Week 3: Model-free Prediction and Control

Bolei Zhou

UCLA

January 19, 2026

Announcement

1. Assignment 1 dues by the end of this week.
   1. `https://github.com/ucla-rlcourse/cs260r-assignment-2026winter/tree/main/assignment1`
   2. You should have already started working on it
2. RL examples: `https://github.com/ucla-rlcourse/RLexample`

This Week's Plan

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

Review on the control in MDP

P = env.P (part of the environment)

1. When the MDP is known?
   1. Both R and P are exposed to the agent
   2. Therefore we can run policy iteration and value iteration
2. Policy iteration: Given a known MDP, compute the optimal policy and the optimal value function
   1. Policy evaluation: iteration on the Bellman expectation backup

$$v_t(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_{t-1}(s'))$$

   2. Policy improvement: greedy on action-value function q

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

$$\pi_{t+1}(s) = \arg\max_a q_{\pi_t}(s,a)$$

Review on the control in MDP introduced last week

1. Value iteration: Given a known MDP, compute the optimal value function
2. Iteration on the Bellman optimality backup

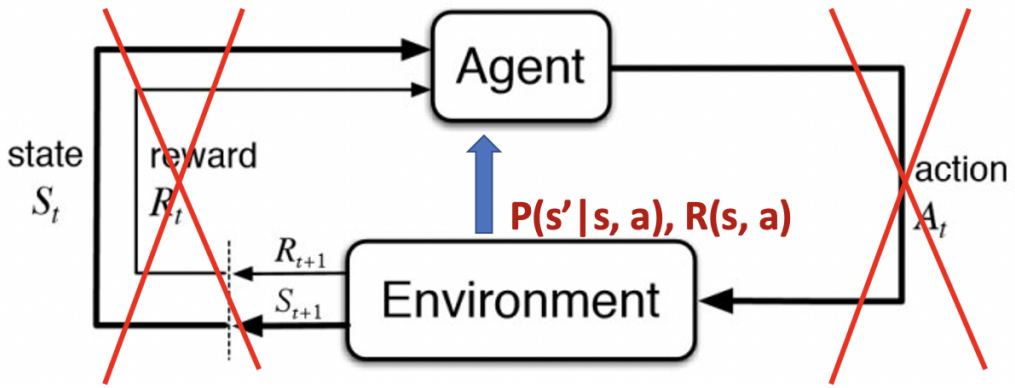
$$v_{t+1}(s) \leftarrow \max_{a \in \mathcal{A}} R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) v_t(s')$$

3. To retrieve the optimal policy after the value iteration:

$$\pi^*(s) \leftarrow \arg\max_{a} R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) v_{end}(s') \tag{1}$$

RL when knowing how the world works

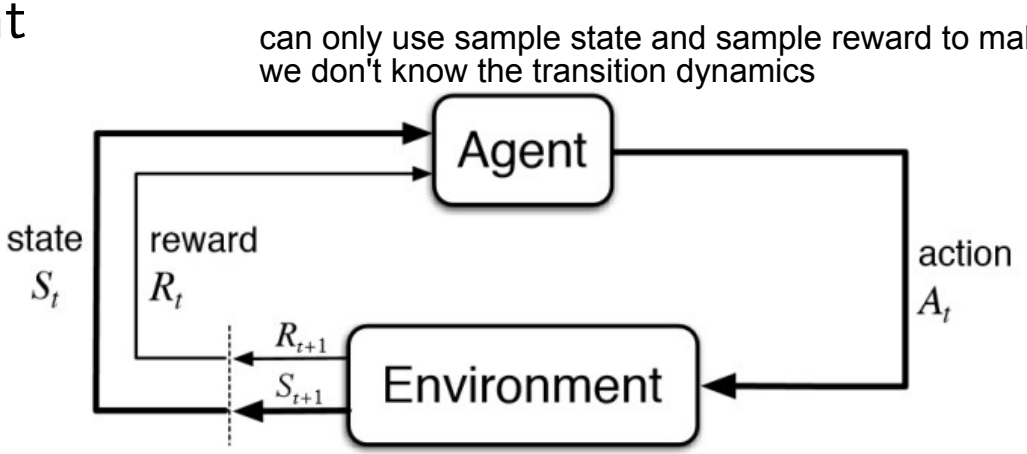
1. Both of the policy iteration and value iteration assume the direct access to the dynamics and rewards of the environment



2. In a lot of real-world problems, MDP model is either unknown or known by too big or too complex to use
   1. Atari Game, Game of Go, Helicopter, Portfolio management, etc

Model-free RL: Learning through interactions

1. Model-free RL can solve the problems through the interaction with the environment



2. No more direct access to the known transition dynamics and reward function
3. Trajectories/episodes are collected by the agent's interaction with the environment
4. Each trajectory/episode contains $\{S_1, A_1, R_2, S_2, A_2, R_3, ..., S_T\}$

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

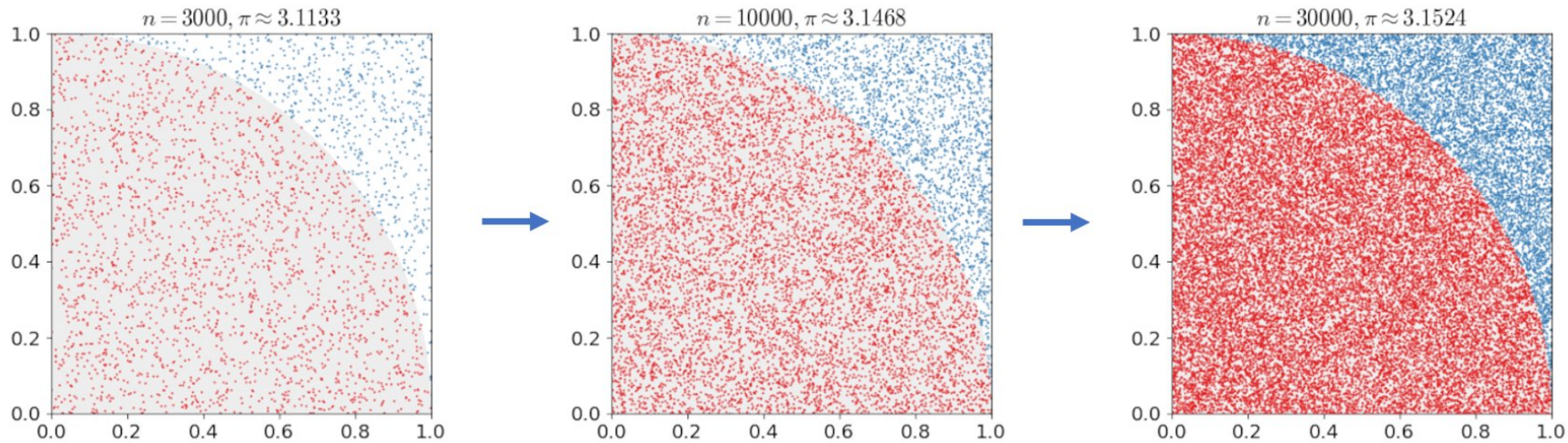
given a fixed policy, evaluate the state values in a model-free prediction

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

What is Monte-Carlo Method?

random sampling to estimate some numerical values
take pi for example

1. A broad class of computational algorithms that rely on repeated random sampling to compute numerical values
2. Example: Estimate the value of π



Monte Carlo Estimation of π

```python
import random
import math

def estimate_pi(num_samples):
    inside_circle = 0

    for _ in range(num_samples):
        # Sample a point uniformly in the square [-1, 1] x [-1, 1]
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)

        # Check if the point is inside the unit circle
        if x**2 + y**2 <= 1:
            inside_circle += 1

    # Area ratio: (pi * r^2) / (2r)^2 = pi / 4
    return 4 * inside_circle / num_samples


if __name__ == "__main__":
    N = 1_000_000
    pi_estimate = estimate_pi(N)

    print(f"Estimated   : {pi_estimate}")
    print(f"True    :      {math.pi}")
    print(f"Error:        {abs(pi_estimate - math.pi)}")
```

Monte-Carlo Policy Evaluation

return = cumulative future rewards after starting at some state s

1. Return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$ under policy π
2. $v^\pi(s) = \mathbb{E}_{\tau \sim \pi}[G_t | s_t = s]$, thus expectation over trajectories τ generated by following π sample multiple times (entire trajectory) every state and take the average return
3. MC simulation: we can simply sample a lot of trajectories, compute the actual returns for all the trajectories, then average them
4. MC policy evaluation uses empirical mean return instead of expected return
5. MC does not require MDP dynamics/rewards, no bootstrapping, and does not assume state is Markov.
6. Only applied to episodic MDPs (each episode terminates)

Example: Monte Carlo Algorithm for Computing Value of a MRP

Algorithm 1 Monte Carlo simulation to calculate MRP value function

$i \leftarrow 0, G_t \leftarrow 0$
while $i \neq N$ **do**
    generate an episode, starting from state s and time t
    Using the generated episode, calculate return $g = \sum_{i=t}^{H-1} \gamma^{i-t} r_i$
    $G_t \leftarrow G_t + g, i \leftarrow i + 1$
end while
$V_t(s) \leftarrow G_t / N$

1. For example: to calculate $V(s_4)$ we can generate a lot of trajectories then take the average of the returns:
    1. return for s_4, s_5, s_6, s_7 : $0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 10 = 2.5$
    2. return for s_4, s_3, s_2, s_1 : $0 + \frac{1}{2} \times 0 + \frac{1}{4} \times 5 = 1.25$
    3. return for s_4, s_5, s_6, s_6: 0
    4. more trajectories

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 $N(s) \rightarrow \infty$

Monte-Carlo Policy Evaluation (prediction)

process of estimating the value function v_\pi(s)

solve this by averaging the returns (total accumulated rewards) observed from actual interactions with the env

1. How to calculate $G(s)$: compute backward

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode): put agent in environment

Gt-1

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$ generate trajectory

G0 G1

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$: go backwards thru the trajectory

$G \leftarrow \gamma G + R_{t+1}$ Recursively, G_t = R_{t+1} + \gamma G_{t+1}

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$: (First visit)

Append G to $Returns(S_t)$

$V(S_t) \leftarrow$ average$(Returns(S_t))$

this works because G represents future rewards

Incremental Mean

from the moving average of samples $x_1, x_2, ...$

$$\begin{aligned}
\mu_t &= \frac{1}{t}\sum_{j=1}^{t} x_j \\
&= \frac{1}{t}\left(x_t + \sum_{j=1}^{t-1} x_j\right) \\
&= \frac{1}{t}(x_t + (t-1)\mu_{t-1}) \\
&= \mu_{t-1} + \frac{1}{t}(x_t - \mu_{t-1}) \\
&\qquad\qquad \text{residual}
\end{aligned}$$

need to cache all the return information before, but this way we can get the same value with just the past value and the t value

Incremental MC Updates

1. Collect one episode $(S_1, A_1, R_2, ..., S_t)$
2. For each state s_t with computed return G_t

$$N(S_t) \leftarrow N(S_t) + 1$$

$$v(S_t) \leftarrow v(S_t) + \frac{1}{N(S_t)}(G_t - v(S_t))$$

we can think of this as alpha

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

alpha allows us to "forget" old episodes

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

Difference between DP and MC for policy evaluation

1. Dynamic Programming (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:

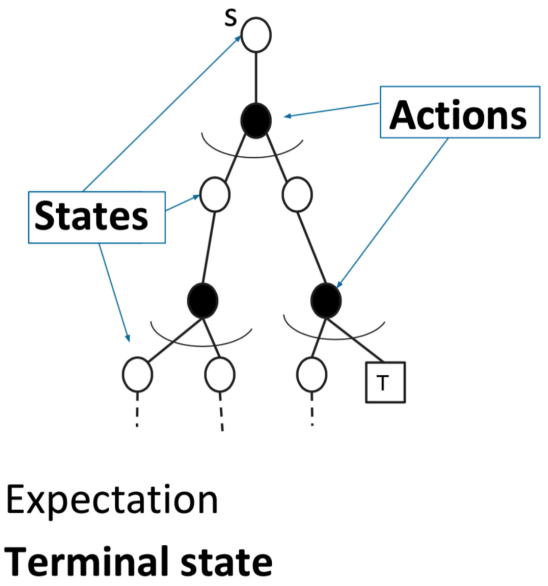
DP is slower because there are two summations

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

over all actions

over al outcome states

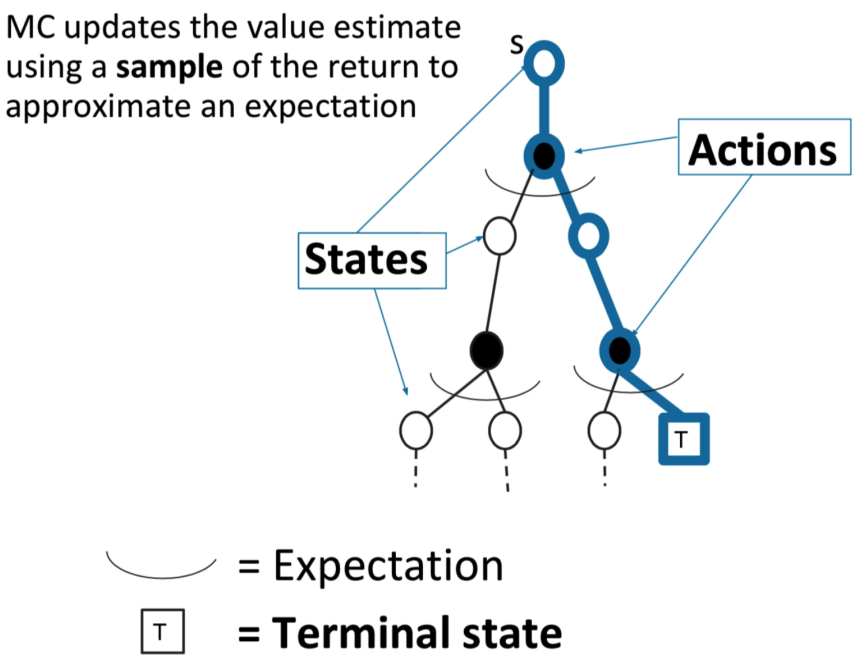
so many states and actions



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

mc -> if we are interested in a single state, then we can just run several episodes starting from that stat
and its chill

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

Introduction of 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**
    1. Key idea of bootstrapping: updating an estimate using *other estimates* (instead of waiting for the final outcome).

Introduction of Temporal-Difference (TD) Learning

plug in agent with policy \pi

\alpha is learning rate

online = learn while going thorugh the episode

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

S_t, a_t, r_{t+1}, s_{t+1}

v(S_t) is the value of the current state?

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

get residual of the TD target

3. $R_{t+1} + \gamma v(S_{t+1})$ is called TD target take immediate reward and discounted value of the next state
4. $\delta_t = R_{t+1} + \gamma v(S_{t+1}) - v(S_t)$ is called the TD error update based on the error
5. Comparison: Incremental Monte-Carlo
   1. Update $v(S_t)$ toward actual return G_t given an episode i

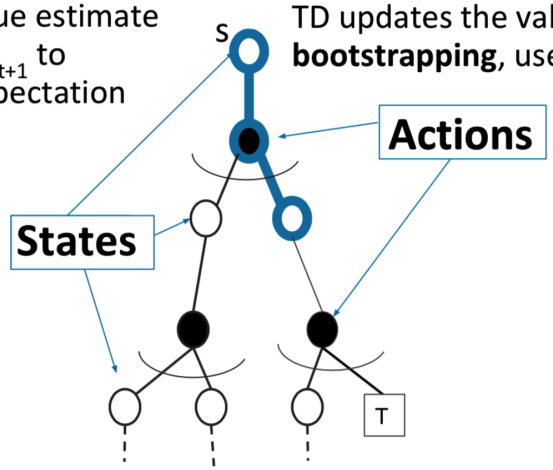
$$v(S_t) \leftarrow v(S_t) + \alpha(G_{i,t} - v(S_t))$$

TD approach: You drive for 5 minutes and reach a familiar landmark.
You realize, "Oh, from here it usually takes 20 minutes."
You immediately update your original prediction: 5 minutes elapsed + 20 minutes remaining = 25 minutes total.
You don't need to finish the drive to correct your initial error.

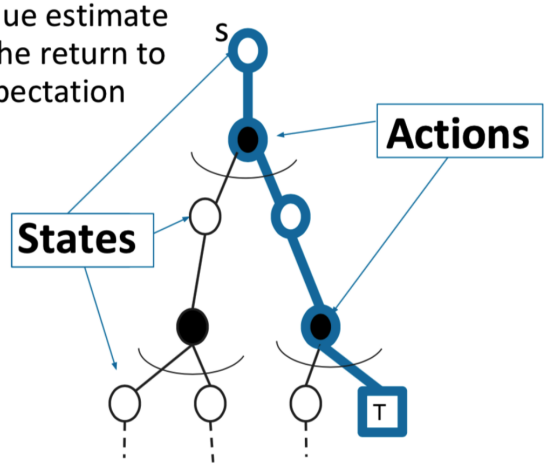
Advantages of TD over MC

TD updates the value estimate using a **sample** of s_{t+1} to approximate an expectation

TD updates the value estimate by **bootstrapping**, uses estimate of $V(s_{t+1})$



MC updates the value estimate using a **sample** of the return to approximate an expectation



= Expectation

T = **Terminal state**

Comparison of TD and MC

1. TD can learn online after every step
2. MC must wait until end of episode before return is known

3. TD can learn from incomplete sequences
4. MC can only learn from complete sequences

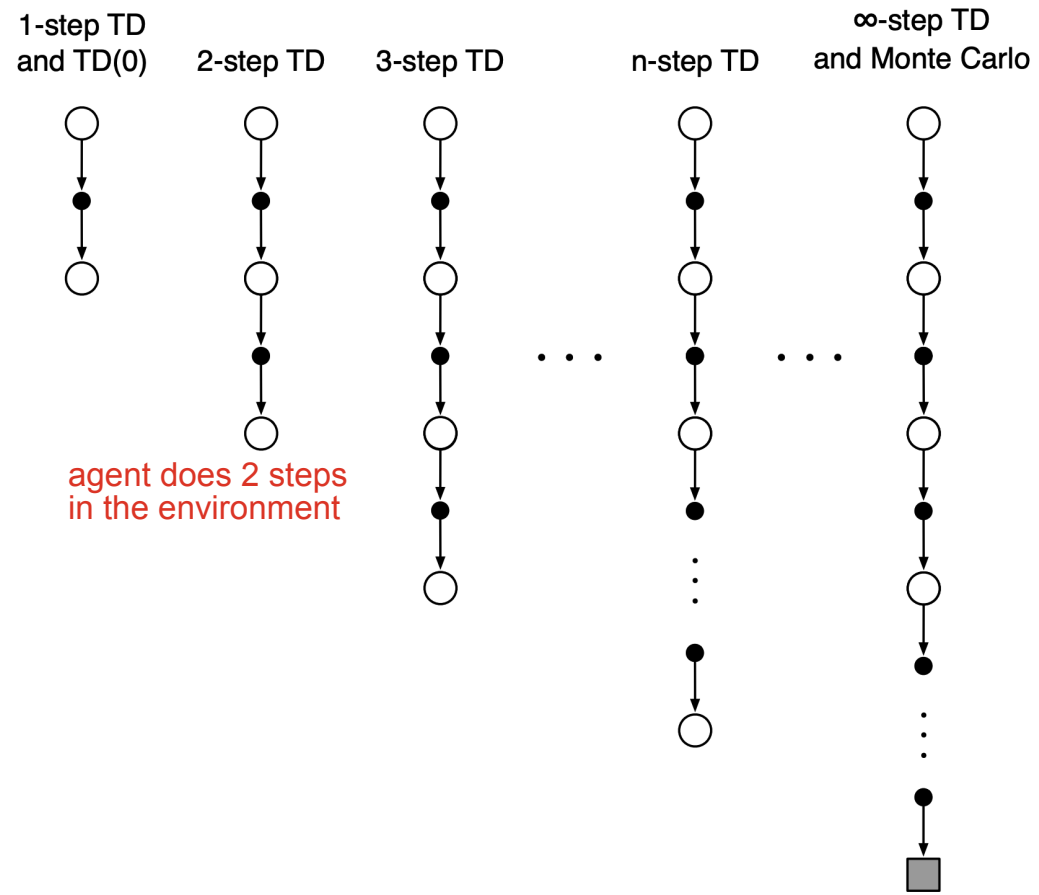
don't need to finish an episode

5. TD works in continuing (non-terminating) environments
6. MC only works for episodic (terminating) environments

7. TD exploits Markov property, more efficient in Markov environments
8. MC does not exploit Markov property, more effective in non-Markov environments

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 $n = 1, 2, \infty$

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

2. Thus the n-step return is defined as

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

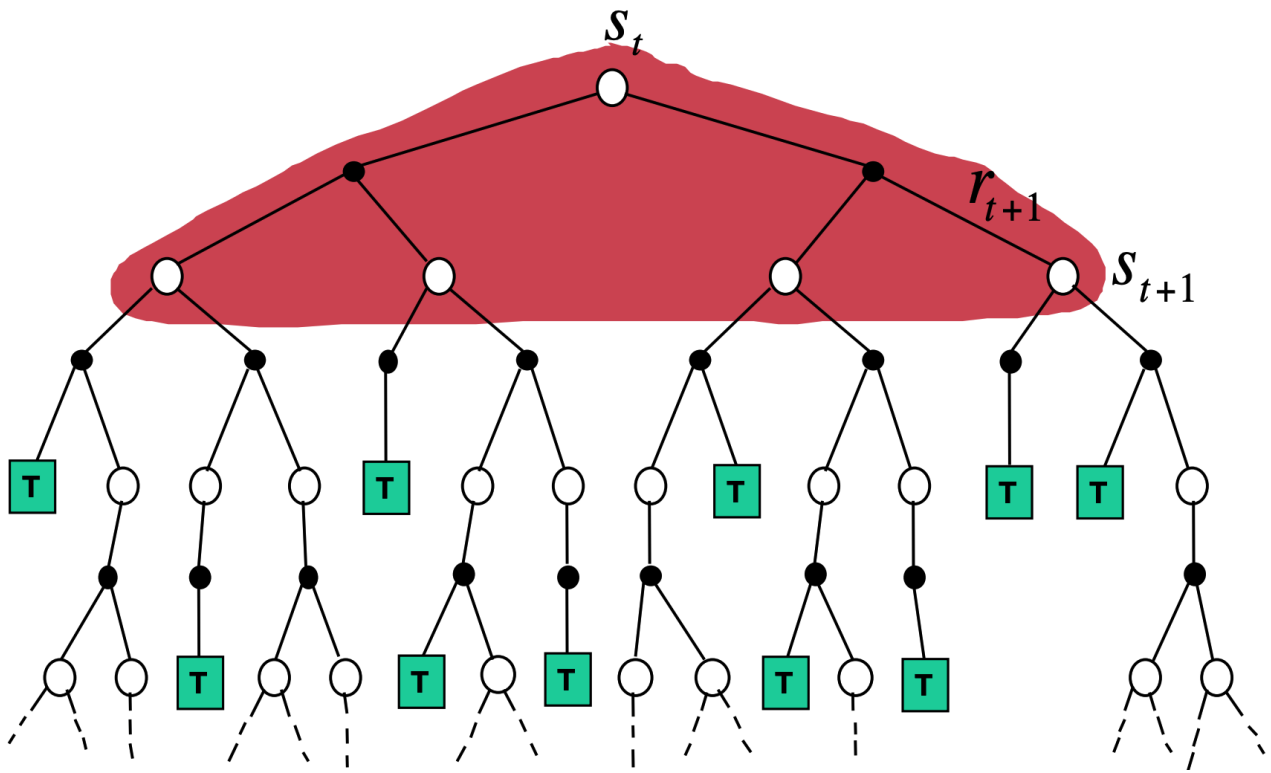
3. n-step TD: $v(S_t) \leftarrow v(S_t) + \alpha\Big(G_t^n - v(S_t)\Big)$

Bootstrapping and Sampling for DP, MC, and TD

1. Bootstrapping: update involves a prior 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

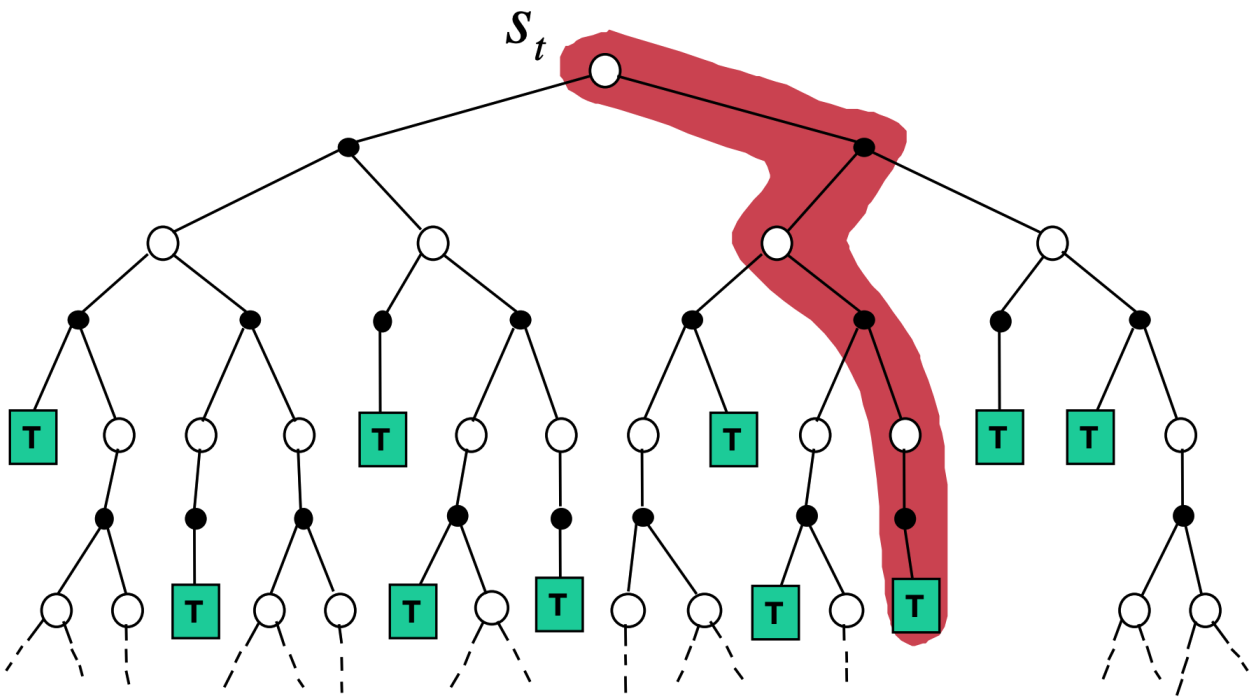
Unified View: Dynamic Programming Backup

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



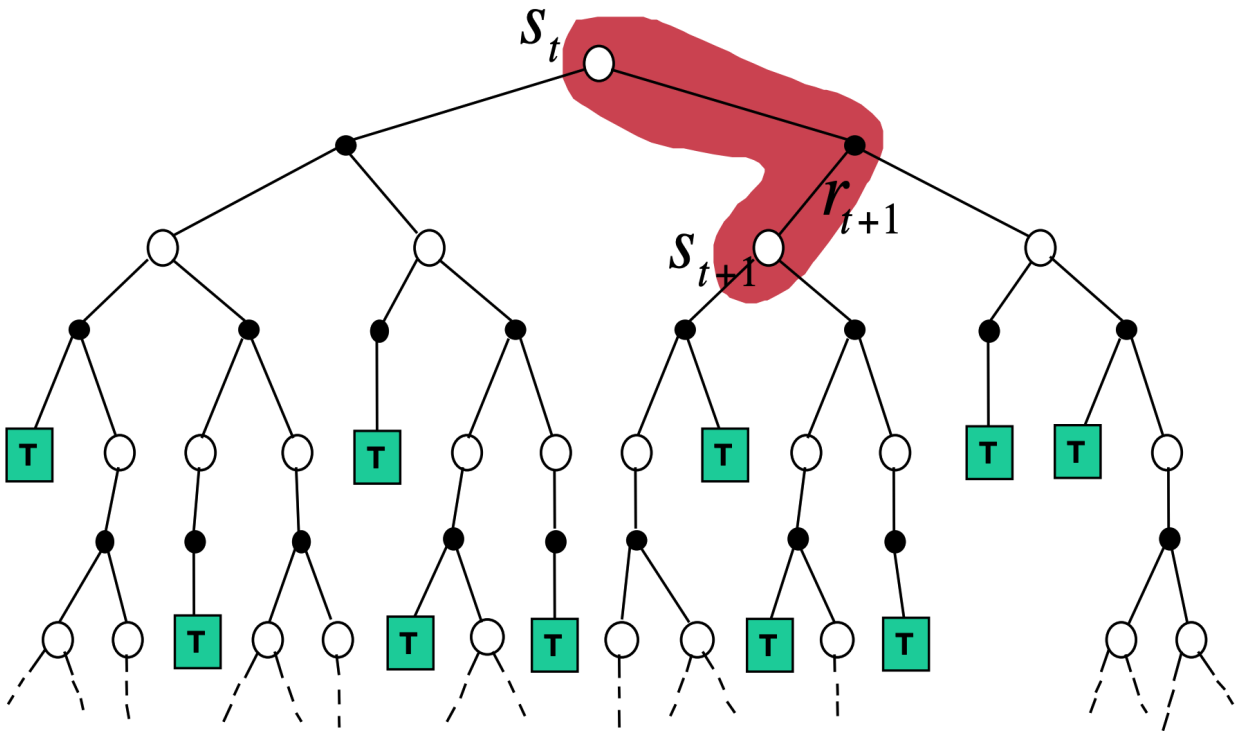
Unified View: Monte-Carlo Backup

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

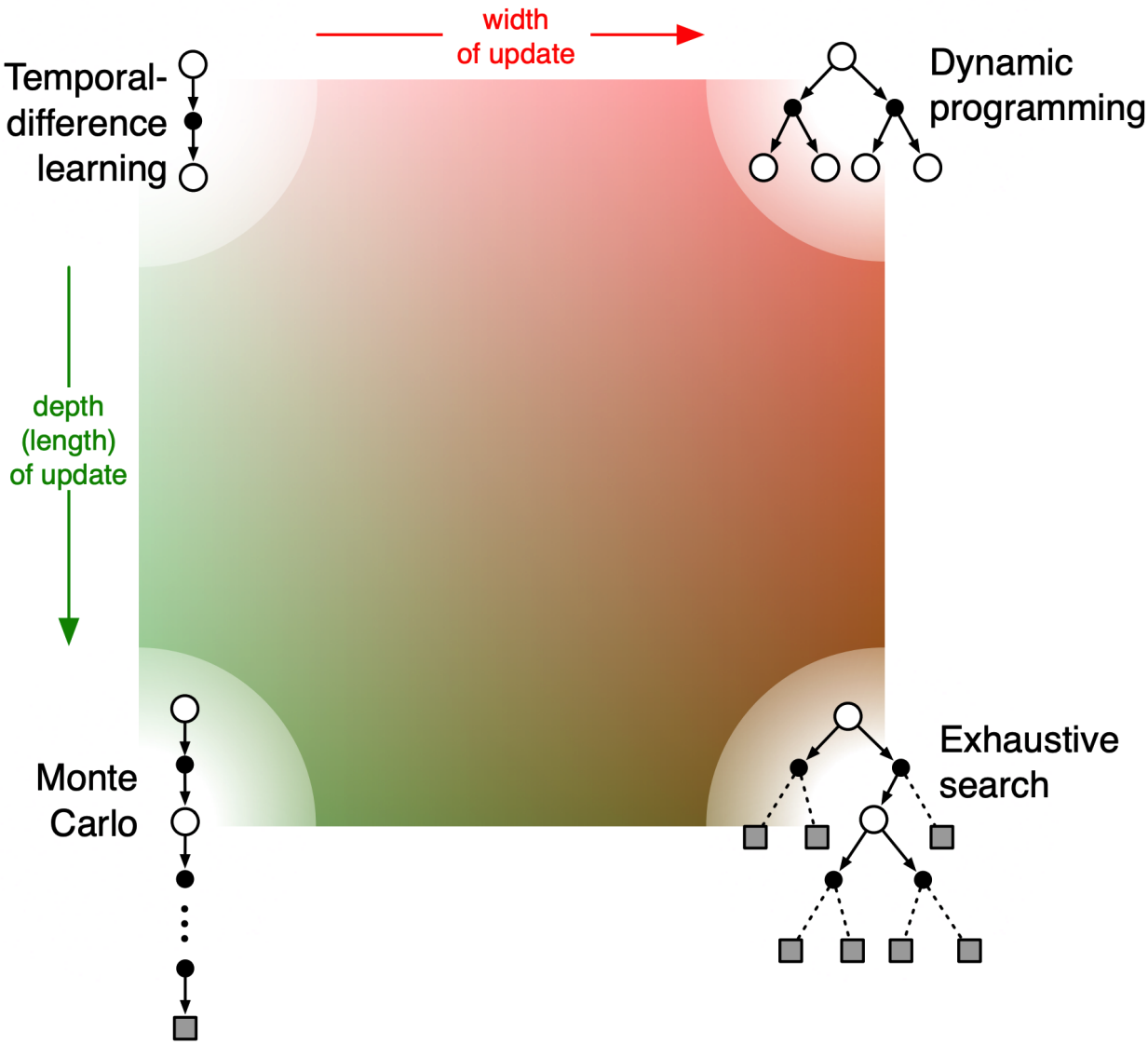


Unified View: Temporal-Difference Backup

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



Unified View of Reinforcement Learning



A Short Summary

1. Model-free prediction
   1. Evaluate the state value by only interacting with the environment
   2. Many algorithms can do it: Temporal Difference Learning and Monte-Carlo method

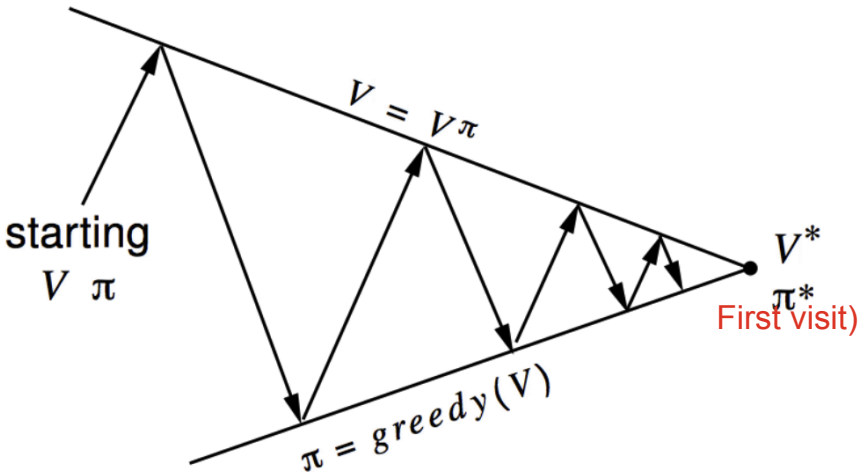
Model-free Control for MDP

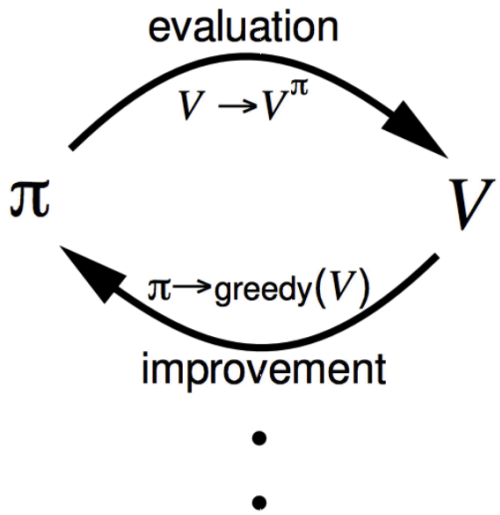
1. Model-free control:
   1. Optimize the value function of an unknown MDP
   2. Generate a optimal control policy
2. Generalized Policy Iteration (GPI) with MC or TD in the loop

Revisiting 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

$$\pi' = \text{greedy}(v_\pi) \tag{2}$$

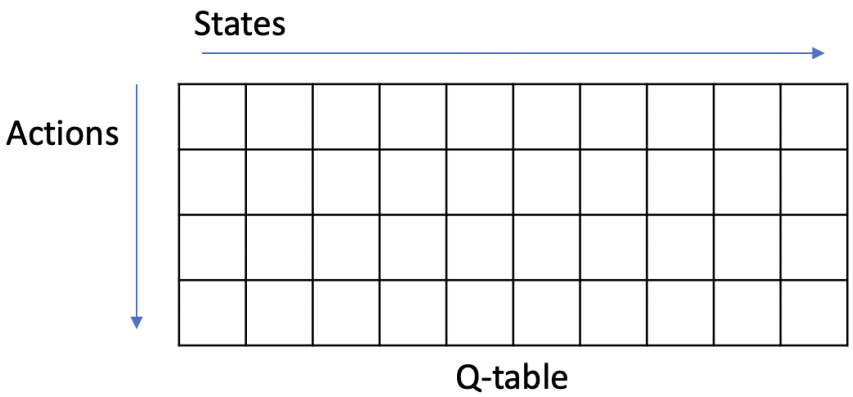




Policy Iteration for a Known MDP

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

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



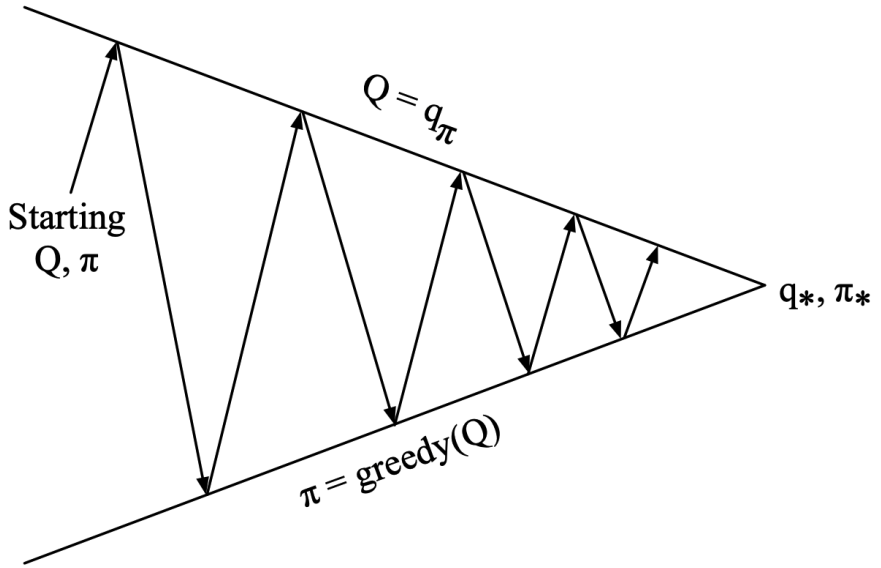
2. Compute new policy π_{i+1} for all $s \in \mathcal{S}$ following

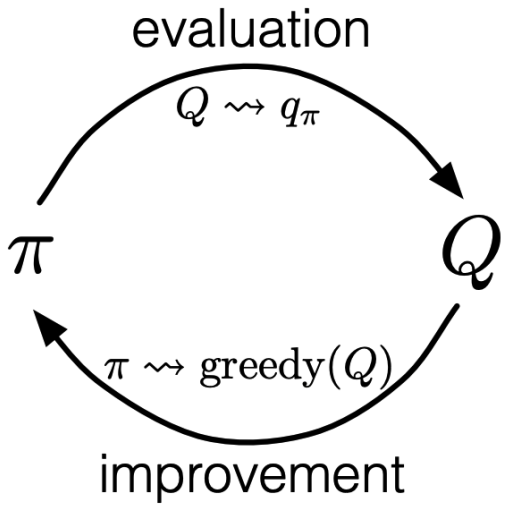
$$\pi_{i+1}(s) = \arg\max_a q_{\pi_i}(s,a) \tag{3}$$

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

Generalized Policy Iteration with Action-Value Function

Monte Carlo version of policy iteration





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

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

Monte Carlo with Exploring Starts

1. One assumption to obtain the guarantee of convergence in PI: Episode has exploring starts
2. Exploring starts can ensure all actions are selected infinitely often

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:

$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):

Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0

Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$\pi(S_t) \leftarrow \text{argmax}_a Q(S_t, a)$

estimate the Q(s,a) from a random start and action

maintain a q table

Monte Carlo with ϵ-Greedy Exploration

we want the agent to explore different states and actions

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

1. ϵ-Greedy Policy improvement theorem (textbook pg-101): For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_{a \in \mathcal{A}} \pi'(a|s) q_\pi(s, a) \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \max_a q_\pi(s, a) \\
&\geq \frac{\epsilon}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} q_\pi(s, a) + (1 - \epsilon) \sum_{a \in \mathcal{A}} \frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}|}}{1 - \epsilon} q_\pi(s, a) \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) = v_\pi(s)
\end{aligned}
$$

Therefore from the policy improvement theorem $v_{\pi'}(s) \geq v_\pi(s)$

Monte Carlo with ϵ-Greedy Exploration

Algorithm 2

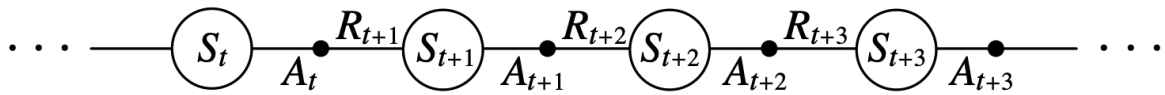
Initialize $Q(S, A) = 0, N(S, A) = 0, \epsilon = 1, k = 1$
$\pi_k = \epsilon$-greedy(Q)
loop
collect episode
  Sample k-th episode $(S_1, A_1, R_2, ..., S_T) \sim \pi_k$
  for each state S_t and action A_t in the episode **do**
    $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$
    get frequency of each state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

MC vs. TD for Prediction and Control

1. Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
   1. Lower variance
   2. Online
   3. Incomplete sequences
2. So we can 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:



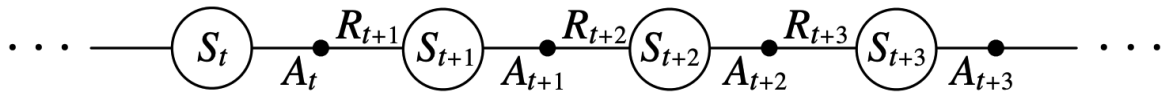
2. TD(0) method for estimating the value function $V(S)$

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

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

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:

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

recall that this is the TD target (the expected return from the future state)

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

Type text here

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 $n = 1, 2, \infty$

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

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

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

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

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

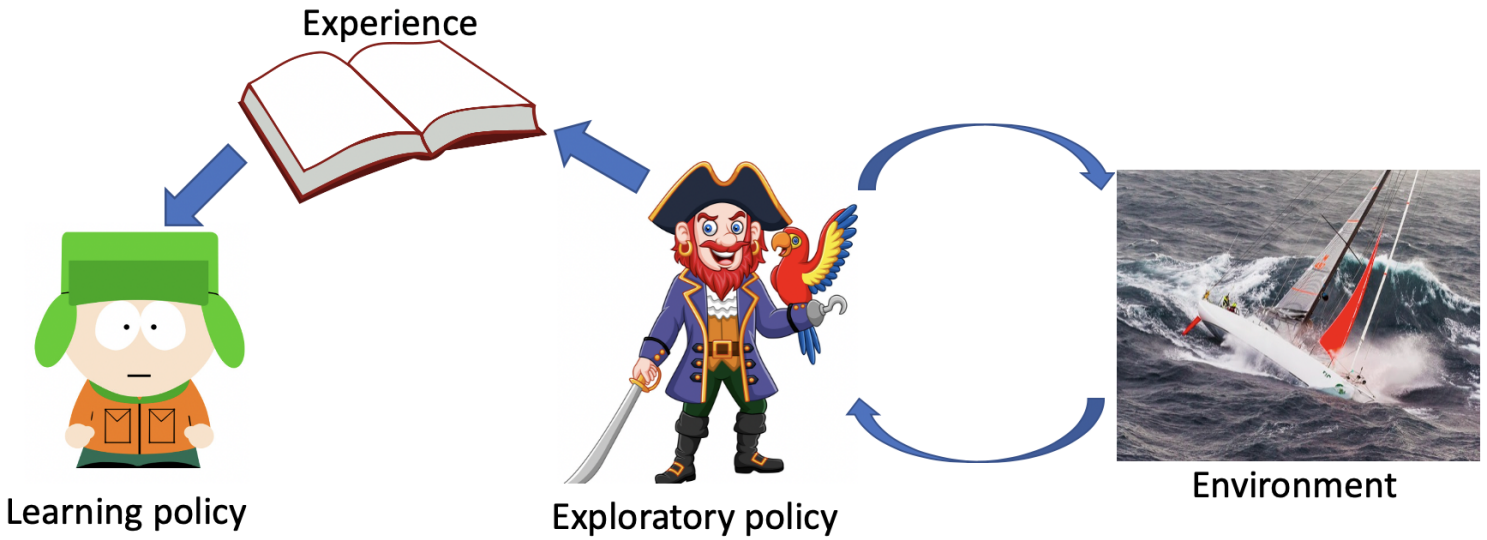
On-policy Learning vs. Off-policy Learning

on-policy: use current policy to collect data from environment

1. On-policy learning: Learn about policy π from the experience collected from π
   1. Behave non-optimally in order to explore all actions, then reduce the exploration. e.g., ϵ-greedy
2. Another important approach is **off-policy learning** which essentially uses **two different polices**:
   1. the one which is being learned about and becomes the optimal policy
   2. the other one which is more exploratory and is used to generate trajectories
3. Off-policy learning: Learn about policy π from the experience sampled from another policy μ
   1. π: target policy
   2. μ: behavior policy

we use a behavior policy to play around in the environment and collect data

Off-policy Learning



1. Following behaviour policy $\mu(a|s)$ to collect data

$$S_1, A_1, R_2, ..., S_T \sim \mu$$
$$\text{Update } \pi \text{ using } S_1, A_1, R_2, ..., S_T$$

2. It leads to many benefits:
   1. Learn about optimal policy while following exploratory policy
   2. Learn from observing humans or other agents
   3. Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$

Off-Policy Control with Q Learning

1. Off-policy learning of action values $Q(s, a)$
2. No importance sampling is needed
3. Next action in TD target is selected from the alternative action $A' \sim \pi(.|S_t)$
4. update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)\big)$$

Off-Policy Control with Q-Learning

1. We allow both behavior and target policies to improve
2. The target policy π is greedy on $Q(s, a)$

   in inference, greedy is the most optimal

$$\pi(S_{t+1}) = \arg\max_{a'} Q(S_{t+1}, a')$$

3. The behavior policy μ could be totally random, but we let it improve following ϵ-greedy on $Q(s, a)$
4. Thus Q-learning target:

$$\begin{aligned} R_{t+1} + \gamma Q(S_{t+1}, A') &= R_{t+1} + \gamma Q(S_{t+1}, \arg\max Q(S_{t+1}, a')) \\ &= R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') \end{aligned}$$

5. Thus the Q-Learning update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a} Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Q-learning algorithm

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

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
  Repeat (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

Comparison of Sarsa and Q-Learning

1. Sarsa: On-Policy TD control

   Choose action A_t from S_t using policy derived from Q with ϵ-greedy

   Take action A_t, observe R_{t+1} and S_{t+1}

   Choose action A_{t+1} from S_{t+1} using policy derived from Q with ϵ-greedy

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

2. Q-Learning: Off-Policy TD control

   Choose action A_t from S_t using policy derived from Q with ϵ-greedy

   Take action A_t, observe R_{t+1} and S_{t+1}

   we are always taking the value of the best action after the next step

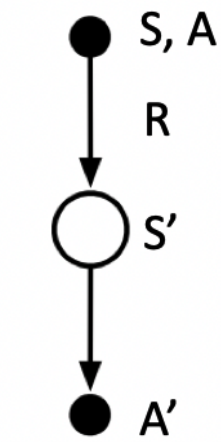
   Then 'imagine' A_{t+1} as $\arg\max Q(S_{t+1}, a')$ in the update target

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

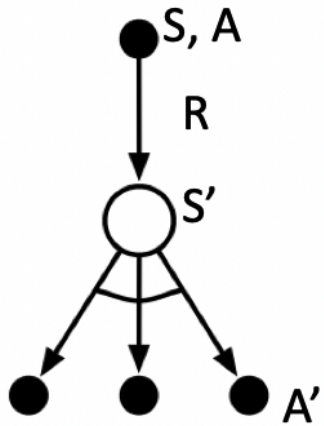
   this is different from the actual action we might take after reaching the next state

Comparison of Sarsa and Q-Learning

1. Backup diagram for Sarsa and Q-learning



Sarsa



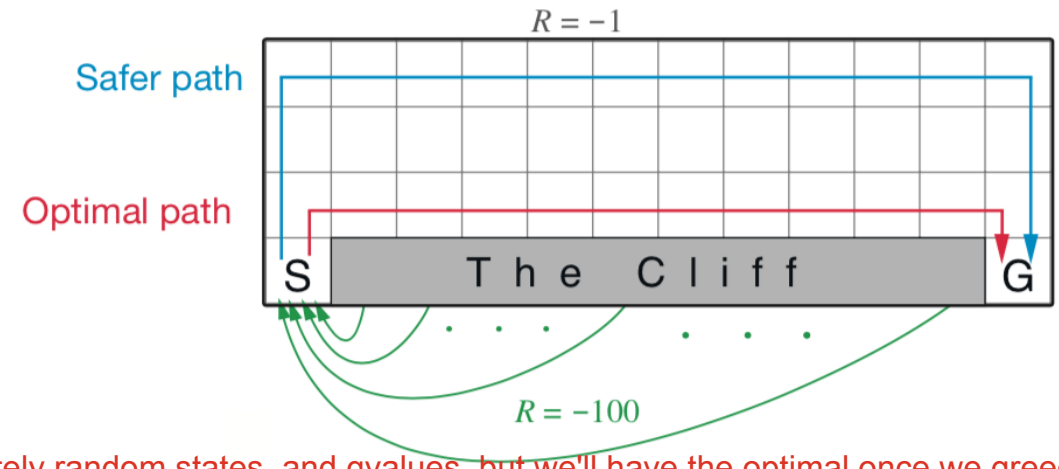
Q-learning

2. In Sarsa, A and A' are sampled from the same policy so it is on-policy
3. In Q Learning, A and A' are from different policies, with A being more exploratory and A' determined directly by the max operator

Example on Cliff Walk (Example 6.6 from Textbook)

`https://github.com/ucla-rlcourse/RLexample/blob/master/modelfree/cliffwalk.py`

in sarsa, it will take a conservative path, q values of the red will be a lot less



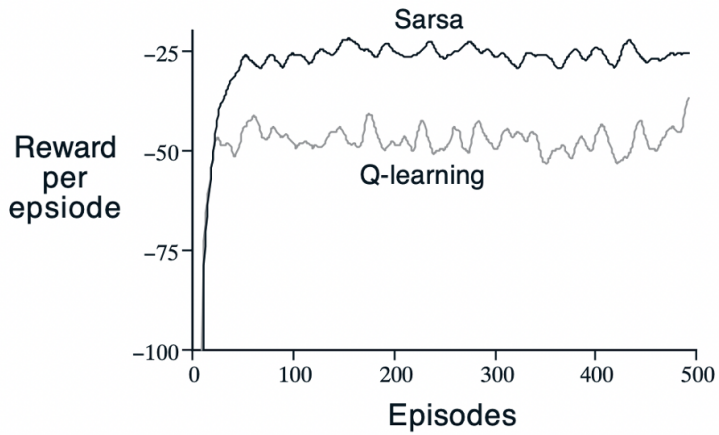
q-learning, we have completely random states, and qvalues, but we'll have the optimal once we greedy

```
| 0 | 0 | 0 | 0 | R | R | R | R | R | R | R | R |
| R | R | R | R | R | 0 | 0 | 0 | 0 | 0 | 0 | R |
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | R |
| R | * | * | * | * | * | * | * | * | * | * | G |
```

Result of Sarsa

```
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | R | R | R | R | R | R | R | R | R | R | R |
| R | * | * | * | * | * | * | * | * | * | * | G |
```

Result of Q-Learning



On-line performance of Q-learning is worse than that of Sarsa

Summary of DP and TD

Expected Update (DP)	**Sample Update (TD)**
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[R + \gamma V(S')\|s]$	TD Learning $V(S) \leftarrow^{\alpha} R + \gamma V(S')$
Q-Policy Iteration $Q(S, A) \leftarrow \mathbb{E}[R + \gamma Q(S', A')\|s, a]$	Sarsa $Q(S, A) \leftarrow^{\alpha} R + \gamma Q(S', A')$
Q-Value Iteration $Q(S, A) \leftarrow \mathbb{E}[R + \gamma \max_{a' \in \mathcal{A}} Q(S', A')\|s, a]$	Q-Learning $Q(S, A) \leftarrow^{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$

where $x \leftarrow^{\alpha} y$ is defined as $x \leftarrow x + \alpha(y - x)$

Code Example of Sarsa and Q-Learning

https://github.com/ucla-rlcourse/RLexample/tree/master/modelfree

Off-policy Learning with Importance Sampling

What is importance sampling?

1. Estimate the expectation of a function $f(x)$

$$E_{x\sim P}[f(x)] = \int f(x)P(x)dx \approx \frac{1}{n}\sum_i f(x_i)$$

2. But sometimes it is difficult to sample x from $P(x)$, then we can sample x from another distribution $Q(x)$, then correct the weight

$$\begin{aligned}\mathbb{E}_{x\sim P}[f(x)] &= \int P(x)f(x)dx \\ &= \int Q(x)\frac{P(x)}{Q(x)}f(x)dx \\ &= \mathbb{E}_{x\sim Q}\left[\frac{P(x)}{Q(x)}f(x)\right] \approx \frac{1}{n}\sum_i \frac{P(x_i)}{Q(x_i)}f(x_i)\end{aligned}$$

how do we get P(x) though?
we don't, we evaluate a value x_i

Off-policy Learning with Importance Sampling

1. Expected return: $\mathbb{E}_{\tau\sim\pi}[r(\tau)]$ where $r(.)$ is the reward function and π is the policy
2. Estimate the expectation of return using trajectories τ_i sampled from another policy (behavior policy μ)

$$\begin{aligned}\mathbb{E}_{\tau\sim\pi}[r(\tau)] &= \int P_\pi(\tau) r(\tau) d\tau \\ &= \int P_\mu(\tau) \frac{P_\pi(\tau)}{P_\mu(\tau)} r(\tau) d\tau \\ &= \mathbb{E}_{\tau\sim\mu}\left[\frac{P_\pi(\tau)}{P_\mu(\tau)} r(\tau)\right] \\ &\approx \frac{1}{n}\sum_i \frac{P_\pi(\tau_i)}{P_\mu(\tau_i)} r(\tau_i)\end{aligned}$$

Off-Policy Monte Carlo with Importance Sampling

1. Generate episode from behavior policy μ and compute the generated return G_t

$$S_1, A_1, R_2, ..., S_T \sim \mu$$

2. Weight return G_t according to similarity between policies
   1. Multiply importance sampling corrections along whole episode

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} ... \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t$$

\pi(A_t | S_t) represents the probability of choosing A_t given S_t

3. Update value towards correct return

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{\pi/\mu} - V(S_t))$$

Off-Policy TD with Importance Sampling

1. Use TD targets generated from μ to evaluate π
2. Weight TD target $R + \gamma V(S')$ by importance sampling
3. Only need a single importance sampling correction

$$V(S_t) \leftarrow V(S_t) + \alpha\Big(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\big(R_{t+1} + \gamma V(S_{t+1})\big) - V(S_t)\Big)$$

4. Policies only need to be similar over a single step

Why don't use importance sampling on Q-Learning?

1. Off-policy TD

$$V(S_t) \leftarrow V(S_t) + \alpha\Big(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\big(R_{t+1} + \gamma V(S_{t+1})\big) - V(S_t)\Big) \tag{4}$$

2. Short answer: 1,Q-learning uses a deterministic policy so no action probability. 2,Q-learning does not make expected value estimates over the policy distribution. For the full answer click here
3. Remember bellman optimality backup from value iteration

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{a'} Q(s',a') \tag{5}$$

   1. Q-learning can be considered as sample update of value iteration, except instead of using the expected value over the transition dynamics, we use the sample collected from the environment

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a') \tag{6}$$

      Q-learning is over the transition distribution, not over policy distribution thus no need to correct different policy distributions

Eligibility Traces (Textbook Chapter 12)

1. Remember in TD learning, return for n-step TD is

$$G_{t:t+n} = R_{t+1} + \lambda R_{t+2} + .. + \lambda^{n-1} R_{t+n} + \lambda^n v(S_{t+n}) \tag{7}$$

2. a backup can be done toward a target that is half of a two-step return and half of a four-step return:

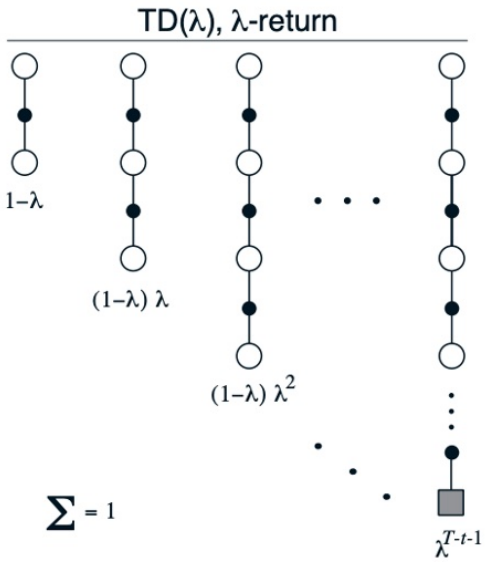
$$\frac{1}{2} G_{t:t+1}(V_t(S_{t+2})) + \frac{1}{2} G_{t:t+4}(V_t(S_{t+4})) \tag{8}$$

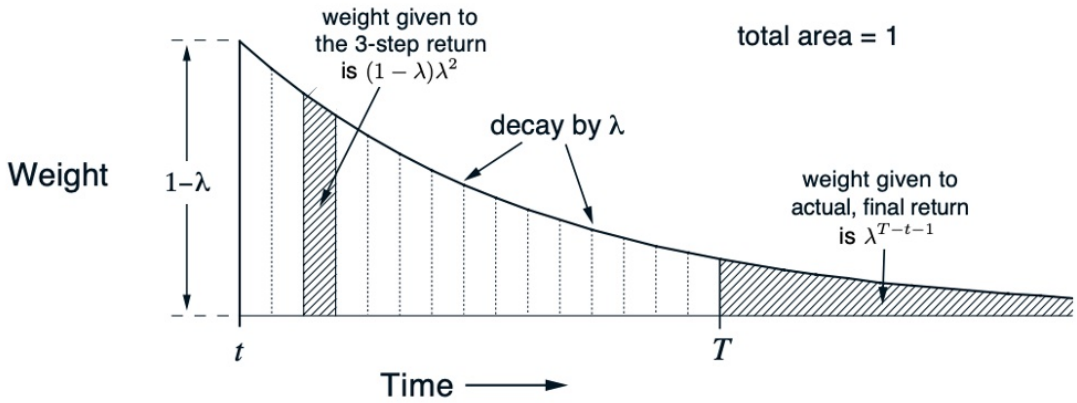
3. Such averaging can obtain another way of interrelating TD and Monte Carlo methods

Eligibility Traces

how we agregate the returns of multple steps

1. The λ-return is then defined as $G_t^\lambda = (1-\lambda)\sum_n \lambda^{n-1} G_{t:t+n}$



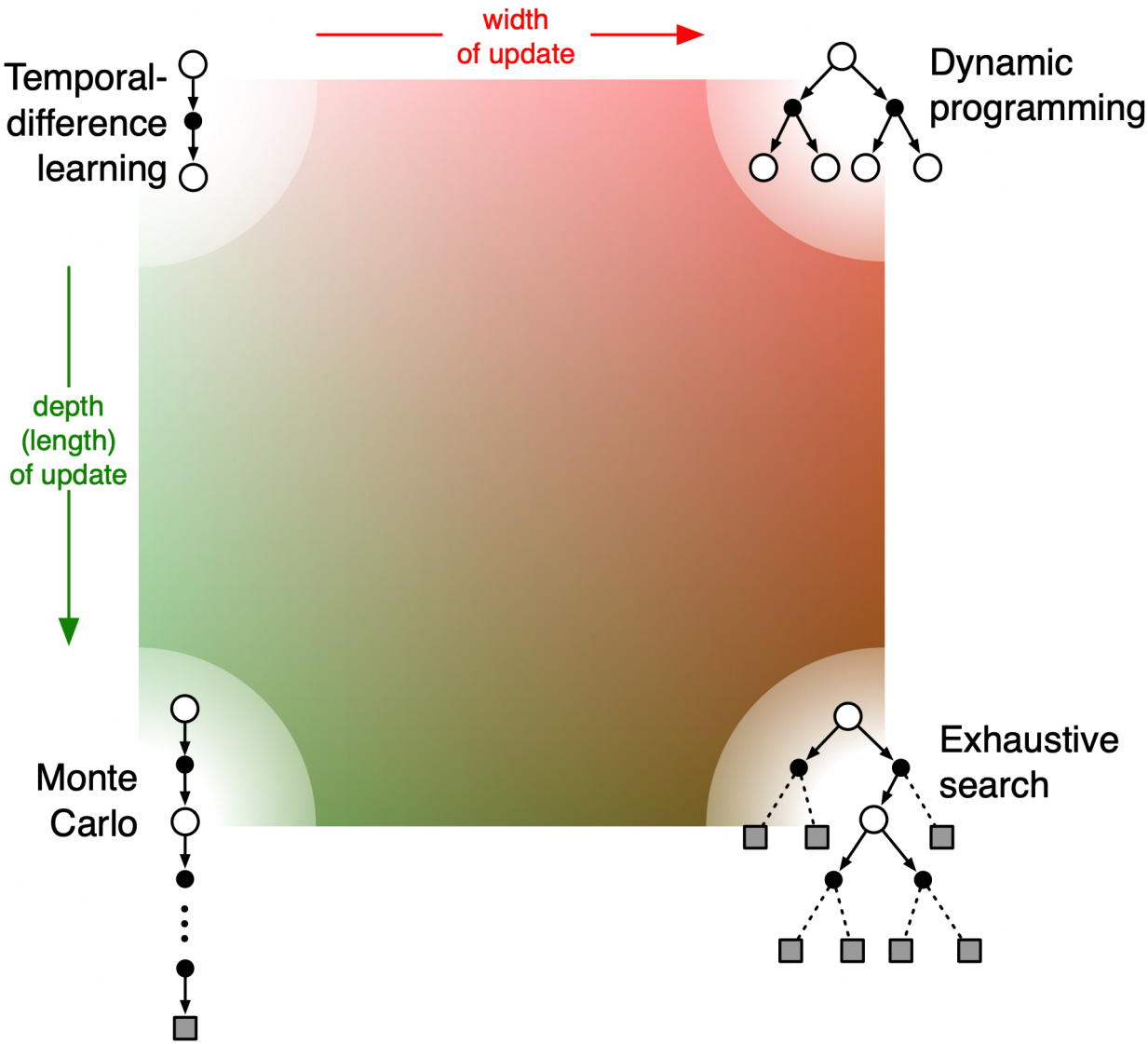


2. For $\lambda = 1$, updating according to the $\lambda-$return is a Monte Carlo algorithm. On the other hand, if $\lambda = 0$ it becomes a one-step TD method.
3. The $\lambda-$return gives an alternative way of moving smoothly between Monte Carlo and one-step TD methods

Eligibility Traces

1. Based on the $\lambda-$return we can derive TD(λ), SARSA(λ), Q-learning(λ), see Chapter 12
2. Eligibility traces provide an efficient, incremental way of shifting and choosing between Monte Carlo and TD methods
3. It is similar to n-step method but offers different computational complexity tradeoffs.

Unified View of Reinforcement Learning



Summary of the key concepts in this lecture

1. Model-free prediction and control
2. Monte Carlo method
3. Temporal Difference learning
    1. SARSA (on-policy TD control)
    2. Q-Learning (off-policy TD control)
4. Importance Sampling
5. Eligibility traces
6. Next week:
    1. Value function approximation and Deep Q-learning
    2. Textbook Chapter 9