IIIT Hyderabad **Aditi Gangrade**
Topics in Machine Learning (CSE975) **Shreya Singh**
Instructor: Dr. Naresh Manwani **Lecture #16, Oct 6, 2017**

Outline. *Discussing* ***Monte-Carlo Methods****,* ***First-Visit Monte-Carlo*** *and* ***Every-Visit Monte-Carlo****.* ***Exploring Starts*** *algorithm for Monte-Carlo control.*

1 Monte Carlo Methods

One of the major requirements of DP is complete knowledge of the environment. Since the state value of a state depends upon the values of other states, as given by the following equation,

$$v_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s)[r(s,a) + \gamma \sum_{s' \in \mathcal{S}} v_\pi(s') \sum_r p(r, s'|s, a)]$$

it can clearly be seen that we need a complete probabilistic model of the system for solving the model using DP. But in real scenarios, there exists cases where it is difficult to build a model of the system beforehand, i.e. the model is unknown. In such cases, we cannot use DP. The solution to such a problem is applying Monte Carlo Methods.

Instead of depending upon the model of the system, Monte Carlo Methods use actual experience for learning. They use sample sequences of states, actions and rewards for estimating value functions and discovering optimal policies. MC methods learn from experiences in the form of episodes. An episode can be thought of as a single sequence of states, actions and rewards, which ends in a terminal state. For example, playing an entire game can be considered as one episode, the terminal state being reached when one player loses/wins/draws.

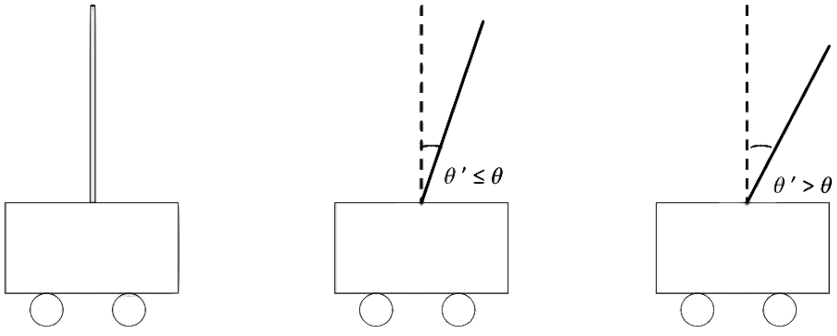

Figure 1: Sample episode of cart-pole balancing game

In the cart-pole balancing game, an episode starts with pole standing upright. As the cart moves, the pole tilts a bit. In this finite-time horizon game example, the episode ends when the pole falls below a certain angle θ and the game is said to be lost.

2 Monte Carlo Policy Evaluation

Given a policy π, let the value of a state s, be denoted by $v_\pi(s)$. Then, the average of the returns observed after passing the state s over a set of episodes gives the value $v_\pi(s)$. The main idea underlying Monte Carlo Methods is that this average converges to the expected value as sufficiently many episodes are observed, thus, giving us the required value.

2.1 The Naive Approach

- For each $s \in \mathcal{S}$, run π from s for m times, where the i^{th} run (episode) is L_i.
- Let r_i be the return of L_i.
- Estimate the value of the policy π starting from s, by

$$\hat{v}_\pi(s) = \frac{1}{m}\sum_{i=1}^{m} r_i$$

Note that, the variables r_i, in the above formula, are independent since the runs L_i are independent.

Assuming r_i's are bounded $\forall$i, we can apply ***Chernoff's Theorem***, which states that

$$Pr\left[|\hat{v}_\pi(s) - v_\pi(s)| \geq \lambda\right] \leq 2e^{-2m\lambda^2}$$

Let, the given probability be upper bounded by δ.

$$\begin{aligned}
\delta &\geq 2e^{-2m\lambda^2} \\
\delta/2 &\geq e^{-2m\lambda^2} \\
\ln(\delta/2) &\geq -2m\lambda^2 \\
\ln(2/\delta) &\leq 2m\lambda^2 \\
m &\geq (1/2\lambda^2)\ln(2/\delta)
\end{aligned}$$

Thus, given the values of λ and δ, we can obtain the total number of episodes required.

2.2 Categorization of Monte Carlo Approaches

Visit : Each occurrence of the given state s in an episode is called a visit to s.
First Visit : First occurrence of the state s in an episode is called the first visit to s.

Based on the visits made to a state s, the MC approaches are classified as **First-Visit MC** and **Every-Visit MC**. While the former estimates $v_\pi(s)$ as the average of the returns following first visits to s, the latter averages the returns following all visits to s. Both these approaches converge asymptotically.

2.2.1 First Visit Monte Carlo Policy Evaluation

- Run π from s for m times, where the i^{th} run(episode) is L_i
- For each run L_i and state s in it, let $r(s, L_i)$ be the return of π in run L_i from the first appearance of s in L_i until the run ends (reaching terminal state).
- Thus the value of state s under policy π be estimated as:

$$\hat{v}_\pi(s) = \frac{1}{m} \sum_{i=1}^{m} r(s, L_i)$$

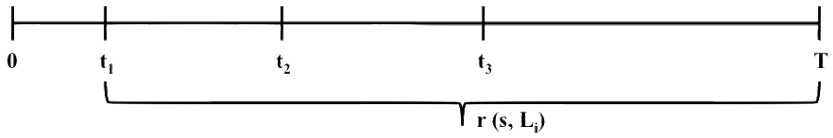

Figure 2: Return in first visit MC

where,

t_1, t_2, t_3 represent the time instances of visits of state s and,
$r(s, L_i)$ represents the return obtained after first visit of s.

Note that, the random variable $r(s, L_i)$, in the above formula, for a given state s and different L_i's are independent since different runs (episodes) are independent.

2.2.2 Every Visit Monte Carlo Policy Evaluation

- Run π from s for m times, where the i^{th} run(episode) is L_i
- For each run L_i and state s in it, let $r(s, L_i, j)$ be the return of π in run L_i from the j^{th} appearance of s in L_i.
- Let N_i(s) be the number of times state s has been visited in the run L_i.
- Let the value of state s under policy π be :

$$\hat{v}_\pi(s) = \frac{1}{\sum_{i=1}^{m} N_i(s)} \sum_{i=1}^{m} \sum_{j=1}^{N_i(s)} r(s, L_i, j)$$

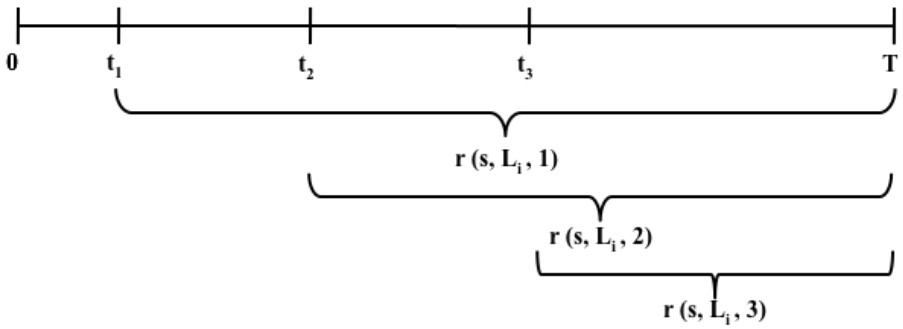

Figure 3: Returns in every visit MC

where,

t_1, t_2, t_3 represent the time instances of visits of state s and,
$r(s, L_i, j)$ represents the return obtained after j^{th} visit of s.

Note that, here the random variable $r(s, L_i, j)$ for a given state s and L_i are dependent for different j.

3 Example :

Let us try to understand the concept of Monte Carlo methods using an example. The example we consider here is the famous casino card game, Blackjack.

3.1 Rules

To understand the rules, let us consider a scenario where a player plays independently against the dealer. Both the dealer and the player are dealt two cards, with one card of the dealer facing up. The ultimate goal of the player is to get a total sum greater than that of the dealer, without exceeding 21. Each card is assigned a value as follows -

- Numbered cards : their corresponding rank (number)
- Face cards (jack, queen, king) : 10
- Ace : 1 or 11, depending upon the player's need

The episode ends in one of the following terminal conditions -

- Win : (sum of dealer's cards $<$ sum of player's cards ≤ 21) or (sum of dealer's cards > 21)
- Lose : (sum of player's cards $<$ sum of dealer's cards) or (sum of player's cards > 21)
- Draw : sum of player's cards $=$ sum of dealer's cards

A player is said to go *bust* if the sum exceeds 21 and loses the game immediately. If the sum of player's cards exactly equals 21, its is called a *natural*.
A player can perform the following actions -

- Hit : request additional cards if the sum < 21
- Stick : stop receiving cards and transfer the turn to the dealer

The dealer follows a fixed approach -

- Stick : stop receiving cards if the sum ≥ 17
- Hit : request additional cards otherwise

3.2 Blackjack as Episodic Finite MDP

Formulating Blackjack as an episodic finite time horizon MDP $M =< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$, we have,

- **Goal:** Obtain card sum greater than dealer without exceeding 21.
- **States** ($\mathcal{S}$) **:** Total 200 states where each state can be represented as a combination of three variables :
  - player's current sum (Note that here the sum will always lie in the range 12-21. If the sum is ≤ 11 then, he will follow a deterministic approach as he can hit without going bust.)
  - the dealer's one showing card which can be anything among {ace, 2, 3, 4, 5, 6, 7, 8, 9, 10}. (Since the face cards have a value 10, so they are also included in the given set.)
  - whether or not the player holds a usable ace. (An ace can take value either 1 or 11.)
- **Actions** ($\mathcal{A}$)**:**
  - Stick
  - Hit
- **Reward** ($\mathcal{R}$) **:**
  - +1 for winning.
  - 0 for draw.
  - -1 for losing.
- **Discount Factor** (γ) **:**
  γ here is assumed to be 1, as the rewards are not discounted and hence, the terminal rewards are also the returns.

3.3 Sample Observations

Consider the policy that sticks if the player's sum is 20 or 21, and hits otherwise. Simulating the Blackjack game following the approach defined above, we obtain the state values by playing multiple episodes. A noteworthy fact is that in this task, the same state never recurs within one episode, so there is no difference between first-visit and every-visit MC methods. The results of the experiment are shown in Figure 4. The estimates for states with a usable ace are less certain

and less regular because these states are less common. In any event, after 500,000 games the value function is very well approximated.

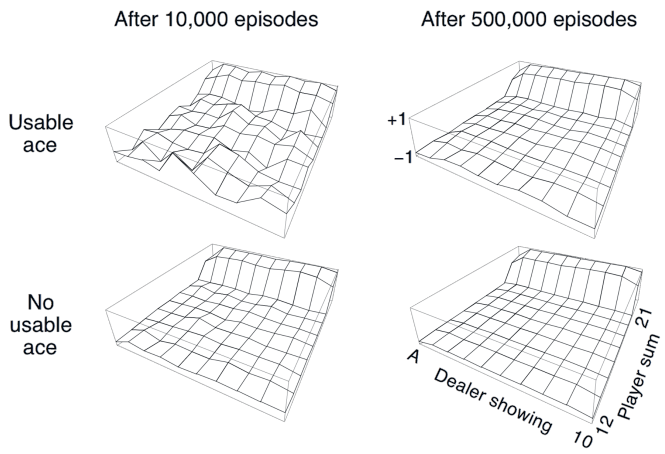

Figure 4: Approximate state-value functions for the blackjack policy that sticks only on 20 or 21, computed by Monte Carlo policy evaluation.

Could DP have been used as an alternative approach for the given problem?
Though we have complete knowledge of the environment in this task, it would still not be easy to apply DP. DP methods require the distribution of next events-in particular, they require the probabilities $p(s', r|s, a)$ and it is not easy to determine these for blackjack. For example, suppose the player?s sum is 14 and he chooses to stick. The expected rewards and transition probabilities must be computed before DP can be applied to calculate the expected reward of a given state, and such computations are often complex and error-prone.

3.4 Dynamic programming vs Monte Carlo

- **Dynamic programming (DP):**
  - Requires full knowledge of environment
    (Pr $\{R_{t+1} = r,\ S_{t+1} = s'|S_t = s,\ A_t = a\}$, $\forall s, s' \in \mathcal{S}$, $\forall a \in \mathcal{A}(s)$, $\forall r$)
  - Must have model built beforehand.
  - Complex, error-prone.
- **Monte Carlo :**
  - Generating sample games easy.
  - MC methods can be better, even when complete knowledge of environment?s dynamics is known.
  - State value estimates for each state are independent.
  - The estimate for one state does not build upon the estimate of any other state.
  - Monte Carlo methods do not bootstrap.

– Computational expense of estimating the value of a single state is independent of the number of states.

– One can generate the episodes starting from the state of interest, averaging the returns from only these states ignoring others.

3.5 Monte Carlo Estimation of Action Values

In particular scenarios, where the model is not available, a policy π cannot be chosen based upon state values. It therefore becomes necessary to estimate the value of each action so as to evaluate and ultimately, determine an efficient policy. Thus, in MC methods, we aim towards finding the value of q_*.

Policy evaluation problem for action values : To be able to evaluate a policy, we must have a good estimate of $q_\pi(s, a)$ for all state-action pairs. To guarantee that each state-action pair is visited, we start episodes with each state-action pair, i.e. assign a non-zero probability of being the start of episode to all state-actions. This assumption is known as ***Exploration Starts***. Note that if all state-action pairs are not explored, we would not have sufficient returns to evaluate a policy and with no returns to average, MC estimates of the other actions will not improve with experience.

3.6 Monte Carlo Control

Similar to the policy iteration method followed in DP, Generalized Policy Iteration estimates value function and policy alternately as shown in Figure 5. The value function is made to approximate the actual values for a given policy (policy evaluation), and based upon the values obtained, one tries to estimate a new improved policy (policy improvement).

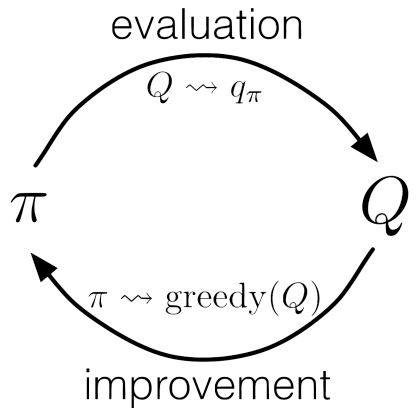

Figure 5: Generalized policy iteration

Let the iteration begin with an initial estimate of π_0. With alternately evaluating and estimating a policy as described above, one can find the optimal value of q_*.

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} ... \xrightarrow{I} \pi_* \xrightarrow{E} q_*$$

where $\xrightarrow{E}$ denotes a complete policy evaluation and $\xrightarrow{I}$ denotes a complete policy improvement.

3.6.1 Assumptions

- An infinite number of episodes are observed.
- Episodes are generated following the assumption of *Exploring Starts*.

3.6.2 Policy improvement step

Policy improvement is done by making the policy greedy with respect to the current value function.

$$\pi_{k+1}(s) = arg\max_{a}(q_{\pi_k}(s,a))$$

3.6.3 Policy improvement theorem

$$\begin{aligned} q_{\pi_k}(s,\pi_{k+1}(s)) &= q_{\pi_k}(s, arg\max_{a}(q_{\pi_k}(s,a)) \\ &= \max_{a\epsilon\mathcal{A}}(q_{\pi_k}(s,a)) \\ &\geq q_{\pi_k}(s,\pi_k(s)) \\ &\geq v_{\pi_k}(s) \end{aligned}$$

This shows that π_{k+1} is, in worst case, as good as π_k, thus, ensuring the convergence to optimal policy and optimal value function.

3.7 Monte Carlo Exploring Starts

Algorithm 1 Monte Carlo Exploring Starts

Initialize: for all $s \epsilon \mathcal{S}$, $a \epsilon \mathcal{A}(s)$:
  $q(s,a)$ arbitrary
  $\pi(s)$ arbitrary
  Returns(s,a) empty list
Repeat forever:
  Choose $s_0 \epsilon \mathcal{S}$ and $a_0 \epsilon \mathcal{A}(s_0)$ s.t. all pairs have probability > 0
  Generate an episode starting from (s_0, a_0) following π
  For each pair (s,a) appearing in the episode:
    G return following the first occurrence of (s,a)
    Append G to $Returns(s,a)$
    $q(s,a)$ = average($Returns(s,a)$)
  For each s in the episode:
    $\pi(s) = \arg\max_a\ q(s,a)$

In the above algorithm, returns are averaged irrespective of the policy followed. For performing policy improvement, we need to know if $q_\pi(s,a) > v_\pi(s)\ \exists(s,a)$. While in *Exploring Starts*, we do not calculate q based upon the policy π. Though *Exploring Starts* ensures one time selection of a

state-value pair but, to make certain that actions are selected infinitely many times (and not just at the start of episode), we start with a soft policy s.t. $\pi(s, a) > 0 \ \forall (s, a)$. Thus, we come up with new approaches namely, *on-policy* and *off-policy* methods, which will be discussed in subsequent lectures.

References

[1] Sutton, Richard S and Andrew G. Barto *Reinforcement learning: An introduction.* Vol.1. No.1. Cambridge: MIT press, 1998.