Reinforcement Learning 04 - Monte Carlo

Elena, Xi

Previous lecture

Markov Decision Processes

Markov decision processes formally describe an environment for reinforcement learning where the environment is *fully observable*

A finite MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, p(), \mathcal{R} \rangle$

$\mathcal{S}$	is a finite set of possible states
$\mathcal{A}(S_t)$	is a finite set of actions in state S_t
$p(s' \mid s,a)$	is a state transition probability matrix, $p(s' \mid s,a) = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
$\mathcal{R}$	is a final set of all possible rewards

Planning by Dynamic Programming

Dynamic programming assumes that we know the MDP for our problem
It is used for *planning* in an MDP

For **prediction**:

Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ and policy π

Output: value function v_{π}

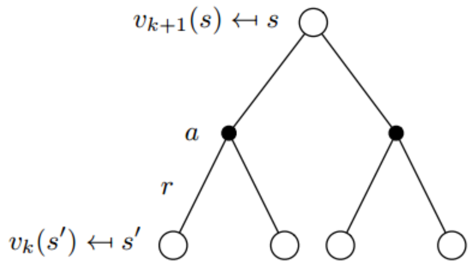
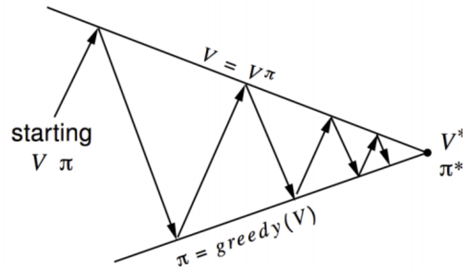
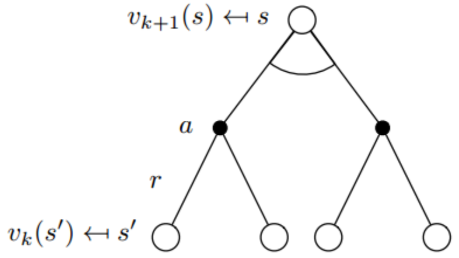
For **control**:

Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$

Output: optimal policy π_{*}

(optimal value function v_{*})

Dynamic Programming Algorithms

Algorithm	*Iterative Policy Evaluation*  $v_{k+1}(s) \leftarrow s$ a r $v_k(s') \leftarrow s'$	*Policy Iteration*  starting V π $V = V\pi$ $\pi = greedy(V)$ V^* π^*	*Value Iteration*  $v_{k+1}(s) \leftarrow s$ a r $v_k(s') \leftarrow s'$
Bellman Equation	Bellman Expectation Equation	Bellman Expectation Equation Policy Iteration + Greedy Policy Improvement	Bellman Optimality Equation
Problem	Prediction	Control	Control

This lecture

Like previous
but with blackjack

Model-Free Reinforcement Learning

Previous lecture:

Planning by dynamic programming

Solve a known MDP

This lecture:

Model-free **prediction**

Estimate the value function of an unknown MDP using Monte Carlo

Model-free **control**

Optimise the value function of an unknown MDP using Monte Carlo

Monte Carlo Method Introduction

MC method - any method which solves a problem by generating suitable random numbers and observing that fraction of the numbers obeying some property or properties.

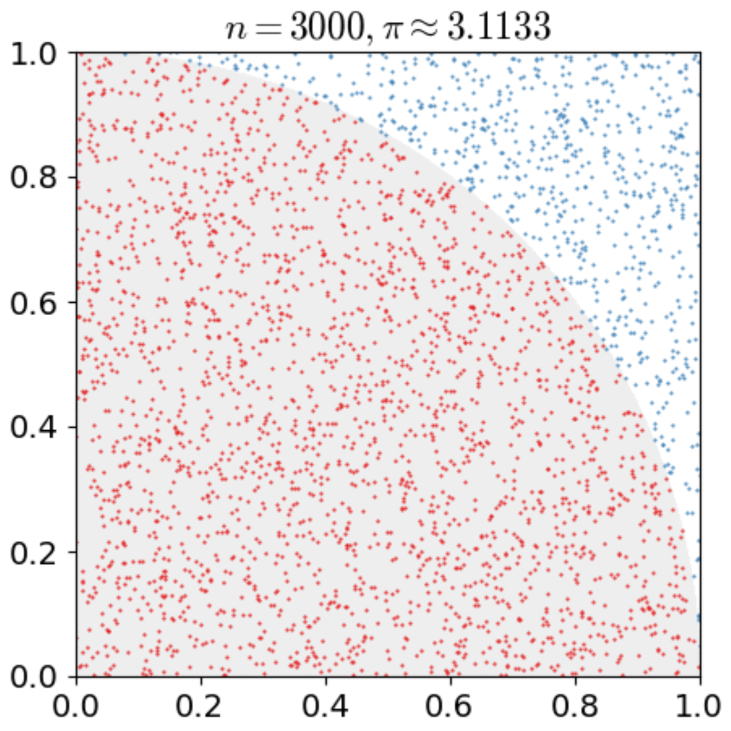
$\mathbb{E}[X] = 1/n \sum_{i=1}^{n} x_i$

Modern version of MC was named by Stanislaw Ulam in 1946 in honor of his uncle who often borrowed money from relatives to gamble in Monte Carlo Casino (Monaco)

S. Ulam came up with this idea while recovering from surgery and playing solitaire. He tried to estimate the probability of wining given the initial state.

Monte Carlo Method Simple Example

Monte Carlo method applied to approximating the value of π. After placing 30,000 random points, the estimate for π is within 0.07% of the actual value.



Monte Carlo Reinforcement Learning

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

MC uses the simplest possible idea: value = mean return

Caveat: can only apply MC to episodic MDPs

All episodes must terminate

Monte Carlo method introduction

Monte Carlo **Prediction**

Monte Carlo **Control**

Monte Carlo method introduction

Monte Carlo **Prediction**

Monte Carlo **Control**

Monte Carlo Policy Evaluation

Goal: learn $v_\pi(s)$ from episodes of experience under policy π

$S_1, A_1, R_2, \ldots, S_k \sim \pi$

Recall that the **return** is the **total discounted reward**:

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

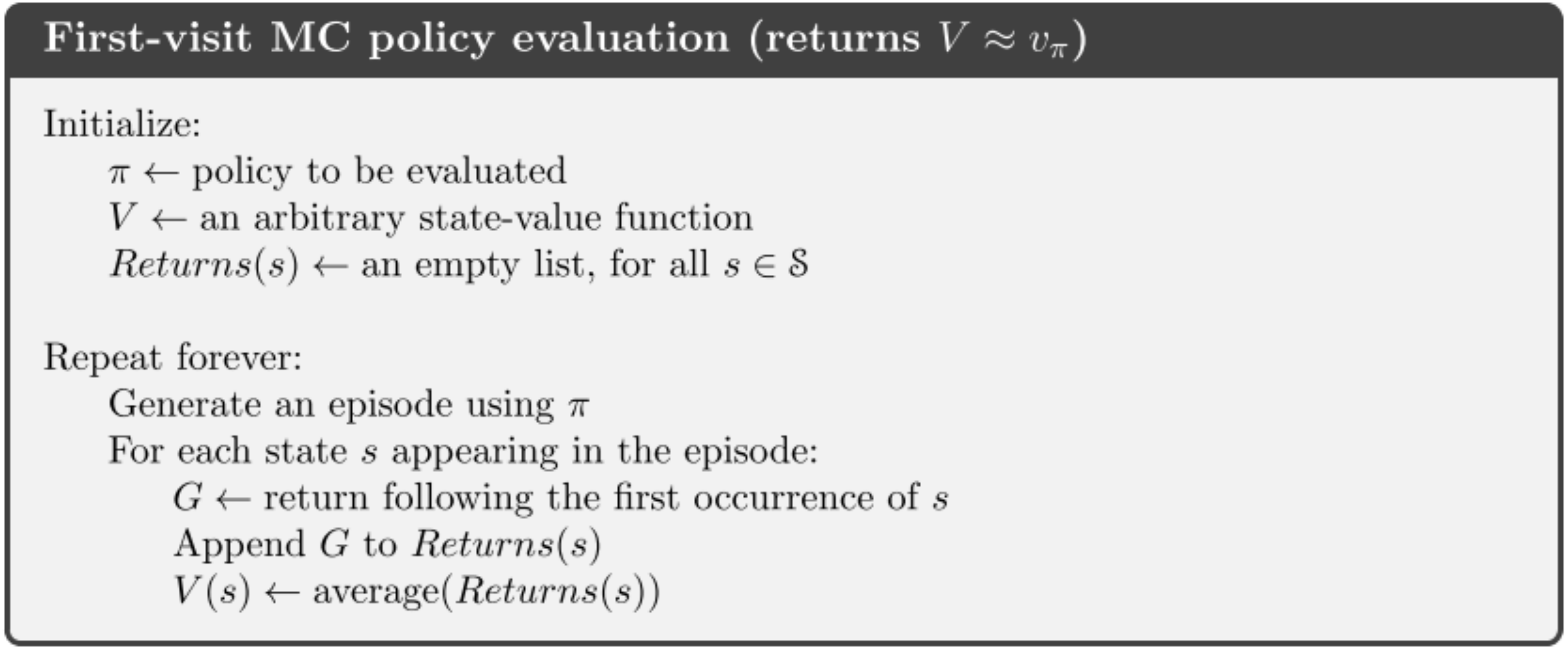
Recall that the **value function** is the **expected return**:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

MC policy evaluation uses **empirical mean** return instead of **expected** return

First-visit MC: average returns only for first time s is visited in an episode
Every-Visit MC: average returns for every time s is visited in an episode
Both converge asymptotically

First-visit Monte Carlo policy evaluation



First-visit MC policy evaluation (returns $V \approx v_\pi$)

```
Initialize:
    $\pi \leftarrow$ policy to be evaluated
    $V \leftarrow$ an arbitrary state-value function
    $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
    Generate an episode using $\pi$
    For each state $s$ appearing in the episode:
        $G \leftarrow$ return following the first occurrence of $s$
        Append $G$ to $Returns(s)$
        $V(s) \leftarrow$ average($Returns(s)$)
```

By the law of large numbers, $V(s) \rightarrow v_{\pi}(s)$ *as number of episods* $\rightarrow \infty$

MC policy evaluation EXAMPLE

undiscounted Markov Reward Process

two states A and B

transition matrix and reward function are unknown

observed two sample episodes

$$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$$

$$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$$

A+3 → A indicates a transition from state A to state A, with a reward of +3

Using **first-visit**, state-value functions **V(A), V(B) - ?**

Using **every-visit**, state-value functions **V(A), V(B) - ?**

MC policy evaluation EXAMPLE Solution

$$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$$

$$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$$

first-visit

V(A) = 1/2(2 + 0)=1

V(B) = 1/2(-3 + -2)= -5/2

every-visit

V(A) = 1/4(2 + -1 + 1 + 0) = 1/2

V(B) = 1/4(-3 + -3 + -2 + -3) = -11/4

Blackjack Example

States (200 of them):

- Current sum (12-21)
- Dealer’s showing card (ace-10)
- Do I have a “useable” ace? (yes-no)

Action **stick**: Stop receiving cards (and terminate)

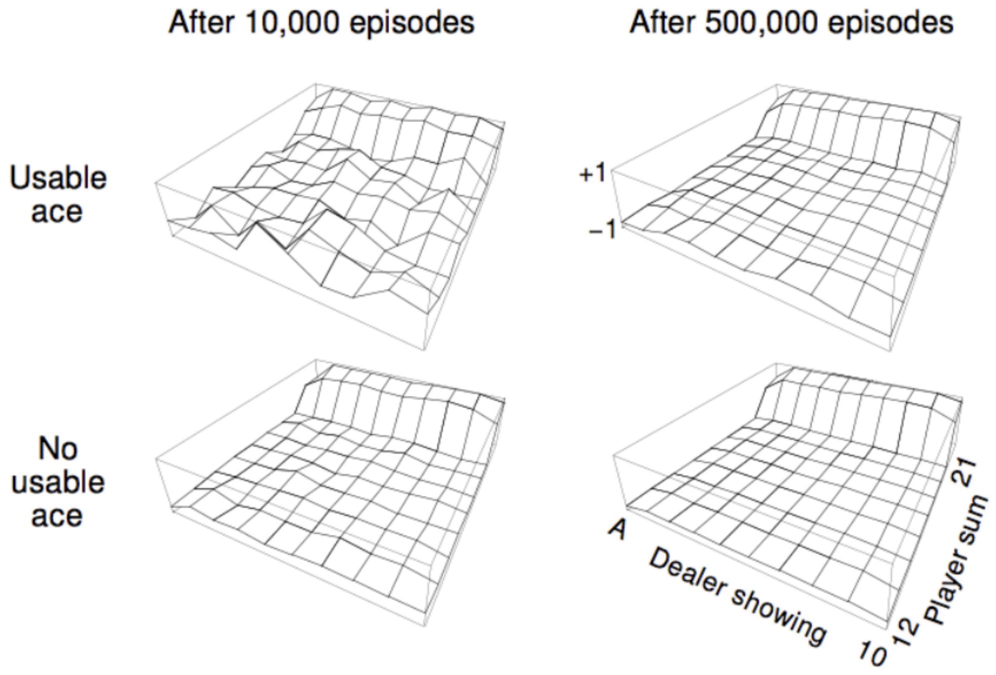
Action **hit**: Take another card (no replacement)

Reward for **stick**: +1 if sum of cards > sum of dealer cards
0 if sum of cards = sum of dealer cards
-1 if sum of cards < sum of dealer cards

Reward for **hit**: -1 if sum of cards > 21 (and terminate)
0 otherwise

Transitions: automatically **hit** if sum of cards < 12

Blackjack Value Function after Monte Carlo Learning



Policy: **stick** if sum of cards ≥ 20, otherwise **hit**

Incremental Mean

The mean μ_1, μ_2, ... of a sequence x_1, x_2, ... can be computed incrementally

$$\boxed{\begin{aligned}\mu_k &= \frac{1}{k}\sum_{j=1}^{k} x_j \\ &= \frac{1}{k}\left(x_k + \sum_{j=1}^{k-1} x_j\right) \\ &= \frac{1}{k}\left(x_k + (k-1)\mu_{k-1}\right) \\ &= \mu_{k-1} + \frac{1}{k}\left(x_k - \mu_{k-1}\right)\end{aligned}}$$

Incremental Monte Carlo Updates

Update V(s) incrementally after episode S_1, A_1, R_2, ..., S_T
For each state S_t with return G_t

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

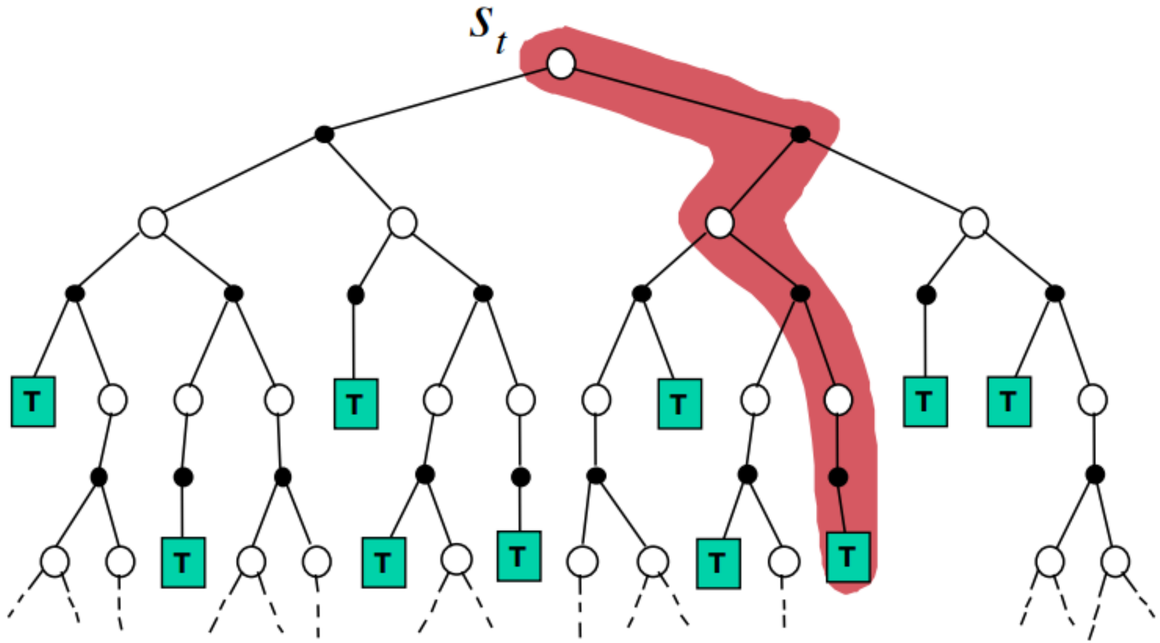
$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t))$$

In non-stationary problems, it can be useful to track a running mean, i.e. forget old episodes.

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

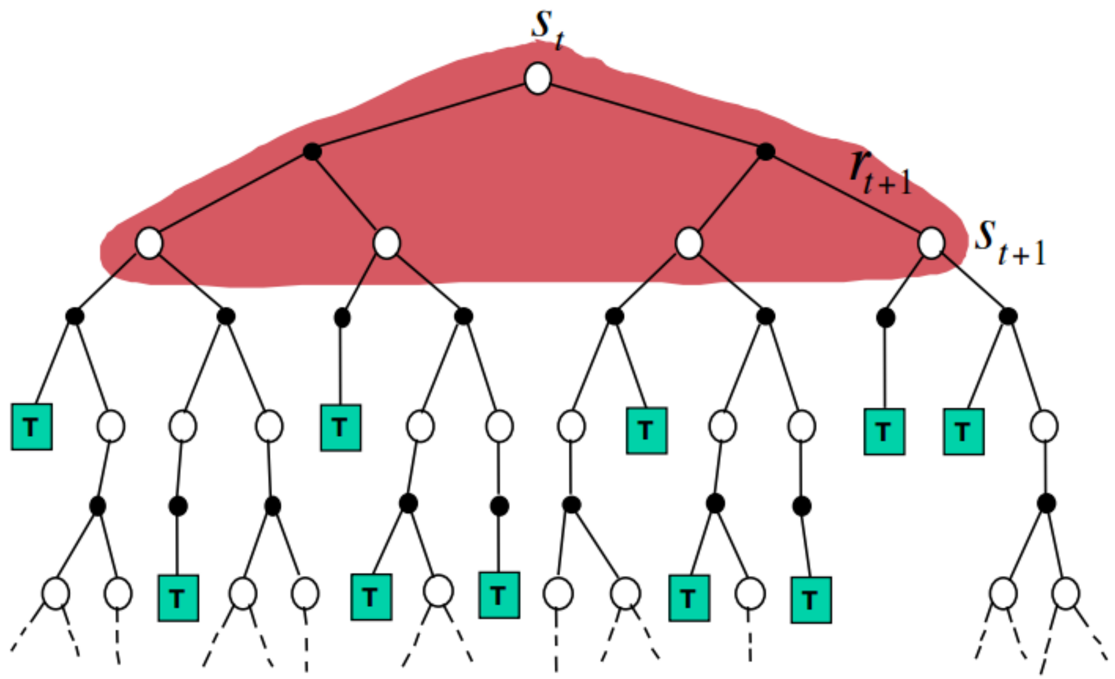
Monte Carlo Backup

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



Dynamic Programming

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



Backup diagram for Monte Carlo

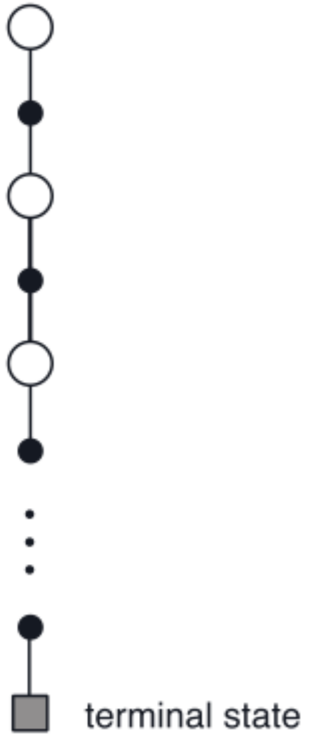
Entire episode included

Only one choice at each state (unlike DP)

MC does not bootstrap (update estimates on the basis of other estimates)

Estimates for each state are independent

Time required to estimate one state does not depend on the total number of states



Monte Carlo method introduction

Monte Carlo **Prediction**

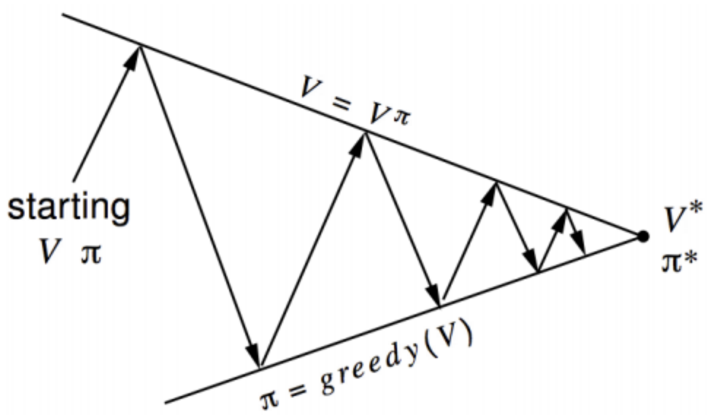
Monte Carlo **Control**

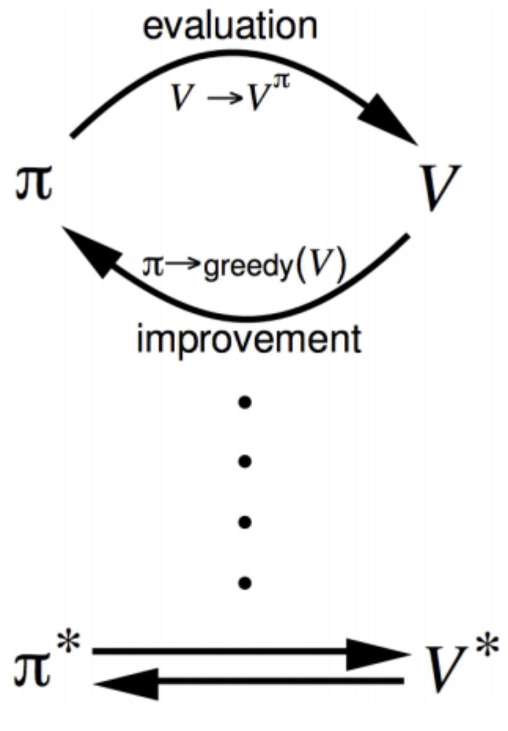
Monte Carlo method introduction

Monte Carlo **Prediction**

Monte Carlo **Control**

Generalised Policy Iteration (Refresher)

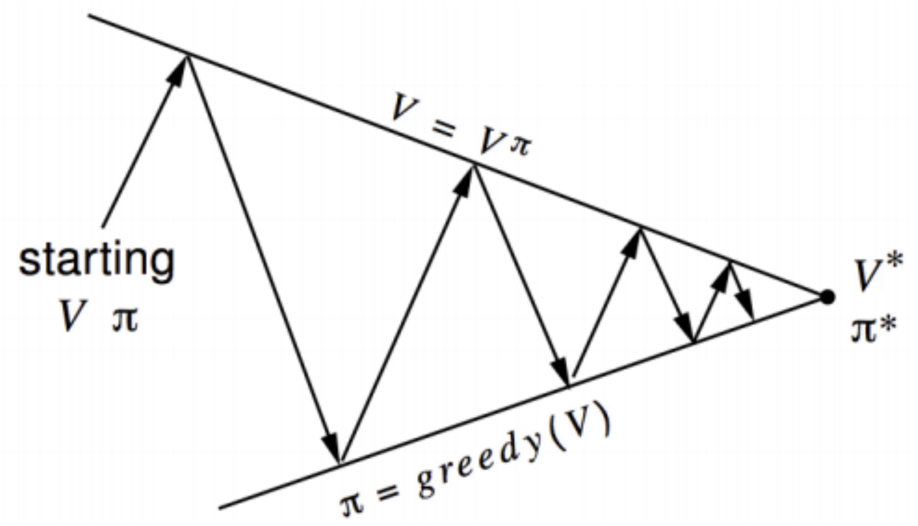




Policy evaluation - Estimate $v↓\pi$ e.g. Iterative policy evaluation

Policy improvement - Generate $\pi' \geq \pi$ e.g. Greedy policy improvement

Generalised Policy Iteration With Monte Carlo Evaluation



Policy evaluation - Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement - Greedy policy improvement?

Model-Free Policy Iteration Using Action-Value Function

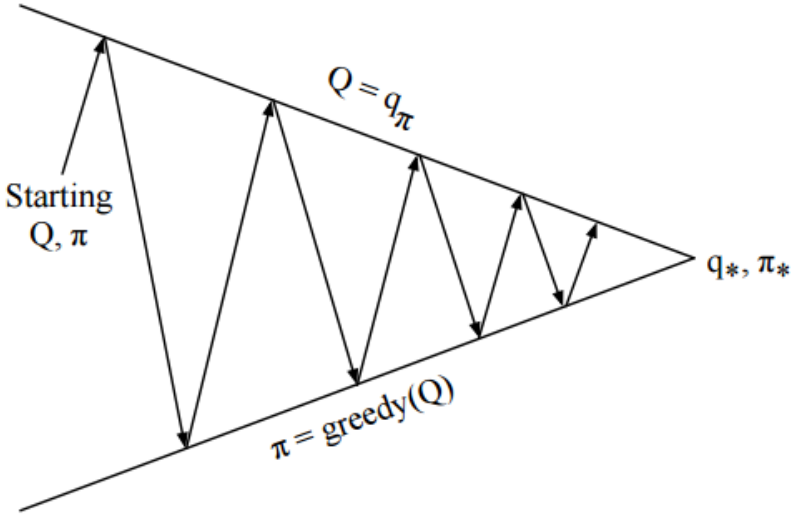
Greedy policy improvement over V(s) requires model of MDP

$$\pi'(s) = \operatorname{argmax}_{a \in A} \sum_{s', r} p(s', r \mid s, a)\,[r + \gamma v_\pi(s')]$$

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

$$\pi'(s) = \operatorname{argmax}_{a \in A} Q(s, a)$$

Generalised 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

There are two doors in front of you.

You open the left door and get reward 0
V(left) = 0
You open the right door and get reward +1
V(right) = +1
You open the right door and get reward +3
V(right) = +2
You open the right door and get reward +2
V(right) = +2

. . .

Are you sure you've chosen the best door?

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

$\pi as = \{ ■ \varepsilon / m + 1 - \varepsilon, \quad if\ a \upharpoonleft * = \mathrm{argmax} \top a \in A\ Q(s,a)\ \varepsilon / m\ , \qquad otherwise$

ε-Greedy Policy Improvement

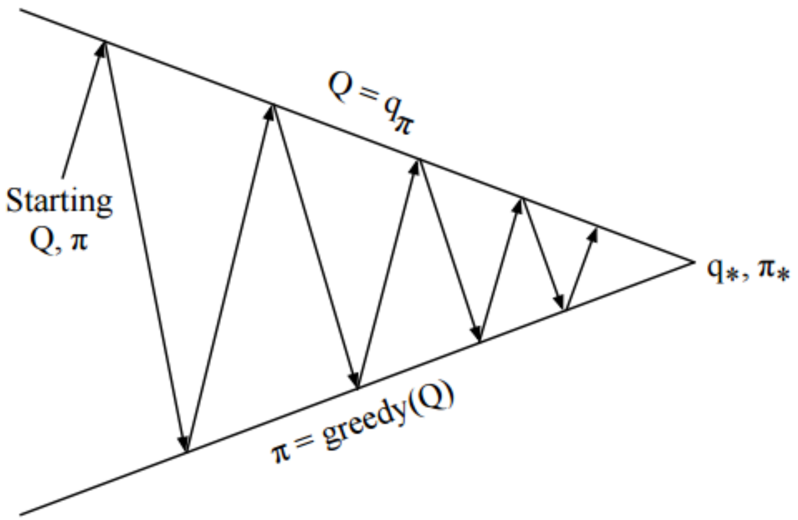
Theorem

For any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to q_π is an improvement, $v_{\pi'}(s) \geq v_\pi(s)$

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

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

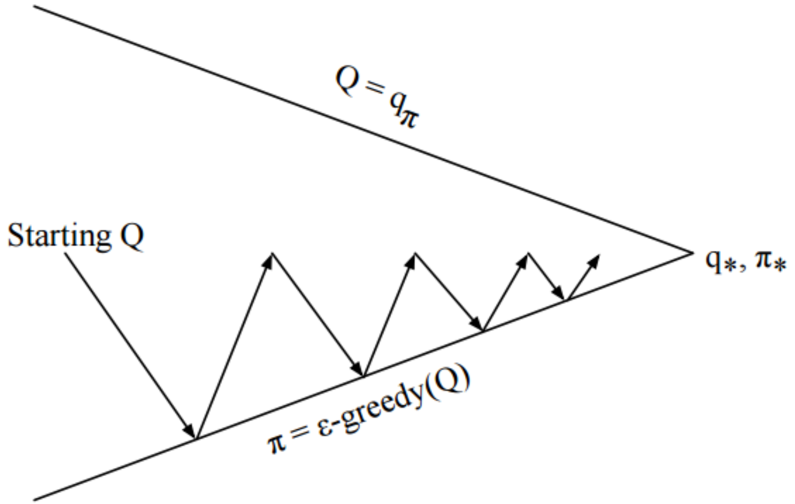
Monte Carlo Policy Iteration



Policy evaluation - Monte Carlo policy evaluation, $Q=q_{\pi}$

Policy improvement - **ε-greedy** policy improvement

Monte Carlo Control

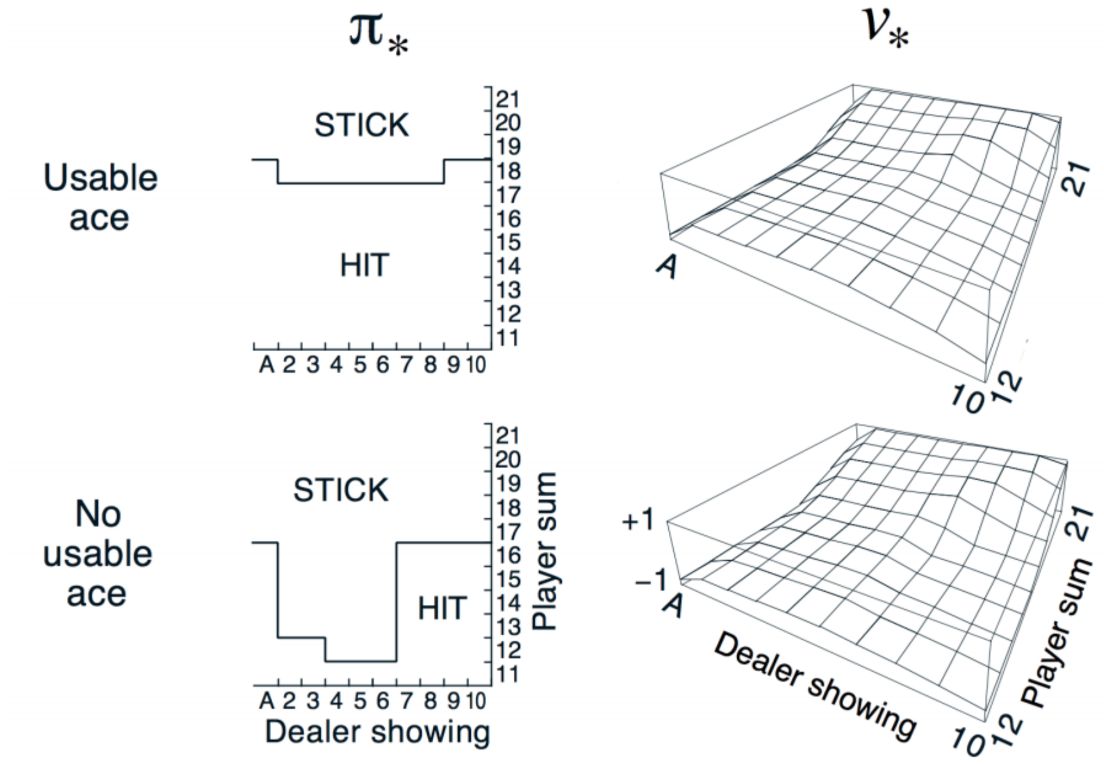


Every episode:

Policy evaluation - Monte Carlo policy evaluation, $Q \approx q_\pi$

Policy improvement - ε-greedy policy improvement

Monte Carlo Control in Blackjack



On-policy vs Off-policy

On-policy vs Off-policy

There are two ideas to take away the **Exploring Starts assumption**:

- **On-policy methods:**

Learning while doing the job

Learning policy π from the episodes that generated using π

- **Off-policy methods:**

Learning while watching other people doing the job

Learning policy π from the episodes generated using another policy μ

On-policy

In On-policy control methods the policy is generally *"soft"*, meaning that:

$$\pi(a|s) > 0 \quad \forall s \in S, a \in A(s)$$

ε-Greedy Policy Improvement:
All policies have a probability to be chosen, but gradually the selected policy is closer and closer to a deterministic optimal policy by controlling the ε value.

Other ways of soft policies improvement

- Uniformly random policy: $\pi(s,a)=1/|A(s)|$
- ε-soft policy: $\pi(s,a)\geq\epsilon/|A(s)|$
- ε-greedy policy: $\pi(s,a)=\epsilon/|A(s)|$, and $\pi(s,a)=1-\in+\epsilon/|A(s)|$ for the greedy action

Off-policy

Learning policy $\boldsymbol{\pi}$ by following the data generated using policy $\boldsymbol{\mu}$

Why is it important?

- Learn from observing humans or other agents
- Re-use experience generated from old policies
- Learn about optimal policy while following exploratory policy

We call:

- π the **target policy:** the policy being learned about
- μ the **behavior policy:** the policy generates the moves

Off-policy

However we need μ to satisfy a condition:

$$\pi(a, s)>0 \quad \rightarrow \quad \mu(a, s)>0$$

Every action which is taken under policy π must have a non-zero probability to be taken as well under policy μ. We call this the **assumption of coverage**.

Typically the target policy π would be a greedy policy with respect to the current action-value function

Off-policy: Importance Sampling

The tool we use for estimation is called **importance sampling**. It is a general technique for estimating expected values of one distribution given samples from another.

$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

$\prod_{k=t}^{T-1} \pi A_k S_k \; p(S_{k+1} | S_k, A_k)$

Where $\mathrm{p}(S_{k+1} | S_k, A_k)$ is the state-transition probability.

Off-policy: Importance Sampling

The relative probability of the trajectory under the target and behavior policies, or the **importance sampling ratio**, is :

$$p_f^T = \prod_{k=t}^{T-1} \pi A_k S_k \; p(S_{k+1} | S_k, A_k) \Big/ \prod_{k=t}^{T-1} \mu A_k S_k \; p(S_{k+1} | S_k, A_k) = \prod_{k=t}^{T-1} \pi A_k S_k \Big/ \mu A_k S_k$$

The state-transition probability depend on the MDP, which are generally unknown, cancel each other out.

Off-policy: Importance Sampling

Ordinary importance sampling: scale the returns by the ratios and average the results.

$$V(s) \doteq \frac{\sum_{t\in\mathfrak{T}(s)} \rho_t^{T(t)} G_t}{|\mathfrak{T}(s)|}.$$

$\mathfrak{T}(s)$	Episodes follow behavior policy
$p\downarrow t\uparrow T(t)$ ratio	Importance sampling
$G\downarrow t$	Episode reward

Weighted importance sampling: scale the returns use weighted average.

$$V(s) \doteq \frac{\sum_{t\in\mathfrak{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t\in\mathfrak{T}(s)} \rho_t^{T(t)}}$$

Off-policy: Importance Sampling

Ordinary importance sampling: scale the returns by the ratios and average the results.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{|\mathcal{T}(s)|}.$$

$$V(s) \doteq \rho_t^{T(t)} G_t$$

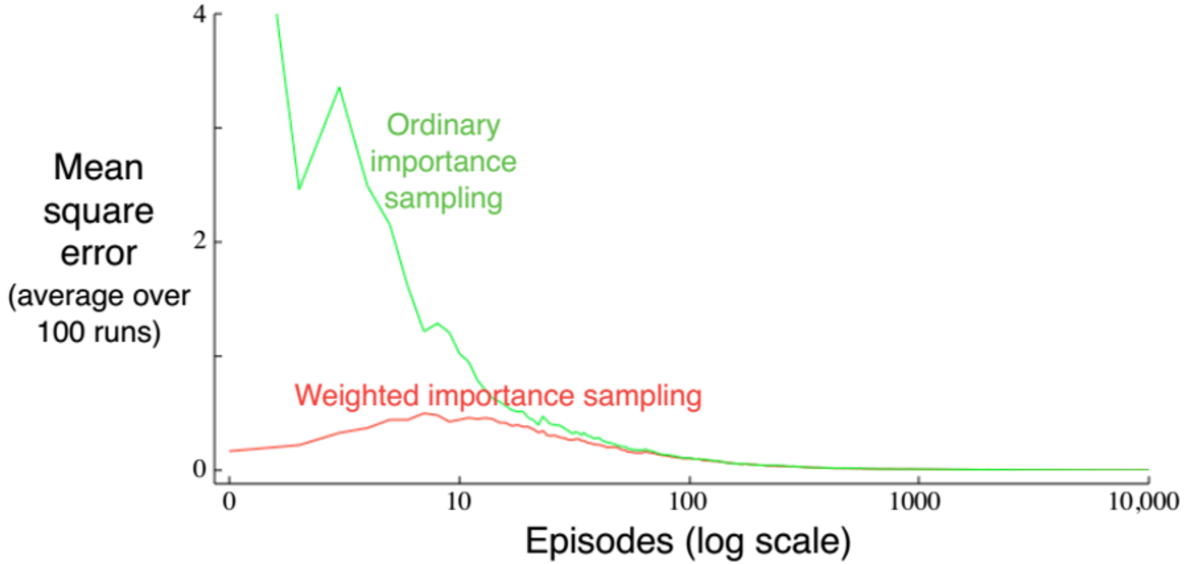
Weighted importance sampling: scale the returns use weighted average.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_t^{T(t)}}$$

$$V(s) \doteq G_t$$

Off-policy: Importance Sampling

In practice the weighted estimator has dramatically lower variance and is therefore strongly preferred. Example of a blackjack state



Pros and cons of MC

MC has several advantages over DP:

- Can learn V and Q directly from interaction with environment (using episodes!)
- No need for full models (using episodes!)
- No need to learn about ALL states (using episodes!)

However, there are some limitations:

- MC only works for episodic (terminating) environments
- MC learns from complete episodes, so no bootstrapping
- MC must wait until the end of an episode before return is known

Next lecture

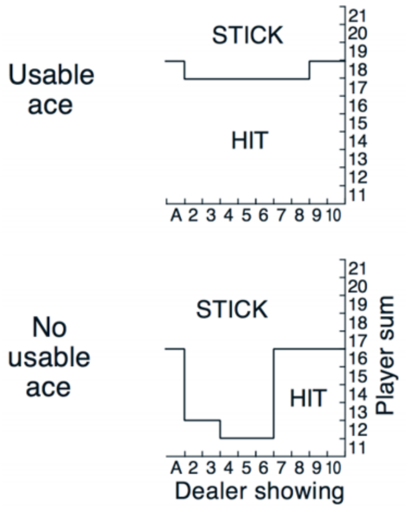
Solution: Temporal-Difference

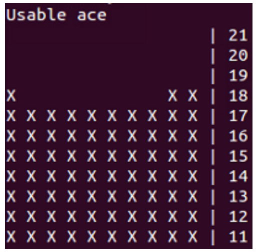
- TD works in continuing (non-terminating) environments
- TD can learn online after every step
- TD can learn from incomplete sequences

Assignment: Blackjack

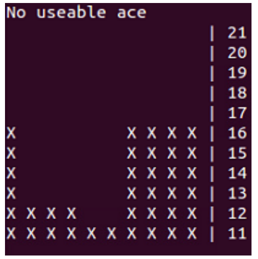
Play Blackjack using Monte Carlo with exploring starts.

- Implement the part for updating Q(s,a) value inside the function monte_carlo_es(n_iter).
- Try different methods to select the start state and action. (in the code it is totally random)
- Play with different reward and iteration number





```
Usable ace
                    | 21
                    | 20
                    | 19
X               X X | 18
X X X X X X X X X X | 17
X X X X X X X X X X | 16
X X X X X X X X X X | 15
X X X X X X X X X X | 14
X X X X X X X X X X | 13
X X X X X X X X X X | 12
X X X X X X X X X X | 11
```



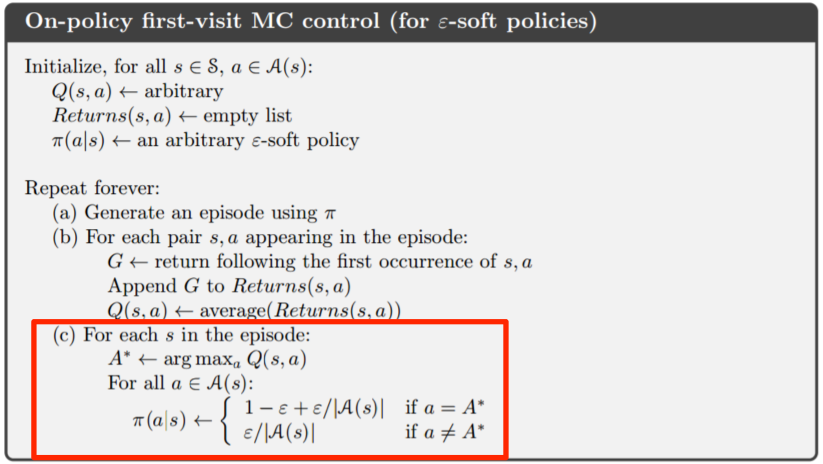
```
No useable ace
                    | 21
                    | 20
                    | 19
                    | 18
                    | 17
X           X X X X | 16
X           X X X X | 15
X           X X X X | 14
X           X X X X | 13
X X X X     X X X X | 12
X X X X X X X X X X | 11
```

You should get the similar result to the example in the book.

Assignment: Blackjack

Modify the code and implement “Monte Carlo without exploring starts” using on-policy learning with ε-greedy policies.

What is the difference between these two methods?



On-policy first-visit MC control (for ε-soft policies)

```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list
    $\pi(a|s) \leftarrow$ an arbitrary $\varepsilon$-soft policy

Repeat forever:
    (a) Generate an episode using $\pi$
    (b) For each pair $s, a$ appearing in the episode:
            $G \leftarrow$ return following the first occurrence of $s, a$
            Append $G$ to $Returns(s,a)$
            $Q(s,a) \leftarrow \text{average}(Returns(s,a))$
    (c) For each $s$ in the episode:
            $A^* \leftarrow \arg\max_a Q(s,a)$
            For all $a \in \mathcal{A}(s)$:
                $\pi(a|s) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq A^* \end{cases}$
```

References

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT, 2016

Online lectures:
M. Heinzer, E. Profumo. Reinforcement Learning – Monte Carlo Methods, 2016 [PDF slides]. Retrieved from https://stat.ethz.ch/education/semesters/ss2016/seminar/files/slides/RL_MCM_heinzer_profumo.pdf

D. Silver. Reinforcement Learning Course, Lecture 4-5, 2015 [YouTube video] Retrieved from https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT