Monte Carlo Methods

Reading

- Sutton, Richard S., and Barto, Andrew G. Reinforcement learning: An introduction. MIT press, 2018.
  - http://www.incompleteideas.net/book/the-book-2nd.html
  - Chapter 5
- David Silver lecture on Dynamic Programming
  - https://www.youtube.com/watch?v=PnHCvfgC_ZA&t=585s
  - First part of the lecture (on Monte-Carlo learning)

Overview

Rensselaer

- Monte Carlo methods are very effective for estimating distribution parameters through sampling
  - Means, variances, etc.
  - Strong convergence theory, due to the law of large numbers
- Suppose we have access to a number of episodes of length T
  - "Training data"
  - We don't have access to the full MDP anymore
- Estimate value functions averaging over multiple episodes
  - If we can sample a $v_{\pi,i}(s)$ for a given policy π and for each episode i, then the $v_{\pi,i}(s)$ are IID
  - By law of large numbers, the average of the $v_{\pi,i}(s)$ will converge to the true $v_{\pi}(s)$ as $i \to \infty$

Policy Evaluation

Rensselaer

- Suppose we are given N runs of an MDP, each of length T
  - Each run has the form $S_0, A_0, R_1, S_1, A_1, \dots, R_T, S_T$
  - Also works for an MRP
  - For now, assume we used the same policy π in all
- Suppose we wish to estimate the value of a given state, $v_\pi(s)$
  - How do we do that?
  - High-level idea: every time we visit state s, compute the discounted return from then on
    - Then average all returns
  - What issues can you spot?

Policy Evaluation, cont'd

- High-level idea: every time we visit state s, compute the discounted return from then on
- We might visit s at different times in each trace
- Furthermore, might visit s multiple times per trace
  - Breaks IID assumption (Law of Large numbers doesn't apply)
- How do we proceed?
  - Problem is very hard for time-dependent MDPs
    - Ideally, we try to estimate $v_\pi^t(s)$ for each t
    - Would require multiple examples visiting s for each t
  - The book focuses purely on time-independent MDPs
    - E.g., MDPs with terminal states such as games
    - Takes of care of the time-dependency challenge
      - What about the multiple visits problem?

MDPs with a Terminal State

- MDP assumption:
  - MDP is assumed to have a terminal state
  - Each available trace must reach the terminal state
- Why is this convenient?
  - The infinite-horizon and finite-horizon are now the same
  - Terminal node can be assumed to have a self-transition w.p. 1 and reward of 0
  - An infinite trace that reaches terminal node at T looks like $S_0, A_0, R_1, S_1, A_1, \ldots, S_T, A_T, R_{T+1}, S_{T+1}, A_{T+1}, 0, \ldots$
    - Where $S_t = S_T$ for all $t > T$
  - Now, $v_\pi(s)$ is time-independent
  - Monte Carlo method now gives us a sample of $v_\pi(s)$

First-visit Method

- The first-visit method works by only calculating the average return after the first time we visit state s in an episode
  - Future visits in the same episode are ignored

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

      Append G to $Returns(S_t)$

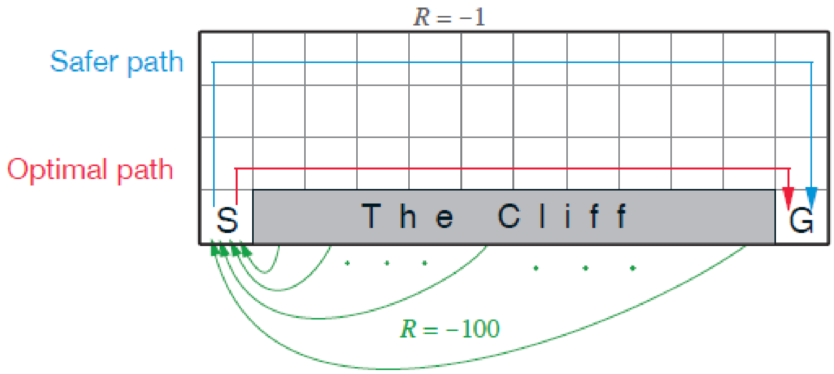
      $V(S_t) \leftarrow \text{average}(Returns(S_t))$

First-Visit Method Issue

- May not visit each state enough times for good convergence
  - Either due to the policy or insufficient number of traces
- What's one way around it?
  - Average over all visits!
  - Samples no longer IID
    - If two visits occurred in the same trace, not independent
- Turns out the every-visit method is biased!
  - i.e., the expectation is not exactly $v_\pi(s)$
  - But it is consistent!
    - i.e., the bias converges to 0 with more data
  - Overall, trade-off between more samples and bias

Comparison between first-visit and every-visit

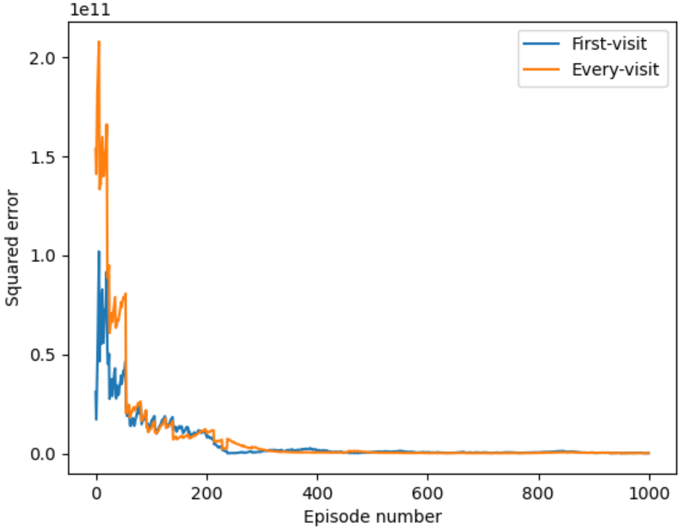
- Consider the following environment



- Goal is to reach G from S
- Actions are up, down, left, right
- Reward of -1 after each step
- Reward of -100 if you fall off The Cliff
- Goal is a sink state (so no more negative reward at that point)

Comparison between first-visit and every-visit, cont'd

- Estimate values for a random policy
- Difficult for MC methods since rewards vary a lot
  - May take a long time to get to goal
- Both methods converge slowly
  - Every-visit method has larger error initially
  - Due to bias



Monte Carlo Control

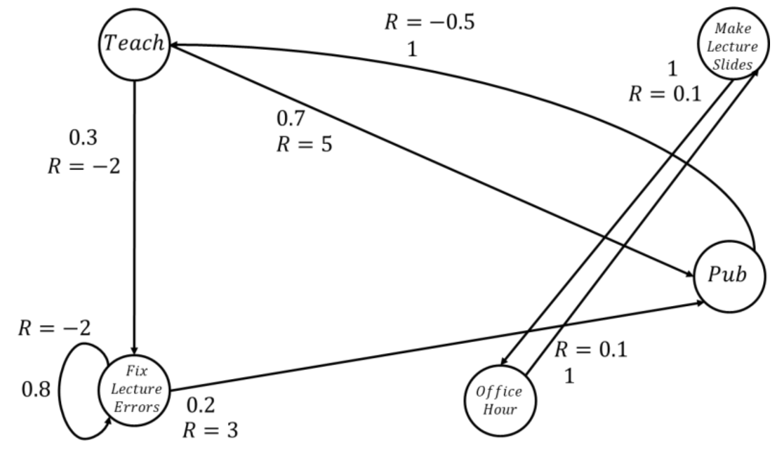
- So far, we've seen how to estimate state values for a given policy π
- Now, we'd like to find a better policy (i.e., learn)
- It is tempting to use the value iteration recursion, since we already have the values

$$v_{k+1}(s) = \max_a \left[\mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a]\right]$$

  - What's wrong with that?
- We don't have an estimate for the expected reward
- Need to compute expected rewards for all actions
  - I.e., q values
  - What is a potential challenge with this?

Monte Carlo Control, cont'd

- Recall a Workday Example policy π we had:
  - $\pi(Teach) = Relax$
  - $\pi(OH) = Work$
  - $\pi(MLS) = Work$
  - $\pi(FLE) = Relax$
  - $\pi(Pub) = Work$



- If I collect traces with π, what q values won't I see?

  $q_\pi(OH, Relax), q_\pi(Pub, Relax)$, etc.

- Will not see all q values in general, especially with a deterministic policy
  - What can we do?

ϵ-greedy policies

- Will not see all q values in general, especially with a deterministic policy
- This is part of the RL exploration vs exploitation challenge
  - Need to explore more state/action pairs while maximizing intermediate rewards
- One way around this is to make π probabilistic
  - For each s, add an ϵ probability that you don't select $\pi(s)$
  - i.e., define π_ϵ s.t.
    - $\pi_\epsilon(s) = \pi(s)$ w.p. $1 - \epsilon$
    - $\pi_\epsilon(s) = a$ w.p. $\frac{\epsilon}{|A|-1}$, where $a \neq \pi(s)$, $|A|$ is the number of actions
- This way, all state/action pairs will be observed eventually

ϵ-greedy Monte Carlo Control

Rensselaer

- Use policy iteration with Monte Carlo estimates of q values
- We use ϵ-greedy policies instead of deterministic ones
- Issues?
  - q estimates may be wrong (due to finite data)
  - Does policy improvement theorem still work?
  - After each policy update, need more data to re-estimate

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:

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

ϵ-greedy Monte Carlo Control

Rensselaer

- Issues?
  - q estimates may be wrong (due to finite data)
    - Always a concern with RL methods
    - Especially in high-dimensional problems
  - Does policy improvement theorem still work?
    - Yes, see proof in book
    - Will find the best ϵ-greedy policy

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:

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

ϵ-greedy Monte Carlo Control

- Issues?
  - After each policy update, need more data to re-estimate
    - A major issue with Monte Carlo methods
    - Cannot really get around it
    - But can alleviate it. Any ideas?

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:

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

Off-Policy vs. On-Policy Methods

Rensselaer

- An important distinction in RL
- In on-policy training, our training data was collected using our current policy π
  - I.e., data can be directly used to calculate state values
- In off-policy training, training data was collected using a policy π' that is different from current policy π
  - Need to know some relation between π and π' in order to calculate state values for π

Off-Policy vs. On-Policy, cont'd

- Exploration vs. exploitation
  - Often need to take suboptimal actions in order to explore new state space
  - On-policy methods not well suited for this because we are using the same policy to explore and to make optimal decisions
  - Off-policy allows you to have one exploratory policy (*behavior* policy) and one learning policy (*target* policy)
- Overall, on-policy is simpler but may not explore enough
  - Off-policy is more complex and may converge slowly
    - but may find non-trivial solutions

Off-policy Prediction/Evaluation

- Suppose you have a behavior policy b and a target policy π
  - i.e., your traces were collected using b but you will eventually use π (since it brings higher rewards)
- How do we evaluate $v_\pi(s)$?
  - Intuitively, we should be able to use past knowledge about state/action rewards
- Need to know something about relationship between b and π
  - Note that both need to be probabilistic in this setting
  - If b is not probabilistic, we won't see all state/action pairs
  - π (or π_ϵ) will often be a behavior policy in the next iteration anyway

Off-policy Prediction/Evaluation, cont'd

- First, let's look at $v_b(s)$, in the two-step case:

$$v_b(s) = \mathbb{E}_b[G_t | S_t = s]$$
$$= \mathbb{E}_b[R_{t+1} + \gamma R_{t+2} | S_t = s]$$

- Expanding the expectation:

$$\mathbb{E}_b[G_t | S_t = s] = \sum_g g \, P_b[G_t = g | S_t = s]$$

$$= \sum_g g \sum_{a,a\prime,s\prime} P_b[G_t = g, A_t = a, S_{t+1} = s', A_{t=1} = a' | S_t = s]$$

$$= \sum_g g \sum_{a,a\prime,s\prime} P_b[G_t = g | A_t = a, S_{t+1} = s', A_{t=1} = a', S_t = s] *$$
$$* \, \mathbb{P}_b[A_{t+1} = a', S_{t+1} = s', A_t = a | S_t = s]$$

$$= \sum_{a,a\prime,s\prime} \mathbb{P}_b[A_{t+1} = a', S_{t+1} = s', A_t = a | S_t = s] *$$
$$\mathbb{E}_b[G_t | A_t = a, S_{t+1} = s', A_{t=1} = a', S_t = s]$$

Off-policy Prediction/Evaluation, cont'd

Rensselaer

- First, let's look at $v_b(s)$, in the two-step case:

$$v_b(s) = \mathbb{E}_b[G_t | S_t = s]$$
$$= \mathbb{E}_b[R_{t+1} + \gamma R_{t+2} | S_t = s]$$

- Expanding the expectation:

$$\mathbb{E}_b[G_t|S_t = s] = \sum_{a,a\prime,s\prime} \mathbb{P}_b[A_{t+1} = a', S_{t+1} = s', A_t = a | S_t = s] *$$
$$\mathbb{E}_b[G_t | A_t = a, S_{t+1} = s', A_{t=1} = a', S_t = s]$$

- Note that s, a, a', s' is just the current episode's trace
  - Call that $tr = (s, a, a', s')$
- Expectation becomes

$$\mathbb{E}_b[G_t|S_t = s] = \sum_{tr} \mathbb{P}_b[tr|S_t = s]\mathbb{E}_b[G_t|tr, S_t = s]$$

  - where tr loops over all possible sequences $A_t, S_{t+1}, \ldots, S_T$

Off-policy Prediction/Evaluation, cont'd

- In the general finite-horizon case:

$$\mathbb{E}_b[G_t|S_t = s] = \sum_{tr} \mathbb{P}_b[tr|S_t = s]\mathbb{E}_b[G_t|tr, S_t = s]$$

- Note that the transition probabilities are determined by b
  - But the rewards are independent of b
  - Any policy that generates tr will observe the same rewards:

$$\mathbb{E}_b[G_t|tr, S_t = s] = \mathbb{E}_\pi[G_t|tr, S_t = s]$$

- Allows us to write v_π as a function of v_b:

$$v_\pi(s) = \sum_{tr} \mathbb{P}_\pi[tr|S_t = s]\mathbb{E}_\pi[G_t|tr, S_t = s]$$

$$= \sum_{tr} \frac{\mathbb{P}_\pi[tr|S_t = s]}{\mathbb{P}_b[tr|S_t = s]} \mathbb{P}_b[tr|S_t = s]\mathbb{E}_\pi[G_t|tr, S_t = s]$$

Off-policy Prediction/Evaluation, cont'd

- Allows us to write v_π as a function of v_b:

$$v_\pi(s) = \sum_{tr} \frac{\mathbb{P}_\pi[tr|S_t = s]}{\mathbb{P}_b[tr|S_t = s]} \mathbb{P}_b[tr|S_t = s]\mathbb{E}_\pi[G_t|tr, S_t = s]$$

$$= \sum_{tr} \rho_{t:T} \mathbb{P}_b[tr|S_t = s] \sum_g g\, \mathbb{P}[G_t = g|tr, S_t = s]$$

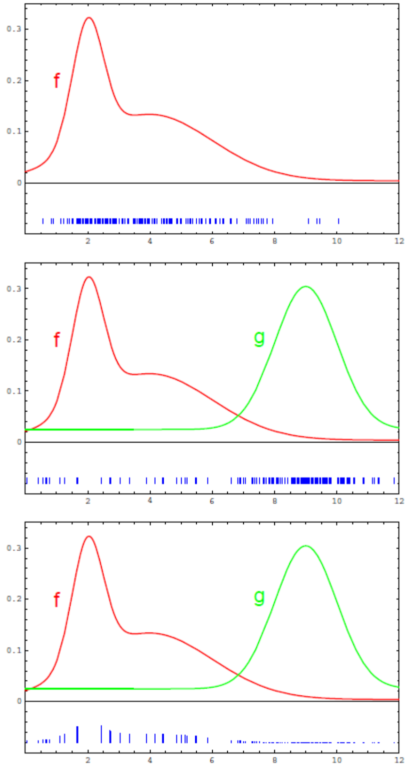
$$= \sum_{g,tr} \rho_{t:T} g \mathbb{P}_b[G_t = g, tr|S_t = s]$$

$$= \mathbb{E}_b[\rho_{t:T} G_t|S_t = s]$$

  - where $\rho_{t:T} = \mathbb{P}_\pi[Tr_t|S_t = s]/\mathbb{P}_b[Tr_t|S_t = s]$
    - Tr_t is a random variable corresponding to the trace starting at t

Importance Sampling

- Since we have no (or little) data from the target policy, we can't use vanilla Monte Carlo to estimate state values
  - Must instead use the relationship between the two policies
  - Importance sampling!
- In importance sampling, we generate data using some distribution, g
  - Called a proposal distribution
- We'd like to reweight the data according to another distribution, f
  - Called the target distribution
- Almost as if we had generated data with the target distribution



Source: Thrun, S., Burgard, W. and Fox, D., 2006. Probalistic robotics. *Kybernetes*.

Computing state values with importance sampling

- In summary, to compute $v_\pi(s)$, we note that

$$v_\pi(s) = \mathbb{E}_b[\rho_{t:T-1} G_t | S_t = s]$$

- Note that the ratio can be simplified

$$\begin{aligned}\rho_{t:T-1} &= \frac{\mathbb{P}_\pi[tr|S_t = s]}{\mathbb{P}_b[tr|S_t = s]} \\ &= \frac{\pi(A_t|S_t)P(S_t, A_t, S_{t+1})\pi(A_{t+1}|S_{t+1}) \ldots P(S_{T-1}, A_{T-1}, S_T)}{b(A_t|S_t)P(S_t, A_t, S_{t+1})b(A_{t+1}|S_{t+1}) \ldots P(S_{T-1}, A_{T-1}, S_T)} \\ &= \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}\end{aligned}$$

  - where the MDP transitions conveniently cancel out

Computing state values with importance sampling, cont'd

- Recall that we are given a set of trajectories from b
  - Call it $\mathcal{T}(s)$
- To estimate $v_b(s)$, we just average all the returns as before

$$\hat{v}_b(s) = \frac{\sum_{tr \in \mathcal{T}(s)} G_t}{|\mathcal{T}(s)|}$$

- Of course, we want to estimate v_π, so we need the extra ρ factor

$$V(s) \coloneqq \hat{v}_\pi(s) = \frac{\sum_{tr \in \mathcal{T}(s)} \rho_{t:T-1} G_t}{|\mathcal{T}(s)|}$$

  - Any challenges with this approach?
  - The ratio ρ can get quite large
    - Multiple divisions by small numbers
    - Variance is unbounded

Weighted Importance Sampling

- Ordinary importance sampling is unbiased but the ratio ρ can be quite large
- To alleviate this issue, one can use weighted sampling instead

$$V(s) := \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T-1}}$$

- Alleviates the effect of very large ratios
- This estimate is biased, but it works well in practice
- Bias converges asymptotically to 0
  – Consistent estimator!

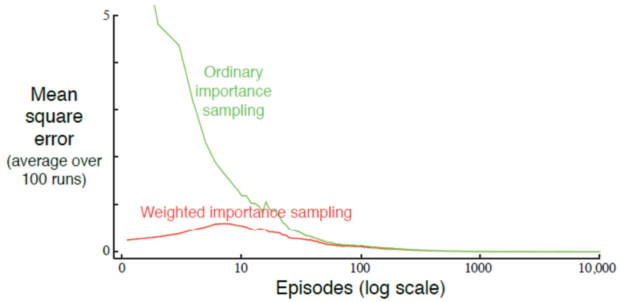


Figure 5.3: Weighted importance sampling produces lower error estimates of the value of a single blackjack state from off-policy episodes.

Incremental Implementation

- Similar to the case of bandits, we don't re-compute the average after every new return
  - Rather, we keep a moving average and update it every time
  - Implementation much faster
- The estimate after k trajectories can be computed as follows

$$V^k(s) = \frac{\rho_{t:T,1}G_{t,1} + \rho_{t:T,2}G_{t,2} + \cdots + \rho_{t:T,k}G_{t,k}}{\rho_{t:T,1} + \rho_{t:T,2} + \cdots + \rho_{t:T,k}}$$

$$= \frac{\sum_{j=1}^{k-1} \rho_{t:T,j}}{\sum_{j=1}^{k} \rho_{t:T,j}} \frac{\left(\rho_{t:T,1}G_{t,1} + \cdots + \rho_{t:T,k-1}G_{t,k-1}\right)}{\sum_{j=1}^{k-1} \rho_{t:T,j}} + \frac{1}{\sum_{j=1}^{k} \rho_{t:T,j}} \rho_{t:T,k}G_{t,k}$$

$$= V^{k-1}(s) + \frac{\rho_{t:T,k}}{\sum_{j=1}^{k} \rho_{t:T,j}} \left(G_{t,k} - V^{k-1}(s)\right)$$

- This is almost temporal difference learning, as we'll see next

Off-policy Monte Carlo Control

- Similar to on-policy, except state values estimated using off-policy methods
  - Behavior policy could be anything but we need enough episodes for each state-action pair

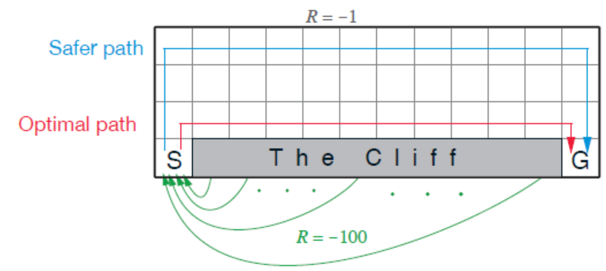
Off-policy MC control, for estimating $\pi \approx \pi_*$

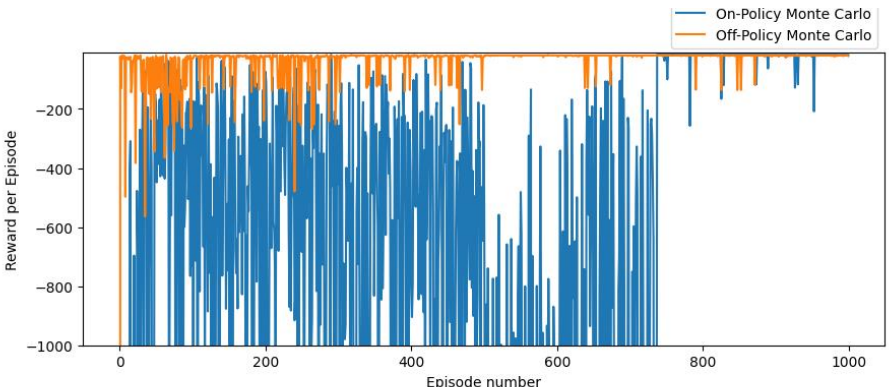
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

Comparison between on- and off-policy control

- Consider the cliff environment again
- We compare the two MC variants
- Both eventually converge to the safe path
  - On-policy approach needs much more data (why?)
    - Off-policy separates exploration from learning
- Epsilons gradually reduced to reduce variance
- Both converge to safer path
  - Why?
  - Not enough exploration
  - Need more data and a lower ϵ





Off-policy Monte Carlo Control, cont'd

- What issues do you see with the algorithm?
- If the behavior policy b is more or less random, it will take a lot of episodes to converge
- An improved approach would be to update b once in a while
  - How can we update it?
  - You can use the current π (or an ϵ-greedy version)
  - This technique actually used frequently in deep RL
    - Stabilizes training

Summary

- Monte Carlo methods are more flexible than dynamic programming
  - Do not require knowledge of the MDP – just prior runs
  - Can use simulators, which may be very complex to model
- The main drawback of Monte Carlo methods is that they require a lot of data
  - Also do not work out-of-the-box on infinite-state systems
- Reinforcement learning is effectively a Monte Carlo method where we learn the value functions instead of approximating them from past data