COMP 3200
Artificial Intelligence

Lecture 19
Reinforcement Learning
Monte Carlo Methods

Monte Carlo Methods

Chapter 5 in RL Textbook

http://incompleteideas.net/book/first/ebook/node50.html

Monte Carlo Methods

- Methods for estimating value functions and discovering optimal policies
- Unlike DP, we do not assume complete knowledge of the environment or a model
- MC methods require only experience, sequence of states, actions, rewards from an actual or simulated environment

Actual vs. Simulated Experience

- Actual Experience: physical robot, etc
- Learning from actual experience does not require a model of the environment
- We do not need to know the dynamics, physics, layout, or full state space of env
- Agent will take an action, carry it out, and get a reward from the environment

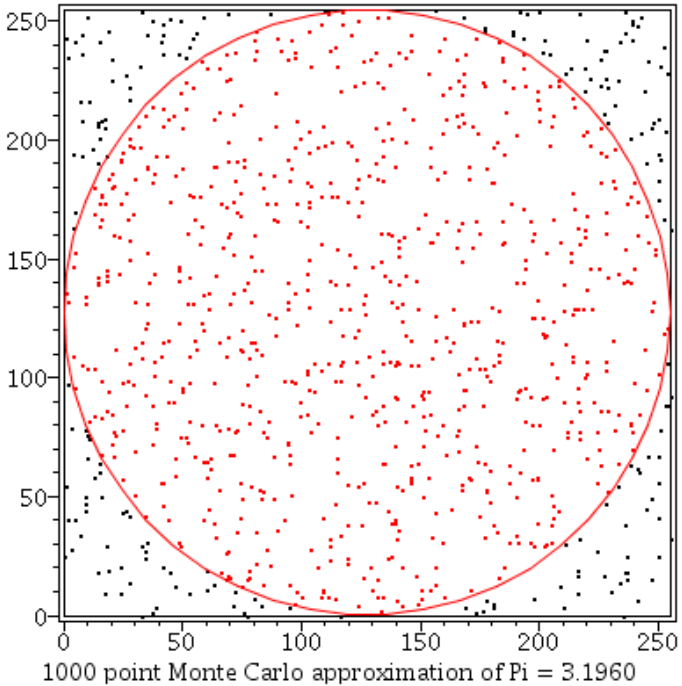
Actual vs. Simulated Experience

- Simulated Experience: Video Game / Sim
- Learning from simulated experience is also powerful, because we do not need to know HOW the model works, we only need the state transition function (game engine)
- MC doesn’t need to know all action / state probability distributions, whereas DP does

MC Methods use Sampling

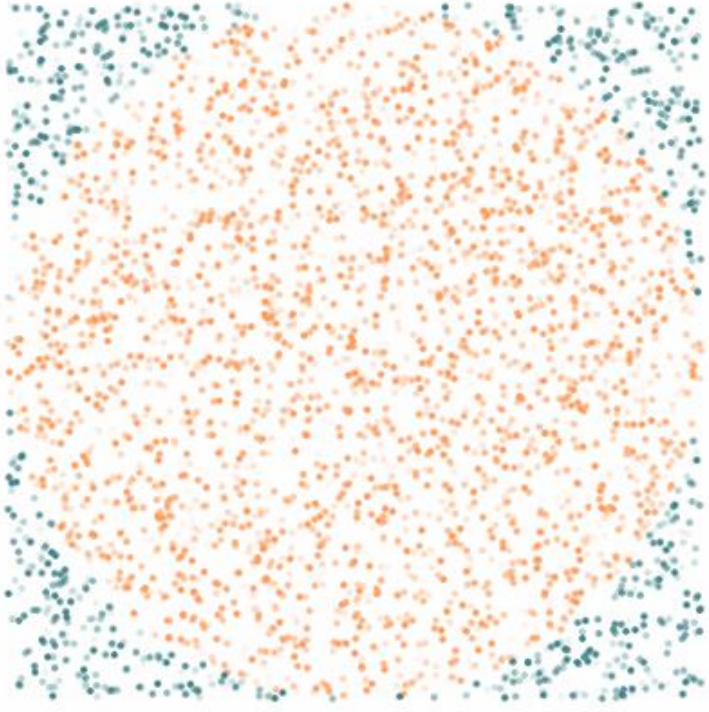
- Square length 1x1
- Circle radius 0.5
- Square area = 1x1 = 1
- Circle Area = $\pi r^2 = \pi/4$

$$\frac{\pi}{4} \approx \frac{N_{inner}}{N_{total}} \qquad \pi \approx 4\frac{N_{inner}}{N_{total}}$$



https://academo.org/demos/estimating-pi-monte-carlo/

Monte Carlo Simulation of Pi



Animation frames

100

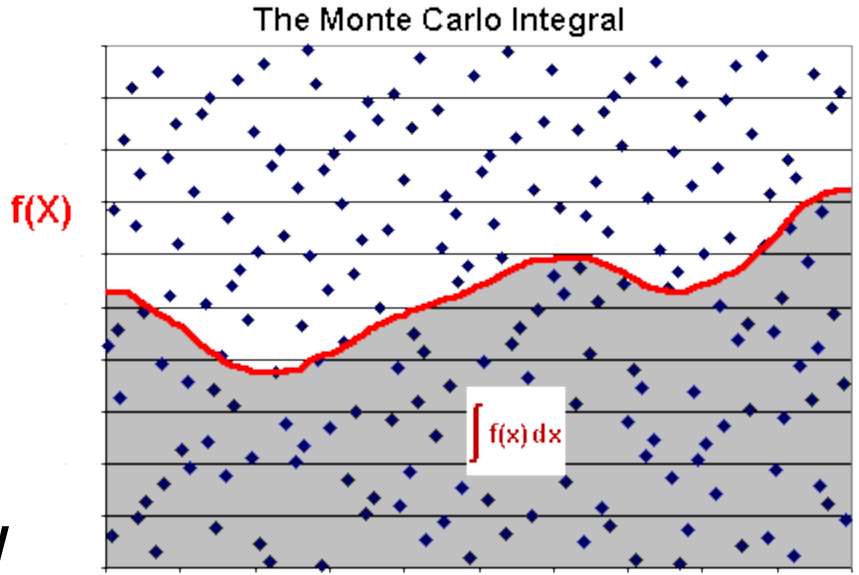
6

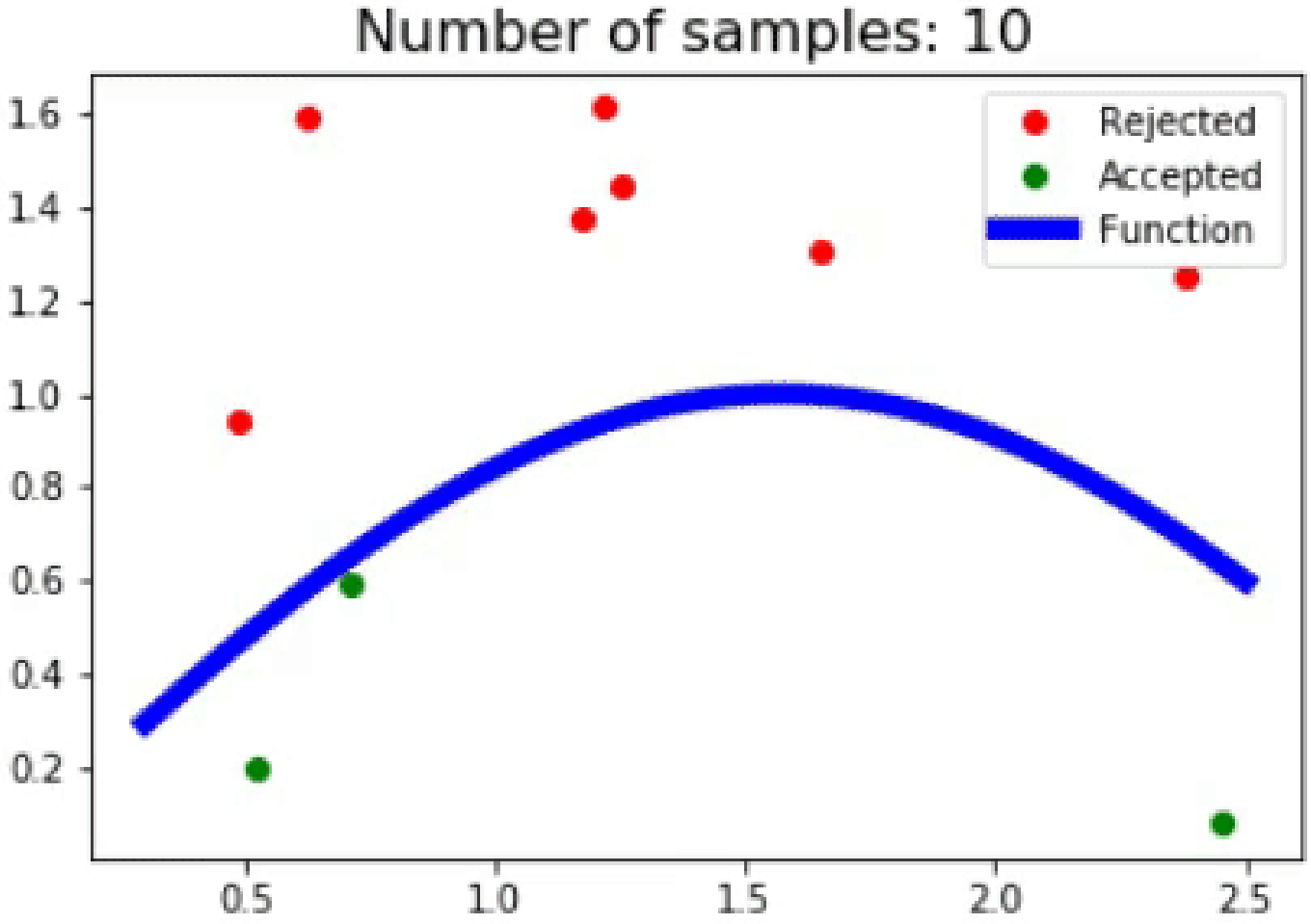
Show history

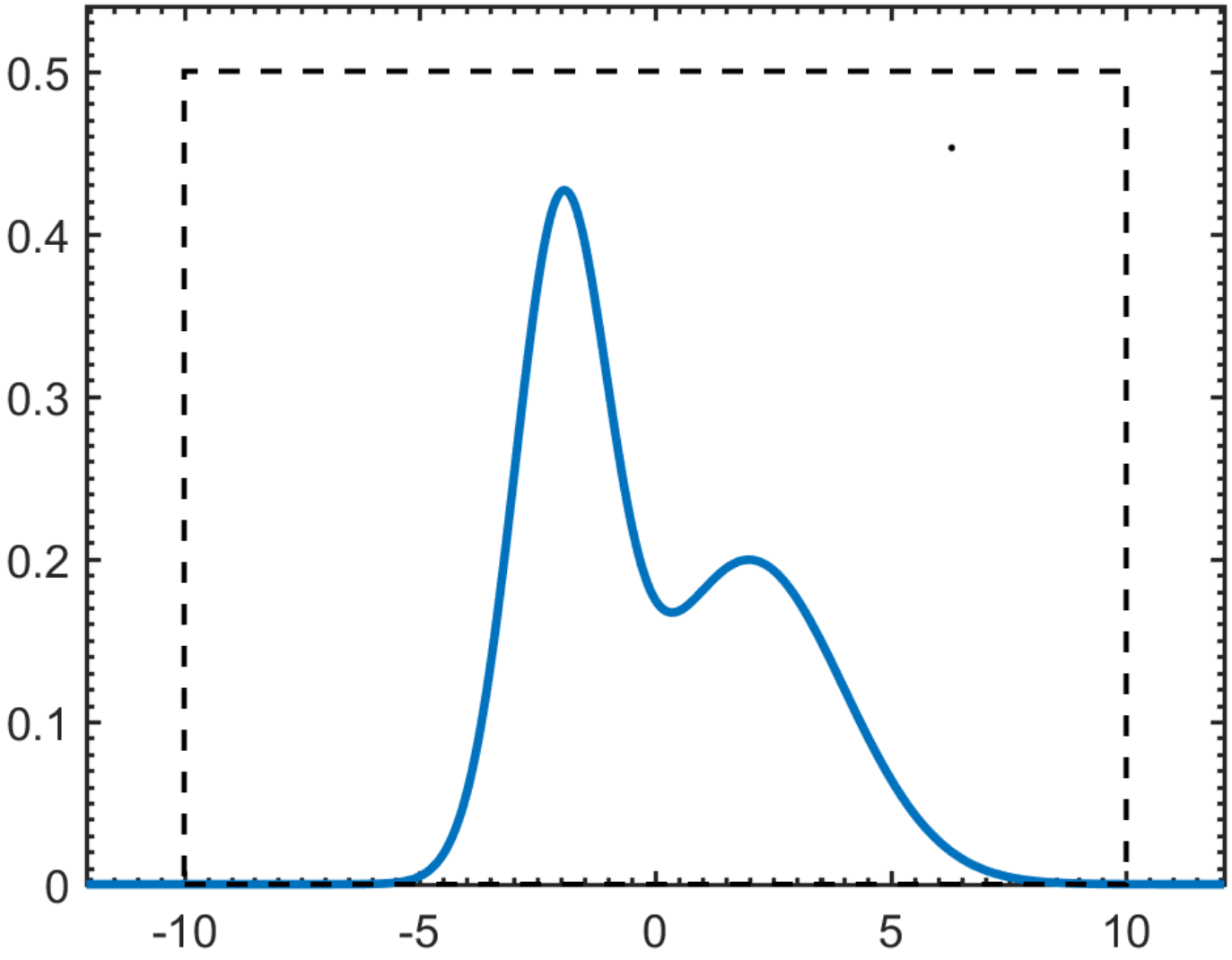
Numbers Inside Circle	2,751
Numbers Total	3,500
Ratio	0.78600
PI (Ratio*4)	3.14400

MC Methods use Sampling

- Generate randomized points in area
- Use f(x) to determine if above or below line
- Area is the ratio of points above or below times total area




Number of samples: 10
Rejected
Accepted
Function
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.5
1.0
1.5
2.0
2.5


0.5
0.4
0.3
0.2
0.1
0
-10
-5
0
5
10

Monte Carlo Methods

- Able to solve RL problems using samples
- Learns based on averaging sample returns
- We will discuss MCM for episodic tasks
  - All experience divided into finite episodes
  - All episodes eventually terminate
- Value estimate and policy updates are performed only at the end of an episode

Monte Carlo Methods

- MCM sample and average returns for each state-action pair, like bandit methods do
- MCM treat each state as a bandit problem
- Each of these bandit problems are related, because their return after actions depend on neighbour future actions

Monte Carlo Prediction

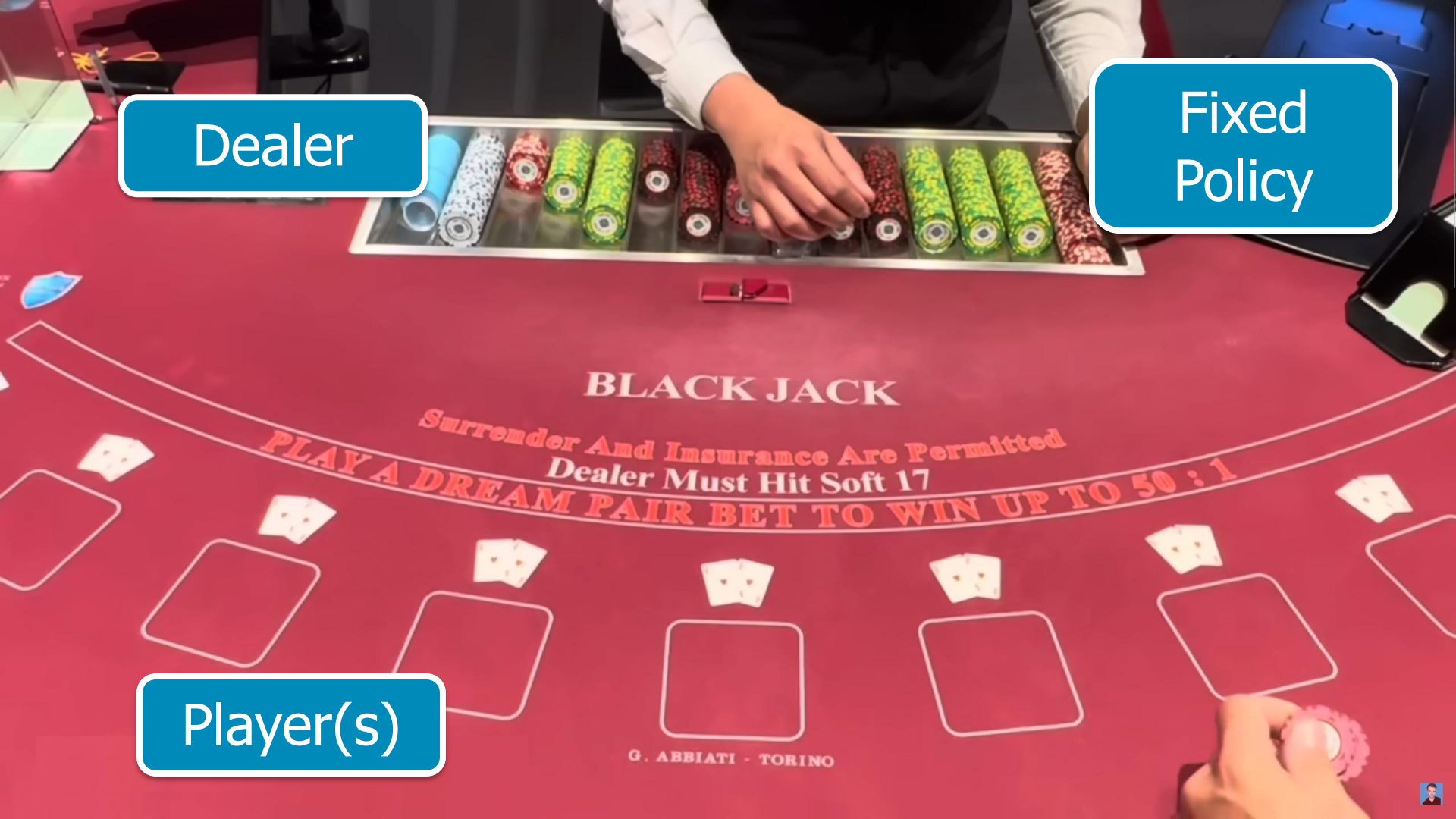
- Goal: learn state-values for a given policy
- Value of a state = expected return (sum of discounted rewards) starting from the state
- Intuitive way to obtain state value:
  - Carry out an episode using a given policy
  - Record all states visited in the episode
  - Average the return after visiting the state
  - Update the value estimate of the state with the average
- This converges to the expected value over time

Syntax Note

- What is a ‘value’ ?
  - Expected return from a specific situation
- $V_\pi(s)$ = expected future return when starting in state s and following policy π
- $q_\pi(s,a)$ = expected future return when starting in state s, taking action a, and following with policy π

MC Example: Blackjack

- Blackjack played in ‘hands’ (episodes)
- Reward 0 at every state, except terminal
- Win gets +1 reward, loss gets -1 reward
- ‘Blackjack’ win gets +1.5 reward


Dealer
Fixed Policy
BLACK JACK
Surrender And Insurance Are Permitted
Dealer Must Hit Soft 17
PLAY A DREAM PAIR BET TO WIN UP TO 50 : 1
G. ABBIATI - TORINO
Player(s)


LACK JACK
Insurance Are Permitted
Must Hit Soft 17
PAIR BET TO WIN UP TO 50 : 1
G. ABBIATI - TORINO

Dealer Cards

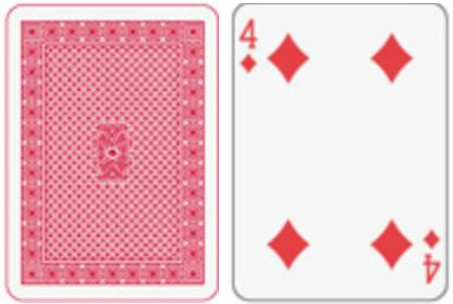
Player Cards

Dealer Cards

Player Cards

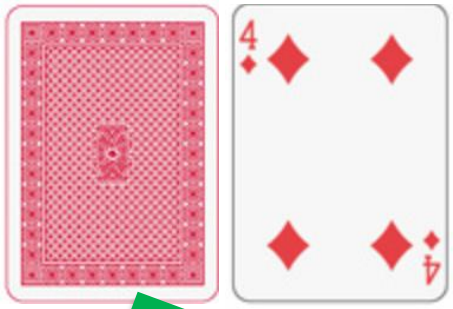
Cards shuffled, then dealt to Dealer and Player

Dealer Cards



Player Cards

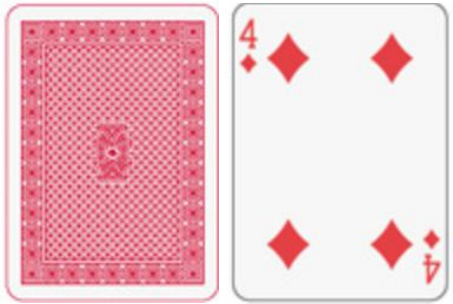
Dealer Cards



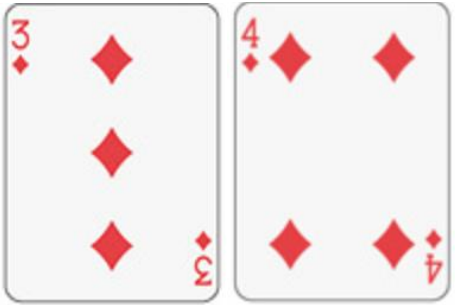
Player Cards

One Dealer Card
Hidden from Player

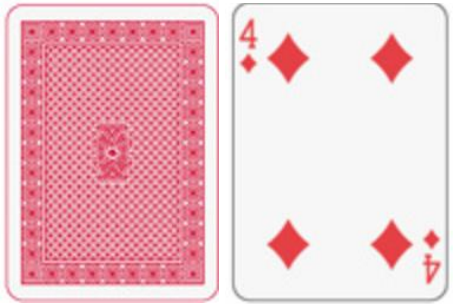
Dealer Cards



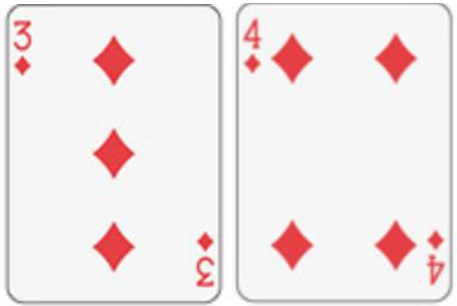
Player Cards



Dealer Cards

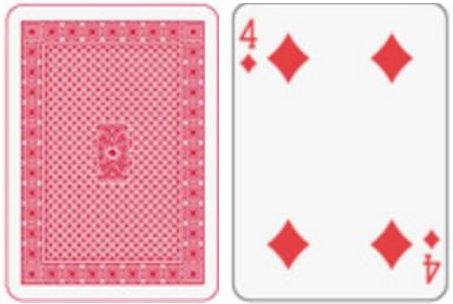


Player Cards

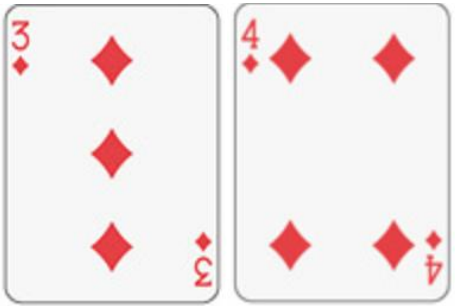


Starting State s_0 = Player 7, Dealer 4

Dealer Cards

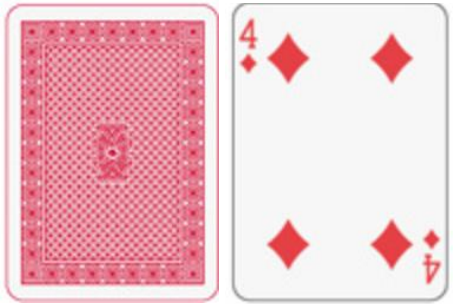


Player Cards

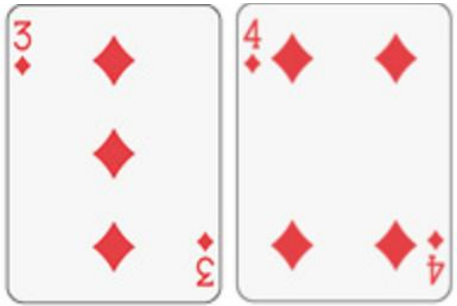


$s_0[7][4]$

Dealer Cards



Player Cards



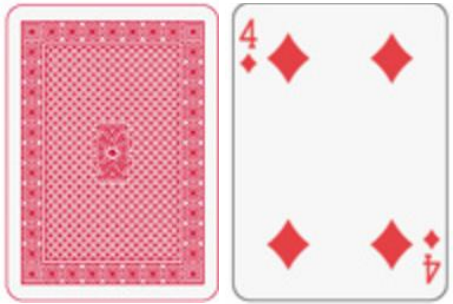
Actions

Hit

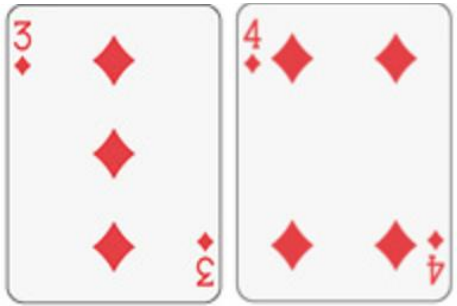
Stand

$s_0[7][4]$

Dealer Cards



Player Cards



$s_0[7][4]$

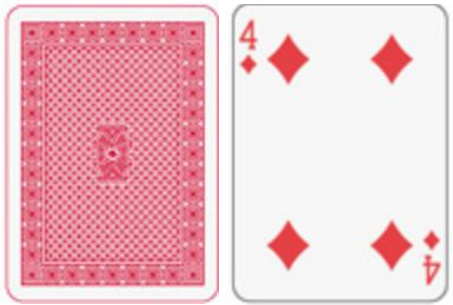
a_0

Actions Chosen From Policy π

Hit

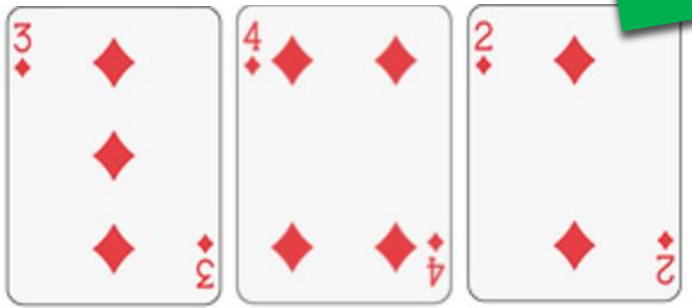
Stand

Dealer Cards



State Updated

Player Cards



Reward From Environment

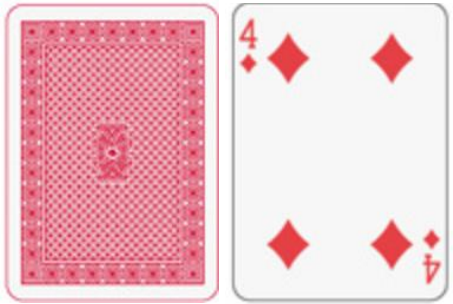
$s_1[9][4]$

$r_1=0$

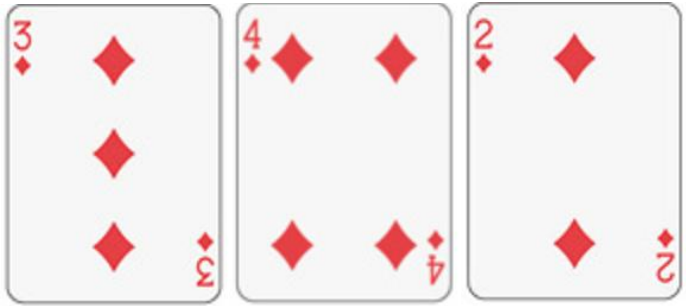
Hit

Stand

Dealer Cards



Player Cards



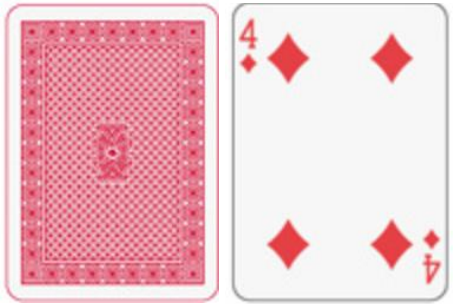
$s_1[9][4]$

$r_1=0$

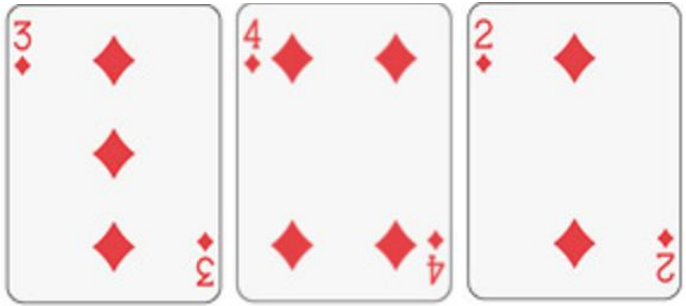
Hit

Stand

Dealer Cards



Player Cards



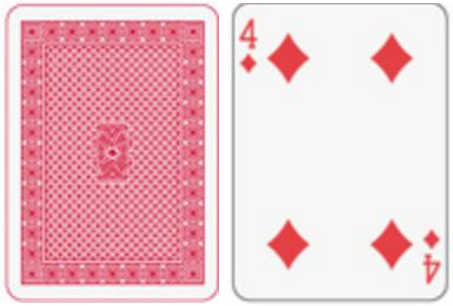
$s_1[9][4]$

a_1

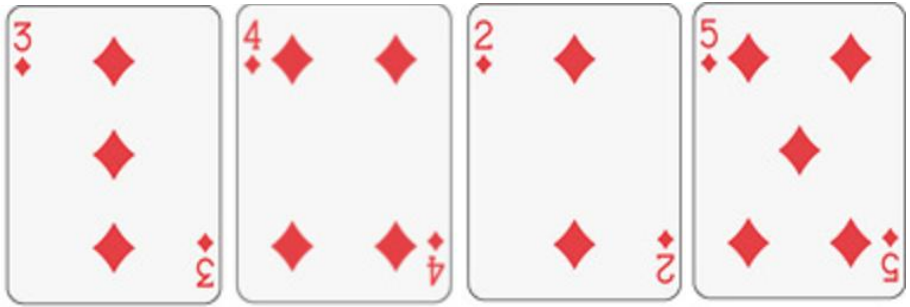
Hit

Stand

Dealer Cards



Player Cards

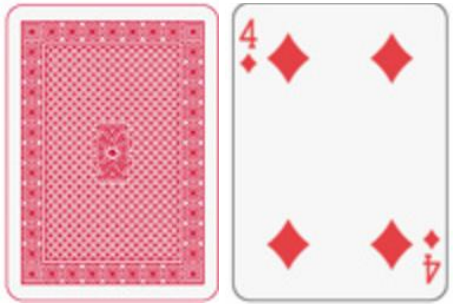


$s_2[14][4]$

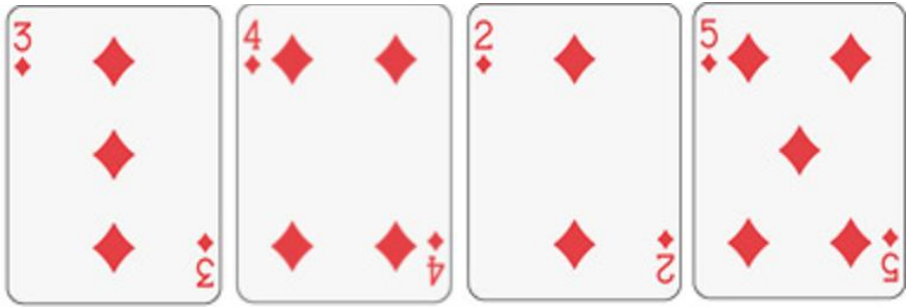
$r_2=0$

Hit Stand

Dealer Cards



Player Cards



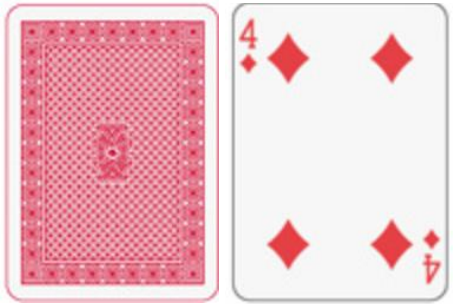
$s_2[14][4]$

a_2

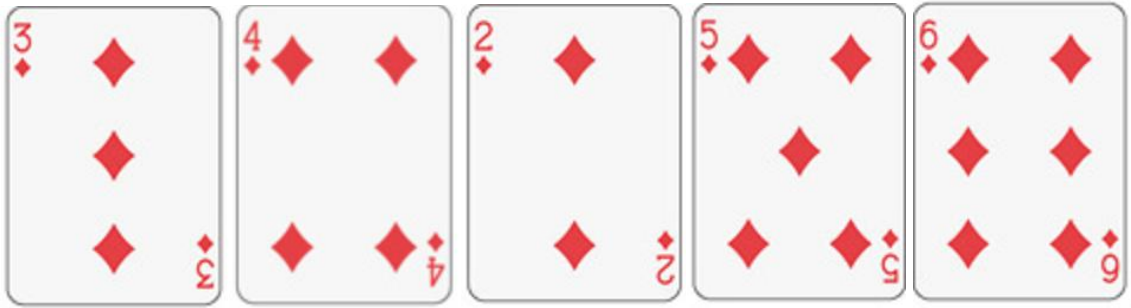
Hit

Stand

Dealer Cards



Player Cards



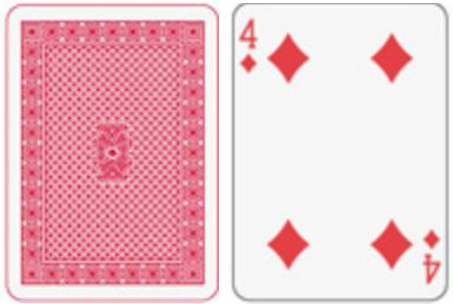
$s_3[20][4]$

$r_3=0$

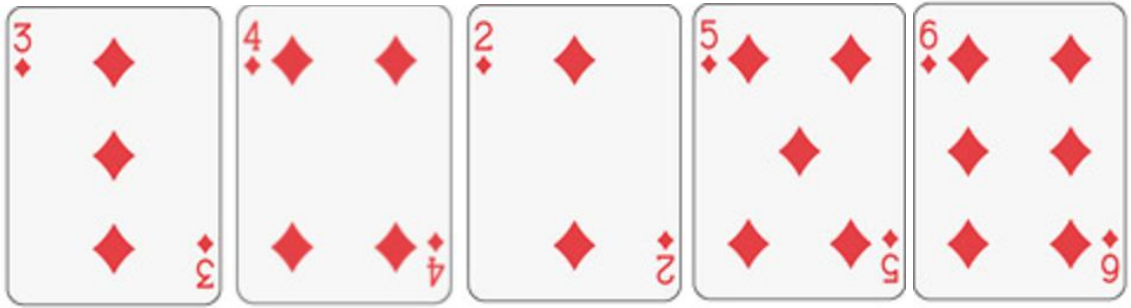
Hit

Stand

Dealer Cards



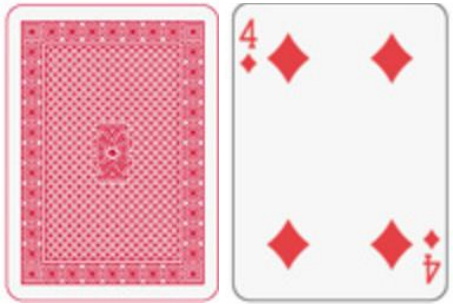
Player Cards



$s_3[20][4]$

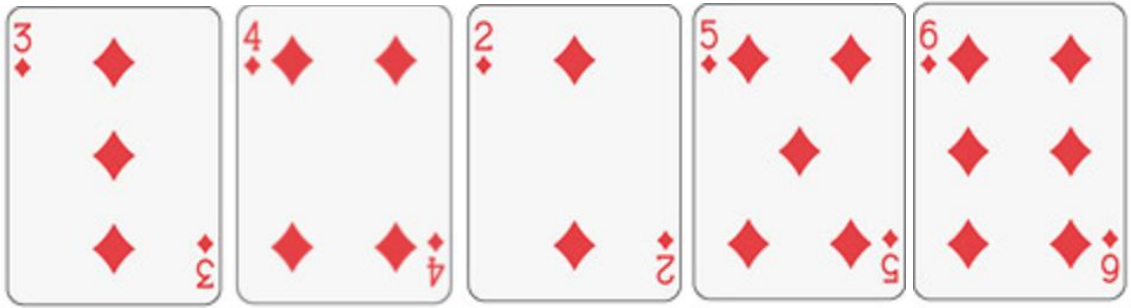
a_3

Dealer Cards



Dealer Hit Until >= 17

Player Cards



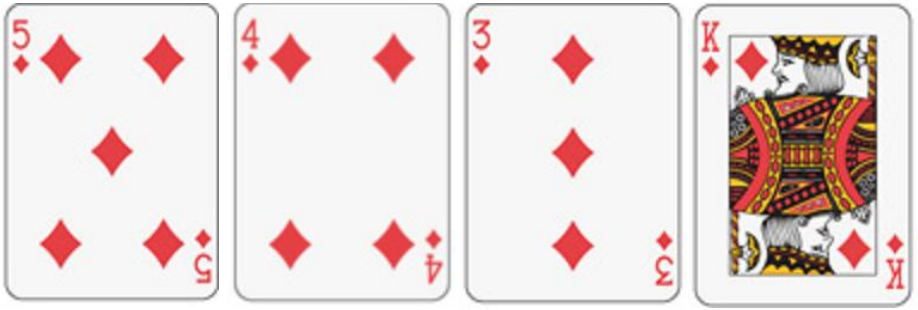
$s_3[20][4]$

a_3

Hit

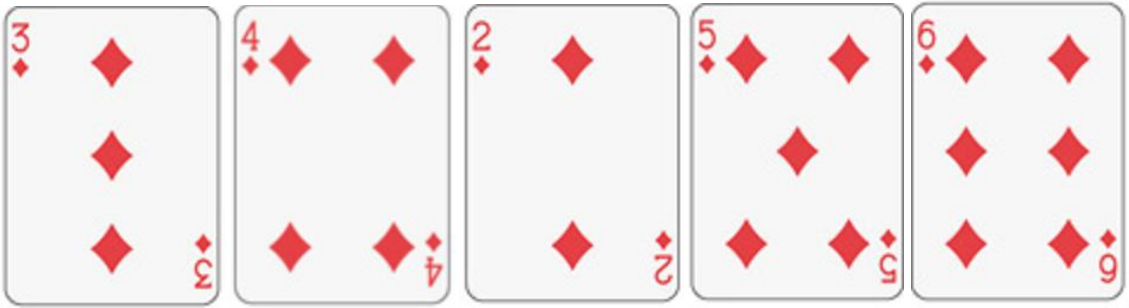
Stand

Dealer Cards



Dealer 22
Player Win

Player Cards



Hit

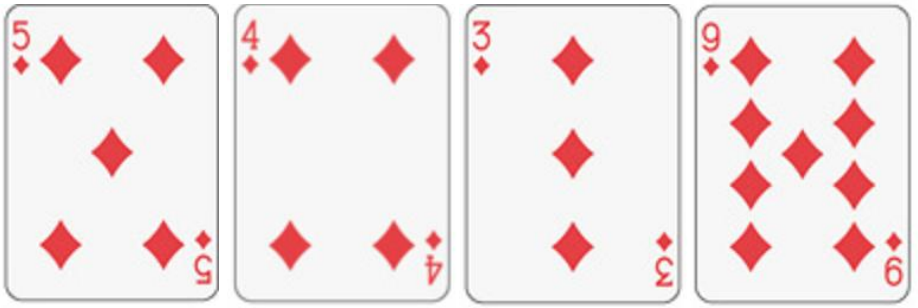
Stand

s_4[win]

$r_4=1$

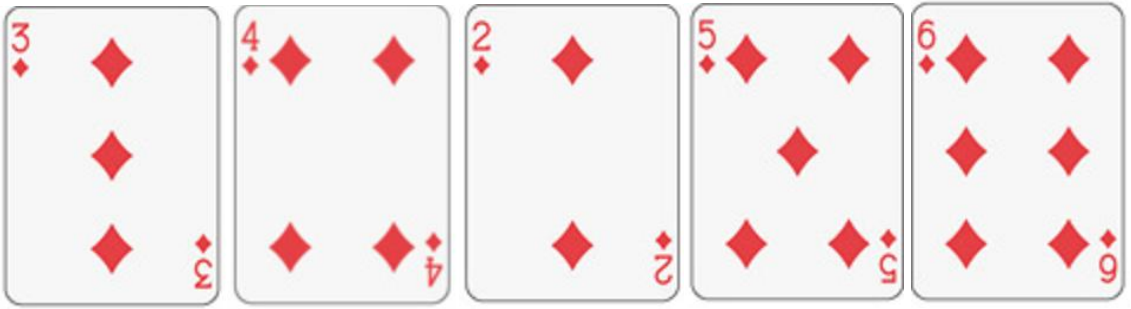
But Maybe…

Dealer Cards



Dealer 21
Player Lose

Player Cards



Hit

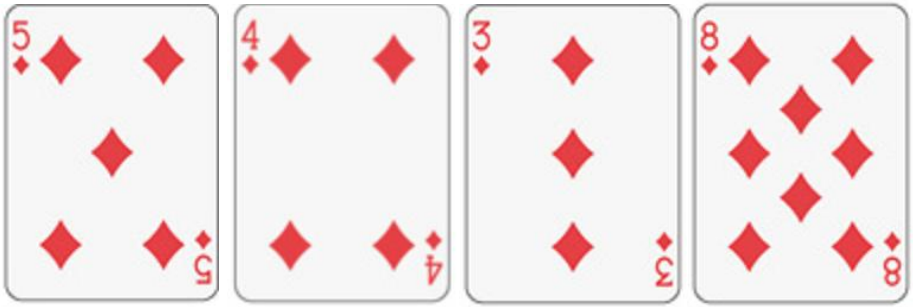
Stand

s_4[lose]

r_4=-1

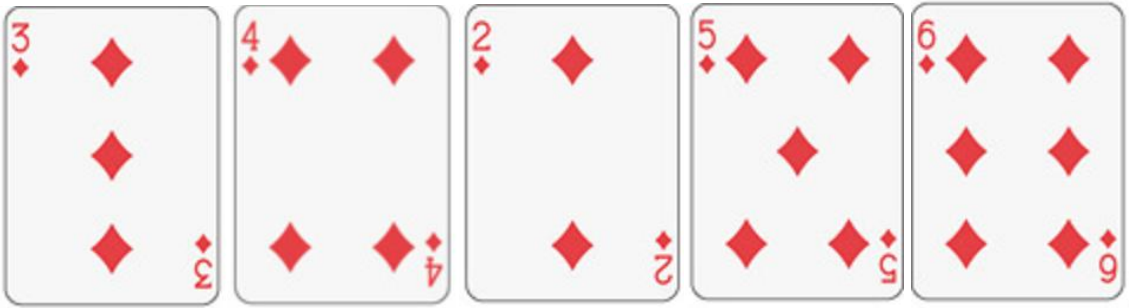
But Maybe…

Dealer Cards



Dealer 20
Push

Player Cards



Hit

Stand

s_4[draw]

$r_4=0$

Ex: Blackjack Hand (Episode)

- Player gets dealt 4, 3 – Dealer showing 4
- S_0 = [7][4] a_0 = hit (Two) r_1=0
- S_1 = [9][4] a_1 = hit (Five) r_2=0
- S_2 = [14][4] a_2 = hit (Six) r_3=0
- S_3 = [20][4] a_3 = stand r_4=1
  - Dealer policy led to a bust this episode
- S_4 = [win]

Ex: Blackjack Hand (Episode)

- State Sequence:
  - seq = [(7,4),(9,4),(14,4),(20,4),(win)]
- Action Sequence:
  - Act = [hit, hit, hit, stand] or [1, 1, 1, 0]
- Reward Sequence:
  - rew = [0, 0, 0, 1]
- Future Return Sum:
  - ret = [1, 1, 1, 1]

Ex: Blackjack Hand (Episode)

- State Sequence:
  - seq = [(7,4),(9,4),(14,4),(20,4),(win)]
- Action Sequence:
  - Act = [hit, hit, hit, stand] or [1, 1, 1, 0]
- Update $q_\pi(s,a)$ value estimates
  - q[7,4][1] = update(r=1)
  - q[9,4][1] = update(r=1)
  - q[14,4][1] = update(r=1)
  - q[20,4][0] = update(r=1)

Ex: Blackjack Hand (Episode)

- State Sequence:
  - seq = [(7,4),(9,4),(14,4),(20,4),(win)]
- Action Sequence:
  - Act = [hit, hit, hit, stand] or [1, 1, 1, 0]
- Update $v_\pi(s)$ value estimates
  - v[7,4] = update(r=1)
  - v[9,4] = update(r=1)
  - v[14,4] = update(r=1)
  - v[20,4] = update(r=1)

Every-Visit MC Prediction – $q_\pi(s,a)$

1. Function **EveryVisitMC**(policy π)
2. q[s][a] = initial values for each state-action pair
3. returns[s][a] = empty list for each state
4. **while**(true):
5. generate episode with π ($s_0,a_0,r_1,s_1,a_1,r_2,\ldots,a_{T-1},r_T$)
6.
7. **for** (t = 0 to T): // for each time step in episode
8.
9. G = sum(r_{t+1} … r_T) // return = sum future rewards
10. append G to returns[s_t][a_t] // keep track of returns from (s_t,a_t)
11. q[s_t][a_t] = average(returns[s_t][a_t])
12.

Monte Carlo Prediction

- We want to estimate $V_\pi(s)$ or $q_\pi(s,a)$ given a set of episodes obtained by following policy π
- Each occurrence of a state s in an episode is called a visit to s, and each state may be visited multiple times in an episode
- First-visit MC methods estimate V as the average of returns following the first visit (studied more)
- Every-visit MCM have different theoretical properties, used for things like function approx

Every-Visit vs First-Visit

- Every-Visit MC may update state value for a single state multiple times if it is visited more than once during an episode
- This is not always desired, since the first visit will contain a ‘deeper’ return calculation than subsequent visits
- Try only updating states on first-visit

Every-Visit MC Prediction – $q_\pi(s,a)$

1. Function **EveryVisitMC**(policy π)
2. q[s][a] = initial values for each state-action pair
3. returns[s][a] = empty list for each state
4. **while**(true):
5. generate episode with π ($s_0,a_0,r_1,s_1,a_1,r_2,\ldots,a_{T-1},r_T$)
6.
7. **for** (t = 0 to T): // for each time step in episode
8.
9. G = sum(R_{t+1} … T_T) // return = sum future rewards
10. append G to returns[s_t][a_t] // keep track of returns from (s_t,a_t)
11. q[s_t][a_t] = average(returns[s_t][a_t])
12.

First-Visit MC Prediction – $q_\pi(s,a)$

1. Function **FirstVisitMC**(policy π)
2. q[s][a] = initial values for each state-action pair
3. returns[s][a] = empty list for each state
4. **while**(true):
5. generate episode with π ($s_0,a_0,r_1,s_1,a_1,r_2,\ldots,a_{T-1},r_T$)
6. visited = {} // record visited states
7. **for** (t = 0 to T): // for each time step in episode
8. **if** ((s_t,a_t) in visited) **continue**; // if s_t has been visited, skip
9. G = sum(R_{t+1} … T_T) // return = sum future rewards
10. append G to returns[s_t][a_t] // keep track of returns from (s_t,a_t)
11. q[s_t][a_t] = average(returns[s_t][a_t]) // sample average update
12. visited.add((s_t,a_t)) // mark (s,a) as visited

Blackjack Using DP?

- It would be very difficult to compute values for Blackjack using DP, even though we have model
- For DP, we are required to know the distribution of next events, rewards, etc
- Ex: s=[13][7], a = stick, what is the probability of terminating with reward +1?
- These probabilities must be known prior to DP
- MC sampling does not need them – just play!

MC Estimation of Action Values

- Now that we have state values, how do we form a policy based on those?
- Without a model, we can't tell which actions lead to other states just using samples
- We must calculate the value of doing an action at a particular state in order to know which action best
  - Recall: Bandit Example (policy based on action values)
- $q_\pi(s,a)$ = expected return when starting in state s, taking action a, and following with policy π

Exploration

- For many problems, the number of states and actions is quite large
- If we follow a given policy, we will do the same thing over and over (not explore)
- We must be careful to explore – choose actions at states that have not been done
- Also explore with states – sample from states that have not yet been visited

Monte Carlo Control

- Now that we have values for state-action pairs, how to update the policy?

- Remember bandit action selection

Monte Carlo Control

- We have seen how to estimate values given that we follow a fixed policy
- How do we now update that policy to perform better over time?
- Hopefully, by learning the values of q(s,a) over time, we can learn a policy that approximates the optimal policy

Policy Iteration

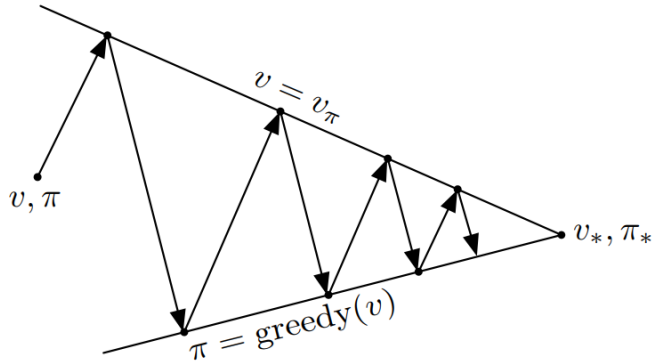
- Once we have a value estimate, we will update our policy incrementally over time
  - Generalized Policy Iteration (GPI)
- In GPI, we maintain estimates of both:
  - Current value function estimates
  - Current policy estimates
- The value function is repeatedly updated to more closely resemble the true value function
- The policy is repeatedly improved based on value

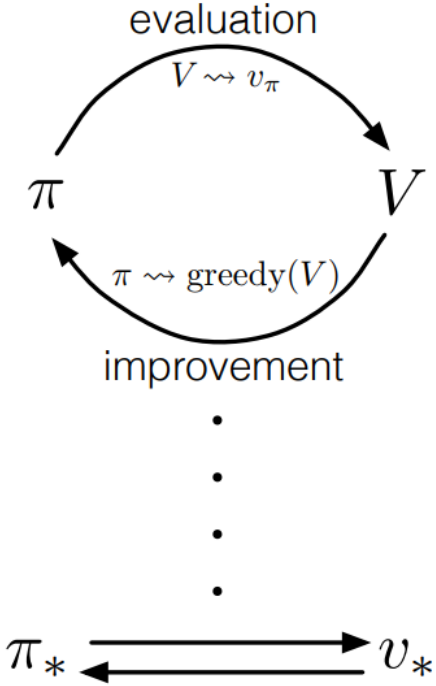
Generalized Policy Iteration

- Policy iteration consists of two processes:
  1. Policy Evaluation: Making the value function more closely estimate the value of the current policy
  2. Policy Iteration: Making the policy greedy with respect to the current value function
- In policy iteration, these two processes alternate, completing before the other begins
- GPI refers to the general idea of letting PE and PI interact to improve a policy over time
- Almost all reinforcement learning methods resemble GPI

Generalized Policy Iteration

- Over time, two things occur:
  - Values approach true values
  - Policy approaches optimal
- When processes stabilize (no changes) the process has completed





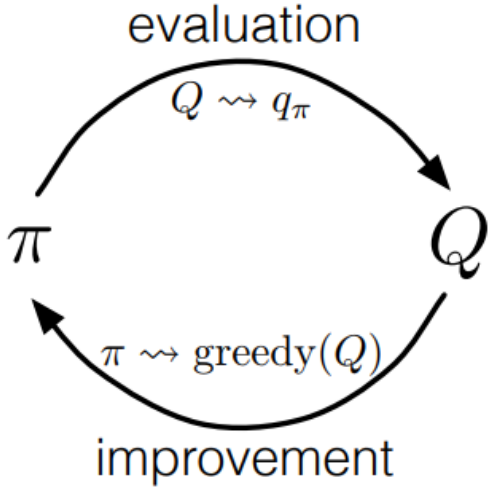
MC Policy Iteration

- How to apply policy iteration with MC methods?
- Policy Evaluation: Update value estimates after an episode has been generated
- Policy Improvement: Update policy based on the new value estimates

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*,$$

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

MC Policy Iteration



1. Function **MCPolicyIteration**()
2. q[s][a] = initial value estimates
3. p[s][a] = initially equiprobable policy
4. **while** true:
5. e = generate episode based on policy
6. q = update value estimates based on episode returns
7. p = update policy to choose actions with max values

q estimates true values, p estimates optimal policy

Blackjack Policy

	Dealer's up card									
Player's hand	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**A**
11 or less	H	H	H	H	H	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
16	S	S	S	S	S	H	H	H	H	H
17-21	S	S	S	S	S	S	S	S	S	S

H Hit

S Stand

Blackjack Hit or Stand Chart

MC – Exploring Starts

- Many problems have large state and action spaces, and in practice, many (s,a) pairs will not be visited often
- When generating episodes, it is important to vary the starting states (if possible) to ensure all states get sampled
- This process: ‘Exploring Starts’ (ES)

Monte Carlo ES

- In MC ES, all returns for (s,a) pairs are simply accumulated and averaged
- MC ES cannot converge to any suboptimal policy
- If it did, the value function would converge to the value for that policy, and the policy would change
- Convergence to the optimal policy is intuitively inevitable as changes in the action-value function decrease over time, but has not yet been proven

MC Exploration in Practice

- In practice, MC with ES converges slowly because not enough diff states visited
- We can also explore while generating the episodes using Epsilon Greedy or UCB
- By including these randomized actions, more states get visited, more actions get sampled, and we learn more quickly

Monte Carlo Overview

- Monte Carlo methods use sampling to generate individual episodes with (states, actions, rewards)
- Value function estimate is updated at the end of each episode via rewards that were obtained
- Policy updated to reflect new estimates (GPI)
- New episode generated with updated policy
- Varying starting states (ES) and action selection can help visit more states & converge to optimal policy

Matchbox Machine Learning

- **MENACE: the pile of matchboxes which can learn**

https://youtu.be/R9c-_neaxeU

- This video content CAN BE ON THE EXAM

Exam Questions

- Define Monte Carlo Methods
- MC Algorithms
- V(s) vs Q(s,a) estimation
- Given samples, perform MC calculation
- Explain exploring starts
- GPI Definition
- MENACE video content