RL Reading Group – Prep Session 1

Zhiyu Zhang

09/16/2019

Overview

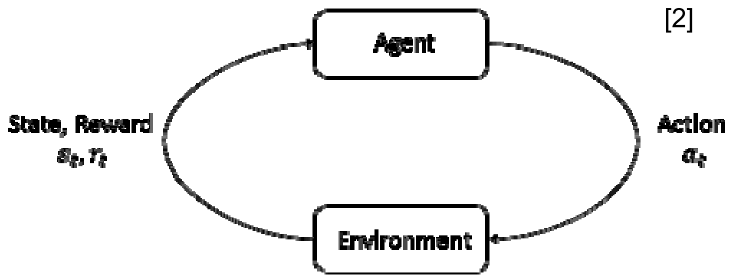
- Problem formulation
- Solution techniques
  - Model given: Dynamic Programming (DP)
  - Curse of dimensionality and approximate DP
  - Model unknown: Reinforcement learning (RL)
    - Tabular Monte Carlo
    - Tabular Temporal Difference
    - Q learning with function approximation, the “deadly triad”

- Next time: policy gradient, actor-critic, RL subcategories...

Finite Markovian Decision Problems (finite MDP)

- A finite MDP is a 5-tuple $\mathcal{M} =< S, A, R, P, \rho_0 >$:
  - S is the state space, finite
  - A is the action space, finite
  - $R: S \times A \times S \rightarrow \mathbb{R}$ is the reward function

    The reward at time t is $r_t = R(s_t, a_t, s_{t+1})$
  - $P: S \times A \times S \rightarrow [0,1]$ is the transition probability of the Markov Chain induced by the applied action, e.g. $P(s_{t+1}|s_t, a_t)$
  - ρ_0 is the starting state distribution



[2]

- Can be extended to continuous state / action space, with technical assumptions

[1] Bertsekas, Dimitri P. Abstract dynamic programming. Athena Scientific, 2018.
[2] https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

Finite Markovian Decision Problems (finite MDP)

- We pick a stationary (stochastic) policy π for the agent

$$a_t \sim \pi(\cdot\,|s_t)$$

- When applying π, denote the first N step sample path starting at s_0 as $\tau_N(s_0) = \{s_t, a_t\}_{t=1}^{N}$
- Denote the infinite horizon sample path starting at s_0 as $\tau(s_0) = \{s_t, a_t\}_{t=1}^{\infty}$

- We focus on the infinite horizon setting.
- The performance of π is characterized by one of the three objectives we choose:
  - γ-discounted cumulative reward, with discount factor $\gamma \in (0,1)$ Well-behaved
  - Undiscounted cumulative reward, with discount factor $\gamma = 1$ Needs restriction
  - Infinite horizon average reward Very useful but not popular in RL due to difficulty

RL mainly consider the first two

Finite Markovian Decision Problems (finite MDP)

- Consider only the first two types, they can both be expressed as the value function

$$V_\pi(s_0) = \limsup_{N\to\infty} E_{\tau_N|\pi,s_0}\left[\sum_{t=0}^{N-1} \gamma^t R(s_t, a_t, s_{t+1})\right]$$

- We want to find an optimal policy π^* that maximizes the value function for all starting state s_0:

$$V(s_0) = \sup_\pi V_\pi(s_0) \qquad \forall s_0 \in S$$

- π^* can be any policy that attains this supremum; V is the optimal value function

---

- Note that the value function is defined in the DP fashion [1]; in (Deep) RL [2], it is

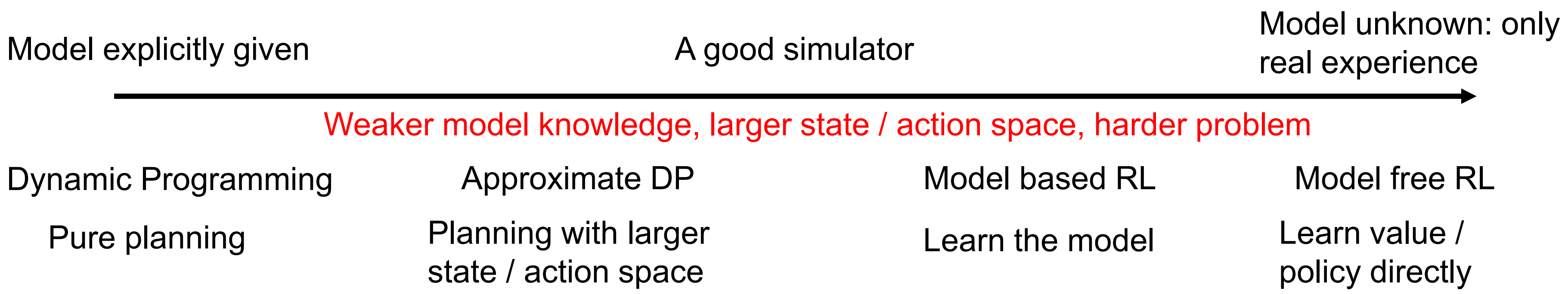
$$V_\pi(s_0) = E_{\tau|\pi,s_0}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right]$$

Less concrete but in “good cases” (discounted, bounded R) they are the same, and lim sup becomes lim

[1] Bertsekas, Dimitri P. Abstract dynamic programming. Athena Scientific, 2018.
[2] https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

Solution techniques: DP, ADP and RL

- The solution technique largely depends on what is available to us: model given or unknown
- Define the model as the reward function R and the transition probability P


Model explicitly given
A good simulator
Model unknown: only real experience
Weaker model knowledge, larger state / action space, harder problem
Dynamic Programming
Approximate DP
Model based RL
Model free RL
Pure planning
Planning with larger state / action space
Learn the model
Learn value / policy directly

- P is objective: if we don't know it, we cannot use it
- R can be objective / subjective:
  As the real outcome has multiple ways to quantify, we can often pick R (weird...)

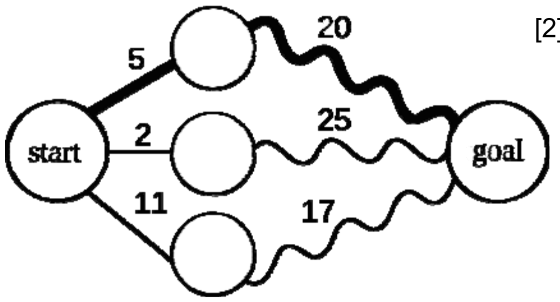
Dynamic Programming: The principle of optimality

- In DP, since the dynamics is known, there is no need to explore; assume π is deterministic

  The action a is chosen from the allowable control set at s: $a \in A(s)$

- Bellman's priciple of optimality [1]:

  An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.



[2]

In the deterministic case, we expect something like

(Bellman equation)
Value at current state = max over action [immediate reward + value at next state | action]

Need more careful analysis

[1] https://en.wikipedia.org/wiki/Bellman_equation
[2] https://en.wikipedia.org/wiki/Dynamic_programming

DP operators

- Let $\mathcal{X}$ be the set of functions $S \to \mathbb{R}$; (the set of all possible value functions)

- For any policy π, define the DP operator T_π on any $v \in \mathcal{X}$ as

$$(T_\pi v)(s) = E_{s'}[R(s, \pi(s), s') + \gamma v\ (s')]$$

- Define the optimal DP operator (Bellman operator) T on $v \in \mathcal{X}$ as

$$(Tv)(s) = \max_{a \in A(s)} E_{s'}[R(s, a, s') + \gamma v(s')]$$

- The DP operators are characterized by two properties that are central to solving the problem:
  Monotonicity and contraction

Monotonicity

- Monotonicity of DP operators are defined as follows:

- (Assumption 1.2.1, [1]): The DP operator $T_{(\cdot)}$ without π specified is monotone if $\forall v_1, v_2 \in \mathcal{X}$ such that $\forall s \in S$, $v_1(s) \leq v_2(s)$, we have
$$T_\pi v_1(s) \leq T_\pi v_2(s) \qquad \forall s \in S, \forall \pi \text{ admissible}$$

- If $T_{(\cdot)}$ is monotone, then the Bellman operator T is monotone

- In the two typical types of MDP problems we consider (cumulative discounted or undiscounted reward problem with assumptions), $T_{(\cdot)}$ is monotonic

[1] Bertsekas, Dimitri P. Abstract dynamic programming. Athena Scientific, 2018.

Contraction

- The definition of contraction requires a norm; we use the sup-norm here:
$$\|v\| = \sup_s |v(s)|$$
- It can be shown that the set of v with $|v(\cdot)|$ bounded is complete (Appendix B, [1])
  It is a subset of $\mathcal{X}$, denote this set as $B(\mathcal{X})$

- (Assumption 1.2.1, [1]): The DP operator $T_{(\cdot)}$ without π specified is contractive if $\forall \pi$ admissible, $\forall v \in B(\mathcal{X})$ we have $T_\pi v \in B(\mathcal{X})$, and $\exists \alpha \in (0,1)$ such that
$$\|T_\pi v_1 - T_\pi v_2\| \leq \alpha \|v_1 - v_2\| \qquad \forall v_1, v_2 \in B(\mathcal{X}), \forall \pi \text{ admissible}$$

- If $T_{(\cdot)}$ is contractive, then the Bellman operator T is contractive

[1] Bertsekas, Dimitri P. Abstract dynamic programming. Athena Scientific, 2018.

Contraction

- The DP operators corresponding to the γ-discounted cumulative reward problem are contractive with modulus γ, therefore has the strongest theoretical result

- With the undiscounted cumulative reward problem they are not contractive, but when π is restricted to a subset of policies (“proper”), T_π is a contraction
  Such a restriction is called the “stochastic shortest path” problem, will come back later

- The contraction principle: A contraction on a complete metric space has a unique fixed point
- Therefore, define the Bellman equations as $v = T_\pi v$ and $v = Tv$, they have unique solutions

Discounted problems

- Consider maximizing the γ-discounted cumulative reward, with discount factor $\gamma \in (0,1)$
- Assume the reward function R is bounded; therefore, monotone and contractive

"Continuing" in RL terms

The results: (Chap. 2, [1])

- Bellman equations $v = T_\pi v$ and $v = Tv$ have unique solutions, which are V_π and V respectively
- The optimal cost in non-stationary policies can be attained to within an arbitrary accuracy with a stationary policy

  It suffices to consider only the stationary policies
- V_π and V can be obtained by value iteration, starting from $\forall v \in B(\mathcal{X})$:

$$V_\pi = \lim_{k\to\infty} T_\pi^k v$$
$$V = \lim_{k\to\infty} T^k v$$

  The convergence is geometric

[1] Bertsekas, Dimitri P. Abstract dynamic programming. Athena Scientific, 2018.

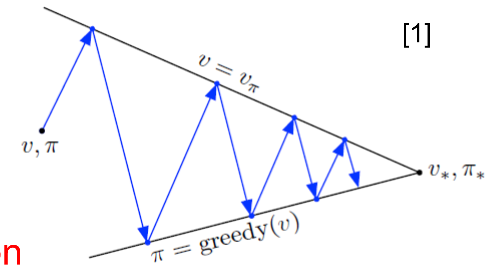
Policy iteration

- In the above problem, V can also be obtained from policy iteration:
- Start with any policy π^0
- In the k^{th} iteration, first do the policy evaluation step that solves $V_{\pi^{k-1}}$ from the linear equation

$$V_{\pi^{k-1}} = T_{\pi^{k-1}} V_{\pi^{k-1}}$$

- Then do the policy improvement step that computes a new policy π^k (argmax instead of max in the Bellman operator)

$$\pi^k(s) = \arg\max_{a \in A(s)} E_{s'}[R(s,a,s') + \gamma V_{\pi^{k-1}}(s')]$$



[1]

- The value function V_{π^k} converges to V as $k \to \infty$
- Value iteration can be seen as a "one step approximation" of policy iteration
- RL is largely inspired by this idea of iteratively moving V and π closer, called Generalized Policy Iteration (GPI); it often works, but not always...

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

Undiscounted problems

- Consider maximizing the undiscounted cumulative reward, with discount factor $\gamma = 1$
- Monotonicity still holds, but only contractive for some policies (semi-contractive)

- Make a restriction: assume one state is the termination state o that

$$P_{oo}(a) = 1 \qquad \forall a \in A(o)$$

“Episodic” in RL terms

  Any self transition loop at o gives zero reward.

- π is proper if

$$\max_{s \in S} P\{s_{|S|} \neq o | s_0 = s, \pi\} < 1$$

  That is, starting with any state, there is a positive probability to reach termination in $|S|$ steps;

  In other words, termination is reached in finite time with probability 1

- In this restricted “SSP” problem, theoretical tools from last page are available

From DP to Approximate DP

- The problem of DP: “curse of dimensionality”

  The number of states grows exponentially with the amount of describing variables

  Consider the state as a vector in $\mathbb{R}^d$, each element is binary; $|S| = 2^d$

  The same is true for discretizing continuous state space

- Three main ideas in Approximate DP:
  - Use function approximation for policy and value function

    The approximator can be global and parametric (DNN) or local (aggregation, kernels, ...)
  - Asynchronous policy iteration
  - Sample path based methods (simulation)

    Motivated by the fact that there are often LOTs of “null states”

The advantages of simulation

- The simulation method when combining other two ideas has the following advantages:
  - Avoid exhaustive sweeps of the state space by focusing on the simulated sample path

    Many states cannot be visited

    Many states have low visitation frequency, therefore possibly less important; in simulation it is updated less often (the idea of asynchronous update)
  - Simplify the DP backup (the evaluation of DP operators) by sampling

    This is essentially the idea of Monte Carlo integration, which simplifies the E step
  - Require less model knowledge than exact DP

    In complicated systems it is hard to explicitly formulate transition dynamics, but it is possible to construct simulators (video games, robots, ...)

Reinforcement Learning

- Approximate DP becomes RL when the model knowledge is further weakened; the boundary is vague
- Need to trade-off exploration vs exploitation: the policy π needs to be stochastic
- We do not know (in terms of probability) the next state when taking an action, therefore value functions $V(s)$ become action-value functions $Q(s, a)$

- In continuing problems,

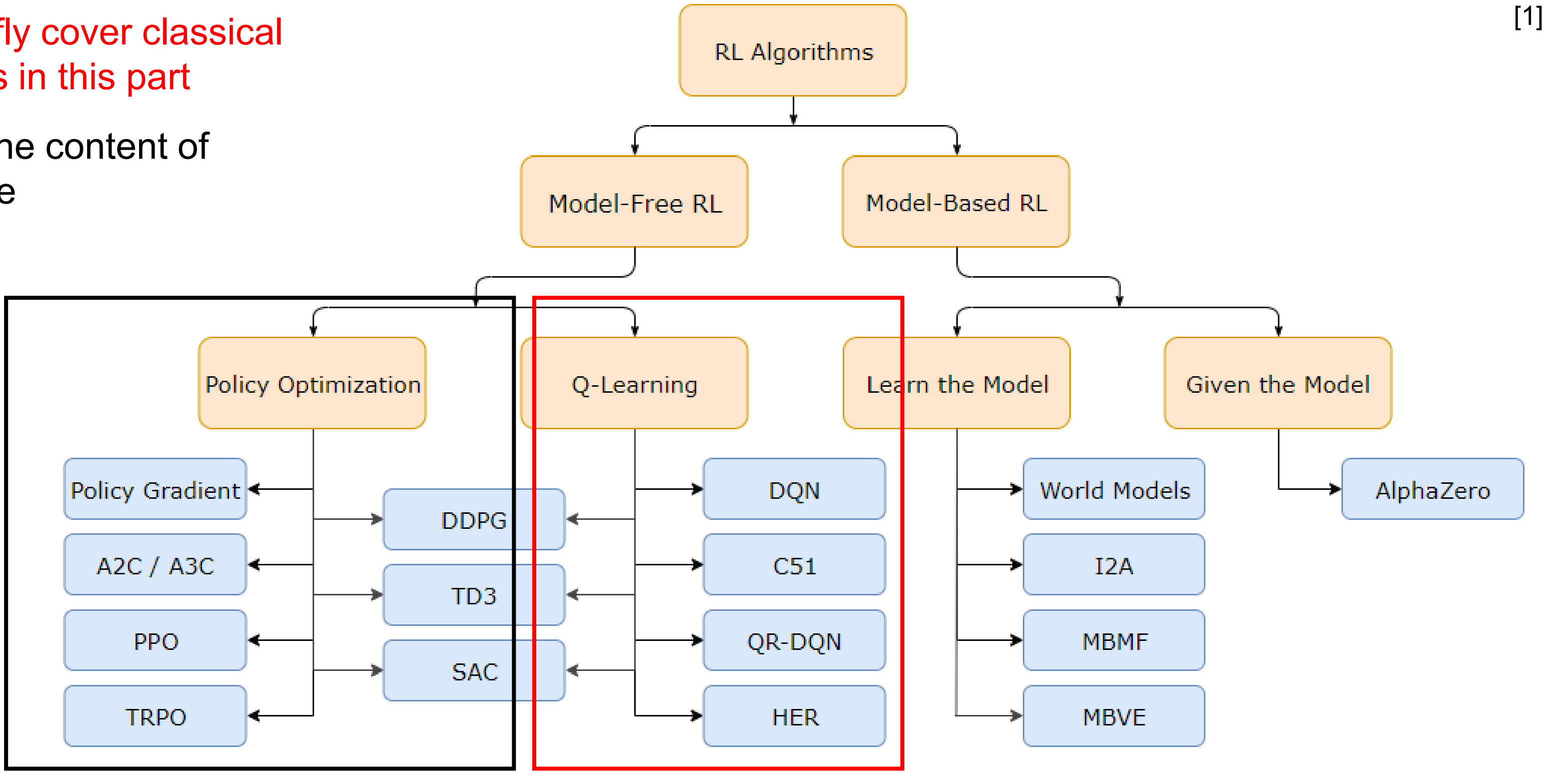
$$Q_\pi(s_0, a) = E_{\tau|\pi, s_0, a}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\,|a_0 = a\right]$$

  The optimal action-value function is its supremum
- In episodic problems it is the same (omitting the “proper” restriction for now)

An incomplete overview of (Deep) RL algorithms

We briefly cover classical methods in this part

This is the content of next time



[1] https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

Monte Carlo Estimation

- Consider the simplest problem: finite, discounted problem with bounded reward, and there is a termination state that can be reached in finite time w.p.1 from any state with any policy

- Monte Carlo estimation is an algorithm that finds $Q_\pi(s,a)$ for a policy π
- The idea is a direct implementation of the definition

$$Q_\pi(s_0,a) = E_{\tau|\pi,s_0,a}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t,s_{t+1})\,|a_0 = a\right]$$

- In each episode, we rollout π till termination
- Use the sample path to generate a stochastic approximation of the component inside the bracket, for all state action pairs encountered along this sample path; store it in the buffer
- Between episodes, use the content of the buffer to evaluate the E step and generate an estimate $\hat{Q}_\pi$; it provably converges to Q_π, no stepsizes

Monte Carlo Control

- We want to optimize π while estimating Q_π, hopefully they converge to π^* and Q
- Between episodes, with a newly estimated Q_π, the best policy would be greedy w.r.t. Q_π
- This is a bad idea since it does not explore...

- Two possible solutions:
  - Exploration start: make the initial state distribution $\rho_0(s) > 0, \forall s \in S$

    Works but no theoretical guarantee...
  - Make π approximately greedy: greedy with a large probability; explore with a small probability

    Examples: ϵ-greedy, Boltzmann exploration, ...

    For ϵ-greedy, it provably converges to the best ϵ-greedy policy, but not the optimal in general

On policy vs. Off policy

- An important concept in RL that is vaguely defined as follows:
- When we rollout π and learn from the sample path, our algorithm is "on policy" when we estimate Q_π and generate control; it is "off policy" when we estimate $Q_{\pi'}$ (corresponding to another policy) and generate control
- π' is often chosen as the optimal policy π^*

- The general scheme of Off Policy Monte Carlo: importance sampling

  Check Sec. 5.5 & 5.7 in [1], or Monte Carlo statistics

- Qualitatively, off policy methods nowadays has better sample complexity but less stable (converges in tabular case but may diverge with function approximation)

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

Temporal Difference

- Monte Carlo methods are intuitive, but they only update between episodes
- In other words, they do not “bootstrap”, i.e. generate Q_π estimates based on older estimate

- TD learning is a method that bootstraps; the idea is to “cut-off” Monte Carlo rollouts and replace the remainder with the current Q_π estimate

- The “cut-off” length is a hyperparameter in TD algorithms
  - Denote one-step cut-off algorithm as TD(0); denote no cut-off as TD(1), i.e. Monte Carlo
  - On this spectrum is TD(λ) where $\lambda \in [0,1]$; See Chap. 12 in [1]

- Empirical evaluation shows that a modest bootstrapping has the best performance

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.

One step On-policy TD control: SARSA

- Start with any action-value estimate $\hat{Q}$
- After each step, with the current policy π, observe the state-action-state (s_t, a_t, s_{t+1}) sample; receive the reward r_{t+1}, and sample the next action a_{t+1} using π
- Use this SARSA sample $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ to update $\hat{Q}$
$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha_t[r_{t+1} + \gamma\hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t)]$$
  where α_t is the step size
- Update π to be approximately greedy
$$\pi \leftarrow \text{aprox\_greedy}(\hat{Q})$$
- Convergence to optimal policy is guaranteed w.p.1 by the following two conditions:
  - GLIE (Greedy in the Limit with Infinite Exploration) policy: every action is chosen infinitely often at states that are visited infinitely often; the policy has a greedy policy as its limit
  - Robbins-Monro step sizes: $\sum_{t=1}^{\infty} a_t = \infty$, $\sum_{t=1}^{\infty} a_t^2 < \infty$

One step Off-policy TD control: Q learning

- Start with any action-value estimate $\hat{Q}$, exploratory rollout policy π
- After each step, observe the state-action-state (s_t, a_t, s_{t+1}) sample; receive the reward r_{t+1}
- Update $\hat{Q}$

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha_t[r_{t+1} + \gamma \max_{a \in A(s_{t+1})} \hat{Q}(s_{t+1}, a) - \hat{Q}(s_t, a_t)]$$

  where α_t is the step size

- Although we sample with π, we estimate the optimal action-value function directly
- Convergence to optimal Q is guaranteed w.p.1 by the following two conditions:
  - All admissible state-action pairs are updated infinitely often
  - Robbins-Monro step sizes
- A possible paper for discussion: Q learning with UCB exploration achieves near-optimal regret

[1] Jin, Chi, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I. Jordan. "Is q-learning provably efficient?." In Advances in Neural Information Processing Systems, pp. 4863-4873. 2018.

Q learning with function approximation: The “deadly triad”

- The trend of Q learning with DNN action value approximation started several years ago and initated the field of “Deep RL”; the technique itself dates back to the 90’s
- The “deadly triad” is a concept formulated by Sutton & Barto that illustrate the difficulty of “Deep Q learning” algorithms:
- Whenever an RL algorithm has all the following three components, it is problematic:

  Function approximation, Bootstrapping, Off-Policy training
- Moreover, the loss function for the DNN component is not clear: the popular heuristic “Bellman loss” is potentially also problematic... See Chap. 11 in [1]
- This triad is so “deadly” that people gradually shifted their attention to policy optimization methods in the past few years. A lot needs to be done...

[1] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018.