'''

CS 5180 Fall 2022

Exercise 7: Function Approximation

Hongyan Yang

'''

1. 1 point. (RL2e 10.1) On-policy Monte-Carlo control with approximation.

1. On-policy Monte Carlo control with approximation

① Gradient Monte Carlo for Estimating $\hat{q} \approx q^*$ or q_π

Input: a differentiable action-value function parameterization $\hat{q}: S \times A \times R^d \to R$

Input: a policy π (if estimating q_π)

Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$

Initialize value-function weights $w \in R^d$ arbitrarily (eg. $w = 0$)

Loop for each episode:

  Initialize and store $S_0 \neq$ terminal, $t = 0$

  Select and store an action $A_0 \sim \pi(\cdot|S_0)$ or ε-greedy wrt $\hat{q}(S_0, \cdot, w)$

  Observe and store the next reward as R_{t+1} and next state as S_{t+1}

  While S_{t+1} is not terminal, then:

    Select and store an action $A_{t+1} \sim \pi(\cdot|S_0)$ or ε-greedy wrt $\hat{q}(S_0, \cdot, w)$

    $t += 1$, Observe and store the next reward as R_{t+1} and next state S_{t+1}

  $T = t+1$

  $G = \Sigma_{i=1}^{T} \gamma^{i-1} R_i$

  $w \leftarrow w + \alpha [G - \hat{q}(S_0, A_0, w)] \nabla \hat{q}(S_0, A_0, w)$

② It is reasonable not to give pseudocode for them because Monte-Carlo control is an ∞-step TD method in nature. It takes a step size of the length of the whole episode.

③ As can be inferred from Figure 10.4 and because of the nature of Monte-Carlo method. The algorithm will perform poorly as it suffers from high variance in w due to non-bootstrap and slow convergence.

2. 1 point. (RL2e 10.2) Semi-gradient expected SARSA and Q-learning.

2. Semi-gradient expected SARSA and Q-learning

(a). Episodic Semi-gradient Expected SARSA for estimating $\hat{q} \approx q^*$

Input: a differentiable action-value function parameterization $\hat{q}: S \times A \times R^d \to R$

Algorithm parameters: Step size $\alpha > 0$, small $\varepsilon > 0$

Initialize value-function weights $w \in R^d$ arbitrarily (e.g., $w=0$)

Loop for each episode:

  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)

  Loop for each step of episode:

    Choose and take action A, observe R, S' based on policy (e.g., ε-greedy)

    If S' is terminal:

      $w \leftarrow w + \alpha[R - \hat{q}(S,A,w)] \nabla \hat{q}(S,A,w)$

      Go to next episode

    $w \leftarrow w + \alpha[R + \sum_a \pi(a|S) \hat{q}(S',a,w) - \hat{q}(S,A,w)] \nabla \hat{q}(S,A,w)$

    $S \leftarrow S'$

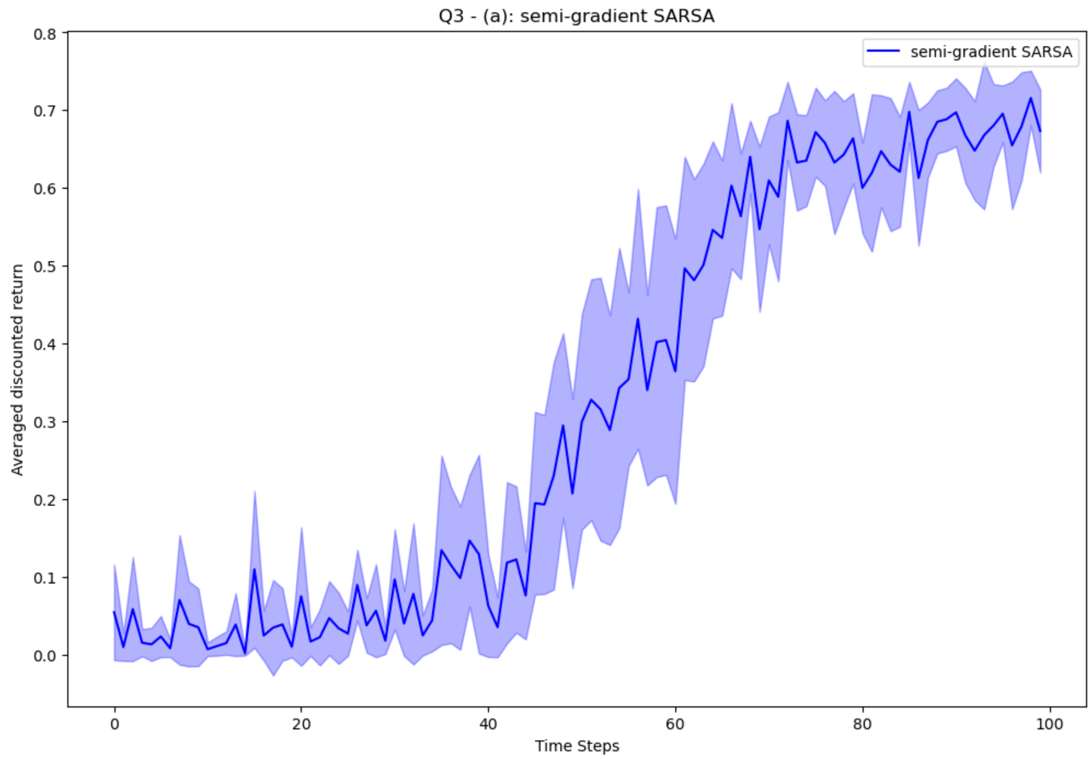
(b). Change the third to last above line of algorithm to

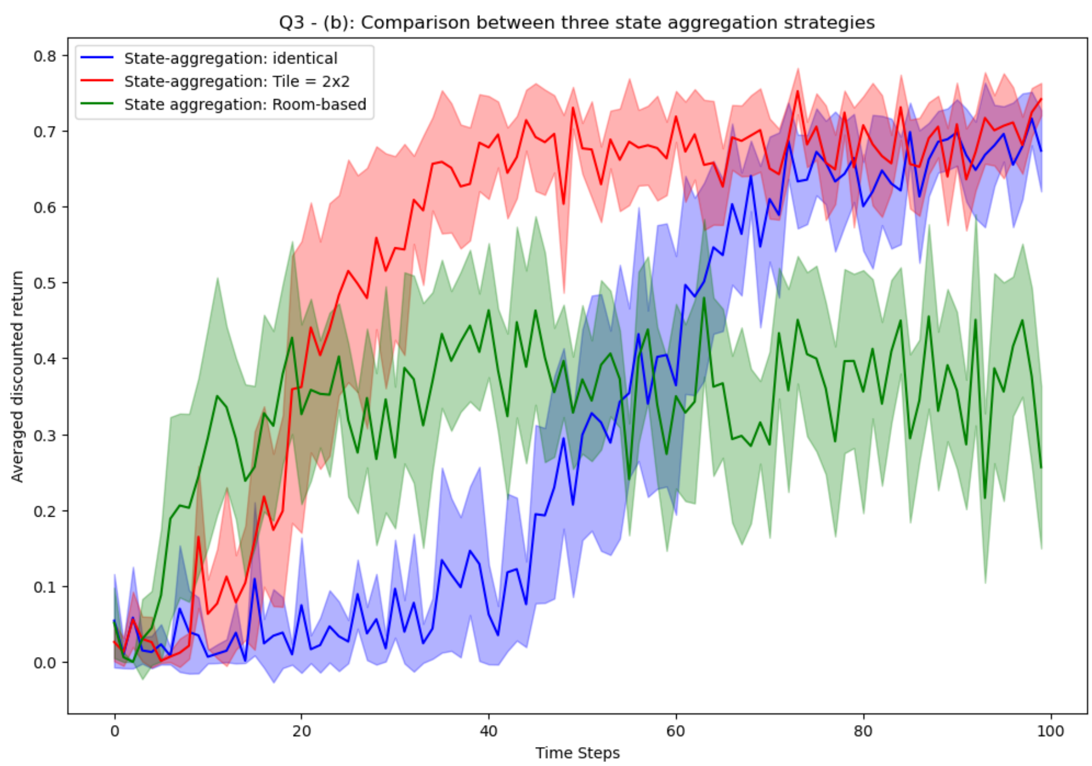
$$w \leftarrow w + \alpha[R + \max_a \hat{q}(S',a,w) - \hat{q}(S,A,w)] \nabla \hat{q}(S,A,w)$$

to derive semi-gradient Q-learning

3. 4/6 points. Four rooms, yet again.

(b) Code/plot: [5180] Please implement the following two state aggregation strategies, and plot the learning curves.





3(b) Written:

Based on the Averaged discounted return plot. Aggregating the states will help to boost the learning speed. Because there's fewer feature weights to be estimated along with a smaller state space. However, we should not aggregate the states as much as possible because an over aggregated state space will be too small to derive an "accurate" optimal policy and a vague policy will result in poor overall performance measured by accumulated return.

3(c) Written:

Yes, there's a difference between the results of semi-gradient SARSA and Q-learning. ① With identical state-aggregation, Q-learning converges to a greedy policy while SARSA converges to an $\mathcal{E}$-greedy policy. The discounted return is slightly higher under Q-learning and have quicker learning rate

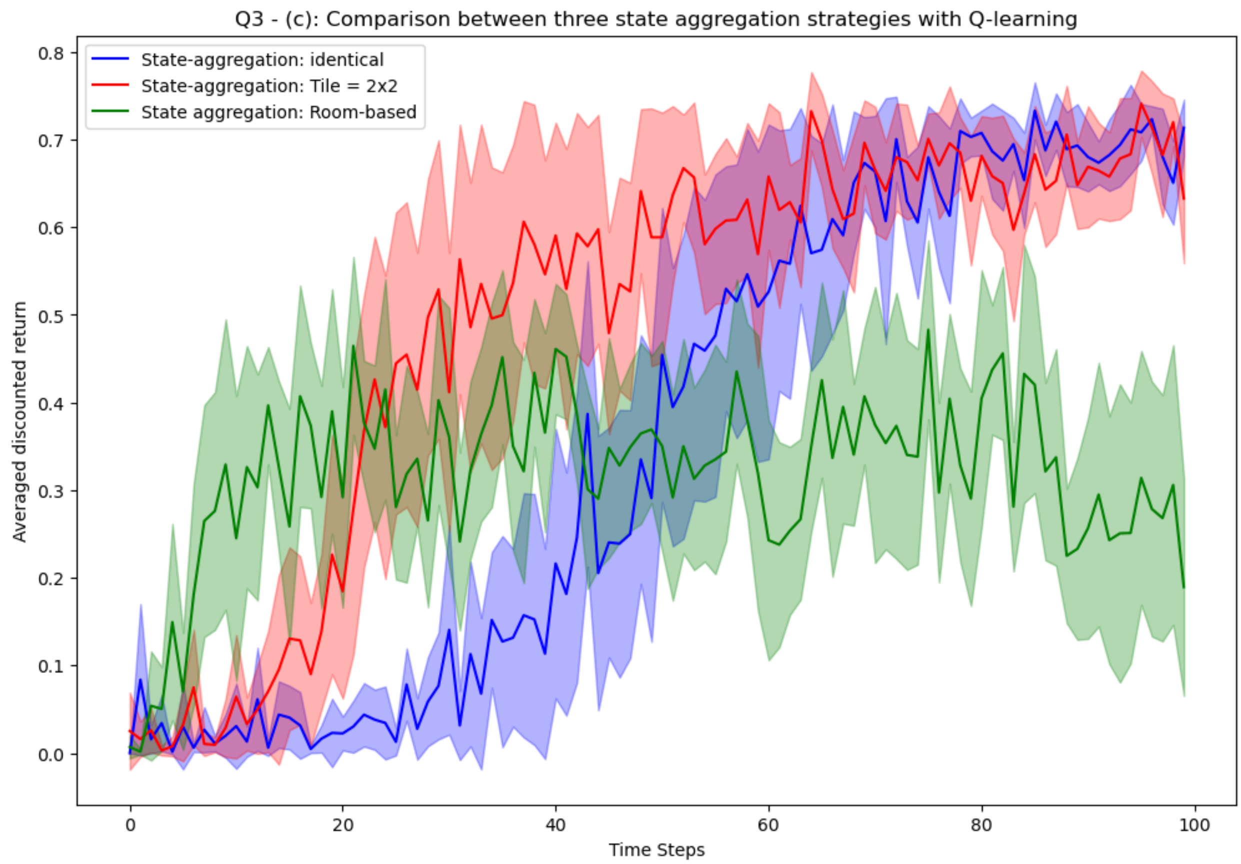
② With 2x2 and room aggregation, Q-learning converged with a slower learning rate and have higher variance.

3(d) Written:

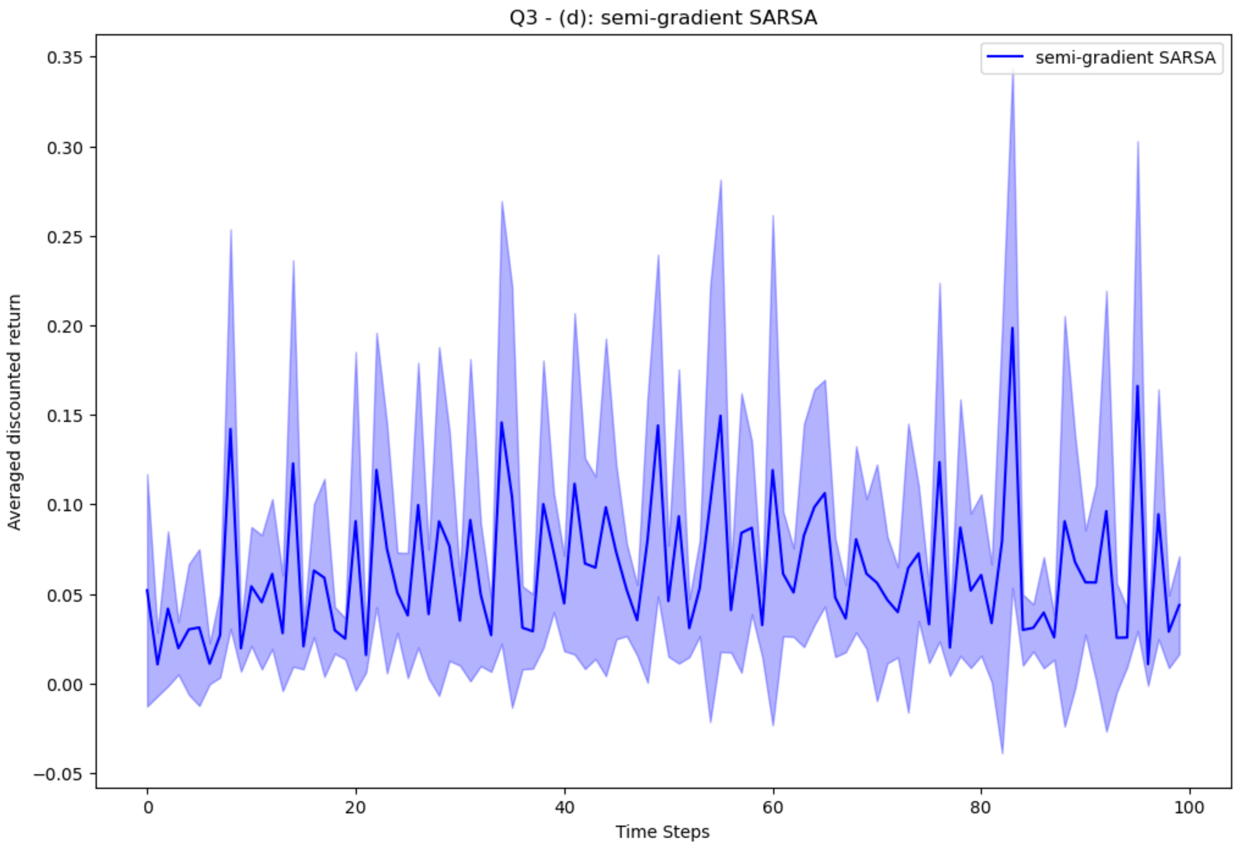
① The constant feature is necessary as it prevents the bias resulting from forcing the linear function approximation to pass the origin. Without it, the performance measured by discounted return will suffer from higher variance because of the inherent bias to pass the origin and will have lower convergence results. I incorporated actions into features by concatenate "one-hot" action after state-features in the form of [x, y, 1, up, down, left, right].

② It performs worse than state aggregation. Because this time we incorporate actions into features, which approximates the original Q values in a much higher degree. Also, the feature generated by only state's coordinates doesn't consider any polynomial terms to consider interactions between features.

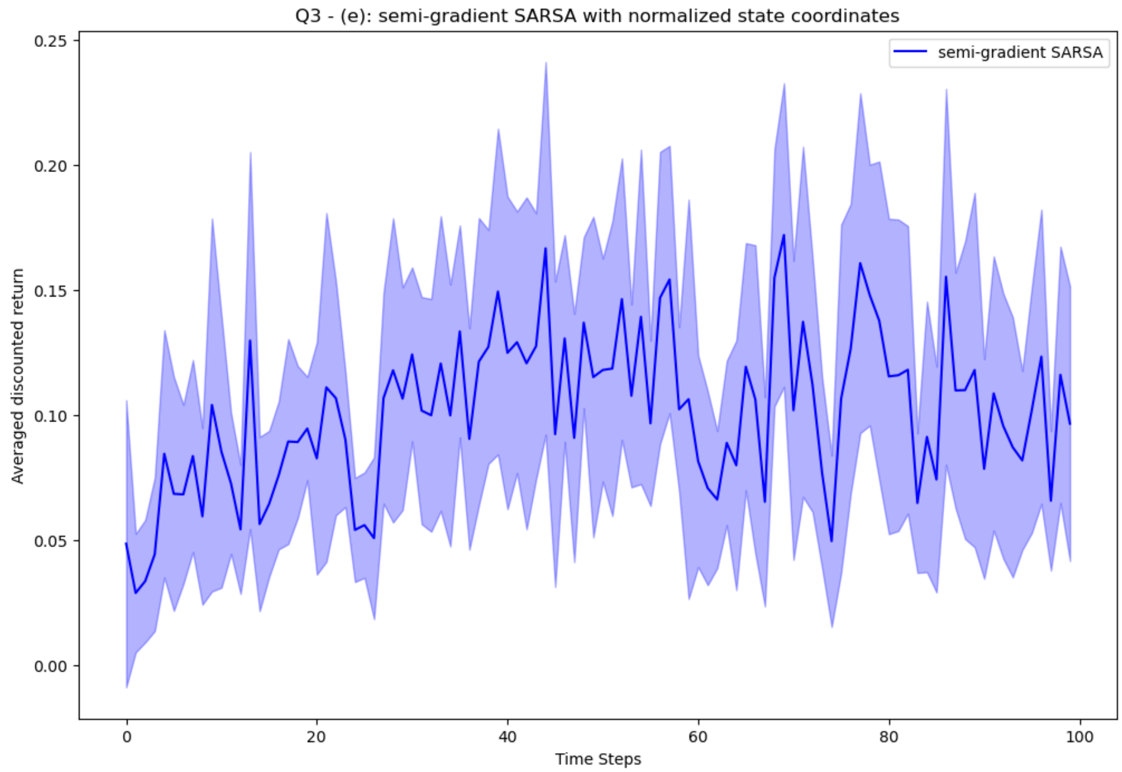
(c) [Extra credit.] 1 point. Implement the semi-gradient one-step Q-learning and resolve (a) and questions above. Comment on whether there is a difference between the results of semi-gradient SARSA and semi-gradient Q-learning?

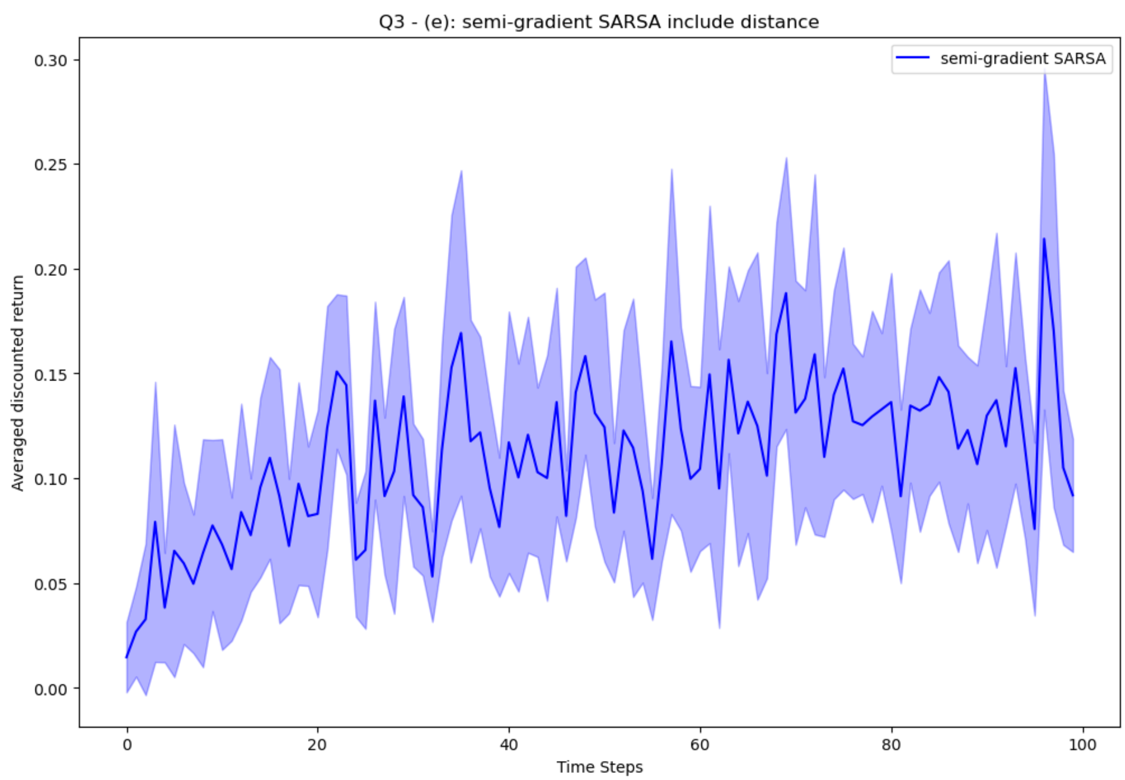


3. (d) Code/plot/written:



3. (e) Code/plot/written:





3(e) Written:

The linear function approximation with normalized state coordinates performs better than the strategy without normalization. And the strategy includes distance performs better than all the others. It shows the importance to pick the right features in linear function approximation and with only a feature added the overall performance differs greatly.