AE 598 RL - Homework 1: An introduction to reinforcement learning algorithms

Scott Bout[*]
University of Illinois at Urbana-Champaign

This file represents the submission for AE 598 RL taken during the Spring 2023 semester at the University of Illinois at Urbana-Champaign.

I. Introduction

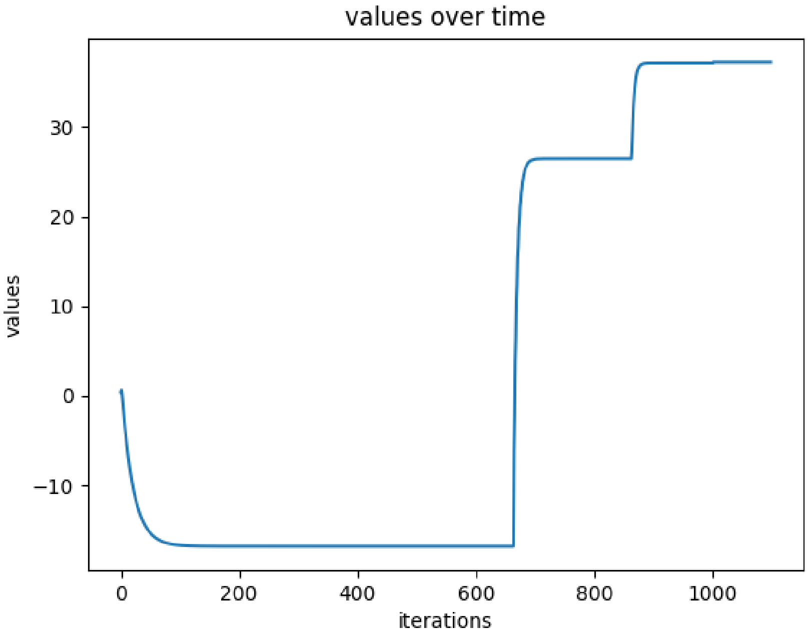
This file aims to very briefly describe a summary of the results, and provide some clarification with respect to the algorithms and code.

II. Results

This section aims to provide the plots required per the readme file from the given repository. Each one will be labelled and at least two sentences will be used to describe it. The order will start with the gridworld results in order from the readme file.

A. Gridworld Results

1. Learning Curves

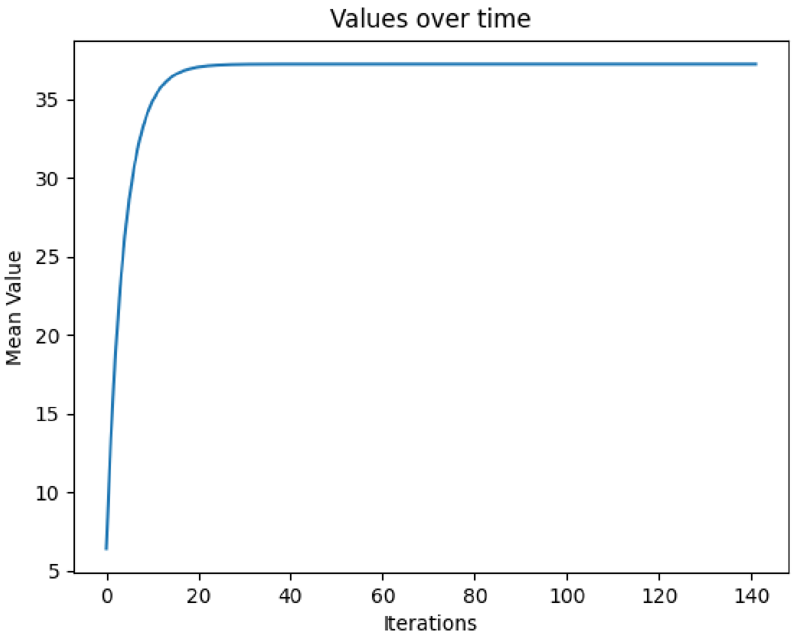


Mean Values Over Time For Policy Iteration

The above represents the learned mean values over the states throughout the learning process of policy iteration. We can see during the near infinite learning transients, this is when a new policy is being applied to the environment

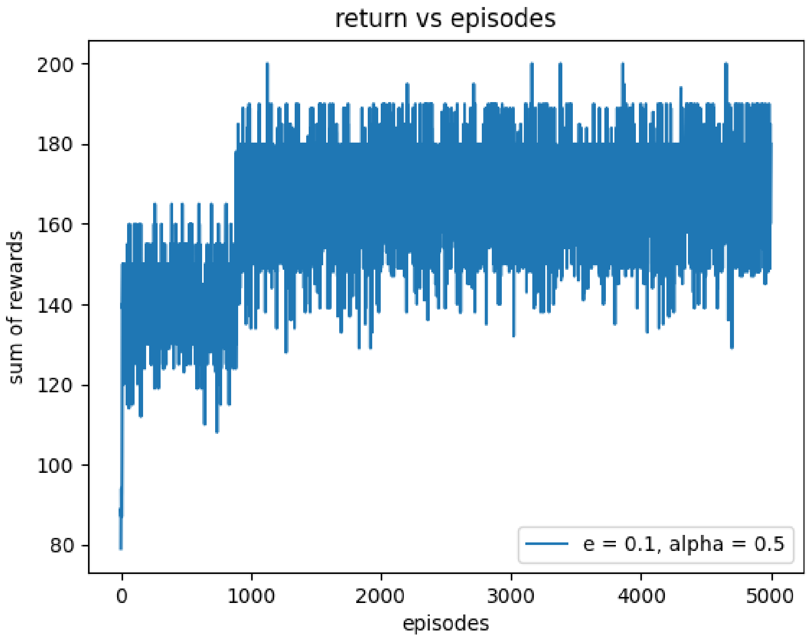
[*] Graduate Research Assistant, Department of Aerospace Engineering

at which the value must now be evaluated at - essentially, the "smooth" curves are the policy evaluation, and the near-non-differentiable points are where policy improvement occurs.



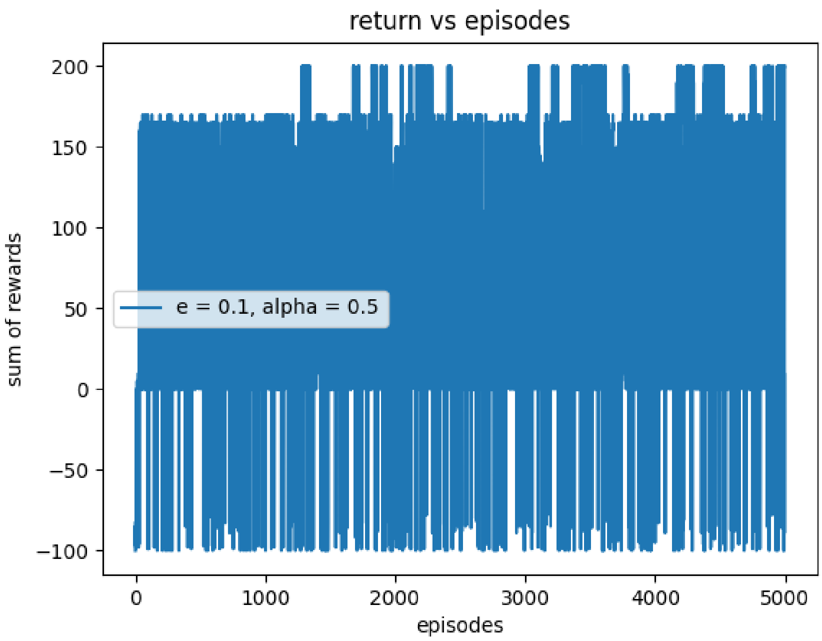
Mean Values Over Time For Value Iteration

As we can see, compared to the learning curve for policy iteration, this curve is much smoother because the policy is always defined as taking the argmax of the value and finding the values based on that so it is all done in one go. That being said, comparing the two we can easily see that they converge to around the same mean values.



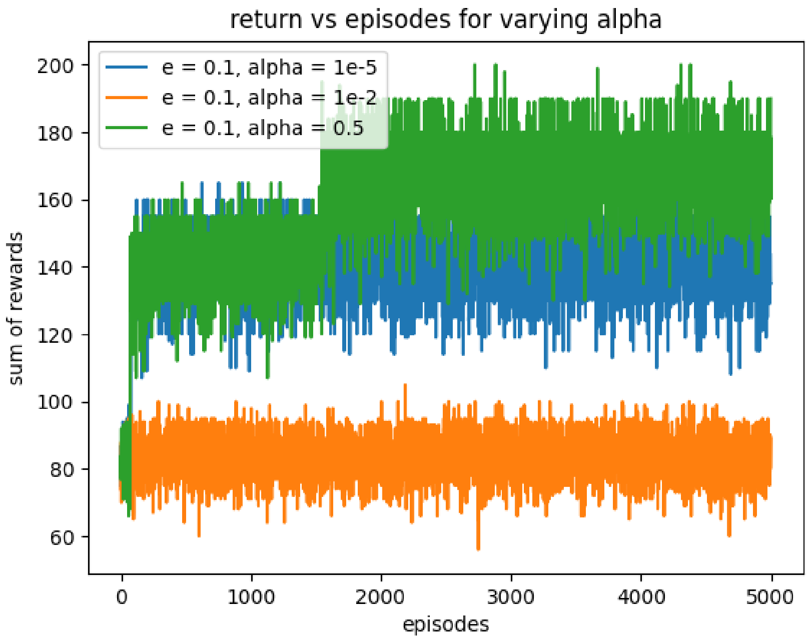
Q-Learning Return During Training Interval

As shown, the return for Q-Learning is clearly improving as it encounters more episodes suggesting that the algorithm is indeed learning. It eventually plateaus.



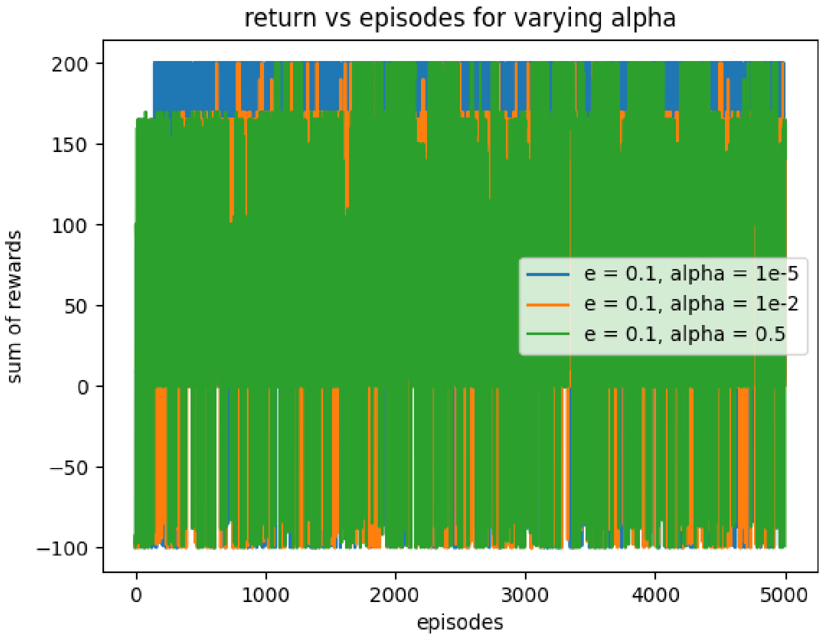
SARSA Return During Training Interval

Not entirely sure why there is such massive oscillations in the returns for the SARSA algorithm.



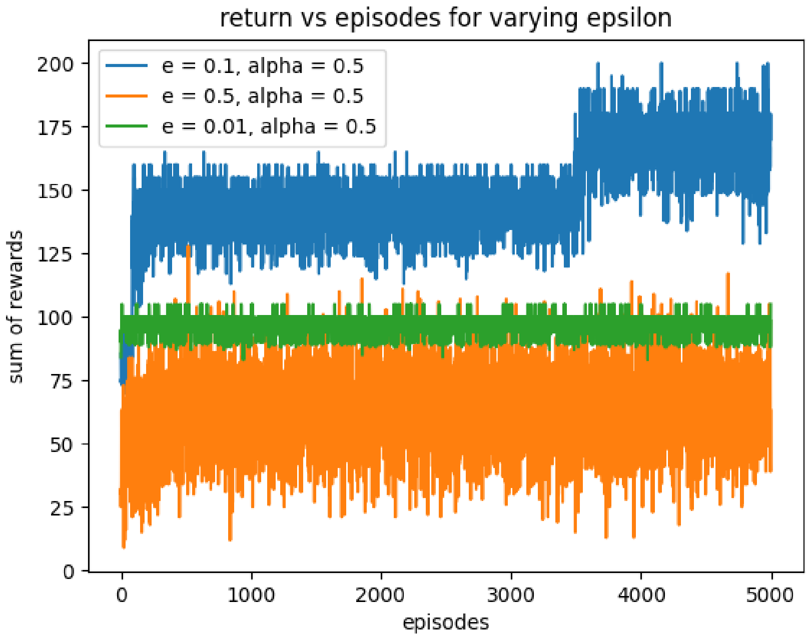
Q-Learning Learning Curve For Varying Alpha

Above are the results for Q-Learning if epsilon is kept constant and the learning rate alpha is varied. It appears that the larger the learning rate the better the performance. Of course, this is a bit nuanced. A large learning rate could converge to a suboptimal solution and technically a very small learning rate would eventually reach a near optimal solution just very slowly. This is indicated on the graph.



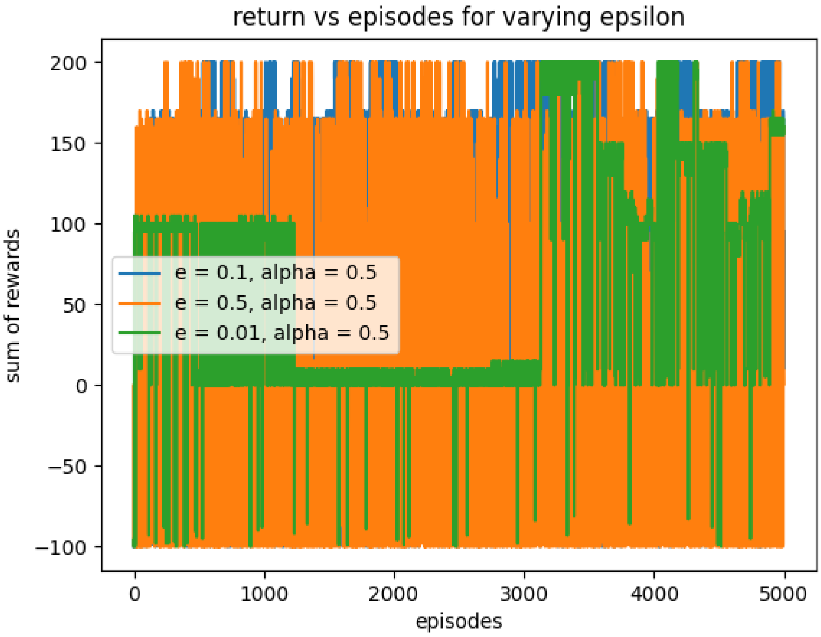
SARSA Learning Curve For Varying Alpha

Once again, not entirely sure why such massive oscillations occur here. Regardless, a similar rule should hold true.



Q-Learning Learning Curve For Varying Epsilon

Here a lower epsilon appears to perform better. Once again however, this is nuanced. Truly if we wanted to completely optimize any exploration based reinforcement learning algorithm (so model free), it is best practice to incorporate some annealing, both in the learning rate and the epsilon in this case. For gridworld, a small epsilon was sufficient.

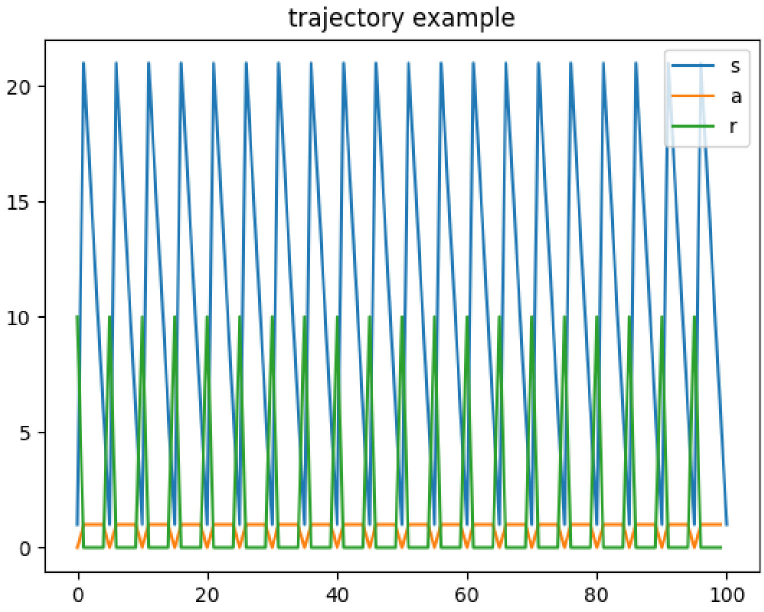


SARSA Learning Curve FOr Varying Epsilon

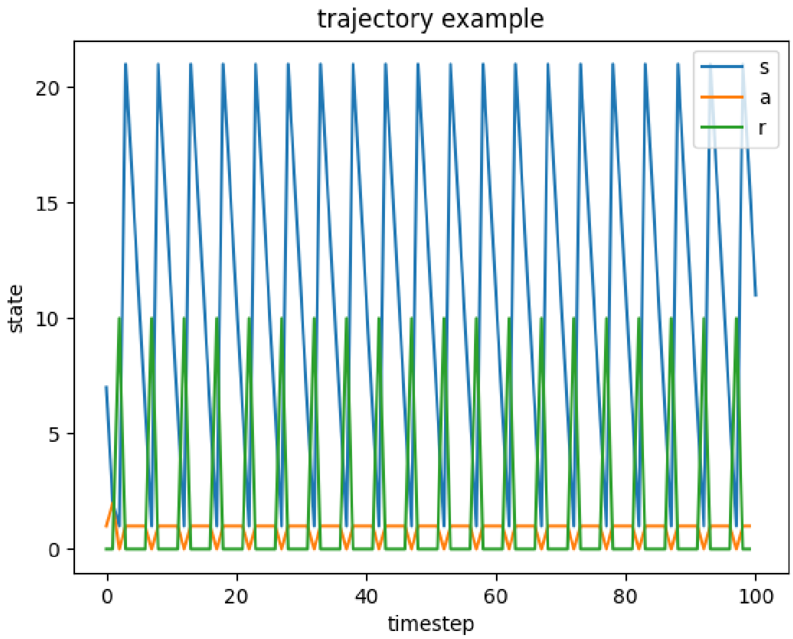
Still very large oscillations. Same general rule should apply. It appears that the SARSA algorithm takes longer to figure out a decent policy.

2. Example Trajectories

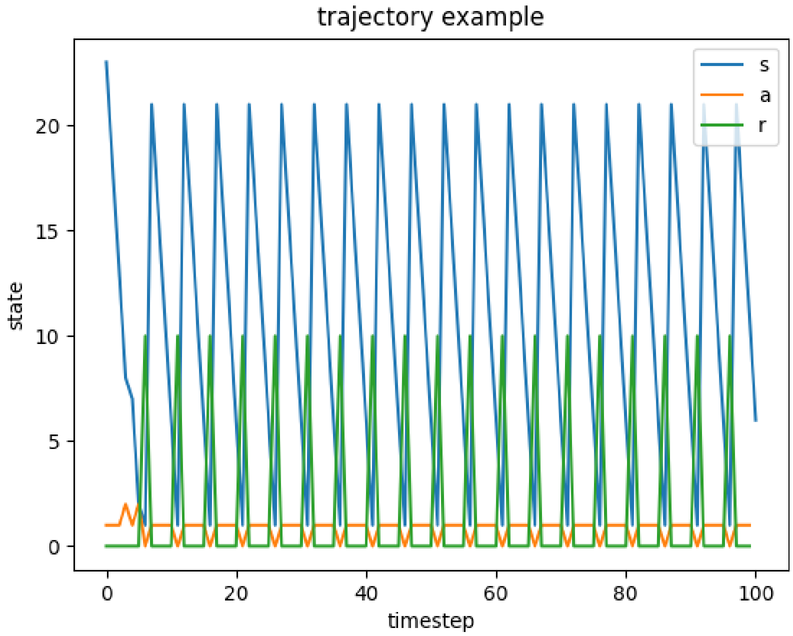
The example trajectories for all trained algorithms are identical in the gridworld setting - not surprising.



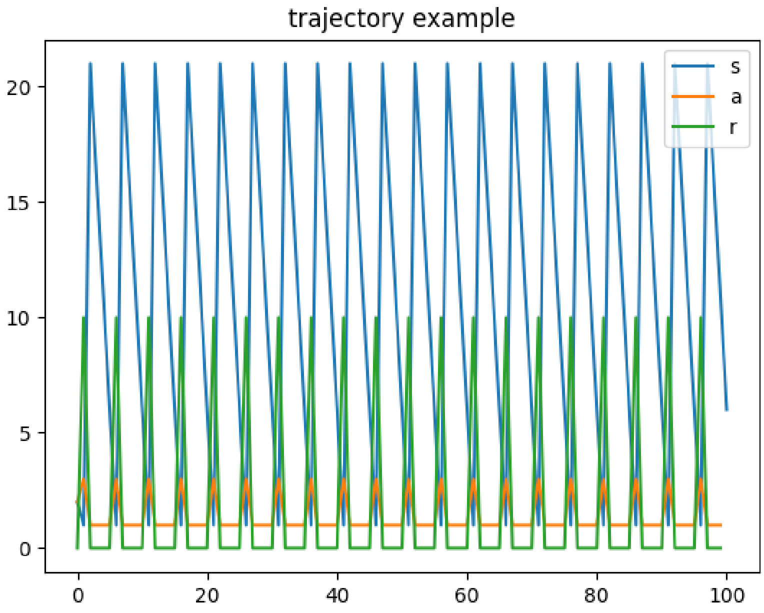
Policy Iteration Example Trajectory



Value Iteration Example Trajectory



Q-Learning Example Trajectory

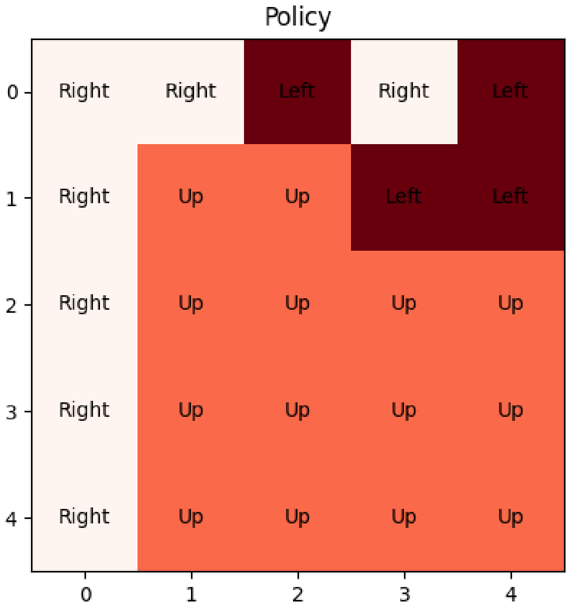


SARSA Example Trajectory

As described earlier, the example trajectory for all is identically. They all try to move to highest reward state 2 where they get teleported and continue the pattern.

3. Policy

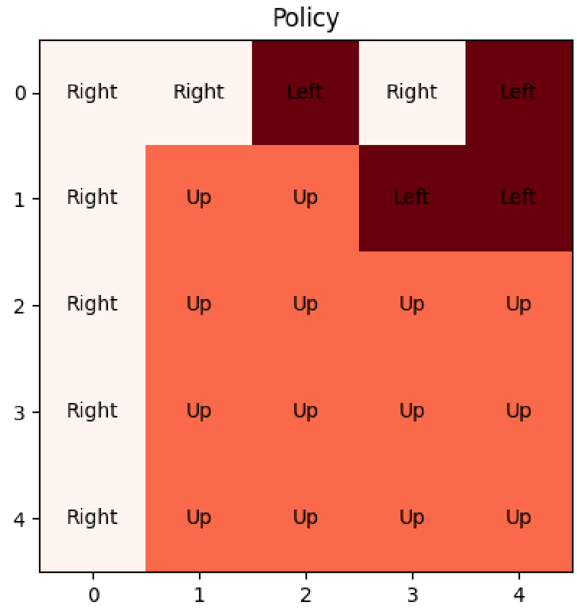
Below are plots for the trained or learned policy under each algorithm.



	0	1	2	3	4
0	Right	Right	Left	Right	Left
1	Right	Up	Up	Left	Left
2	Right	Up	Up	Up	Up
3	Right	Up	Up	Up	Up
4	Right	Up	Up	Up	Up

Policy Iteration Learned Policy

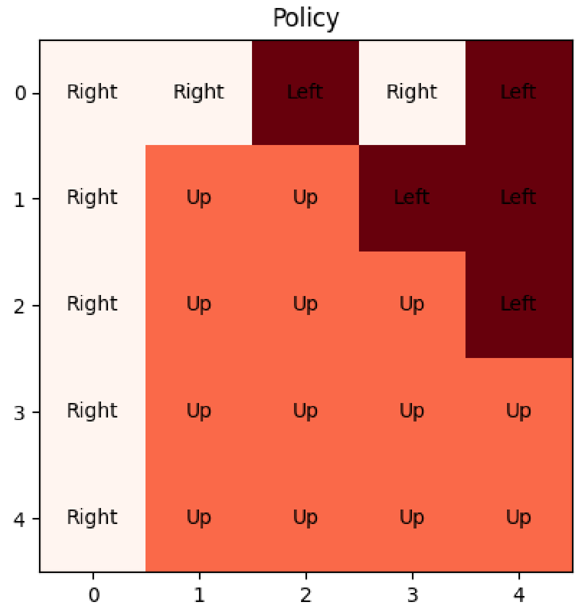
Under policy iteration we see that it does indeed attempt to reach optimal state, with perhaps a slightly strange result at state (0, 3) aka state 3 where it may be better to go left.



	0	1	2	3	4
0	Right	Right	Left	Right	Left
1	Right	Up	Up	Left	Left
2	Right	Up	Up	Up	Up
3	Right	Up	Up	Up	Up
4	Right	Up	Up	Up	Up

Value Iteration Learned Policy

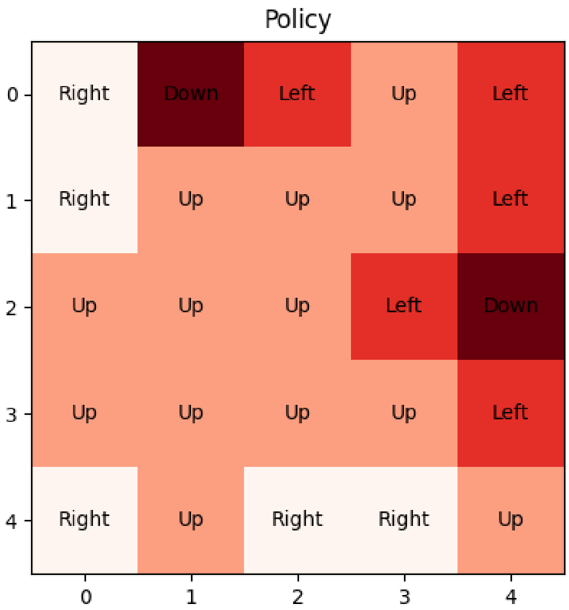
Unsurprisingly this is identical to the one learned through policy iteration. These are dynamic programming methods after all, they should converge to the same solution and that solution should be optimal.



	0	1	2	3	4
0	Right	Right	Left	Right	Left
1	Right	Up	Up	Left	Left
2	Right	Up	Up	Up	Left
3	Right	Up	Up	Up	Up
4	Right	Up	Up	Up	Up

Q-Learning Learned Policy

Here there is likely to be some variation between the us students. Depending on the degree of exploration dictated by the epsilon parameter, some may be optimal, some may not. In this case, the solution converged to the same optimal solution as the dynamic programming methods applied earlier.



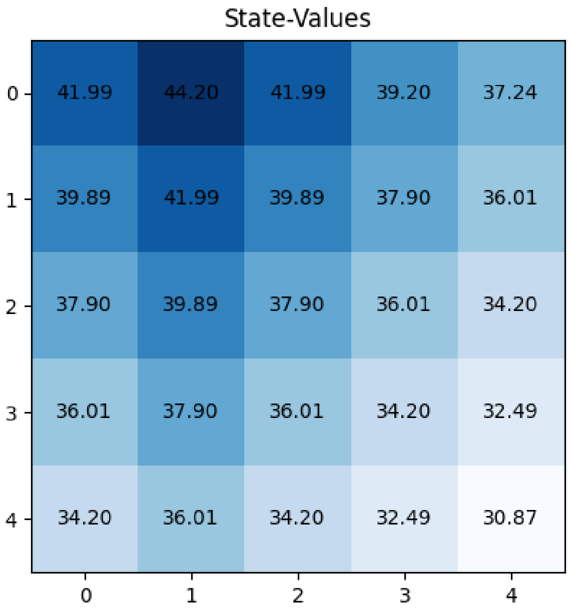
	0	1	2	3	4
0	Right	Down	Left	Up	Left
1	Right	Up	Up	Up	Left
2	Up	Up	Up	Left	Down
3	Up	Up	Up	Up	Left
4	Right	Up	Right	Right	Up

SARSA Learned Policy

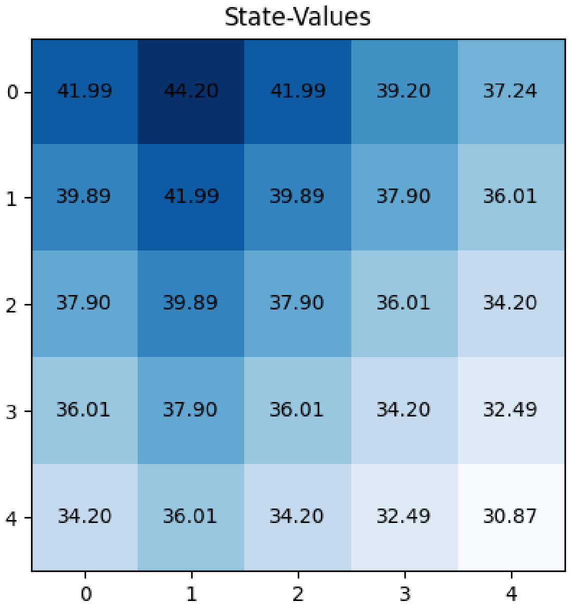
We see the interesting result here where SARSA has converged to a different policy than the others. It includes some strange behaviour including a couple "down"s. That being said, as we saw with the trajectory, it still figures out how to get to the same state.

4. State-Value and State-Action Values

We will first cover the state-value, which is easily described for the dynammic programming methods before moving on to the TD(0) learned state-values based on the Q-Learning and SARSA trained algorithms.

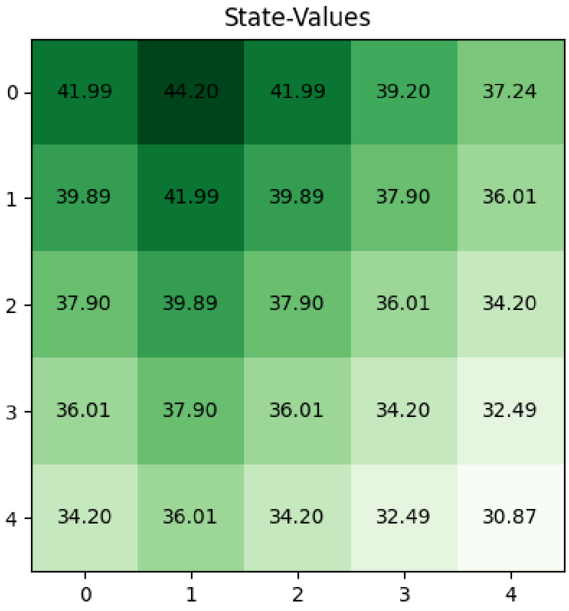


Policy Iteration State-Values



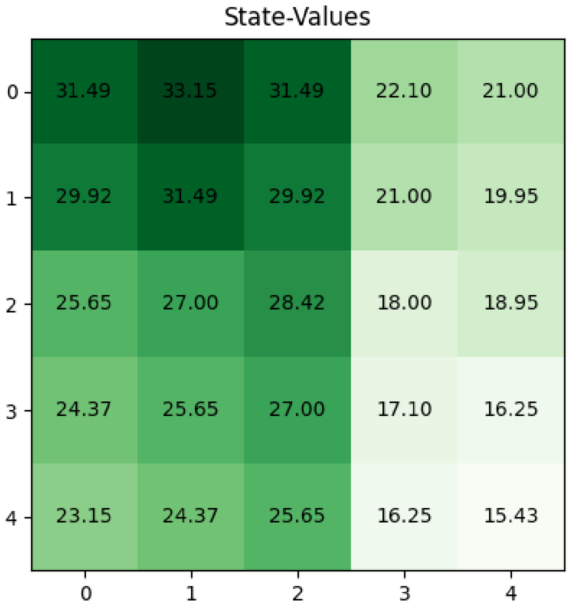
Value Iteration State-Values

As we can see, and unsurprisingly, the state-values for the dynammic programming methods are identical. This is a good check that the algorithms are indeed working.



Q-Learning TD(0) Learned State-Values

Once again the Q Learning Algorithm is identical to the dynamic programming algorithms.



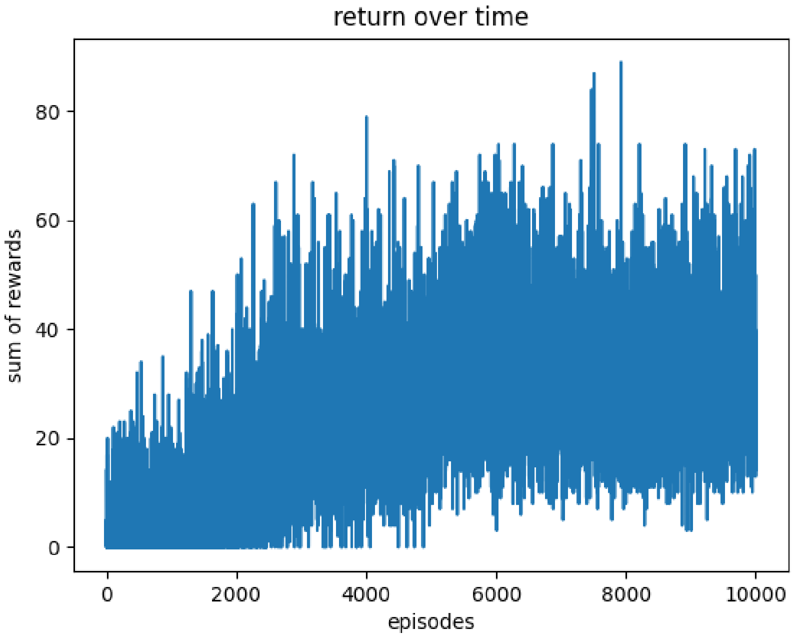
SARSA TD(0) Learned State-Values

We can clearly see here that SARSA produces a different state value plot than the others, but notice that it does still appear to converge the same state and follows similar shading.

B. Pendulum Results

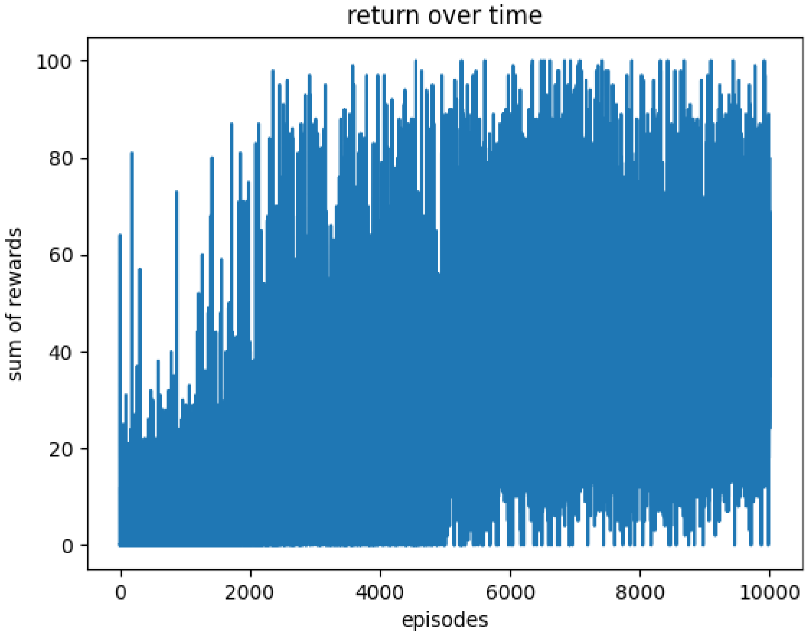
We will start with the learning curves.

1. Learning Curves



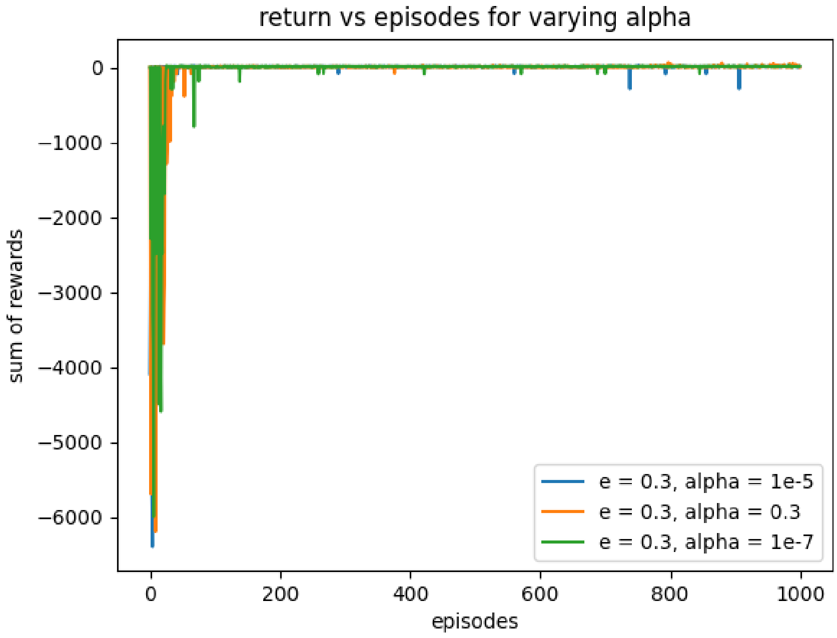
Q-Learning Learning Curve

We can clearly see improvement suggesting the algorithm is indeed learning.

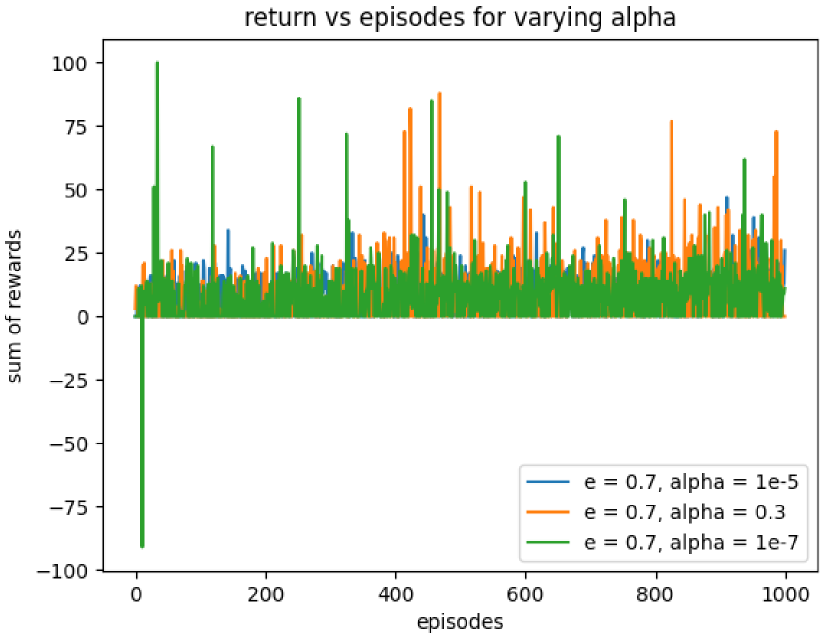


SARSA Learning Curve

While there is certainly significantly more oscillation, there is a clear trend that the algorithm is improving.

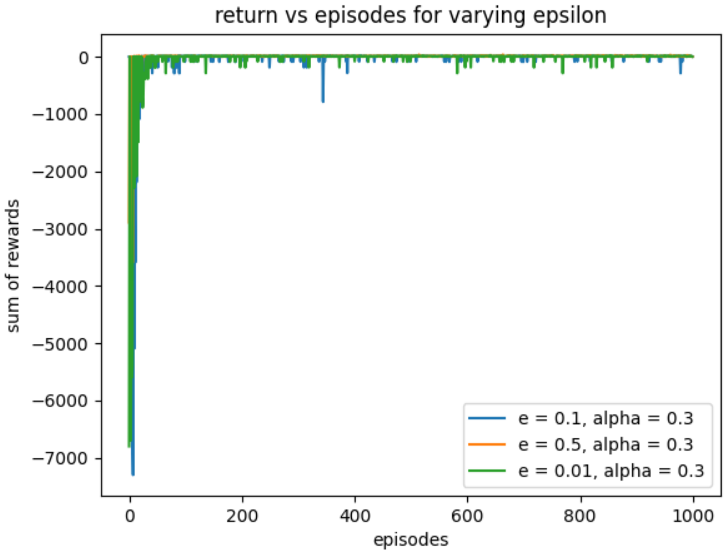


Q-Learning Learning Curve with Varying Alpha

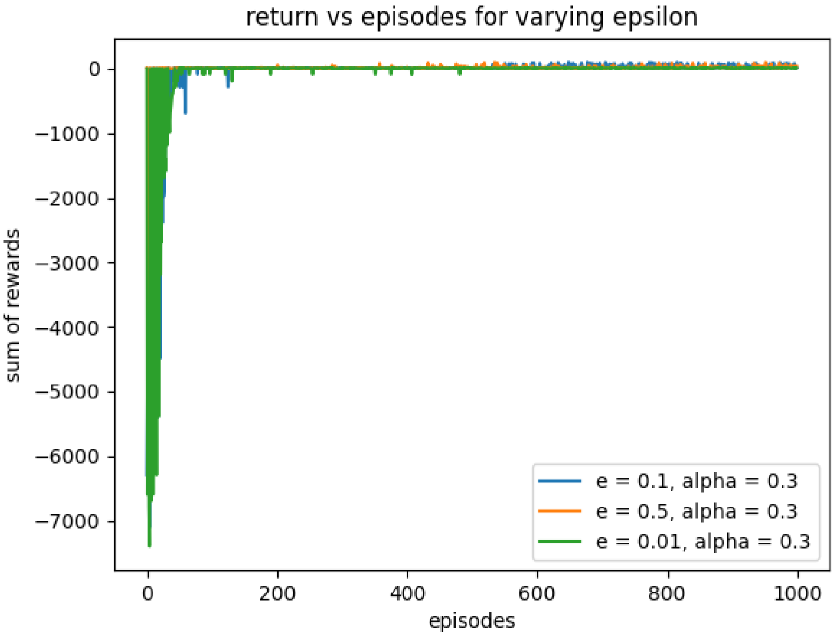


SARSA Learning Curve with Varying Alpha

As can be seen by both plots of Q-Learning and SARSA, the learning rate does not appear to have much of an effect. From these plots it certainly seems that epsilon is the dominating factor, which can be seen by the accidental change in epsilon between the two



Q-Learning Learning Curve with Varying Epsilon

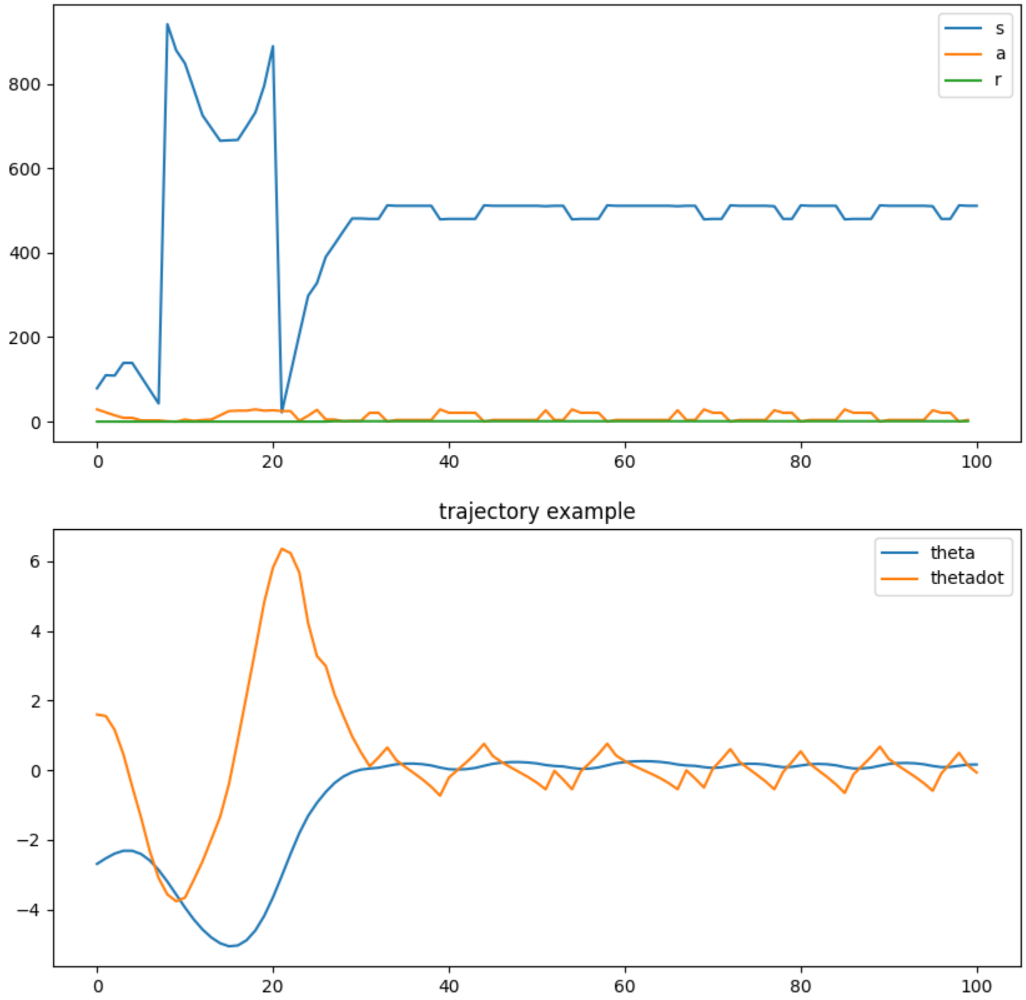


SARSA Learning Curve with Varying Epsilon

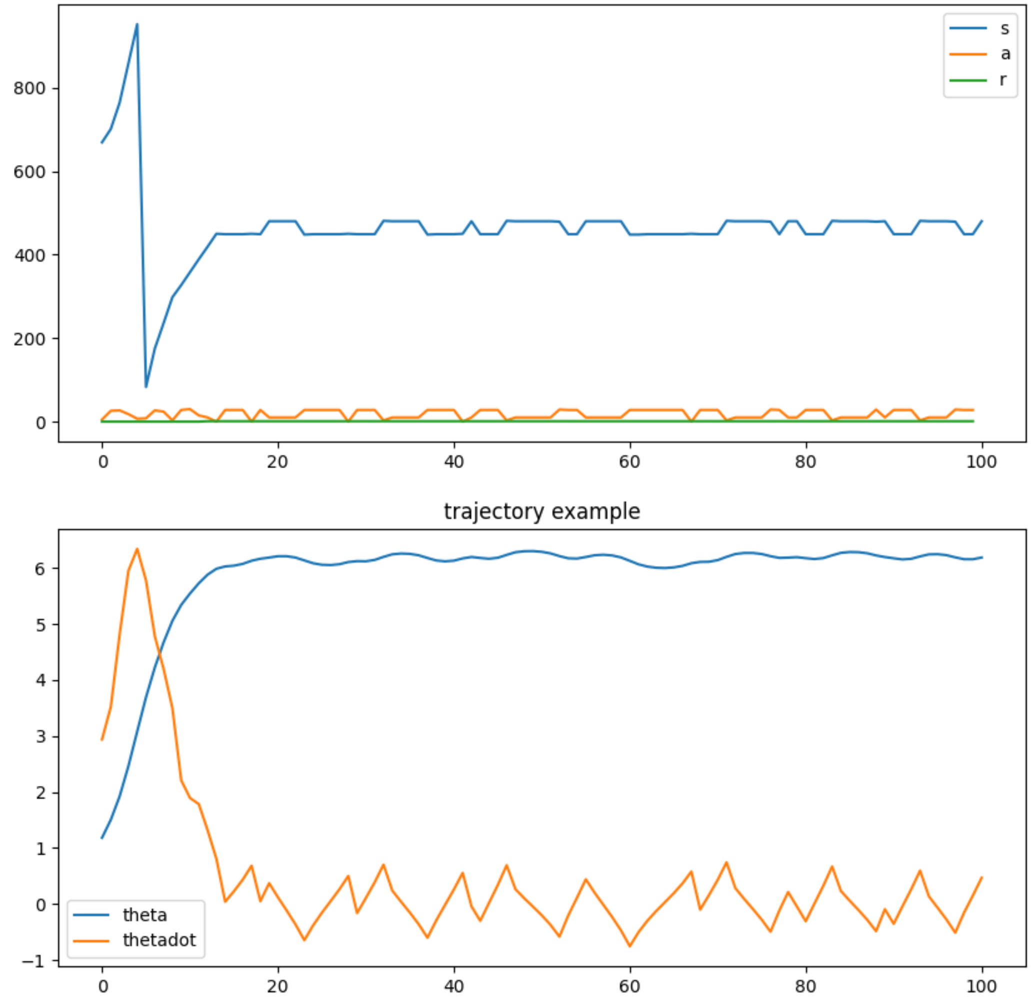
So a small epsilon clearly does not have much of an affect if it remains small, however, and it is a bit difficult to see, increasing epsilon to a larger value like 0.5 has a major effect, notably that large beginning negative return does not occur with a larger epsilon.

2. Trajectory Examples

A note before this section starts. I am not sure if these are correct, but it does appear that the pendulum stabilizes something.



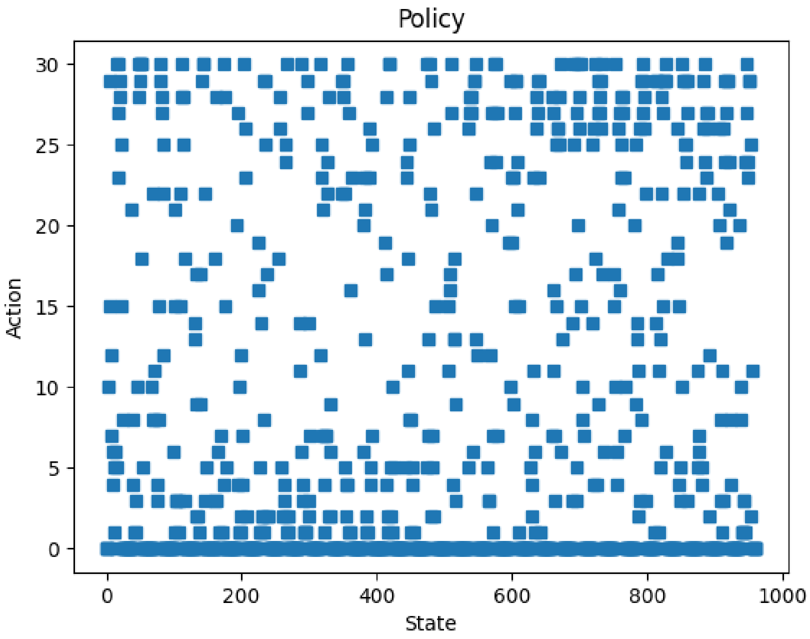
Q-Learning Learned Trajectory



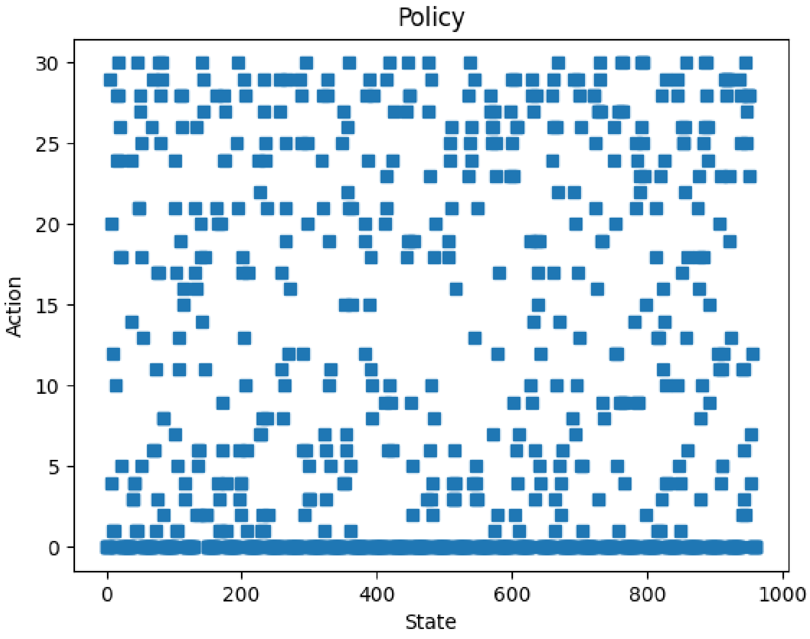
SARSA Learned Trajectory

So I am not sure if this is the correct answer but it certainly seems that SARSA stabilizes the pendulum quite well. This was achieved by taking advantage of epsilon annealing. Every 1000 episodes, epsilon decreased by a decent amount, in this case by -0.1. Same was applied to Q-Learning which is showing a more predictable result.

3. Policy



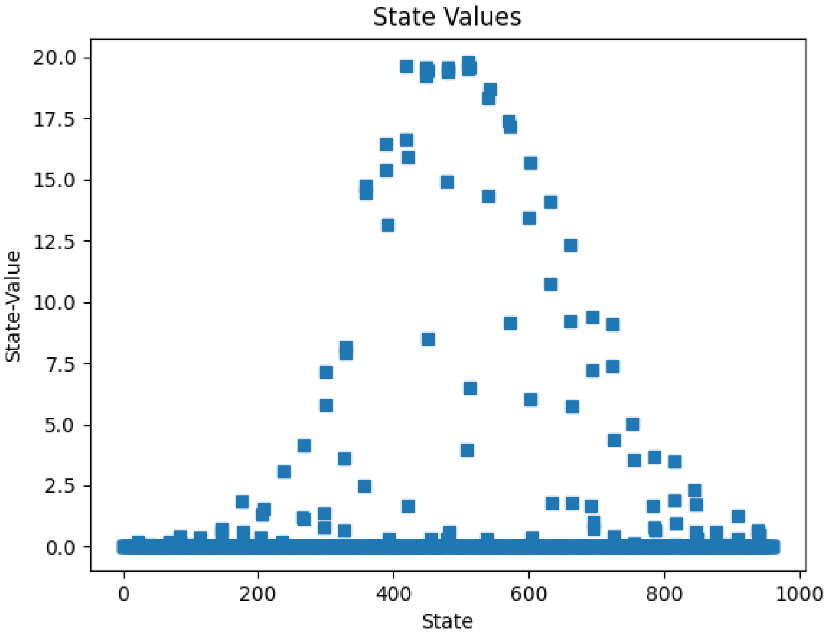
Q Learning Learned Policy



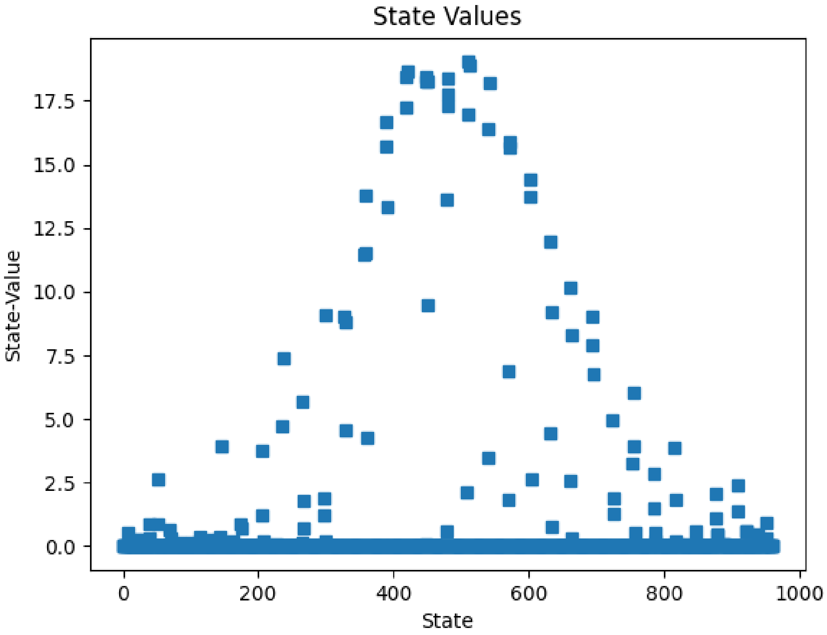
SARSA Learned Policy

This looks pretty random, it is difficult to dissect without taking a deeper dive into the environment what exactly this means but I do not see any clear patterns.

4. State-Values



Q-Learning Trained TD(0) Learned State-Values



SARSA Trained TD(0) Learned State-Values

It certainly appears that the state we are trying to reach is between 400 and 600, and this is backed by the trajectory plot for both Q-Learning and SARSA. We can also see the state-values are very similar for both algorithms.

III. Code and Algorithms

I read the instructions that there should be two files to run and plot the algorithms for gridworld and pendulum and took that literally. What this means is that I have two files that contain all the algorithms for that environment and all the learning and plotting. It is ugly. Also, the main function to be run is not so simple and includes some hidden features, so a quick explanation will be necessary. The main loop is commented to include how modes work. That is easy enough to follow, but in order to plot what the user may want to plot, it is necessary to go into the main function and change the graph parameter to either 0, 1, or 2. 0 produces plots for varying alpha, 1 for varying epsilon. Setting graph to 2 is an attempt at optimal training and will produce the resulting learning curve, trajectory, policy, and state-action values based on argmax over the actions. This applies to both the gridworld and pendulum files. Furthermore, TD(0) is separate, whereby it retrains a hopefully optimal algorithm and returns values based on that, so that is a separate mode and requires looking at the submode comments to apply TD(0) to sarsa or q learning. This is why the state-values described for SARSA seem different than the policy for Gridworld.

Hyperparameters were adjusted to return optimal solutions. I originally thought, from past experience with deep learning, that the learning rate should be quite small but found relatively large learning rates to be quite successful.

The code for the algorithms were generated by following the pseudocode provided in the textbook. While all algorithms have some near-optimal results, I am suspicious primarily of SARSA. Intuitively, I think the idea is to update the Q table via the expected next action from the policy, and I do believe I do this where the policy is epsilon greedy, but the massive oscillations are what concern me.

Finally, some plots may be different in the repository. Most notably, look at

IV. Conclusion

I believe my dynamic programming and Q-Learning algorithms are correct, but am suspicious of my SARSA algorithm. I would also like to apologize in advance for any difficulty reading through the code. Usually I would split up all the algorithms to separate files but in this case I was simply trying to cram everything into two files.