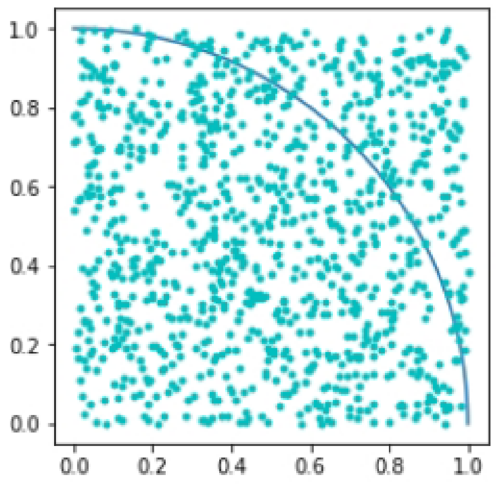
C4: Gaming with Monte Carlo Methods

▼ Table of Contents

Monte Carlo Methods

- MC Methods find approximate solutions through random sampling through approximation of the probability of an outcome after running multiple trials.
- Powerful in finding optimal policy without the need to know the model dynamics (i.e. transition and reward probabilities)
- Fun fact: Monte Carlo is named after Stanislaw Ulam's uncle who often borrowed money for gambling in Monte Carlo casino. And the first MC machine running in computer is through the help of John Von Neuman.

Estimating value of pi using Monte Carlo



$$\frac{Area\ of\ circle}{Area\ of\ square} = \frac{\#Points\ in\ circle}{\#Points\ in\ square}$$
$$\frac{\pi r^2}{a^2} = \frac{\#Points\ in\ circle}{\#Points\ in\ square}$$
$$\frac{\pi(\frac{1}{2})^2}{1^2} = \frac{\#Points\ in\ circle}{\#Points\ in\ square}$$
$$\pi = 4 * \frac{\#Points\ in\ circle}{\#Points\ in\ square}$$

To estimate π:

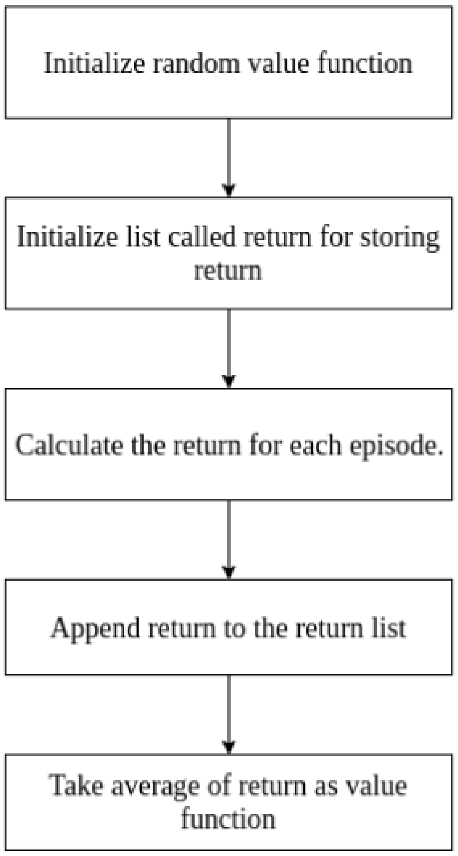
1. Generate random points inside square
2. Calculate number of points that is inside the circle using formula $x^2 + y^2 \leq size$
3. Calculate π using formula above
4. Increase number of samples for better estimation

Monte Carlo Prediction

- Previously, in dynamic programming, we solved Markov Decision Process (MDP) (i.e. finding optimal policy and value function) using Value Iteration (i.e. Deriving best policy based on estimated value function) and Policy Iteration (i.e. evaluating and improving a policy using value function). However, that is given the model dynamics (e.g. transition and reward probabilities) is known.

- MC methods only requires sample sequences of states, actions and rewards for only episodic tasks.
- MC methods does not need any model, hence, it is model-free learning algorithm
- Instead of calculating the expected return, in MC, mean return is calculated (since dynamics is unknown)
- MC prediction can be used to estimate value function of any given policy using following steps:

1. Initialize a random value to value function
2. Initialize an empty return list that stores our returns of every action in every episode
3. For each state in the episode we calculate return as the cumulative rewards
4. Return is appended to the return list
5. Take average of return as our value function


Initialize random value function
Initialize list called return for storing return
Calculate the return for each episode.
Append return to the return list
Take average of return as value function

First Visit Monte Carlo Prediction

- In first visit MC prediction, only the first observed state in a single episode is record and averaged for the calculation of mean return for the value table.

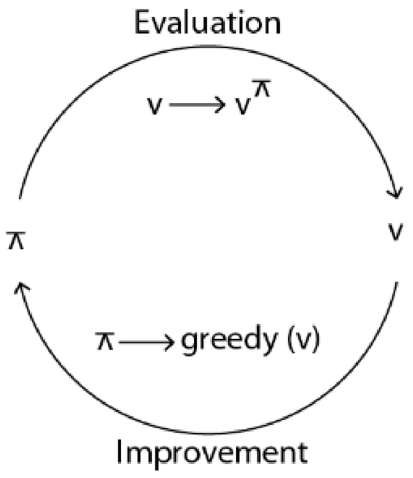
- For example, consider an agent is playing the snakes and ladder games,
  there is a good chance the agent will return to the state if it is bitten by a snake.
  When the agent revisits the state, we don't consider an average return.
  We consider an average return only when the agent visits the state for the first time.

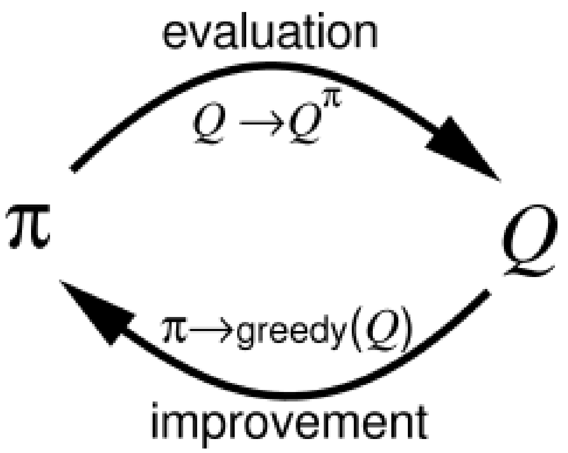
Every Visit Monte Carlo Prediction

- In every visit MC prediction, all the observed state in a single episode is recorded and averaged for the calculation of mean return for the value table.
- Consider the same snakes and ladders game example:
  if the agent returns to the same state after a snake bites it,
  we can think of this as an average return although the agent is revisiting the state.
  In this case, we average return every time the agents visit the state

Monte Carlo Control

- In MC Control, the goal is to solve MDP through optimizing the value function as well as the policy function, that is the main parameter to decide the values in a value function.
  - We can even consider a value function as a quantification of how good a single policy function is.
  - In Dynamic Programming, estimating a state value alone is sufficient to giving optimal policy as the model dynamics is known. (Recall the Bellman Equation for Value Function C3: Markov Decision Process (MDP) & Dynamic Programming)
  - However, in MC Control, estimating an action value makes more sense as intuitively, a state values depends heavily on the policy that we choose
    - ▼ E.g.
      - In Blackjack, the state given is player_sum = 20
        If policy is to hit, then the state is a bad state
        But, if policy is to stand, then the state is a good state.
  - To estimate a action value, we need to use the Bellman Equation for Q-Function to find $Q(s, a)$
- This is where the G**eneralized Policy Iteration (GPI)**, is introduced were policy evaluation(i.e. approximation of the value function) and policy improvement(choosing the best policy given a state) interact with each other.





- Under MC Control, there are several algorithms available to accomplish the generalized policy iteration as listed in following subsections.

Monte Carlo Exploration Start Control (MC-ES)

- MC-ES solve the Exploration vs Exploitation dilemma through randomly initialising Q function and policy and start the episode with randomly initialized policy so as to accomplish the Exploration Start.
- However, the cons of MC-ES is large number of episode is needed as the exploration only occurs at the first action of any episode.

Exploration vs Exploitation Dilemma

- In RL, we needs our agent not only to be able to receive suboptimal positive rewards (through greedy approach), but to receive the best reward possible given the current state.
- To accomplish this, the agent needs to make a trade-off between exploring the environment (so as to find the optimal policy) and also making profit and earning rewards to what the agent is doing (so as to improve the policy).

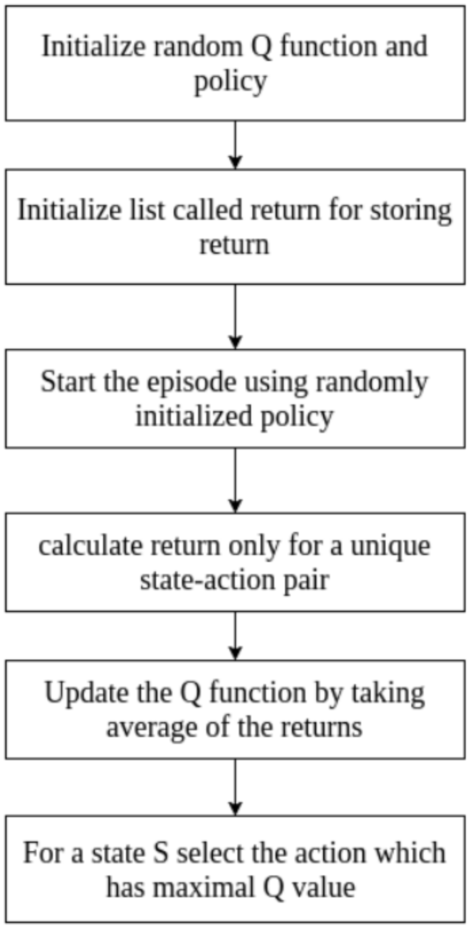
MC-ES Algorithm

To be more specifics, the following are steps on how MC-ES works:

1. Initialize Q function and policy with random values and prepare a empty return list
2. Start episode with randomly initialized policy
3. Calculate return for unique state-action pairs in an episode and append to our return list
    a. Unique return is calculated as same action pair occur in an episode multiple times is redundant information for us
4. Average the returns in return list and assign the value to Q function
5. Finally, we select an optimal policy for a state, choosing action that has the maximum $Q(s, a)$ for that state.

$$\pi^*(s) = argmax\ Q(s, a)$$

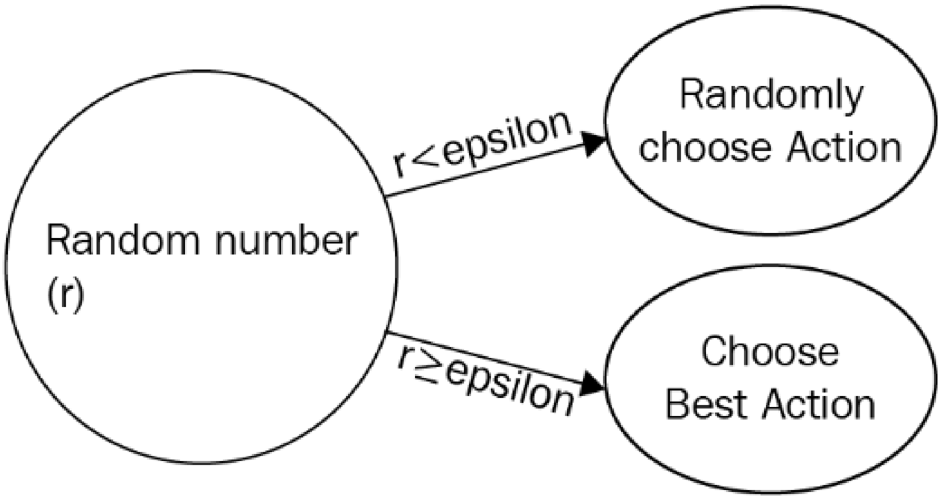
6. Repeat the whole process forever or for large number of episode to cover all different states and action pairs


Initialize random Q function and policy
Initialize list called return for storing return
Start the episode using randomly initialized policy
calculate return only for a unique state-action pair
Update the Q function by taking average of the returns
For a state S select the action which has maximal Q value

On-Policy Monte Carlo Control (ϵ-greedy policy)

- To overcome the long exploration time of MC-ES, in On-policy MC control, all actions are tried with a non-zero probability (epsilon) that represents choosing Exploration over Exploitation for an action which the probability will decay overtime.

Epsilon-Greedy Strategy



```
def epsilon_greedy_policy(state, epsilon):
    if random.uniform(0,1) < epsilon:
        return env.action_space.sample() # Exploration
    else:
        return max(list(range(env.action_space.n)), key = lambda x: q[(state,x)]) # Exploitation
```

Epsilon Greedy Algorithm

http://incompleteideas.net/book/ebook/node53.html

To be more specifics, the following are steps on how Epsilon Greedy MC Control works:

1. Initialize Q function and policy with random values and prepare a empty return list
2. Generate an episode using the random policy π
3. Store return of every state action pair occurring in the episode to return list
4. Then, we take an average of the returns in the return list and assign that value to the Q function
5. Now the probability of selection action a in state s will be decided by epsilon
6. If the probability is 1-epsilon, we pick action which has maximal Q-value
7. If probability is epsilon, we explore for different actions

Off-Policy Monte Carlo Control (Importance Sampling)

- Off-policy MC Control make use of two independent policy function π and b where b is used for “Exploration Purpose” and the good action-values pair will be reflected to the targetted policy function π using Importance Sampling.

> Still pending more research and understanding of Off-Policy MC Control

Blackjack with MC

Blackjack Environment

Actions

- Hit : Add new card (1)
- Stand : Don't add new card (0)

Rewards

- Draw : No one wins or lose (0)
- Win : Player sum more than dealer sum (1)
- Lose : Player sum more than 21 or dealer has higher sum (-1)

Policy & Episode

- For simplicity sake, we use a naive policy which is to Stand (Don't add new card) when the player sum $<$ 20 or else Hit (Add new card).
- For every episode, we will continue use the policy above until the terminal state is reached:
  - Player have won (i.e. Player Sum > Dealer Sum or Dealer Sum > 21)
  - Dealer have won (i.e. Player Sum < Dealer Sum or Player Sum > 21)
- We also records the *First Visit States*, *Action taken* and the *Immediate Rewards after each action* for each episode return the list at the end of the episode

```
def generate_episode(policy, env):
    # we initialize the list for storing states, actions, and rewards
    states, actions, rewards = [], [], []

    # Initialize the gym environment
    # observation : (player_sum, dealer_first_card, usable_ace)
    observation = env.reset()

    while True:
        # append the states to the states list
        states.append(observation)

        # now, we select an action using our sample_policy function
        # and append the action to actions list

        action = sample_policy(observation)
        actions.append(action)

        # We perform the action in the environment according to our
        # sample_policy, move to the next state
        # and receive reward
        observation, reward, done, info = env.step(action)
        rewards.append(reward)

        # Break if the state is a terminal state
        if done:
             break

    return states, actions, rewards
```

First Visit MC Prediction

- The goal of MC Prediction is to calculate the mean return for each state so as to estimate the value table for all the states available
- To accomplish the goal, we need to simulate high number of episode (for better estimations) and records down the following attributes across all attributes:
    - Value Table: (`dict`) with format { state_observation : mean_return_sum }
    - Number of Visit: (`dict`) with format { state_observation: count_of_observation }
- In each episode. the calculation of mean returns for each state_observation is done with following characteristics:
    - We start looping at each states on Reverse chronology order. and in each iteration, we accumulate all the immediate rewards into a `return` variable.
        - The reasoning for such reverse iteration is to associate the return of each action as the sum of immediate return and all future returns.

- ▼ E.g.

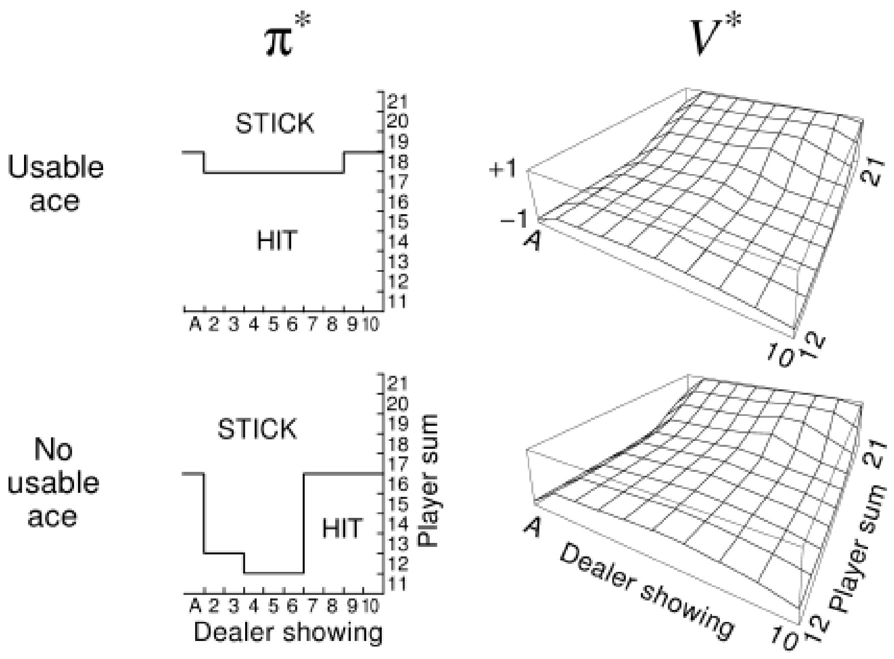
  (In Chronology Order)
  State Format : (Player Sum, Dealer 1st Card, Usable Ace)

  1st Step: (8, 7, 0) -> Action Hit (1) -> Reward 0
  2nd Step: (20, 7, 0) -> Action Stand (0) -> Reward 1

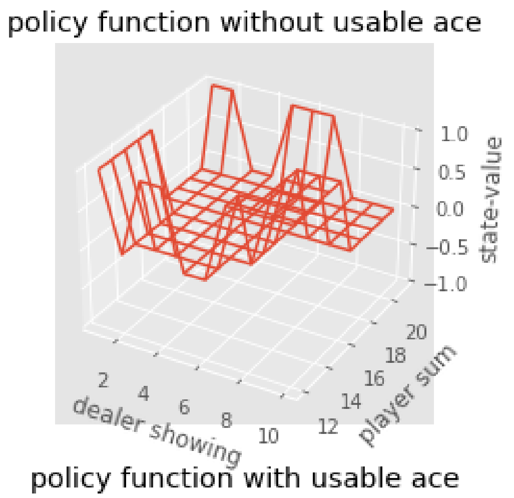
  Hence, we can say that the reward of the 1st step should be (0+1 = 1) as the decision finally lead to a victory.

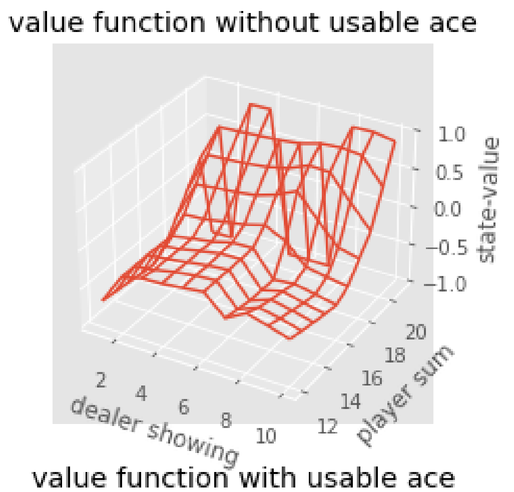
- Then, we determine whether is the current state present in any of the previous observations, so as to achieve First Visit MC Prediction
  - However, in Blackjack, there aren't anyways for a state to repeat twice in an episode as the only way to retain a same state is through the action of "Stand". And after "Stand" is called, the game will automatically return to a terminal state and thus, determining the winner.

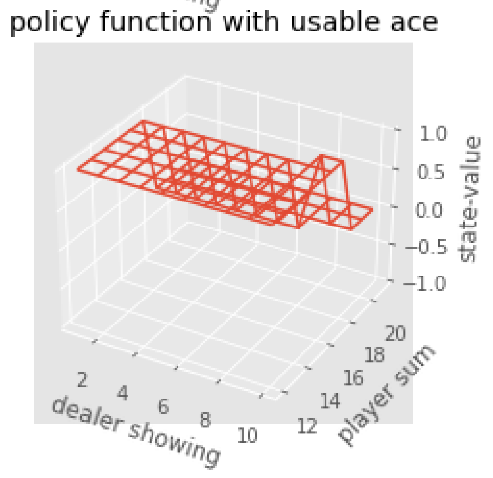
MC-ES Control

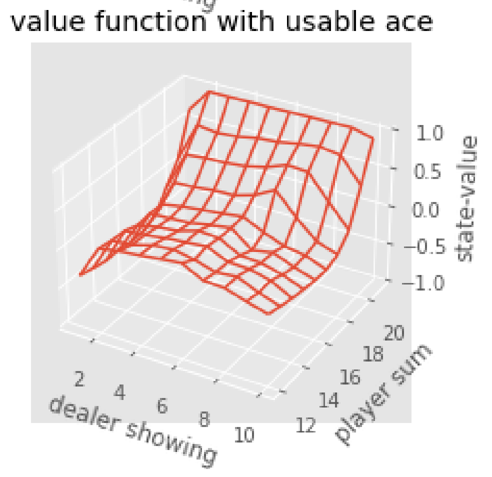


m









Refer to MC-ES Algorithm for detailed explanation of the Blackjack Codes