Reinforcement Learning Assignment 2

Chunlin Chen, 519021910463

March 22, 2023

1 Problem Description

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

Figure 1: Gridworld

Evaluate a uniform random policy: $\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$ in the Gridworld problem illustrated in Figure 1 using the following algorithms:

- First-Visit Monte-Carlo Policy Evaluation
- Every-Visit Monte-Carlo Policy Evaluation
- TD(0) Policy Evaluation

2 Implementation of Model-free Prediction Algorithms

2.1 First-Visit Monte-Carlo Method

Let us take a look at the pseudocode of the first-visit Monte-Carlo method illustrated in Figure 2:

Here we process the episode sequence backwards temporally only for the computational convenience. The first check will make sure that the returns of

Algorithm 1: First-Visit MC Prediction

Input: policy π, positive integer *num_episodes*
Output: value function V ($\approx v_\pi$, if *num_episodes* is large enough)
Initialize $N(s) = 0$ for all $s \in \mathcal{S}$
Initialize $\text{Returns}(s) = 0$ for all $s \in \mathcal{S}$
for *episode* $e \leftarrow 1$ ***to*** $e \leftarrow$ *num_episodes* **do**
  Generate, using π, an episode $S_0, A_0, R_1, S_1, A_1, R_2 \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for *time step* $t = T-1$ ***to*** $t = 0$ *(of the episode e)* **do**
    $G \leftarrow G + R_{t+1}$
    if *state* S_t *is* ***not*** *in the sequence* $S_0, S_1, \ldots, S_{t-1}$ **then**
      $\text{Returns}(S_t) \leftarrow \text{Returns}(S_t) + G_t$
      $N(S_t) \leftarrow N(S_t) + 1$
  end
end
$V(s) \leftarrow \frac{\text{Returns}(s)}{N(s)}$ for all $s \in \mathcal{S}$
return V

Figure 2: The Pseudocode of First-Visit Monte-Carlo

a certain state will only be updated at its first appearence in one episode. The code implementation is shown below:

```
def firstVisitMonteCarlo(env, policy, num_episodes, discount_factor=1.0):
    N = {j:0 for j in range(env.nS)}
    V = np.zeros(env.nS)

    for episode_index in range(1, num_episodes + 1):
        if episode_index % 1000 == 0:
            print("\rEpisode {}/{}.".format(episode_index, num_episodes),
                 end="")
            sys.stdout.flush()
        # Generate an episode, which is an array of (state, action,
             reward) tuples
        episode = []
        state = env.reset()
        G = 0

        for step in range(100):
            action = random.choice(policy[state])
            next_state, reward, done = env.step(state, action)

            actions = ['N', 'E', 'S', 'W']
            action = actions[action]

            episode.append((state, action, reward))
            if done:
```

```
            break
        state = next_state

    for index, state_tuple in enumerate(episode[::-1]):
        G = discount_factor * G + state_tuple[2]
        # first visit check
        if str(state_tuple[0]) not in np.asarray(episode)[:,
             0][:len(episode)-index-1]:
            N[state_tuple[0]] += 1
            # incremental update
            V[state_tuple[0]] = V[state_tuple[0]] + (G -
                V[state_tuple[0]]) / N[state_tuple[0]]

return V
```

You may notice that we actually applied the incremental Monte-Carlo updates, therefore, the variable Returns were no longer used, we updated the V(s) directly instead.

2.2 Every-Visit Monte-Carlo Method

The pseudocode of the every-visit Monte-Carlo method is shown in Figure 3. We can see that the only difference of every-visit Monte-Carlo from first-visit

---

Algorithm 2: Every-Visit MC Prediction

Input: policy π, positive integer *num_episodes*
Output: value function V ($\approx v_\pi$, if *num_episodes* is large enough)
Initialize $N(s) = 0$ for all $s \in \mathcal{S}$
Initialize Returns$(s) = 0$ for all $s \in \mathcal{S}$
for *episode* $e \leftarrow 1$ ***to*** $e \leftarrow$ *num_episodes* **do**
    Generate, using π, an episode $S_0, A_0, R_1, S_1, A_1, R_2 \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    for *time step* $t = T - 1$ ***to*** $t = 0$ *(of the episode e)* **do**
        $G \leftarrow G + R_{t+1}$
        Returns$(S_t) \leftarrow$ Returns$(S_t) + G_t$
        $N(S_t) \leftarrow N(S_t) + 1$
    end
end
$V(s) \leftarrow \frac{\text{Returns}(s)}{N(s)}$ for all $s \in \mathcal{S}$
return V

Figure 3: The Pseudocode of Every-Visit Monte-Carlo

Monte-Carlo is that the former does not do the first check and it updates the returns of a certain state as long as it appears in the episode sequence. Therefore, every-visit Monte-Carlo can be easily implemented by slightly modifying the implementation of first-visit Monte-Carlo. And the code is here:

```python
def everyVisitMonteCarlo(env, policy, num_episodes, discount_factor):
    N = {j:0 for j in range(env.nS)}
    V = np.zeros(env.nS)

    for episode_index in range(1, num_episodes + 1):
        if episode_index % 1000 == 0:
            print("\rEpisode {}/{}.".format(episode_index, num_episodes),
                end="")
            sys.stdout.flush()
        # Generate an episode, which is an array of (state, action,
            reward) tuples
        episode = []
        state = env.reset()
        G = 0

        for step in range(100):
            action = random.choice(policy[state])
            next_state, reward, done = env.step(state, action)

            actions = ['N', 'E', 'S', 'W']
            action = actions[action]

            episode.append((state, action, reward))
            if done:
                break
            state = next_state

        for index, state_tuple in enumerate(episode[::-1]):
            G = discount_factor * G + state_tuple[2]

            N[state_tuple[0]] += 1
            # incremental update
            V[state_tuple[0]] = V[state_tuple[0]] + (G -
                V[state_tuple[0]]) / N[state_tuple[0]]

    return V
```

2.3 TD(0) Policy Evaluation

The pseudocode of the TD(0) method is shown in Figure 4.
The implementation is here:

```python
def TemporalDifference0(env, policy, num_episodes, discount_factor,
    learning_rate):
    V = np.zeros(env.nS)

    for episode_index in range(1, num_episodes + 1):
        if episode_index % 1000 == 0:
```

```
            print("\rEpisode {}/{}.".format(episode_index, num_episodes),
                end="")
            sys.stdout.flush()

        state = env.reset()

        while True:
            action = random.choice(policy[state])
            next_state, reward, done = env.step(state, action)

            V[state] = V[state] + learning_rate * (reward +
                discount_factor * V[next_state] - V[state])
            state = next_state

            if done:
                break

    return V
```

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

Figure 4: The Pseudocode of TD(0)

3 Results Analysis

3.1 Monte-Carlo Prediction

3.1.1 First-Visit MC

We set the discount factor $\gamma = 0.9$ and the number of episodes be 10000. The final value function under the uniform random policy is shown in Figure 5:

```
● (rl) PS C:\Users\Muriate_C\Desktop\RL\Assignment2> python MCpred.py
  Episode 10000/10000.
   [[-4.5834399   0.          -5.65087147 -7.91268567 -8.77644271 -9.07934609]
   [-6.68057928 -5.94555124 -7.37387825 -8.42365087 -8.89328186 -9.01222512]
   [-8.13344438 -8.03897344 -8.42889666 -8.70800072 -8.84492588 -8.75322218]
   [-8.86609132 -8.86339839 -8.8623737  -8.71979521 -8.38404704 -7.88262073]
   [-9.18786242 -9.16203063 -8.92754908 -8.39596339 -7.25456114 -5.63118071]
   [-9.30323885 -9.2230188  -8.89019773 -7.94356978 -5.63620763  0.        ]]
```

Figure 5: The Result of First-Visit Monte-Carlo

3.1.2 Every-Visit MC

We set the discount factor $\gamma = 0.9$ and the number of episodes be 10000. The final value function under the uniform random policy is shown in Figure 6:

```
● (rl) PS C:\Users\Muriate_C\Desktop\RL\Assignment2> python MCpred.py
  Episode 10000/10000.
   [[-4.50088258  0.          -5.66587377 -7.86497803 -8.64433179 -8.88355481]
   [-6.5975323  -5.83273409 -7.29970876 -8.28370665 -8.69557888 -8.80459157]
   [-7.97641292 -7.83906898 -8.22379758 -8.5380284  -8.65354344 -8.58325932]
   [-8.68849222 -8.6576289  -8.67541386 -8.52461641 -8.26085961 -7.80660043]
   [-9.02322868 -8.93875916 -8.75428254 -8.22595299 -7.29853789 -5.67686081]
   [-9.13642946 -8.99649165 -8.68764255 -7.83484382 -5.74353368  0.        ]]
```

Figure 6: The Result of Every-Visit Monte-Carlo

3.2 TD(0) Prediction

We set the discount factor $\gamma = 0.9$, the learning rate $\alpha = 1^{-5}$, and the number of episodes be 500000. The final value function under the uniform random policy is shown in Figure 7:

```
● (rl) PS C:\Users\Muriate_C\Desktop\RL\Assignment2> python TD0pred.py
  Episode 500000/500000.
   [[-2.04931282  0.          -3.01912893 -4.3062253  -4.87832991 -5.10124768]
   [-3.36957589 -3.11129168 -3.95759592 -4.58035902 -4.91214724 -5.04231495]
   [-4.37039272 -4.36370986 -4.59680051 -4.78516804 -4.84795959 -4.83143012]
   [-4.96605081 -4.93793786 -4.91218804 -4.79894287 -4.55545351 -4.28049085]
   [-5.27937299 -5.19660663 -5.00451417 -4.61759014 -3.93486365 -3.00216839]
   [-5.40915349 -5.28945343 -4.99019177 -4.36347027 -3.03474643  0.        ]]
```

Figure 7: The Result of TD(0)

3.3 Conclusion

In the experiment, we find that compared to the MC method, TD(0) requires much more episodes to converge, but it also runs much faster.