2018 CS420, Machine Learning, Lecture 11

Introduction to Reinforcement Learning

Weinan Zhang

Shanghai Jiao Tong University

http://wnzhang.net

http://wnzhang.net/teaching/cs420/index.html

REVIEW

What is Machine Learning

A more mathematical definition by Tom Mitchell

- Machine learning is the study of algorithms that
  - improve their performance *P*
  - at some task *T*
  - based on experience *E*
  - with non-explicit programming

- A well-defined learning task is given by <*P, T, E*>

REVIEW

Machine Learning

- What we have learned so far

- Supervised Learning
  - To perform the desired output given the data and labels
  - e.g., to build a loss function to minimize

- Unsupervised Learning
  - To analyze and make use of the underlying data patterns/structures
  - e.g., to build a log-likelihood function to maximize

REVIEW

Supervised Learning

- Given the training dataset of (data, label) pairs,

$$D = \{(x_i, y_i)\}_{i=1,2,\ldots,N}$$

  let the machine learn a function from data to label

$$y_i \simeq f_\theta(x_i)$$

- Learning is referred to as updating the parameter θ
- Learning objective: make the prediction close to the ground truth

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, f_\theta(x_i))$$

REVIEW

Unsupervised Learning

- Given the training dataset

$$D = \{x_i\}_{i=1,2,\ldots,N}$$

let the machine learn the data underlying patterns

- Sometimes build latent variables

$$z \longrightarrow x$$

- Estimate the probabilistic density function (p.d.f.)

$$p(x;\theta) = \sum_z p(x|z;\theta)p(z;\theta)$$

- Maximize the log-likelihood of training data

$$\max_\theta \frac{1}{N} \sum_{i=1}^{N} \log p(x;\theta)$$

Two Kinds of Machine Learning

- Prediction
  - Predict the desired output given the data (supervised learning)
  - Generate data instances (unsupervised learning)
  - We mainly covered this category in previous lectures

- Decision Making
  - Take actions based on a particular state in a dynamic environment (reinforcement learning)
    - to transit to new states
    - to receive immediate reward
    - to maximize the accumulative reward over time
  - Learning from interaction

REVIEW

Machine Learning Categories

- Supervised Learning
  - To perform the desired output given the data and labels

$$p(y|x)$$

- Unsupervised Learning
  - To analyze and make use of the underlying data patterns/structures

$$p(x)$$

- Reinforcement Learning
  - To learn a policy of taking actions in a dynamic environment and acquire rewards

$$\pi(a|x)$$

Reinforcement Learning Materials

Our course on RL is mainly based on the materials from these masters.

Prof. Richard Sutton

- University of Alberta, Canada
- http://incompleteideas.net/sutton/index.html
- Reinforcement Learning: An Introduction (2nd edition)
- http://www.incompleteideas.net/book/the-book-2nd.html

Dr. David Silver

- Google DeepMind and UCL, UK
- http://www0.cs.ucl.ac.uk/staff/d.silver/web/Home.html
- UCL Reinforcement Learning Course
- http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Prof. Andrew Ng

- Stanford University, US
- http://www.andrewng.org/
- Machine Learning (CS229) Lecture Notes 12: RL
- http://cs229.stanford.edu/materials.html

Content

- Introduction to Reinforcement Learning

- Model-based Reinforcement Learning
  - Markov Decision Process
  - Planning by Dynamic Programming

- Model-free Reinforcement Learning
  - On-policy SARSA
  - Off-policy Q-learning
  - Model-free Prediction and Control

Content

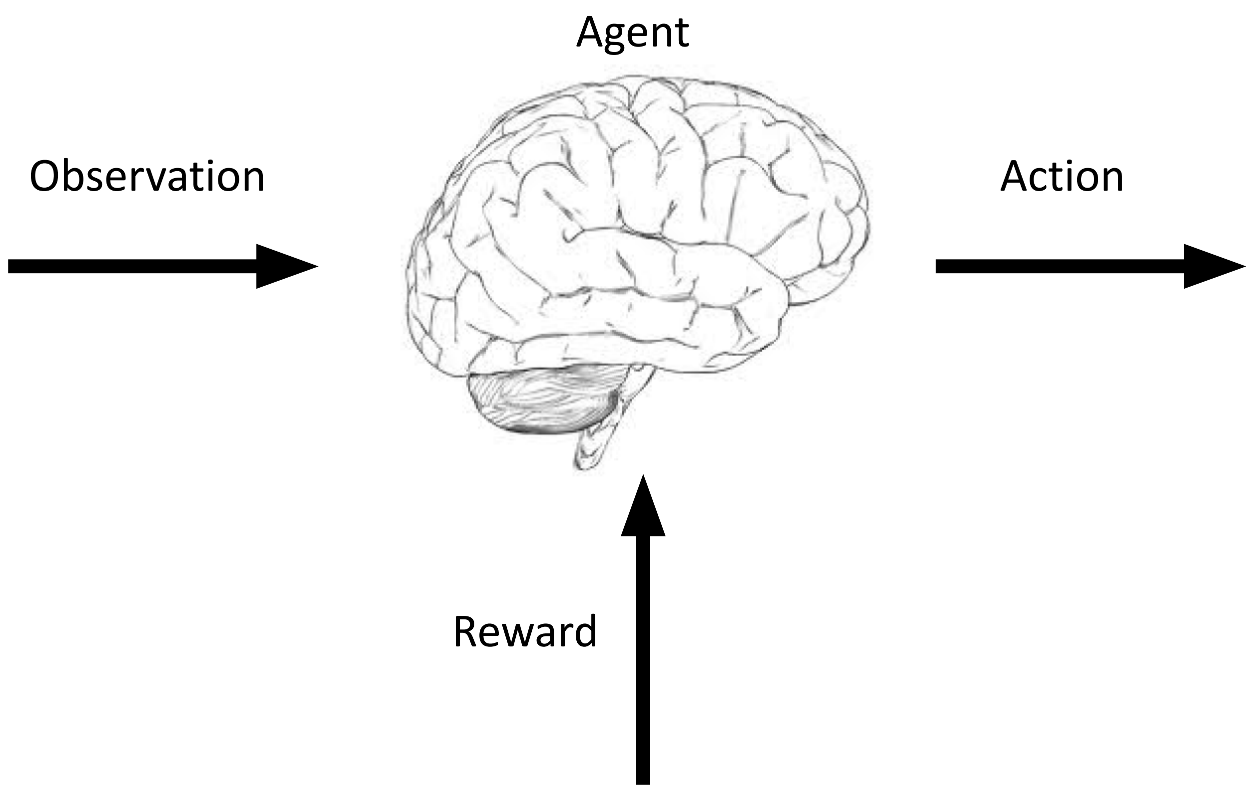
- Introduction to Reinforcement Learning

- Model-based Reinforcement Learning
  - Markov Decision Process
  - Planning by Dynamic Programming

- Model-free Reinforcement Learning
  - On-policy SARSA
  - Off-policy Q-learning
  - Model-free Prediction and Control

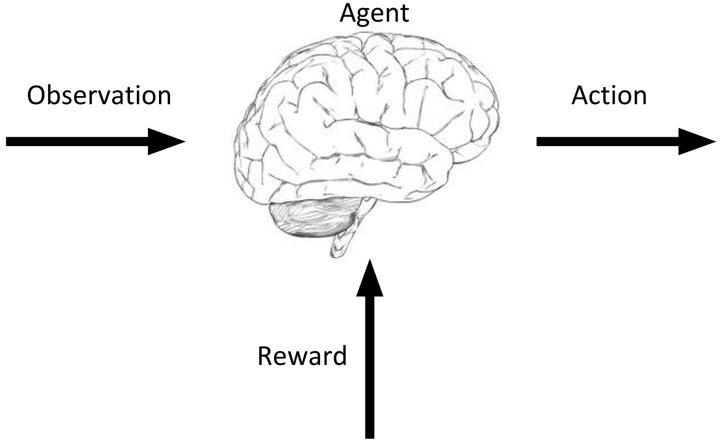
Reinforcement Learning

- Learning from interaction
  - Given the current situation, what to do next in order to maximize utility?



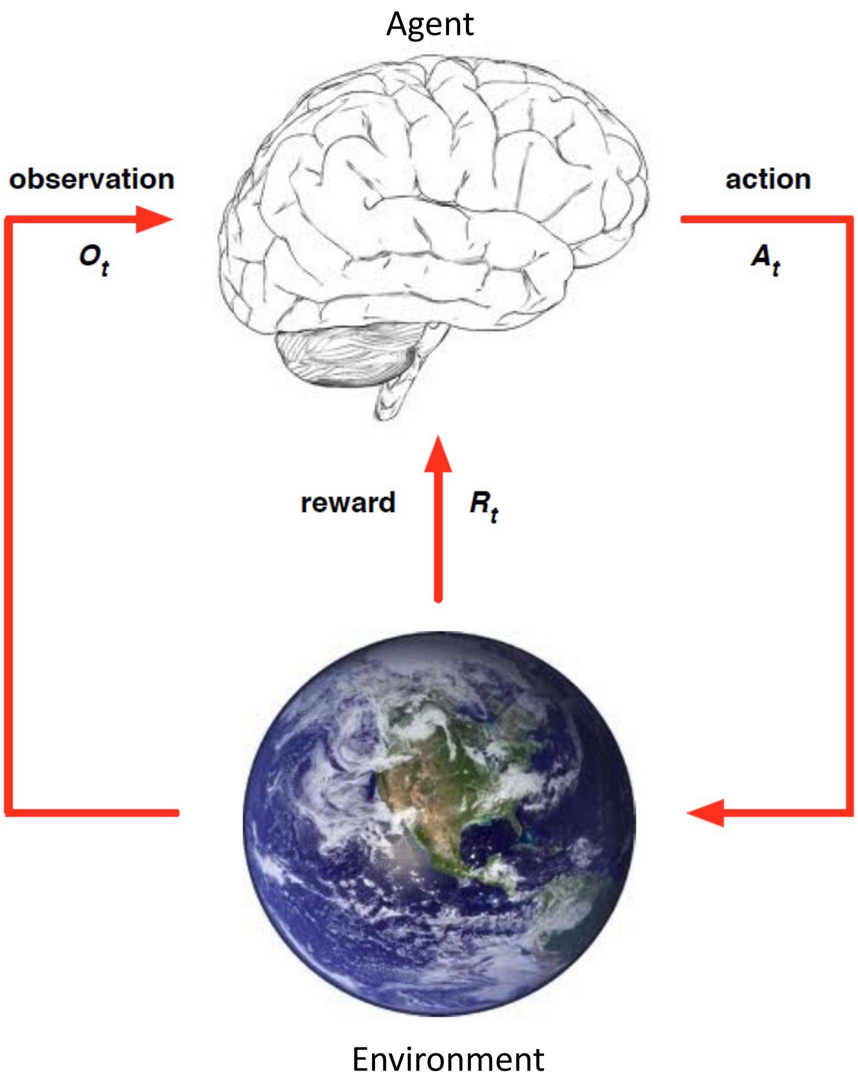
Reinforcement Learning Definition

- A computational approach by learning from interaction to achieve a goal



- Three aspects
  - Sensation: sense the state of the environment to some extent
  - Action: able to take actions that affect the state and achieve the goal
  - Goal: maximize the cumulative reward over time

Reinforcement Learning



- At each step t, the agent
  - Receives observation O_t
  - Receives scalar reward R_t
  - Executes action A_t
- The environment
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}
- t increments at environment step

Elements of RL Systems

- History is the sequence of observations, action, rewards

$$H_t = O_1, R_1, A_1, O_2, R_2, A_2, \dots, O_{t-1}, R_{t-1}, A_{t-1}, O_t, R_t$$

  - i.e. all observable variables up to time *t*
  - E.g., the sensorimotor stream of a robot or embodied agent

- What happens next depends on the history:
  - The agent selects actions
  - The environment selects observations/rewards
- State is the information used to determine what happens next (actions, observations, rewards)
- Formally, state is a function of the history

$$S_t = f(H_t)$$

Elements of RL Systems

- Policy is the learning agent's way of behaving at a given time
  - It is a map from state to action
  - Deterministic policy

$$a = \pi(s)$$

  - Stochastic policy

$$\pi(a|s) = P(A_t = a | S_t = s)$$

Elements of RL Systems

- Reward
  - A scalar defining the goal in an RL problem
  - For immediate sense of what is good

- Value function
  - State value is a scalar specifying what is good in the long run
  - Value function is a prediction of the cumulative future reward
    - Used to evaluate the goodness/badness of states (given the current policy)

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]$$

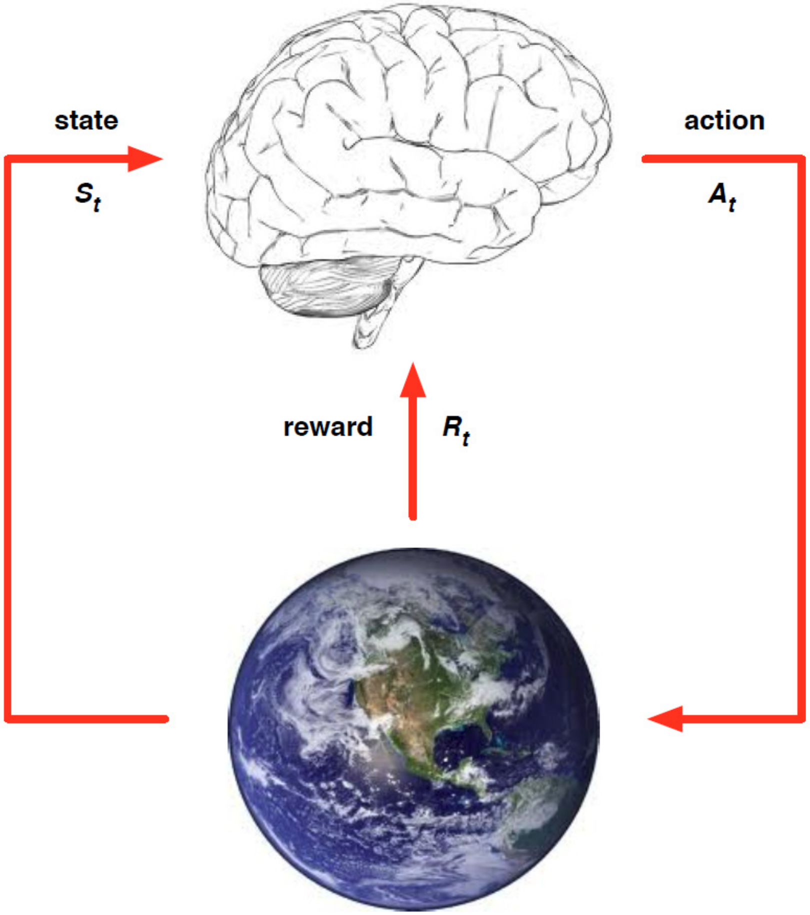
Elements of RL Systems

- A Model of the environment that mimics the behavior of the environment
  - Predict the next state

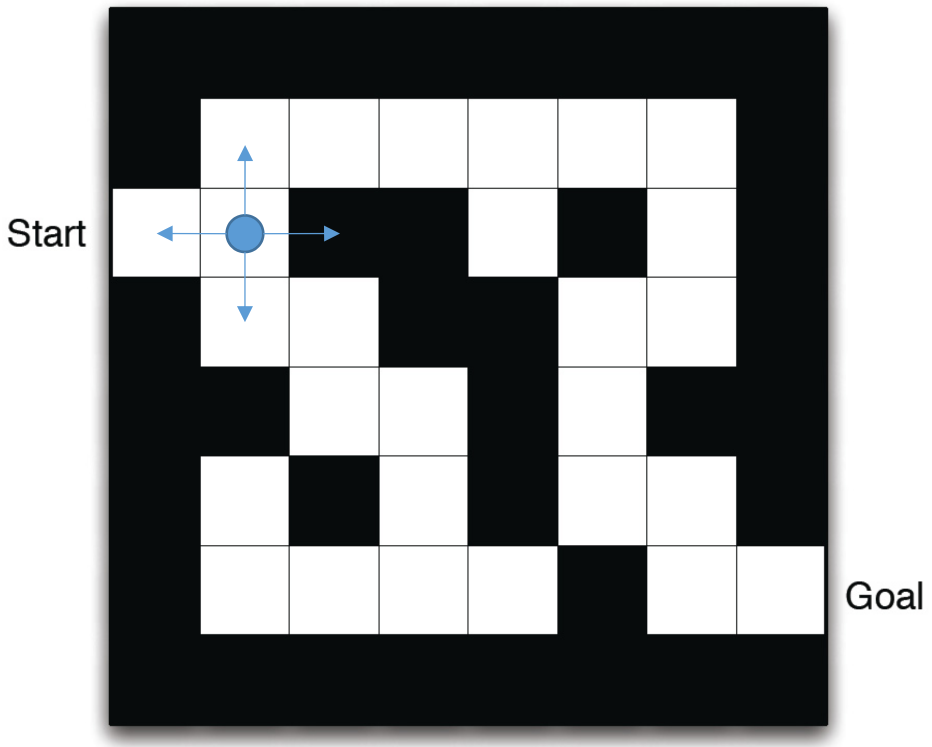
$$\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

  - Predicts the next (immediate) reward

$$\mathcal{R}_{s}^{a} = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

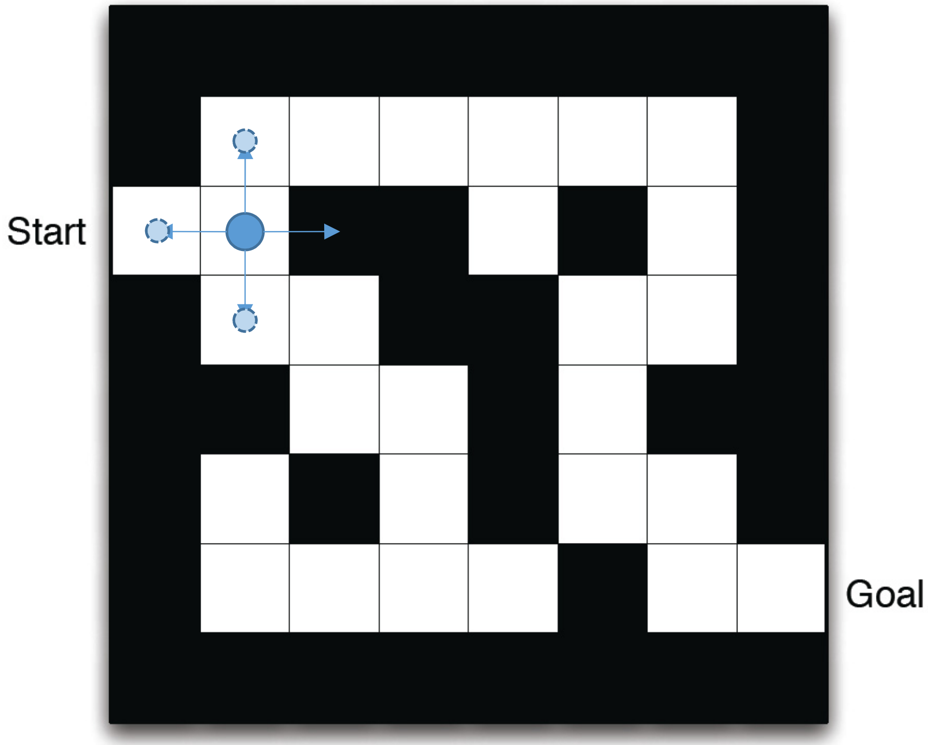


Maze Example



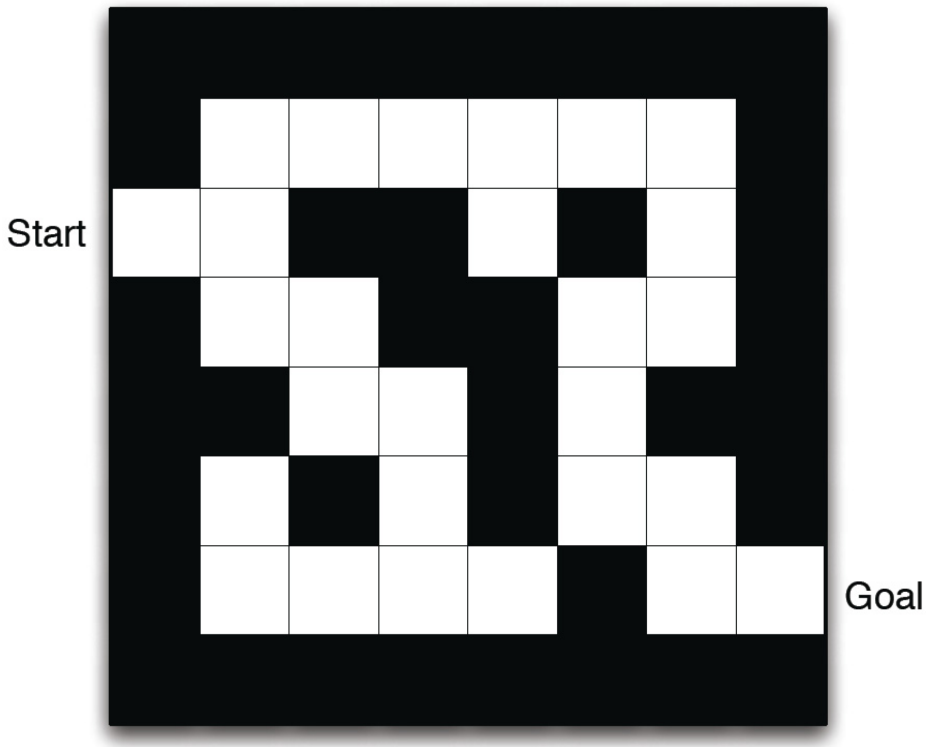
- State: agent's location
- Action: N,E,S,W

Maze Example



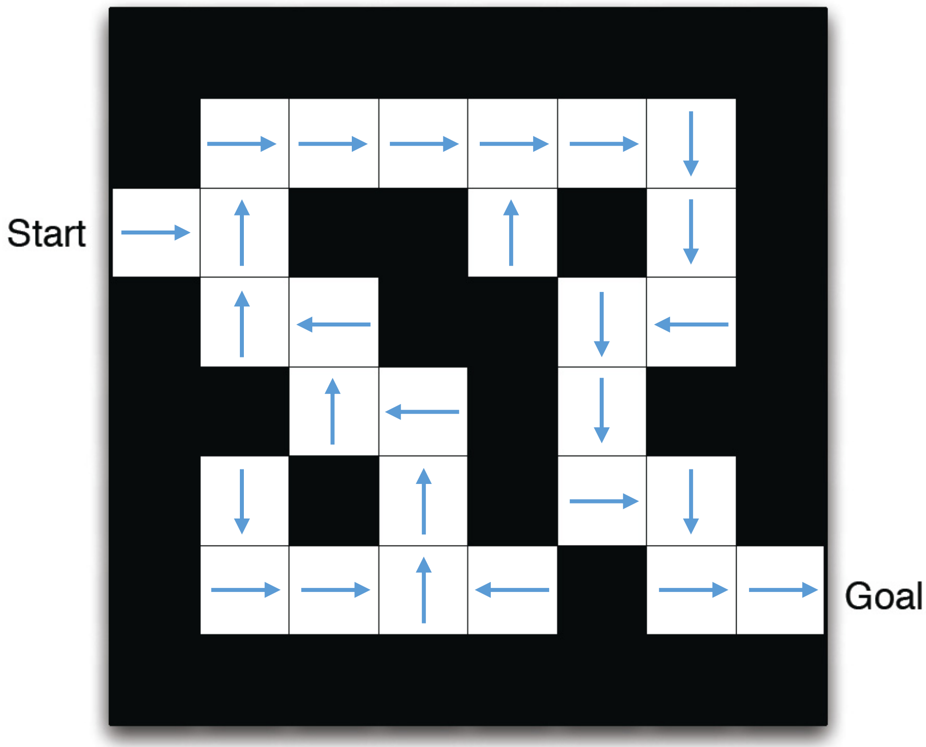
- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
  - No move if the action is to the wall

Maze Example



- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
- Reward: -1 per time step

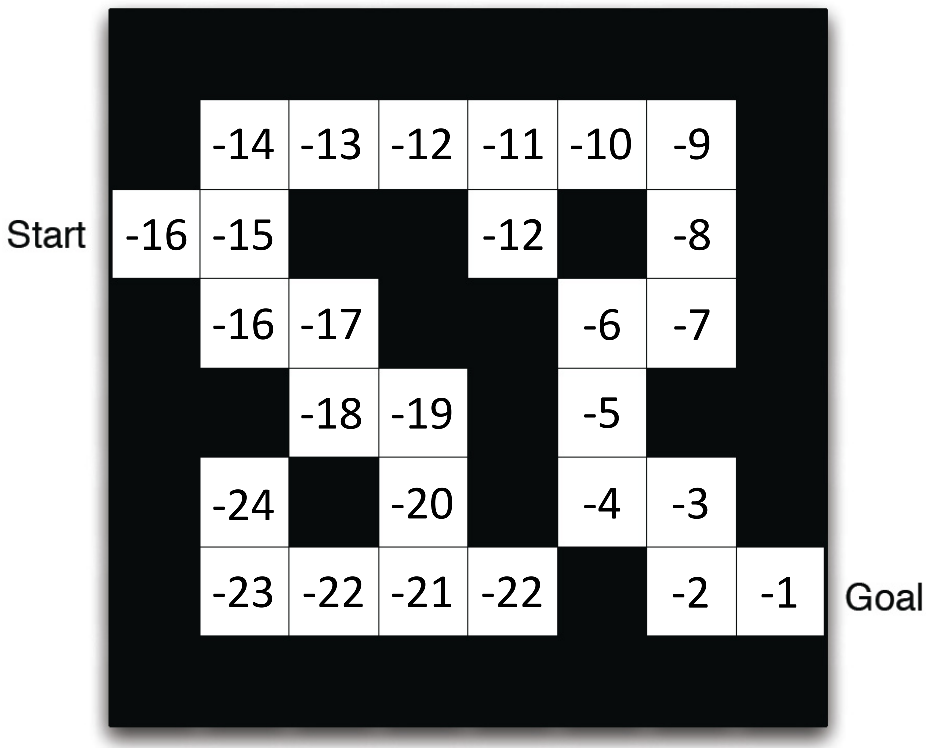
Maze Example



- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
- Reward: -1 per time step

- Given a policy as shown above
    - Arrows represent policy $\pi(s)$ for each state s

Maze Example



- State: agent's location
- Action: N,E,S,W
- State transition: move to the next grid according to the action
- Reward: -1 per time step

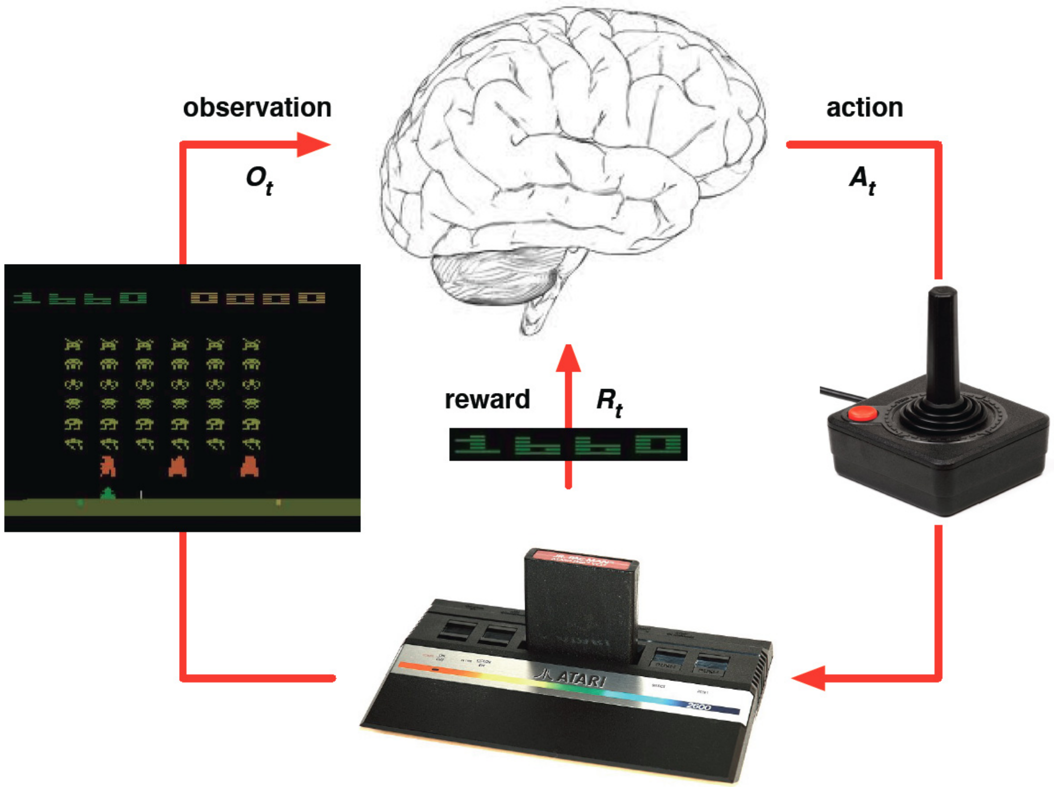
- Numbers represent value $v_\pi(s)$ of each state s

Categorizing RL Agents

- Model based RL
  - Policy and/or value function
  - Model of the environment
  - E.g., the maze game above, game of Go

- Model-free RL
  - Policy and/or value function
  - No model of the environment
  - E.g., general playing Atari games

Atari Example



- Rules of the game are unknown
- Learn from interactive game-play
- Pick actions on joystick, see pixels and scores

Categorizing RL Agents

- Value based
  - No policy (implicit)
  - Value function

- Policy based
  - Policy
  - No value function

- Actor Critic
  - Policy
  - Value function

Content

- Introduction to Reinforcement Learning
- Model-based Reinforcement Learning
  - Markov Decision Process
  - Planning by Dynamic Programming
- Model-free Reinforcement Learning
  - On-policy SARSA
  - Off-policy Q-learning
  - Model-free Prediction and Control

Markov Decision Process

- Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker.

- MDPs formally describe an environment for RL
  - where the environment is FULLY observable
  - i.e. the current state completely characterizes the process (Markov property)

Markov Property

"The future is independent of the past given the present"

- Definition
  - A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

- Properties
  - The state captures all relevant information from the history
  - Once the state is known, the history may be thrown away
  - i.e. the state is sufficient statistic of the future

Markov Decision Process

- A Markov decision process is a tuple (S, A, $\{P_{sa}\}$, γ, R)
- S is the set of states
  - E.g., location in a maze, or current screen in an Atari game
- A is the set of actions
  - E.g., move N, E, S, W, or the direction of the joystick and the buttons
- P_{sa} are the state transition probabilities
  - For each state $s \in S$ and action $a \in A$, P_{sa} is a distribution over the next state in S
- $\gamma \in [0,1]$ is the discount factor for the future reward
- $R : S \times A \mapsto \mathbb{R}$ is the reward function
  - Sometimes the reward is only assigned to state

Markov Decision Process

The dynamics of an MDP proceeds as

- Start in a state s_0
- The agent chooses some action $a_0 \in A$
- The agent gets the reward $R(s_0,a_0)$
- MDP randomly transits to some successor state $s_1 \sim P_{s_0 a_0}$
- This proceeds iteratively

$$s_0 \xrightarrow[R(s_0,a_0)]{a_0} s_1 \xrightarrow[R(s_1,a_1)]{a_1} s_2 \xrightarrow[R(s_2,a_2)]{a_2} s_3 \cdots$$

- Until a terminal state s_T or proceeds with no end
- The total payoff of the agent is

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \cdots$$

Reward on State Only

- For a large part of cases, reward is only assigned to the state
  - E.g., in maze game, the reward is on the location
  - In game of Go, the reward is only based on the final territory
- The reward function $R(s) : S \mapsto \mathbb{R}$
- MDPs proceed

$$s_0 \xrightarrow[R(s_0)]{a_0} s_1 \xrightarrow[R(s_1)]{a_1} s_2 \xrightarrow[R(s_2)]{a_2} s_3 \cdots$$

- cumulative reward (total payoff)

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

MDP Goal and Policy

- The goal is to choose actions over time to maximize the expected cumulative reward

$$\mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots]$$

- $\gamma \in [0,1]$ is the discount factor for the future reward, which makes the agent prefer immediate reward to future reward
  - In finance case, today's $1 is more valuable than $1 in tomorrow
- Given a particular policy $\pi(s) : S \mapsto A$
  - i.e. take the action $a = \pi(s)$ at state *s*
- Define the value function for π

$$V^{\pi}(s) = \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi]$$

  - i.e. expected cumulative reward given the start state and taking actions according to π

Bellman Equation for Value Function

- Define the value function for π

$$V^\pi(s) = \mathbb{E}[R(s_0) + \underbrace{\gamma R(s_1) + \gamma^2 R(s_2) + \cdots}_{\gamma V^\pi(s_1)} | s_0 = s, \pi]$$

$$= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s')$$ Bellman Equation

Immediate Reward

Time decay

State transition

Value of the next state

Optimal Value Function

- The optimal value function for each state *s* is best possible sum of discounted rewards that can be attained by any policy

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- The Bellman's equation for optimal value function

$$V^*(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- The optimal policy

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

- For every state *s* and every policy π

$$V^*(s) = V^{\pi^*}(s) \geq V^{\pi}(s)$$

Value Iteration & Policy Iteration

- Note that the value function and policy are correlated

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^{\pi}(s')$$

- It is feasible to perform iterative update towards the optimal value function and optimal policy
  - Value iteration
  - Policy iteration

Value Iteration

- For an MDP with finite state and action spaces

$$|S| < \infty, |A| < \infty$$

- Value iteration is performed as

  1. For each state *s*, initialize *V*(*s*) = 0.
  2. Repeat until convergence {

     For each state, update

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s')V(s')$$

  }

- Note that there is no explicit policy in above calculation

Synchronous vs. Asynchronous VI

- Synchronous value iteration stores two copies of value functions

    1. For all *s* in *S*

$$V_{\text{new}}(s) \leftarrow \max_{a \in A} \left(R(s) + \gamma \sum_{s' \in S} P_{sa}(s') V_{\text{old}}(s') \right)$$

    2. Update $V_{\text{old}}(s') \leftarrow V_{\text{new}}(s)$

- In-place asynchronous value iteration stores one copy of value function

    1. For all *s* in *S*

$$V(s) \leftarrow \max_{a \in A} \left(R(s) + \gamma \sum_{s' \in S} P_{sa}(s') V(s') \right)$$

Value Iteration Example: Shortest Path

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Policy Iteration

- For an MDP with finite state and action spaces

$$|S| < \infty, |A| < \infty$$

- Policy iteration is performed as

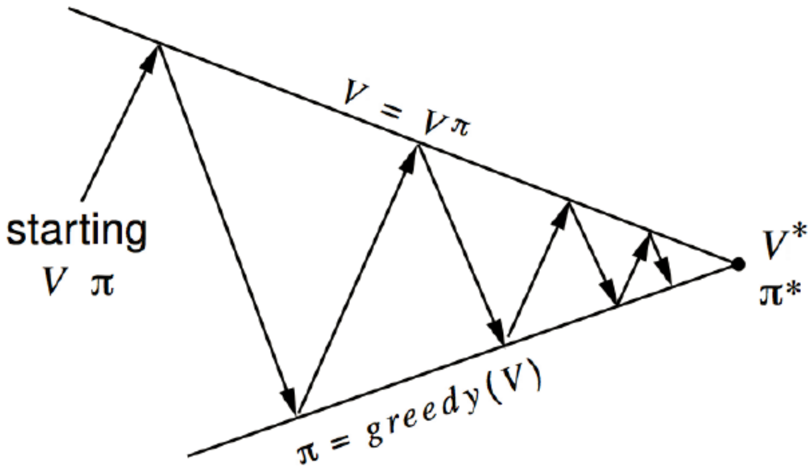
1. Initialize π randomly
2. Repeat until convergence {
   a) Let $V := V^{\pi}$
   b) For each state, update

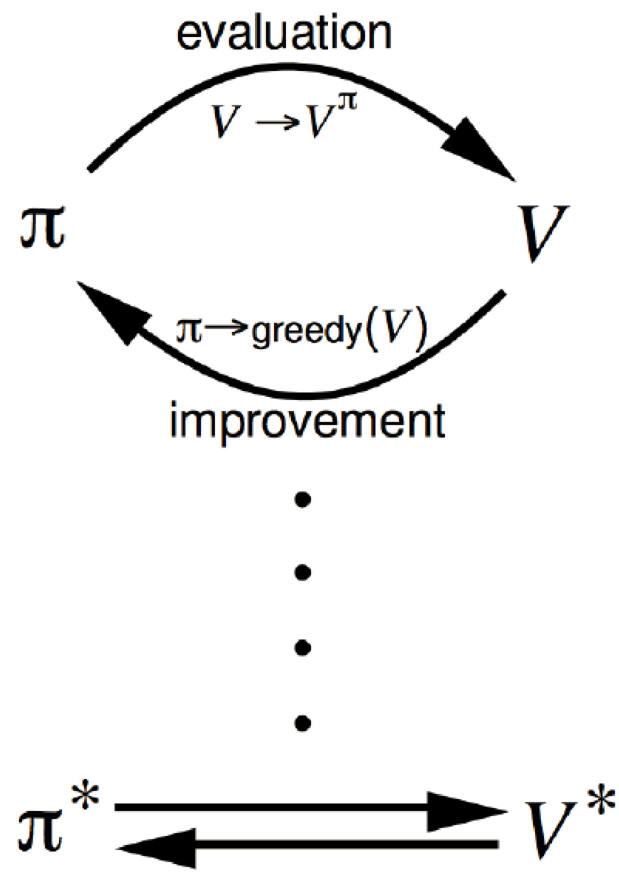
$$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s')V(s')$$

   }

- The step of value function update could be time-consuming

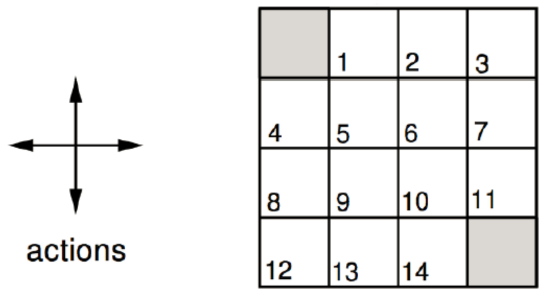
Policy Iteration





- Policy evaluation
  - Estimate V^π
  - Iterative policy evaluation
- Policy improvement
  - Generate $\pi' \geq \pi$
  - Greedy policy improvement

Evaluating a Random Policy in a Small Gridworld



r = -1
on all transitions

- Undiscounted episodic MDP (γ=1)
- Nonterminal states 1,…,14
- Two terminal states (shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows a uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

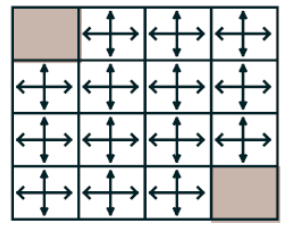
Evaluating a Random Policy in a Small Gridworld

V_k for the random policy

Greedy policy w.r.t. V_k

K=0

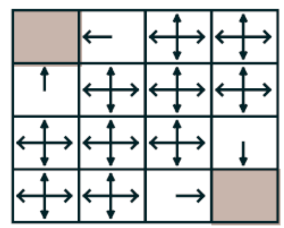
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Random policy

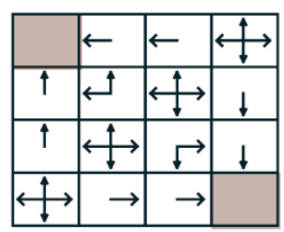
K=1

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



K=2

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



Evaluating a Random Policy in a Small Gridworld

V_k for the random policy

Greedy policy w.r.t. V_k

K=3

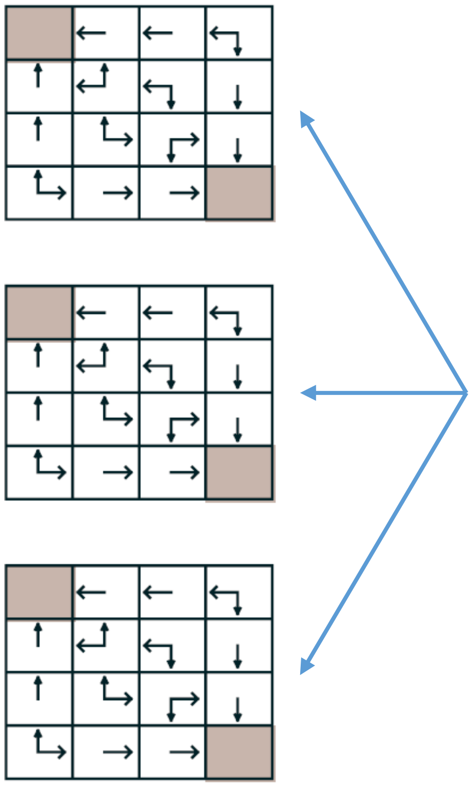
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

K=10

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

K=∞

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



$$V := V^{\pi}$$

Optimal policy

Value Iteration vs. Policy Iteration

Value iteration

1. For each state *s*, initialize *V*(*s*) = 0.
2. Repeat until convergence {

   For each state, update

   $$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s' \in S} P_{sa}(s') V(s')$$

   }

Policy iteration

1. Initialize π randomly
2. Repeat until convergence {
   a) Let $V := V^{\pi}$
   b) For each state, update

   $$\pi(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s')$$

   }

Remarks:

1. Value iteration is a greedy update strategy
2. In policy iteration, the value function update by bellman equation is costly
3. For small-space MDPs, policy iteration is often very fast and converges quickly
4. For large-space MDPs, value iteration is more practical (efficient)
5. If there is no state-transition loop, it is better to use value iteration

My point of view: value iteration is like SGD and policy iteration is like BGD

Learning an MDP Model

- So far we have been focused on
  - Calculating the optimal value function
  - Learning the optimal policy

  given a known MDP model
  - i.e. the state transition $P_{sa}(s')$ and reward function $R(s)$ are explicitly given
- In realistic problems, often the state transition and reward function are not explicitly given
  - For example, we have only observed some episodes

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

Learning an MDP Model

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

⋮ ⋮

- Learn an MDP model from "experience"
  - Learning state transition probabilities $P_{sa}(s')$

$$P_{sa}(s') = \frac{\#\text{times we took action } a \text{ in state } s \text{ and got to state } s'}{\#\text{times we took action } a \text{ in state } s}$$

  - Learning reward $R(s)$, i.e. the expected immediate reward

$$R(s) = \text{average}\left\{R(s)^{(i)}\right\}$$

Learning Model and Optimizing Policy

- Algorithm

1. Initialize π randomly.
2. Repeat until convergence {
   a) Execute π in the MDP for some number of trials
   b) Using the accumulated experience in the MDP, update our estimates for P_{sa} and R
   c) Apply value iteration with the estimated P_{sa} and R to get the new estimated value function V
   d) Update π to be the greedy policy w.r.t. V

   }

Learning an MDP Model

- In realistic problems, often the state transition and reward function are not explicitly given
  - For example, we have only observed some episodes

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

- Another branch of solution is to directly learning value & policy from experience without building an MDP
- i.e. Model-free Reinforcement Learning

Content

- Introduction to Reinforcement Learning

- Model-based Reinforcement Learning
  - Markov Decision Process
  - Planning by Dynamic Programming

- Model-free Reinforcement Learning
  - Model-free Prediction
    - Monte-Carlo and Temporal Difference
  - Model-free Control
    - On-policy SARSA and off-policy Q-learning

Content

- Introduction to Reinforcement Learning
- Model-based Reinforcement Learning
  - Markov Decision Process
  - Planning by Dynamic Programming
- Model-free Reinforcement Learning
  - Model-free Prediction
    - Monte-Carlo and Temporal Difference
  - Model-free Control
    - On-policy SARSA and off-policy Q-learning

Model-free Reinforcement Learning

- In realistic problems, often the state transition and reward function are not explicitly given
  - For example, we have only observed some episodes

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

- Model-free RL is to directly learn value & policy from experience without building an MDP
- Key steps: (1) estimate value function; (2) optimize policy

Value Function Estimation

- In model-based RL (MDP), the value function is calculated by dynamic programming

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi] \\ &= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^\pi(s') \end{aligned}$$

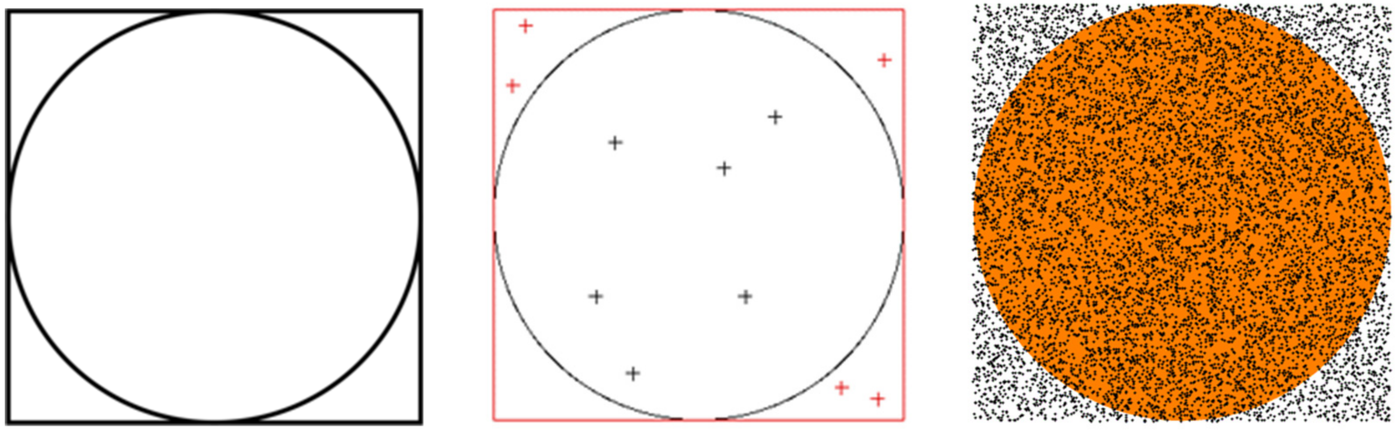
- Now in model-free RL
  - We cannot directly know P_{sa} and R
  - But we have a list of experiences to estimate the values

Episode 1: $s_0^{(1)} \xrightarrow[R(s_0)^{(1)}]{a_0^{(1)}} s_1^{(1)} \xrightarrow[R(s_1)^{(1)}]{a_1^{(1)}} s_2^{(1)} \xrightarrow[R(s_2)^{(1)}]{a_2^{(1)}} s_3^{(1)} \cdots s_T^{(1)}$

Episode 2: $s_0^{(2)} \xrightarrow[R(s_0)^{(2)}]{a_0^{(2)}} s_1^{(2)} \xrightarrow[R(s_1)^{(2)}]{a_1^{(2)}} s_2^{(2)} \xrightarrow[R(s_2)^{(2)}]{a_2^{(2)}} s_3^{(2)} \cdots s_T^{(2)}$

Monte-Carlo Methods

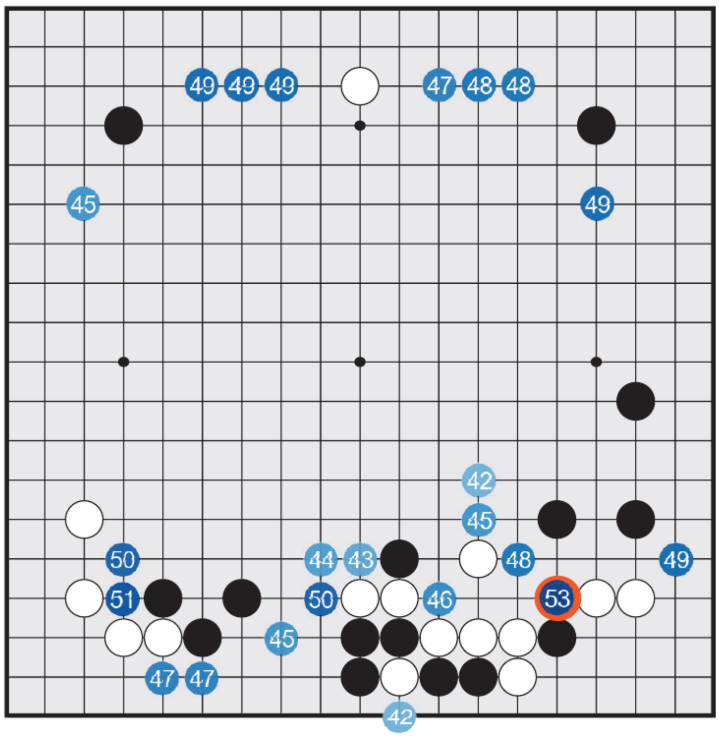
- Monte-Carlo methods are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.
- Example, to calculate the circle's surface

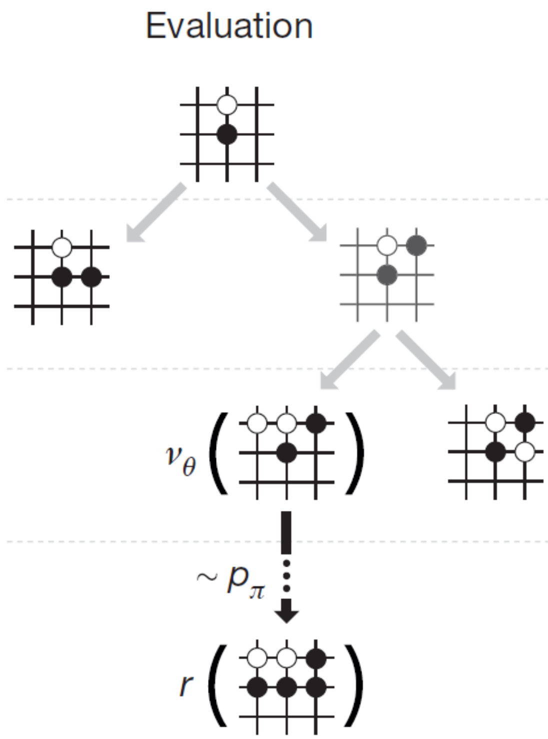


$$\text{Circle Surface} = \text{Square Surface} \times \frac{\#\text{points in circle}}{\#\text{points in total}}$$

Monte-Carlo Methods

- Go: to estimate the winning rate given the current state





$$\text{Win Rate}(s) = \frac{\#\text{win simulation cases started from } s}{\#\text{simulation cases started from } s \text{ in total}}$$

Monte-Carlo Value Estimation

- Goal: learn V^π from episodes of experience under policy π

$$s_0^{(i)} \xrightarrow[R_1^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[R_2^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[R_3^{(i)}]{a_2^{(i)}} s_3^{(i)} \cdots s_T^{(i)} \sim \pi$$

- Recall that the return is the total discounted reward

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

- Recall that the value function is the expected return

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots | s_0 = s, \pi] \\ &= \mathbb{E}[G_t | s_t = s, \pi] \\ &\simeq \frac{1}{N} \sum_{i=1}^{N} G_t^{(i)} \end{aligned}$$

- Sample *N* episodes from state *s* using policy *π*
- Calculate the average of cumulative reward

- Monte-Carlo policy evaluation uses empirical mean return instead of expected return

Monte-Carlo Value Estimation

- Implementation
  - Sample episodes policy π

$$s_0^{(i)} \xrightarrow[R_1^{(i)}]{a_0^{(i)}} s_1^{(i)} \xrightarrow[R_2^{(i)}]{a_1^{(i)}} s_2^{(i)} \xrightarrow[R_3^{(i)}]{a_2^{(i)}} s_3^{(i)} \cdots s_T^{(i)} \sim \pi$$

  - Every time-step *t* that state *s* is visited in an episode
    - Increment counter $N(s) \leftarrow N(s) + 1$
    - Increment total return $S(s) \leftarrow S(s) + G_t$
    - Value is estimated by mean return $V(s) = S(s)/N(s)$
    - By law of large numbers

$$V(s) \rightarrow V^{\pi}(s) \text{ as } N(s) \rightarrow \infty$$

Incremental Monte-Carlo Updates

- Update $V(s)$ incrementally after each episode
- For each state S_t with cumulative return G_t

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

- For non-stationary problems (i.e. the environment could be varying over time), it can be useful to track a running mean, i.e. forget old episodes

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

Monte-Carlo Value Estimation

Idea: $$V(S_t) \simeq \frac{1}{N}\sum_{i=1}^{N} G_t^{(i)}$$

Implementation: $$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

- MC methods learn directly from episodes of experience
- MC is model-free: no knowledge of MDP transitions / rewards
- MC learns from **complete** episodes: no bootstrapping (discussed later)
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to episodic MDPs
  - All episodes must terminate

Temporal-Difference Learning

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = R_{t+1} + \gamma V(S_{t+1})$$

$$V(S_t) \leftarrow V(S_t) + \alpha(\underbrace{R_{t+1}}_{\text{Observation}} + \gamma \underbrace{V(S_{t+1})}_{\text{Guess of future}} - V(S_t))$$

- 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

Monte Carlo vs. Temporal Difference

- The same goal: learn V^{π} from episodes of experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t

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

- Simplest temporal-difference learning algorithm: TD
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$

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

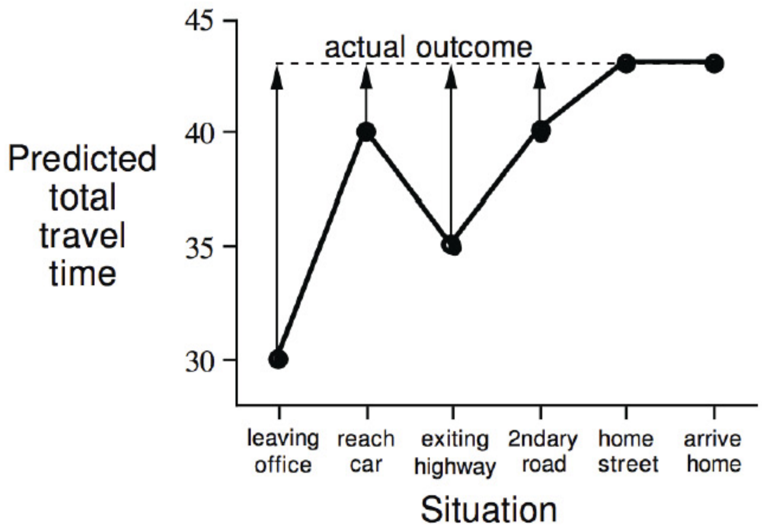
  - TD target: $R_{t+1} + \gamma V(S_{t+1})$
  - TD error: $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

Driving Home Example

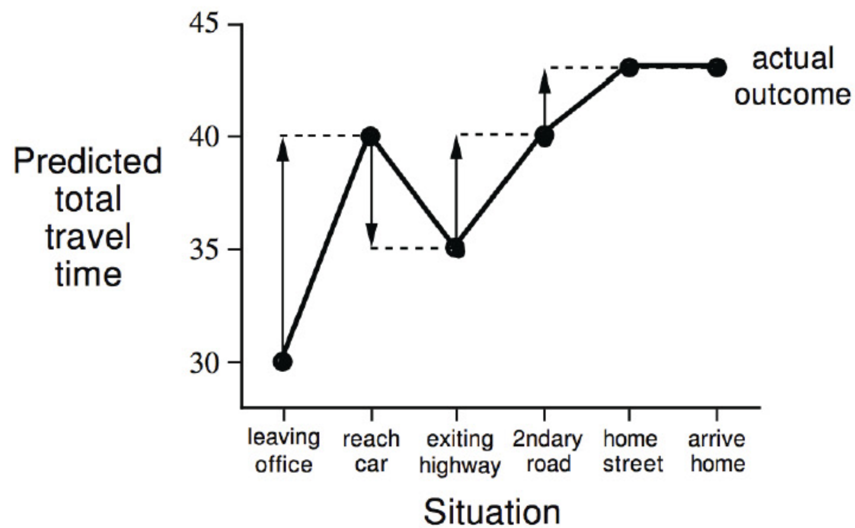
State	Elapsed Time (Minutes)	Predicted Time to Go	Predicted Total Time
Leaving office	0	30	30
Reach car, raining	5	35	40
Exit highway	20	15	35
Behind truck	30	10	40
Home street	40	3	43
Arrow home	43	0	43

Driving Home Example: MC vs. TD

Changes recommended by Monte Carlo methods (α=1)



Changes recommended by TD methods (α=1)



Advantages and Disadvantages of MC vs. TD

- TD can learn before knowing the final outcome
  - TD can learn online after every step
  - MC must wait until end of episode before return is known

- TD can learn without the final outcome
  - TD can learn from incomplete sequences
  - MC can only learn from complete sequences
  - TD works in continuing (non-terminating) environments
  - MC only works for episodic (terminating) environments

Bias/Variance Trade-Off

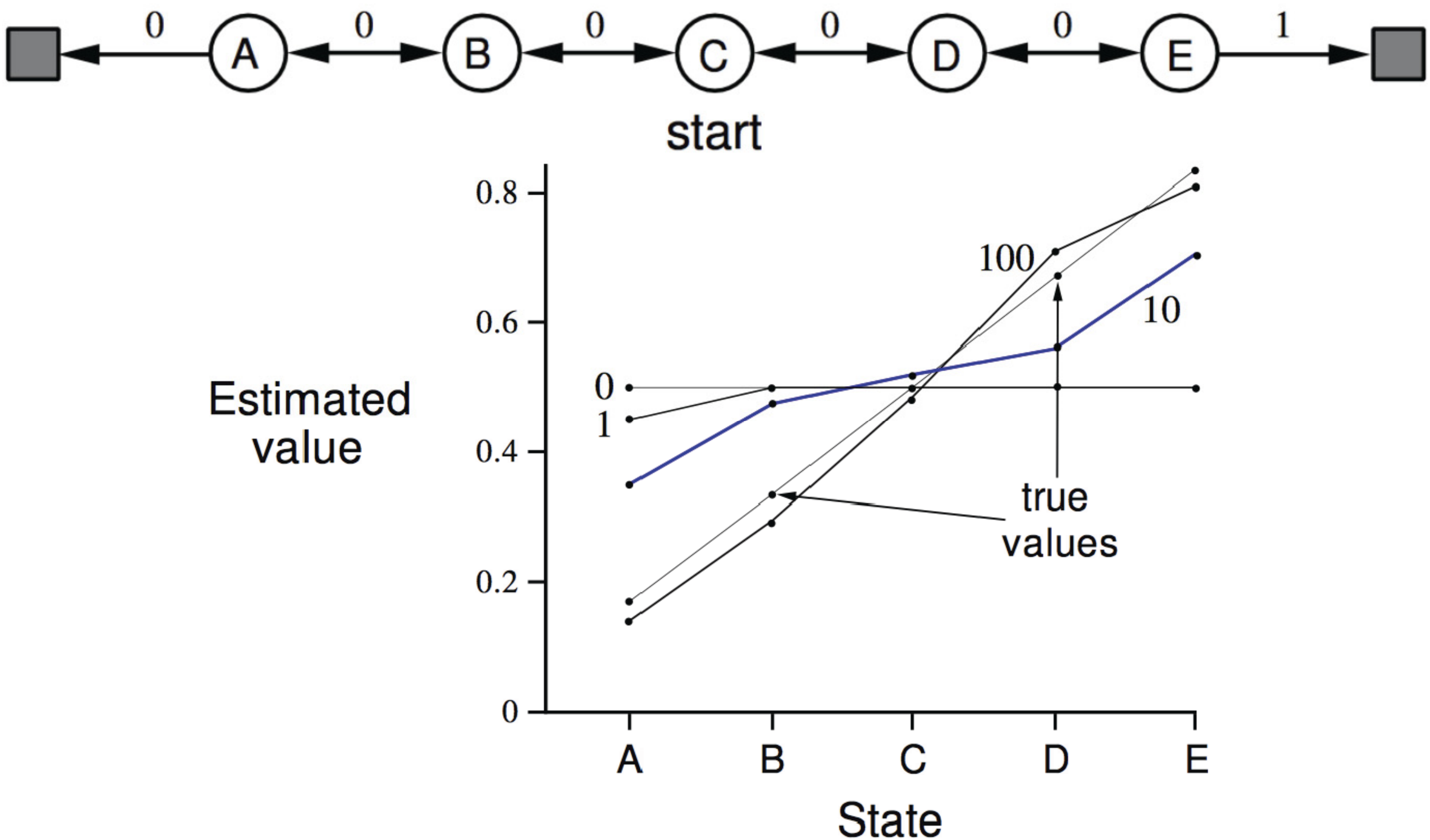
- Return $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$ is unbiased estimate of $V^\pi(S_t)$
- True TD target $R_{t+1} + \gamma V^\pi(S_{t+1})$ is unbiased estimate of $V^\pi(S_t)$
- TD target $R_{t+1} + \gamma \underbrace{V(S_{t+1})}_{\text{current estimate}}$ is biased estimate of $V^\pi(S_t)$
- TD target is of much lower variance than the return
  - Return depends on many random actions, transitions and rewards
  - TD target depends on one random action, transition and reward

Advantages and Disadvantages of MC vs. TD (2)

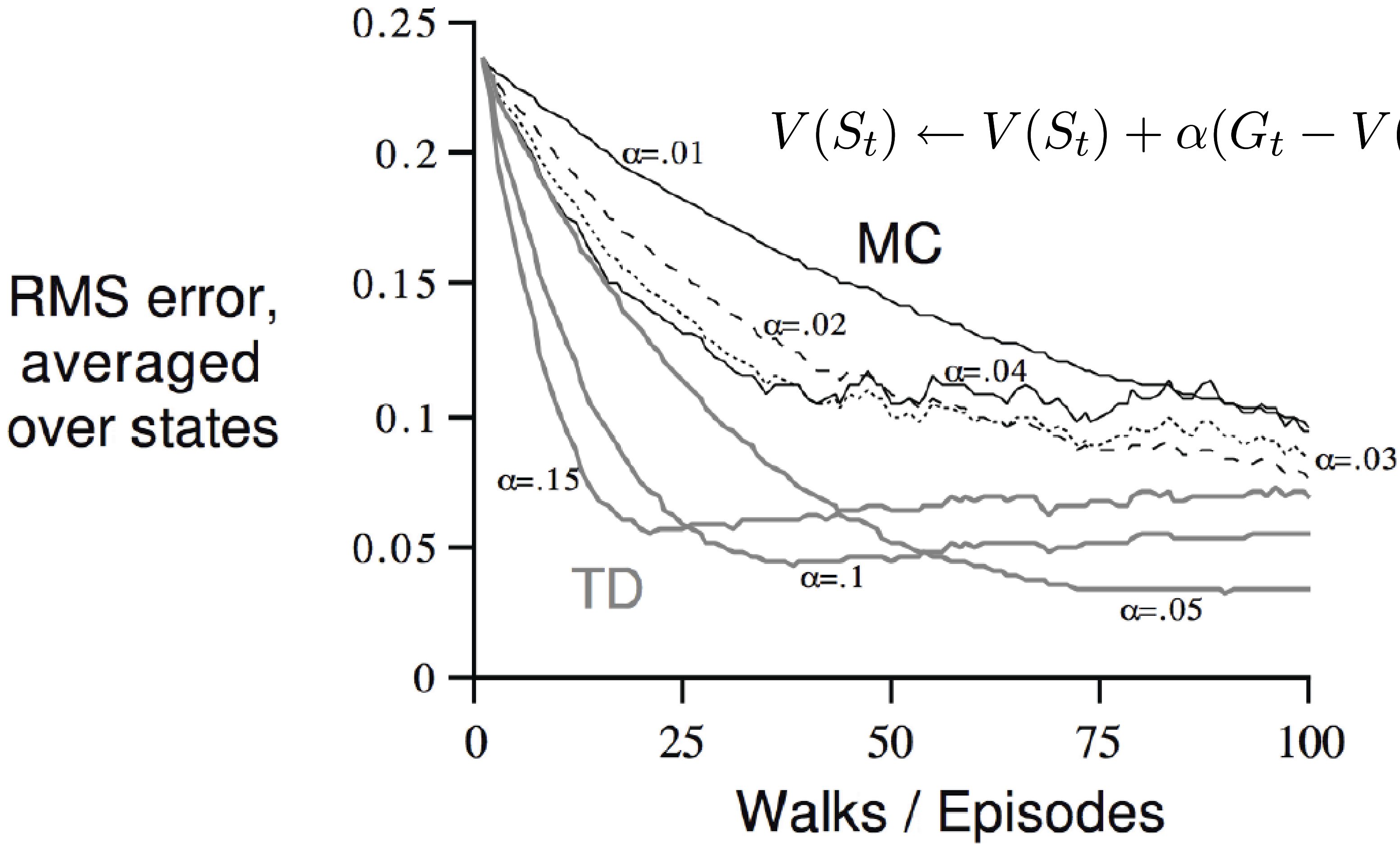
- MC has high variance, zero bias
  - Good convergence properties
  - (even with function approximation)
  - Not very sensitive to initial value
  - Very simple to understand and use

- TD has low variance, some bias
  - Usually more efficient than MC
  - TD converges to $V^{\pi}(S_t)$
    - (but not always with function approximation)
  - More sensitive to initial value than MC

Random Walk Example



Random Walk Example

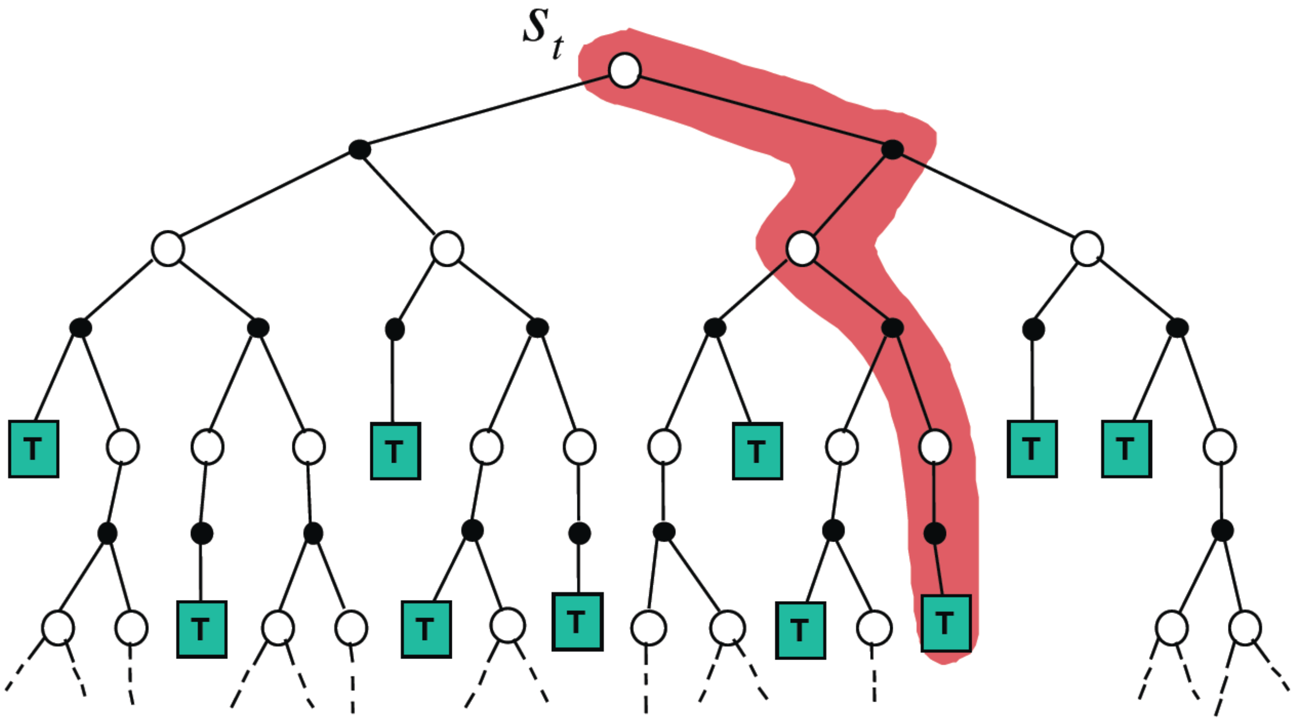


$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$

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

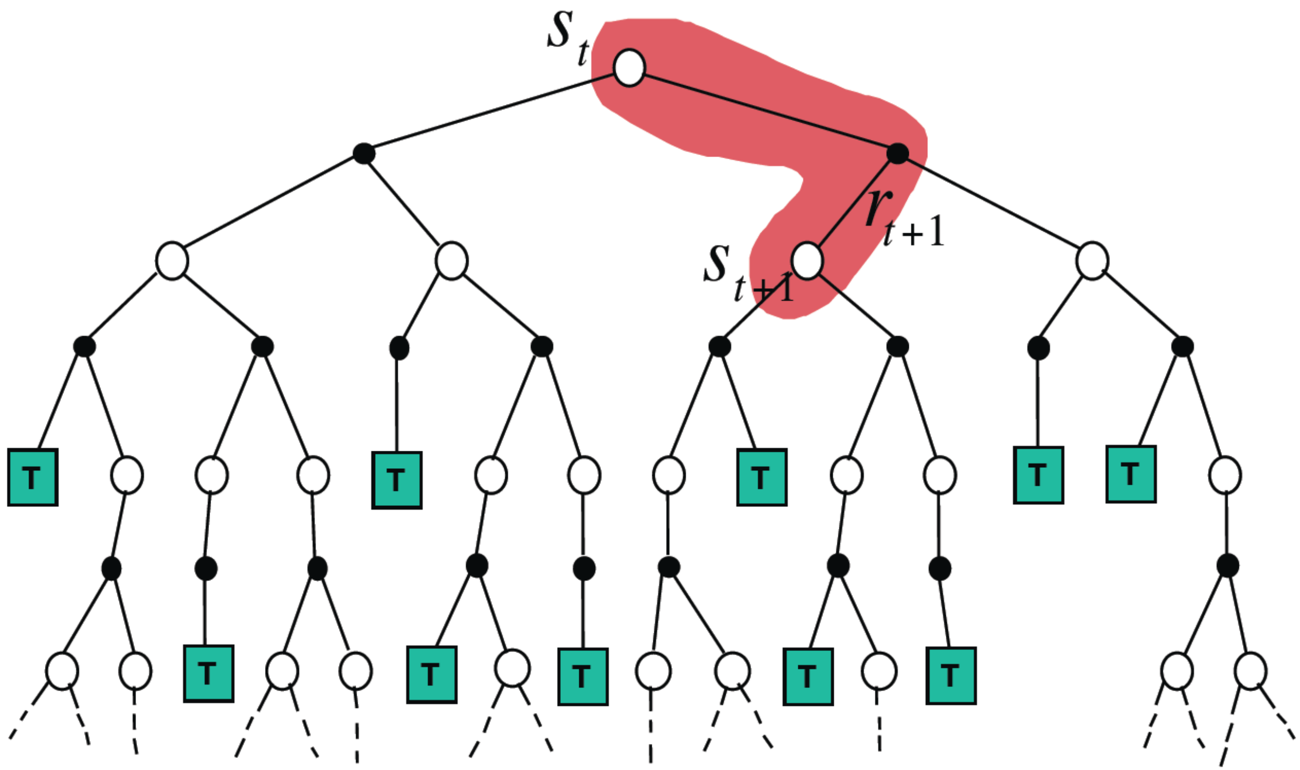
Monte-Carlo Backup

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



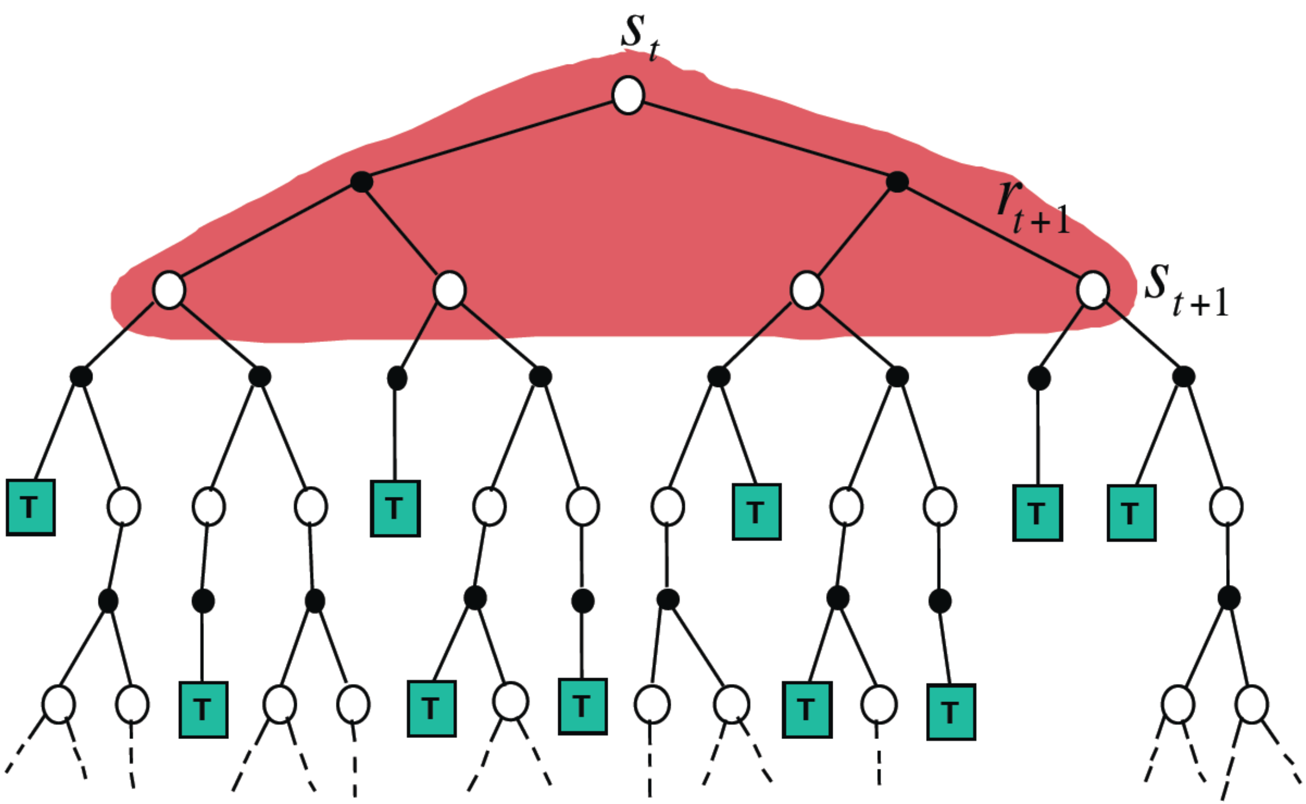
Temporal-Difference Backup

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



Dynamic Programming Backup

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



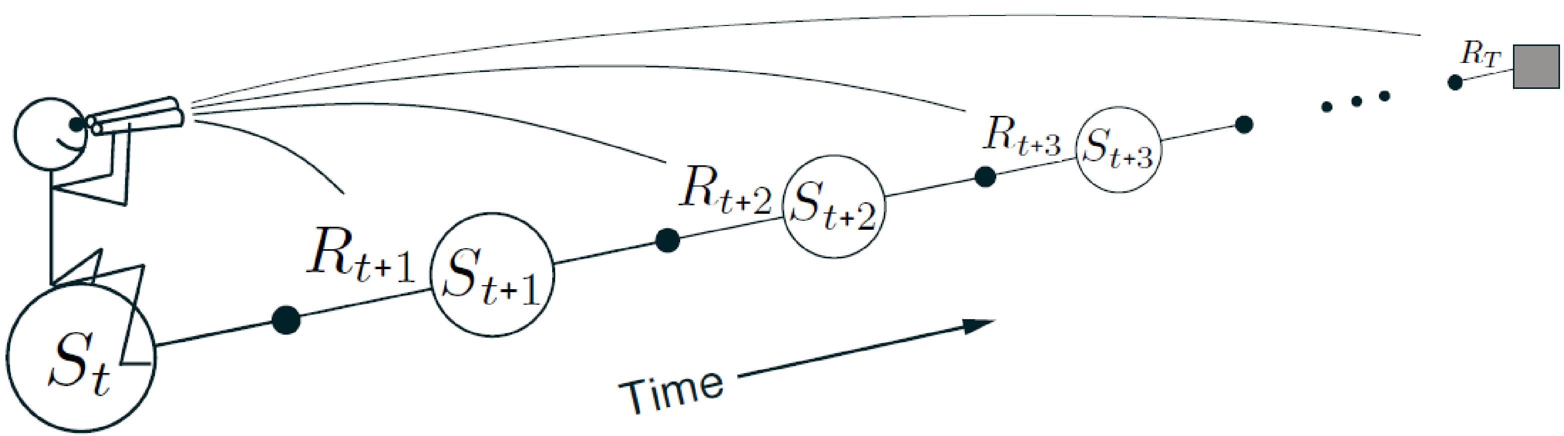
- For time constraint, we may jump n-step prediction section and directly head to model-free control

- Define the *n*-step return

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

- *n*-step temporal-difference learning

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t))$$



Content

- Introduction to Reinforcement Learning

- Model-based Reinforcement Learning
  - Markov Decision Process
  - Planning by Dynamic Programming

- Model-free Reinforcement Learning
  - Model-free Prediction
    - Monte-Carlo and Temporal Difference
  - Model-free Control
    - On-policy SARSA and off-policy Q-learning

Uses of Model-Free Control

- Some example problems that can be modeled as MDPs
  - Elevator
  - Parallel parking
  - Ship steering
  - Bioreactor
  - Helicopter
  - Aeroplane logistics
  - Robocup soccer
  - Atari & StarCraft
  - Portfolio management
  - Protein folding
  - Robot walking
  - Game of Go
- For most of real-world problems, either:
  - MDP model is unknown, but experience can be sampled
  - MDP model is known, but is too big to use, except by samples
- Model-free control can solve these problems

On- and Off-Policy Learning

- Two categories of model-free RL

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π

- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from another policy μ

State Value and Action Value

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

- State value
  - The state-value function $V^\pi(s)$ of an MDP is the expected return starting from state s and then following policy π

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- Action value
  - The action-value function $Q^\pi(s,a)$ of an MDP is the expected return starting from state s, taking action a, and then following policy π

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Bellman Expectation Equation

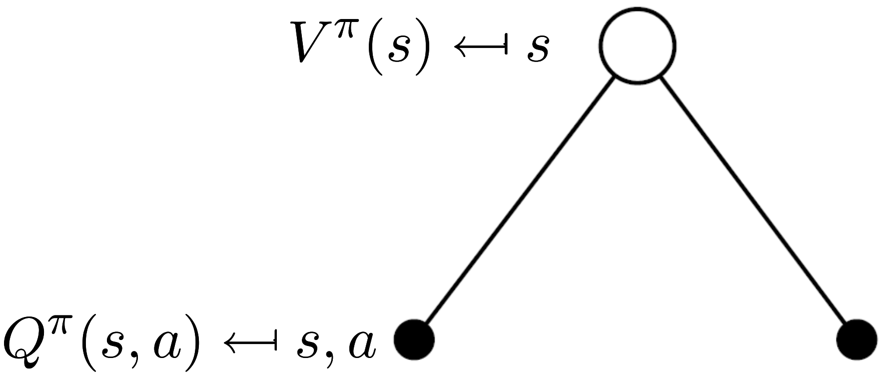
- The state-value function $V^{\pi}(s)$ can be decomposed into immediate reward plus discounted value of successor state

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

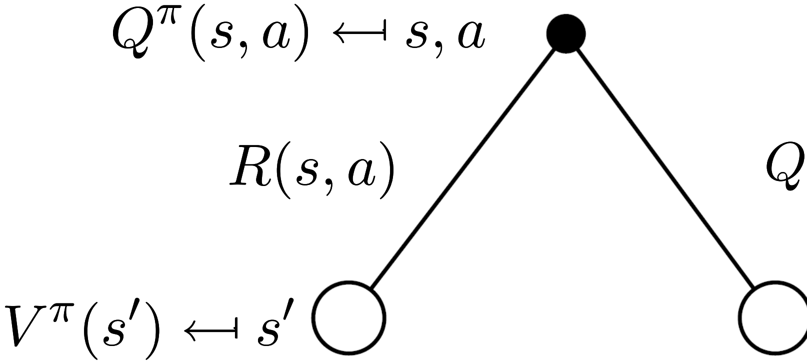
- The action-value function $Q^{\pi}(s,a)$ can similarly be decomposed

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma Q^{\pi}(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

State Value and Action Value



$$V^\pi(s) = \sum_{a \in A} \pi(a|s) Q^\pi(s,a)$$



$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} P_{sa}(s') V^\pi(s')$$

Model-Free Policy Iteration

- Given state-value function $V(s)$ and action-value function $Q(s,a)$, model-free policy iteration shall use action-value function

- Greedy policy improvement over $V(s)$ requires model of MDP

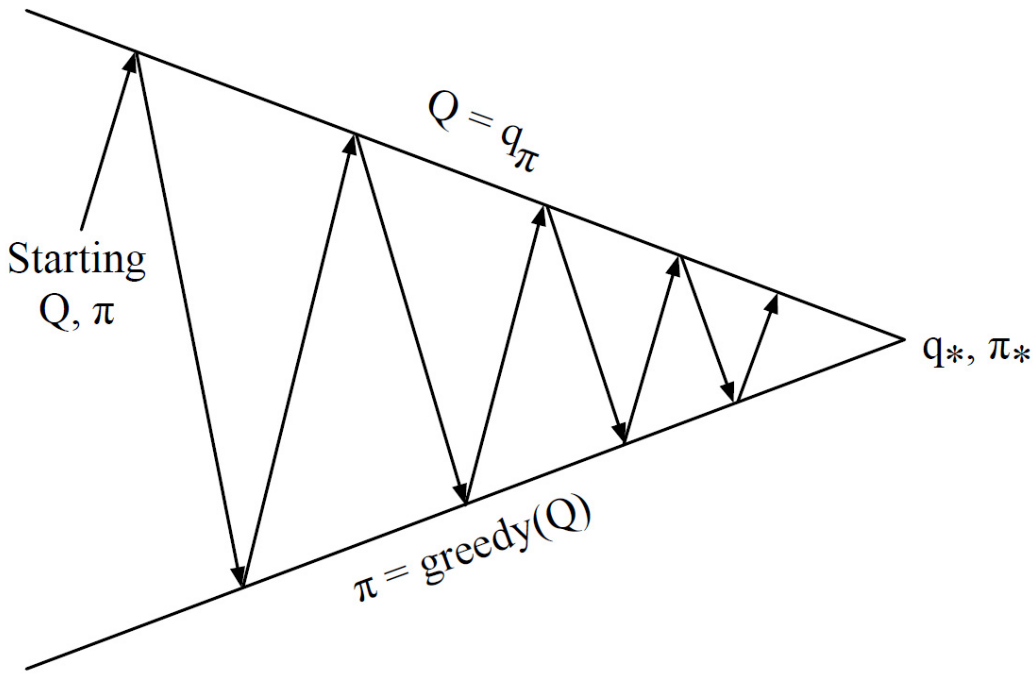
$$\pi^{\text{new}}(s) = \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} P_{sa}(s') V^{\pi}(s') \right\}$$

We don't know the transition probability

- Greedy policy improvement over $Q(s,a)$ is model-free

$$\pi^{\text{new}}(s) = \arg\max_{a \in A} Q(s,a)$$

Generalized Policy Iteration with Action-Value Function



- Policy evaluation: Monte-Carlo policy evaluation, $Q = Q^\pi$
- Policy improvement: Greedy policy improvement?

Example of Greedy Action Selection

- Greedy policy improvement over $Q(s,a)$ is model-free

$$\pi^{\text{new}}(s) = \arg\max_{a \in A} Q(s,a)$$

- Given the right example
  - What if the first action is to choose the left door and observe reward=0?
  - The policy would be suboptimal if there is no exploration

Left:
20% Reward = 0
80% Reward = 5

Right:
50% Reward = 1
50% Reward = 3

"Behind one door is tenure – behind the other is flipping burgers at McDonald's."

ε-Greedy Policy Exploration

- Simplest idea for ensuring continual exploration
- All *m* actions are tried with non-zero probability
- With probability 1-ε, choose the greedy action
- With probability ε, choose an action at random

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

ε-Greedy Policy Improvement

- Theorem
  - For any *ε*-greedy policy *π*, the *ε*-greedy policy *π'* w.r.t. Q^π is an improvement, i.e. $V^{\pi'}(s) \geq V^\pi(s)$

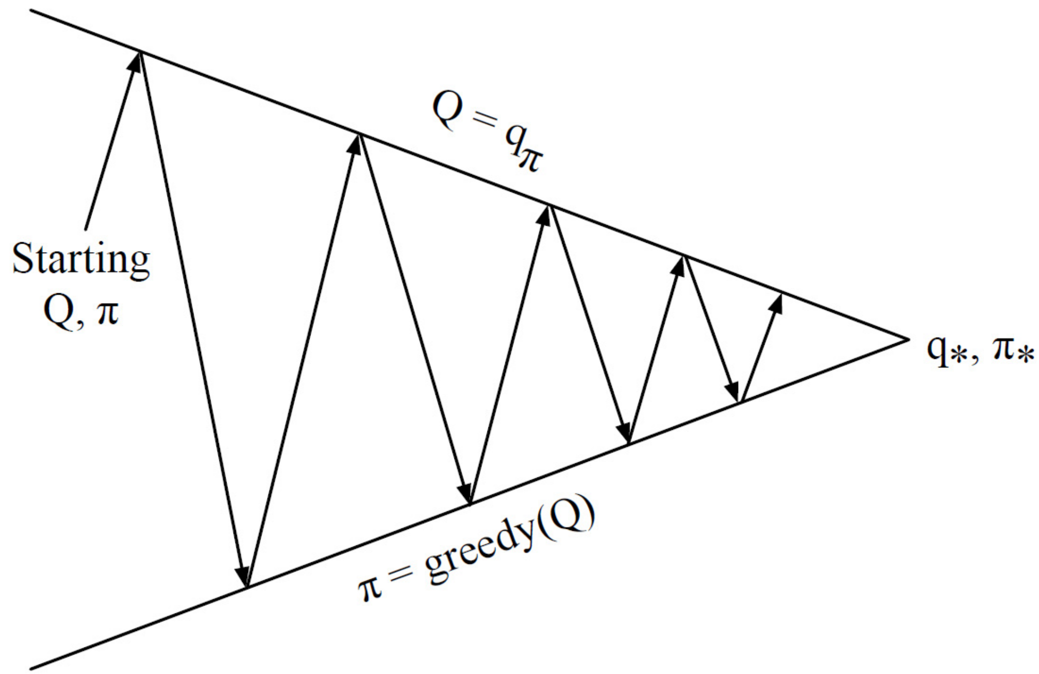
$$V^{\pi'}(s) = Q^\pi(s, \pi'(s)) = \sum_{a \in A} \pi'(a|s) Q^\pi(s,a)$$

m actions
$$= \frac{\epsilon}{m} \sum_{a \in A} Q^\pi(s,a) + (1-\epsilon) \max_{a \in A} Q^\pi(s,a)$$

$$\geq \frac{\epsilon}{m} \sum_{a \in A} Q^\pi(s,a) + (1-\epsilon) \sum_{a \in A} \frac{\pi(a|s) - \epsilon/m}{1-\epsilon} Q^\pi(s,a)$$

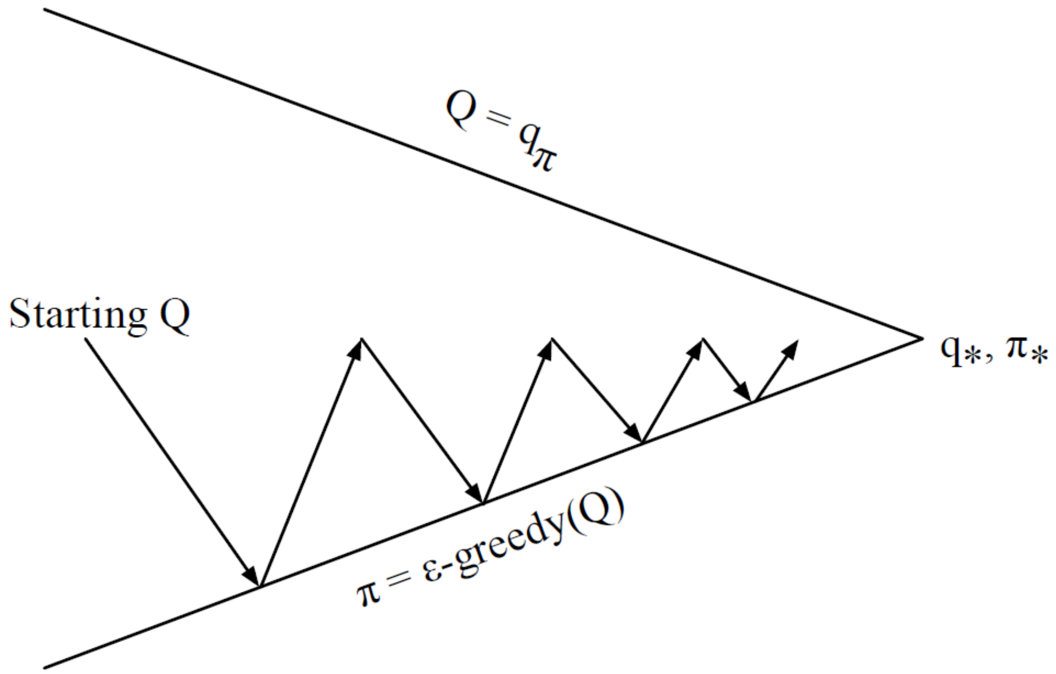
$$= \sum_{a \in A} \pi(a|s) Q^\pi(s,a) = V^\pi(s)$$

Generalized Policy Iteration with Action-Value Function



- Policy evaluation: Monte-Carlo policy evaluation, $Q = Q^\pi$
- Policy improvement: ε-greedy policy improvement

Monte-Carlo Control



Every episode:

- Policy evaluation: Monte-Carlo policy evaluation, $Q \approx Q^\pi$
- Policy improvement: ε-greedy policy improvement

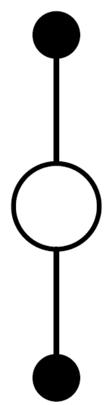
MC Control vs. TD Control

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences

- Natural idea: use TD instead of MC in our control loop
  - Apply TD to update action value $Q(s,a)$
  - Use ε-greedy policy improvement
  - Update the action value function every time-step

SARSA

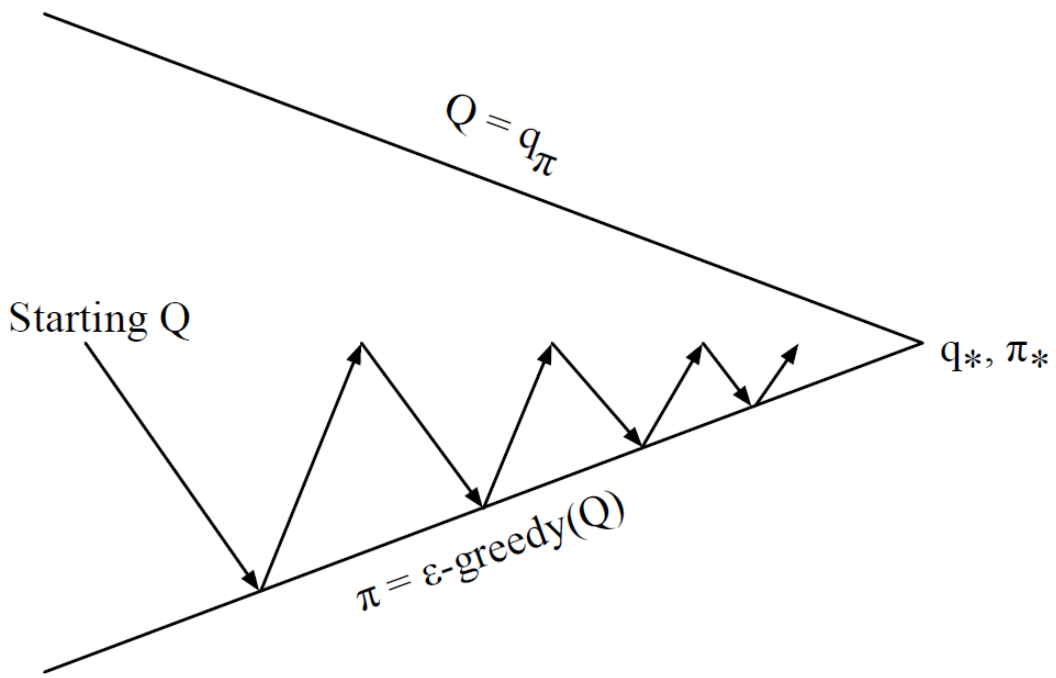
- For each state-action-reward-state-action by the current policy



- Updating action-value functions with Sarsa

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

On-Policy Control with SARSA



Every time-step:

- Policy evaluation: Sarsa $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$
- Policy improvement: ε-greedy policy improvement

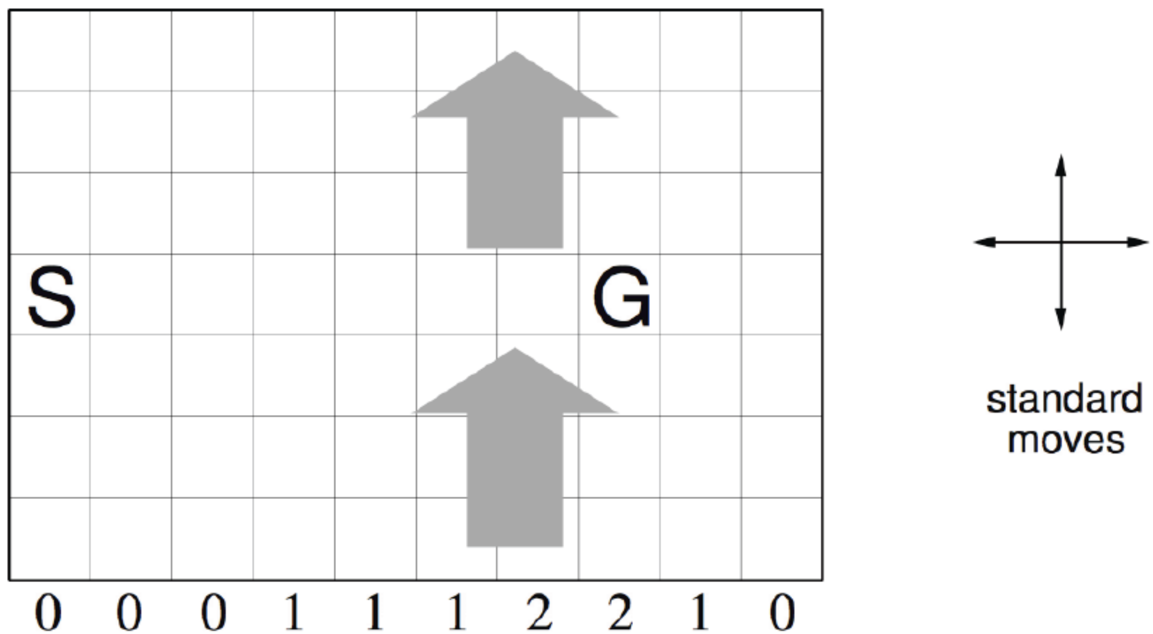
SARSA Algorithm

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, 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

- NOTE: on-policy TD control sample actions by the current policy, i.e., the two ‘A’s in SARSA are both chosen by the current policy

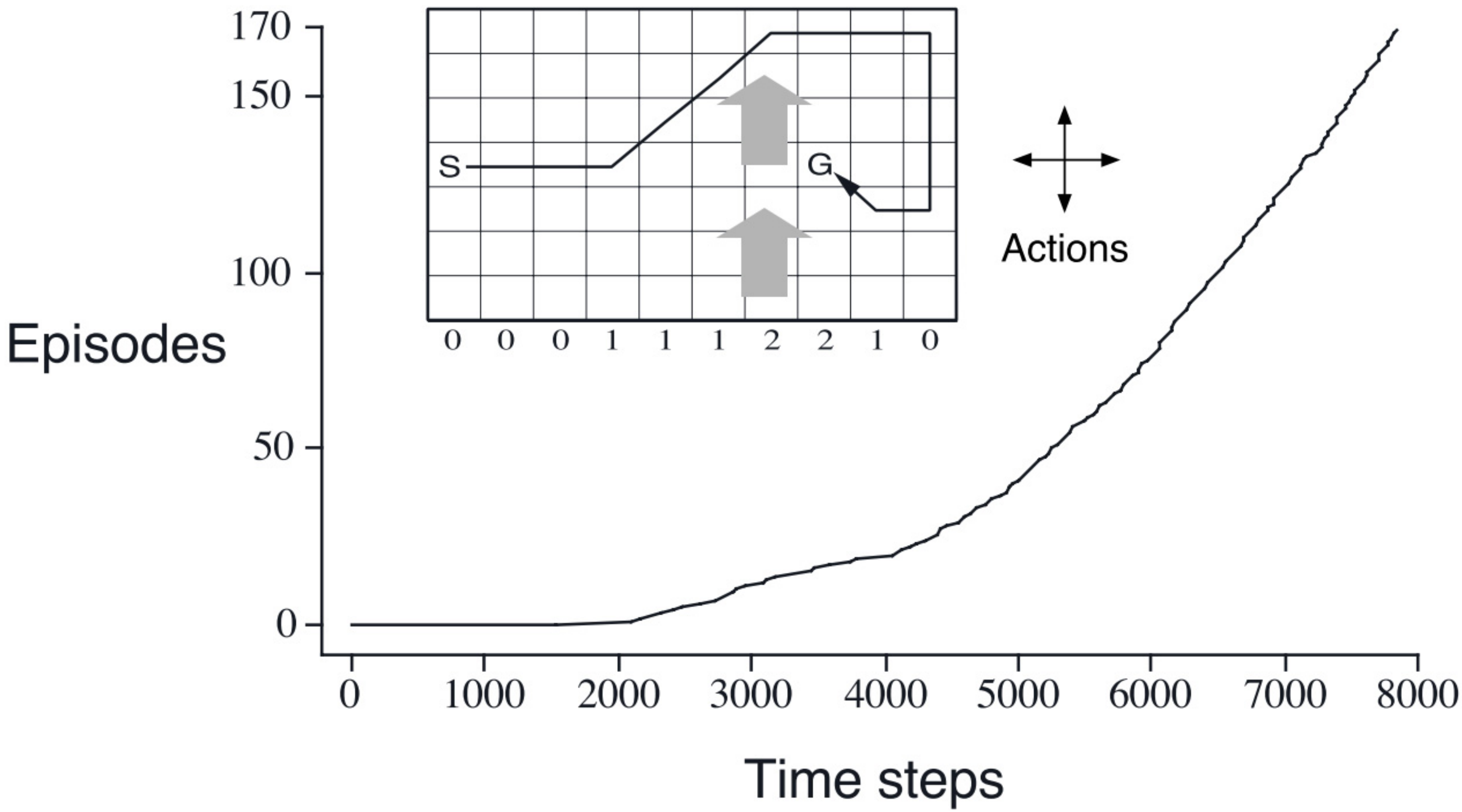
SARSA Example: Windy Gridworld



- Reward = -1 per time-step until reaching goal
- Undiscounted

SARSA Example: Windy Gridworld

optimal a trajectory



Note: as the training proceeds, the Sarsa policy achieves the goal more and more quickly

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $V^{\pi}(s)$ or $Q^{\pi}(s,a)$
- While following behavior policy $\mu(a|s)$

$$\{s_1, a_1, r_2, s_2, a_2, \ldots, s_T\} \sim \mu$$

- Why off-policy learning is important?
  - Learn from observing humans or other agents
  - Re-use experience generated from old policies
  - Learn about optimal policy while following exploratory policy
  - Learn about multiple policies while following one policy
  - An example of my research in MSR Cambridge
    - Collective Noise Contrastive Estimation for Policy Transfer Learning. AAAI 2016.

Importance Sampling

- Estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{x\sim p}[f(x)] &= \int_x p(x) f(x) dx \\ &= \int_x q(x) \frac{p(x)}{q(x)} f(x) dx \\ &= \mathbb{E}_{x\sim q}\Big[\frac{p(x)}{q(x)} f(x)\Big]\end{aligned}$$

- Re-weight each instance by $\beta(x) = \dfrac{p(x)}{q(x)}$

Importance Sampling for Off-Policy Monte-Carlo

- Use returns generated from μ to evaluate π
- Weight return G_t according to importance ratio between policies
- Multiply importance ratio along with episode

$$\{s_1, a_1, r_2, s_2, a_2, \ldots, s_T\} \sim \mu$$

$$G_t^{\pi/\mu} = \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)} \frac{\pi(a_{t+1}|s_{t+1})}{\mu(a_{t+1}|s_{t+1})} \cdots \frac{\pi(a_T|s_T)}{\mu(a_T|s_T)} G_t$$

- Update value towards corrected return

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

  - Cannot use if μ is zero when π is non-zero
  - Importance sample can dramatically increase variance

Importance Sampling for Off-Policy TD

- Use TD targets generated from μ to evaluate π
- Weight TD target $r+\gamma V(s')$ by importance sampling
- Only need a single importance sampling correction

$$V(s_t) \leftarrow V(s_t) + \alpha \left(\underbrace{\frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}}_{\text{importance sampling correction}} \underbrace{(r_{t+1} + \gamma V(s_{t+1}))}_{\text{TD target}} - V(s_t) \right)$$

- Much lower variance than Monte-Carlo importance sampling
- Policies only need to be similar over a single step

Q-Learning

- For off-policy learning of action-value $Q(s,a)$
- No importance sampling is required (why?)
- The next action is chosen using behavior policy $a_{t+1} \sim \mu(\cdot|s_t)$
- But we consider alternative successor action $a \sim \pi(\cdot|s_t)$
- And update $Q(s_t,a_t)$ towards value of alternative action

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a') - Q(s_t, a_t))$$

action
from π
not μ

Off-Policy Control with Q-Learning

- Allow both behavior and target policies to improve
- The target policy π is greedy w.r.t. $Q(s,a)$

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

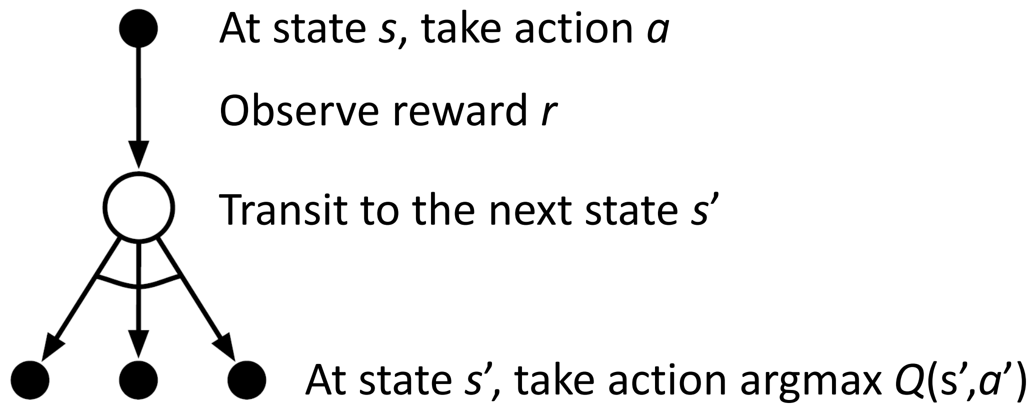
- The behavior policy μ is e.g. ε-greedy policy w.r.t. $Q(s,a)$
- The Q-learning target then simplifies

$$\begin{aligned} r_{t+1} + \gamma Q(s_{t+1}, a') &= r_{t+1} + \gamma Q(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a')) \\ &= r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') \end{aligned}$$

- Q-learning update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

Q-Learning Control Algorithm

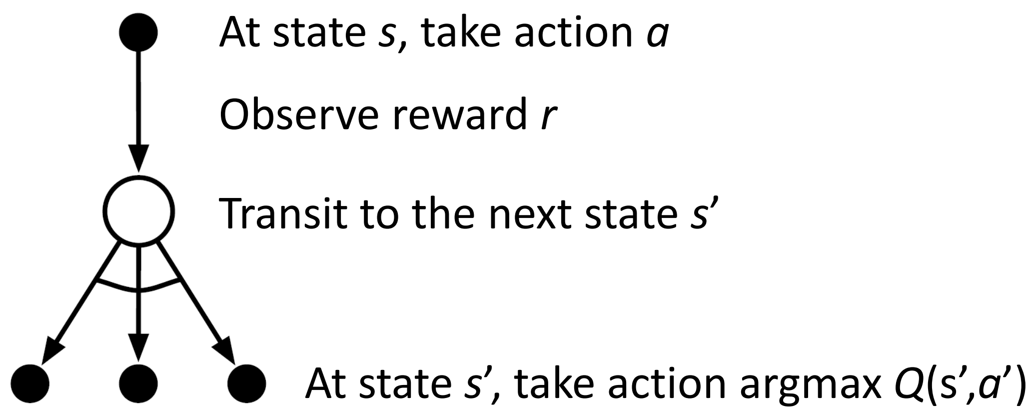


$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

- Theorem: Q-learning control converges to the optimal action-value function

$$Q(s, a) \rightarrow Q^*(s, a)$$

Q-Learning Control Algorithm

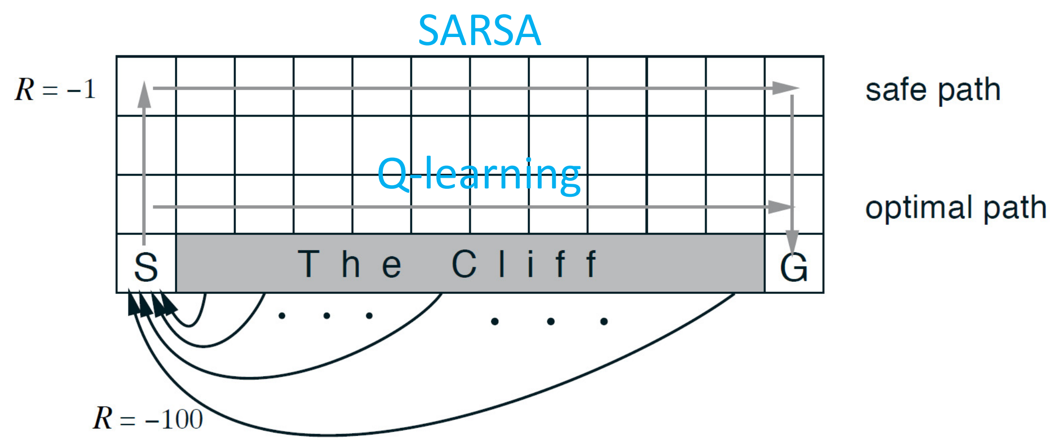


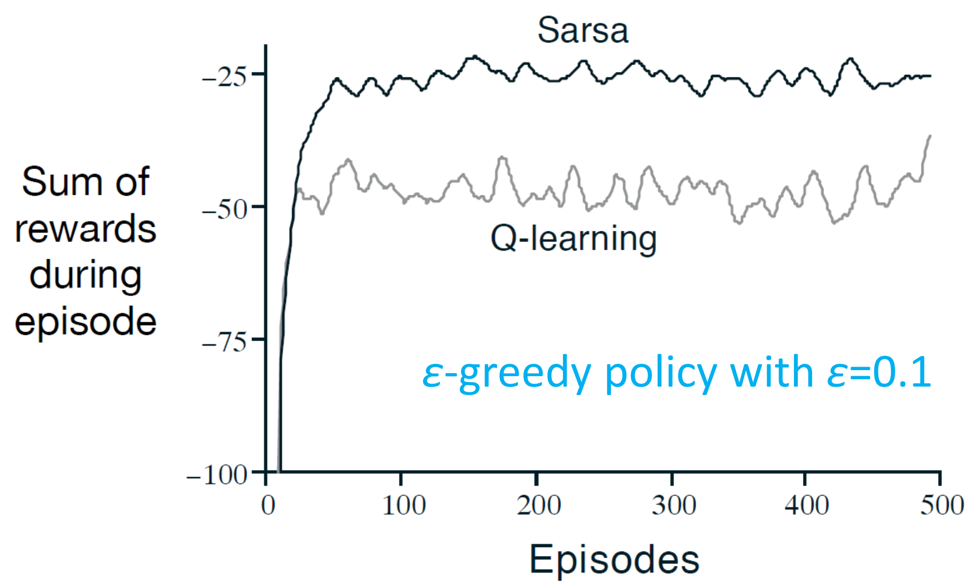
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

- Why Q-learning is an off-policy control method?
  - Learning from SARS generated by another policy μ
  - The first action *a* and the corresponding reward *r* are from μ
  - The next action *a'* is picked by the target policy $\pi(s_{t+1}) = \arg\max_{a'} Q(s_{t+1}, a')$
- Why no importance sampling?
  - Action value function not state value function

SARSA vs. Q-Learning Experiments

- Cliff-walking
  - Undiscounted reward
  - Episodic task
  - Reward = -1 on all transitions
  - Stepping into cliff area incurs -100 reward and sent the agent back to the start
- Why the results are like this?

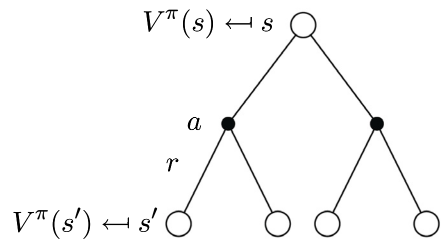
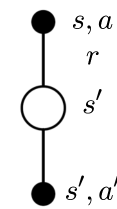
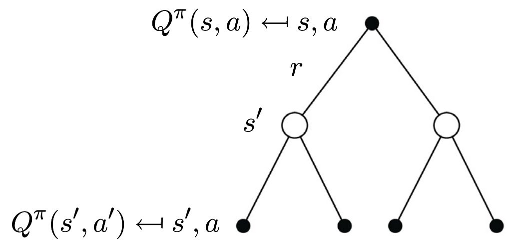
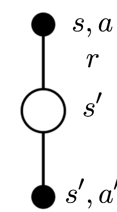
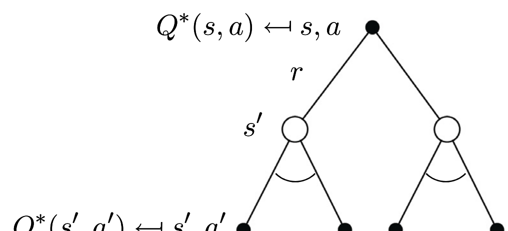
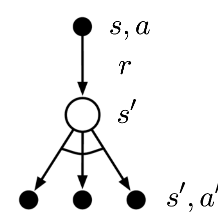




Further Readings

- You can learn following content offline

Relationship Between DP and TD

	Full Backup (DP)	Sample Backup (TD)
Bellman Expectation Equation for $V^\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $Q^\pi(s, a)$	 Q-Policy Iteration	 SARSA
Bellman Optimality Equation for $Q^*(s, a)$	 Q-Value Iteration	 Q-Learning

Relationship Between DP and TD

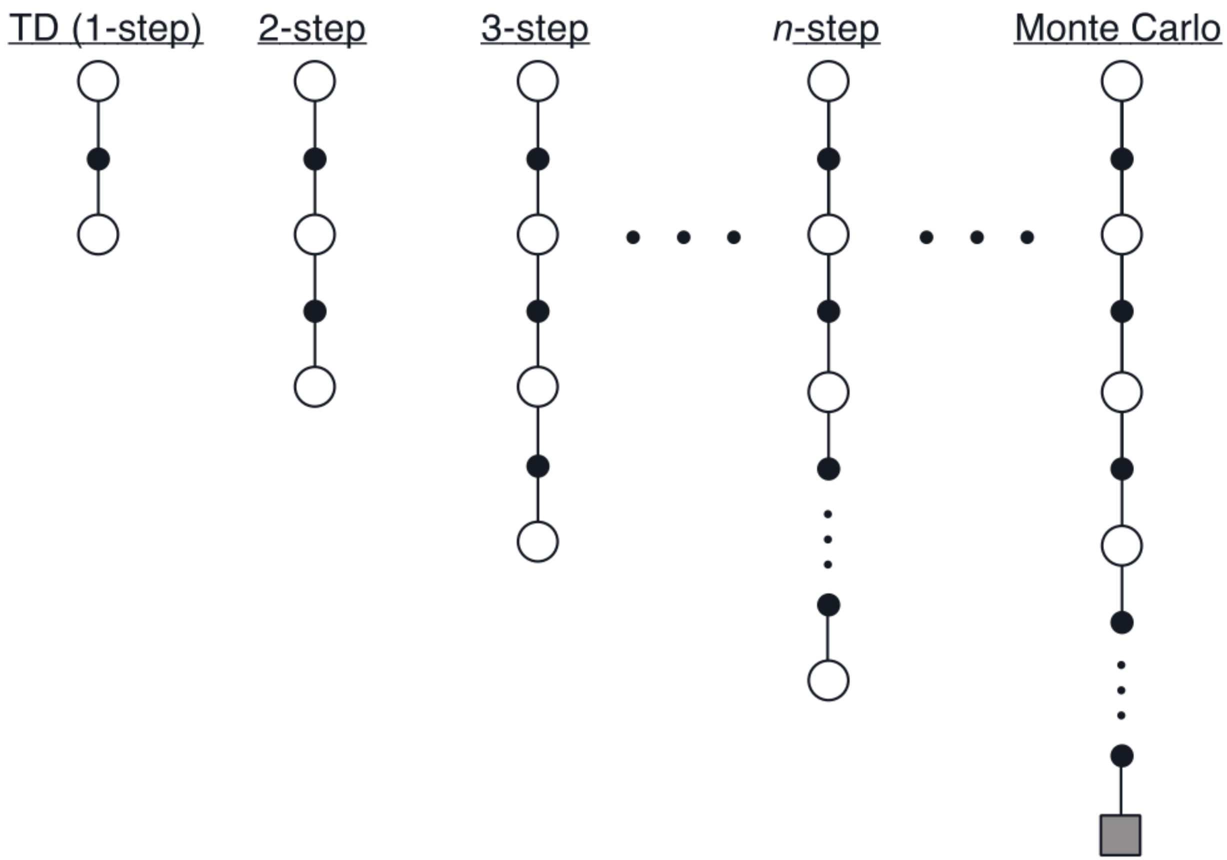
Full Backup (DP)	Sample Backup (TD)
Iterative Policy Evaluation $V(s) \leftarrow \mathbb{E}[r + \gamma V(s')\|s]$	TD Learning $V(s) \xleftarrow{\alpha} r + \gamma V(s')$
Q-Policy Iteration $Q(s,a) \leftarrow \mathbb{E}[r + \gamma Q(s',a')\|s,a]$	SARSA $Q(s,a) \xleftarrow{\alpha} r + \gamma Q(s',a')$
Q-Value Iteration $Q(s,a) \leftarrow \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a')\|s,a\right]$	Q-Learning $Q(s,a) \xleftarrow{\alpha} r + \gamma \max_{a'} Q(s',a')$

where

$$x \xleftarrow{\alpha} y \equiv x \leftarrow x + \alpha(y - x)$$

n-Step Prediction

- Let TD target look *n* steps into the future



n-Step Return

- Consider the following *n*-step return for *n*=1,2,…,∞

$$n = 1 \qquad \text{(TD)} \qquad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1})$$

$$n = 2 \qquad \qquad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})$$

$$\vdots \qquad \qquad \vdots$$

$$n = \infty \qquad \text{(MC)} \qquad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

- Define the *n*-step return

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

- *n*-step temporal-difference learning

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^{(n)} - V(S_t)\right)$$

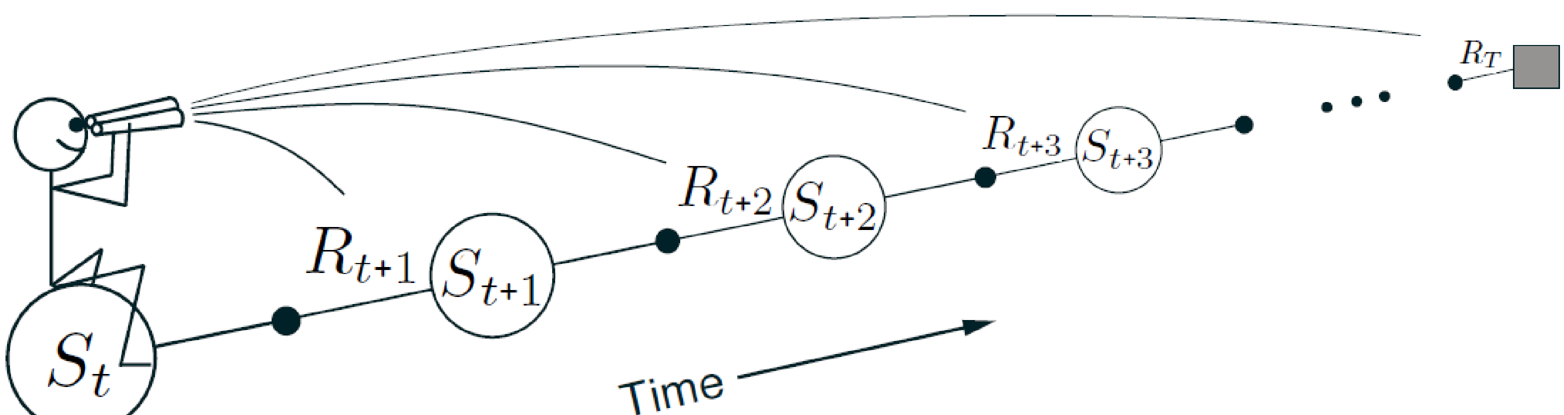
n-Step Return

- Define the *n*-step return

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

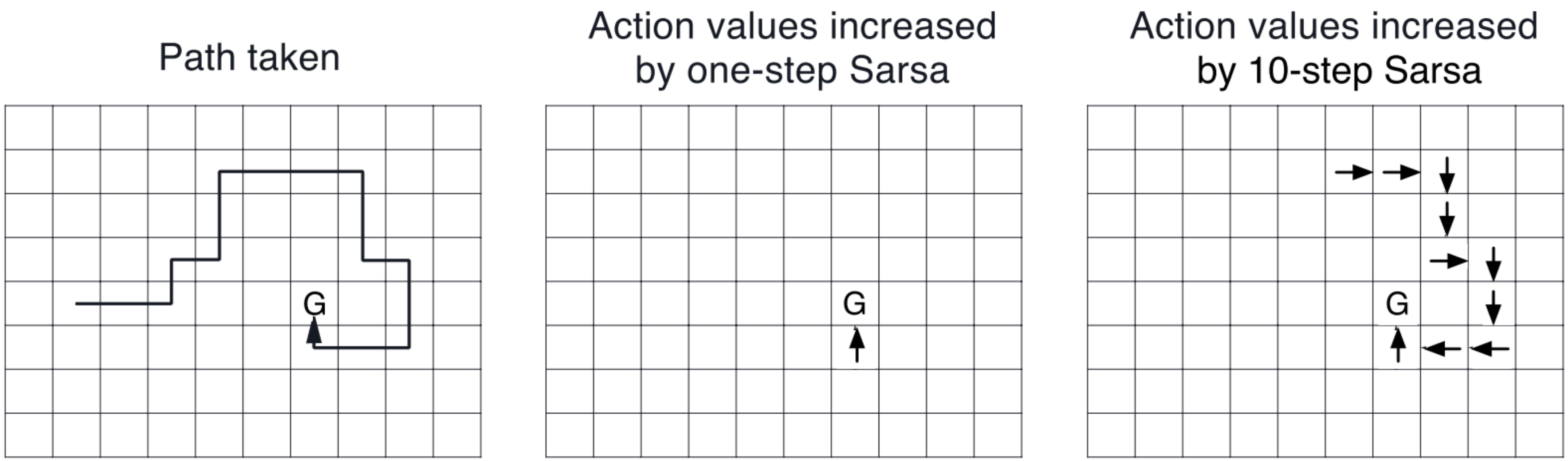
- *n*-step temporal-difference learning

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t^{(n)} - V(S_t))$$



n-Step Return

- Why it can speed up learning compared to one-step methods

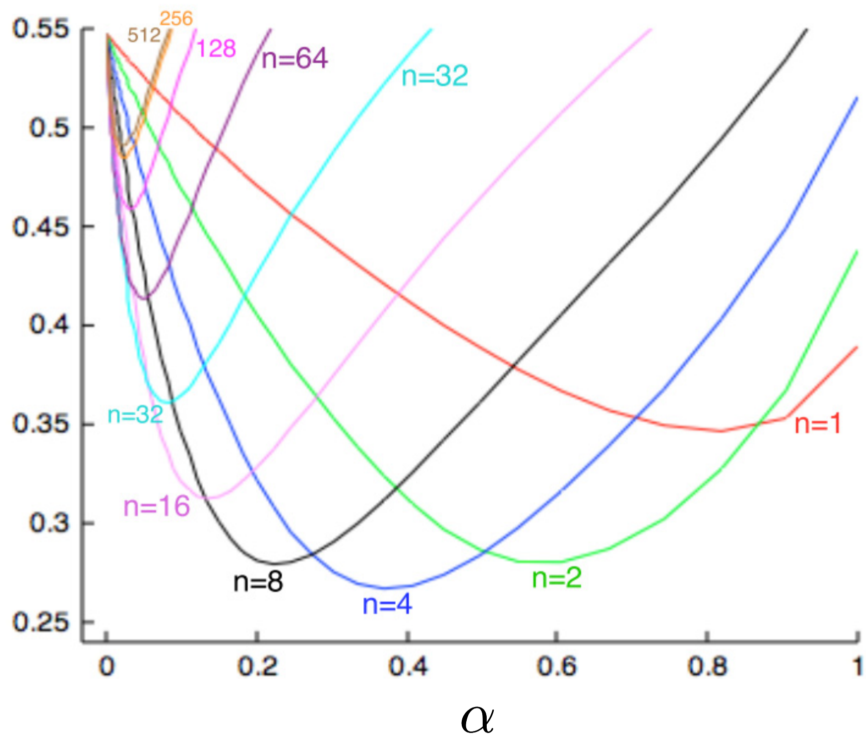


$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

$$V(S_t) \leftarrow V(S_t) + \alpha (G_t^{(n)} - V(S_t))$$

Random Walk Example for *n*-step TDs

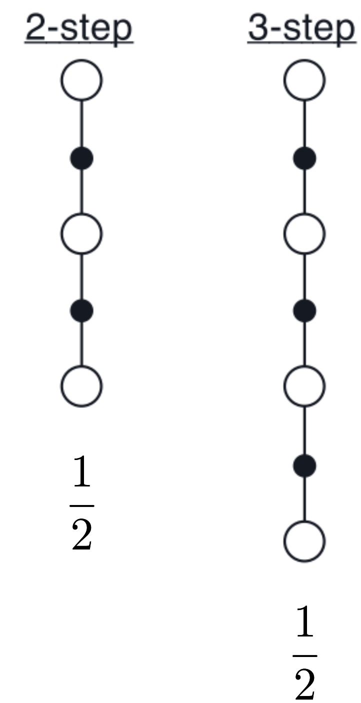
Average RMS error over 19 states and first 10 episodes



Averaging *n*-Step Returns

- We can further average *n*-step returns over different *n*
- e.g. average the 2-step and 3-step returns

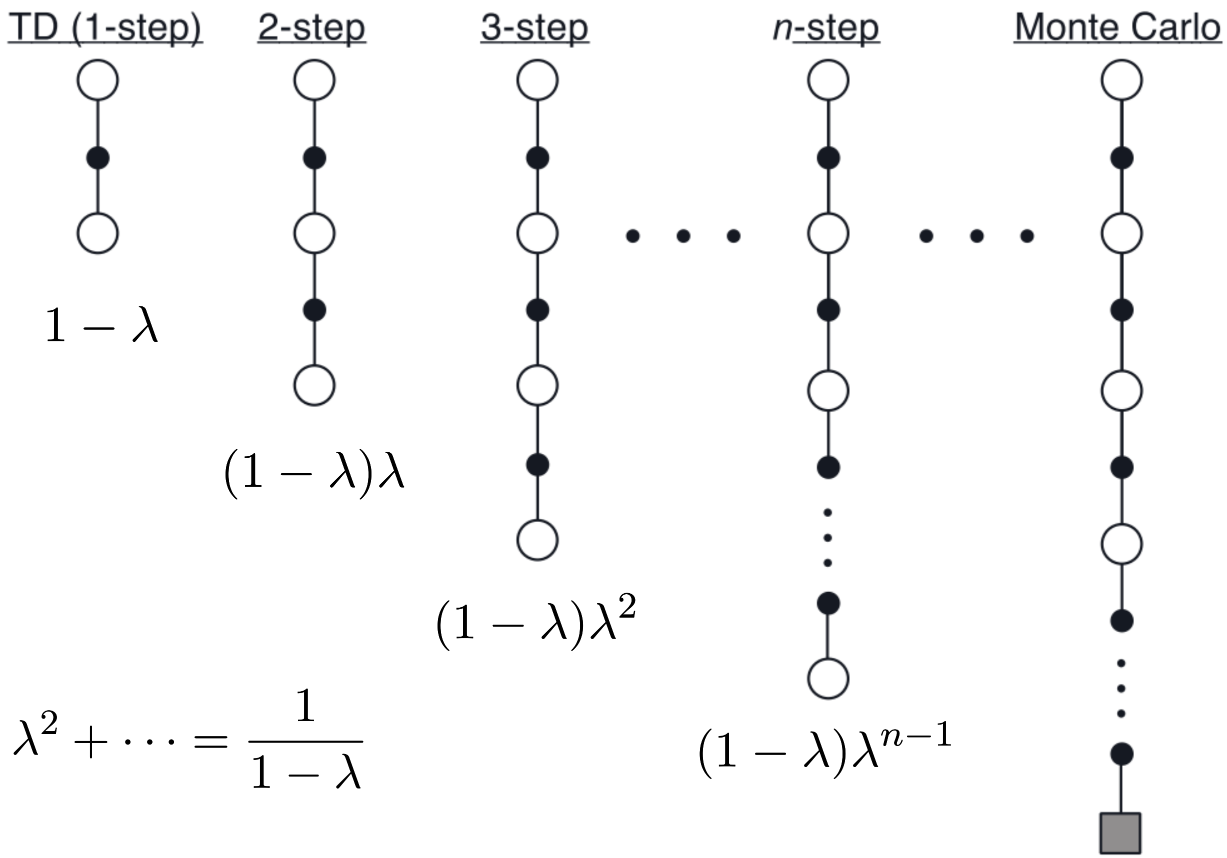
$$\frac{1}{2}G^{(2)} + \frac{1}{2}G^{(3)}$$



- Combines information from two different time-steps
- Can we efficiently combine information from all time-steps?

TD(λ) for Averaging n-Step Returns

TD(λ), λ-return



$1-\lambda$

$(1-\lambda)\lambda$

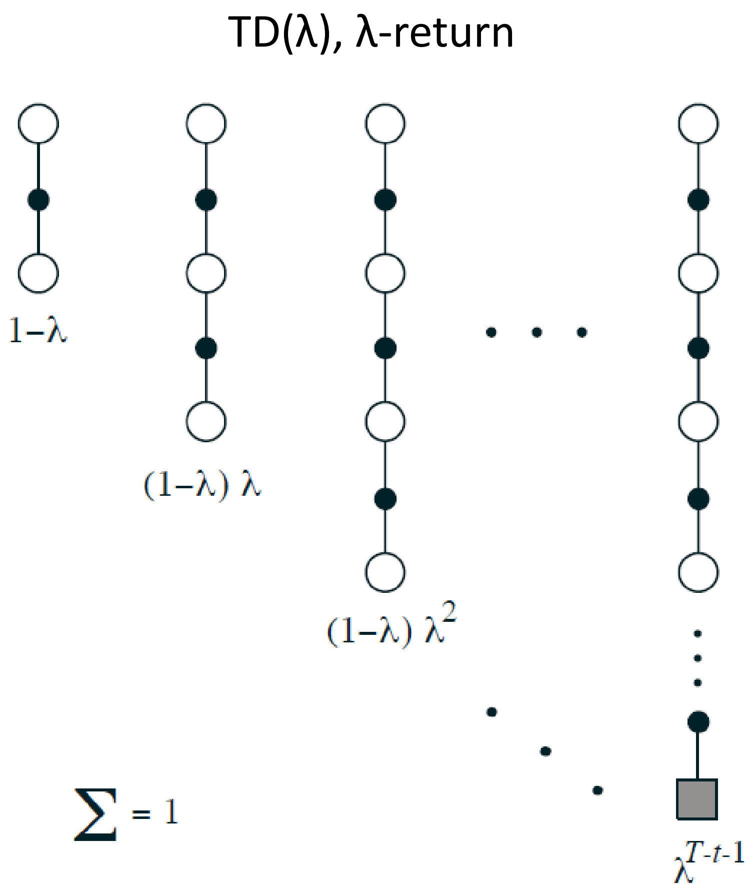
$(1-\lambda)\lambda^2$

$(1-\lambda)\lambda^{n-1}$

$(1-\lambda)\lambda^{T-t-1}$

$$1+\lambda+\lambda^2+\cdots=\frac{1}{1-\lambda}$$

TD(λ) for Averaging *n*-Step Returns



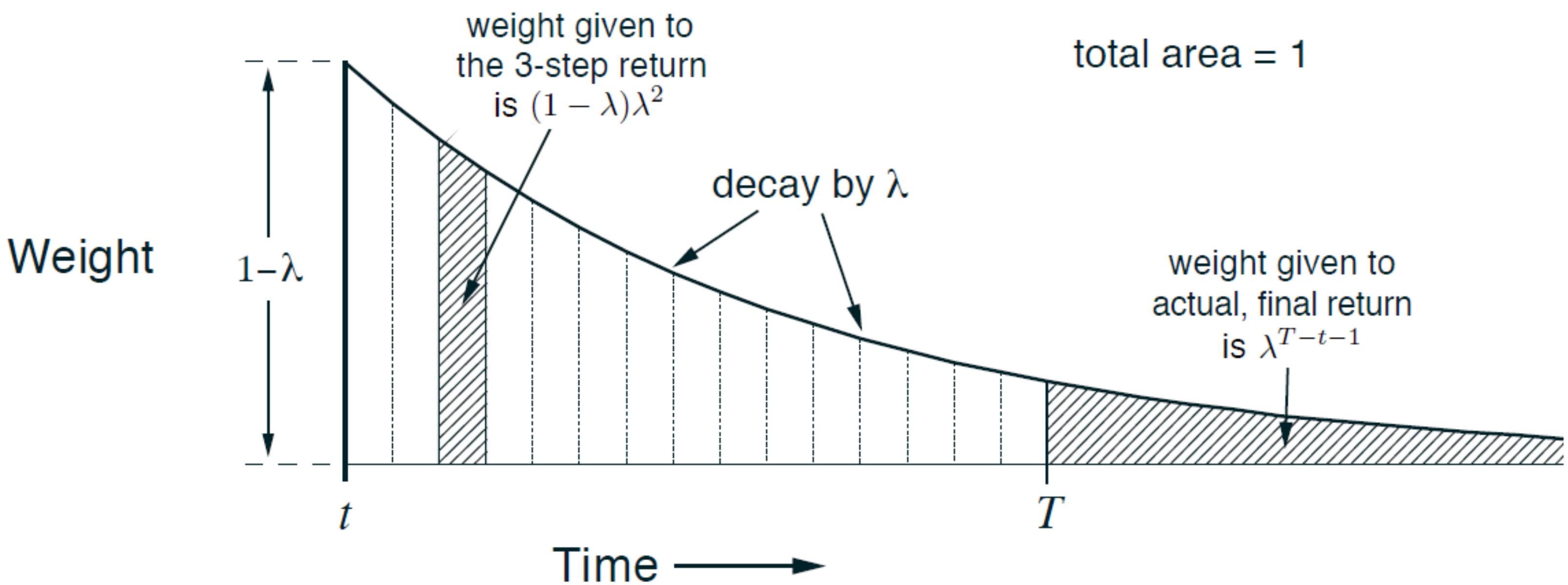
- The λ-return G_t^λ combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\lambda^{n-1}$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

- Forward-view TD(λ)

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

TD(λ) for Averaging *n*-Step Returns

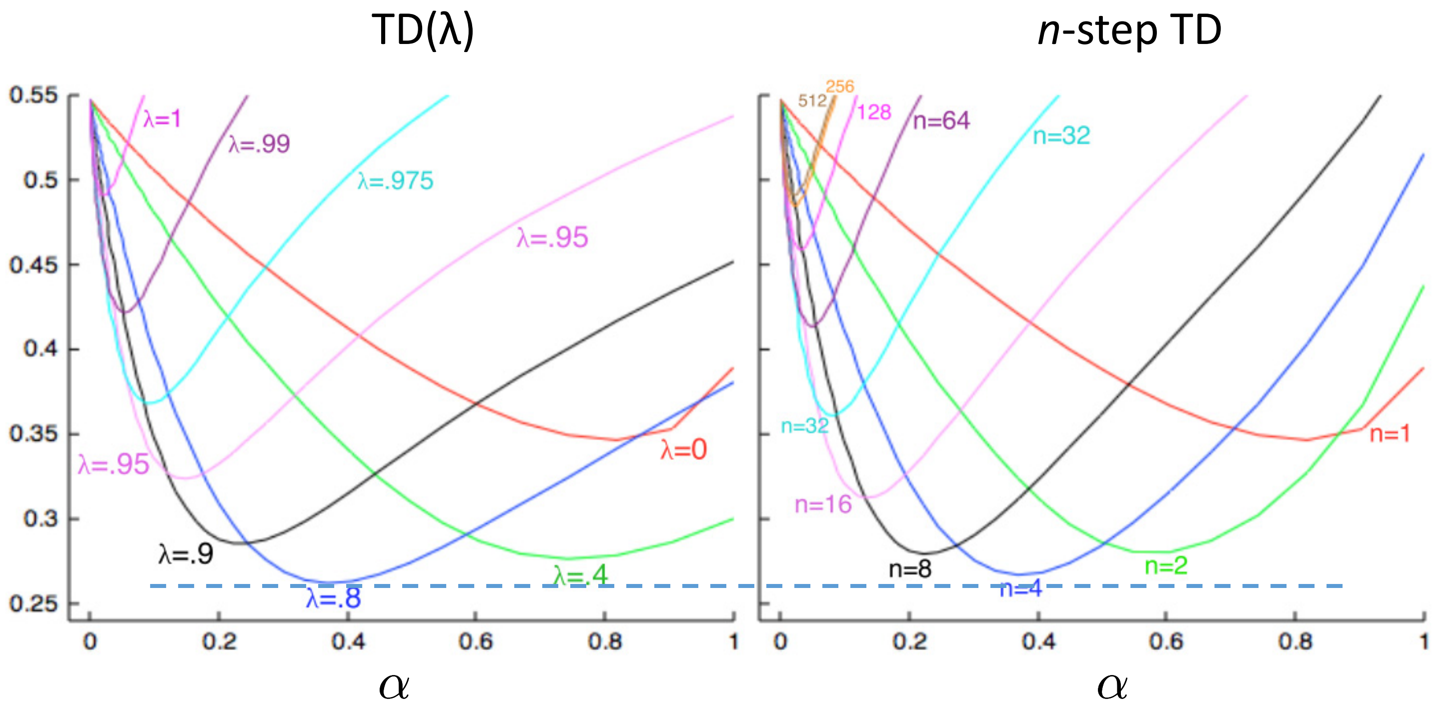


$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{T-t-1}G_t$$

- When λ=1, $G_t^\lambda = G_t$, which returns to Monte-Carlo method
- When λ=0, $G_t^\lambda = G_t^{(1)}$, which returns to one-step TD

TD(λ) vs. *n*-step TD

RMS error at the end of the episode over the first 10 episodes Off-line



19-state Random walk results

- The results with off-line λ-return algorithms are slightly better at the best value of α and λ