En 7: FUNCTION APPROXIMATION

Friday, November 18, 2022   8:56 PM

1. **1 point.** (RL2e 10.1) *On-policy Monte-Carlo control with approximation.*
Written: We have not explicitly considered or given pseudocode for any Monte Carlo methods in this chapter.
    - What would they be like?
    - Why is it reasonable not to give pseudocode for them?
    - How would they perform on the Mountain Car task?

Ans 1

As MonteCarlo calculate return at each time step & then has to back-track & update the states of the episode, making G to include reward but not the next state.

The book have not provided the pseudocode for the MonteCarlo methods explicitly in the function approximation because the computational cost of the Monte Carlo method is high in the complex setting which makes it unreasonable to use.

Monte Carlo method is a variation of TD method so writing the pseudocode will be similar to n-step SARSA not considering terminal state.

In the Mountain Car task, the Monte Carlo method will not learn until the end of the episode, the amount of step increases the computational complexity.

2. **1 point.** (RL2e 10.2) *Semi-gradient expected SARSA and Q-learning.*
Written:
    (a) Give pseudocode for semi-gradient one-step Expected Sarsa for control.
    (b) What changes to your pseudocode are necessary to derive semi-gradient Q-learning?

Ans 2.(a)

Expected SARSA + Control

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode

Else
$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \sout{\gamma\hat{q}(S', A', \mathbf{w})} - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
$S \leftarrow S'$

$\downarrow$ $[\gamma \sum \pi(a|s')\, \hat{q}(s', a, w)]$

2. b) SEMI Gradient Q Learning

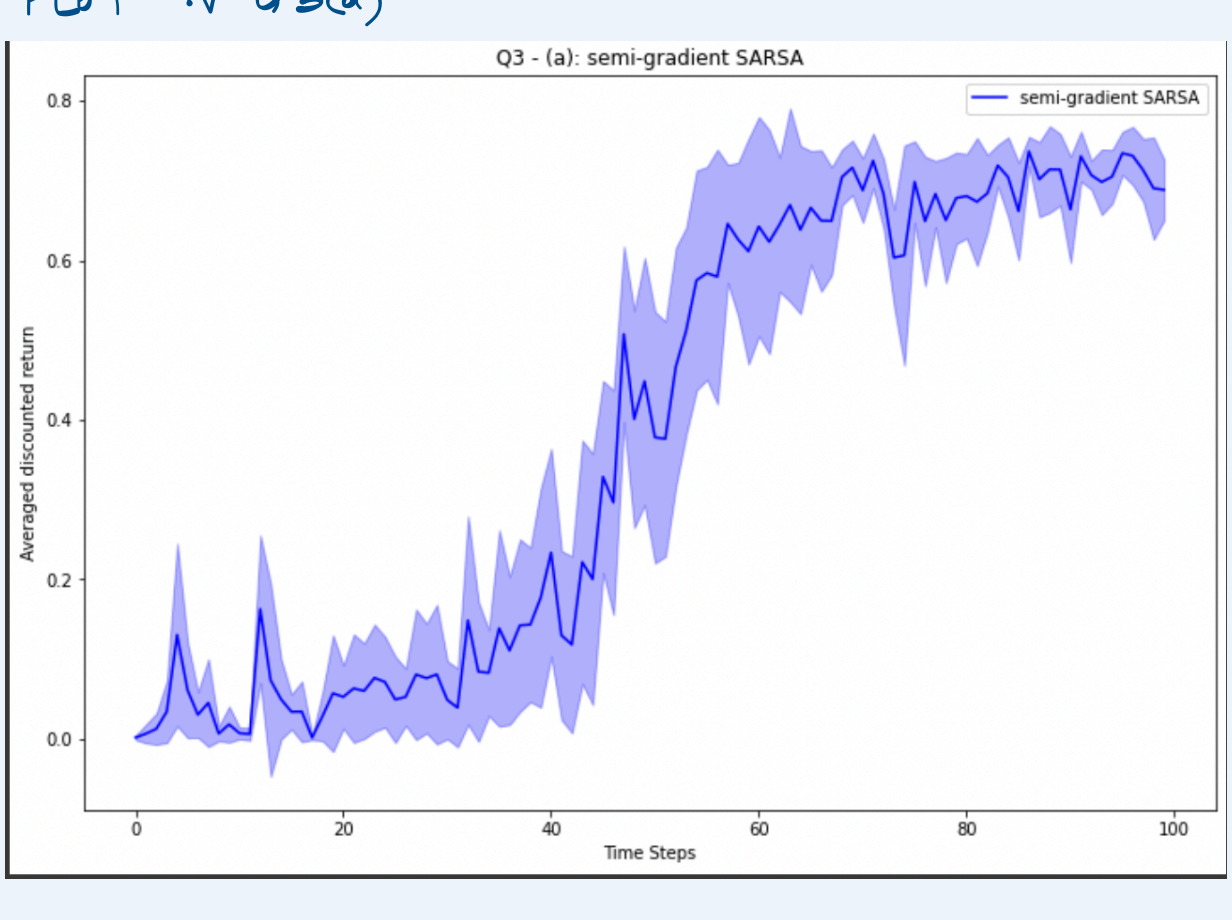
Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode

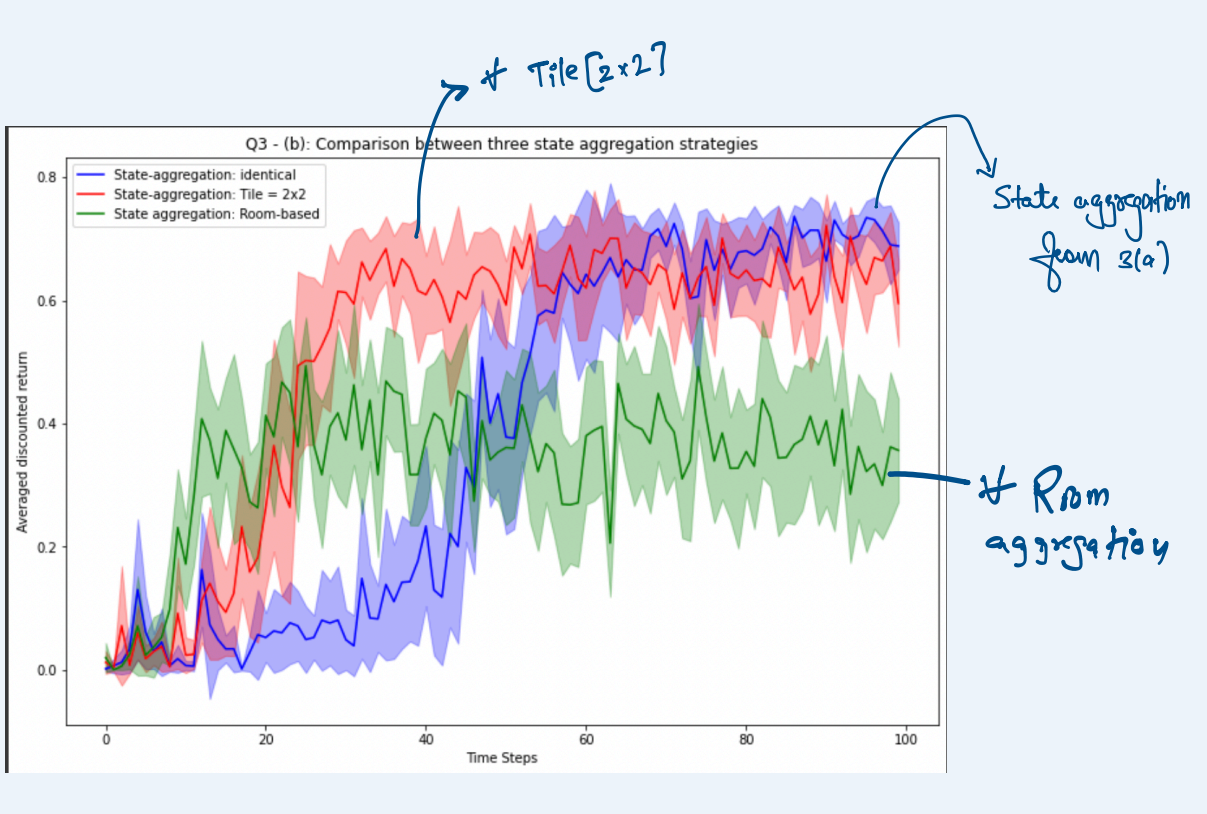
Else
$w \leftarrow w + \alpha\left[R + \gamma \max_a \hat{q}(s, a, w)\right] \nabla \hat{q}(S, A, w)$
$S \leftarrow S'$

Q.3.

a) PLOT of Q3(a)



3(b)

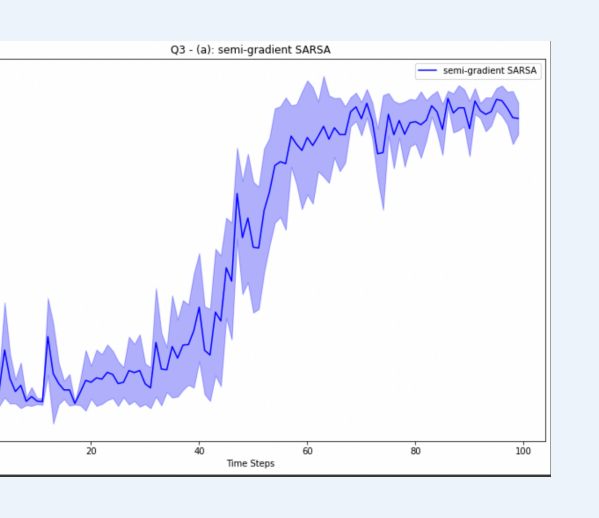


The return of aggregation for the Tile [2x2] & the room, can be observed from the above graph.
In the Tile [2x2] aggregation the learning speed is fast initially then it become somewhat constant which is expected as intially it learns using the reduced tile state space.

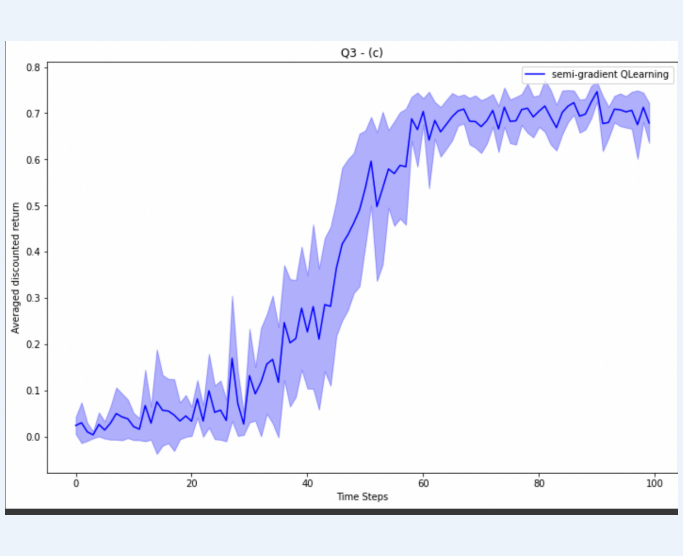
for the Room aggregation the learning slightly better till around 50 timesteps then the without state aggregation but as the time step increase it doesn't learn max because the workable state space has reduced a lot to only 8 state space from around 110-120 state space.

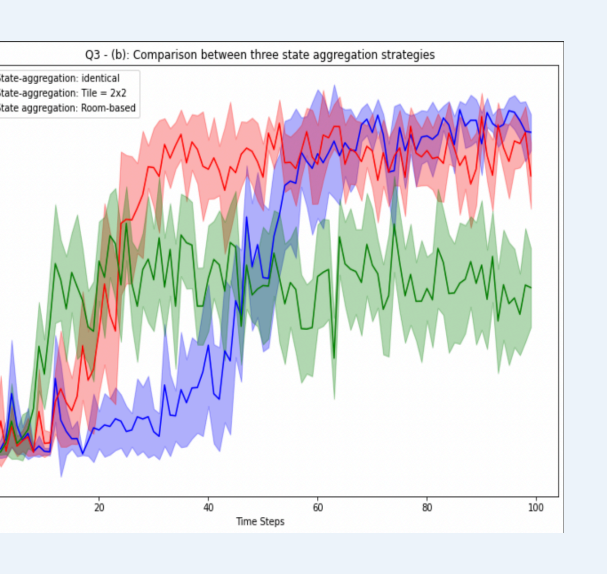
3(c)

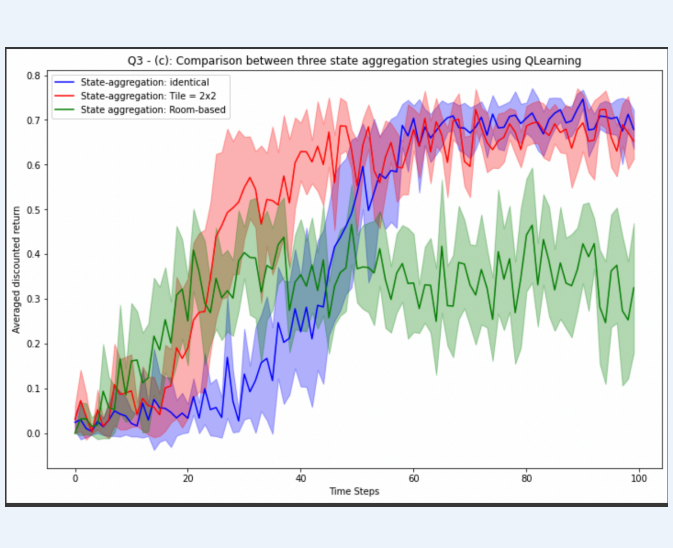
Semi Gradient SARSA



SEMI GRADIENT Q Learning







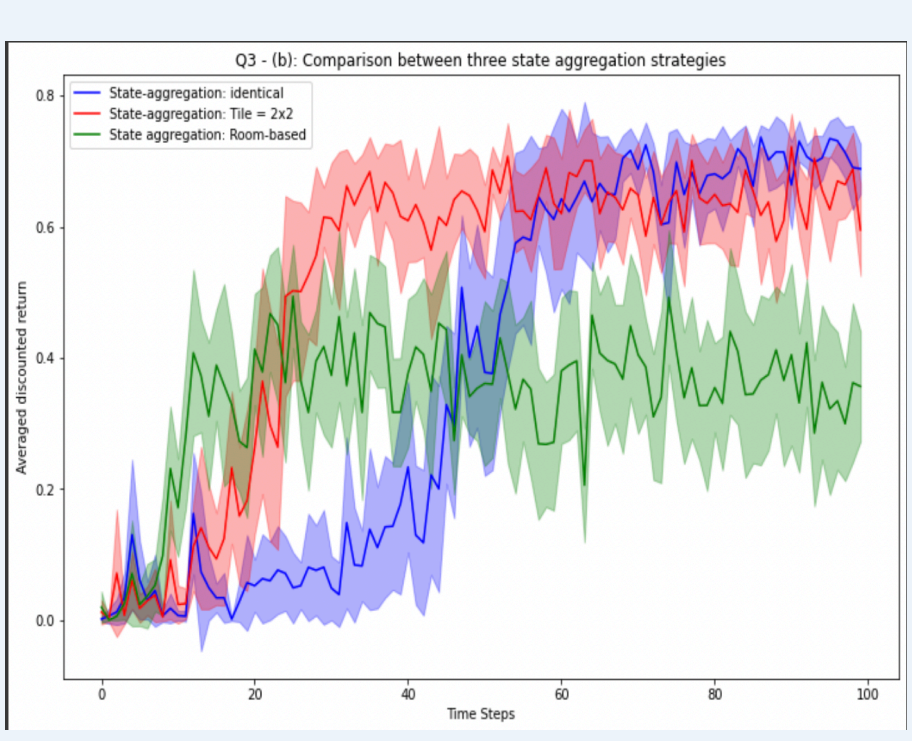
As we can observed with the non aggregated SEMI Gradient SARSA & SEMI Gradient Q learning.
In the Semi Gradient Q Learning the Graph has more smoother learning curve than the Semi-Gradient SARSA, as we can expect as the Q Learning choose from the value of $\max_a$. but for the state aggregation much improvement not observed as the states get reduced & choosing the $\max_a$ among them also limits.

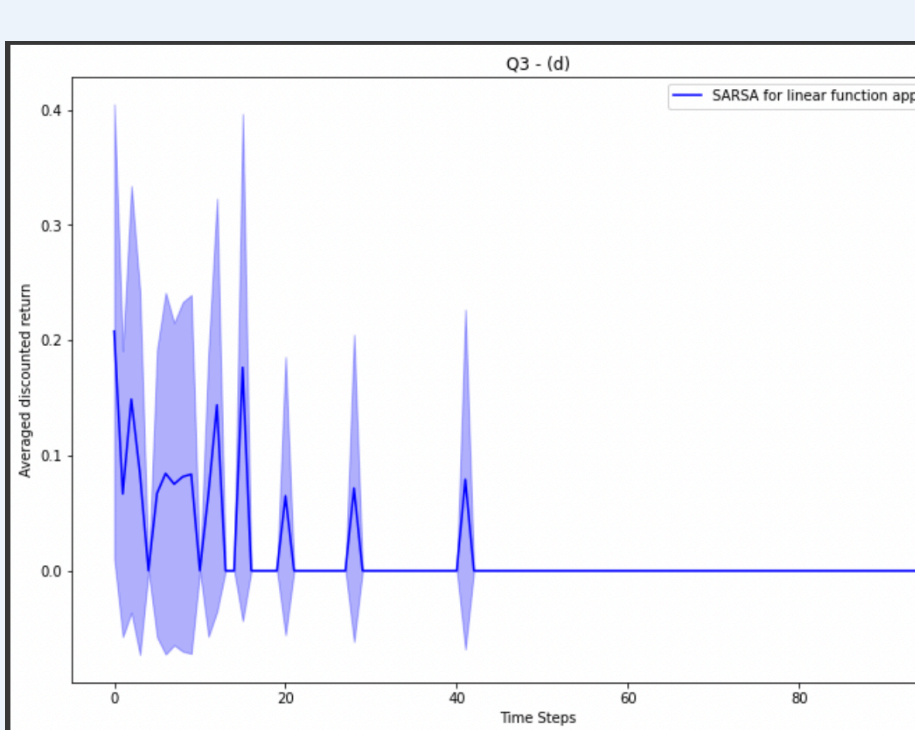
Ans 3.(d) The constant factor is necessary as it make the function approximation less biased & less error induced. If we do not incorporate the constant the function will be more biased as the function will cross the (0,0) & the error will be large.

The feature vector used is [x, y, 1, UP, down, left, right]
for action I have defined like as follows

UP [1, 0, 0, 0]
Down [0, 1, 0, 0]
Left [0, 0, 1, 0]
Right [0, 0, 0, 1]

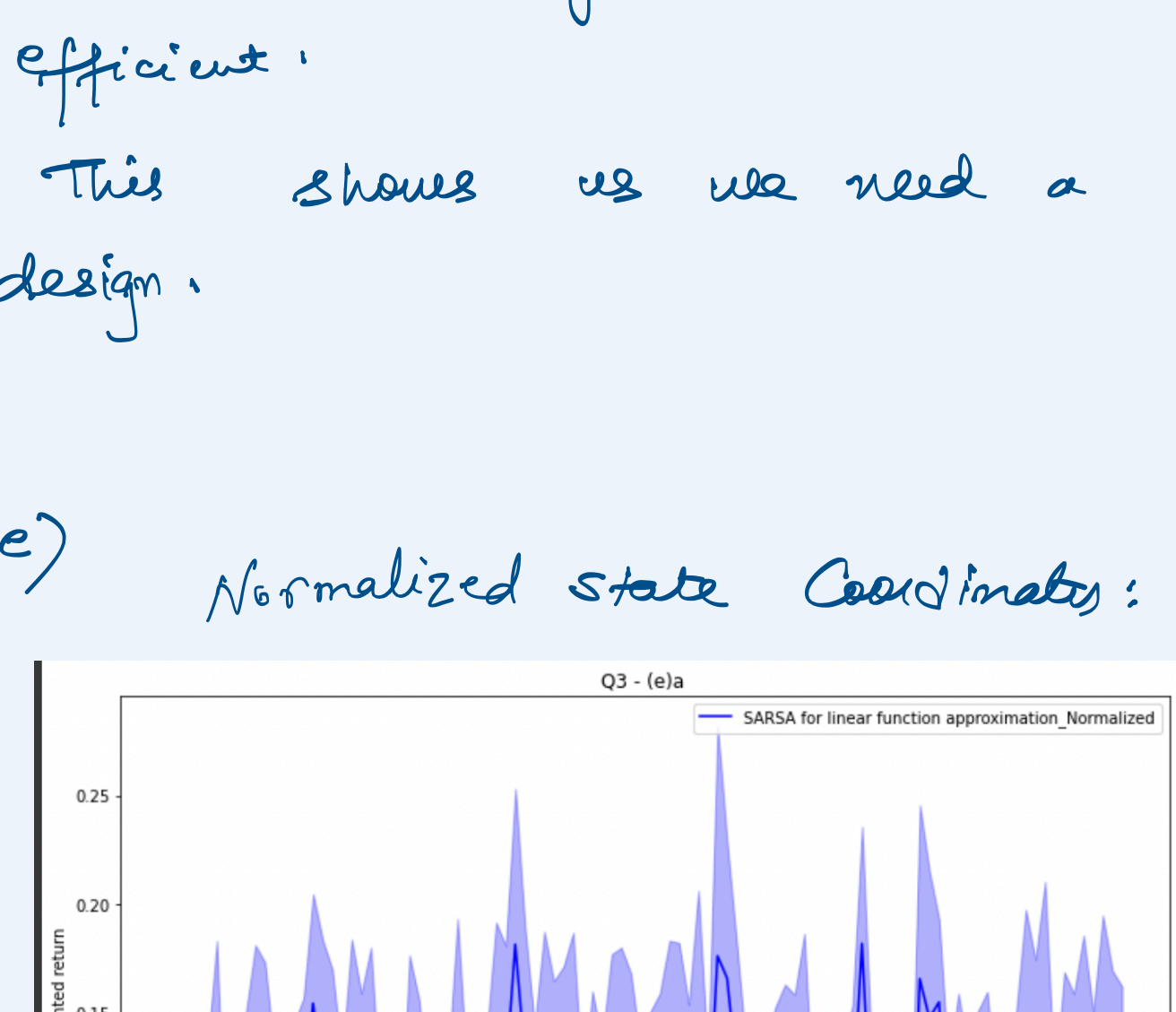
← for State aggregation





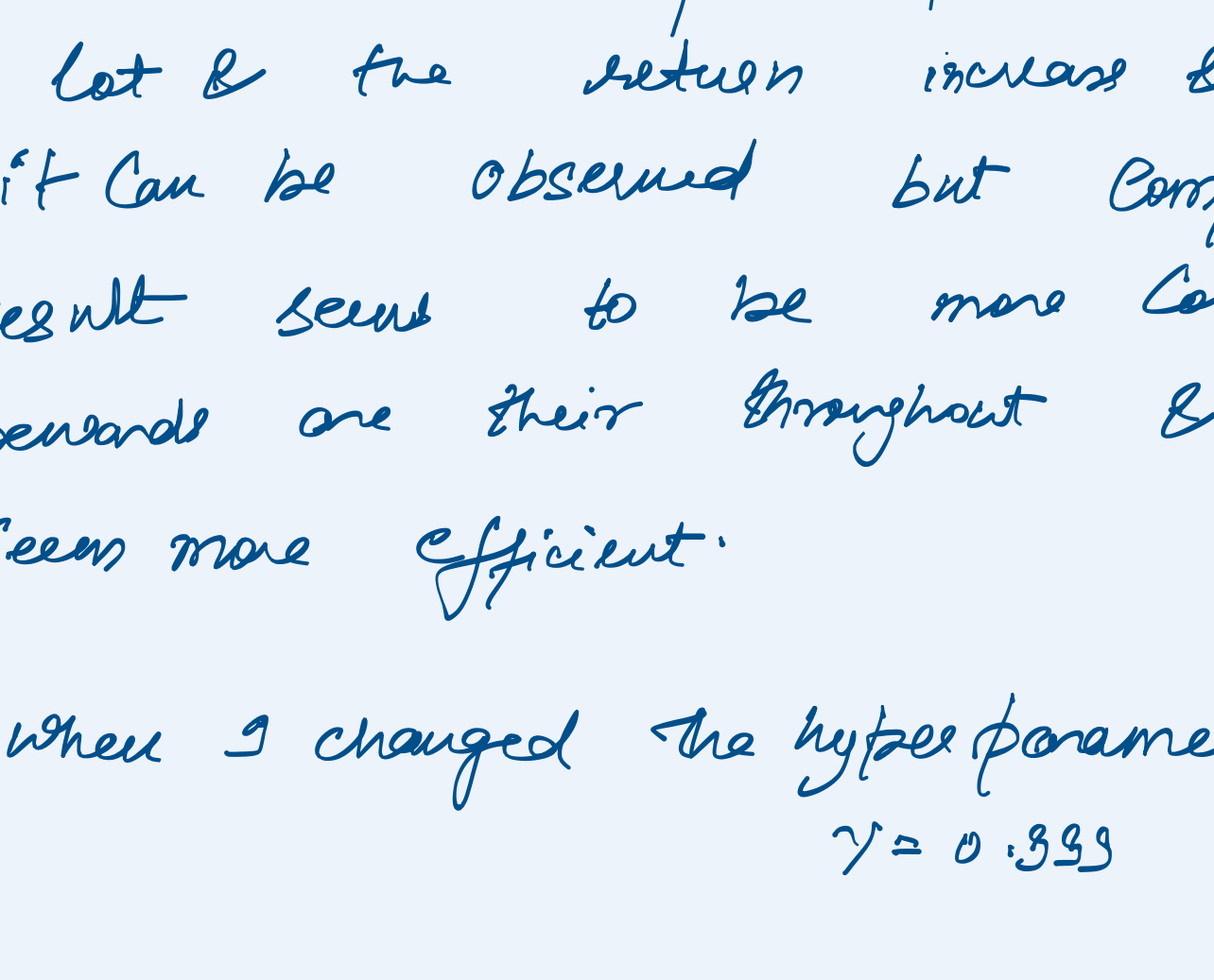
After using the feature (x, y, 1, UP, Down, Left, right) It performs very poorly compared to the State aggregation as the feature vector very simple, In the above the weight vector was large but the update is more efficient.
This shows us we need a good feature design.

Ans 3(e) Normalized state coordinates: feature [x/10, y/10, 1]



In the normalized feature space a curve oscillate a lot & the return increase & decreases as it can be observed but compared to 3(d) result seems to be more consistent & rewards are their throughout & learning seems more efficient.

When I changed the hyperparameter $\gamma = 0.999$ much better learning curve is observed.



When distance included in feature [d, x, y, 1]

In this value seems to vary more compared to the above graph but the returns are better