CS5180 – Ex7

Sudhendra Kambhamettu
NUID: 002786797

Q1.

Regarding on-policy Monte-Carlo control with function approximation, since the chapter does not explicitly provide pseudocode for Monte-Carlo methods with function approximation, we can infer the following:

a. Monte-Carlo methods rely on complete episodes for learning. They would adjust the parameters of the function approximator to minimize the difference between the actual returns received at the end of episodes and the values predicted by the function approximator. This adjustment would likely be performed for every episode, using the returns as targets to improve the policy over time.
b. It's reasonable not to provide specific pseudocode for these methods within the context of function approximation because Monte-Carlo methods' implementation can vary significantly depending on the function approximator used (e.g., linear functions, neural networks). The general idea is to update the approximator's parameters to better predict returns, but the specifics can differ.
c. Regarding performance on the Mountain Car task, Monte-Carlo methods might struggle due to the episodic nature of the task and the sparse reward structure. Since Monte-Carlo methods update at the end of episodes and Mountain Car episodes can be quite long without reaching the goal, learning could be slow. The methods might eventually learn a good policy if they explore sufficiently, but they would likely require many episodes and might be less efficient compared to methods that can update more incrementally, such as Temporal Difference learning methods.

Q2.

Pseudo-code for Semi-gradient one-step Expected SARSA for control

```
Initialize weight vector w arbitrarily (for each action value)
Repeat (for each episode):
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        Q_hat = sum over a of pi(a|S') * Q(S', a, w)  // Expected
value over actions, where pi is the policy
        delta = R + gamma * Q_hat - Q(S, A, w)
        w = w + alpha * delta * grad_w Q(S, A, w)
        S = S'; A = A';
    until S is terminal
```

Deriving semi-gradient Q-learning

The following changes must be made to the Expected SARSA control pseudo-code in order to derive Semi-gradient Q-learning.

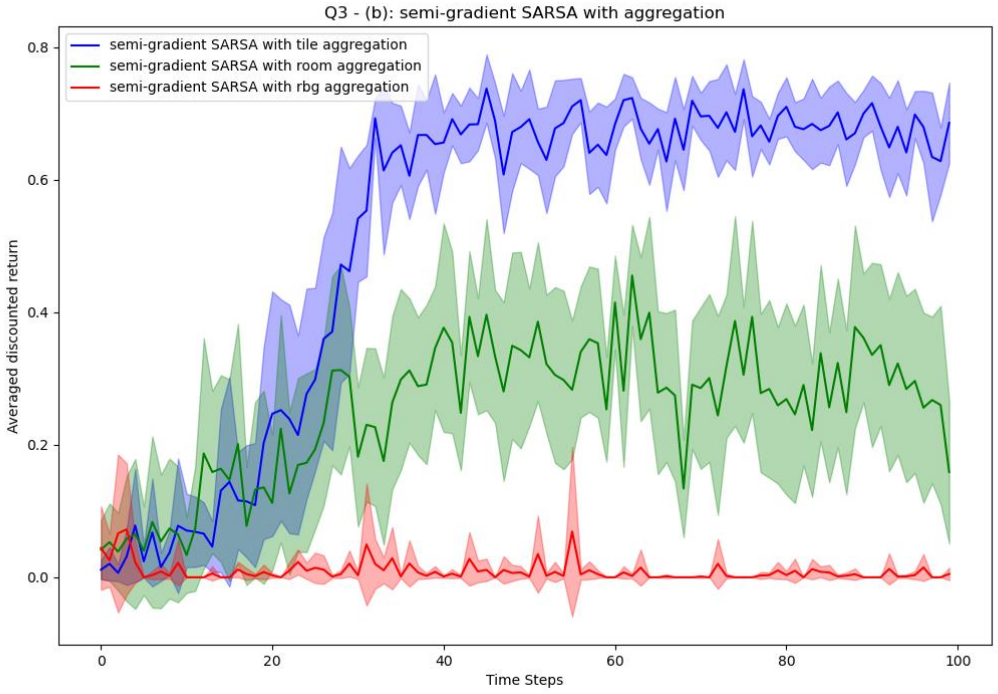
- Instead of calculating the expected value of the next state-action pairs under the current policy π, we select the maximum estimated action value in the next state S' directly. This represents a policy of greedy action at the next step regardless of the current policy π.

Which means, the Q_hat computing part of the pseudocode must be modified as follows:

```
Q_hat = max over a' of Q(S', a', w)  // Maximum value for the next state
```

Q3.

Plots

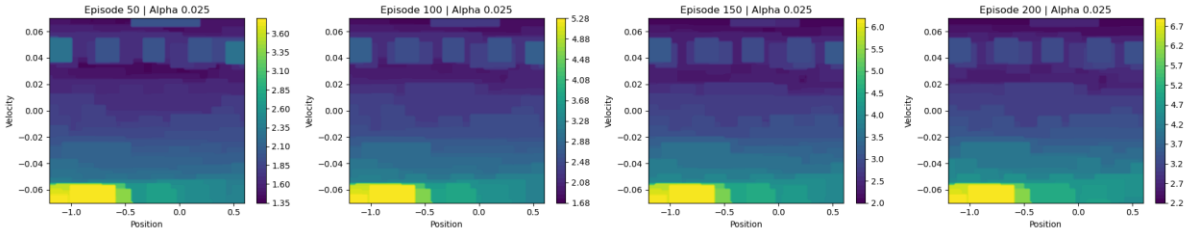


Q4.

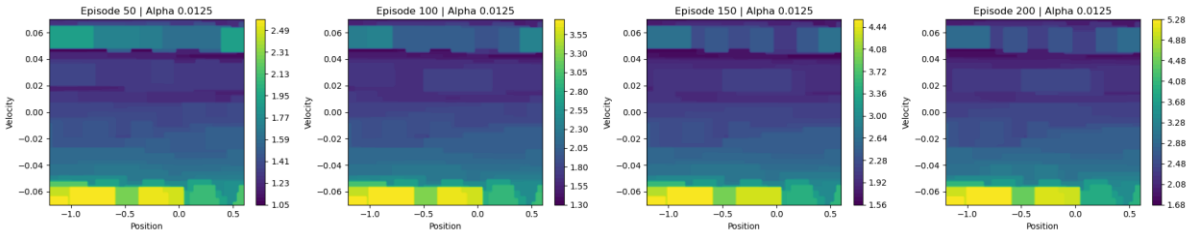
Part a)

Re-creating figure 10.1 (using imshow instead of 3d graphs) for 3 episode stops over 200 episodes.

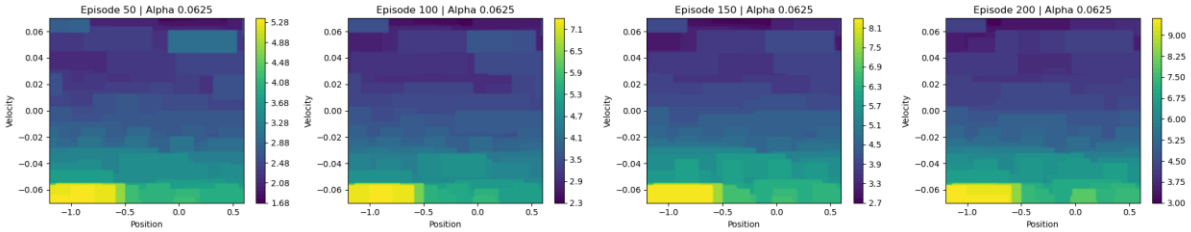
Alpha = 0.025



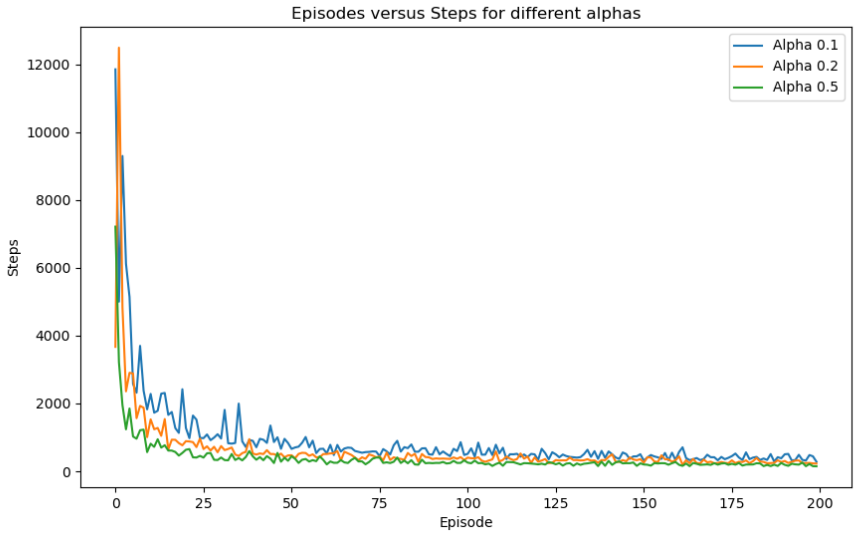
Alpha = 0.0125



Alpha = 0.0625



Re-creating figure 10.2 for 200 episodes averaged over 10 trials.



All the other parts will be submitted at the later date shown.

Q5.

Part a)

To move from semi-gradient to true-gradient TD, we need to adjust the learning rule to account for the gradient of $\hat{v}(S', w)$ with respect to the weights w. In semi-gradient TD(0) the update rule for the weight vector w is given by the following equation:

$$w_{t+1} = w_t + \alpha[R_{t+1} + \gamma\hat{v}(S', w_t) - \hat{v}(S, w_t)]\, \nabla_w \hat{v}(S, w_t)$$

In order to re-derive it, we take into account another term to tom incorporate the gradient with respect to the future value estimate directly in the gradient update. This adjustment makes the learning rule account for the true gradient of the value function approximation across transitions. The learning rule for one step TD would then look like the following:

$$\therefore \boldsymbol{w_{t+1} = w_t + \alpha[R_{t+1} + \gamma\hat{v}(S', w_t) - \hat{v}(S, w_t)]\left(\nabla_w \hat{v}(S, w_t) - \gamma\nabla_w \hat{v}(S', w_t)\right)}$$

The idea here is that the true-gradient adjustment incorporates this consideration i.e., $-\gamma\nabla_w \hat{v}(S', w_t)$, making the learning process more accurate by ensuring that updates fully account for how changes in weights affect not just the current state's estimate but also future states' estimates.

Part b)

The updated learning rule from part (a) which includes an additional term accounting for the next state, optimizes mean-squared Bellman Error (MSBE), which also incorporates the changes in the value estimate of the next state as well. MSBE can be defined as follows:

$$\boldsymbol{MSBE(w) = \mathbb{E}_\pi\left[(E[\left(R + \gamma\hat{v}(S', w)|S\right] - \hat{v}(S, w)\right)^2\right]}$$

Here, the expectation is taken over the distribution of states, actions, and rewards defined by the policy π. Unlike the semi-gradient approach, this formulation directly accounts for how changes in the weight vector ww affect the estimate $\hat{v}(S', w)$ of the next state, in addition to the current state estimate $\hat{v}(S, w)$.

However, by incorporating the gradient of future state value estimates, aim to improve convergence by more accurately aligning updates with the minimization of Mean Squared Bellman Error (MSBE) across the state space. However, this method introduces complexity and potential instability by increasing the computational burden and sensitivity to hyperparameters like the learning rate and discount factor. Moreover, the reliance on precise gradient estimates of future values may heighten the variance of updates and present challenges in high-dimensional environments or where obtaining accurate gradient information is difficult, possibly affecting the algorithm's efficiency and practical applicability.

Part c)

Using the mean-squared Bellman Error to derive a TD-learning rule which optimizes the objective function.

Lets start with defining the MSBE

$$MSBE(w) = \mathbb{E}_\pi\left[(E[\left(R + \gamma\hat{v}(S', w)|S\right] - \hat{v}(S, w)\right)^2\right]$$

Now, lets take the derivative of MSBE with respect to the weight vector w. This can be expressed as follows:

$$\nabla_w MSBE(w) = -2\mathbb{E}_\pi\left[E[R + \gamma\hat{v}(S', w)|s] - \hat{v}(S, w)\right)\left(\nabla_w E[R + \gamma\hat{v}(S', w)|S] - \nabla_w \hat{v}(S, w)\right)]$$

This expression indicates that to minimize the MSBE, we need to adjust the weight vector in the opposite direction of this gradient. Hence, the update rule for ww in gradient TD learning becomes as follows:

$$w_{t+1} = w_t + \alpha[(R_{t+1} + \gamma E[\hat{v}(S', w_t)|S_t] - \hat{v}(S_t, w_t))\ (\nabla_w E[R_{t+1} + \gamma \hat{v}(S', w_t)|S_t\ ] - \nabla_w \hat{v}(S_t, w_t))]$$

Where alpha is the learning-rate.

Simplifying the above update step as the expectation and its gradient may not be directly computable. Approximations or sampling can be used in place of this and therefore the practical update rule will look like the following:

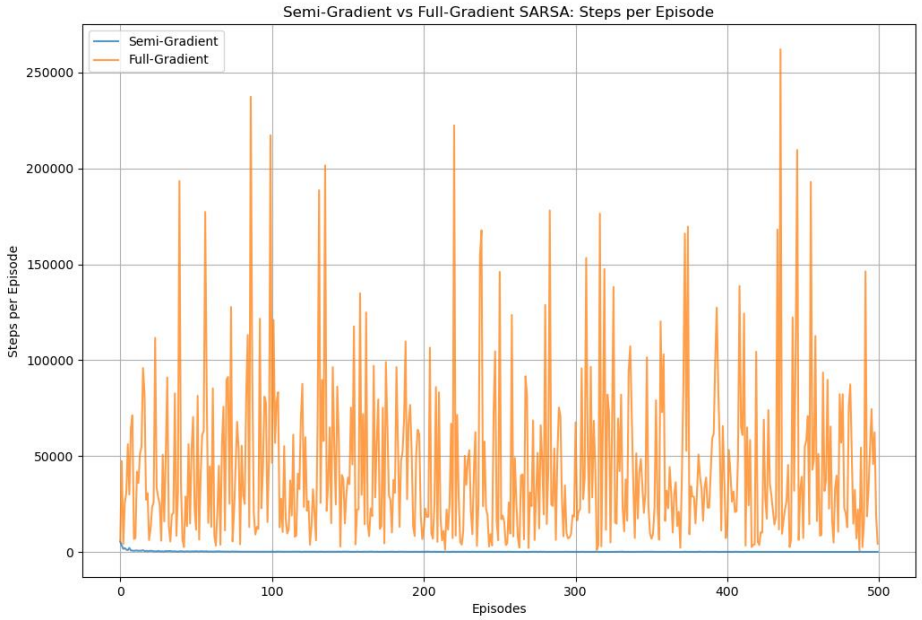
$$\therefore \boldsymbol{w_{t+1} = w_t + \alpha\big(R_{t+1} + \gamma \hat{v}(S'_{t+1}, w_t) - \hat{v}(S_t, w_t)\big)\big(\gamma \nabla_w \hat{v}(S'_{t+1}, w_t) - \nabla_w \hat{v}(S_t, w_t)\big)}$$

Part d)

Implementing the learning rule derived for optimizing the mean-squared Bellman error (MSBE) in reinforcement learning faces challenges like estimating expectations in large or unknown state spaces, computational complexity, high variance in updates, and the difficulty in balancing exploration and exploitation. These issues can hinder practical application, especially in environments with continuous spaces or stochastic elements. However, strategies such as sample-based approximations, leveraging function approximation techniques (like neural networks), employing variance reduction methods (experience replay, target networks), and sophisticated exploration strategies (e.g., ε-greedy, softmax) can mitigate these challenges. Additionally, incorporating regularization or constrained optimization may further stabilize learning and ensure meaningful policy improvements. These solutions' effectiveness largely depends on the environment and task specifics, suggesting a tailored approach to overcoming the inherent difficulties in implementing gradient TD-learning rules for MSBE optimization.

Part e)

Plot



\

As expected, and outlined in the answer above, full gradient is indeed less stable. As suggested above, incorporating regularization should help.