CS 4180/5180: Reinforcement Learning and Sequential Decision Making (Fall 2020)– Lawson Wong

Name: [Virender Singh]
Collaborators: [Sunny Shukla]

To run the codes please use main.py in the src folder. All the plots can also be seen in the plt folder.

Question 1. *On-policy Monte-Carlo control with approximation.*

Response:
If the authors were to give the Monte Carlo methods, it would most probably be the same with a slight change of n=T. Since for monte carlo, we have to complete the whole episode before updating.
I think it is not reasonable to give pseudocode for the monte carlo methods because they are more like a special case of the TD methods with n=T that's why the pseudocode was not given separately.
The performance of the monte carlo methods on the mountain car would be costly. As it would not start learning unless the first episode is completed and in this task for every step where episode does not terminate, there will be a negative reward. So monte carlo is computationally quite costly here. The proposed methods like TD and SARSA would learn during training and converge much faster.

Question 2. *Semi-gradient expected SARSA and Q-learning*

Response:

(a) To derive semi gradient expected SARSA learning from the Episodic Semi gradient SARSA, we modify the weight update step by replacing $\gamma\hat{q}(S', A', w)$ with $\gamma\Sigma_a\pi(a|S')\hat{q}(S', a, w)$.

```
Algorithm 1: Semi Gradient expected SARSA
  Input: a differentiable action-value function parameterization $\hat{q} : S \times A \times R^d \to R$
  Algorithm parameters: step size $\alpha > 0$, small $\epsilon > 0$
  Initialize value-function weights w $\epsilon R^d$ arbitrarily (e.g., w = 0)
  Loop for each episode:
      S, A $\leftarrow$ initial state and action of episode($\epsilon$-greedy)
      Loop for each step of episode
          Take action A, observe R, S'
          if S' is terminal:
              $w \leftarrow w + \alpha[R - \hat{q}(S, A, w)]\nabla\hat{q}(S, A, w)$
              Go to next episode
          Choose A' as a function of $q(S', ., w)(e.g.\epsilon$-greedy)
          $w \leftarrow w + \alpha[R + \gamma\Sigma_a\pi(a|S')\hat{q}(S', a, w) - \hat{q}(S, A, w)]$ //changed w.r.t SARSA
          $S \leftarrow S'$
          $A \leftarrow A'$
```

(b)

To derive semi gradient Q learning from the Episodic Semi gradient SARSA, we swap the order of choosing action and the weight updation method. Since Q learning is an off policy learning unlike SARSA which will take action based on the previous weights.

Algorithm 2: Semi Gradient Q learning

```
Input: a differentiable action-value function parameterization $\hat{q} : S \times A \times R^d \rightarrow R$
Algorithm parameters: step size $\alpha > 0$, small $\epsilon > 0$
Initialize value-function weights w $\epsilon R^d$ arbitrarily (e.g., w = 0)
Loop for each episode:
    S, A ← initial state and action of episode($\epsilon$-greedy)
    Loop for each step of episode
        Take action A, observe R, S′
        if S′ is terminal:
            $w \leftarrow w + \alpha[R - \hat{q}(S, A, w)]\nabla\hat{q}(S, A, w)$
            Go to next episode
        $w \leftarrow w + \alpha[R + \gamma\hat{q}(S', A', w) - \hat{q}(S, A, w)]$ //changed w.r.t SARSA
        Choose A′ as a function of $q(S', ., w)(e.g.\epsilon$-greedy)
        $S \leftarrow S'$
        $A \leftarrow A'$
```

Question 3.

Response:

Question 4.

Response:

(a) By implementing semi-gradient one-step SARSA for Mountain car, using linear function approximation with the tile-coding features described in the text. I created figures 10.1 shown in figure 1 and figure 2 which shows the mountain car task and the cost-to-go function ($-max_a\hat{q}(s, a, w)$). Figure 3 shows the learning curve as shown in figure 10.2 in the textbook. Here it shows that for different values of α, we can see that the semi-gradient one step SARSA converges with number of steps to complete episode decreasing as it learn.

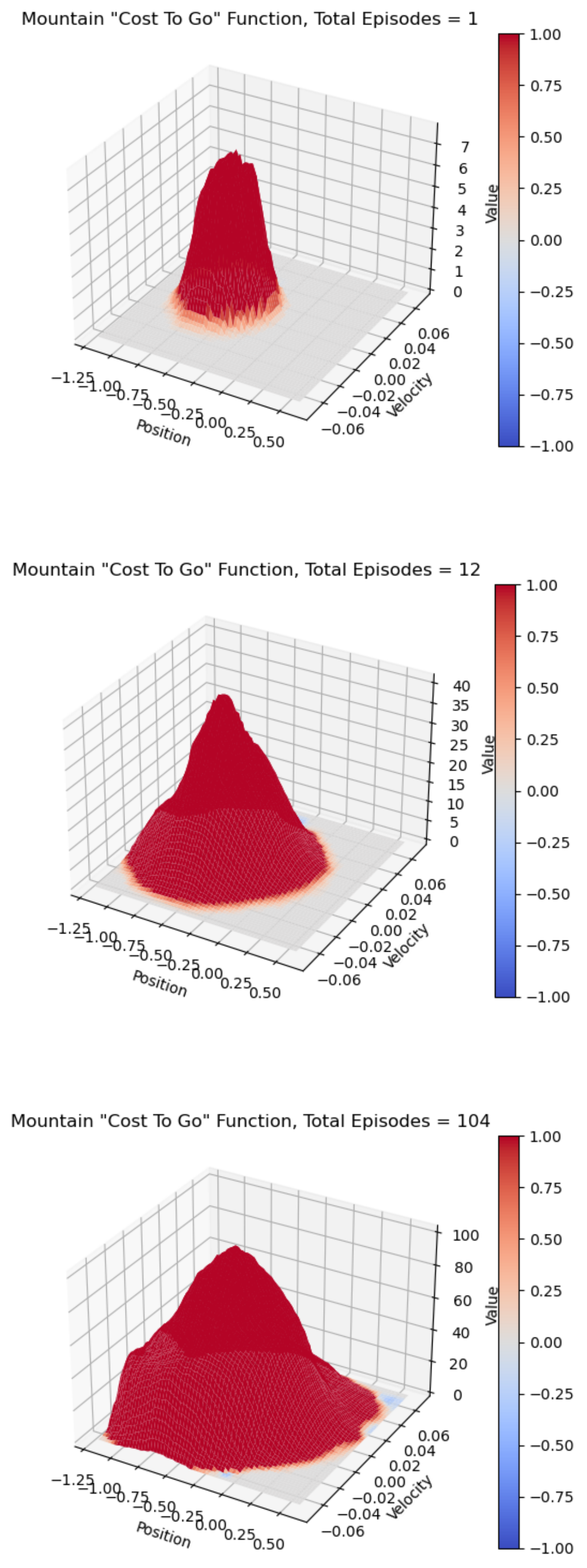

Figure 1: Q4: Figure 10.1(textbook) for The Mountain Car task (upper left panel) and the cost-to-go function, for episodes 1, 12, 104 respectively

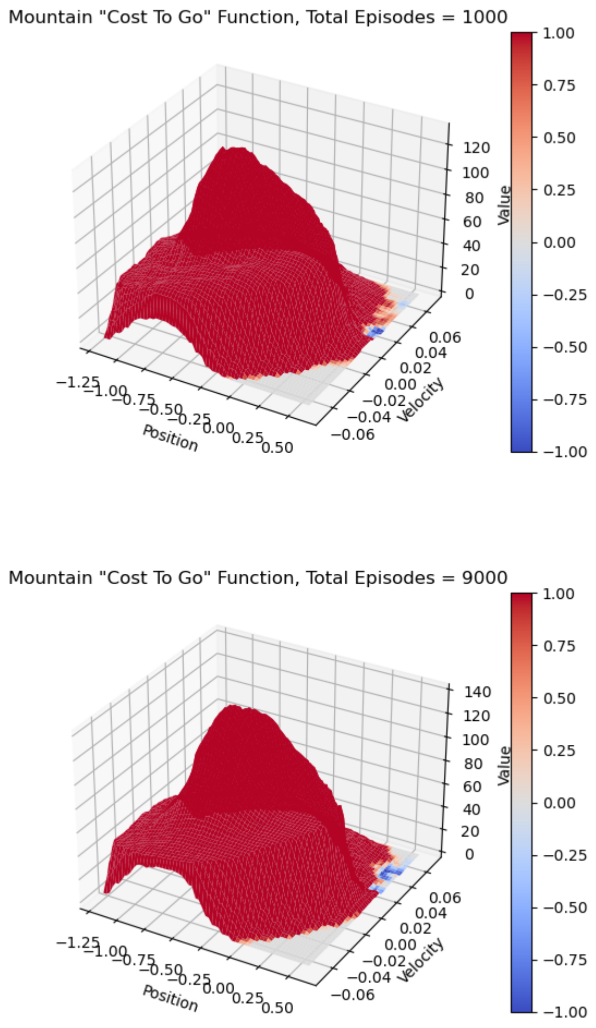

Figure 2: Q4: Figure 10.1(textbook) for The Mountain Car task (upper left panel) and the cost-to-go function, for episodes 1000, 9000 respectively

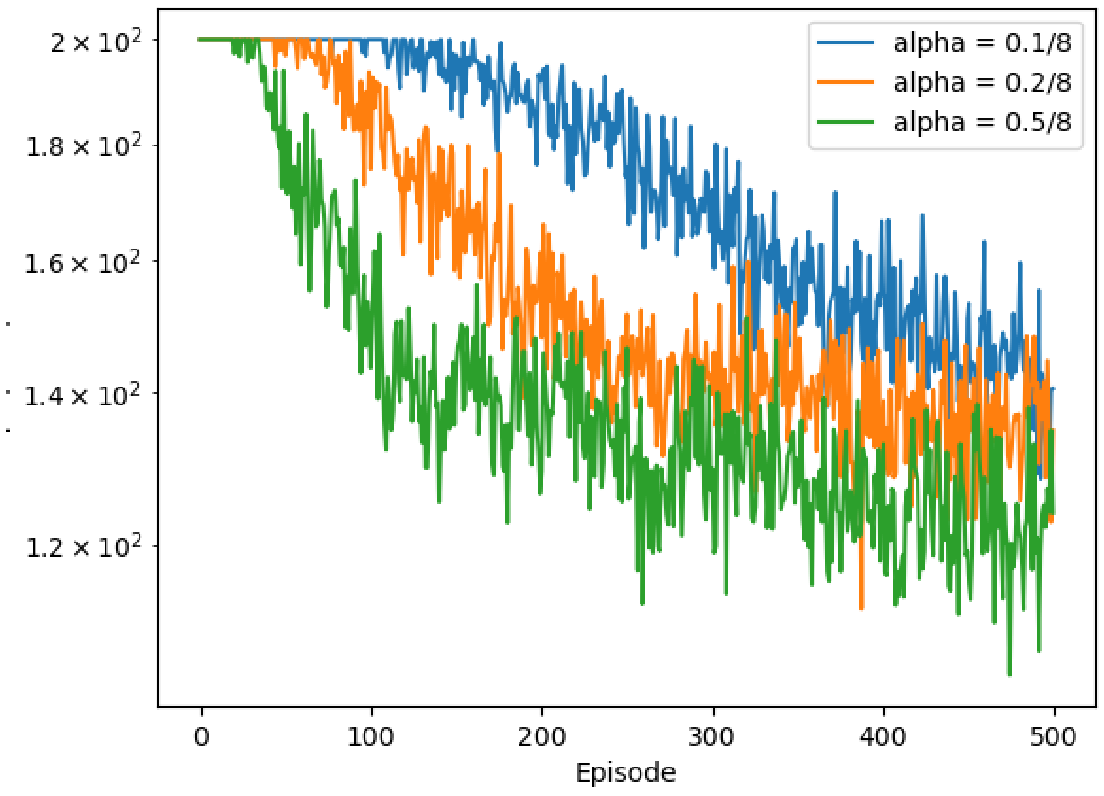

Figure 3: Q4: Figure 10.2(textbook) Mountain Car learning curves for the semi-gradient Sarsa method with tile-coding function approximation and ϵ-greedy action selection.

Question 5.

Response:

$$\overline{VE(w)} \triangleq \Sigma_{s\in S}\mu(s)[v_\pi(s)-\hat{v}(s,w)]^2 \tag{1}$$

(a) Let's substitute $v_\pi(s) = R+\gamma\hat{v}(S',w)$ in (1), we get,

$$\overline{VE(w)} \triangleq \Sigma_{s\in S}\mu(s)[R+\gamma\hat{v}(s',w)-\hat{v}(s,w)]^2 \tag{2}$$

Let us denote $R+\gamma\hat{v}(s',w)-\hat{v}(s,w)$ as δ and put it in the equation 2. We get following equation.

$$\begin{aligned}\overline{VE(w)} &\triangleq \Sigma_{s\in S}\mu(s)\delta^2 &(3)\\ &= E[\delta^2](\text{since } \mu \text{ is the distribution}) &(4)\end{aligned}$$

$$\begin{aligned}\nabla\overline{VE(w)} &= \nabla E[\delta^2] &(5)\\ &= E[\nabla\delta^2] &(6)\\ &= E[2\delta\nabla\delta] &(7)\end{aligned}$$

putting the value of δ back, we get

$$\nabla\overline{VE(w)} = 2E[\delta(\gamma\nabla\hat{v}(S',w)-\nabla\hat{v}(S,w))] \tag{8}$$

Putting it in the weight update equation, we get

$$\begin{aligned}w_{t+1} &= w_t - \frac{1}{2}\alpha\nabla\overline{VE(w)} &(9)\\ &= w_t + \alpha E[\delta(\nabla\hat{v}(S,w)-\gamma\nabla\hat{v}(S',w))] &(10)\\ &= w_t + \alpha E[(R+\gamma\hat{v}(S',w)-\hat{v}(S,w))(\nabla\hat{v}(S,w)-\gamma\nabla\hat{v}(S',w))] &(11)\end{aligned}$$

(b) The objective function for this new learning rule:

$$\overline{VE(w)} = E[(R+\gamma\hat{v}(S',w)-\hat{v}(S,w))^2] \tag{12}$$

We want to minimize this objective function which is the expected value of the squared error of the value of the current state and the sum of reward and the discounted value function of the next state.

We can call it mean squared TD error. No, this is not a good idea as now the loss function is dependent on the next state as well and this will lead the algorithm to not converge correctly.
(c)

$$\overline{BE}(w) \triangleq \Sigma_{s\in S}\mu(s)(E_\pi[R+\gamma\hat{v}(s',w)|s]-\hat{v}(s,w))^2 \tag{13}$$

Let $\delta = R+\gamma\hat{v}(s',w)$, putting this in above equation, we get

$$\begin{aligned}\overline{BE}(w) &\triangleq \Sigma_{s\in S}\mu(s)(E_\pi[\delta]-\hat{v}(s,w))^2 && (14)\\ &= E[(E_\pi[\delta]-\hat{v}(s,w))^2] && (15)\end{aligned}$$

To optimize this objective function, we will take the gradient

$$\begin{aligned}\nabla\overline{BE}(w) &= \nabla E[(E_\pi[\delta]-\hat{v}(s,w))^2] && (16)\\ &= E[\nabla(E_\pi[\delta]-\hat{v}(s,w))^2] && (17)\\ &= E[2(E_\pi[\delta]-\hat{v}(s,w))\nabla(E_\pi[\delta]-\hat{v}(s,w))] && (18)\\ &= 2E[(E_\pi[\delta]-\hat{v}(s,w)(E_\pi(\nabla\delta)-\nabla\hat{v}(s,w))] && (19)\\ & && (20)\end{aligned}$$

Putting the value of $\delta = R+\gamma\hat{v}(s',w)$ and $\nabla\delta = \gamma\nabla\hat{v}(s',w)$.

$$\begin{aligned}\nabla\overline{BE}(w) = 2E[(E_\pi(R+\gamma\hat{v}(s',w)-\hat{v}(s,w))(E_\pi(\gamma\nabla\hat{v}(s',w))-\nabla\hat{v}(s,w))] && (21)\\ && (22)\end{aligned}$$

Putting it in the weight update equation, we get

$$\begin{aligned}w_{t+1} &= w_t - \frac{1}{2}\alpha\nabla\overline{BE(w)} && (23)\\ &= w_t - \alpha E[(E_\pi(R+\gamma\hat{v}(s',w)-\hat{v}(s,w))(E_\pi(\gamma\nabla\hat{v}(s',w))-\nabla\hat{v}(s,w))] && (24)\\ &= w_t + \alpha E[(E_\pi(R+\gamma\hat{v}(s',w)-\hat{v}(s,w))(E_\pi(\nabla\hat{v}(s,w)-\gamma\nabla\hat{v}(s',w)))] && (25)\end{aligned}$$

(d) In this approach expectations are multiplied together. To get an unbiased sample of the product, two independent samples of the next state are required, but during normal interaction with an external environment only one is obtained. One expectation or the other can be sampled, but not both. Therefore, this is a biased algorithm and hence won't converge to the true values. This can work under few conditions, One can be the case of deterministic environments. If the transition to the next state is deterministic, then the two samples will necessarily be the same, and this algorithm is valid. The other way is to obtain two independent samples of the next state, S_{t+1} , from S_t , one for the first expectation and another for the second expectation. In real interaction with an environment, this would not seem possible, but when interacting with a simulated environment, it is. In both these conditions, the algorithm is guaranteed to converge to a minimum of the BE under the usual conditions on the step-size parameter.

Question 6.

Response: