Assignment 2 Report

Mariam Qotob

Introduction
Implementation
    On-Policy MC with Exploring Starts
    On-Policy MC Control
    Off-Policy MC Prediction
    Off-Policy MC Control
Results Analysis
    On-Policy MC with Exploring Starts
    On-Policy MC Control
    Off-Policy MC Prediction
    Off-Policy MC Control
References & Resources

Introduction

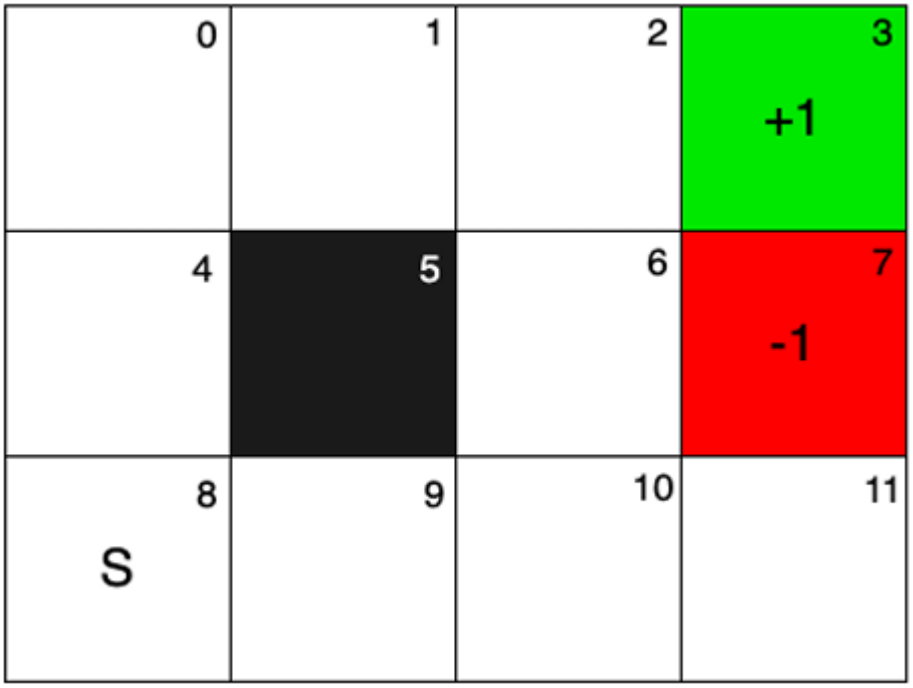
Objective: This assignment aims to solve a simple Gridworld problem using various Monte Carlo (MC) methods. Unlike MDPs, MC methods do not require access to actual Q-values. Instead, they rely on many trials, often starting randomly, to approximate the true values.

MC methods solve the full RL problem by averaging sample returns, similar to sample averaging methods. Instead of averaging immediate rewards for each action, they average the return (discounted, cumulative reward) for each state-action pair. This approach is suitable when simulated experiences can be generated according to some distribution, even if the distribution itself is unobtainable.

Environment: The Gridworld environment has the following properties:

- The agent starts in state 8 (bottom left) and must find the goal state on a stationary grid.
- The agent can move Up, Right, Left, or Down, moving one state per step. If at the edge, it returns to its original position.

- State 5 is a wall and is impassable.
- The goal state gives a +1 reward.
- A danger state gives a -1 reward.
- Each step incurs a -0.1 reward.



Implementation

On-Policy MC with Exploring Starts

The `on_policy_mc_with_exploring_starts` function implements the on-policy Monte Carlo (MC) algorithm with exploring starts. This algorithm estimates the Q function by averaging the returns for each state-action pair during episodes generated from the environment. It assumes that every state-action pair must be visited and have a non-zero probability.

The process begins by initializing variables to store and calculate the returns over many episodes.

Next, a policy function is defined to select actions using an ε-greedy approach. With probability ε, a random action is chosen (exploration), and with probability (1 - ε), the action with the highest estimated value is chosen (exploitation). This policy is what the algorithm aims to evaluate.

The algorithm then loops over episodes, using the policy to select actions until a terminal state is reached.

The return is computed by iterating backward through the episode, summing the discounted rewards from a given state-action pair to the end of the episode.

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
$\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
Generate an episode from S_0, A_0, 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 $Returns(S_t, A_t)$
$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

fig. Pseudo algorithm for MC Exploring Starts

```
1 simple_gw.print_policy(Q_on_policy_exploring_starts)
```

```
right  right  right   GOAL
 left   ----     up  DANGR
   up   left   left   left
```

fig. Policy found by exploring starts

On-Policy MC Control

The `on_policy_mc_control` function implements an on-policy Monte Carlo (MC) control algorithm. Unlike the previous version, this algorithm does not assume exploring starts. Instead, it begins each episode from a fixed starting state, providing a more restricted exploration approach. This means the agent starts from the same state in each episode, focusing on learning the optimal policy from this fixed point.

Employing the epsilon greedy policy solves the problem of maintaining exploration since it ensures that all state-action pairs have a nonzero probability of being visited.

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 $a \in \mathcal{A}(S_t)$:

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

fig. Pseudo algorithm for MC Control

```
1 simple_gw.print_policy(Q_on_policy_control)
```

```
right  right  right   GOAL
 left   ----  right  DANGR
   up   left     up     up
```

fig. Policy found by on-policy control

Off-Policy MC Prediction

The `off_policy_mc_prediction()` function implements an off-policy MC prediction algorithm, marking a shift from on-policy to off-policy methods. This algorithm requires two different policies to simulate episodes: the behavior policy, used to generate transitions, and the target policy, used to evaluate the current policy. Essentially, the goal is to create an association between a behavior policy (b)

and a target policy (π), which results in a slower algorithm but with greater variance.

Importance sampling is a technique used to estimate expected values under one distribution based on samples from another. This method evaluates the Q-values of the target policy using trials generated by the behavior policy.

While the implementation is similar to previous versions in terms of generating episodes, calculating returns, and updating the policy, it differs in the action generation process and the policy being evaluated. Additionally, it includes an update step for importance sampling. The target policy is ε-greedy, while the behavior policy follows a uniform distribution.

Off-policy MC prediction (policy evaluation) for estimating $Q \approx q_\pi$

Input: an arbitrary target policy π
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$

Loop forever (for each episode):
    $b \leftarrow$ any policy with coverage of π
    Generate an episode following 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, $t = T-1, T-2, \ldots, 0$, while $W \neq 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
        $W \leftarrow W \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$

fig. Pseudo algorithm for off-policy MC Prediction

```
1 simple_gw.print_policy(Q_off_policy)
```

```
down  right  right   GOAL
left   ----     up  DANGR
  up   left     up   left
```

fig. Policy found by off-policy control

Off-Policy MC Control

The function `off_policy_mc_control` implements an off-policy MC control algorithm. It also tries to learn an optimal policy using a behavior policy. However, it focuses on maximizing the returns for the learned policy. The behavior policy used is an epsilon greedy policy. The target policy is improved after evaluating the gain with trials using the behavior policy. But basically, it's the same as the previous algorithm (off-policy MC prediction).

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)
$C(s,a) \leftarrow 0$
$\pi(s) \leftarrow \operatorname{argmax}_a Q(s,a)$ (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, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
$C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
$\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
$W \leftarrow W \frac{1}{b(A_t|S_t)}$

fig. Pseudo algorithm for off-policy control

```
1 simple_gw.print_policy(Q_off_policy_control)
```

```
down  right  right   GOAL
left   ----     up  DANGR
  up   left     up   left
```

fig. Results of off-policy control

Results Analysis

We will analyze the results obtained from experiments conducted with different hyperparameters. The main hyperparameters we varied were the number of episodes, discount factor (gamma), and exploration rate (epsilon).

On-Policy MC with Exploring Starts

- **Number of Episodes**: 10,000; 50,000; 100,000
- **Gamma**: 0.9; 0.95; 1.0

Challenges:

- There exists a high variance in returns due to random exploration starts.

Effectiveness:

- **Number of Episodes**: More episodes generally led to better policy convergence. With 100,000 episodes, the policy was near-optimal.
- **Gamma**: Higher values of gamma (closer to 1) placed more importance on future rewards. A gamma of 1.0 provided the best results, indicating that considering long-term rewards is crucial in this environment.

Results:

- For 100,000 episodes, gamma of 1.0 the algorithm found a near-optimal policy that consistently navigated towards the goal while avoiding the danger state.
- Lower episodes and lower gamma values resulted in sub-optimal policies that sometimes failed to avoid the danger state.

On-Policy MC Control

- **Number of Episodes**: 10,000; 50,000; 100,000
- **Gamma**: 0.9; 0.95; 1.0
- **Epsilon**: 0.1; 0.2; 0.3

Challenges:

- Limited exploration from a fixed start state may miss some state-action pairs, leading to a sub-optimal policy.

- Balancing exploration and exploitation is crucial to avoid local optima.

Effectiveness:

- **Number of Episodes**: Similar to the exploring starts method, more episodes resulted in better policies.
- **Gamma**: A gamma of 1.0 again provided the best results, highlighting the importance of future rewards.
- **Epsilon**: An epsilon of 0.1 was found to be effective, providing enough exploration without excessive randomness.

Results:

- For 100,000 episodes, gamma of 1.0, and epsilon of 0.1, the algorithm found a near optimal policy from the fixed start state.
- Lower episodes or gamma values often led to policies that did not avoid the danger state consistently.

Off-Policy MC Prediction

- **Number of Episodes**: 10,000; 50,000; 100,000
- **Gamma**: 0.9; 0.95; 1.0
- **Epsilon (Target Policy)**: 0.1; 0.2; 0.3

Challenges:

- Importance sampling introduces high variance, requiring more episodes for stable convergence.

Effectiveness:

- **Number of Episodes**: More episodes were necessary to achieve stable convergence due to high variance. However with overly high number of episodes it eliminated a number of states.
- **Gamma**: The values of gamma didn’t cause any change in the output policy.
- **Epsilon (Behavior Policy)**: A higher epsilon (0.3) encouraged more exploration but no update for the output policy.

Results:

- I couldn’t find the parameter with the most effective role since al trials resulted in the same policy which is a policy near the optimum.

Off-Policy MC Control

- **Number of Episodes**: 10,000; 50,000; 100,000
- **Gamma**: 0.9; 0.95; 1.0
- **Epsilon**: 0.1; 0.2; 0.3

Challenges:

- Ensuring sufficient exploration while maximizing returns for the target policy.

Effectiveness:

- **Number of Episodes**: High variance necessitated more episodes for stable convergence.
- **Gamma**: A change in gamma didn’t affect the results.
- **Epsilon**: I suppose a higher epsilon (0.3) was beneficial for exploration but this didn’t lead to a better convergence. Results were the same for all values

References & Resources

https://www.youtube.com/watch?v=bpUszPiWM7o

https://www.youtube.com/watch?v=C3p2wI4RAi8

https://medium.com/@numsmt2/reinforcement-learning-chapter-5-monte-carlo-methods-part-1-monte-carlo-prediction-fcc60c9ab726

https://medium.com/@numsmt2/reinforcement-learning-chapter-5-monte-carlo-methods-part-4-off-policy-via-importance-sampling-22190cedc91

https://github.com/nums11/rl/tree/main/chapter_5_monte_carlo