L7: Framework of Reinforcement Learning (II)

Hao Su

(slides prepared with the help from Tongzhou Mu)

Winter, 2023

Contents are based on Reinforcement Learning: An Introduction from Prof. Richard S. Sutton and Prof. Andrew G. Barto, and COMPM050/COMPGI13 taught at UCL by Prof. David Silver.

Agenda

- Optimal Policy and Optimal Value Function
- Estimating Value Function for a Given Policy
- Q-Learning for Tabular RL

click to jump to the section.

Optimal Policy and Optimal Value Function

Optimal Value Function

- Due to the Markovian property, the return starting from a state s is independent of its history. Therefore, we can compare the return of all policies starting from s and find the optimal one.
- The optimal state-value function $V^*(s)$ is the maximum value function over all policies
  - $V^*(s) = \max_\pi V_\pi(s)$
- The optimal action-value function $Q_*(s,a)$ is the maximum action-value function over all policies
  - $Q^*(s,a) = \max_\pi Q_\pi(s,a)$
- The optimal value function specifies the best possible performance in the MDP.

Optimal Policy

- Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } V_{\pi}(s) \geq V_{\pi'}(s), \forall s$$

  Theorem: For any Markov Decision Process
  - *There exists an optimal policy* π_* *that is better than, or equal to, all other policies,* $\pi_* \geq \pi, \ \forall \pi$
  - *All optimal policies achieve the optimal value function,* $V_{\pi^*}(s) = V^*(s)$
  - *All optimal policies achieve the optimal action-value function,* $Q_{\pi^*}(s,a) = Q^*(s,a)$
- An optimal policy can be found by maximizing over $Q^*(s,a)$,

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } \ a = \text{argmax}_{a \in \mathcal{A}} \, Q^*(s,a) \\ 0, & \text{otherwise} \end{cases}$$

Bellman Optimality Equation

- Optimal value functions also satisfy recursive relationships

$$\begin{aligned} V^*(s) &= \max_a \mathbb{E}_{\pi_*}[G_t|S_t = s, A_t = a] \\ &= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1})|S_t = s, A_t = a] \end{aligned}$$

- Similarly, for action-value function, we have

$$Q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a')|S_t = s, A_t = a]$$

- They are called **Bellman Optimality Equations**

Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear (because there is the max operation).
- No closed form solution (in general)

- Many iterative solution methods:
  - Value Iteration
  - Policy Iteration
  - Q-learning (we will talk about this later)
  - SARSA

Estimating Value Function for a Given Policy

Goal: Given a policy $\pi(a|s)$, estimate the value of the policy.

Monte-Carlo Policy Evaluation

- Basic idea: MC uses the simplest possible idea: **value = mean return**

Monte-Carlo Policy Evaluation

- Basic idea: MC uses the simplest possible idea: **value = mean return**
- Learn V_π from K episodes under policy π
  - $\{S_{k,0}, A_{k,0}, R_{k,1}, \ldots, S_{k,T}\}_{k=1}^{K} \sim \pi$
- Recall that the *return* is the total discounted reward:
  - $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T$
- Recall that the value function is the expected return:
  - $V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$

Monte-Carlo Policy Evaluation

- Suppose that we have collected a number of trajectories. Monte-Carlo policy evaluation uses *empirical mean return* to approximate *expected return*
  - For each episode:
    - For each time step t:
      - Compute empirical return G_t from the current state s
      - Increment total return $S(s) \leftarrow S(s) + G_t$
      - Increment state visit counter $N(s) \leftarrow N(s) + 1$
  - Value is estimated by mean return $V(s) = S(s)/N(s)$

Monte-Carlo Methods

- Quick facts:
  - MC is unbiased (average of the empirical return is the true return)
  - MC methods learn directly from episodes of experience
  - MC is model-free: no knowledge of MDP transitions / rewards
- Caveat: can only apply MC to episodic MDPs
  - All episodes must terminate

Temporal-Difference Learning

- Basic idea: TD leverages **Bellman expectation equation** to update the value function.

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]$$

Temporal-Difference Learning

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]$$

- Learn V_π from K episodes under policy π
  - $\{S_{k,0}, A_{k,0}, R_{k,1}, \ldots, S_{k,T}\}_{k=1}^{K} \sim \pi$
- Simplest temporal-difference learning algorithm: TD(0)
  - Loop for a new iterations:
    - Sample $(S_t, A_t, R_{t+1}, S_{t+1})$ with replacement from the transitions in the episodes

    - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$:
      - $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

Temporal-Difference Learning

$$V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s]$$

- Learn V_π from K episodes under policy π
  - $\{S_{k,0}, A_{k,0}, R_{k,1}, \ldots, S_{k,T}\}_{k=1}^{K} \sim \pi$
- Simplest temporal-difference learning algorithm: TD(0)
  - Loop for a new iterations:
    - Sample $(S_t, A_t, R_{t+1}, S_{t+1})$ with replacement from the transitions in the episodes
    - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$:
      - $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*
- If we expand one step further, we got TD(1)
  - $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma R_{t+1} + \gamma^2 V(S_{t+2}) - V(S_t))$
  - Similarly, we can have TD(2), TD(3), ...

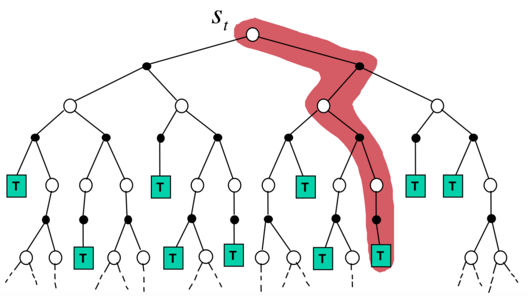
Temporal-Difference Learning

- Quick facts:
  - TD methods learn directly from episodes of experience
  - TD is model-free: no knowledge of MDP transitions / rewards
  - TD learns from incomplete episodes, by **bootstrapping**
  - TD updates a guess towards a guess

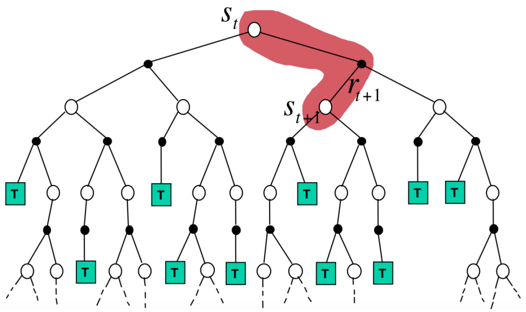
Key Differences between MC and TD

- MC estimates values based on *rollout*
- TD estimates values based on Bellman equation

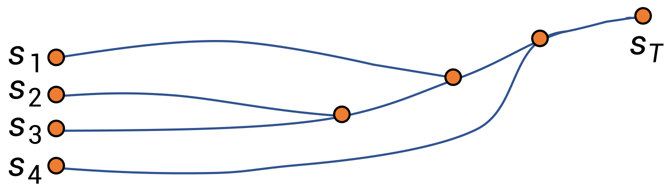
$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

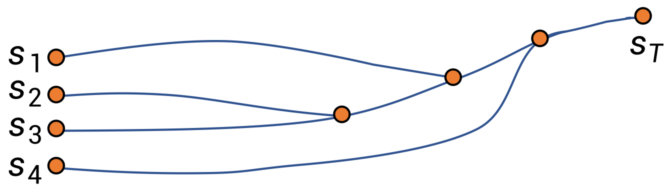


Pros and Cons of MC vs. TD



- TD can learn *before* knowing the final outcome
  - MC must wait until the end of episodes
  - TD can learn online after every step

Pros and Cons of MC vs. TD



- TD can learn *without* the final outcome
  - MC can only learn from complete sequences
  - When some episodes are incomplete, TD can still learn
  - MC only works for episodic (terminating) environments
  - TD works in non-terminating environments

Bias/Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T$ is always an unbiased estimate of $V_\pi(S_t)$
- The true TD target $R_{t+1} + \gamma V_\pi(S_{t+1})$ is an unbiased estimate of $V_\pi(S_t)$
- If we will **update** π slowly along the learning process,

Bias/Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T$ is always an unbiased estimate of $V_\pi(S_t)$
- The true TD target $R_{t+1} + \gamma V_\pi(S_{t+1})$ is an unbiased estimate of $V_\pi(S_t)$
- If we will **update** π slowly along the learning process,
  - the TD target $R_{t+1} + \gamma V(S_{t+1})$ becomes a *biased estimate* of $V_\pi(S_t)$, because
    - $V(S_{t+1})$ is a return estimation from the previous π.

Bias/Variance Trade-Off

- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T$ is always an unbiased estimate of $V_\pi(S_t)$
- The true TD target $R_{t+1} + \gamma V_\pi(S_{t+1})$ is an unbiased estimate of $V_\pi(S_t)$
- If we will **update** π slowly along the learning process,
  - the TD target $R_{t+1} + \gamma V(S_{t+1})$ becomes a *biased estimate* of $V_\pi(S_t)$, because
    - $V(S_{t+1})$ is a return estimation from the previous π.
  - However, the TD target also has *much lower variance* than the return G_t, because
    - the return G_t is from a single rollout, heavily affected by the randomness (actions, transitions, and rewards) in all the future steps;
    - whereas the TD target is affected by the randomness in the *next one step*, and the low variance of the $V(S_{t+1})$ estimation from many historical rollouts.

Pros and Cons of MC vs TD (2)

- MC has high variance, zero bias
  - Good convergence properties (even with function approximation)
  - Independent of the initial value of V
  - Very simple to understand and use
- TD has low variance, some bias
  - Usually more efficient than MC
  - TD(0) converges to $V_\pi(s)$ (but not always with function approximation)
  - Has certain dependency on the initial value of V

Bellman, or no Bellman, that is the question

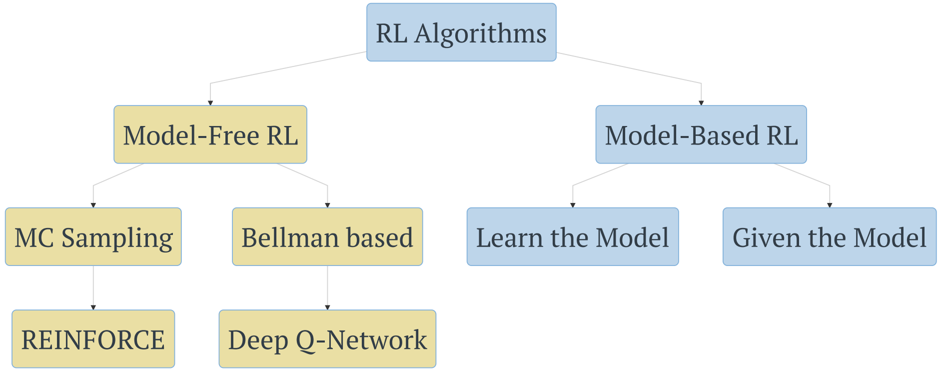
- *Monte-Carlo sampling* and *Bellman equation* are two fundamental tools of value estimation for policies.
- Each has its pros and cons.

- Based on them, there are two families of model-free RL algorithms, both well developed. Some algorithms leverage both.
- Fundamentally, it is about the balance between bias and variance (sample complexity).

We start from REINFORCE and Deep Q-Learning

- Reason I:
    - REINFORCE uses Monte-Carlo sampling
    - Deep Q-Learning (DQN) uses Bellman equation
- Reason II:
    - REINFORCE only has a **policy network**
    - DQN only has a **value network**

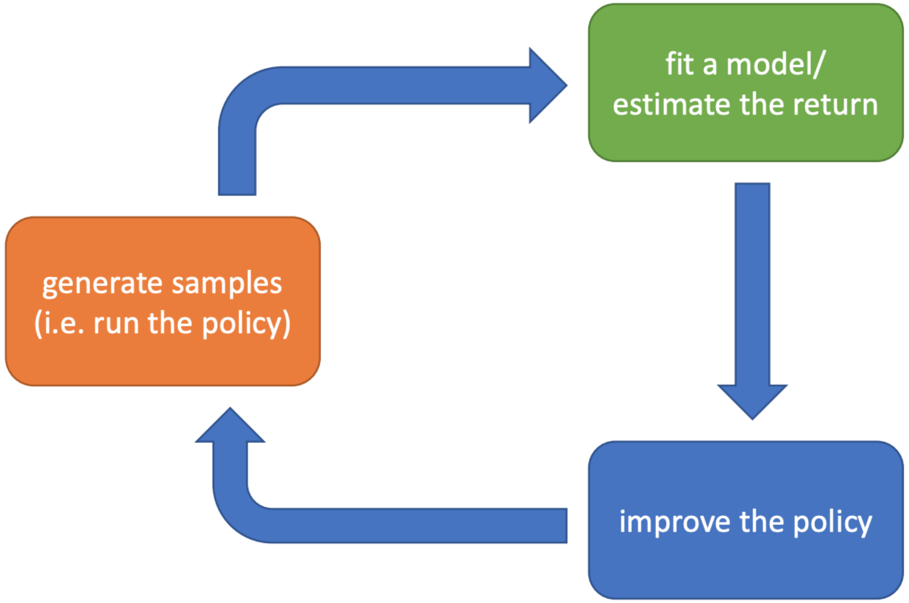
Taxonomy of RL Algorithms and Examples



Q-Learning for Tabular RL

Tabular RL: RL with discrete and finite state space, convenient for algorithm development and convergence analysis

The Anatomy of an RL algorithm



this often uses Q-functions or value functions

CS285 taught at UC Berkeley by Prof. Sergey Levine.

- Suppose we are going to learn the Q-function. Let us follow the previous flow chart and answer three questions:
  1. Given transitions $\{(s, a, s', r)\}$ from some trajectories, how to improve the current Q-function?
  2. Given Q, how to improve policy?
  3. Given π, how to generate trajectories?

- Suppose we are going to learn the Q-function. Let us follow the previous flow chart and answer three questions:

1. Given transitions $\{(s, a, s', r)\}$ from some trajectories, how to improve the current Q-function?
   - By Temporal Difference learning, the update target for $Q(S, A)$ is
     - $R + \gamma \max_a Q(S', a)$
   - Take a small step towards the target
     - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
2. Given Q, how to improve policy?

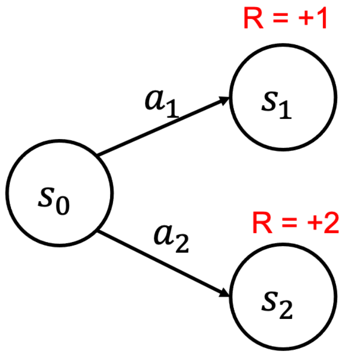
3. Given π, how to generate trajectories?

- Suppose we are going to learn the Q-function. Let us follow the previous flow chart and answer three questions:
  1. Given transitions $\{(s, a, s', r)\}$ from some trajectories, how to improve the current Q-function?
     - By Temporal Difference learning, the update target for $Q(S, A)$ is
       - $R + \gamma \max_a Q(S', a)$
     - Take a small step towards the target
       - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
  2. Given Q, how to improve policy?
     - Take the greedy policy based on the current Q
       - $\pi(s) = \text{argmax}_a Q(s, a)$
  3. Given π, how to generate trajectories?

- Suppose we are going to learn the Q-function. Let us follow the previous flow chart and answer three questions:
  1. Given transitions $\{(s, a, s', r)\}$ from some trajectories, how to improve the current Q-function?
     - By Temporal Difference learning, the update target for $Q(S, A)$ is
       - $R + \gamma \max_a Q(S', a)$
     - Take a small step towards the target
       - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
  2. Given Q, how to improve policy?
     - Take the greedy policy based on the current Q
       - $\pi(s) = \mathrm{argmax}_a Q(s, a)$
  3. Given π, how to generate trajectories?
     - Simply run the greedy policy in the environment.
     - Any issues?

Failure Example

- Initialize Q
  - $Q(s_0, a_1) = 0, Q(s_0, a_2) = 0$
  - $\pi(s_0) = a_1$
- Iteration 1: take a_1 and update Q
  - $Q(s_0, a_1) = 1, Q(s_0, a_2) = 0$
  - $\pi(s_0) = a_1$
- Iteration 2: take a_1 and update Q
  - $Q(s_0, a_1) = 1, Q(s_0, a_2) = 0$
  - $\pi(s_0) = a_1$
- ...
- **Q stops to improve because the agent is too greedy!**



ϵ-Greedy Exploration

- The simplest and most effective idea for ensuring continual exploration
- With probability $1-\epsilon$ choose the greedy action
- With probability ϵ choose an action at random
- All m actions should be tried with non-zero probability
- Formally,

$$\pi_*(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, & \text{if } \ a = \text{argmax}_{a\in\mathcal{A}} Q(s,a) \\ \epsilon/m, & \text{otherwise} \end{cases}$$

Exploration vs Exploitation

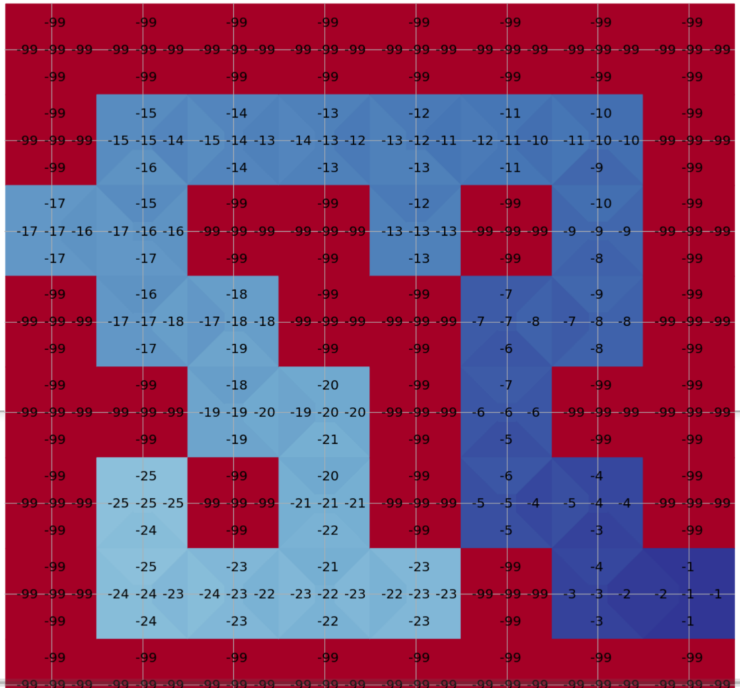
- Exploration
  - finds more information about the environment
  - may waste some time
- Exploitation
  - exploits known information to maximize reward
  - may miss potential better policy
- Balancing exploration and exploitation is a key problem of RL. We use will spend a lecture to discuss advanced exploration strategies.

Q-Learning Algorithm

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\mathit{terminal}, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal



-99 -99 -99 -99 -99 -99 -99 -99
-99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 -99
-99 -99 -99 -99 -99 -99 -99 -99
-99 -15 -14 -13 -12 -11 -10 -99
-99 -99 -99 -15 -15 -14 -15 -14 -13 -14 -13 -12 -13 -12 -11 -12 -11 -10 -11 -10 -10 -99 -99 -99
-99 -16 -14 -13 -13 -11 -9 -99
-17 -15 -99 -99 -12 -99 -10 -99
-17 -17 -16 -17 -16 -16 -99 -99 -99 -99 -99 -99 -13 -13 -13 -99 -99 -99 -9 -9 -9 -99 -99 -99
-17 -17 -99 -99 -13 -99 -8 -99
-99 -16 -18 -99 -99 -7 -9 -99
-99 -99 -99 -17 -17 -18 -17 -18 -18 -99 -99 -99 -99 -99 -99 -7 -7 -8 -7 -8 -8 -99 -99 -99
-99 -17 -19 -99 -99 -6 -8 -99
-99 -99 -18 -20 -99 -7 -99 -99
-99 -99 -99 -99 -99 -99 -19 -19 -20 -19 -20 -20 -99 -99 -99 -6 -6 -6 -99 -99 -99 -99 -99 -99
-99 -99 -19 -21 -99 -5 -99 -99
-99 -25 -99 -20 -99 -6 -4 -99
-99 -99 -99 -25 -25 -25 -99 -99 -99 -21 -21 -21 -99 -99 -99 -5 -5 -4 -5 -4 -4 -99 -99 -99
-99 -24 -99 -22 -99 -5 -3 -99
-99 -25 -23 -21 -23 -99 -4 -1
-99 -99 -99 -24 -24 -23 -24 -23 -22 -23 -22 -23 -22 -23 -23 -99 -99 -99 -3 -3 -2 -2 -1 -1
-99 -24 -23 -22 -23 -99 -3 -1
-99 -99 -99 -99 -99 -99 -99 -99

31/32

< step-31 >

End