CMPUT 655 Assignment 4

Seth Akins

September 2024

1 Monte Carlo Theory

1.1 Difference Between Monte Carlo and Dynamic Programming

Compared to dynamic programming, Monte Carlo does not compute the value of states based on the values of other states, but instead it averages samples of the returns for the states. This means we do not need the dynamics of the environment, we can just learn from experience interacting with the environment.

1.2 Recursive Loop for Computing MC Estimate of the Value Function

Let n be any value in $\mathbb{R}$. Then, the loop is as follows:

$V(s) \leftarrow n, \forall s \in S$
$G \leftarrow 0$
$R(s) \leftarrow$ empty list, $\forall s \in S$
for Each step of the episode, t = T - 1, T - 2, ..., 0 **do**
    $G \leftarrow \gamma G + R_{t+1}$
    Append G to $R(S_t)$
    $V(S_t) \leftarrow$ average$(R(S_t))$
end for

1.3 Bootstrapping

Unlike in dynamic programming, Monte Carlo does not use the estimates of other states to estimate a given state s, or in other words, we do not use a "guess from a guess".

1.4 When are First-Visit and Every-Visit Monte Carlo the Same

First-visit and every-visit Monte Carlo are the same when it is only possible to be in a state once in every episode, such as in Blackjack.

1.5 Requirements for Monte Carlo to Converge

In order for Monte Carlo to converge, we make the assumption of continual exploration, which is initially done using exploring starts, so we sample all possible state-action combinations. The other assumption is that we have an infinite number of episodes to do policy evaluation on.

These are not at all practical, as we are unable to do exploring starts with all possible states in larger problems, such as a self-driving car. Practically, we also would never have an infinite amount of training data, or episodes in this case.

1.6 Drawbacks and Advantages of Monte Carlo Methods

The main advantage of Monte Carlo methods is that we do not require the environment's dynamics in order to get our optimal state-value function and policy. We only need to interact with the environment to get episodes.

The disadvantage is that for large problems, it is very inefficient and we need a large number of samples. Another issue is because of the need for continual exploration, we are unable to converge to the optimal policy in on-policy methods and have to deal with more complex computations when using off-policy methods. This primarily includes having to use importance sampling due to using two different policies, one for learning and one for generating episodes.

1.7 Why Importance Sampling is Needed

For off-policy methods, importance sampling is needed because we are using two different policies; one to generate our data that is focused on exploration, and one to learn our optimal policy. These two policies have different probabilities of choosing each action, so we need to account for this when we use the data generated with our exploration policy to improve our learning policy.

2 Exercise 5.5

2.1 First-Visit Estimate

Using first-visit Monte Carlo, we only have one estimate of the non-terminal state in the entire episode. As such, the value is the reward we observe after visiting that state for the first time, which is our total reward as it is our first state and $\gamma = 1$.

$$V_{\text{first-visit}(S_t)=10}$$

2.2 Every-Visit Estimate

For every-visit Monte Carlo, we have an estimate of the state value for each time we visit the state. As we In this case, we end up having the following:

$$V_{\text{every-visit}}(S_t) = \frac{(1+2+3+4+5+6+7+8+9+10)}{10} = 5.5$$

3 Exercise 5.11

This is correct because we only update W if the action taken in the rollout is the greedy action, or in other words the action in the episode was the action for which $Q(s,a)$ was largest for state S_t. If the action was the action was associated with the largest value of $q(s,a)$, by the greedy deterministic policy we are trying to learn, we would pick it with probability 1. In other words, $\pi(A_t|S_t) = 1$, the term in the update equation for W.

4 Coding Implementation

All the code for this assignment is located here and is also in the ZIP folder I submitted.

5 Coding Part 1

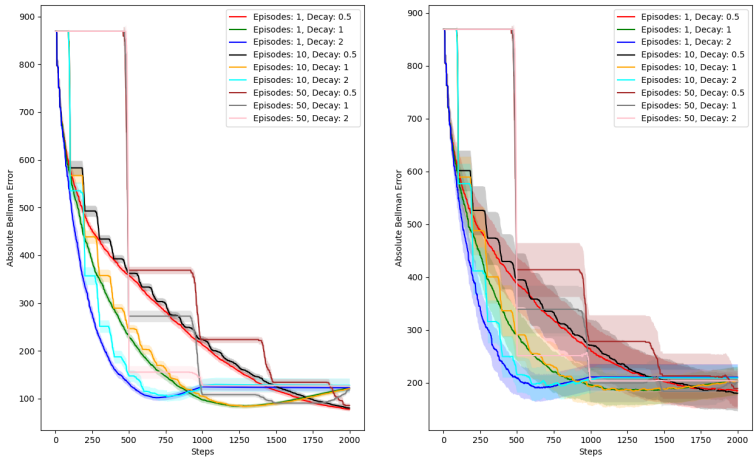

Figure 1: Absolute Bellman Errors for On-Policy Monte Carlo With and Without Stochastic Rewards

Generally, lower values of ϵ decay converged to smaller Bellman errors. Smaller values of episodes per iteration also resulted in smaller Bellman errors, and thus our learned Q is closer to the true Q for our current policy.

Collecting more episodes per update results in each of our updates being more accurate for the current policy and doing fewer updates; however, it also means that the algorithm may choose more sub-optimal actions, especially as our epsilon becomes smaller and our policy begins to favour the greedy action more, even if it is not actually the best action. On average, this leads to larger values of episodes per iteration converging to slightly higher bellman error values and worse Q function approximations.

Faster epsilon decay mean that the algorithm becomes greedier sooner, leading to less exploration. For our grid world, exploring for longer always lead to lower final Bellman errors, and out of curiosity, I verified in my code that decaying epsilon faster made it far less likely that MC would learn the optimal policy.

Changing the number of episodes per iteration does not change the actual value of epsilon, as we still decay at the same rate, and perform the same number of updates. Initializations with lower decay and episodes per iteration did result in the best performance though, likely because the more frequent updates to the action-value function allowed for better actions to be selected more frequently sooner.

6 Coding Part 2

With the addition of stochastic rewards, the trends are different compared to part 1, being more jagged, having much higher variance, and converging to worse estimates of the action-value function for the current policy. This is because the addition of stochastic rewards means the agent needs to explore far more to find the true values of states, as the rewards are much more varied, making the large variance problem in MC even worse.

I think that having random initial states would help the agent, as it would be more likely to explore state-action pairs which are further from the starting state and closer to the terminal state, so it would experience the terminal state more often, which would have a higher reward on average. This would only be equivalent to exploring starts if each state-action pair as a nonzero chance to be selected, not just that the starting state was randomly chosen.

Stochastic rewards highlight the problem that MC estimates have very high variance, which was mentioned as to why the graph trends are different. This is especially highlighted by the much larger confidence intervals seen on the second graph compared to the first.

7 Coding Part 3

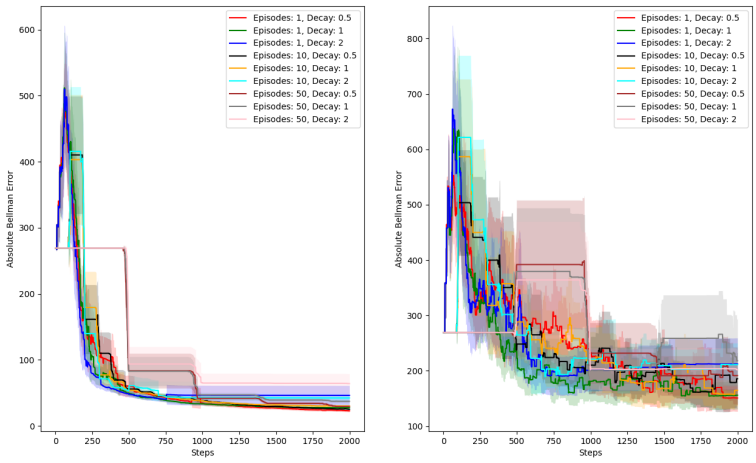

Figure 2: Absolute Bellman Errors for Monte Carlo With Importance Sampling With and Without Stochastic Rewards

Compared to part 1, the plots with importance sampling are far better, converging to lower Bellman errors, meaning they did a better job approximating the true action-value function. IS did help as it allows our target policy to be nearly fully greedy, resulting in a better final policy, but still maintaining a high degree of exploration and updating values of our target policy for states it would be unlikely to visit. It also means that the policy we are learning the whole time is the policy we want, instead of eventually converging to the one we want with a decaying value of epsilon.