{\xi}_k), \quad k = 0, 1, \dots$$

Assume the noise sequence satisfies

- ▶ Zero mean: $\mathbb{E}[\boldsymbol{\xi}_k | \mathcal{F}_k] = \mathbf{0}$
- ▶ Bounded variance (relative to magnitude of \boldsymbol{x}_k): $\mathbb{E}[\|\boldsymbol{\xi}_k\|^2 | \mathcal{F}_k] \leq C_0 (1 + \|\boldsymbol{x}_k\|^2)$

Here \mathcal{F}_k is the σ -algebra generated by $\{\boldsymbol{x}_0, \boldsymbol{\xi}_0, \dots, \boldsymbol{x}_k\}$.

- ▶ The asymptotic behavior of SA is closely related to the solution of the mean-path ODE on h .

- ▶ Consider the ordinary differential equation (ODE):

$$\frac{d}{dt} \mathbf{x}(t) = h(\mathbf{x}(t)) \quad \text{or} \quad \dot{\mathbf{x}} = h(\mathbf{x})$$

- ▶ Let \mathbf{x}^* be an equilibrium point such that $h(\mathbf{x}^*) = 0$
- ▶ \mathbf{x}^* is **asymptotically stable** if $\mathbf{x}(t) \rightarrow \mathbf{x}^*$ for any initial point in a neighborhood of \mathbf{x}^* .
- ▶ \mathbf{x}^* is **globally asymptotically stable** if $\mathbf{x}(t) \rightarrow \mathbf{x}^*$ for any initial point.

Example: Linear System

Consider the linear system

$$\dot{\mathbf{x}} = A\mathbf{x}$$

- ▶ If A is non-singular, it has a unique equilibrium point $\mathbf{x}^* = 0$.
- ▶ If A is singular, there exists infinitely many equilibrium points.
- ▶ The solution is $\mathbf{x}(t) = e^{At}\mathbf{x}(0)$.

Necessary and sufficient condition for Asymptotic Stability

The origin is a globally asymptotically stable equilibrium point if and only if A is Hurwitz, namely, all eigenvalues of A have strictly negative real parts.

Example: Nonlinear System

Consider the nonlinear system

$$\dot{\mathbf{x}} = h(\mathbf{x})$$

Sufficient Condition for Asymptotic Stability

The origin is a globally asymptotically stable equilibrium point if there exists a twice differentiable Lyapunov function $V(\mathbf{x})$ such that

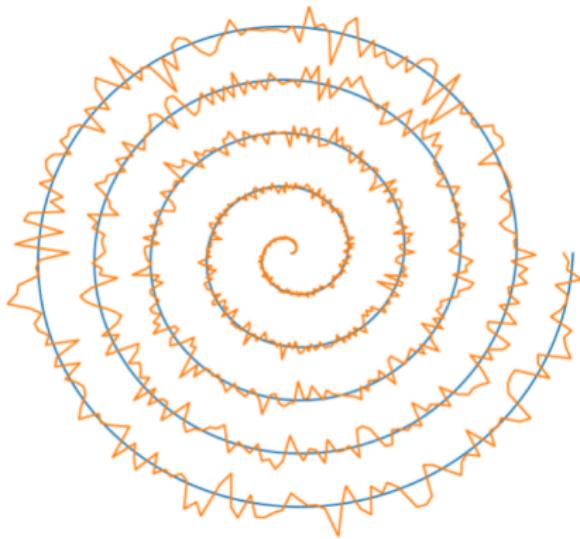
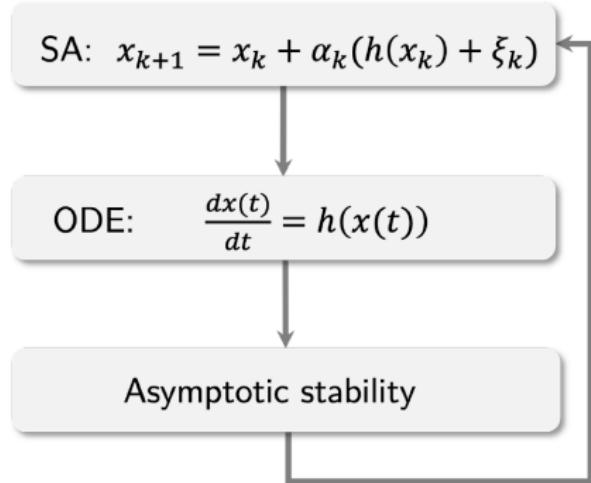
$$c_1 \|\mathbf{x}\|^\alpha \leq V(\mathbf{x}) \leq c_2 \|\mathbf{x}\|^\alpha$$

$$\frac{dV}{d\mathbf{x}} h(\mathbf{x}) \leq -c_3 \|\mathbf{x}\|^\alpha$$

for some positive constants $\alpha, c_1, c_2, c_3 > 0$.

NB: Easy to verify that $V(0) = 0$, $\frac{dV(\mathbf{x}(t))}{dt} \leq 0$, and $V(\mathbf{x}) \rightarrow \infty$ if $\mathbf{x} \rightarrow \infty$. Monotonic decrease over time and bounded from below $\implies 0$ is a stable equilibrium point.

The ODE method



- ▶ One can analyze the convergence behavior of this stochastic approximation by tracking stability of these underlying ODEs.
- ▶ If asymptotically stable \implies stochastic approximation will approximate the ODE (in expectation) and will converge asymptotically.

The ODE Method

Theorem (Borkar & Meyn)

Suppose

- ▶ Function h is globally Lipschitz continuous
- ▶ Stepsize satisfies $\sum_{k=0}^{\infty} \alpha_k = \infty$, $\sum_{k=0}^{\infty} \alpha_k^2 < \infty$
- ▶ The limiting ODE $\dot{x} = h_{\infty}(x) := \lim_{c \rightarrow \infty} \frac{h(cx)}{c}$ (scaled version) is asymptotically stable at the origin
- ▶ The original ODE $\dot{x} = h(x)$ has an equilibrium point x^* , which is globally asymptotically stable,

then we have that

$$x_k \rightarrow x^* \quad \text{as} \quad k \rightarrow \infty, \text{ for any initial } x_0$$

Example: Convergence of Linear TD Learning

- ▶ Recall TD update with linear function approximation:

$$\boldsymbol{\eta}_{k+1} = \boldsymbol{\eta}_k + \alpha_k \left(r_k + \gamma \boldsymbol{\phi}(s_{k+1})^\top \boldsymbol{\eta}_k - \boldsymbol{\phi}(s_k)^\top \boldsymbol{\eta}_k \right) \boldsymbol{\phi}(s_k)$$

- ▶ Taking expectation over \mathcal{S} , the corresponding ODE is

$$\dot{\boldsymbol{\eta}} = \Phi^\top D_s (\gamma P^\pi - \mathbf{I}) \Phi \boldsymbol{\eta} + \Phi^\top D_s \mathbf{r}$$

or equivalently, in a non-affine form

$$\frac{d}{dt}(\boldsymbol{\eta}(t) - \boldsymbol{\eta}^*) = \underbrace{\Phi^\top D_s (\gamma P^\pi - \mathbf{I})}_{A} (\boldsymbol{\eta}(t) - \boldsymbol{\eta}^*)$$

where $\boldsymbol{\eta}^*$ is the fixed point of the projected Bellman equation.

P is a stochastic matrix $\implies \gamma P^\pi - \mathbf{I} < 0$ for $0 \leq \lambda \leq 1$

- ▶ If Φ is full rank, then the matrix A is negative definite, thus also Hurwitz.
- ▶ This implies that the ODE is globally asymptotically stable.

Convergence of Linear TD Learning (cont'd)

Theorem: Convergence of Linear TD Learning

Suppose Φ is full rank and the stepsize satisfies:

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty$$

Then the iterates of linear TD converge: $\eta_k \rightarrow \eta^*$

- ▶ Immediate result by invoking Borkar & Meyn Theorem.
- ▶ Applicable to many other TD-type algorithms with linear function approximation.

Convergence of Q-Learning

- ▶ First, consider the **synchronous** version of Q-learning with **all** (s, a) updated at each iteration:

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha_t \left[r(s, a) + \gamma \max_a Q_k(s', a) - Q_k(s, a) \right]$$

- ▶ The corresponding ODE, which is now a **nonlinear** system, is

$$\dot{\mathbf{Q}} = \mathcal{T}\mathbf{Q} - \mathbf{Q} := (\gamma P^{\pi_Q} - \mathbf{I})\mathbf{Q} + \mathbf{r} \quad (\text{not linear in } \mathbf{Q})$$

- ▶ Because of the γ -contraction of the Bellman operator \mathcal{T} , one can easily show that the ODE is globally asymptotically stable.
- ▶ For **asynchronous** Q-learning, the corresponding ODE reduces to switched linear system, whose asymptotic stability can also be proven.
- ▶ Q-learning with **linear** function approximation is not guaranteed to converge, asymptotic stability of the corresponding ODE could be established **if** behavior policy is close to the optimal policy.

Summary: Valued-based RL Methods

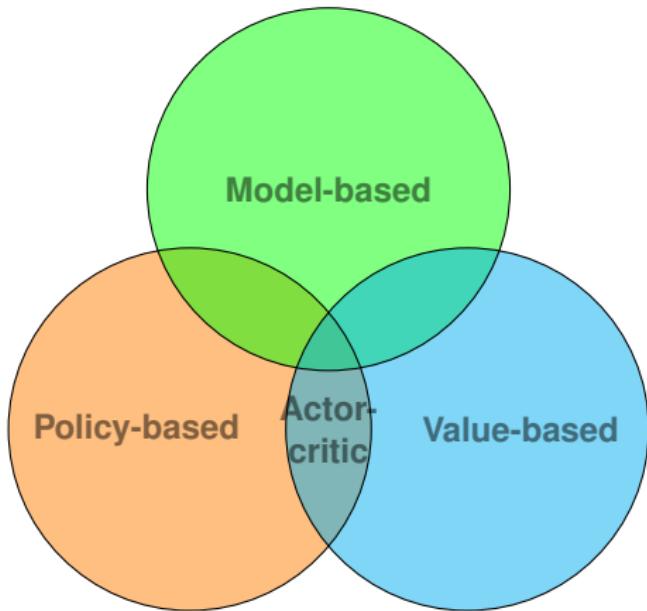
Model-free prediction and control: MC, TD, Q-learning, and many variants

- ▶ Tabular settings: good theory, but does not scale
- ▶ Linear function approximation: reasonably good theory, but requires good features
- ▶ Neural networks: less understood, but scale well

Sommaire

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
 - Overview of Policy-based RL
 - Policy gradient estimation
 - Policy gradient methods
 - Natural Policy Gradient Method (NPG)
 - Beyond PG/NPG
- 7 Deep Reinforcement Learning
- 8 Imitation learning
- 9 Going beyond

Recall Reinforcement Learning Approaches



- **Value-based RL**
 - ▶ First learn the optimal value functions V^*, Q^* (or best approximate V_η^*, Q_η^*)
 - ▶ Generate optimal policy
$$\pi^*(a | s) = \arg \max_{a \in \mathcal{A}} Q^*(s, a)$$
 - ▶ Example: Monte Carlo, SARSA, Q-learning, DQN, etc.
- **Policy-based RL**
 - ▶ Learn directly the optimal policy π^*
 - ▶ Example: Policy Gradient, NPG, TRPO, PPO, etc.
- **Model-based RL**
 - ▶ Learn both the model P, r and then do planning
 - ▶ Example: Dyna, UCRL2, UCB-VI, etc.

Value-based methods

Advantages

- ▶ Easy to generate policy from the *learned* value function
- ▶ Leverage bootstrap and N -step returns instead of full episodes
- ▶ Easy to control bias-variance tradeoff
- ▶ Good theory for tabular settings and linear function approximation settings

Value-based methods

Advantages

- ▶ Easy to generate policy from the *learned* value function
- ▶ Leverage bootstrap and N -step returns instead of full episodes
- ▶ Easy to control bias-variance tradeoff
- ▶ Good theory for tabular settings and linear function approximation settings

Disadvantages

- ▶ Policies are inherently deterministic.
Not always optimal (ex: rock, paper, cissor) (non-stationary MDPs, partially observable, ...).
Optimization is smoother and easier in stochastic env. (gradient vs combinatorial).
- ▶ Do not scale to high-dimensional or continuous action spaces
- ▶ Instability with off-policy learning under function approximation (deadly triad)
- ▶ Small value error may lead to large policy error

Policy-based methods

Sometimes easier to directly solve for π as opposed to V or Q .

Ex. LQR where π^* is linear.

Idea: represent policy as $\pi_\theta(\cdot | s)$ and find the best parameter θ that maximizes the long-term rewards

Policy Optimization (Episodic Reward)

$$\max_{\theta} J(\pi_{\theta}) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, \pi_{\theta} \right] = \mathbb{E}_{s \sim \mu} [V^{\pi_{\theta}}(s)]$$

- Here μ is the initial state distribution.

Policy-based methods

Sometimes easier to directly solve for π as opposed to V or Q .

Ex. LQR where π^* is linear.

Idea: represent policy as $\pi_\theta(\cdot | s)$ and find the best parameter θ that maximizes the long-term rewards

Policy Optimization (Episodic Reward)

$$\max_{\theta} J(\pi_{\theta}) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, \pi_{\theta} \right] = \mathbb{E}_{s \sim \mu} [V^{\pi_{\theta}}(s)]$$

- ▶ Here μ is the initial state distribution.
- ▶ In a continuous task (streaming, non-episodic), one may consider the average reward objective:

$$J_{\text{avg}}(\pi_{\theta}) = \mathbb{E}_{s \sim \mathcal{D}_{\mu}^{\pi_{\theta}}(s)} [V^{\pi_{\theta}}(s)] = \sum_s \mathcal{D}_{\mu}^{\pi_{\theta}}(s) V^{\pi_{\theta}}(s)$$

where $\mathcal{D}_{\mu}^{\pi}(s)$ is the state stationary distribution under policy π . More involved than episodic since θ is involved in both $V^{\pi_{\theta}}$ and $\mathcal{D}_{\mu}^{\pi_{\theta}}$.

Policy-based methods

Sometimes easier to directly solve for π as opposed to V or Q .

Ex. LQR where π^* is linear.

Idea: represent policy as $\pi_\theta(\cdot | s)$ and find the best parameter θ that maximizes the long-term rewards

Policy Optimization (Episodic Reward)

$$\max_{\theta} J(\pi_{\theta}) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, \pi_{\theta} \right] = \mathbb{E}_{s \sim \mu} [V^{\pi_{\theta}}(s)]$$

- ▶ Here μ is the initial state distribution.
- ▶ In a continuous task (streaming, non-episodic), one may consider the average reward objective:

$$J_{\text{avg}}(\pi_{\theta}) = \mathbb{E}_{s \sim \mathcal{D}_{\mu}^{\pi_{\theta}}(s)} [V^{\pi_{\theta}}(s)] = \sum_s \mathcal{D}_{\mu}^{\pi_{\theta}}(s) V^{\pi_{\theta}}(s)$$

where $\mathcal{D}_{\mu}^{\pi}(s)$ is the state stationary distribution under policy π . More involved than episodic since θ is involved in both $V^{\pi_{\theta}}$ and $\mathcal{D}_{\mu}^{\pi_{\theta}}$.

- ▶ **Stochastic policies:** $\pi_{\theta}(\cdot | s) = \mathcal{D}_{\pi}(\cdot | s; \theta)$ is a distribution over action space

How to parameterize policies? (Discrete Action)

Direct parameterization

$$\pi_{\theta}(a | s) = \theta_{s,a}$$

where $\theta_{s,a} \geq 0$ and $\sum_{a \in \mathcal{A}} \theta_{s,a} = 1$

→ simplex-based constrained optimization

Softmax policy

$$\pi_{\theta}(a | s) = \frac{\exp(\theta_{s,a})}{\sum_{a' \in \mathcal{A}} \exp(\theta_{s,a'})}$$

→ unconstrained

How to parameterize policies? (Discrete Action)

Direct parameterization

$$\pi_{\theta}(a | s) = \theta_{s,a}$$

where $\theta_{s,a} \geq 0$ and $\sum_{a \in \mathcal{A}} \theta_{s,a} = 1$

→ simplex-based constrained optimization

Function approximation

Log-linear policy

$$\pi_{\theta}(a | s) = \frac{\exp(\boldsymbol{\theta} \cdot \phi(s, a))}{\sum_{a' \in \mathcal{A}} \exp(\boldsymbol{\theta} \cdot \phi(s, a'))}$$

where $\phi(s, a) \in \mathbb{R}^d$ and $\boldsymbol{\theta} \in \mathbb{R}^d$

Softmax policy

$$\pi_{\theta}(a | s) = \frac{\exp(\theta_{s,a})}{\sum_{a' \in \mathcal{A}} \exp(\theta_{s,a'})}$$

→ unconstrained

(Neural) Softmax policy

$$\pi_{\theta}(a | s) = \frac{\exp(f_{\theta}(s, a))}{\sum_{a' \in \mathcal{A}} \exp(f_{\theta}(s, a'))}$$

where $f_{\theta}(s, a)$ represents a neural network

How to parameterize policies? (Continuous Action)

Continuous probability distributions: Gaussian, Beta, etc.

Gaussian parameterization

$$\pi_{\theta}(a | s) = \frac{1}{\sqrt{2\pi} \sigma_{\theta}(s)} \exp\left(-\frac{(a - \mu_{\theta}(s))^2}{2\sigma_{\theta}^2(s)}\right)$$

where $\mu_{\theta}(s)$ and $\sigma_{\theta}(s)$ are two differentiable function approximators

How to Optimize over the Policy Class?

Optimizing θ is usually a non-convex problem

Gradient-free methods

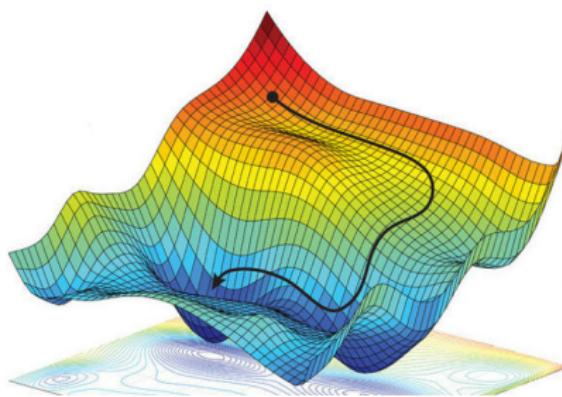
- ▶ Rely on small random perturbations
- ▶ Hill climbing
- ▶ Simulated annealing
- ▶ Evolutionary strategies
- ▶ Can be very slow but do not require π to be differentiable
- ▶ ...

How to Optimize over the Policy Class?

Optimizing θ is usually a non-convex problem

Gradient-free methods

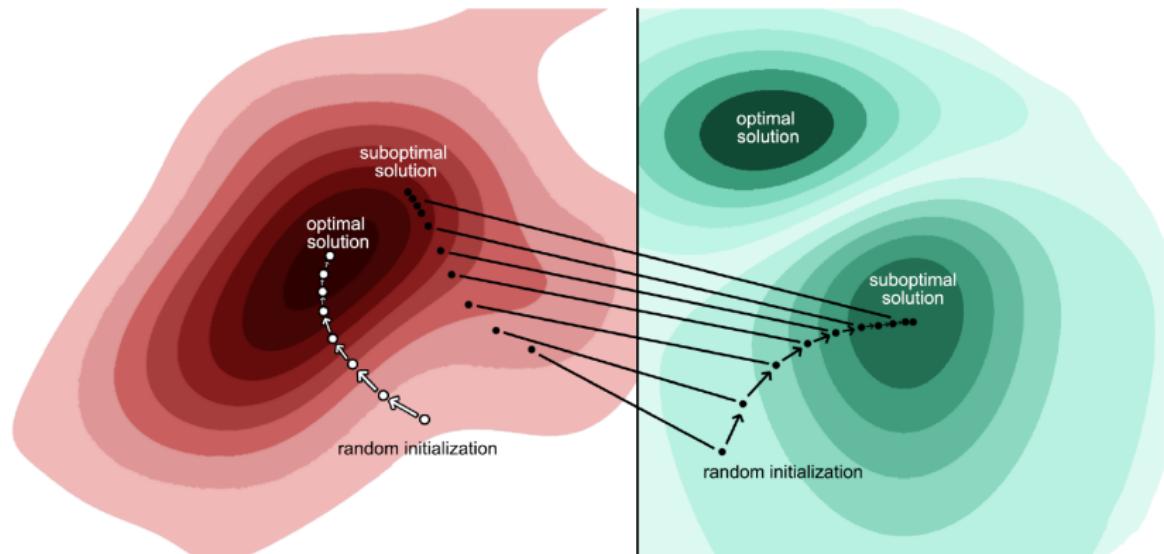
- ▶ Rely on small random perturbations
- ▶ Hill climbing
- ▶ Simulated annealing
- ▶ Evolutionary strategies
- ▶ Can be very slow but do not require π to be differentiable
- ▶ ...



Gradient-based methods

- ▶ Policy gradient method
- ▶ Natural policy gradient method
- ▶ ...

How to optimize over policy class?



Policy space vs. parameter space

Figure from Tessler et al., *Distributional Policy Optimization: An Alternative Approach for Continuous Control*, 2019.

Policy Gradient Method

- ▶ In general, we cannot exactly compute the gradient $\nabla_{\theta} J(\pi_{\theta})$ of the objective.
- ▶ A natural idea is to consider stochastic gradient:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \hat{\nabla}_{\theta} J(\pi_{\theta_t})$$

where $\hat{\nabla}_{\theta} J(\pi_{\theta_t})$ is some stochastic estimate of the gradient at θ_t .

Policy Gradient Method

- ▶ In general, we cannot exactly compute the gradient $\nabla_{\theta} J(\pi_{\theta})$ of the objective.
- ▶ A natural idea is to consider stochastic gradient:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \hat{\nabla}_{\theta} J(\pi_{\theta_t})$$

where $\hat{\nabla}_{\theta} J(\pi_{\theta_t})$ is some stochastic estimate of the gradient at θ_t .

Fundamental questions

- ▶ How to construct a good (unbiased and low variance for fast convergence) estimate of $\nabla_{\theta} J(\pi_{\theta})$?
- ▶ Where do PG methods converge to and how fast? (Stationary point? Local solution? Global optimal solution?)
- ▶ How to improve them?

MonteCarlo estimation of the gradient

Consider

$$F(\boldsymbol{\theta}) = \mathbb{E}_{\xi \sim p(\xi)} [f(\boldsymbol{\theta}, \xi)]$$

- ▶ Gradient:

$$\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \int f(\boldsymbol{\theta}, \xi) p(\xi) d\xi = \int \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \xi) p(\xi) d\xi = \mathbb{E}_{\xi \sim p(\xi)} [\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \xi)]$$

- ▶ Unbiased gradient estimators (empirical)

$$\hat{\nabla}_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \xi), \quad \text{where } \xi \sim p(\xi)$$

$$\hat{\nabla}_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \xi_i), \quad \text{where } \xi_1, \dots, \xi_n \sim p(\xi)$$

Monte Carlo Estimation with Score Function

Now consider

$$F(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\xi} \sim p_{\boldsymbol{\theta}}(\boldsymbol{\xi})} [f(\boldsymbol{\xi})]$$

Monte Carlo Estimation with Score Function

Now consider

$$F(\boldsymbol{\theta}) = \mathbb{E}_{\xi \sim p_{\boldsymbol{\theta}}(\xi)} [f(\xi)]$$

► Gradient:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) &= \int f(\xi) \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\xi) d\xi = \int p_{\boldsymbol{\theta}}(\xi) f(\xi) \underbrace{\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\xi)}_{\text{score function}} d\xi \\ &= \mathbb{E}_{\xi \sim p_{\boldsymbol{\theta}}(\xi)} [f(\xi) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\xi)]\end{aligned}$$

► Unbiased gradient estimators

$$\widehat{\nabla}_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = f(\xi) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\xi), \quad \text{where } \xi \sim p_{\boldsymbol{\theta}}(\xi)$$

$$\widehat{\nabla}_{\boldsymbol{\theta}} F(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n f(\xi_i) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\xi_i), \quad \text{where } \xi_1, \dots, \xi_n \sim p_{\boldsymbol{\theta}}(\xi)$$

Does not require information about the gradient of f

Relevance for Policy Optimization

Recall the episodic reward objective:

$$\max_{\theta} J(\pi_{\theta}) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, \pi_{\theta} \right] = \mathbb{E}_{h \sim p_{\theta}} [R(h)]$$

- ▶ $h = (s_0, a_0, s_1, \dots)$ is a random trajectory with probability $p_{\theta}(h)$:

$$p_{\theta}(h) := \mu(s_0) \prod_{t=0}^{\infty} \pi_{\theta}(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$$

- ▶ $R(h) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$ is the total reward over the random trajectory
- ▶ θ only appears in the distribution of the trajectory h
- ▶ We have $\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} [R(h) \cdot \nabla_{\theta} \log p_{\theta}(h)]$
- ▶ Note that $\nabla_{\theta} \log p_{\theta}(h) = \sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(a_t \mid s_t)$

Policy Gradient Theorem I.1: REINFORCE Expression

Combining these two results:

Policy Gradient Theorem (REINFORCE, Williams'92)

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[R(h) \sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ▶ The term $\nabla_{\theta} \log \pi_{\theta}(a | s) = \frac{\nabla_{\theta} \pi_{\theta}(a | s)}{\pi_{\theta}(a | s)}$ is called the **score function**
- ▶ For differentiable policies, the score function can often be easily computed
- ▶ For example, for log-linear policy $\pi_{\theta}(a | s) = \frac{\exp(\theta \cdot \phi(s, a))}{\sum_{a' \in \mathcal{A}} \exp(\theta \cdot \phi(s, a'))}$, we have
$$\nabla_{\theta} \log \pi_{\theta}(a | s) = \phi(s, a) - \mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [\phi(s, a)]$$
- ▶ Note that the expectation of the score function wrt π_{θ} is zero:
$$\mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [\nabla_{\theta} \log \pi_{\theta}(a | s)] = 0$$

REINFORCE Estimator

- ▶ Generate an episode $h = (s_0, a_0, r_0, s_1, \dots)$ from policy π_θ
 - ▶ Construct $\widehat{\nabla}_\theta J(\pi_\theta) = \left(\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right) \cdot \left(\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t, s_t) \right)$ [unbiased]
-
- ▶ A single trajectory under π_θ is enough to obtain an **unbiased** policy gradient estimator without knowing transition probability.

REINFORCE Estimator

- ▶ Generate an episode $h = (s_0, a_0, r_0, s_1, \dots)$ from policy π_θ
- ▶ Construct $\widehat{\nabla}_\theta J(\pi_\theta) = \left(\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right) \cdot \left(\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t, s_t) \right)$ [unbiased]

- ▶ A single trajectory under π_θ is enough to obtain an **unbiased** policy gradient estimator without knowing transition probability.
 - ▶ REINFORCE has a high variance due to correlation between $R(h)$ and $\{\pi_\theta(a_t | s_t)\}_{t=1}^\infty$
 - ▶ The variance scales with the horizon
 - ▶ Notice that $\pi_\theta(a_{t_2} | s_{t_2})$ does not affect $\sum_{t=0}^{t_1} r(s_t, a_t)$ if $t_2 > t_1$.
- Can one use this observation?

Policy Gradient Theorem I.2: Action Value Expression

The observations allow to simplify expressions:

Policy Gradient Theorem (Action-Value Function)

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t Q^{\pi_{\theta}}(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

For comparison,

- ▶ The REINFORCE expression is

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[\sum_{t=0}^{\infty} \sum_{t'=0}^{\infty} \gamma^{t'} r(s_{t'}, a_{t'}) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ▶ The Action-Value expression is

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[\sum_{t=0}^{\infty} \sum_{t'=t}^{\infty} \gamma^{t'} r(s_{t'}, a_{t'}) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

Less correlation than with $R(h)$ → lower variance of the estimator

Policy Gradient Estimator using Reward-to-Go

REINFORCE Estimator using Reward-to-Go

- ▶ Generate an episode $h = (s_0, a_0, r_0, s_1, \dots)$ from policy π_θ
 - ▶ Construct $\widehat{\nabla}_\theta J(\pi_\theta) = \sum_{t=0}^{\infty} \gamma^t G_t \cdot \nabla_\theta \log \pi_\theta(a_t, s_t)$ where $G_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$.
-
- ▶ Unbiased estimator of the policy gradient.
 - ▶ Can we further reduce the variance?

Policy Gradient Theorem I.3: Baseline Expression

Policy Gradient Theorem (Baseline)

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t (Q^{\pi_{\theta}}(s_t, a_t) - b(s_t)) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ▶ For any baseline $b(s)$ that does not depend on the actions:

$$\mathbb{E}_{a \sim \pi_{\theta}} [b(s) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] = \mathbf{0}$$

The idea is to inform how to move θ based on the performance (return) wrt a baseline (better sensitivity)

Policy Gradient Theorem I.3: Baseline Expression

Policy Gradient Theorem (Baseline)

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t (Q^{\pi_{\theta}}(s_t, a_t) - b(s_t)) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ▶ For any baseline $b(s)$ that does not depend on the actions:

$$\mathbb{E}_{a \sim \pi_{\theta}} [b(s) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] = \mathbf{0}$$

The idea is to inform how to move θ based on the performance (return) wrt a baseline (better sensitivity)

- ▶ A natural choice of baseline is the value function: $b(s) = V^{\pi_{\theta}}(s)$
- ▶ We call $Q^{\pi_{\theta}}(s, a) - V^{\pi_{\theta}}(s) := A^{\pi_{\theta}}(s, a)$ the **advantage function**.
 $V^{\pi_{\theta}}$ is positively very correlated with $Q^{\pi_{\theta}}$ → the variance of the estimator is low

Proof of Baseline Expression

Proof

Notice that $\sum_a \pi_{\theta}(a | s) = 1$ for any $s \in \mathcal{S}$. For any $b(s)$ that is independent of actions, we have:

$$\begin{aligned}\mathbb{E}_{a \sim \pi_{\theta}} [b(s) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] &= b(s) \sum_a \pi_{\theta}(a | s) \frac{\nabla_{\theta} \pi_{\theta}(a | s)}{\pi_{\theta}(a | s)} \\ &= b(s) \sum_a \nabla_{\theta} \pi_{\theta}(a | s) \\ &= b(s) \nabla_{\theta} \sum_a \pi_{\theta}(a | s) \\ &= b(s) \nabla_{\theta} 1 \\ &= \mathbf{0}\end{aligned}$$

Summary: Policy Gradient Theorem I

- ▶ REINFORCE expression based on the total reward of the trajectory:

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[R(h) \sum_{t=0}^{\infty} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ▶ Action value expression:

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t Q^{\pi_{\theta}}(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

- ▶ Baseline expression:

$$\nabla_{\theta} J(\pi_{\theta}) = \mathbb{E}_{h \sim p_{\theta}} \left[\sum_{t=0}^{\infty} \gamma^t (Q^{\pi_{\theta}}(s_t, a_t) - b(s_t)) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

Implies infinitely long trajectories

Policy Gradient Theorem II

Define the discounted state visitation distribution under policy π as

$$\mathcal{D}_\mu^\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid s_0 \sim \mu, \pi)$$

$(1 - \gamma)$ is such that \mathcal{D}_μ^π is a probability distribution

Policy Gradient Theorem II

Define the discounted state visitation distribution under policy π as

$$\mathcal{D}_\mu^\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid s_0 \sim \mu, \pi)$$

$(1 - \gamma)$ is such that \mathcal{D}_μ^π is a probability distribution

Policy Gradient Theorem II

- ▶ Action value expression:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \frac{1}{1 - \gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\boldsymbol{\theta}}} } \left[\mathbb{E}_{a \sim \pi_{\boldsymbol{\theta}}(\cdot \mid s)} [Q^{\pi_{\boldsymbol{\theta}}}(s, a) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)] \right]$$

No need for a whole trajectory.

One sample (s, a) allows to inform the gradient.

- ▶ Advantage expression:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \frac{1}{1 - \gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\boldsymbol{\theta}}} } \left[\mathbb{E}_{a \sim \pi_{\boldsymbol{\theta}}(\cdot \mid s)} [A^{\pi_{\boldsymbol{\theta}}}(s, a) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)] \right]$$

The proof follows immediately based on the definition of $\mathcal{D}_\mu^\pi(s)$.

Policy Gradient Methods (PG)

Exact Policy Gradient Method

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_t \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}_t})$$

Stochastic Policy Gradient Method

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha_t \widehat{\nabla}_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}_t})$$

- ▶ REINFORCE
- ▶ REINFORCE with baseline
- ▶ Actor-Critic
- ▶ ...

Monte Carlo Policy Gradient Method

We now know how to estimate policy gradients

Monte Carlo Policy Gradient Method

We now know how to estimate policy gradients

REINFORCE: Monte-Carlo Policy-Gradient Method

Initialize policy parameter $\theta \in \mathbb{R}^d$, step size $\alpha > 0$, baseline $b(\cdot)$

for each episode **do**

 Generate an episode $s_0, a_0, r_0, s_1, \dots, s_T, a_T, r_T$ following π_θ

for each step of the episode $t = 0, 1, \dots, T$ **do**

 Compute return $G_t \leftarrow \sum_{i=t}^T \gamma^{i-t} r_i$

 Compute advantage estimate $A_t \leftarrow G_t - b(s_t)$

$\theta \leftarrow \theta + \alpha \gamma^t A_t \nabla_\theta \log \pi_\theta(a_t | s_t)$

[SGD → unbiased]

end for

end for

- ▶ Will converge to stationary point with a slow rate
- ▶ The policy is updated only after generating a whole trajectory, which may not be efficient (high variance).
- ▶ Can utilize the idea of temporal difference learning to build policy gradient estimators, for instance for R_t and/or $b(s_t)$.

Policy Gradient Method with Value Function Estimation

Instead of MC to estimate Q , use TD learning

Online Actor-Critic Algorithm

Initialize $\theta_0, \eta_0, s_0 \sim \mu, a_0 \sim \pi_{\theta_0}(\cdot | s_0)$

for each step of the episode $t = 0, 1, \dots, T$ **do**

 Obtain (r_t, s_{t+1}, a_{t+1}) from π_{θ_t}

 Compute temporal difference $\delta_t = r_t + \gamma Q_{\eta_t}(s_{t+1}, a_{t+1}) - Q_{\eta_t}(s_t, a_t)$

 Compute policy gradient estimator:

$$\hat{\nabla}_{\theta} J(\pi_{\theta_t}) = Q_{\eta_t}(s_t, a_t) \nabla_{\theta} \log \pi_{\theta_t}(a_t | s_t)$$

 Update θ : $\theta_{t+1} \leftarrow \theta_t + \alpha \hat{\nabla}_{\theta} J(\pi_{\theta_t})$

 Update η : $\eta_{t+1} \leftarrow \eta_t - \beta \delta_t \nabla_{\eta} Q_{\eta_t}(s_t, a_t)$

[TD learning with function approx]

[Could be $m \geq 1$ or eligibility traces]

end for

- ▶ Much lower variance than MC gradient,
- ▶ Approximating the value function in policy gradient introduces extra bias since
 - ▶ function approx is not always accurate
 - ▶ (s, a) should \sim state-action distrib while here from the rollout from π_{θ}
- ▶ Various ways to estimate the advantage function.

Policy Gradient Methods

Advantages

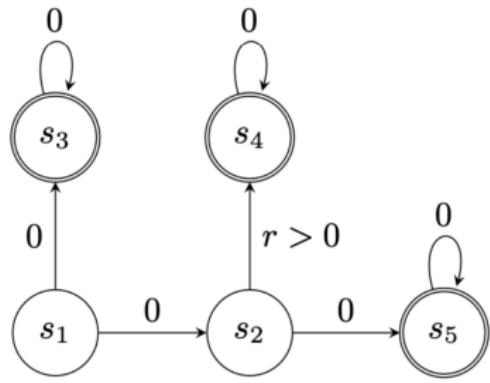
- ▶ Direct objective
- ▶ Can deal with high-dimensional and continuous action spaces
- ▶ Can learn stochastic policies

Optimization Challenges

- ▶ Nonconvex landscape (in general, only converges to stationary points, not necessarily extrema)
- ▶ Sensitive to stepsize choice, not always very stable
- ▶ High variance/bias of the policy gradient estimators

Optimization Challenge I: Nonconcavity

- ▶ In general, the objective $J(\pi_\theta)$ is nonconcave.
- ▶ This holds even for tabular setting with direct or softmax parametrization.



a_1 : move up

a_2 : move right

Example (direct parametrization)

$$V^\pi(s_1) = \pi(a_2 | s_1)\pi(a_1 | s_2)r$$

- ▶ Consider $\pi_{\text{mid}} = (\pi_1 + \pi_2)/2$, where

$$\pi_1(a_2 | s_1) = 3/4, \quad \pi_1(a_1 | s_2) = 3/4$$

$$\pi_2(a_2 | s_1) = 1/4, \quad \pi_2(a_1 | s_2) = 1/4$$

$$\pi_{\text{mid}}(a_2 | s_1) = 1/2, \quad \pi_{\text{mid}}(a_1 | s_2) = 1/2$$

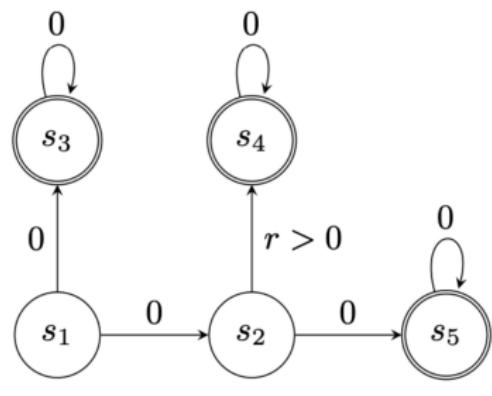
- ▶ $V^{\pi_1}(s_1) = \frac{9}{16}r, \quad V^{\pi_2}(s_1) = \frac{1}{16}r$

- ▶ $V^{\pi_{\text{mid}}}(s_1) = \frac{1}{4}r < \frac{1}{2}(V^{\pi_1}(s_1) + V^{\pi_2}(s_1))$

→ not a concave function

Optimization Challenge I: Nonconcavity

- ▶ In general, the objective $J(\pi_\theta)$ is nonconcave.
- ▶ This holds even for tabular setting with direct or softmax parametrization.



a_1 : move up
 a_2 : move right

Example (softmax parametrization)

$$\theta = (\theta_{a_1, s_1}, \theta_{a_2, s_1}, \theta_{a_1, s_2}, \theta_{a_2, s_2})$$

$$V^{\pi_\theta}(s_1) = \frac{e^{\theta_{a_2, s_1}}}{e^{\theta_{a_1, s_1}} + e^{\theta_{a_2, s_1}}} \frac{e^{\theta_{a_1, s_2}}}{e^{\theta_{a_1, s_2}} + e^{\theta_{a_2, s_2}}} r$$

- ▶ Consider

$$\theta_1 = (\log 1, \log 3, \log 3, \log 1)$$

$$\theta_2 = (-\log 1, -\log 3, -\log 3, -\log 1)$$

$$\theta_{\text{mid}} = (\theta_1 + \theta_2) / 2 = (0, 0, 0, 0)$$

- ▶ $V^{\pi_{\theta_{\text{mid}}}}(s_1) < \frac{1}{2} (V^{\pi_{\theta_1}}(s_1) + V^{\pi_{\theta_2}}(s_1))$

Convergence to stationary points

Convergence of exact policy gradient methods: $\theta_{t+1} = \theta_t + \alpha_t \nabla_{\theta} J(\pi_{\theta_t})$

If the objective $J(\pi_{\theta})$ is L -smooth and $\alpha_t = 1/L$ then

$$\min_{t=0, \dots, T-1} \|\nabla_{\theta} J(\pi_{\theta_t})\|_2^2 \leq \frac{2L(J(\pi_{\theta^*}) - J(\pi_{\theta_0}))}{T}$$

→ rate in $1/T$ with constant step size

Convergence of stochastic policy gradient methods: $\theta_{t+1} = \theta_t + \alpha_t \widehat{\nabla}_{\theta} J(\pi_{\theta_t})$

If the objective $J(\pi_{\theta})$ is L -smooth and $\widehat{\nabla}_{\theta} J(\pi_{\theta})$ is unbiased and has variance bounded by σ^2 then with proper step size:

$$\min_{t=0, \dots, T-1} \mathbb{E} \left[\left\| \widehat{\nabla}_{\theta} J(\pi_{\theta_t}) \right\|_2^2 \right] = \mathcal{O} \left(\sqrt{\frac{L(J(\pi_{\theta^*}) - J(\pi_{\theta_0})) \sigma^2}{T}} \right)$$

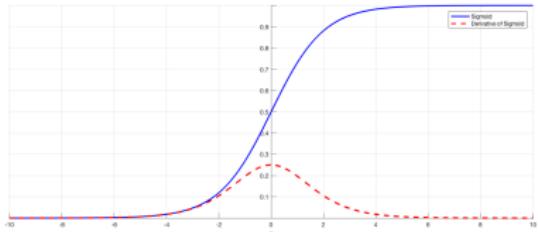
→ rate in $1/\sqrt{T}$

Optimization Challenge II: Vanishing Gradient and Saddle Points

- ▶ In general, there are no guarantees on the quality of stationary points.

Optimization Challenge II: Vanishing Gradient and Saddle Points

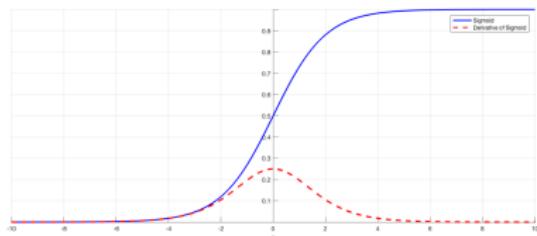
- ▶ In general, there are no guarantees on the quality of stationary points.
- ▶ Vanishing gradients can happen when using softmax parametrization.



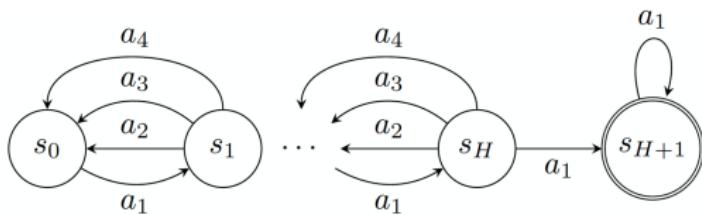
$$\text{Softmax function: } \frac{e^\theta}{1 + e^\theta} = \frac{1}{1 + e^{-\theta}}$$

Optimization Challenge II: Vanishing Gradient and Saddle Points

- In general, there are no guarantees on the quality of stationary points.
- Vanishing gradients can happen when using softmax parametrization.
- Vanishing gradients can happen when lacking sufficient exploration.



$$\text{Softmax function: } \frac{e^\theta}{1 + e^\theta} = \frac{1}{1 + e^{-\theta}}$$



Example with $H + 2$ states and $\gamma = H/(H + 1)$: rewards are 0 everywhere except at s_{H+1} . For small order p and θ such that $\theta_{s,a_1} < 1/4 \forall s$:

$$\|\nabla^p V^{\pi_\theta}(s_0)\| \leq (1/3)^{H/4}$$

→ gradient is exponentially small in terms of H

Nonconvex optimization aids

- ▶ Natural Policy Gradient
 - ▶ leverage good curvature information to avoid vanishing gradient
- ▶ Trust Region
 - ▶ alleviate step-size tuning
- ▶ Entropy Regularization
 - ▶ encourage exploration
- ▶ Using Second-Order Information
 - ▶ escape from saddle point, reducing variance
- ▶ ...

Natural Policy Gradient Method for Policy Optimization

$$\max_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, \pi_{\boldsymbol{\theta}} \right] = \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\boldsymbol{\theta}}}} [V^{\pi_{\boldsymbol{\theta}}}(s)]$$

Natural policy gradient (Kakade, 2002)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha F_{\boldsymbol{\theta}_t}^+ \nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}_t})$$

F^+ is the Moore-Penrose pseudo-inverse of F

- ▶ $\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}})$ is the policy gradient:

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\boldsymbol{\theta}}}, a \sim \pi_{\boldsymbol{\theta}}(\cdot \mid s)} [\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) A^{\pi_{\boldsymbol{\theta}}}(s, a)]$$

- ▶ $F_{\boldsymbol{\theta}}$ is the Fisher Information Matrix (FIM):

$$F_{\boldsymbol{\theta}} = \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\boldsymbol{\theta}}}, a \sim \pi_{\boldsymbol{\theta}}(\cdot \mid s)} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s)^T \right]$$

Interpretation of NPG

NPG can be viewed as repeatedly solving the quadratic approximation of the subproblem:

$$\boldsymbol{\theta}_{t+1} \approx \arg \max_{\boldsymbol{\theta}} [J(\pi_{\boldsymbol{\theta}})], \quad \text{s.t.} \quad \text{KL}(p_{\boldsymbol{\theta}_t}(h) \parallel p_{\boldsymbol{\theta}}(h)) \leq \delta$$

where $p_{\boldsymbol{\theta}}(h)$ is the probability distribution of the random trajectory $h = (s_0, a_0, r_1, \dots)$

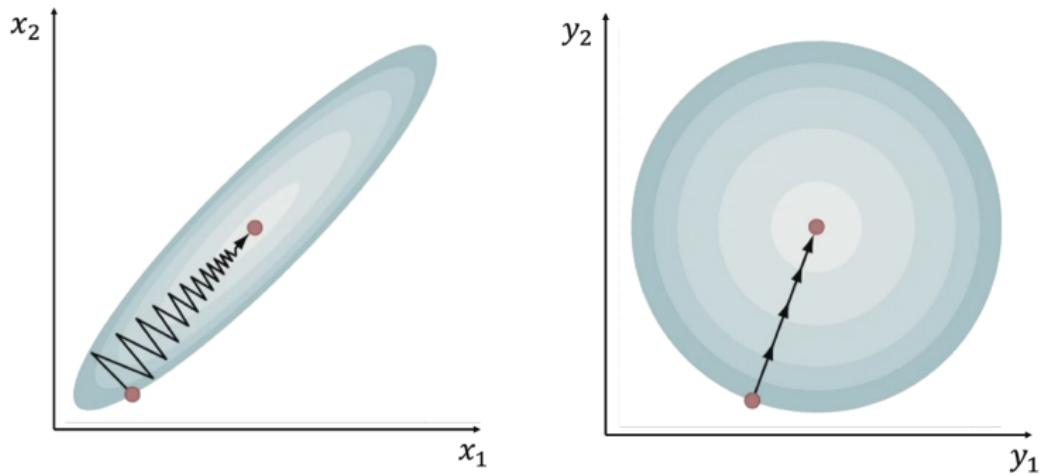
- ▶ Approximate the objective with first-order Taylor expansion: $\nabla J(\pi_{\boldsymbol{\theta}_t})^T (\boldsymbol{\theta} - \boldsymbol{\theta}_t)$
- ▶ Approximate the constraint with second-order Taylor expansion:

$$\text{KL}(p_{\boldsymbol{\theta}_t} \parallel p_{\boldsymbol{\theta}}) \approx \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^T F_{\boldsymbol{\theta}_t} (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \leq \delta$$

The Fisher Information Matrix $F_{\boldsymbol{\theta}_t}$ is the Hessian of the KL divergence:

$$F_{\boldsymbol{\theta}_t} = \nabla_{\boldsymbol{\theta}}^2 \text{KL}(p_{\boldsymbol{\theta}_t} \parallel p_{\boldsymbol{\theta}})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$$

Revisit Gradient Descent



Computing Natural Policy Gradient

Equivalent form of NPG

$$(1 - \gamma) F_{\theta}^+ \nabla_{\theta} J(\pi_{\theta}) = w^*(\theta)$$

where $w^*(\theta)$ is the solution to the least square problem:

$$w^*(\theta) \in \arg \min_w \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_\theta}, a \sim \pi_\theta(\cdot | s)} \left[\left(w^\top \nabla_\theta \log \pi_\theta(a | s) - A^{\pi_\theta}(s, a) \right)^2 \right]$$

- ▶ Follows immediately by first order optimality condition.
- ▶ Equivalently, can rewrite NPG as:

$$\theta_{t+1} = \theta_t + \frac{\alpha}{1 - \gamma} w^*(\theta_t)$$

- ▶ Can obtain $w^*(\theta_t)$ by solving the equivalent form with any least squares solver, e.g., conjugate gradient or SGD.

NPG under softmax parameterization

Consider $\pi_{\theta}(a | s) = \frac{\exp(\theta_{s,a})}{\sum_{a'} \exp(\theta_{s,a'})}$

NPG parameter update

$$\theta_{t+1} = \theta_t + \frac{\alpha}{1 - \gamma} A^{\pi_{\theta_t}}$$

- ▶ Simple expression
- ▶ Computationally easier since does not require \mathcal{D}_{μ}^{π}

NPG under softmax parameterization

Consider $\pi_{\theta}(a | s) = \frac{\exp(\theta_{s,a})}{\sum_{a'} \exp(\theta_{s,a'})}$

NPG parameter update

$$\theta_{t+1} = \theta_t + \frac{\alpha}{1 - \gamma} A^{\pi_{\theta_t}}$$

- ▶ Simple expression
- ▶ Computationally easier since does not require \mathcal{D}_{μ}^{π}

Convergence

For any $\alpha \geq (1 - \gamma)^2 \log |\mathcal{A}|$ and $T > 0$,

$$J(\pi^*) - J(\pi_{\theta_T}) \leq \frac{2}{(1 - \gamma)^2 T}$$

- ▶ Dimension-free convergence, no dependence on $|\mathcal{A}|, |\mathcal{S}|$
- ▶ Faster convergence than policy gradient

NPG with linear function approximation

Consider $\pi_{\theta}(a | s) = \frac{\exp(\boldsymbol{\theta}^T \phi(s, a))}{\sum_{a'} \exp(\boldsymbol{\theta}^T \phi(s, a'))}$

NPG parameter update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \frac{\alpha}{1 - \gamma} \mathbf{w}_t$$

$$\mathbf{w}_t \in \arg \min_{\mathbf{w}} \mathbb{E}_{\substack{s \sim \mathcal{D}_{\mu}^{\pi_{\theta}} \\ a \sim \pi_{\theta}(\cdot | s)}} \left[\left(\mathbf{w}^T \phi(s, a) - A^{\pi_{\theta}}(s, a) \right)^2 \right]$$

NPG policy update

$$\pi_{\theta_{t+1}}(s, a) = \pi_{\theta_t}(s, a) \frac{\exp(\alpha \mathbf{w}_t^T \phi(s, a) / (1 - \gamma))}{Z_t(s)}$$

Sample-based NPG

Sample-based NPG

Initialize policy parameter $\theta_0 \in \mathbb{R}^d$, step size $\alpha > 0$, $\beta > 0$

for $t = 0, 1, \dots, T - 1$ **do**

 Initialize w_0 , denote $\pi_t = \pi_{\theta_t}$

for $n = 0, 1, \dots, N - 1$ **do**

 Obtain sample $s \sim \mathcal{D}_\mu^{\pi_t}$, $a \sim \pi_t(\cdot | s)$

 Obtain an unbiased estimate $\hat{A}(s, a)$ for $A^{\pi_t}(s, a)$

(e.g., rollout π and MC return)

 Update w :

$$w = w - \beta \left(w^\top \nabla_{\theta} \log \pi_t(a | s) - \hat{A}(s, a) \right) \cdot \nabla_{\theta} \log \pi_t(a | s)$$

end for

 Set $w_t = w$ (or the average)

 Update $\theta_{t+1} = \theta_t + \frac{\alpha}{1 - \gamma} w_t$

end for

Policy gradient: from theory to practice

What key differences from using SGD for supervised / deep learning?

Policy gradient: from theory to practice

What key differences from using SGD for supervised / deep learning?

- ▶ In RL, for different policies, one generates different data. Policies and data change over time/episode

Policy gradient: from theory to practice

What key differences from using SGD for supervised / deep learning?

- ▶ In RL, for different policies, one generates different data. Policies and data change over time/episode
- ▶ Supervised learning: somehow robust to stepsize \neq RL where
 - ▶ too large stepsize \longrightarrow poor policy \longrightarrow poor data
 - ▶ too small stepsize \longrightarrow policy (and data) too similar to previous one \longrightarrow poor exploration [line search, trust region, clipping]

Policy gradient: from theory to practice

What key differences from using SGD for supervised / deep learning?

- ▶ In RL, for different policies, one generates different data. Policies and data change over time/episode
- ▶ Supervised learning: somehow robust to stepsize \neq RL where
 - ▶ too large stepsize \longrightarrow poor policy \longrightarrow poor data
 - ▶ too small stepsize \longrightarrow policy (and data) too similar to previous one \longrightarrow poor exploration [line search, trust region, clipping]
- ▶ Sample inefficiency: cannot use data collected with previous policies [importance sampling, experience replay]

Policy gradient: from theory to practice

What key differences from using SGD for supervised / deep learning?

- ▶ In RL, for different policies, one generates different data. Policies and data change over time/episode
- ▶ Supervised learning: somehow robust to stepsize \neq RL where
 - ▶ too large stepsize \longrightarrow poor policy \longrightarrow poor data
 - ▶ too small stepsize \longrightarrow policy (and data) too similar to previous one \longrightarrow poor exploration [line search, trust region, clipping]
- ▶ Sample inefficiency: cannot use data collected with previous policies [importance sampling, experience replay]
- ▶ High variance: stochastic policy gradient estimator suffers from high variance since variance accumulates during an episode [actor-critic]

Trust Region Policy Optimization

John Schulman

Sergey Levine

Philipp Moritz

Michael Jordan

Pieter Abbeel

JOSCHU@EECS.BERKELEY.EDU

SLEVINE@EECS.BERKELEY.EDU

PCMORITZ@EECS.BERKELEY.EDU

JORDAN@CS.BERKELEY.EDU

PABBEEL@CS.BERKELEY.EDU

University of California, Berkeley, Department of Electrical Engineering and Computer Sciences

TRPO (ICML, 2015)

Proximal Policy Optimization Algorithms

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI

{joschu, filip, prafulla, alec, oleg}@openai.com

PPO (arXiv, 2017)

OpenAI implementation: <https://github.com/openai/baselines>

Trust Region Policy Optimization (TRPO)²

TRPO

$$\begin{aligned} \max_{\theta} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\theta_t}}, a \sim \pi_{\theta_t}(\cdot | s)} & \left[\frac{\pi_\theta(a | s)}{\pi_{\theta_t}(a | s)} A^{\pi_{\theta_t}}(s, a) \right] \\ \text{s.t. } \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\theta}}} [\text{KL}(\pi_\theta(\cdot | s) \| \pi_{\theta_t}(\cdot | s))] & \leq \delta \end{aligned}$$

- ▶ Explicit KL divergence in the constraint
- ▶ Allows to use samples from π_{θ_t} instead of π (unknown)
- ▶ The surrogate objective can be viewed as linear approximation (in π instead of θ) of $J(\pi_\theta)$:

$$J(\pi) = J(\pi_t) + \frac{1}{1 - \gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^\pi, a \sim \pi(\cdot | s)} [A^{\pi_t}(s, a)] \quad [\text{Performance Difference Lemma}]$$

- ▶ Difficulties: objective is nonlinear in θ and it is difficult to deal with KL divergence
→ approximated in practice by NPG
- ▶ Use line-search to ensure performance improvement and no constraint violation.
- ▶ Extension to trust regions based on other divergences, e.g., Wasserstein distance, Sinkhorn divergence, etc.

²J. Schulman, S. Levine, P. Abbeel, M. Jordan & P. Moritz, Trust region policy optimization, PMLR, 2015.

Proximal Policy Optimization (PPO) (2nd version)

Tries to avoid the constraint on the KL divergence

Proximal Policy Optimization (PPO) (2nd version)

Tries to avoid the constraint on the KL divergence

Key idea

$$\max_{\theta} \mathbb{E}_{\substack{s \sim \mathcal{D}_\mu^{\pi_\theta} \\ a \sim \pi_{\theta_t}(\cdot | s)}} \min \left\{ \frac{\pi_\theta(a | s)}{\pi_{\theta_t}(a | s)} A^{\pi_{\theta_t}}(s, a), \text{clip} \left(\frac{\pi_\theta(a | s)}{\pi_{\theta_t}(a | s)}, 1 - \epsilon, 1 + \epsilon \right) A^{\pi_{\theta_t}}(s, a) \right\}$$

- ▶ PPO penalizes large deviation from the current policy directly inside the objective function through clipping the ratio $\frac{\pi_\theta}{\pi_{\theta_t}}$

$$\text{clip}(x, 1 - \epsilon, 1 + \epsilon) = \begin{cases} 1 - \epsilon, & \text{if } x < 1 - \epsilon \\ 1 + \epsilon, & \text{if } x > 1 + \epsilon \\ x, & \text{otherwise} \end{cases}$$

- ▶ Run SGD. No need to deal with the KL divergence and trust region constraint.

No theoretical guarantees

Applications using PPO



Robots



Locomotion



Muti-agent Games

- ▶ Example 1
- ▶ Example 2
- ▶ Building block of ChatGPT

From theory to practice



Theory



Practice

Illustration from J. Schulman

Variance-reduced PG Methods

- ▶ In general, stochastic policy gradient methods require $\mathcal{O}(\epsilon^{-4})$ trajectories to achieve an ϵ -approximate first-order stationary point.
- ▶ Adapting variance reduction techniques in nonconvex optimization, one can achieve an improved $\mathcal{O}(\epsilon^{-3})$ sample complexity for general policy parametrization.
- ▶ Key idea: consider the optimization problem $\max_{\theta \in \mathbb{R}^d} \mathbb{E}_{z \sim p(z)} [f(\theta, z)]$:

Common framework of Variance Reduction

- ▶ compute recursive gradient estimator:

$$h_t = \begin{cases} \frac{1}{|\mathcal{B}_{\text{check}}|} \sum_{z \in \mathcal{B}_{\text{check}}} \nabla f(\theta_t, z), & \text{if } t \equiv 0 \pmod{Q}, \\ h_{t-1} + \frac{1}{|\mathcal{B}|} \sum_{z \in \mathcal{B}} (\nabla f(\theta_t, z) - \nabla f(\theta_{t-1}, z)), & \text{otherwise.} \end{cases}$$

- ▶ $\theta_{t+1} \leftarrow \theta_t + \alpha h_t$

Variance-reduced policy gradient methods

Method	SC	$ \mathcal{B} $	$ \mathcal{B}_{check} $	Checkpoint	IS
SVRPG (Xu, Gao, and Gu 2020)	$O(\frac{1}{\epsilon^{10/3}})$	$O(\frac{1}{\epsilon^{4/3}})$	$O(\frac{1}{\epsilon^{4/3}})$	Needed	Needed
HAPG (Shen et al. 2019)	$O(\frac{1}{\epsilon^3})$	$O(\frac{1}{\epsilon})$	$O(\frac{1}{\epsilon^2})$	Needed	Not needed
VRMPO (Yang et al. 2019)	$O(\frac{1}{\epsilon^3})$	$O(\frac{1}{\epsilon})$	$O(\frac{1}{\epsilon^2})$	Needed	Not needed
SRVR-PG (Xu, Gao, and Gu 2019)	$O(\frac{1}{\epsilon^3})$	$O(\frac{1}{\sqrt{\epsilon}})$	$O(\frac{1}{\epsilon})$	Needed	Needed
HSPGA (Pham et al. 2020)	$O(\frac{1}{\epsilon^3})$	$O(1)$	-	Not needed	Needed
MBPG (Huang et al. 2020)	$\tilde{O}(\frac{1}{\epsilon^3})$	$O(1)$	-	Not needed	Needed
PAGE-PG (Gargiani et al. 2022)	$O(\frac{1}{\epsilon^3})$	$O(1)$	$O(\frac{1}{\epsilon^2})$	Needed	Needed
SHARP (Masiha et al. 2022)	$O(\frac{1}{\epsilon^3})$	$O(1)$	-	Not needed	Not needed

IS: Importance Sampling

Policy Gradient theorems (REINFORCE, action-value expression, baseline)

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{h \sim p_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{\infty} \gamma^t (Q^{\pi_{\boldsymbol{\theta}}}(s_t, a_t) - b(s_t)) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t) \right]$$
$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim \mathcal{D}_{\mu}^{\pi_{\boldsymbol{\theta}}} } \left[\mathbb{E}_{a \sim \pi_{\boldsymbol{\theta}}(\cdot | s)} [Q^{\pi_{\boldsymbol{\theta}}}(s, a) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a | s)] \right]$$

- ▶ Reduce the variance of stochastic gradient estimators by using a baseline
- ▶ Policy Gradient Methods: “on-policy” nature, “slow” convergence to stationary points
- ▶ Advanced optimization techniques for more efficient policy optimization in practice

Sample efficiency: Global Sample Complexity for Stochastic Policy Gradient Methods

When combined with stochastic policy gradients,

- ▶ Convergence to the globally optimal policy (up to approximation error) can be guaranteed for
 - ▶ Tabular setting with direct/softmax parametrization
 - ▶ log-linear policies (discrete action space)
 - ▶ Certain class of smooth parametrization including Gaussian policies (continuous action space)
- ▶ Large mini-batch (*e.g.*, $\mathcal{O}(\epsilon^{-1})$, $\mathcal{O}(\epsilon^{-2})$) per iteration for stochastic gradient updates
- ▶ Total $\mathcal{O}(\epsilon^{-3})$ sample complexity for batch-free stochastic update
- ▶ Best-known $\mathcal{O}(\epsilon^{-2})$ sample complexity with variance reduction

Sommaire

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
 - Actor-Critic Methods
 - Deep Reinforcement Learning
 - Valued-based Deep RL
 - Policy-based/Actor-Critic Deep RL
- 8 Imitation learning
- 9 Going beyond

Recap: overview of Reinforcement Learning approaches

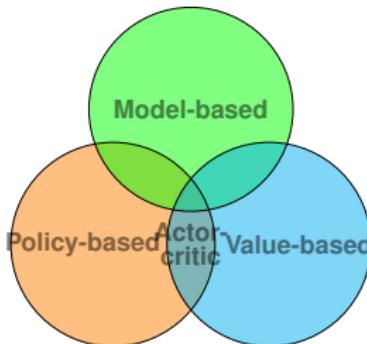
Value-based RL (Critic-only)

- ▶ Learn the optimal value functions V^* , Q^*
- ▶ Algorithms: Monte Carlo, SARSA, Q-learning, etc.

+ : Use temporal difference, have low variance

+ : Good theoretical guarantees

- : Do not scale to large action spaces



Policy-based RL (Actor-only)

- ▶ Learn the optimal policy via gradient methods
- ▶ Algorithms: PG, NPG, TRPO, PPO, etc.

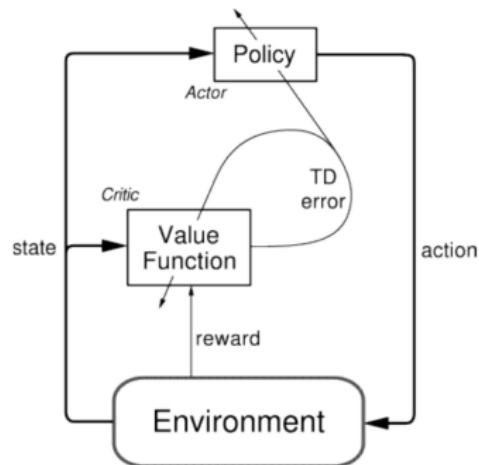
+ : Scale to large or continuous action spaces

- : High variance, sample inefficiency

- : Typically, cannot reuse previous data once the policy is updated

Actor-Critic (AC) Methods

AC methods aim at combining the advantages of actor-only methods and critic-only methods.



- ▶ The actor uses the policy gradient to update the learning policy.
- ▶ The critic uses temporal difference learning to estimate the value function.

Interaction of Actor-Critic^a

^aRichard S Sutton and Andrew G Barto.
Reinforcement learning: An introduction.
MIT press, 2018.

Actor-Critic methods

- ▶ Actor-critic algorithms follow an approximate policy gradient:

$$\nabla_{\theta} J(\pi_{\theta}) \approx \frac{1}{1-\gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\theta}}} [\mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [Q_{\eta}(s, a) \nabla_{\theta} \log \pi_{\theta}(a | s)]]$$

$$\nabla_{\theta} J(\pi_{\theta}) \approx \frac{1}{1-\gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\theta}}} [\mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [A_{\eta}(s, a) \nabla_{\theta} \log \pi_{\theta}(a | s)]]$$

- ▶ **Actor**: adjust the policy parameter θ using policy gradient based on low-variance value function estimator updated by the critic (instead of high-variance rollout or MC method)
- ▶ **Critic**: update the parameter η to estimate action-value or advantage function.

$$Q_{\eta}(s, a) \approx Q^{\pi_{\theta}}(s, a)$$

$$A_{\eta}(s, a) \approx Q^{\pi_{\theta}}(s, a) - V^{\pi_{\theta}}(s)$$

- using the critic allows to use value approximation
- allows the use of off-policy techniques (more sample efficient)

Bias in Actor-Critic Methods

- ▶ Recall action value expression of policy gradient

$$\nabla_{\theta} J(\pi_{\theta}) \approx \frac{1}{1-\gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\theta}}} \left[\mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [Q^{\pi_{\theta}}(s, a) \nabla_{\theta} \log \pi_{\theta}(a | s)] \right]$$

- ▶ Policy gradient estimators used by actor-critic algorithms:

$$\widehat{\nabla}_{\theta} J(\pi_{\theta}) \approx \frac{1}{1-\gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_{\theta}}} \left[\mathbb{E}_{a \sim \pi_{\theta}(\cdot | s)} [Q_{\eta}(s, a) \nabla_{\theta} \log \pi_{\theta}(a | s)] \right]$$

- ▶ Approximating the policy gradient using value function approximation Q_{η} could introduce bias → poor new policy → poor new data → possible divergence
- ▶ Luckily, if the value function approximation Q_{η} is chosen carefully, one may avoid such bias.

Compatible Function Approximation Theorem³

Compatible Function Approximation Theorem

Suppose the following two conditions are satisfied:

- ① Value function approximation at η^* is compatible to the policy, i.e.,

$$\nabla_{\eta} Q_{\eta^*}(s, a) = \nabla_{\theta} \log \pi_{\theta}(a | s)$$

- ② Value function parameter η^* minimizes the mean-squared error, i.e.,

$$\eta^* \in \arg \min_{\eta} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_\theta}, a \sim \pi_\theta(\cdot | s)} [(Q_\eta(s, a) - Q^{\pi_\theta}(s, a))^2]$$

Then the policy gradient using critic $Q_{\eta^*}(s, a)$ is exact:

$$\nabla_{\theta} J(\pi_{\theta}) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_\theta}, a \sim \pi_\theta(\cdot | s)} [\nabla_{\theta} \log \pi_{\theta}(a | s) Q_{\eta^*}(s, a)]$$

- ▶ Proof follows immediately from first-order optimality condition:

$$\nabla_{\eta} = \mathbf{0} \longrightarrow Q_{\eta^*} \log \pi_{\theta} = Q^{\pi} \log \pi$$

- ▶ Example: $Q_{\eta}(s, a) = \nabla_{\theta} \log \pi_{\theta}(a | s)^T \eta$

³R. Sutton, D. McAllester, S. Singh & Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In Advances in neural information processing systems, 2000.

Online Action-Value Actor-Critic

Online Action-Value Actor-Critic Algorithm

Initialize $\theta_0, \eta_0, s_0 \sim \mu, a_0 \sim \pi_{\theta_0}(\cdot | s_0)$

for each step of the episode $t = 0, 1, \dots, T$ **do**

 Obtain (r_t, s_{t+1}, a_{t+1}) from π_{θ_t} .

 Compute policy gradient estimator: $\hat{\nabla}_{\theta} J(\pi_{\theta_t}) = Q_{\eta_t}(s_t, a_t) \nabla_{\theta} \log \pi_{\theta_t}(a_t | s_t)$

 Actor update: $\theta_{t+1} = \theta_t + \alpha_t \hat{\nabla}_{\theta} J(\pi_{\theta_t})$ [one step of policy-gradient]

 Compute temporal difference: $\delta_t = r_t + \gamma Q_{\eta_t}(s_{t+1}, a_{t+1}) - Q_{\eta_t}(s_t, a_t)$

 Critic update: $\eta_{t+1} = \eta_t - \beta_t \delta_t \nabla_{\eta} Q_{\eta_t}(s_t, a_t)$ [one step of SARSA]

end for

- ▶ Uses temporal difference to estimate the value function $Q^{\pi_{\theta}}$.
- ▶ Examples for Q_{η} : linear value function approximation $Q_{\eta}(s, a) = \phi(s, a)^T \eta$
- ▶ Different step sizes: $\alpha_t \neq \beta_t$ (two time-scale algorithm)
- ▶ Could also do multiple critic updates for 1 actor update
- ▶ Can also use A instead of Q

Various Actor-Critic Extensions

- ▶ **Natural Actor-Critic:** use TRPO or NPG to update the actor π_θ
- ▶ **Soft Actor-Critic:** use entropy regularization in the objective to improve exploration

$$\max_{\pi} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) + \lambda \mathcal{H}(\pi(\cdot | s_t)) \right]$$

where $\mathcal{H}(\pi(\cdot | s)) = \mathbb{E}_{a \sim \pi(\cdot | s)} [-\log \pi(a | s)]$

- ▶ If $\lambda \rightarrow \infty \implies \pi \rightarrow$ uniform distribution
- ▶ Also eases optimization since smoother/more convex problem

Convergence of Actor-Critic Methods

- ▶ Usually asymptotic results, e.g., $\lim_{t \rightarrow \infty} \frac{\alpha_t}{\beta_t} = 0$

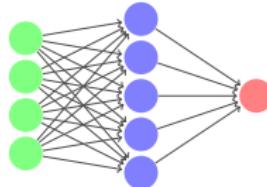
Actor-critic algorithms are closely related to gradient methods for bilevel optimization:

$$\min_{\theta} F(\theta) = f(\theta, \eta^*(\theta)), \quad [\text{upper level: actor}]$$

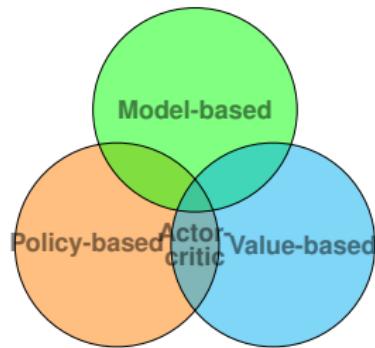
$$\text{s.t. } \eta^*(\theta) \in \arg \min_{\eta} \ell(\theta, \eta), \quad [\text{lower level: critic}]$$

Deep Reinforcement Learning = DL + RL

- ▶ Tabular methods and linear function approximation are insufficient for large-scale RL applications.
- ▶ Using neural networks seems useful.

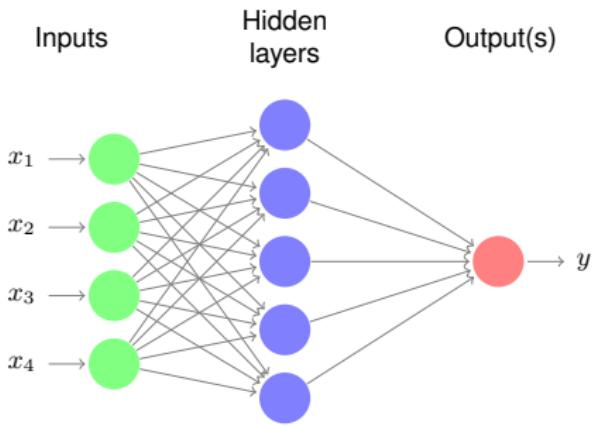
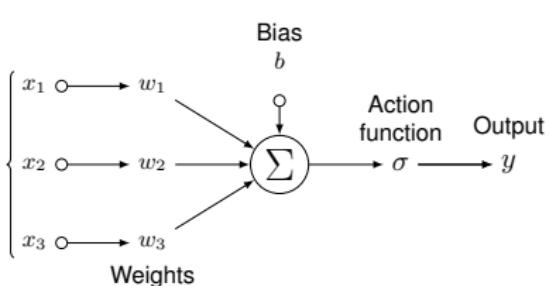


+



Neural network

1-layer network: $x \rightarrow w^T x + b \rightarrow \sigma(w^T x + b) = h(x; w) = y.$



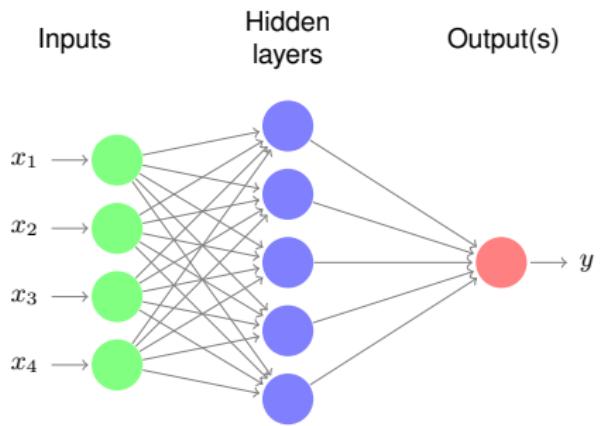
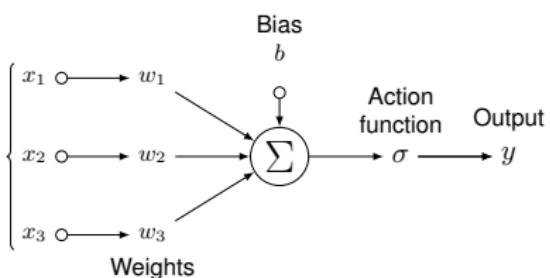
Neural network

1-layer network: $x \rightarrow w^T x + b \rightarrow \sigma(w^T x + b) = h(x; w) = y.$

L -layer network:

$$y = h_L(h_{L-1}(h_{L-2}(\dots; w_{L-2}); w_{L-1}); w_L).$$

Composition of several layers. High flexibility in the choice of the hyperparameters (n_l , σ_l , connectivity, etc.). Many unknowns to train \rightarrow requires a large amount of data.



Activation functions

$\sigma : \mathbb{R} \rightarrow \mathbb{R}$ must be non-linear to go beyond a mere matrix application $y = A \mathbf{x}$.

Name	σ
Heaviside	$H(x)$
Rectified Linear Unit (ReLU)	$\max[0, x]$
Leaky ReLU	$\begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$
Classical choices	
Logistic (sigmoid)	$\frac{1}{1 + \exp(-x)}$
Tanh	$\tanh(x)$
Swish	$\frac{x}{1 + \exp(-\beta x)}$
Softmax	$\frac{\exp[W \mathbf{x}]_k}{\sum_{k'=1}^K \exp[W \mathbf{x}]_{k'}}$

Why Neural Networks?

Universal Approximation

- ▶ Any continuous function on a compact domain can be (uniformly) approximated to arbitrary accuracy by a single-hidden layer neural network with a non-polynomial activation function. [Cybenko, 1989; Hornik et al., 1989; Barron, 1993]
- ▶ But the number of neurons can be large.

Benefits of Depth

- ▶ A deep network cannot be approximated by a reasonably-sized shallow network.
- ▶ For example, there exists a function with $\mathcal{O}(L^2)$ layers and width 2 which requires width $\mathcal{O}(2^L)$ to approximate with $\mathcal{O}(L)$ layers. [Telgarsky 2015]

Challenges with Training Neural Networks in RL

- ▶ Deadly triad (divergence when combining function approximation, bootstrapping, and off-policy learning) since non *iid* data and moving target (value function)
- ▶ Sample inefficiency. In particular, no reuse of data in off-policy PG methods
- ▶ High variance
- ▶ Overfitting
- ▶ Saddle points
- ▶ ...

Common Fixes or RL Tricks

- ▶ **Better data:** e.g., experience replay (mix online data and a buffer from past experience)
 - ▶ Reduce correlation, allow mini-batch update and lowers the variance
- ▶ **Better objective:** e.g., use entropy regularization
 - ▶ Improve optimization landscape, encourage exploration
- ▶ **Better optimizers:** e.g., adaptive SGD such as Adam and RMSProp
 - ▶ Adaptive learning rates
- ▶ **Better estimation:** e.g., use eligibility traces, target networks
 - ▶ Reduce overestimation bias, balance bias-variance tradeoff
- ▶ **Better sampling:** e.g., use prioritized replay (sample based on priority)
 - ▶ Prioritize transitions on which we can learn much; e.g. priority $\propto |\delta_t|$
- ▶ **Better implementation:** e.g., parallel implementation (multithreading of CPU and multi-agent)
 - ▶ Speed up training, reduce correlation (since each agent generates its own data), allow better exploration
- ▶ **Better architectures:** e.g., dueling networks
 - ▶ Encode inductive biases that are good for RL

Value-based DRL

- ▶ Idea: use neural networks for value function approximation
- ▶ Recall Q-learning:

Q-learning with function approximation

$$\eta_{t+1} \leftarrow \eta_t + \alpha_t \left[r_t + \gamma \max_a Q_{\eta_t}(s_{t+1}, a) - Q_{\eta_t}(s_t, a_t) \right] \nabla Q_{\eta_t}(s_t, a_t)$$

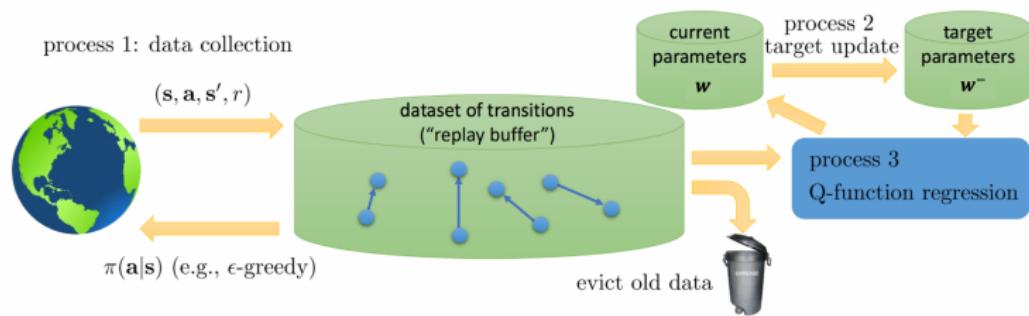
- ▶ Note that Q-learning is not a stochastic gradient descent method.
- ▶ Naive deep Q-learning could diverge due to sample correlation and moving targets (TD term depends on η_t).
- ▶ Deep Q-Networks (DeepMind, 2015): combine several techniques for stabilizing Q-learning
 - ▶ Experience replay (better data efficiency and make data more stationary)
 - ▶ Target networks (prevent target objective from changing too fast)

Deep Q-Networks (DQN)

- Main idea: minimize the following mean-square error by SGD (or adaptive SGD)

$$\min_{\boldsymbol{\eta}} \ell(\boldsymbol{\eta}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\eta}^-) - Q(s, a; \boldsymbol{\eta}) \right)^2 \right]$$

- Experience replay: expectation is on the replay buffer
- The target parameter $\boldsymbol{\eta}^-$ is held fixed and updated periodically



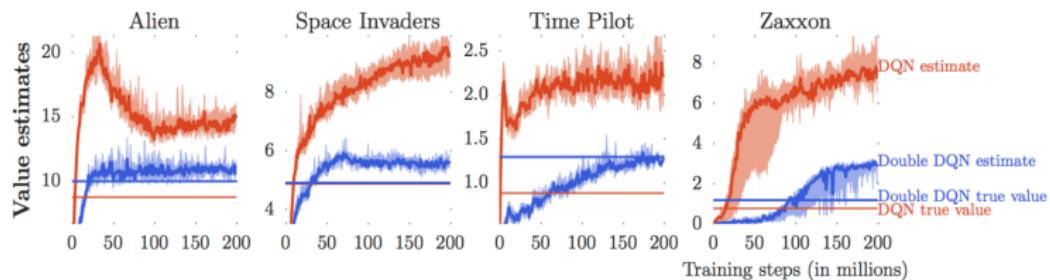
A general view of DQN

Source code: <https://github.com/deepmind/dqn>

DQN Extensions I

Double DQN (DeepMind, 2016): Use separate networks to select best action and evaluate best action to reduce overestimation bias

$$\min_{\boldsymbol{\eta}} \ell(\boldsymbol{\eta}) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma Q \left(s', \arg \max_{a'} Q(s', a'; \boldsymbol{\eta}); \boldsymbol{\eta}^- \right) - Q(s, a; \boldsymbol{\eta}) \right)^2 \right]$$



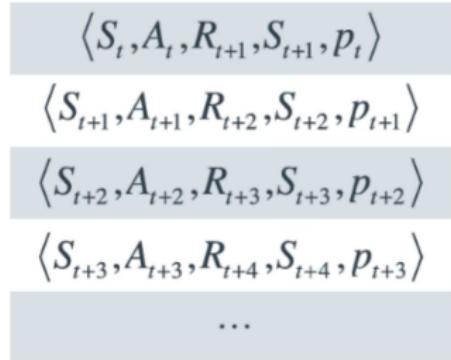
Value estimates by DQN (orange) and Double DQN (blue) on Atari games. The straight horizontal lines are computed by running the corresponding agents after learning concluded, and averaging the actual discounted return obtained from each visited state.

DQN Extensions II

DQN with Prioritized Experience Replay:

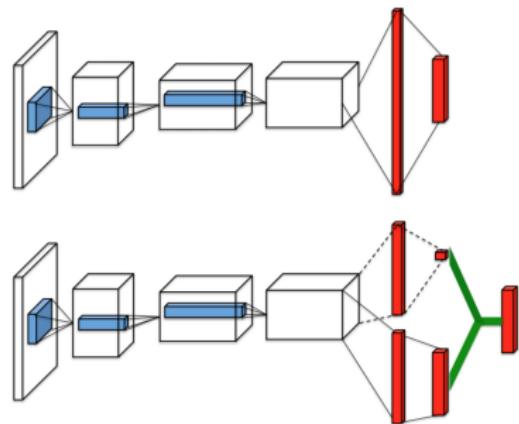
Prioritize transitions in proportion to the absolute Bellman error

$$p \propto \left| r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\eta}) - Q(s, a; \boldsymbol{\eta}) \right|$$



Dueling DQN: Split Q-networks into two streams to estimate value function and advantage function

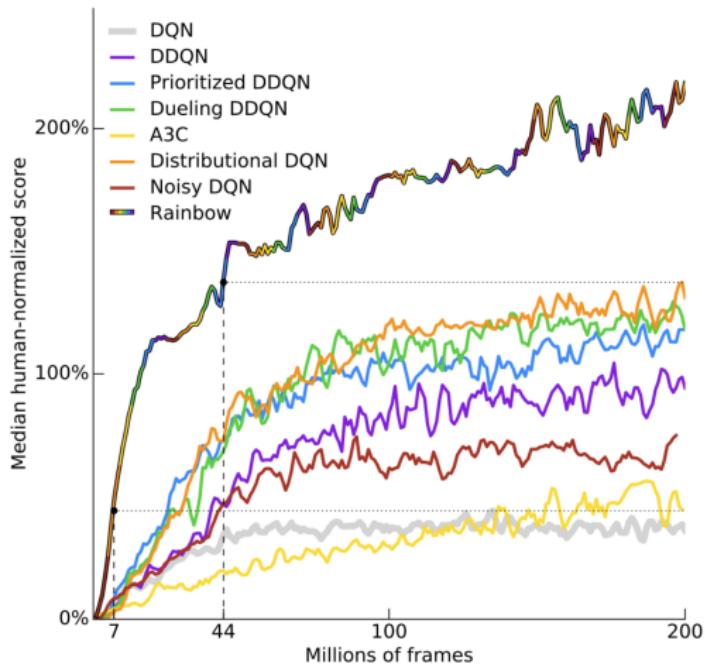
$$Q(s, a; \boldsymbol{\eta}, \alpha, \beta) = V(s; \boldsymbol{\eta}, \beta) + \overline{A}(s, a; \boldsymbol{\eta}, \alpha)$$



V and \overline{A} share parameters

DQN full Extension

Can these extensions be combined? Yes, Rainbow⁴



A whole zoo: [► DQN zoo](#)

⁴Hessel et al., Rainbow: Combining improvements in deep reinforcement learning. In 32nd AAAI Conf. on Artificial Intelligence, 2018

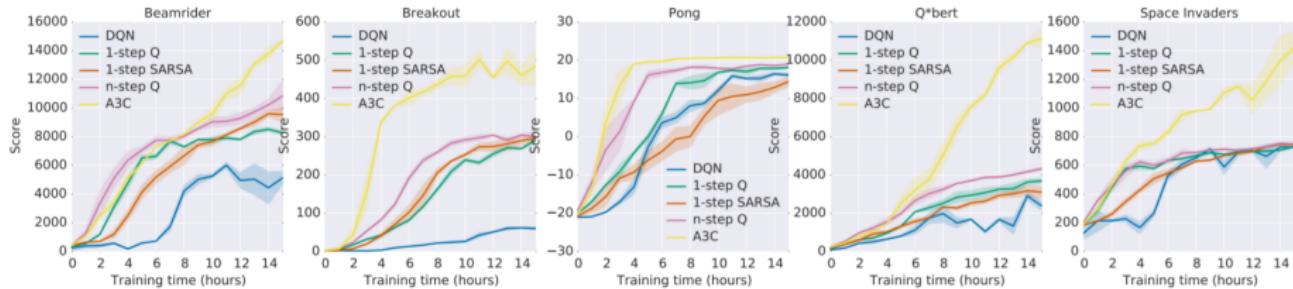
Policy-based/Actor-Critic DRL

Combine the actor-critic approach with Deep Q-Network for the critic update

- ▶ Asynchronous Advantage Actor-Critic (A3C))
- ▶ Soft Actor Critic (SAC)
- ▶ Deep deterministic policy gradient (DDPG): continuous control
- ▶ Twin Delayed DDPG (TD3): continuous control
- ▶ ...

A3C

Idea: advantage actor-critic + deep Q-network + asynchronous implementation



Comparison for DQN and A3C on five Atari 2600 games. 1-step Q means asynchronous one-step Q-learning.

- ▶ Ranking of A3C was worst a few slides back (Rainbow)
- ▶ Discrete action-state space

V. Mnih, A. Puigdomenech Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver & K. Kavukcuoglu,
Asynchronous methods for deep reinforcement learning, In International conference on machine learning,
pages 1928–1937. PMLR, 2016.

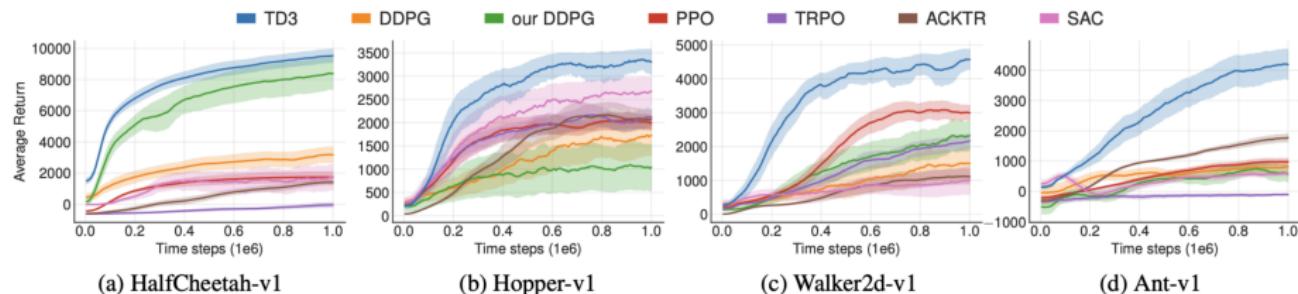
DDPG and TD3

DDPG: deterministic policy gradient + deep Q-network

- ▶ Select action $a \sim \mu(s; \theta) + \mathcal{N}(0, \sigma^2)$ (add noise to enhance exploration)
- ▶ Policy update direct estimate: $\nabla_{\theta} J(\theta) \approx 1/N \sum_i \nabla_a Q_{\eta}(s_i, \mu(s_i; \theta)) \nabla_{\theta} \mu(s_i; \theta)$

TD3: DDPG + clipped action exploration + delayed policy update + pessimistic double Q-learning

- ▶ Select action $a \sim \mu(s; \theta) + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2), -c, c)$
- ▶ Delayed policy update: update critic more frequently than policy



Learning curves for the OpenAI gym continuous control tasks

Summary

Deep Value-based Methods

- ▶ DQN
- ▶ Double DQN
- ▶ Dueling DQN
- ▶ DQN with prioritized experience replay
- ▶ Rainbow
- ▶ ...

Deep Policy-based/Actor-Critic Methods

- ▶ TRPO
- ▶ PPO
- ▶ A3C
- ▶ SAC
- ▶ DDPG/TD3
- ▶ ...

Deep RL Resources

- ▶ OpenAI Spinning up: <https://spinningup.openai.com/>
- ▶ The awesome list of deep RL (libraries and tutorials):
<https://github.com/kengz/awesome-deep-rl>

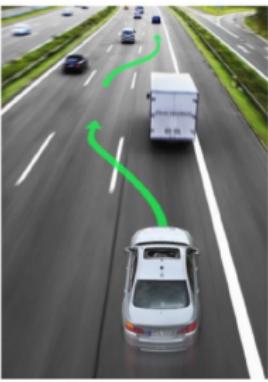
Sommaire

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
- 8 Imitation learning
 - Introduction
 - Offline Imitation Learning: Behavioral Cloning
 - Online Imitation Learning
 - Inverse Reinforcement Learning
 - Generative Adversarial Imitation Learning
- 9 Going beyond

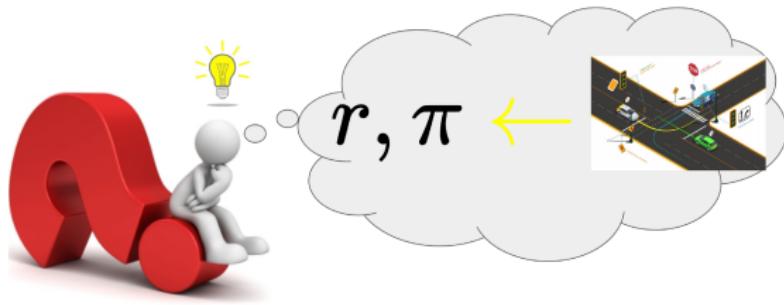
Learning from Demonstrations

Motivation

- ▶ In the previous lectures, we have learnt the optimal policy by maximizing the cumulative rewards. The reward functions are often manually designed to define the task.
- ▶ Can we instead learn the policy by capitalizing an expert's behavior?



Learning from demonstrations



- ▶ The reward function is unknown or is difficult to design in real world problems.
- ▶ It is easier/more natural to use “demonstrations” by experts.

Learning from demonstrations

Setting:

- ▶ Given an expert's demonstrations $\{(s_i, \pi_E(s_i))\}_i$ (offline trajectories or online queries)
- ▶ Reward signal is unobserved
- ▶ Transition model may be known or unknown

Goals and approaches:

- ▶ Recover the expert's policy π_E directly: imitation learning
- ▶ Recover the expert's latent reward function $r^{\pi_E}(s, a)$: inverse reinforcement learning (IRL)

One of the latest applications

Step 1

**Collect demonstration data,
and train a supervised policy.**

A prompt is sampled from our prompt dataset.

Explain the moon landing to a 6 year old



Some people went to the moon...

A labeler demonstrates the desired output behavior.

SFT



This data is used to fine-tune GPT-3 with supervised learning.

Step 2

**Collect comparison data,
and train a reward model.**

A prompt and several model outputs are sampled.

Explain the moon landing to a 6 year old

A
B

Explains gravity...
Explains one...

C
D

Moon is natural satellite of...

People went to the moon...



D > C > A = B

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

RM



D > C > A = B

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

Write a story about frogs

PPO

Once upon a time...

RM

r_k

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Training language models to follow instructions with human feedback, OpenAI, 2022

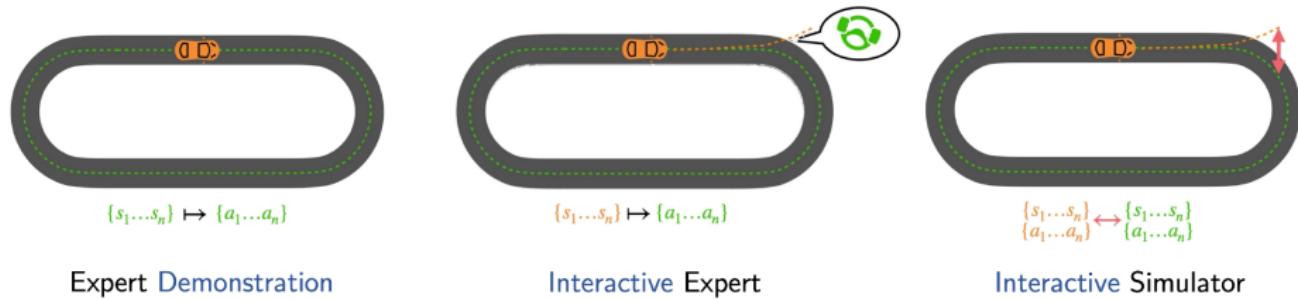
More applications

- ▶ Simulated highway driving (Abbeel and Ng, 2004)
- ▶ Helicopter acrobatics (Abbeel et al., 2006)
- ▶ Urban navigation (Ziebart et al, 2008)
- ▶ Human goal inference
- ▶ Object manipulation
- ▶ ...



Helicopter model and instance of its acrobatics

Three Main Categories of IL Approaches



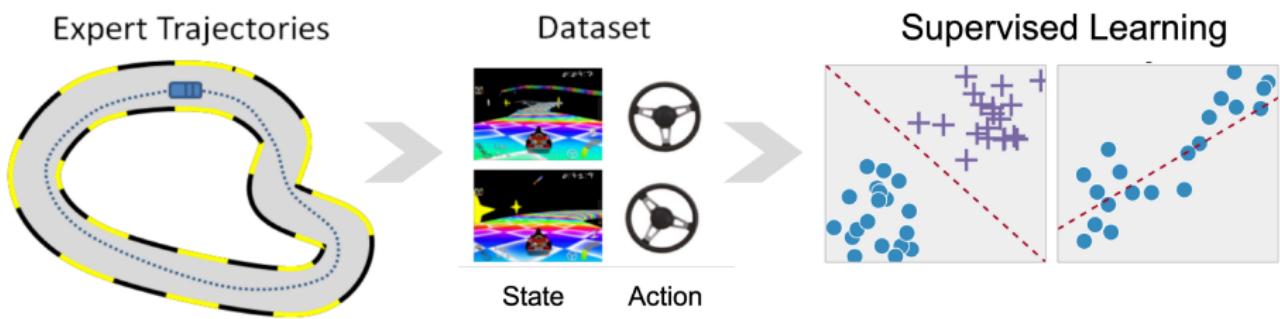
- ▶ Only access to offline expert's demonstrations
- ▶ Access to the environment and a queryable online expert
- ▶ Access to the environment and offline expert's demonstrations

Goal

Learn a policy $\pi \in \Pi$ that minimizes the **imitation gap**: $J(\pi_E) - J(\pi)$

Offline Imitation Learning: Behavioral Cloning

- ▶ We assume there is an expert that has the optimal policy π_E .
- ▶ Input: offline data from expert's demonstration $\mathcal{D} = \{s_i, a_i\}_{i=1}^n$ where $a_i \sim \pi_E(s_i)$
- ▶ Idea: directly learn the expert's policy via supervised learning



Behavioral Cloning

Maximum Likelihood Estimation (MLE)

$$\max_{\pi \in \Pi} \sum_{(s, a) \in \mathcal{D}} \log \pi(a | s)$$

Risk minimization

$$\min_{\theta} \mathbb{E}_{\substack{s \sim \mathcal{D}_\mu^{\pi_E} \\ a \sim \pi_E(\cdot | s)}} [\ell(\pi_\theta(\cdot | s), \pi_E(\cdot | s))]$$

where $\mathcal{D}_\mu^{\pi_E}$ is the state visitation distribution under policy π_E .

- ▶ Note that MLE can be viewed as minimizing the KL divergence between the expert's action distribution and that of the imitating policy:

$$\min_{\theta} \mathbb{E}_{\substack{s \sim \mathcal{D}_\mu^{\pi_E} \\ a \sim \pi_E(\cdot | s)}} [\text{KL}(\pi_E(\cdot | s) \| \pi_\theta(\cdot | s))] = \mathbb{E}_{\substack{s \sim \mathcal{D}_\mu^{\pi_E} \\ a \sim \pi_E(\cdot | s)}} \left[\log \left(\frac{\pi_E(a | s)}{\pi_\theta(a | s)} \right) \right]$$

Theoretical Guarantees of Behavior Cloning

Recall the policy value is defined as

$$J(\pi) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, a_t \sim \pi(\cdot \mid s_t), s_{t+1} \sim P(\cdot \mid s_t, a_t) \right]$$

Theorem (Error Bounds of Imitation Gap)

Assume $|r(s, a)| \leq 1, \forall s, a$. Suppose behavioral cloning returns an imitated policy $\hat{\pi}$ such that

$$\mathbb{E}_{s \sim \mathcal{D}_\mu^{\pi_E}} [\text{KL}(\pi_E(\cdot \mid s) \parallel \hat{\pi}(\cdot \mid s))] \leq \epsilon$$

then we have

$$J(\pi_E) - J(\hat{\pi}) \leq \frac{2\sqrt{2}}{(1-\gamma)^2} \sqrt{\epsilon}$$

- ▶ BC only ensures the learned policy $\hat{\pi}$ is close to π_E under the support of distribution $\mathcal{D}_\mu^{\pi_E}$.
- ▶ The term $1/(1-\gamma)^2$ reflects the cascading errors when performing under policy $\hat{\pi}$.
- ▶ The quadratic dependency on the effect horizon $H = 1/(1-\gamma)$ is not avoidable in worst case.
- ▶ Can be improved if knowing the transition model.

Behavioral Cloning: advantages and disadvantages

Advantages

- ▶ Simple.
- ▶ Effective

Disadvantages

- ▶ No long-term planning.
- ▶ Cascading errors.
- ▶ Possible mismatch between training and testing distributions.

From Pomerleau⁵:

Quote

When driving for itself, the network (ALVINN) may occasionally stray from the center of road and so must be prepared to recover by steering the vehicle back to the center of the road.

⁵Dean A. Pomerleau., Alvinn: An autonomous land vehicle in a neural network, In D. Touretzky, editor, Advances in Neural Information Processing Systems, volume 1. Morgan-Kaufmann, 1989.

A key difference with supervised learning

- ▶ The dataset \mathcal{D} is collected according to π_E , therefore behavioural cloning outputs the policy with parameters

$$\arg \min_{\theta} \mathbb{E}_{s \sim \lambda^{\pi_E}, a \sim \pi_E(\cdot | s)} [\ell(\pi_\theta(\cdot | s), \pi_E(\cdot | s))]$$

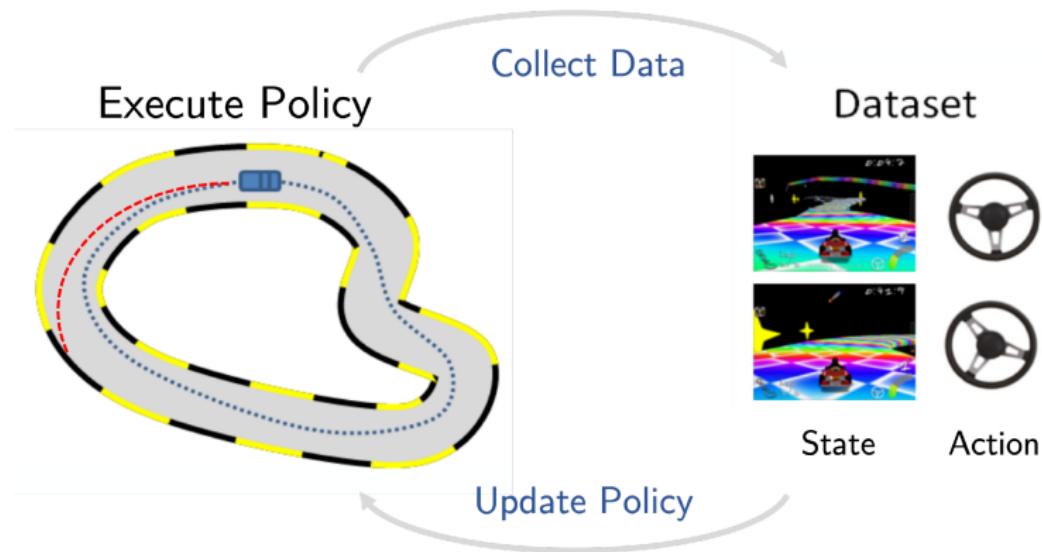
- ▶ However when we act in the environment with π_θ the states are sampled accordingly to λ^{π_θ} .
- ▶ It follows that we would like to minimize

$$\min_{\theta} \mathbb{E}_{s \sim \lambda^{\pi_\theta}, a \sim \pi_E(\cdot | s)} [\ell(\pi_\theta(\cdot | s), \pi_E(\cdot | s))]$$

- ▶ Scenario different from supervised learning where the decisions do not affect the data distribution.

Interactive Imitation Learning

- ▶ Aims to mitigate the cascading errors through interacting with the expert.
- ▶ We assume that we can query the expert π_E at any time and any state during training
- ▶ Idea: lean the expert's policy via online learning



Interactive Imitation Learning with Data Aggregation

- ▶ Dataset Aggregation (DAgger)⁶: iteratively build up a policy via supervised learning on aggregated data from the expert.
- ▶ Policy Aggregation (e.g., SMILe⁷): iteratively build up a policy by mixing newly trained policies.

Data/Policy Aggregation

Initialize π_0

for each iteration $t = 1, \dots, T$ **do**

 Generate trajectories h following π_t

 Collect new data $\mathcal{D}_t = \{(s, \pi_E(s)) \mid s \in h\}$ based on expert's feedback

Data Aggregation: run behavioral cloning with $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \dots \cup \mathcal{D}_t$ and obtain π_t

Policy Aggregation: run behavioral cloning with \mathcal{D}_t and obtain $\hat{\pi}_t$, set

$$\pi_t = \beta \hat{\pi}_t + (1 - \beta) \pi_{t-1}$$

end for

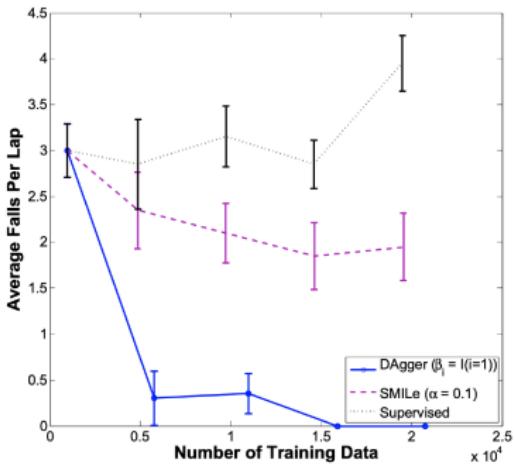
⁶] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In International Conference on Artificial Intelligence and Statistics (AISTATS), 2011.

⁷] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

Online vs. Offline Imitation Learning



3D car racing⁸



⁸S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In International Conference on Artificial Intelligence and Statistics (AISTATS), 2011.

Imitation Learning: can it scale?

- ▶ Imitation learning often relies on neural networks (NNs). NNs require massive data to shine.
- ▶ Interactive expert can be expensive, especially when the expert is human.
- ▶ Humans may have difficulty explaining “natural” actions.
- ▶ Challenging to perform reasoning about outcomes of actions.

Inverse Reinforcement Learning (IRL)

- ▶ The RL and IRL dichotomy:
 - ▶ RL recovers a nearly optimal policy from reward functions
 - ▶ IRL recovers a reward function and nearly optimal policy from demonstrations by an expert

	IRL	RL
Input	Expert demonstrations	Reward function
Output	Optimal policy	Optimal policy
	Reward function	

- ▶ IRL vs. BC:
 - ▶ IRL recovers a reward function and avoids the distribution shift issue in supervised learning.
 - ▶ IRL uses the MDP structure for the learning from expert demonstration, while behavioural cloning does not use the MDP structure.

Inverse Reinforcement Learning (IRL)

- ▶ Assume the expert is optimizing some reward function $r(s, a)$ in mind.
- ▶ The true reward function is unknown; π_E is the optimal policy of the MDP $\mathcal{M}(\mathcal{S}, \mathcal{A}, P, r, \gamma)$

IRL Objective

Find reward function $r(\cdot, \cdot) : \mathcal{S} \times \mathcal{A} \rightarrow [-1, 1]$ that explains the expert's behavior:

$$\pi_E \in \arg \max_{\pi \in \Pi} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, \pi \right]$$

Namely,

$$\mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, \pi_E \right] \geq \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 \sim \mu, \pi \right], \quad \forall \pi \in \Pi$$

NB: Note that this is a convex feasibility problem. But what are the challenges in solving it?

Challenges with Inverse Reinforcement Learning

- ▶ IRL does not necessarily have a unique reward solution. Note a trivial solution is $r = 0$.

Theorem (Reward shaping)

An expert policy π_E optimal in the MDP \mathcal{M} with reward r is optimal also in the MDP \mathcal{M} with reward function \hat{r} given by

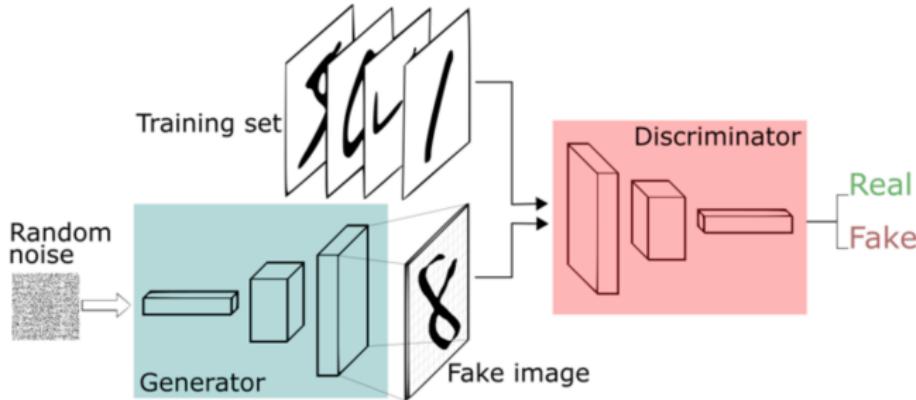
$$\hat{r}(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} [\Phi(s')] - \Phi(s)$$

where $\Phi : \mathcal{S} \rightarrow \mathbb{R}$ is called the potential function.

- ▶ Computationally expensive if we want to enumerate all policies to form the constraints;
- ▶ In practice, we do not observe π_E but only trajectories from π_E ;
- ▶ May be infeasible if the expert's policy is not optimal.

Generative Adversarial Network(GAN)

- GAN is framed as a min-max game between a generator and a discriminator.



- GAN: (\longrightarrow minimizing the Jensen-Shannon divergence)

$$\min_{G_\phi} \max_{D_\theta} \mathbb{E}_{x \sim P_{\text{data}}} [\log D_\theta(x)] + \mathbb{E}_z [\log (1 - D_\theta(G_\phi(z)))]$$

- Wasserstein GAN: (\longrightarrow minimizing the Wasserstein divergence)

$$\min_{G_\phi} \max_{f_\theta: \text{1-Lipschitz}} \mathbb{E}_{x \sim P_{\text{data}}} [f_\theta(x)] + \mathbb{E}_z [f_\theta(G_\phi(z))]$$

Generative Adversarial Networks (GANs)



2014
GAN



2018
GAN

Generative Adversarial Imitation Learning (GAIL)

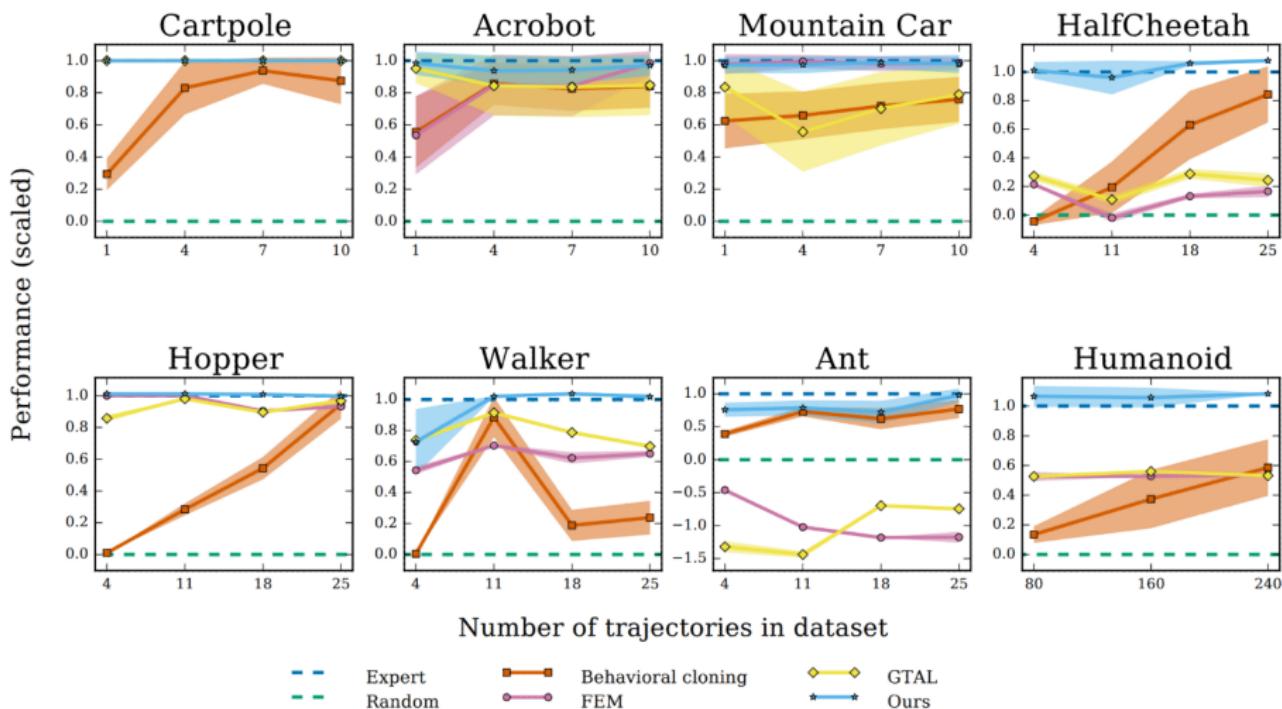
- ▶ GAIL⁹ aims to solve the min-max game for learning the policy given an expert policy π_E .

$$\min_{\theta} \max_{\phi} \mathbb{E}_{s,a \sim \lambda^{\pi_\theta}} [\log C_\phi(s, a)] + \mathbb{E}_{s,a \sim \lambda^{\pi_E}} [\log (1 - C_\phi(s, a))] - c \cdot H(\pi_\theta)$$

- ▶ We assume a differentiable parametrized policy π_θ .
- ▶ The discriminator tries to separate the data generated from learned policy from expert data.
- ▶ When $c = 0$, this is equivalent to minimize the Jensen-Shannon divergence between the state-action distributions between the expert policy and the learned policy.
- ▶ The min-max problem can be solved via efficient primal-dual methods. Unlike Max-Entropy IRL, it does not require expensive RL subroutines to learn the reward function.

⁹ Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 29. Curran Associates, Inc., 2016.

Numerical performance



Performance of learned policies among GAIL, Behavior Cloning (BC), Feature Expectation Matching (FEM), and Game-theoretic Apprenticeship Learning (GTAL)

Summary of Imitation Learning

Method	Reward learning	Access to environment	Interactive demonstrations	Pre-collected demonstrations
Offline IL	NO	NO	NO	YES
Interactive IL	NO	YES	YES	MAYBE
Inverse RL	YES	YES	NO	YES

- ▶ **Offline IL:** simple, scalable, but suffers from cascading error due to distribution shifts
- ▶ **Interactive IL:** alleviates the cascading error, but requires expensive expert queries
- ▶ **Inverse RL:** explains expert's behavior, but has poor sample efficiency and may not scale

Sommaire

- 1 Gentle introduction
- 2 Simplified framework: bandits
- 3 Minimax problems
- 4 Exact solution methods
- 5 Valued-based Reinforcement Learning
- 6 Policy gradient methods
- 7 Deep Reinforcement Learning
- 8 Imitation learning
- 9 Going beyond
 - Offline RL

Topics and Algorithms Covered So far

- ▶ **Dynamic Programming**
 - ▶ Value Iteration
 - ▶ Policy Iteration
 - ▶ Linear Programming
- ▶ **Value-based RL**
 - ▶ Monte Carlo Methods
 - ▶ TD, SARSA, Q-learning
 - ▶ Function Approximation
- ▶ **Policy-based RL**
 - ▶ Policy Gradient Method
 - ▶ Natural Policy Gradient Method
 - ▶ TRPO and PPO
- ▶ **Actor-Critic and Deep RL**
 - ▶ Deep Q Network and Extensions
 - ▶ Deep Actor-Critic (A3C, DDPG, TD3)
- ▶ **Imitation Learning and Inverse RL**
 - ▶ Behavior Cloning, GAIL
 - ▶ Interactive IL (DAgger, SMILe)
 - ▶ Max Margin and Max Entropy IRL

What's beyond?

- ▶ Multi-agent RL and Markov Games (Nash Q-learning)
- ▶ Episodic RL
- ▶ Model-based RL
- ▶ Batch and Offline RL
- ▶ Safety in RL
- ▶ Distributionally Robust RL
- ▶ Preference-based RL
- ▶ ...
- ▶ Partially Observable RL
- ▶ Meta RL
- ▶ Multi-task RL
- ▶ Causal RL
- ▶ Lifelong RL
- ▶ Continual RL
- ▶ Hierarchical RL
- ▶ ...

From Online Interaction to Offline RL

Online interaction can be impractical for many applications:

- ▶ Too expensive, *e.g.*, in robotics, education
- ▶ Too risky, *e.g.*, autonomous driving, medical treatment



Online RL: integrates data collection and optimization



Offline RL: decouples data collection and optimization

Offline (Batch) RL Setting

Objectives

Given a fixed previously collected dataset:

$$\mathcal{D} = \left\{ h^j \right\}_{j=1}^m : h^j = \left(s_0^j, a_0^j, r_0^j, \dots, s_H^j, a_H^j, r_H^j \right)$$

where $s_{t+1} \sim P(\cdot | s_t, a_t)$, $a_t \sim \pi_\beta(\cdot | s_t)$, $r_t \sim R(s_t, a_t)$ and π_β is called the behavior policy, we are interested in:

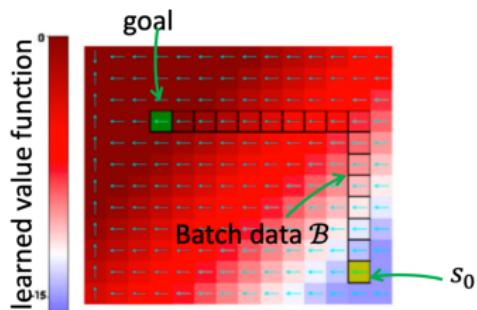
- ▶ Offline Policy Evaluation (OPE): estimate V^π for a given policy π
- ▶ Offline Policy Optimization: find the optimal policy that maximizes V^π
- ▶ Offline Policy Confidence Estimation: estimate the upper and lower bounds of policy values

Remark: This setting is distinct from IRL:

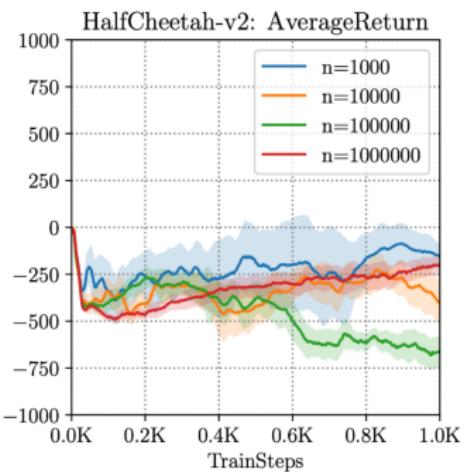
- ▶ Here data are given by a behavior policy (which may be unknown), not by the “expert policy”
- ▶ Here one has access to reward signal from previous experiences

Challenges with Offline RL

- In a nutshell, **distribution shift!** At its core, offline RL is about making and answering counterfactual queries.
- Function approximation further exacerbates this issue.



Value function learned from supervised learning falsely extrapolates the unseen states and suffers from overestimation errors



SAC with offline data: increasing size of the static dataset does not rectify the issue

Offline RL via Dynamic Programming

Fitted-Q Iteration is one of the most popular batch version of Q-learning. Let $\mathcal{D} = \{s_i, a_i, r_i, s'_i\}_{i=1}^n$ with $(s_i, a_i) \sim \mu(s, a)$

Fitted-Q Iteration (FQI)

- ▶ Start with $Q_0 \in \mathcal{F}$
- ▶ Iteratively perform the regression

$$Q_{t+1} \in \arg \min_{Q \in \mathcal{F}} \sum_{i=1}^n \left(r_i + \gamma \max_{a'} Q_t(s'_i, a') - Q(s_i, a_i) \right)^2$$

FQI may oscillate and a fixed point solution may not exist; under (strong) assumptions on data coverage and function class, sample complexity analysis can be obtained.

More Resources

- ▶ NeurIPS 2020 Tutorial:
Aviral Kumar and Sergey Levine, *Offline Reinforcement Learning: From Algorithms to Practical Challenges*
<https://sites.google.com/view/offline-rl-tutorial>
- ▶ Survey paper:
Sergey Levine, Aviral Kumar, George Tucker, Justin Fu, *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*, 2020.
<https://arxiv.org/abs/2005.01643>

Learning objectives: are we there yet?

By the end of the course, you should be able to

- ✓ Identify the strengths and limitations of various RL algorithms,
- ✓ Understand the theoretical properties of RL algorithms,
- ✓ Recognize the common boundary of optimization and RL,
- ✓ Formulate and solve sequential decision-making problems by applying relevant RL tools,
- ✓ Generalize or discover “new” applications, algorithms, or theories of RL towards conducting independent research on the topic.

Where to go from here?

Summer Schools

- ▶ Ten-day Reinforcement Learning Summer School:
<https://rlsummerschool.com/>
- ▶ CIFAR Deep Learning + Reinforcement Learning (DLRL) Summer School
<https://dlrl.ca/>
- ▶ Cooperative AI Summer School:
<https://www.cooperativeai.com/summer-school/summer-school-2024>

Flagship RL Conferences

- ▶ EWRL (European Workshop on Reinforcement Learning):
<https://ewrl.wordpress.com/ewrl17-2024/>
- ▶ L4DC (Learning for Dynamics and Control Conference)
<https://sites.google.com/umich.edu/l4dc2025/>