Reinforcement Learning

Lecture 2 Classical Ideas

Nan Ye

School of Mathematics and Physics
The University of Queensland

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

AMSI 24 WINTER SCHOOL
24 JUN – 5 JUL 2024 | UQ

Roadmap

- Introduction and overview

  motivation, bandits, big picture
- Classical ideas

  temporal difference methods, policy gradient, . . .
- Deep Reinforcement learning

  neural networks, DQN, DDPG, . . .
- Advanced techniques

  representation learning, stabilization, few-shot learning
- Applications

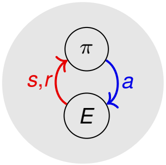
  AlphaGo, AlphaTensor, . . .

environment model

bandits, MDPs, POMDPs

learning target

model, value, policy



behavior policy

exploration vs exploitation

update rules

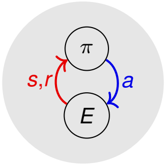
experience, loss

four dimensions

policy evaluation / prediction

$\pi, E/\text{interactions} \rightarrow V_\pi$

value iteration, linear system, Monte Carlo, ...



planning / control

$E \rightarrow \text{argmax}_\pi V_\pi$

value iteration, policy iteration, Monte Carlo, ...

reinforcement learning

interactions with $E \rightarrow \text{argmax}_\pi V_\pi$

Q-learning, SARSA, policy gradient, ...

π = *policy*, V_π = *policy value*, E = *environment*

three problems

Markov Decision Processes (MDPs)

MABs are stateless, some problems have states $\Rightarrow$ MDP.
State: useful environment information for making decisions.

MDP $(p_0, S, A, T, R, \gamma)$

- initial state distribution p_0
- state space S
- action space A
- transition model $T(s' \mid s, a)$
- reward function $R(s, a)$
- discount factor $\gamma \in [0, 1)$

$$\begin{tikzcd}
\pi \arrow[d, bend left, "a_t \sim \pi(\cdot \mid s_t)"] \\
E \arrow[u, bend left, "{s_{t+1}, \; r_t = R(s_t, a_t)}"]
\end{tikzcd}$$

$$s_{t+1} \sim T(\cdot \mid s_t, a_t)$$

Objective: find an optimal (stochastic) policy $\pi : S \to \Delta_A$

$$\max_{\pi} V(\pi) := \mathbb{E}_{s \sim p_0} V_\pi(s) := \mathbb{E}_{s \sim p_0} \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, \pi\right).$$

$V_\pi(s)$: value (expected total discounted reward) of π starting from s.
Δ_A: probability distributions on A.

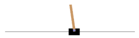
Remarks. (a) often exists an optimal deterministic policy $\pi : S \to A$, so they are popular too.
(b) often interested in finite horizon problems too.

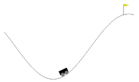
many publicly available benchmark environments

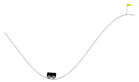
generally follow OpenAI's Gym API

Gym lives as Gymnasium now

Acrobot


Cart Pole

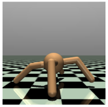

Mountain Car Continuous

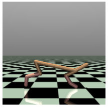

Mountain Car

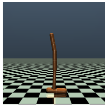
Pendulum

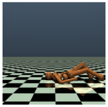
classic control

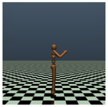
https://gymnasium.farama.org/environments/classic_control/

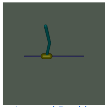

Ant

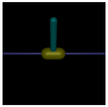

Half Cheetah

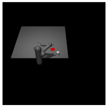

Hopper

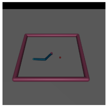

Humanoid Standup


Humanoid


Inverted Double Pendulum


Inverted Pendulum


Pusher

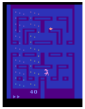

Reacher

MuJoCo

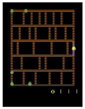
https://gymnasium.farama.org/environments/mujoco/

Adventure

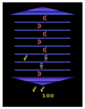
Air Raid



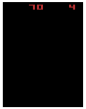
Alien


Amidar

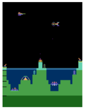
Assault



Asterix

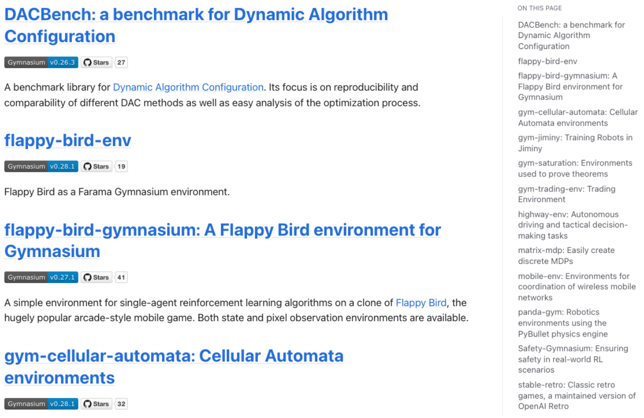


Asteroids


Atlantis

Atari

`https://gymnasium.farama.org/environments/atari/`


DACBench: a benchmark for Dynamic Algorithm Configuration

Gymnasium v0.26.3 Stars 27

A benchmark library for Dynamic Algorithm Configuration. Its focus is on reproducibility and comparability of different DAC methods as well as easy analysis of the optimization process.

flappy-bird-env

Gymnasium v0.28.1 Stars 19

Flappy Bird as a Farama Gymnasium environment.

flappy-bird-gymnasium: A Flappy Bird environment for Gymnasium

Gymnasium v0.27.1 Stars 41

A simple environment for single-agent reinforcement learning algorithms on a clone of Flappy Bird, the hugely popular arcade-style mobile game. Both state and pixel observation environments are available.

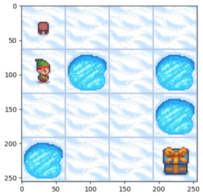
gym-cellular-automata: Cellular Automata environments

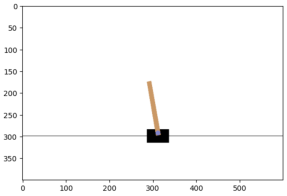
Gymnasium v0.28.1 Stars 32

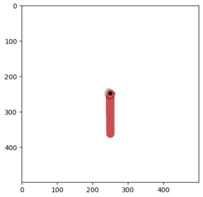
ON THIS PAGE

many 3rd party environments

https://gymnasium.farama.org/environments/third_party_environments/


Frozen Lake
disc. S & A


Cartpole
cont. S & disc. A


Pendulum
cont. S & A

useful toy problems – good starting points

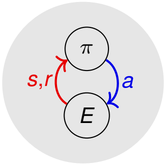
try me
https://colab.research.google.com/drive/1Qr19jg97Q4mRVrQ3yr3_mudEOD6IQCNr

Planning

policy evaluation / prediction

π, E/interactions $\rightarrow V_\pi$

value iteration, linear system, Monte Carlo, ...



planning / control

$E \rightarrow \text{argmax}_\pi V_\pi$

value iteration, policy iteration, Monte Carlo, ...

reinforcement learning

interactions with $E \rightarrow \text{argmax}_\pi V_\pi$

Q-learning, SARSA, policy gradient, ...

plan $\rightarrow$ RL

- a building block: learn a model, then plan
- a source of inspiration: sample-based approximation to exact operations in planning

- The optimal value function V^* satisfies the Bellman optimality equation

$$V^*(s) = \max_a \left(\sum_{s'} T(s' \mid s, a) \left(R(s, a) + \gamma V^*(s') \right) \right).$$

- Equivalently, V^* is the fixed point of the Bellman operator H:

$$V^* = H(V^*),$$

  where the Bellman operator $H : \mathbf{R}^S \to \mathbf{R}^S$ is defined by

$$H(V)(s) \overset{def}{=} \max_a \left(\sum_{s'} T(s' \mid s, a) \left(R(s, a) + \gamma V(s') \right) \right),$$

- If we choose an arbitrary V_0, and $V_{t+1} = H(V_t)$, then V_t converges to V^* (proved using Banach fixed-point theorem).

Algorithm The Value Iteration algorithm for computing $\pi \approx \pi^*$

```
1: Initialize $V_0$                          ▷ often set to 0 if no good estimates available
2: for $t = 1$ to $T$ do
3:     $V_t \leftarrow H(V_{t-1})$            ▷ improve estimates using Bellman operator
4:     $V \leftarrow V_t$                     ▷ use $V$ to remember most recent estimates
5:     Terminate if $\|V_t - V_{t-1}\|_\infty < \epsilon$
6: $\pi(s) = \text{argmax}_a \left(R(s,a) + \gamma \sum_{s'} T(s' \mid s,a) V(s')\right)$.
```

computing π^* from V^* is expensive, and requires knowledge of R and T

- Often easier to work with *action-value function* $Q_\pi(s,a)$, defined as

$$Q_\pi(s,a) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right).$$

- Bellman equation for the optimal action value function Q^*:

$$Q^*(s,a) = \left(\sum_{s'} T(s' \mid s,a)\left(R(s,a) + \gamma \max_{a'} Q^*(s',a')\right)\right).$$

- Equivalently, Q^* is the fixed point of the Bellman operator H (*overloaded notation!*):

$$Q^* = H(Q^*),$$

  where the Bellman operator $H : \mathbf{R}^{S\times A} \to \mathbf{R}^{S\times A}$ is defined by

$$H(Q)(s,a) \overset{def}{=} \left(\sum_{s'} T(s' \mid s,a)\left(R(s,a) + \gamma \max_{a'} Q(s',a')\right)\right),$$

- If we choose an arbitrary Q_0, and $Q_{t+1} = H(Q_t)$, then Q_t converges to Q^*.

Algorithm The Q-Iteration algorithm for computing $\pi \approx \pi^*$

```
1: Initialize $Q_0$                              ▷ often set to 0 if no good estimates available
2: for $t = 1$ to $T$ do
3:     $Q_t \leftarrow H(Q_{t-1})$                ▷ improve estimates using Bellman operator
4:     $Q \leftarrow Q_t$                         ▷ use $Q$ to remember most recent estimates
5:     Terminate if $\|Q_t - Q_{t-1}\|_\infty < \epsilon$
6: $\pi(s) = \mathrm{argmax}_a Q(s, a)$
```

π^* can be computed using Q alone, but $\mathrm{argmax}_a Q(s, a)$ can be hard

Q-learning

MDPs with finitely many states

- Q-learning (Watkins and Dayan, 1992) tries to directly estimate the optimal Q-function by solving the Bellman optimality equation

$$Q^*(s, a) = \sum_{s'} T(s' \mid s, a)(R(s, a) + \max_{a'} Q^*(s', a'))$$

  Key idea: replace expectation wrt s' using a sampled transition.

- If we experience a transition (s, a, s', r), then we can use it to perform an update

$$Q(s, a) \leftarrow Q(s, a) + \alpha\underbrace{(\overbrace{r + \gamma \max_{a'} Q(s', a')}^{\text{TD target}} - Q(s, a))}_{\text{TD}},$$

  where $\alpha > 0$ is the learning rate, s is the current state, and s' and r are the next state and the reward obtained after executing a.

- Think of α as a level of trust on the sampled transition.

Algorithm Tabular Q-learning

```
1: Initialise the state-action value function $Q$
2: while termination condition not met do
3:     Execute the behavior policy to obtain a new experience $(s, a, s', r)$
4:     Perform TD update for $Q$ using the new experience
           $$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a)).$$
```

- Various termination criteria can be used
  - e.g. little change over recent updates, maximum number of interaction, maximum computation time
- A commonly used behavior policy is the ϵ-greedy policy, which executes a random action w.p. $\epsilon > 0$, and the greedy action $\mathrm{argmax}_a Q(s, a)$ w.p. $1 - \epsilon$.

MDPs with a very large state space – function approximation

- If we have too many states, we can't use a table to store the Q-function.
- Typically, we use a parametric representation $Q_\theta(s, a)$ in this case.
- The update step in the Q-learning algorithm becomes

$$\theta \leftarrow \theta - \alpha(Q_\theta(s, a) - r - \gamma \max_{a'} Q_\theta(s', a')) \nabla Q_\theta(s, a).$$

  Why? This performs a gradient descent on the squared TD error

$$(Q_\theta(s, a) - r - \gamma \max_{a'} Q_{\theta^-}(s', a'))^2,$$

  where $\theta^- = \theta$ is treated as fixed parameters.
- Tabular Q-learning is a special case: $Q_\theta(s, a) = \sum_{s' \in S} \theta_{s'} I(s' = s)$.

Algorithm Q-learning with function approximation

```
1: Initialise the state-action value function $Q_\theta$
2: **while** termination condition not met **do**
3:     Execute an appropriate behavior policy to obtain a new experience $(s, a, s', r)$
4:     Perform TD update
           $$\theta \leftarrow \theta - \alpha(Q_\theta(s, a) - r - \gamma \max_{a'} Q_\theta(s', a')) \nabla Q_\theta(s, a).$$
```

SARSA

- SARSA is the same as Q-learning, except that for each update, it first observes a sequence s, a, r, s', a' (that's why the name SARSA), then update

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a)).$$

- Function approximation can be applied too.

$$\text{Q-learning (off-policy):}\quad Q(s,a) \leftarrow Q(s,a) + \alpha(\overbrace{r + \gamma \max_{a'} Q(s',a')}^{\text{target is greedy policy}} - Q(s,a))$$

$$\text{SARSA (on-policy):}\quad Q(s,a) \leftarrow Q(s,a) + \alpha(\overbrace{r + \gamma Q(s',a')}^{\text{target is }\epsilon\text{-greedy policy}} - Q(s,a)).$$

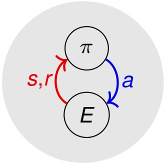
- Q-learning is off-policy as target policy (greedy) $\neq$ behavior policy (ϵ-greedy).
- SARSA is an on-policy as target policy = behavior policy.

environment model

bandits, MDPs, POMDPs

learning target

model, value, policy



behavior policy

ϵ-greedy

update rules

TD minimization

Q-learning/SARSA in the big picture

Policy Optimization Methods

- Why policy optimization?
  - learn Q: computing the policy $\pi(s) = \text{argmax}_a Q(s, a)$ can be hard
  - learn V: requires lookahead and optimize
  - learn π: policy is directly available
- Various policy optimization algorithms: REINFORCE, actor-critic, DPG, …

REINFORCE

- REINFORCE (Williams, 1992) directly optimises a parametric policy $\pi_\theta(a \mid s)$ by maximizing its value function

$$V(\theta) = \sum_{\tau} p(\tau \mid \theta) R(\tau) = \mathbb{E}_{\tau \sim p}\, R(\tau),$$

  where
  - $\tau = (s_0, a_0, s_1, a_1, \ldots)$ is a trajectory (state-action sequence),
  - $p(\tau \mid \theta)$ is the distribution of trajectory τ when playing π_θ, and
  - $R(\tau)$ is the total (discounted) reward collected along τ.
- It computes a stochastic gradient of $V(\theta)$ at each iteration, and then performs gradient ascent.
- A simple parametrization: $\pi_\theta(a \mid s)$ as a logistic regression model.

- Usually, it is often computationally intractable to evaluate $V(\theta)$ first, and then evaluate its gradient,
    - in the discrete state case, $V(\theta)$ involves summing over a large number of trajectories.
    - in the continuous state case, computing $V(\theta)$ involves evaluating a complex integral.

Policy gradient theorem

$$\nabla V(\theta) = \mathbb{E}_{\tau \sim p} R(\tau) \nabla \ln p(\tau \mid \theta).$$

- Why? Because $\nabla \ln p(\tau \mid \theta) = \frac{\nabla p(\tau \mid \theta)}{p(\tau \mid \theta)}$.
- This gives us a Monte Carlo estimate of the gradient

$$\nabla V(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau^{(i)}) \nabla \ln p(\tau^{(i)} \mid \theta),$$

  where the trajectories $\tau^{(1)}, \ldots, \tau^{(N)}$ are randomly sampled from $p(\cdot \mid \theta)$.
- We need to relate $\nabla \ln p(\tau \mid \theta)$ back to the policy π_θ.

Policy gradient theorem

$$\nabla V(\theta) = \mathbb{E}_{\tau \sim p} \sum_{t=0}^{|\tau|-1} R(\tau) \nabla \ln \pi_\theta(a_t \mid s_t).$$

where $|\tau|$ denotes the length of a trajectory (number of state-action pairs).

policy gradient learning = weighted log-likelihood maximization

- Why? Note that

$$p(\tau \mid \theta) = p(s_1) \prod_{t=0}^{|\tau|-1} \pi(a_t \mid s_t, \theta) p(s_{t+1} \mid s_t, a_t)$$

$$\nabla \ln p(\tau^{(i)} \mid \theta) = \sum_{t=0}^{|\tau_i|-1} \nabla \ln \pi_\theta(a_t^{(i)} \mid s_t^{(i)}).$$

- While $p(\tau^{(i)} \mid \theta)$ depends on the transition probabilities, the gradient of the log probability does not!

Algorithm REINFORCE algorithm

```
1: while not terminated do
2:     Simulate $\pi_\theta$ to collect trajectories $\tau^{(1)}, \ldots, \tau^{(N)}$.
3:     Update $\theta$ using
```

$$\theta \leftarrow \theta + \alpha \left(\frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{|\tau^{(i)}|-1} R(\tau^{(i)}) \, \nabla \ln \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) \right) \right).$$

REINFORCE is an on-policy method.

Policy gradient theorem

$$\nabla V(\theta) = \mathbb{E}_{\tau \sim p} \sum_{t=0}^{|\tau|-1} \gamma^t (R(\tau_{\geq t}) - b(s_t)) \nabla \ln \pi_\theta(a_t \mid s_t).$$

where for $\tau_{\geq t} = (s_t, a_t, \ldots)$ for $\tau = (s_1, a_1, \ldots)$, and $b(s)$ is an arbitrary function of state.

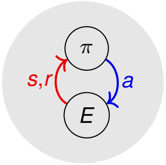
- Focus on future but not past: $\tau_{<t} = (s_0, a_0, \ldots, s_{t-1}, a_{t-1})$ has no effect on $\pi_\theta(a_t \mid s_t)$.
- Use a baseline $b(s)$ for variance reduction.
- More policy gradients: (Schulman et al., 2015)

environment model

bandits, MDPs, POMDPs

learning target

model, value, policy



behavior policy

target policy

update rules

weighted likelihood maximization

policy optimization in the big picture

Model-based RL

Algorithm A general model-based RL approach

```
1: initialize an estimated environment model $\tilde{M}$
2: for $t = 1, 2, \dots$ do
3:     compute optimal policy $\tilde{\pi}^*$ for $\tilde{M}$
4:     collect experience by running the $\epsilon$-greedy policy $\tilde{\pi}^*_\epsilon$
5:     update the environment model $\tilde{M}$ based on collected experience
```

- $\tilde{\pi}^*_\epsilon$ folows $\tilde{\pi}^*$ with probability $1 - \epsilon$ and takes a random action otherwise.

Tabular model-based RL

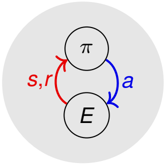
- Initialization
    - Each $R(s, a)$ can be initialized to the maximum possible value to encourage exploration
    - $\tilde{M}$ can be initialized with a "pseudo-count" $n_{s,a,s'}$ for each transition (s, a, s').
- Update
    - Reward update: compute average of rewards encountered.
    - Transition model update: update the transition count $n_{s,a,s'}$ to include both the pseudo-count and the actual count, then compute $T(s'|s, a) = n_{s,a,s'} / \sum_{s''} n_{s,a,s''}$.
- The planning problem of computing $\tilde{\pi}^*$ for $\tilde{M}$ can be solved using value/Q iteration.

environment model

bandits, MDPs, POMDPs

learning target

model, value, policy



behavior policy

ϵ-greedy

update rules

reward regression, transition density estimation

simple model-based RL in the big picture

- computing average reward is least squares regression
- frequency-based transition probability is regularized maximum likelihood estimation

- Advanced model-based RL: PlaNet (Hafner et al., 2019b), DreamerV1 (Hafner et al., 2019a), DreamerV2 (Hafner et al., 2020), DreamerV3 (Hafner et al., 2023)
- Model-based methods can be more sample efficient than model-free methods.

Roadmap

- Introduction and overview

  motivation, bandits, big picture
- Classical ideas

  temporal difference methods, policy gradient, . . .
- Deep Reinforcement learning

  neural networks, DQN, DDPG, . . .
- Advanced techniques

  representation learning, stabilization, few-shot learning
- Applications

  AlphaGo, AlphaTensor, . . .

References I

Hafner, D. et al. (2019a). Dream to control: Learning behaviors by latent imagination. In: *arXiv preprint arXiv:1912.01603*.

Hafner, D. et al. (2019b). Learning latent dynamics for planning from pixels. In: *International Conference on Machine Learning*. PMLR, pp. 2555–2565.

Hafner, D. et al. (2020). Mastering Atari with Discrete World Models. In: *arXiv preprint arXiv:2010.02193*.

Hafner, D. et al. (2023). Mastering diverse domains through world models. In: *arXiv preprint arXiv:2301.04104*.

Schulman, J. et al. (2015). High-dimensional continuous control using generalized advantage estimation. In: *arXiv preprint arXiv:1506.02438*.

Watkins, C. J. and P. Dayan (1992). Q-learning. In: *Machine learning* 8.3-4, pp. 279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. In: *Machine learning* 8.3-4, pp. 229–256.