Lecture 7: Review on Tabular RL

Bolei Zhou

The Chinese University of Hong Kong

bzhou@ie.cuhk.edu.hk

February 18, 2020

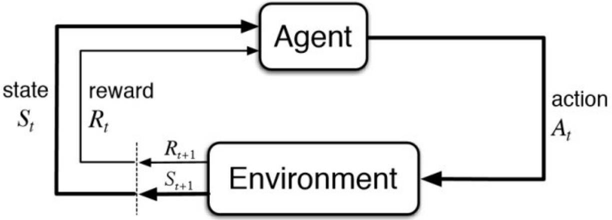
Announcement

1. We continue with ZOOM Online Teaching
   1. 20-30 mins per session
   2. Try to open your webcam
   3. Take note
2. HW1 and project proposal due by the end of Feb.19, please email to cuhkrlcourse@googlegroups.com

Today's Content: Review Session

1. RL Basics
2. MDP prediction and control
   1. Policy evaluation
   2. Control: Policy iteration and value iteration
3. Model-free prediction and control

Reinforcement Learning



1. a computational approach to learning whereby an agent tries to maximize the total amount of reward it receives while interacting with a complex and uncertain environment.
- Sutton and Barto

Difference between Reinforcement Learning and Supervised Learning

1. Sequential data as input (not i.i.d)
2. Trial-and-error exploration (balance between exploration and exploitation)
3. There is no supervisor, only a reward signal, which is also delayed
4. Agent's actions affect the subsequent data it receives (agent's action changes the environment)

Major Components of RL

1. An RL agent may include one or more of these components
   1. Policy: agent's behavior function
   2. Value function: how good is each state or action
   3. Model: agent's state representation of the environment

Define the model of environment

1. Markov Processes
2. Markov Reward Processes(MRPs)
3. Markov Decision Processes (MDPs)

Markov Process

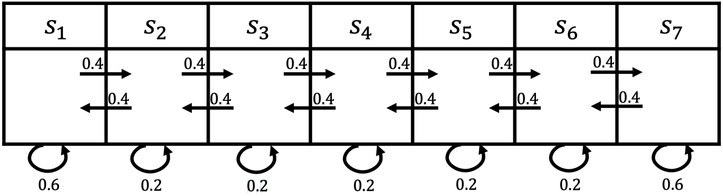
1. P is the dynamics/transition model that specifies $p(s_{t+1} = s'|s_t = s)$
2. State transition matrix: P(To|From)

$$P = \begin{bmatrix} P(s_1|s_1) & P(s_2|s_1) & \dots & P(s_N|s_1) \\ P(s_1|s_2) & P(s_2|s_2) & \dots & P(s_N|s_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(s_1|s_N) & P(s_2|s_N) & \dots & P(s_N|s_N) \end{bmatrix}$$

Markov Reward Process (MRP)

1. Markov Reward Process is a Markov Chain + reward
2. Definition of Markov Reward Process (MRP)
   1. S is a (finite) set of states ($s \in S$)
   2. P is dynamics/transition model that specifies $P(S_{t+1} = s' | s_t = s)$
   3. R is a reward function $R(s_t = s) = \mathbb{E}[r_t | s_t = s]$
   4. Discount factor $\gamma \in [0, 1]$
3. If finite number of states, R can be a vector

MRP Example



Reward: $+5$ in s_1, $+10$ in s_7, 0 in all other states. So that we can represent $R = [5, 0, 0, 0, 0, 0, 10]$

Return Function and Value Function

1. Return: Discounted sum of rewards from time step t to horizon

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... + \gamma^{T-t-1} R_T$$

2. Value function $V_t(s)$: Expected return from t in state s

$$\begin{aligned} V_t(s) =& \mathbb{E}[G_t | s_t = s] \\ =& \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-t-1} R_T | s_t = s] \end{aligned}$$

Computing the Value of a Markov Reward Process

1. Value function: expected return from starting in state s

$$V(s) = \mathbb{E}[G_t | s_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | s_t = s]$$

2. MRP value function satisfies the following Bellman equation:

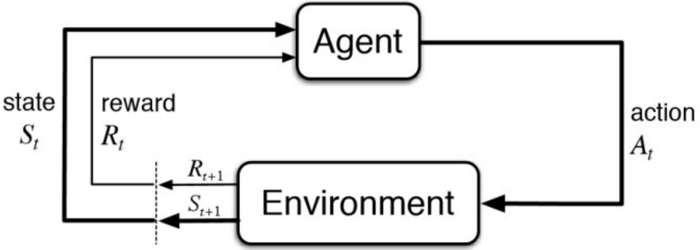
$$V(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P(s'|s) V(s')}_{\text{Discounted sum of future reward}}$$

Iterative Algorithm for Computing Value of a MRP

Algorithm 1 Iterative algorithm to calculate MRP value function

for all states $s \in S, V'(s) \leftarrow 0, V(s) \leftarrow \infty$
while $||V - V'|| > \epsilon$ **do**
    $V \leftarrow V'$
    For all states $s \in S, V'(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s) V(s')$
end while
return $V'(s)$ for all $s \in S$

Markov Decision Process (MDP)



1. Markov Decision Process describes the framework of reinforcement learning. MDP is a tuple: (S, A, P, R, γ)
   1. S is a finite set of states, A is a finite set of actions
   2. P^a is dynamics/transition model for each action $P(s_{t+1} = s' | s_t = s, a_t = a)$
   3. R is a reward function $R(s_t = s, a_t = a) = \mathbb{E}[r_t | s_t = s, a_t = a]$
   4. Discount factor $\gamma \in [0, 1]$

Return Function and Value Function

1. Definition of Horizon
   1. Number of maximum time steps in each episode
   2. Can be infinite, otherwise called finite Markov (reward) Process
2. Definition of Return
   1. Discounted sum of rewards from time step t to horizon

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... + \gamma^{T-t-1} R_T$$

3. Definition of state value function $V_t(s)$ for a MRP
   1. Expected return from t in state s

$$\begin{aligned} V_t(s) =& \mathbb{E}[G_t | s_t = s] \\ =& \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-t-1} R_T | s_t = s] \end{aligned}$$

   2. Present value of future rewards

Value function for MDP

1. The state-value function $v_\pi(s)$ of an MDP is the expected return starting from state s, and following policy π

$$v_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s] \tag{1}$$

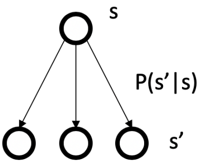
2. The action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

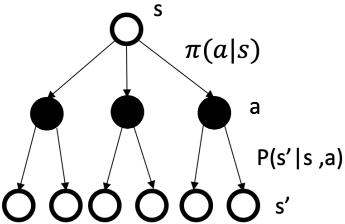
$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, A_t = a] \tag{2}$$

3. We have the relation between $v_\pi(s)$ and $Q_\pi(s, a)$

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) Q_\pi(s, a) \tag{3}$$

Comparison of Markov Process and MDP





Bellman Expectation Equation

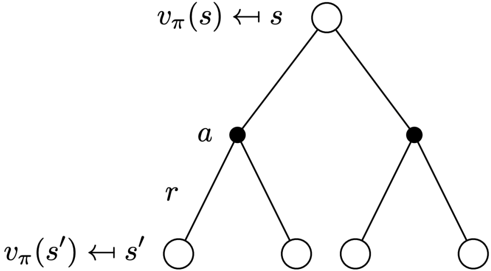
1. The state-value function can be decomposed into immediate reward plus discounted value of the successor state,

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s] \tag{4}$$

2. The action-value function can similarly be decomposed

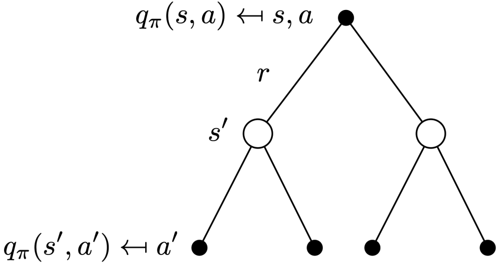
$$Q_\pi(s, a) = E_\pi[R_{t+1} + \gamma Q_\pi(s_{t+1}, A_{t+1}) | s_t = s, A_t = a] \tag{5}$$

Backup Diagram for V^π



$$v_\pi(s) = \sum_{a \in A} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_\pi(s')) \tag{6}$$

Backup Diagram for Q^π



$$q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \sum_{a' \in A} \pi(a'|s') q_\pi(s',a') \quad (7)$$

Decision Making in MDP

1. Prediction: evaluate a given policy
    1. Input: MDP $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma>$ and policy π
    2. Output: value function v_π
2. Control: search the optimal policy
    1. Input: MDP $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma>$
    2. Output: optimal value function v_* and optimal policy π_*
3. Prediction and control can be solved by dynamic programming.

Policy evaluation: Iteration on Bellman expectation backup

Bellman expectation backup for a particular policy
Bootstrapping: estimate on top of estimates

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_k(s')) \tag{8}$$

1. Synchronous backup algorithm:
    1. At each iteration k+1
       update $v_{k+1}(s)$ from $v_k(s')$ for all states $s \in \mathcal{S}$ where s' is a successor state of s

$$v_{k+1}(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_k(s')) \tag{9}$$

2. Convergence: $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_\pi$

MDP Control

1. Compute the optimal policy

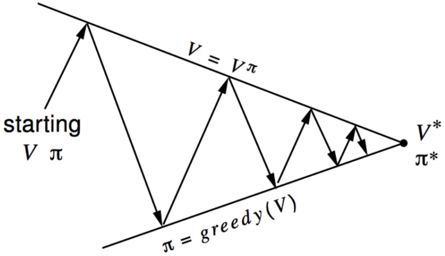
$$\pi_*(s) = \arg\max_{\pi} v_{\pi}(s) \tag{10}$$

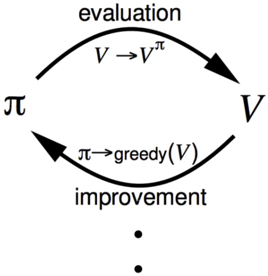
2. Policy iteration and value iteration

Improve a Policy through Policy Iteration

1. Iterate through the two steps:
   1. Evaluate the policy π (computing v given current π)
   2. Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi) \tag{11}$$





Policy Improvement

1. compute the state-action value of a policy π:

$$q_{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{\pi_i}(s') \tag{12}$$

2. Compute a new greedy policy π_{i+1} for all $s \in \mathcal{S}$ following

$$\pi_{i+1}(s) = \arg\max_a q_{\pi_i}(s,a) \tag{13}$$

3. We can prove that Monotonic Improvement in Policy.

Monotonic Improvement and Bellman optimality equation

1. If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

2. Thus the Bellman optimality equation has been satisfied

$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

3. Then we have the following Bellman optimality backup:

$$v_*(s) = \max_a (R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) v_*(s'))$$

Value iteration by turning the Bellman optimality equation as update rule

1. If we know the solution to subproblem $v_*(s')$, which is optimal.
2. Then the solution for the optimal $v_*(s)$ can be found by iteration over the following Bellman Optimality backup rule,

$$v(s) \leftarrow \max_{a \in \mathcal{A}} \Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v(s') \Big)$$

3. The idea of value iteration is to apply these updates iteratively

Summary for Prediction and Control in MDP

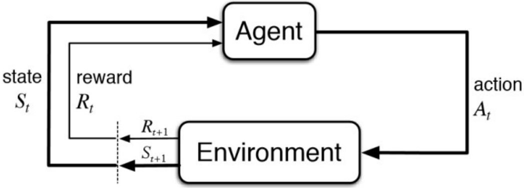
Table: Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

Model-free RL

1. Policy evaluation, policy iteration and value iteration for solving a known MDP
2. Model-free prediction: Estimate value function of an unknown MDP
3. Model-free control: Optimize value function of an unknown MDP

Model-free RL: prediction and optimal control by interaction



1. No more oracle or shortcut of the known transition dynamics and reward function
2. Trajectories/episodes are collected by the interaction of the agent and the environment
3. Each trajectory/episode contains $\{S_1, A_1, R_1, S_2, A_2, R_2, ..., S_T, A_T, R_T\}$

Model-free prediction: policy evaluation without the access to the model

1. Estimating the expected return of a particular policy if we don't have access to the MDP models
   1. Monte Carlo policy evaluation
   2. Temporal Difference (TD) learning

Monte-Carlo Policy Evaluation

1. To evaluate state $v(s)$
   1. Every time-step t that state s is visited in an episode,
   2. Increment counter $N(s) \leftarrow N(s) + 1$
   3. Increment total return $S(s) \leftarrow S(s) + G_t$
   4. Value is estimated by mean return $v(s) = S(s)/N(s)$
2. By law of large numbers, $v(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Incremental MC Updates

1. Collect one episode $(S_1, A_1, R_1, ..., S_t)$
2. For each state s_t with computed return G_t

$$
\begin{aligned}
N(S_t) &\leftarrow N(S_t) + 1 \\
v(S_t) &\leftarrow v(S_t) + \frac{1}{N(S_t)}(G_t - v(S_t))
\end{aligned}
$$

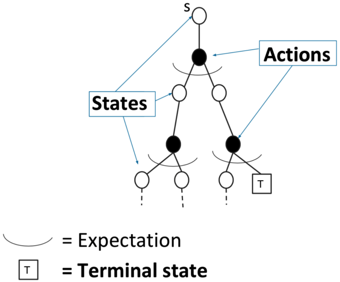
3. Or use a running mean (old episodes are forgotten). Good for non-stationary problems.

$$
v(S_t) \leftarrow v(S_t) + \alpha(G_t - v(S_t))
$$

Difference between DP and MC for policy evaluation

1. DP computes v_i by bootstrapping the rest of the expected return by the value estimate v_{i-1}
2. Iteration on Bellman expectation backup:

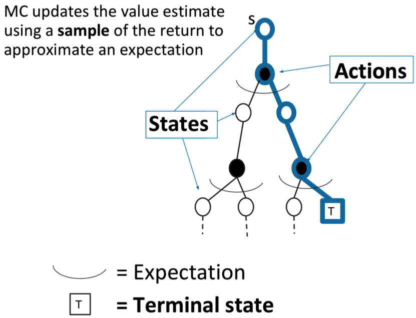
$$v_i(s) \leftarrow \sum_{a \in \mathcal{A}} \pi(a|s)\Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_{i-1}(s')\Big)$$



Difference between DP and MC for policy evaluation

1. MC updates the empirical mean return with one sampled episode

$$v(S_t) \leftarrow v(S_t) + \alpha(G_{i,t} - v(S_t))$$



Advantages of MC over DP

1. MC works when the environment is unknown
2. Working with sample episodes has a huge advantage, even when one has complete knowledge of the environment's dynamics, for example, transition probability is complex to compute
3. Cost of estimating a single state's value is independent of the total number of states. So you can sample episodes starting from the states of interest then average returns

Temporal-Difference (TD) Learning

1. TD methods learn directly from episodes of experience
2. TD is model-free: no knowledge of MDP transitions/rewards
3. TD learns from incomplete episodes, by bootstrapping

Temporal-Difference (TD) Learning

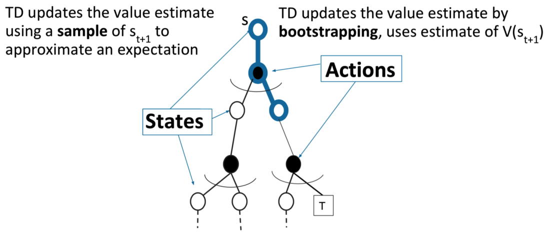
1. Objective: learn v_π online from experience under policy π
2. Simplest TD algorithm: TD(0)
   1. Update $v(S_t)$ toward estimated return $R_{t+1} + \gamma v(S_{t+1})$

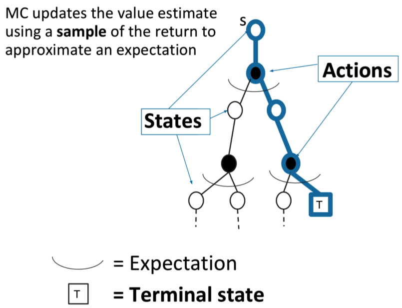
$$v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(S_{t+1}) - v(S_t))$$

3. $R_{t+1} + \gamma v(S_{t+1})$ is called TD target
4. $\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ is called the TD error
5. Comparison: Incremental Monte-Carlo
   1. Update $v(S_t)$ toward actual return G_t given an episode i

$$v(S_t) \leftarrow v(S_t) + \alpha(G_{i,t} - v(S_t))$$

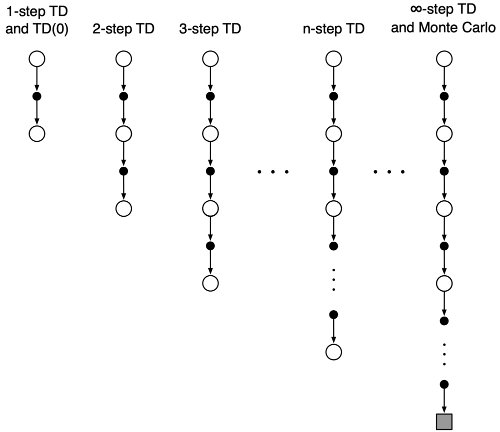
Advantages of TD over MC





n-step TD

1. n-step TD methods that generalize both one-step TD and MC.
2. We can shift from one to the other smoothly as needed to meet the demands of a particular task.



n-step TD prediction

1. Consider the following n-step returns for $n = 1, 2, \infty$

$$\begin{aligned} n = 1(TD) \quad G_t^{(1)} =& R_{t+1} + \gamma v(S_{t+1}) \\ n = 2 \qquad\quad G_t^{(2)} =& R_{t+1} + \gamma R_{t+2} + \gamma^2 v(S_{t+2}) \\ & \vdots \\ n = \infty(MC) \quad G_t^{\infty} =& R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T \end{aligned}$$

2. Thus the n-step return is defined as

$$G_t^n = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v(S_{t+n})$$

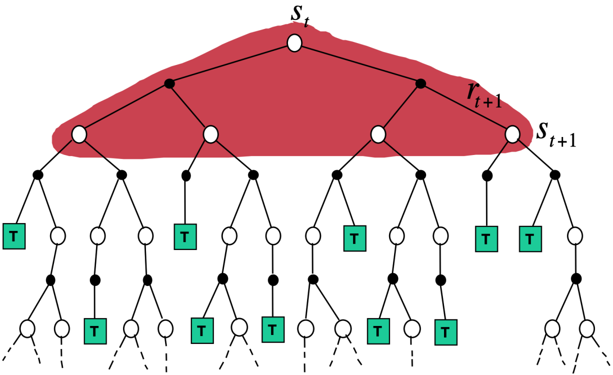
3. n-step TD: $v(S_t) \leftarrow v(S_t) + \alpha\Big(G_t^n - v(S_t)\Big)$

Bootstrapping and Sampling for DP, MC, and TD

1. Bootstrapping: update involves an estimate
    1. MC does not bootstrap
    2. DP bootstraps
    3. TD bootstraps
2. Sampling: update samples an expectation
    1. MC samples
    2. DP does not sample
    3. TD samples

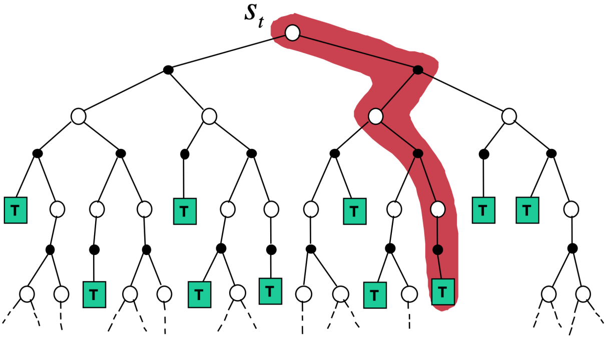
Unified View: Dynamic Programming Backup

$$v(S_t) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma v(S_{t+1})]$$



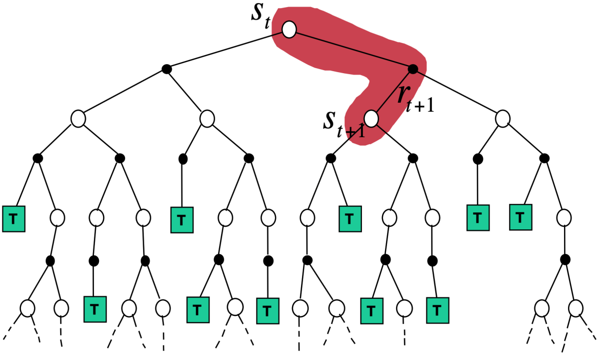
Unified View: Monte-Carlo Backup

$$v(S_t) \leftarrow v(S_t) + \alpha(G_t - v(S_t))$$

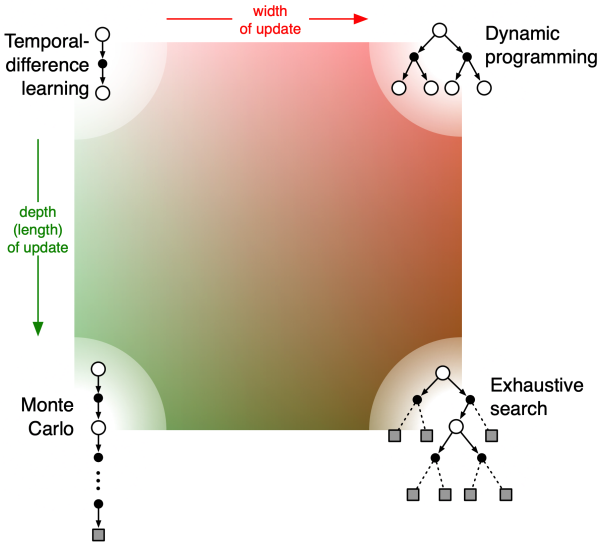


Unified View: Temporal-Difference Backup

$$TD(0) : v(S_t) \leftarrow v(S_t) + \alpha(R_{t+1} + \gamma v(s_{t+1}) - v(S_t))$$



Unified View of Reinforcement Learning



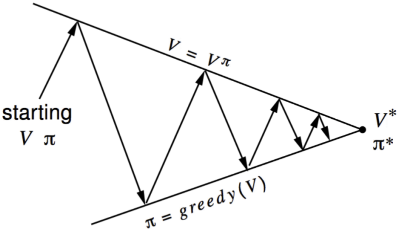
Control: Optimize the policy for a MDP

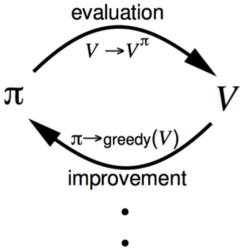
1. Model-based control: optimize the value function with known MDP
2. Model-free control: optimize the value function with unknown MDP
3. Many model-free RL examples: Go, robot locomation, patient treatment, helicopter control, Atari, Starcraft

Policy Iteration

1. Iterate through the two steps:
   1. Evaluate the policy π (computing v given current π)
   2. Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi) \tag{14}$$





Policy Iteration for a Known MDP

1. compute the state-action value of a policy π:

$$q_{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{\pi_i}(s')$$

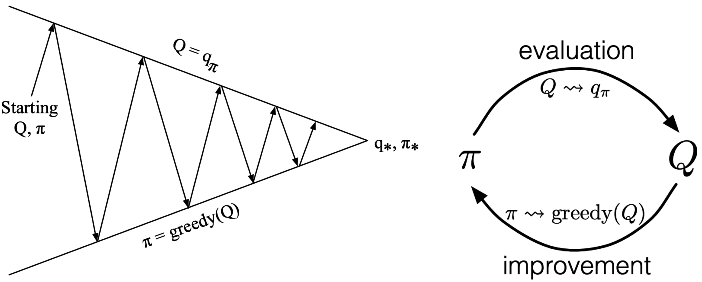
2. Compute new policy π_{i+1} for all $s \in \mathcal{S}$ following

$$\pi_{i+1}(s) = \arg\max_a q_{\pi_i}(s,a) \tag{15}$$

3. Problem: What to do if there is neither $R(s,a)$ nor $P(s'|s,a)$ known/available?

Generalized Policy Iteration

Monte Carlo version of policy iteration



1. Policy evaluation: Monte-Carlo policy evaluation $Q = q_\pi$
2. Policy improvement: Greedy policy improvement?

$$\pi(s) = \arg\max_a q(s, a)$$

ϵ-Greedy Exploration

1. Trade-off between exploration and exploitation (we will talk about this in later lecture)
2. ϵ-Greedy Exploration: Ensuring continual exploration
    1. All actions are tried with non-zero probability
    2. With probability $1 - \epsilon$ choose the greedy action
    3. With probability ϵ choose an action at random

$$\pi(a|s) = \begin{cases} \epsilon/|\mathcal{A}| + 1 - \epsilon & \text{if } a^* = \arg\max_{a \in \mathcal{A}} Q(s, a) \\ \epsilon/|\mathcal{A}| & \text{otherwise} \end{cases}$$

Monte Carlo with ϵ-Greedy Exploration

Algorithm 2

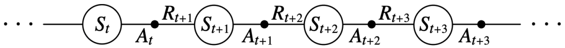
Initialize $Q(S, A) = 0, N(S, A) = 0, \epsilon = 1, k = 1$
$\pi_k = \epsilon\text{-greedy}(Q)$
loop
    Sample k-th episode $(S_1, A_1, R_2, ..., S_T) \sim \pi_k$
    for each state S_t and action A_t in the episode **do**
        $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$
    end for
    $k \leftarrow k + 1, \epsilon \leftarrow 1/k$
    $\pi_k = \epsilon\text{-greedy}(Q)$
end loop

TD for Prediction and Control

1. We can also use TD instead of MC in our control loop
    1. Apply TD to $Q(S, A)$
    2. Use ϵ-greedy policy improvement
    3. Update every time-step rather than at the end of one episode

Recall: TD Prediction

1. An episode consists of an alternating sequence of states and state–action pairs:



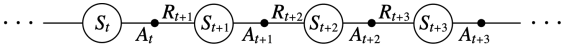
2. TD(0) method for estimating the value function $V(S)$

$$A_t \leftarrow \text{action given by } \pi \text{ for S}$$
$$\text{Take action } A_t\text{, observe } R_{t+1} \text{ and } S_{t+1}$$
$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

3. How about estimating action value function $Q(S)$?

Sarsa: On-Policy TD Control

1. An episode consists of an alternating sequence of states and state–action pairs:



2. ϵ-greedy policy for one step, then bootstrap the action value function:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \Big]$$

3. The update is done after every transition from a nonterminal state S_t
4. TD target $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$

Sarsa algorithm

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s,a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

n-step Sarsa

1. Consider the following n-step Q-returns for $n = 1, 2, \infty$

$$
\begin{aligned}
n = 1 (Sarsa) q_t^{(1)} =& R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \\
n = 2 \qquad\qquad q_t^{(2)} =& R_{t+1} + \gamma R_{t+2} + \gamma^2 Q(S_{t+2}, A_{t+2}) \\
&\vdots \\
n = \infty (MC) \quad q_t^{\infty} =& R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-t-1} R_T
\end{aligned}
$$

2. Thus the n-step Q-return is defined as

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n})$$

3. n-step Sarsa updates Q(s,a) towards the n-step Q-return:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big(q_t^{(n)} - Q(S_t, A_t) \Big)$$

Tomorrow

1. Lecture on on-policy learning and off-policy learning