Anthony Gray
EN.605.649.81
Intro to Machine Learning
Project 7

Abstract

Reinforcement learning was performed by car agents on four ASCII racetracks with two algorithms, Value Iteration and SARSA. In general, the agents performed poorly, though comparably, due to a major implementation flaw preventing the convergence of the learning algorithms. The overall lack of success makes a comparison between the algorithms impossible, though what data could be gathered was collected and analyzed. However, the VI algorithm was able to successfully learn one of the tracks, and both algorithms had some success on another. These limited successes indicate that had proper convergence been allowed to occur, both algorithms would have likely resulted in the completion of all four tracks.

Problem Statement and Algorithms

This project attempts to solve the racetrack problem with two different reinforcement learning algorithms. The racetrack problem is a simulated approximation of a car moving along a racetrack. It starts at one of a few starting positions at one end, then accelerates along the track until it reaches the finish line. The car tracks its location on the grid, as well as it's velocity, which is capped at 5 grid units in any direction. The only way it can influence its velocity is by accelerating along each axis up to one grid unit (in whole number increments) at a time in either or both directions. Additionally, each acceleration action has a 20% chance of failure, so the learned model must consider the possibility of losing control at each timestep. Loss of control is significant because there are two possible outcomes for the car hitting the wall of the track. The first, and default option is that the move halts and the car's velocity is set to zero on both axes. The second outcome is that the car crashes. Its velocity is set to zero on both axes, and it is returned to the starting line and attempts the track again from the beginning.

The first algorithm used to train the agent is Value iteration, or VI (1). VI operates by tracking all possible states in the system, and maintaining a reward value and policy, or appropriate action, with each state. In this case, the states are all combinations of possible velocities and locations, and appropriate actions are drawn from all possible accelerations. The state matrix is initialized to a negative value, then the values for the goal states are set to zero. Then, for each state-action pair, the reward for that state-action combination is calculated by adding the reward currently associated with it to the sum of the reward from each subsequent state-action pair, multiplied by a discount factor. Because this is a nondeterministic scenario, the sum of each subsequent state-action pair is also multiplied by the probability of it occurring, which in this case is 0.8. For each state, the action resulting in the highest reward value is assigned as the policy for that state. Iteration over these state-action pairs continues until the changes in the updates falls below a predefined threshold. Once iteration converges, the result is a value matrix of negative numbers that increase toward zero in relation to how close the associated state-action pair will bring the agent to the goal. The policy matrix will also be filled in with the best action to take when the agent finds itself in any given position.

The SARSA algorithm (3) is like VI in that it also seeks to assign a value to each possible state-action pair, though it does not start the learning process with a complete model of the environment. The acronym SARSA stands for "state, action, reward, state, action," and refers to the algorithm's strategy for building that model. When starting to learn by SARSA, the agent will observe its current state, take a random action, and observe (and remember) the associated reward. If the state already has a policy associated with it, the agent will follow that policy, but otherwise it will follow an epsilon greedy strategy. This strategy involves taking the greedy action that maximize reward most of the time, but it will sometimes act randomly based on a tunable probability (equal to the epsilon parameter). This randomness allows the agent some freedom to explore the environment, recording the reward for each state and assigning policies based on that reward as it goes, rather than strictly following the first successful strategy it finds. By mixing random moves and on-policy behavior, a model of state-action pairs and their associated policies is built and expanded, and successful strategies are improved upon until they converge to the optimal strategy.

Both algorithms will be used to train agents in models based on tracks encoded in text files, and to compare the performance of those agents. It is likely that the agents trained with VI will take more training cycles to converge because VI is an exhaustive search of the state space. On the other hand, SARSA will likely use fewer training iterations, but the randomness inherent in the algorithm might cause it to perform more updates per iteration. Both algorithms converge to an optimal strategy, so once trained, they should perform comparably.

Procedure/Tuning

The algorithms were implemented in Python, using a system of three classes. The Agent class, representing the car, contains variables representing its current location and velocity, as well as lists of all possible velocities and acceleration actions it can take. It also has methods for applying acceleration to its velocity and constraining that velocity to never exceed five grid units in any direction. The Track class imports an ASCII representation of a racetrack, which it stores in a grid. It also maintains lists of all starting, ending, and valid positions on that track. It also tracks the position of an associated Agent, and contains a method for moving that agent, as well as a record of the agent's previous positions. All movement and distance calculations are performed using Bresenham's line algorithm (2), which determines which spaces on a grid are crossed by a line between two such spaces. The final class is the MDP (Markov Decision Process) class, which associates with a Track and an Agent, and contains both the state value matrix and the state policy matrix. These matrices represent the combination of every possible agent velocity state and every possible valid location for a given track and are populated with that state's associated reward or policy, respectively. The MDP class also contains a method for running a simulation by applying acceleration actions to the Agent based on those policies.

The VI and SARSA methods implemented in this project populate the MDP value matrix and assign actions to the MDP policy matrix based on those values. The SARSA training will also be accelerated by starting next to the finish line and setting each episode increasingly farther back along the track. This reduces the number of unexplored spaces between the agent and the finish line and allows for efficient and through training. The locations of those starting positions are shown in the appendix. Training both algorithms will take and long time, so the models will be trained and saved in a file using the *pickle* library for later retrieval and simulation. The algorithms will be tested on the T-track (test track), the 5x5 example given in the assignment before learning the L, O and R tracks.

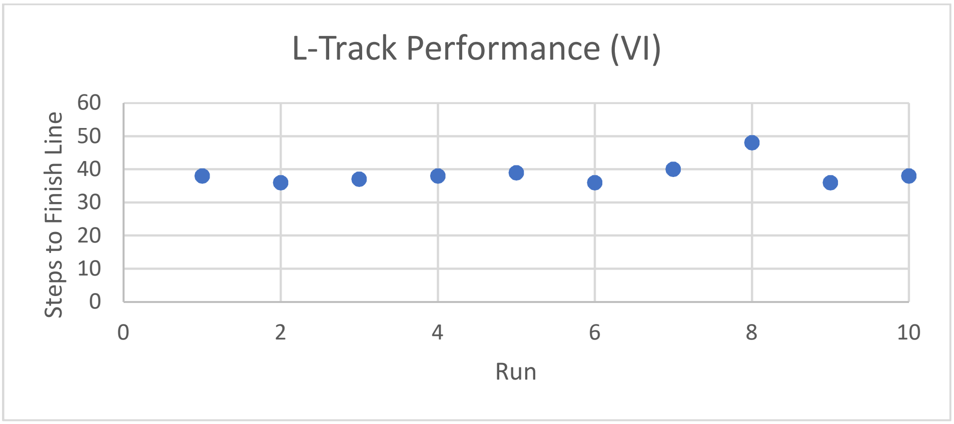
Unfortunately, at least one poor implementation decision lead to the failure of these algorithms to work consistently. A major problem was the tendency for the Q values stored in the value table to grow to huge numbers after a few iterations. The initial reward values and the discount factor, gamma, were set to very low numbers (around |0.0001|), be even so the values would grow be too large to be contained in a single variable. This prevented the algorithms from running to convergence, and the slow convergence rate made debugging correspondingly slow and difficult. Eventually, an iteration cap was set on both algorithms, in order to produce any results at all. Fixing the number of iterations not only prevented the algorithms from working as intended, it also makes it impossible to compare their learning rates. However, what data available was collected, and it is likely that the VI implementation would have worked as intended, provided the bug causing large Q values was solved. It is more difficult to determine if the SARSA implementation was successful, due to its inconsistent performance.

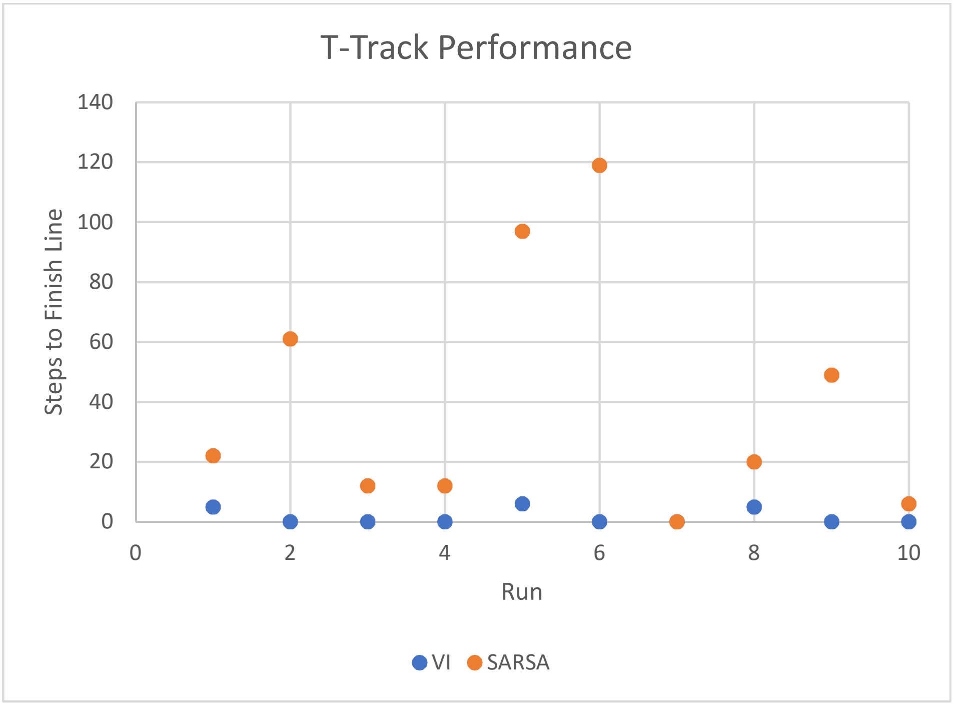
Results

The only two tracks that were successfully completed were the T and L tracks. The VI algorithm allowed the agent to reliably and consistently complete the L track, and both algorithms allowed the agent to complete the T track, though they did so inconsistently.

Neither learning algorithm allowed the agent to complete the O or R tracks. The agent would consistently move upwards to the first turn but would usually get stuck in the corner. It would make the right turn approximately one third of the time, and would rarely make the next downward turn, but it would always get stuck after that, regardless of the algorithm used. The SARSA algorithm also always caused the agent to get stuck on the L track.

When the R track was set to reset the agent at the starting line when it crashed, VI allowed it to go slightly farther than SARSA, but the agent almost never made it past the first right turn with either learning algorithm.





The zero values in the chart below indicate the times in the test runs that the agent got stuck in a corner. It is questionable if SARSA worked on this track at all, considering how small it is. Additionally, while VI allowed the agent to complete the T track in a consistent number of steps, the completion itself was inconsistent.

Discussion

The fact that the VI algorithm allowed the consistent completion of the L track indicates that its implementation was at least minimally functional. The agent reached the finish line every time, with a consistent, if higher than expected number of steps. Its success on the T track must be qualified, because the agent did not always finish, though when the agent finished it did so with a consistent number of steps. Both of these issues are likely due to incomplete convergence. If the model had been allowed to converge, the L track would likely have been completed faster, and the T track would likely have been completed more consistently.

The failure of VI to allow the agent to complete the O and R tracks is also likely a convergence issue. The agent consistently moved in the right direction and would usually get stuck in a turn. A more convergent model probably would have provided more appropriate policies for handling the turns at different velocities. Additionally, allowing for more iterations would have helped provide the necessary number of backups to adequately cover these longer tracks.

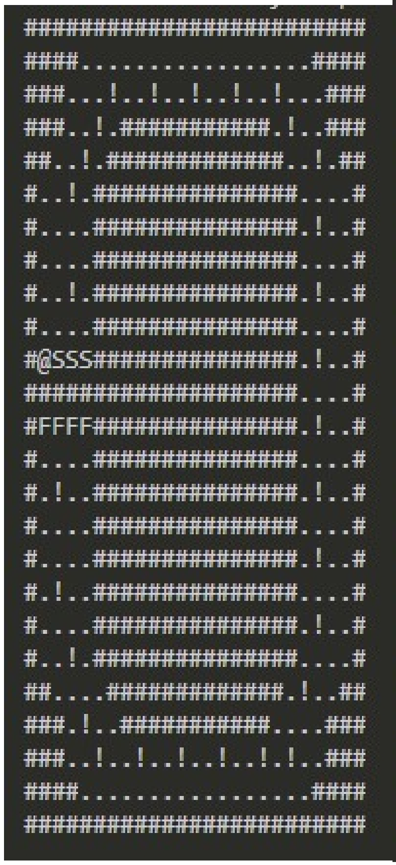
The SARSA implementation did not allow for the consistent completion of any of the main tracks, though it did consistently complete the T track. This also needs to be qualified, as may just be the result of random movement. With only 16 valid locations, step numbers in the multiple tens do not necessarily indicate that learning occurred. Like the agents trained with VI, the SARSA agents also demonstrated a general policy that would have likely improved with further convergence. Both algorithms performed comparably on the O and R tracks, as the agents trained by both algorithms largely reached the same parts of the track with the same frequencies and in the same number of steps. It is possible that a mistake was made in the implementation of SARSA, as its behavior was less consistent than that of VI on the L and T tracks, but the inability to converge was certainly the primary cause of its poor performance.

Summary

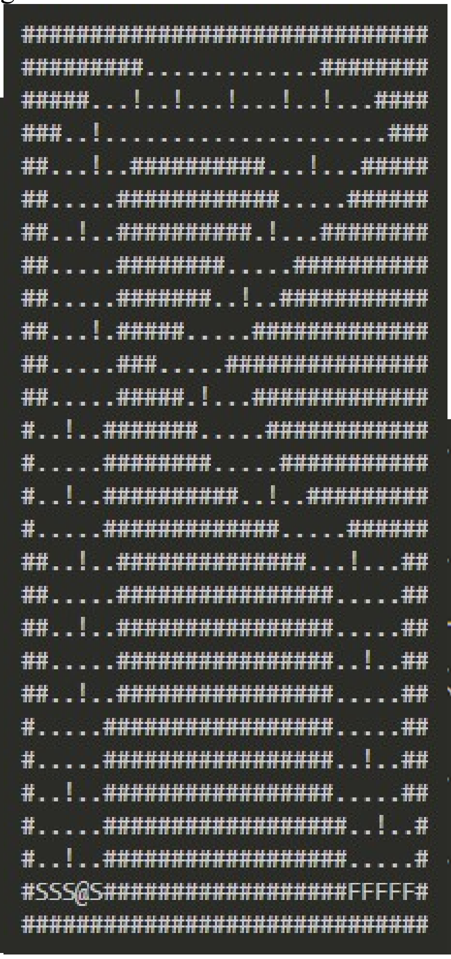
In general, the VI and SARSA implementations performed poorly, likely due to an inability to converge. The implementation bug that caused the Q values to grow too large and crashed the program before convergence was not identified in time, and the incomplete models resulted unsurprisingly poor performance by the agents. The failure of the models to converge also makes it difficult to evaluate what successes were achieved, though the fact that there were any successes at all is promising. The fact that the VI implementation was able to allow the consistent completion of at least one of the tracks indicates that it was working as intended to some extent. The performance of the SARSA implementation was harder to evaluate, as the incomplete model resulted in some random behavior, but it is encouraging to note that in many cases its behavior was like that of VI. This suggests that by resolving the growth of the Q values and allowing for the learning algorithms to fully converge, performance could be improved dramatically.

References
1. Bellman, R. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics*. 6.
2. Bresenham, J.E. (1965) Algorithm for Computer Control of a Digital Plotter, *IBM Systems Journal,* 4-1, pp. 25-30
3. Watkins, C.J.C.H. (1989). Learning from delayed rewards. *PhD Thesis*, University of Cambridge, England

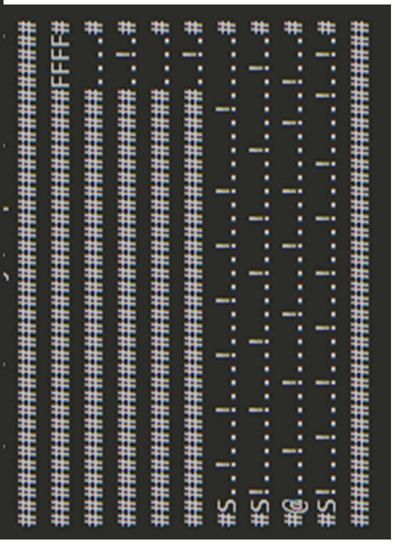
Appendix
! are SARSA episodic starting points
@ is the current position of the agent



```
#########################
####.................####
###...!..!..!..!..!...###
###..!.###########.!..###
##..!.#############..!.##
#..!.###############....#
#....###############.!..#
#....###############....#
#..!.###############.!..#
#....###############....#
#@SSS###############.!..#
####################....#
#FFFF###############.!..#
#....###############....#
#.!..###############.!..#
#....###############....#
#....###############.!..#
#.!..###############....#
#....###############.!..#
#..!.###############....#
##....#############.!..##
###.!..###########....###
###..!..!..!..!..!.!..###
####.................####
#########################
```

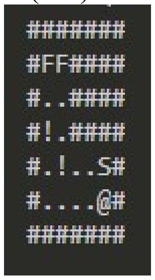


```
##############################
#########.............########
#####...!..!...!...!..!...####
###..!.....................###
##...!..##########...!...#####
##.....############.....######
##..!..##########.!...########
##.....########.....##########
##.....#######..!..###########
##...!.#####.....#############
##.....###.....###############
##.....#####.!...#############
#..!..#######.....############
#.....########.....###########
#..!..##########..!..#########
#.....#############.....######
##..!..##############...!...##
##.....################.....##
##..!..################.....##
##.....################..!..##
##..!..################.....##
#.....#################.....##
#.....#################..!..##
#..!..#################.....##
#.....##################..!..#
#..!..##################.....#
#SSS@S##################FFFFF#
##############################
```



```
#####################################
################################FFFF#
################################....#
################################..!.#
################################....#
################################..!.#
#S..!...!...!...!...!...!.....!.....#
#S!.....!....!....!...!....!.....!..#
#@...!....!....!....!...!....!..!...#
#S!...!...!...!...!...!...!....!..!.#
#####################################
```

T (test) track:



```
#######
#FF####
#..####
#!.####
#.!..S#
#....@#
#######
```