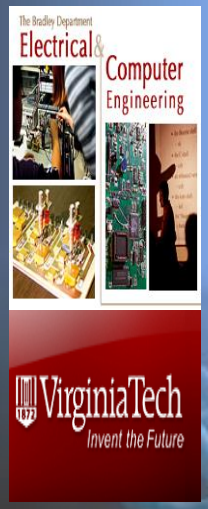
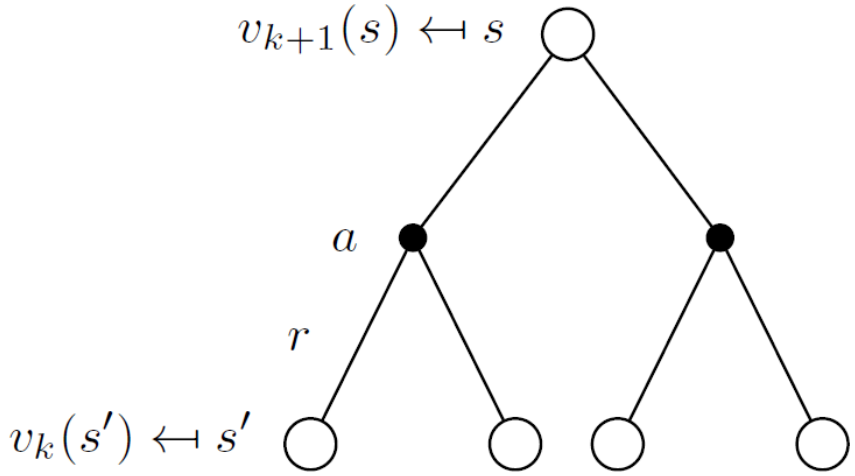


ECE 5984
Model-Free Reinforcement Learning
Monte Carlo Methods

Jason J. Xuan, Ph.D.

Department of Electrical & Computer Engineering
Virginia Tech

MDP: Dynamic Programming

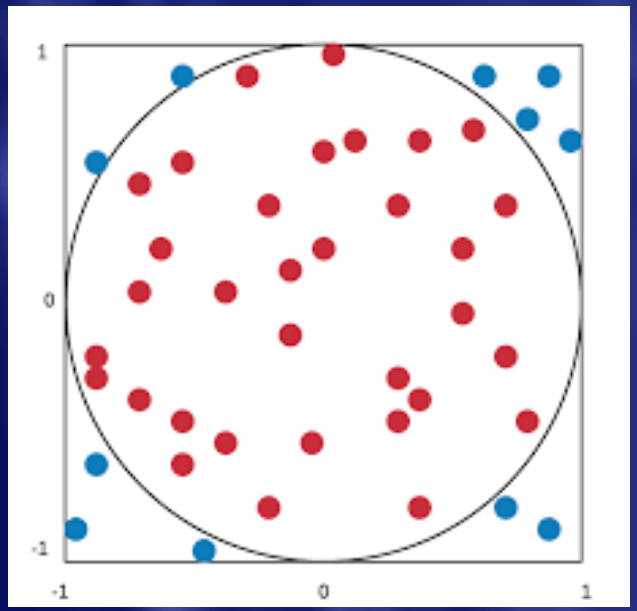


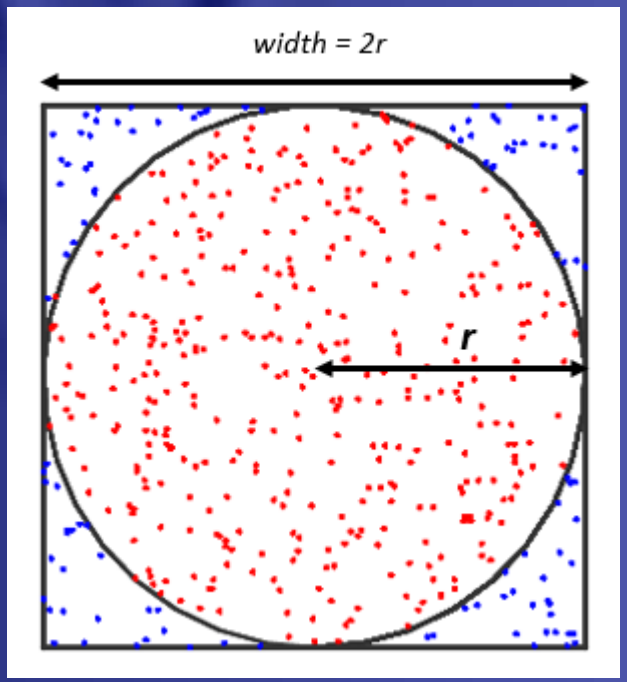
$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

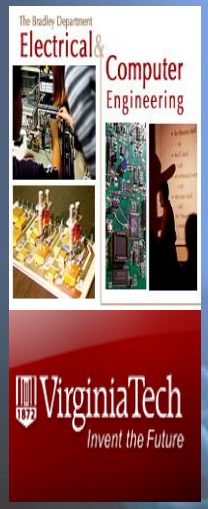
Monte Carlo Method

- Estimating π



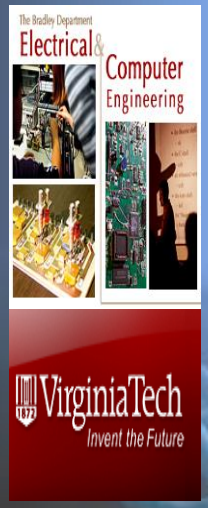


$\pi \approx 4 \times$ (number of points in the circle / total number of points)



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 Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

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

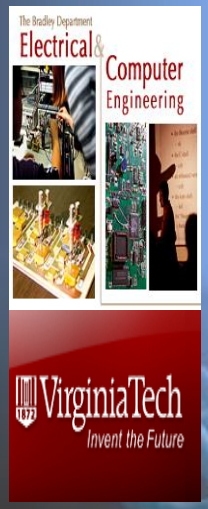
- Recall that the *return* is the total discounted reward:

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

- Recall that the value function is the expected return:

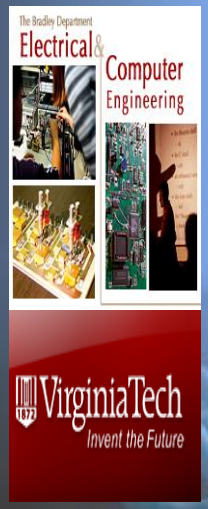
$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

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



First-Visit Monte-Carlo Policy Evaluation

- To evaluate state s
- The first 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, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

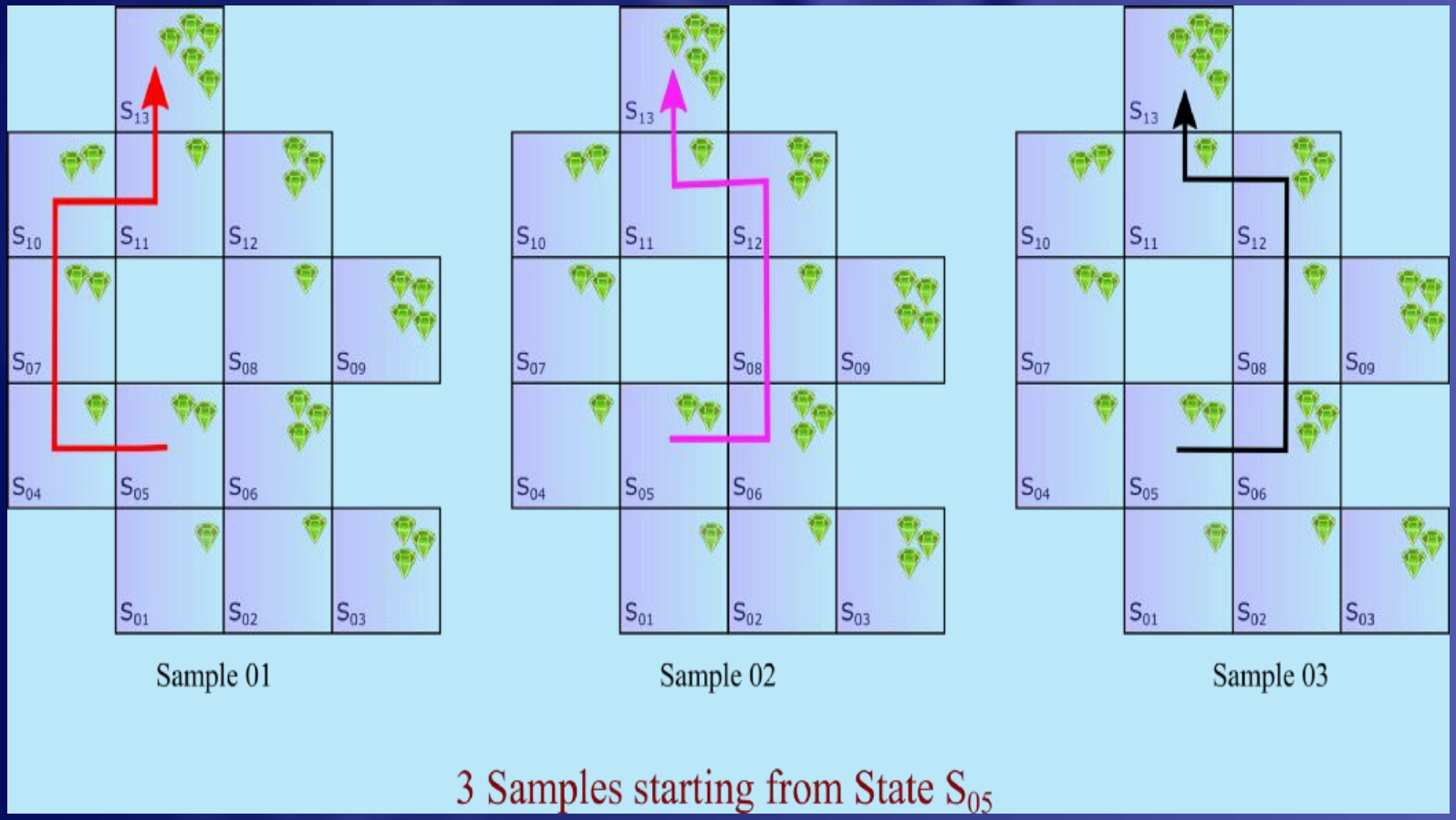


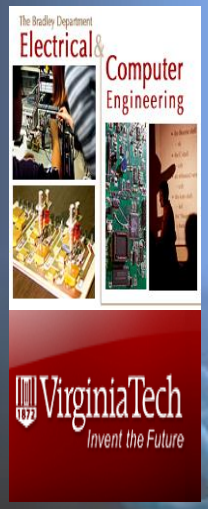
Every-Visit Monte-Carlo Policy Evaluation

- To evaluate state s
- 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 $V(s) = S(s)/N(s)$
- Again, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

An Example

Gem Collection





Expected Return

Return(Sample 01) = 2 + 1 + 2 + 2 + 1 + 5 = 13 gems

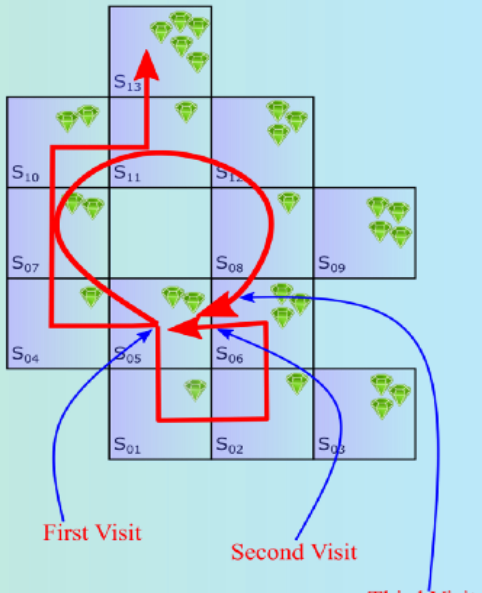
Return(Sample 02) = 2 + 3 + 1 + 3 + 1 + 5 = 15 gems

Return(Sample 03) = 2 + 3 + 1 + 3 + 1 + 5 = 15 gems

Observed mean return (based on 3 samples) = (13 + 15 + 15)/3 = 14.33 gems

Thus state value as per Monte Carlo Method, $v_\pi(S\ 05)$ is 14.33 gems based on 3 samples following policy π.

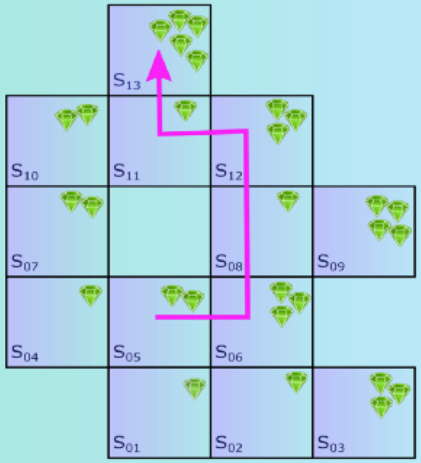
First-Visit Monte-Carlo Method



Sample 01

Return for the state S_{05} = 33 gems

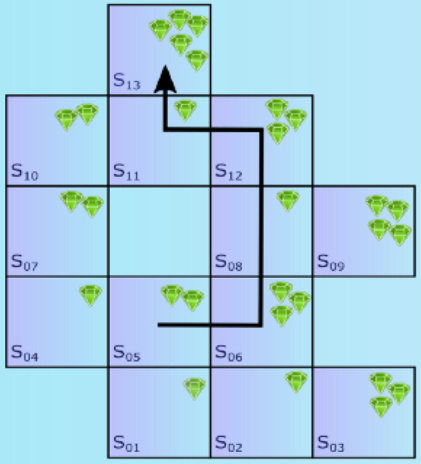
$N(S_{05})$ = 1 (only first visit is counted)



Sample 02

Return for the state S_{05} = 15 gems

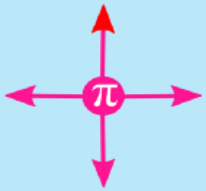
$N(S_{05})$ = 2 (added 1 in this episode)



Sample 03

Return for the state S_{05} = 15 gems

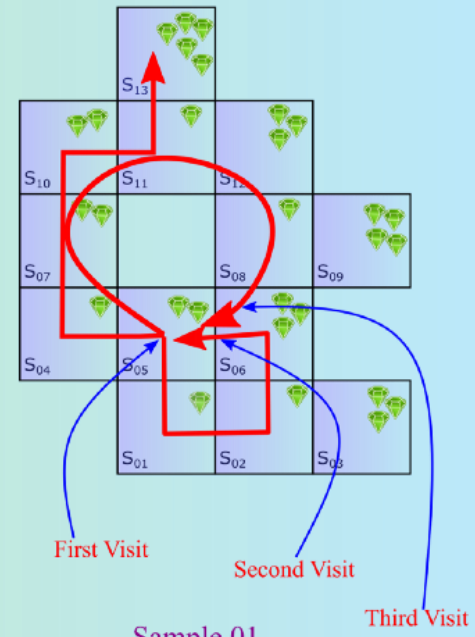
$N(S_{05})$ = 3 (added 1 in this episode)



$$v(S_{05}) = (33 + 15 + 15) / 3 = 21 \text{ gems}$$

First Visit Monte Carlo State Value Evaluation of State S_{05}

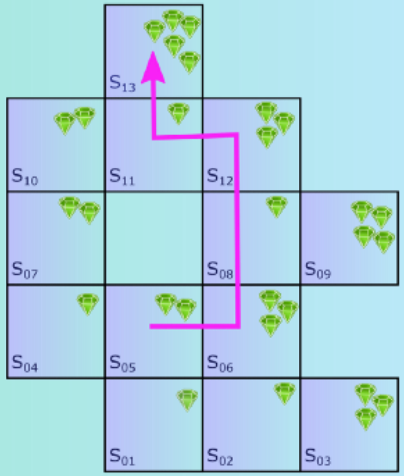
Every-Visit Monte-Carlo Method



Sample 01

Return for the state S_{05} = 33 gems

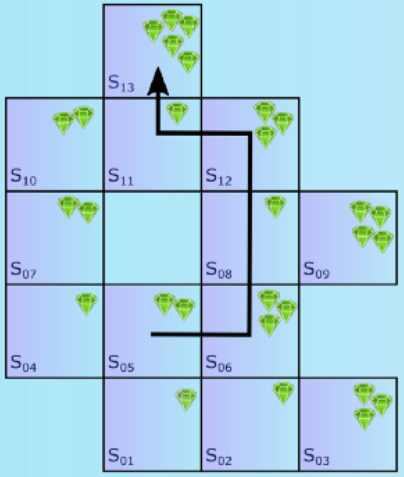
$N(S_{05})$ = 3 (total 3 visit)



Sample 02

Return for the state S_{05} = 15 gems

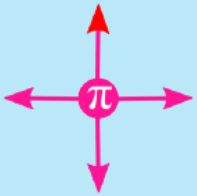
$N(S_{05})$ = 4 (added 1 in this episode)



Sample 03

Return for the state S_{05} = 15 gems

$N(S_{05})$ = 5 (added 1 in this episode)



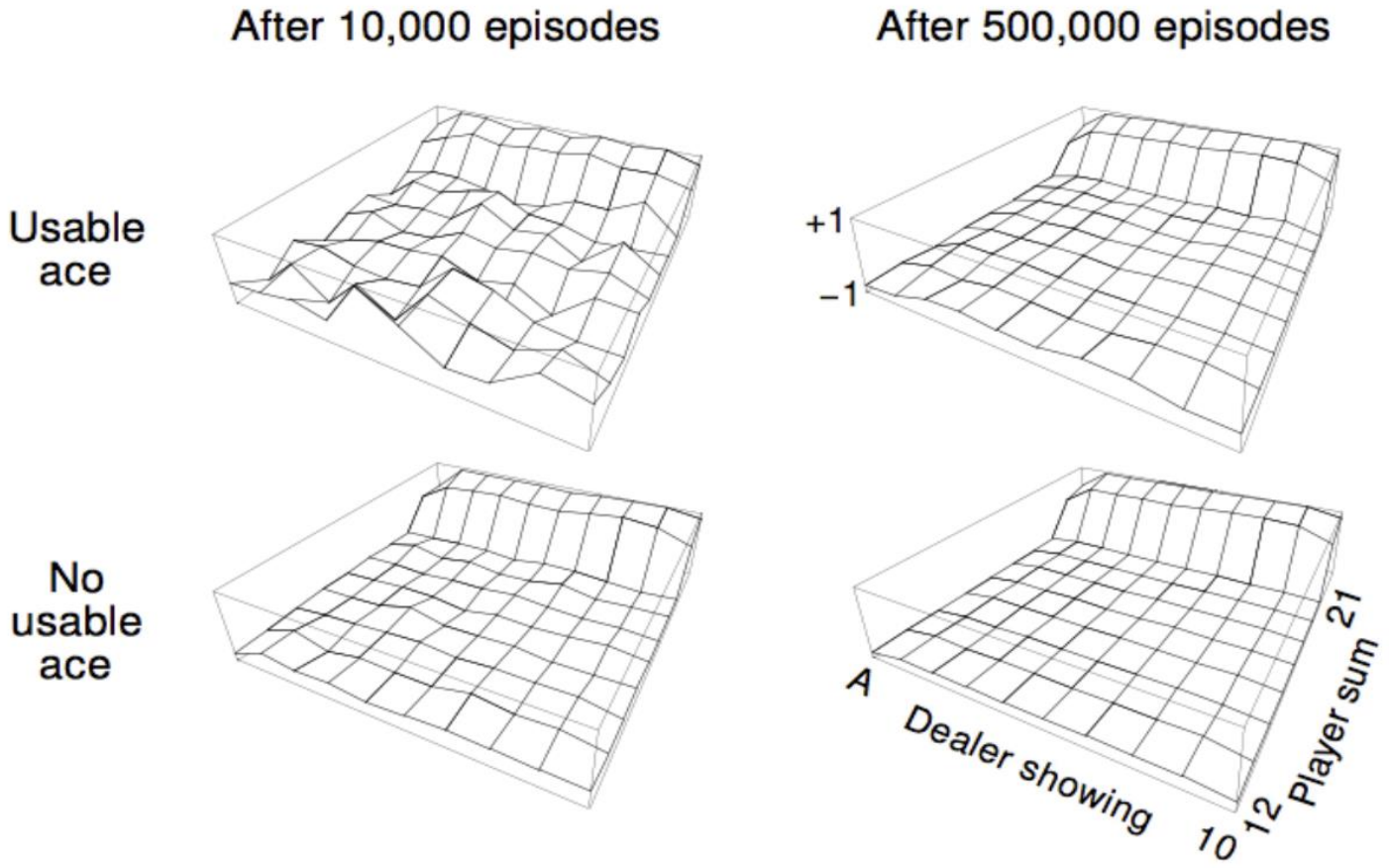
$$v(S_{05}) = (33 + 15 + 15) / 5 = 12.6 \text{ gems}$$

Every Visit Monte Carlo State Value Evaluation of State S_{05}

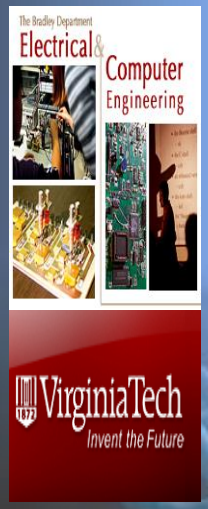
Example: Blackjack

- 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 twist: 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 twist:
  - -1 if sum of cards > 21 (and terminate)
  - 0 otherwise
- Transitions: automatically twist if sum of cards < 12

Value Function: Monte-Carlo Learning



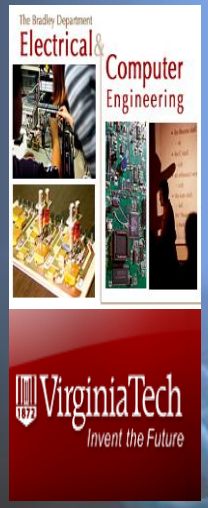
Policy: stick if sum of cards $\geq$ 20, otherwise twist



Incremental Mean

The mean $\mu_1, \mu_2, \ldots$ of a sequence $x_1, x_2, \ldots$ can be computed incrementally,

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



Incremental Monte-Carlo Update

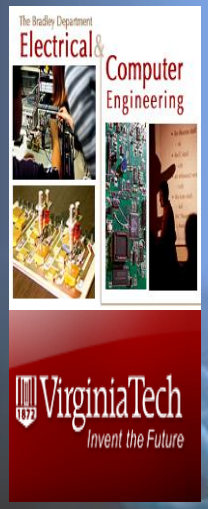
- Update $V(s)$ incrementally after episode $S_1, A_1, R_2, ..., S_T$
- For each state S_t with return G_t

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

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

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

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



Pros and Cons

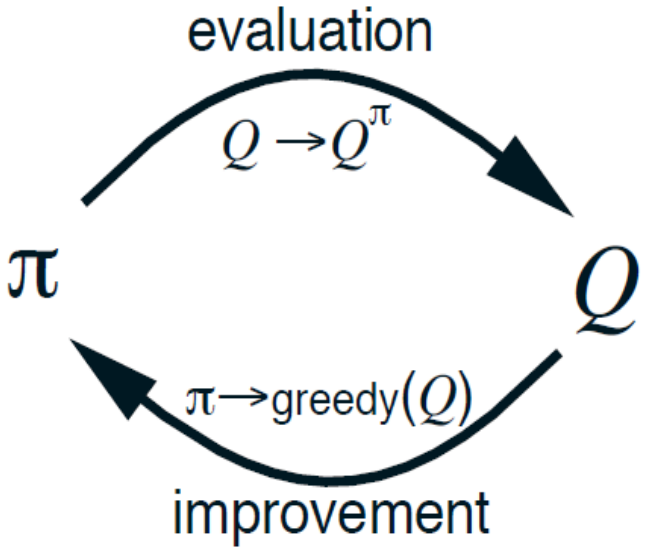
Monte Carlo Methods have below **advantages** :

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

But it has below **limitations** as well:

- MC must wait until end of episode before return is known
- MC has high variance
- MC can only learn from complete sequences
- MC only works for episodic (terminating) environments

Monte Carlo Control



- MC policy iteration: Policy evaluation using MC methods followed by policy improvement
- Policy improvement step: greedify with respect to value (or action-value) function

On-Policy Monte Carlo Control

- *On-policy:* learn about policy currently executing
- How do we get rid of exploring starts?
  - Need *soft* policies: $\pi(s,a) > 0$ for all s and a
  - e.g. ε-soft policy:

$$\frac{\varepsilon}{|A(s)|} \qquad 1 - \varepsilon + \frac{\varepsilon}{|A(s)|}$$

non-max greedy

- Similar to GPI: move policy *towards* greedy policy (i.e. ε-soft)
- Converges to best ε-soft policy

On-Policy Monte Carlo Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s,a) \leftarrow$ arbitrary

$Returns(s,a) \leftarrow$ empty list

$\pi \leftarrow$ an arbitrary ϵ-soft policy

Repeat forever:

(a) Generate an episode using π

(b) For each pair s, a appearing in the episode:

$R \leftarrow$ return following the first occurrence of s, a

Append R to $Returns(s,a)$

$Q(s,a) \leftarrow \text{average}(Returns(s,a))$

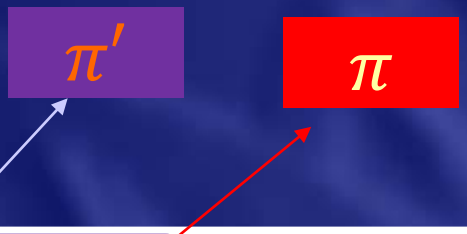
(c) For each s in the episode:

$a^* \leftarrow \arg\max_a Q(s,a)$

For all $a \in \mathcal{A}(s)$:

$$\pi(s,a) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \epsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

Off-Policy Monte Carlo Control



- Behavior policy generates behavior in environment
- Estimation policy is policy being learned about
- Average returns from behavior policy by probability: their probabilities in the estimation policy

Off-Policy Monte Carlo Control

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s,a) \leftarrow$ arbitrary

$N(s,a) \leftarrow 0$; Numerator and

$D(s,a) \leftarrow 0$; Denominator of $Q(s,a)$

$\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:

(a) Select a policy π' and use it to generate an episode:

$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$

(b) $\tau \leftarrow$ latest time at which $a_\tau \neq \pi(s_\tau)$

(c) For each pair s, a appearing in the episode after τ:

$t \leftarrow$ the time of first occurrence (after τ) of s, a

$w \leftarrow \Pi_{k=t+1}^{T-1} \frac{1}{\pi'(s_k,a_k)}$

$N(s,a) \leftarrow N(s,a) + wR_t$

$D(s,a) \leftarrow D(s,a) + w$

$Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$

(d) For each $s \in \mathcal{S}$:

$\pi(s) \leftarrow \arg\max_a Q(s,a)$

$$w \leftarrow \prod_{k=t+1}^{T-1} \frac{\pi(s_k,a_k)}{\pi'(s_k,a_k)}$$

Important sampling ratio

More on Playing Blackjack

Basic strategy

Actions: hit (H), stick (ST), double (D)

Dealer's Upcard

Player's Current Sum

	2	3	4	5	6	7	8	9	10	A
17+	ST	ST	ST	ST	ST	ST	ST	ST	ST	ST
16	ST	ST	ST	ST	ST	H	H	H	H	H
15	ST	ST	ST	ST	ST	H	H	H	H	H
14	ST	ST	ST	ST	ST	H	H	H	H	H
13	ST	ST	ST	ST	ST	H	H	H	H	H
12	H	H	ST	ST	ST	H	H	H	H	H
11	D	D	D	D	D	D	D	D	D	H
10	D	D	D	D	D	D	D	D	H	H
9	H	D	D	D	H	H	H	H	H	H
5-8	H	H	H	H	H	H	H	H	H	H

First-visit: Playing Blackjack

Simplification: hit or stick

Policy: if our hand ≥ 18, stick with a probability of 80% else hit with a probability of 80%

Algorithm:

1.Provide a policy π

2.Create empty dictionaries N, returns_sum and Q to hold the amount of times a State/Action pair has been visited, the amount of returns that State/Action pair has received and finally the value of that State/Action pair.

3.Play a game of Blackjack using our current policy for X games

4.Loop through each turn in the game and check if the current State/Action pair has been seen in that game yet

5.If this is the first time we have seen that pair we increase the count for that pair and add the discounted rewards of each step from that turn onwards to our returns_sum dictionary

6.Finally update the Q values with the average of the returns and amount of each State/Action pair.

Implementation: Playing Blackjack

```
import sys
import gym
import numpy as np

from collections import defaultdict

from plot_utils import plot_blackjack_values, plot_policy

env = gym.make('Blackjack-v0')
```

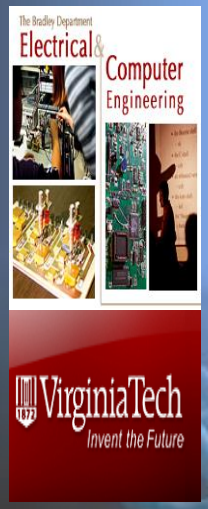
https://towardsdatascience.com/learning-to-win-blackjack-with-monte-carlo-methods-61c90a52d53e

Implementation: play_episode()

```
def play_episode(env):
    """
    Plays a single episode with a set policy in the environment given.
Records the state, action
    and reward for each step and returns the all timesteps for the episode.
    """
    episode = []
    state = env.reset()
    while True:
        probs = [0.8, 0.2] if state[0] >= 18 else [0.2, 0.8]
        action = np.random.choice(np.arange(2), p=probs)
        next_state, reward, done, info = env.step(action)
        episode.append((state, action, reward))
        state = next_state
        if done:
            break
    return episode
```

policy π

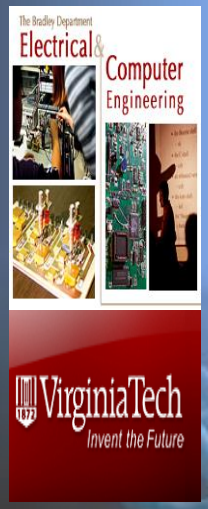
Policy: if our hand ≥ 18, stick with a probability of 80% else hit with a probability of 80%



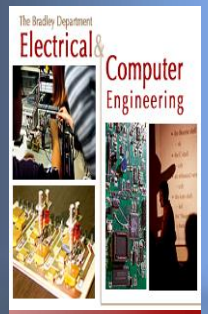

```
def update_Q(episode, Q,returns_sum, N, gamma=1.0):
    """
    For each time step in the episode we carry out the first visit monte carlo method, checking if this is
    the first index of this state. Get the discounted reward and add it to the total reward for that
    state/action pair. Increment the times we have seen this state action pair and finally update the Q values
    """
    for s, a, r in episode:
            first_occurence_idx = next(i for i,x in enumerate(episode) if x[0] == s)
            G = sum([x[2]*(gamma**i) for i,x in
              enumerate(episode[first_occurence_idx:])])
            returns_sum[s][a] += G
            N[s][a] += 1.0
            Q[s][a] = returns_sum[s][a] / N[s][a]
```

reward

Gt

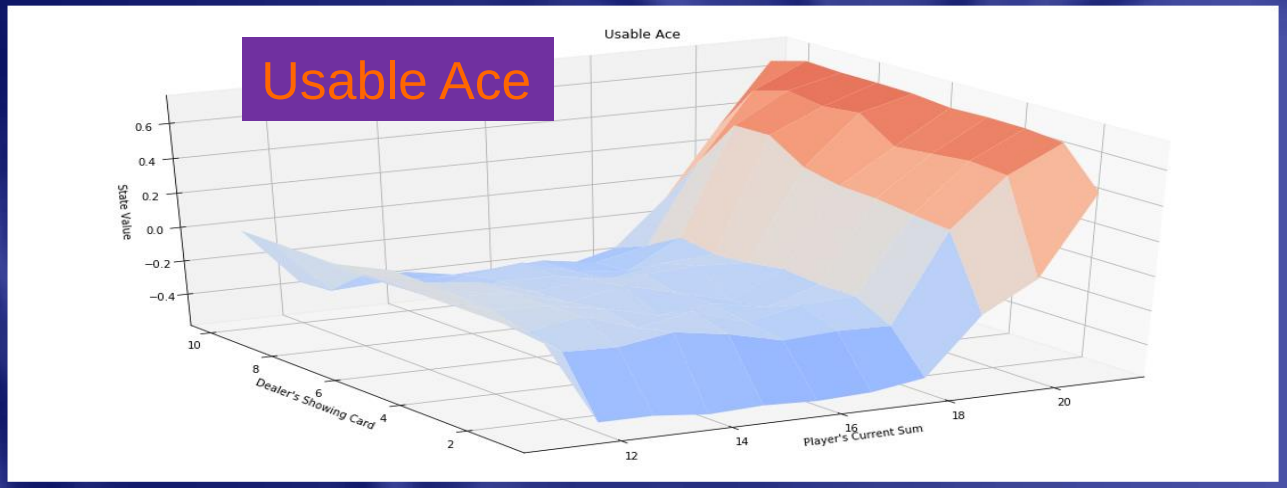


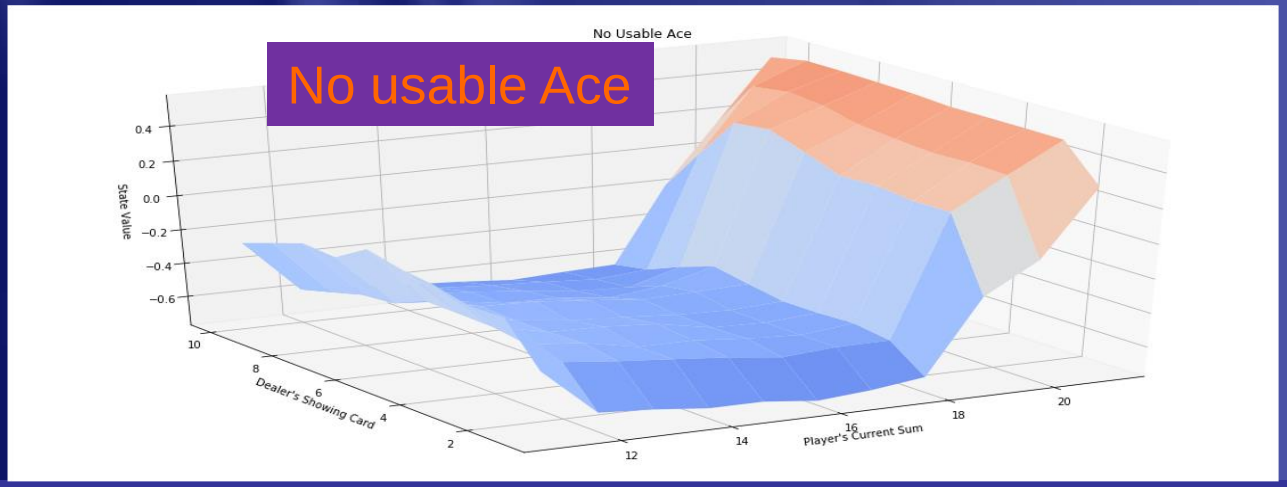

```
def mc_predict(env, num_episodes, gamma=1.0):
    """
    This is the primary method. Plays through several episodes of
the environment.
    """
    returns_sum = defaultdict(lambda:
            np.zeros(env.action_space.n))
    N = defaultdict(lambda: np.zeros(env.action_space.n))
    Q = defaultdict(lambda: np.zeros(env.action_space.n))

    for i_episode in range(1, num_episodes+1):
        if i_episode % 1000 == 0:
            print("\rEpisode {}/{}.".format(i_episode, num_episodes),
             end="")
            sys.stdout.flush()

        episode = play_episode(env)

        update_Q(episode, Q, returns_sum, N)

    return Q
```

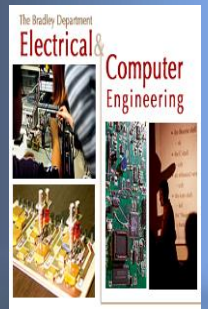



```
#predict the policy values for our test policy
Q = mc_predict(env, 500000)

#get the state value function for our test policy
V_to_plot = dict((k,(k[0]>18)*(np.dot([0.8, 0.2],v)) +
          (k[0]<=18)*(np.dot([0.2, 0.8],v))) \
         for k, v in Q.items())

# plot the state value functions
plot_blackjack_values(V_to_plot)
```

Value Prediction







Question

- Comments are more than welcome!