Q0.

Yes. Whether anonymously or not is okay.

Q1.

Monte-Carlo method will collect reward at each timestep and calculate the return at each timestep backwards then update every state it encountered along the episode. Thus, the learning target G will only be calculated from reward at each timestep but not involve the estimation of next state S'.

I think it's because of two reasons. Firstly Monte-Carlo needs to wait until an episode is completed to update all the state the agent encountered, however sometimes it's not very efficient to do it this way, because we are not sure of how long will the episode be (like in the montaincar case), so we'd better start from TD(0) and increase the step number instead of starting from Monte-Carlo and decrease the step number to find the optimal update method (optimal step number n). Secondly Monte-Carlo introduces large variance to the model, and as I observed through this exercise, improper features in value estimation or inefficient aggregation will further increase the variance, so in order to keep the variance as small as we can, TD method should be preferred over MC.

If we use MC to train a mountaincar agent, then we have to wait until the agent complete an episode to do the update, and that will cause we waste too much time on first several episodes, since it learns nothing when doing transitions. Also it will enlarge the variance of the training process, and thus make the performance unstable.

Q2.

(a)

Input : a differentiable action-value function parameterization q: S x A x R^d ⟶ R
Algorithm parameters: step size α > 0; small ε > 0
Initialize value-function weights **w** arbitrarily (e.g., **w** =0)
Loop for each episode:
    S = initial state and action of episode (e.g. e-greedy)
    Loop for each step of episode:
        Choose action A of state S in terms of policy π(e.g. e-greedy)
        Take action A, observe R, S'
        If S' is terminal:
            w += α[R – q(S,A,w)]∇q(S,A,w)
            Go to next episode
        Else:
            w += α(R + $\gamma \sum_{A' \in A} \pi(A'|S')$ q(S',A',w)– q(S',A',w))∇q(S,A,w)
            S = S'

(b)

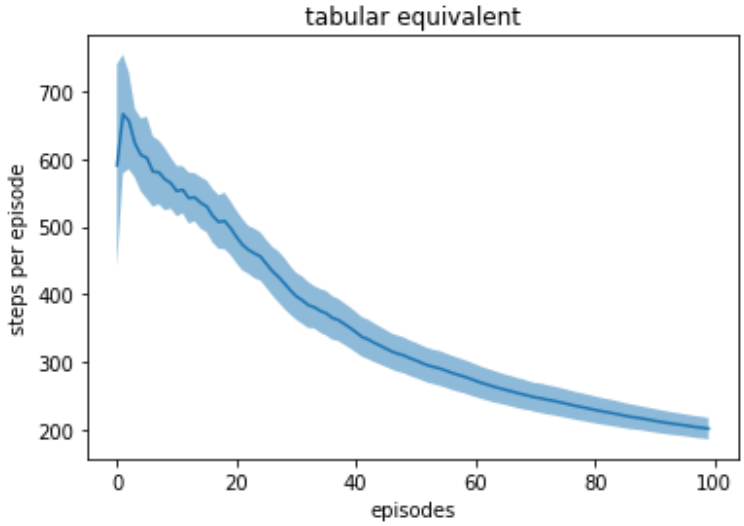
In q-learning, the learning target will use the max state-action value of the next state S' regardless of the policy. Secondly because the learning target and behavior policy won't pick the same state-action pair, so there won't be a ' **A = A'** ' and the next action will be picked by the policy, like the expected SARSA one above.

Q3.

(a)

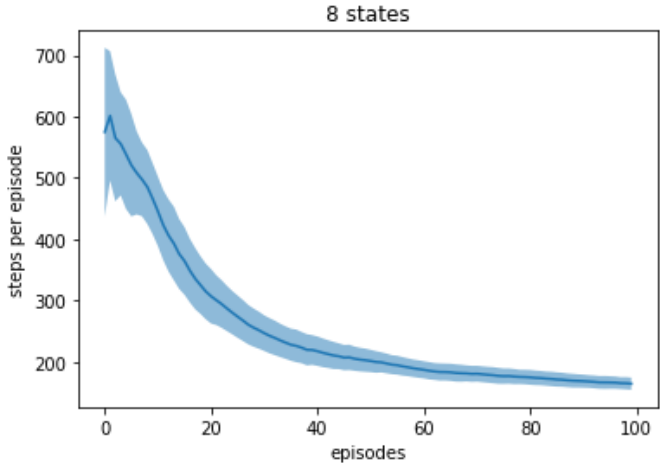
I designed different weight vectors for each aggregation method. In general, I used a weight vector which has the same length as the number of aggregations. Since I use a linear function estimation and one-hot encoding, the gradient of that particular weight is simply the feature corresponding to it.

(b)

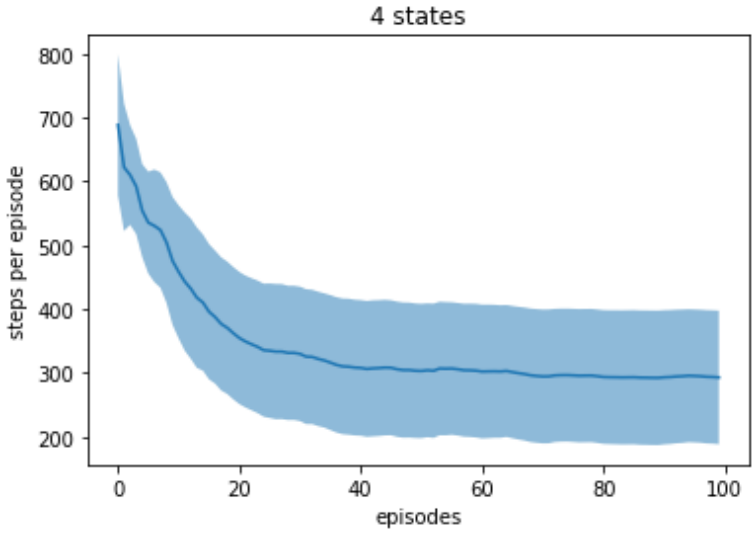


(c)

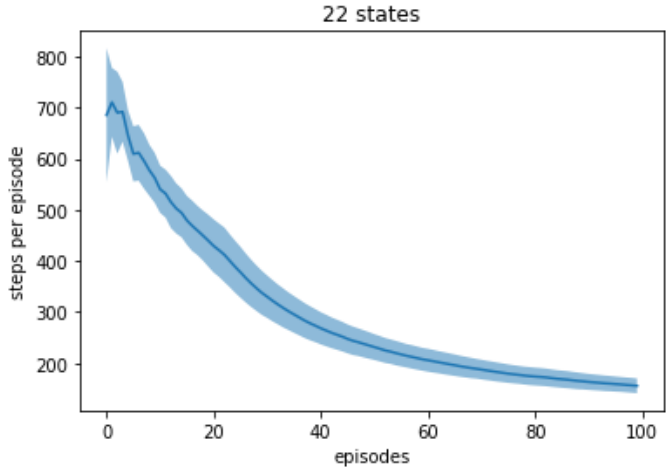
1: I divided the map into 8 states, four rooms as the first four, and four gateways as the last four.



2: I made the states more abstracted, by aggregating all states into 4 states in total. The left two rooms as the first, right two rooms as the second, the two gateways in the middle as the third and fourth.



3: I made the states less abstracted, by dividing each room into 5 states (4 smaller room and 1 gateway), then 2 gateways in the middle (there's an overlapping but I think the agent will handle it well)

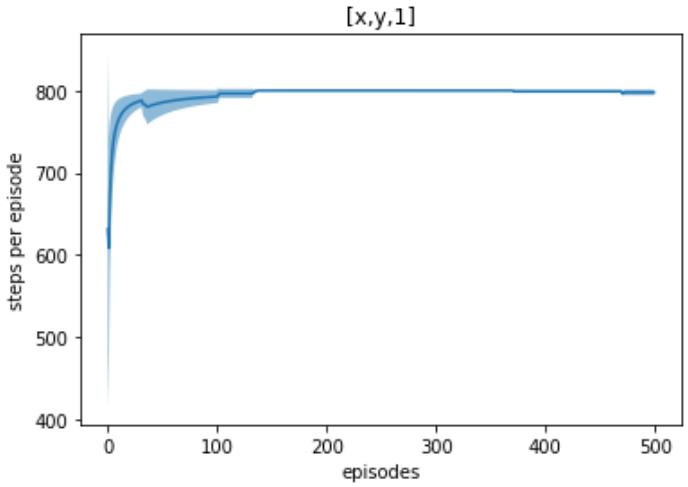


I found that those ones with large aggregations decrease earlier than smaller ones. Although it has only run for 100 episodes, we can still observe it from the comparison between tabular-equivalent aggregation and 4 states aggregation. It makes sense to me, because with less state quantities one can update the state-action pairs more efficiently while in a world with more states the agent will take longer time to explore them all. However, I believe that will more states number, the model can achieve better performance, if given enough length. Though within 100 episode there isn't a huge difference over performance, but in the long run the tabular-equivalent model will outperform all other ones. In a word, choosing appropriate state aggregation could boost the training process without undermine the performance too much.

(d)

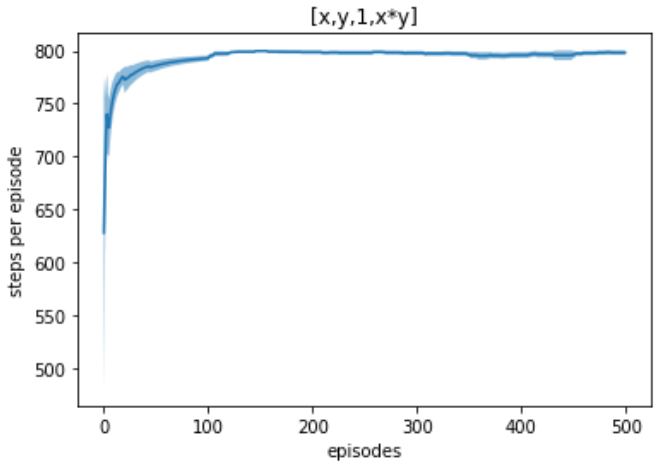
The constant is necessary because it serves like the bias term in classical regression: if there isn't a bias term then the prediction function will always across (0,0) point no mater what update it has taken, and thus the irreducible error will be large.

At first, I used a 1 by 7 vector which is [x, y, 1, 0, 0, 0, 0] (last four digits as action encoding of 'up', 'down', 'left', 'right') as feature vector. However, it didn't work on all the cases, so I turned it into a 4 by 3 one, in which each rows represent the weights on that action.

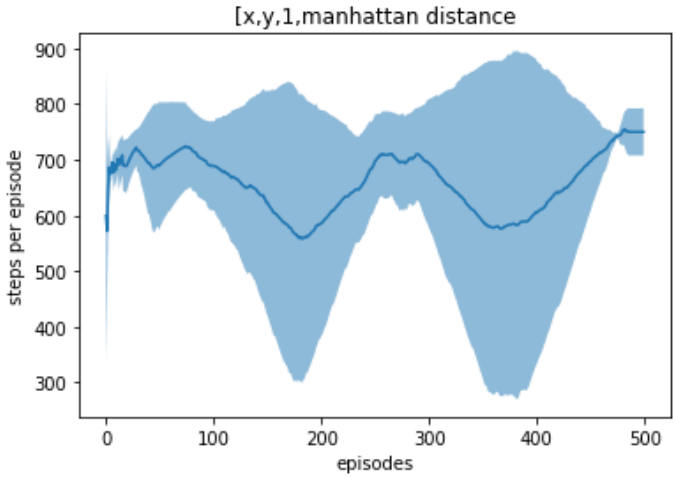


It basically learns nothing, I think it's because the number of features is not enough for estimate the state-action value. Like in (b) and (c), we at least have feature number equals to the total number of states, but in this case we only have 12 weights for 121x4 state-action pairs.

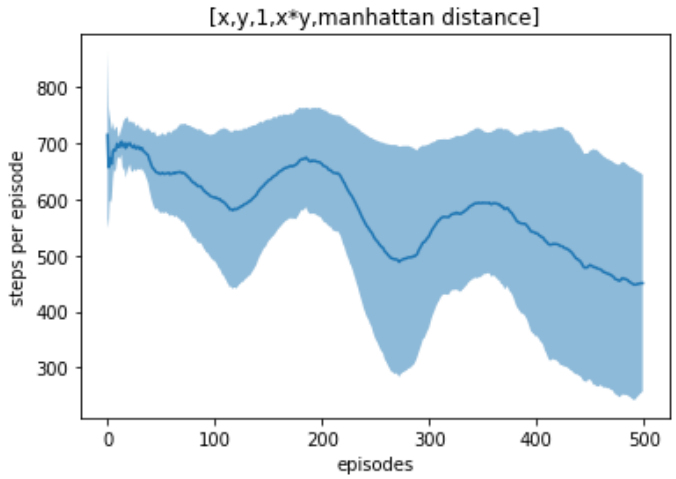
(e)



Naturally, I firstly tried to add a polynomial term, while it shows nothing like improvement.

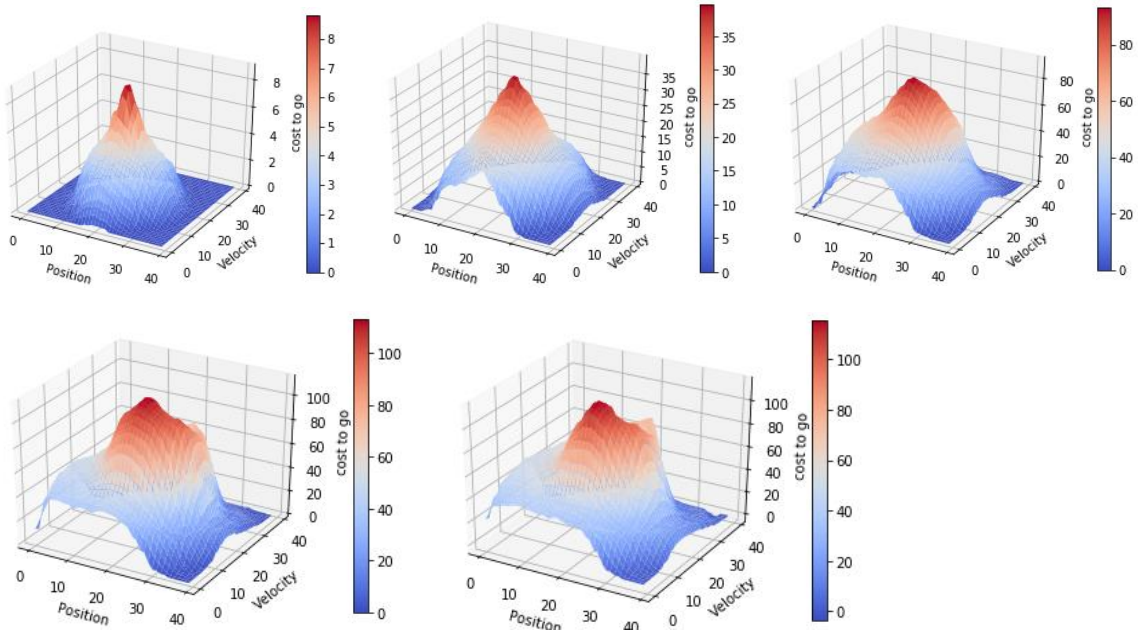


As the second try, I replace the polynomial term with a Manhattan distance element, which equals to abs(11) + abs(11-y). It does learn a bit, but not too much, and obviously the variance is huge.

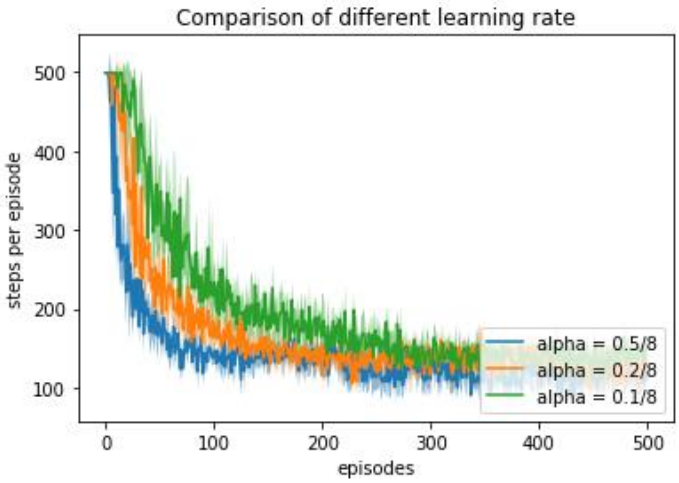


For the third try, I combined polynomial and Manhattan distance, and it appears that although polynomial term doesn't enhance the training process, when it's combined with another feature tog ether it does enhance the performance. That's the surprising result I found.

Q4.



Episode 1000 Episode 12 Episode 9000



Q5.

(a)

$\overline{VE}(w) \triangleq \sum_{s \in S} \mu(s)[v_\pi(s) - \hat{v}(s,w)]^2$; $v_\pi(s) \approx R + \gamma \hat{v}(s',w)$

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

$= \sum_{s \in S} \mu(s) E[(R + \gamma \hat{v}(s',w) - \hat{v}(s,w))^2 | S, A \sim b]$ (assuming on-policy).

$= E_\pi[(R + \gamma \hat{v}(s',w) - \hat{v}(s,w))^2]$

$w_{t+1} = w_t - \frac{1}{2} \alpha \nabla (R + \gamma \hat{v}(s',w) - \hat{v}(s,w))^2$

$= w_t + \alpha (R + \gamma \hat{v}(s',w) - \hat{v}(s,w)) \cdot (\nabla \hat{v}(s,w_t) - \gamma \nabla \hat{v}(s', w_t))$

(b)

by replacing $v_\pi(s)$ with $R + \gamma \hat{v}(s',w)$, we are actually minimizing TD error, which is:

$$\overline{TDE}(w) = \sum_{s \in S} \mu(s) E[\delta_t^2 | S_t = s, A_t \sim \pi]$$

where δ_t is TD error $R + \gamma \hat{v}(s',w) - \hat{v}(s,w)$

It's not a good idea, because like tabular TD learning, minimizing TD error will introduce bias, and it doesn't neccesarily make the predicted value converge to true value of the state. Instead, it more likely to be a smoothing over transition of states. (according to P.272 of the book).

(C)

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

$= E[(E_\pi[R + \gamma \hat{v}(s',w) | s] - \hat{v}(s,w))^2]$

using SGD: (assuming on-policy)

$w_{t+1} = w_t + \alpha (E_\pi[R + \gamma \hat{v}(s',w)] - \hat{v}(s,w)) [\nabla \hat{v}(s,w) - \nabla E_\pi[R + \gamma \hat{v}(s',w)]]$

$= w_t + \alpha [E_\pi[R + \gamma \hat{v}(s',w)] - \hat{v}(s,w)] [\nabla \hat{v}(s,w) - \gamma E_\pi[\nabla \hat{v}(s',w)]]$

(e)

The problem is that this formulation involves production of two expectation, and the production of expectation doesn't necessarily equal to expectation of the production if they are not independent of each other. Because they both rely on the next state S' to be calculated, we cannot get the independent samples for both of these two expectation terms when interacting with real world (since we will only have one S' per transition). However, what we can do is using simulator to simulate two next states for both two terms, and thus they will receive independent samples for update. Another way is that if the S' given S and the policy is deterministic, then the expectation term could be multiplied together, because they only have one single determined outcome. (Got the idea from P272. Of the book)