Journal of Machine Learning Research 1 (2000) 1-48 Submitted 2/23; Published 2/23

Reinforcement Learning Strategies to Solve the Racetrack Problem

Brendan Foley BFOLEY3@JH.EDU
Department of Biotechnology
Johns Hopkins University
Baltimore, MD 21202, USA

Editor: Brendan Foley

Abstract

This paper details the creation and use of three reinforcement learning strategies: value iteration, Q-learning, and SARSA (State-Action-Reward-State-Action). We train these three methods on a classic problem known as the "racetrack problem", in which a car attempts to learn to navigate a track in the shortest amount of time possible. To accomplish this, we first build the algorithms, then tune the method-specific hyperparameters. After tuning, we begin several runs to determine the true effectiveness of the three methods. We find that value iteration performs the best across all three methods when looking at the number of test car steps taken.

Keywords: Assignment 5, Reinforcement Learning, Value Iteration, Q-Learning, SARSA

1. Introduction

While machine learning is often broken down into two popularized subsets, supervised and unsupervised learning, there is a third subset: reinforcement learning. Reinforcement learning concerns itself with how an intelligent agent will learn and act given its objective is to maximize the total cumulative reward generated through the course of its actions. This problem is generally modeled as a Markov Decision Process, where actions may be influenced by randomness as well as the agent. The general process is as follows: an agent has a current state, s, takes an action, a, and receives a reward, r, as designated by an outside force (the programmer). This reward then may influence further states and actions down the line.

In this paper, we create, test, and evaluate three different models of reinforcement learning: value iteration, Q-learning, and SARSA (State-Action-Reward-State-Action) across three tracks, the "L-track", "O-track", and "R-track". The O- and R-tracks are larger and implement more turns than the L-track.

We also handle crashes in two different ways. For the L- and O-tracks, we only consider the mild-crash scenario, where we return the car to the track immediately at the space closest to the crash point. For the R-track, we consider a harsh-crash scenario, where we place the car back at the start line, in the starting position closest to the crash, as well as the mild-crash scenario.

We predict that value iteration will have the shortest average steps to finish the race and that this will occur on track L with mild crash conditions. This is because Track L is the

least complex track, with only one turn, and mild conditions not causing a reset to occur. We also expect Q-learning and SARSA to take longer training time compared to value iteration, since the car must run through the track from the start at the beginning of each episode.

2. Experimental Approach

We define several classes to increase the generalization and ease testing: Analysis, Car, Helpers, QLearning, Racetrack, SARSA, and ValueIteration. Analysis is used to perform testing on our models. Car is used to keep track of our agent's current state and implements a step method to take a given action. Helpers defines several methods that are common auxiliary methods to our reinforcement learning classes. These include methods to handle crashes and identify finishing, as well as the ϵ-greedy algorithm. QLearning, SARSA, and ValueIteration all instantiate their respective reinforcement learning algorithms and have train and test methods, which have their implementations explained below. Lastly, Racetrack is used to load the race track files and has fields for the positions of wall, finish, and start locations.

We consider three reinforcement learning models: value iteration, Q-learning, and SARSA (State-Action-Reward-State-Action). Value iteration is a model-based algorithm that is aware of the entirety of the track. We define a maximum number of iterations for learning, or continue until convergence, as defined by a threshold value. In each iteration we iterate over every possible state, as defined by the tuple (y, x, vy, vx), where y is the y-position on the track, x is the x-position on the track, vy is the velocity in the y-direction of the car, and vx is the velocity in the x-direction of the car, where vx and vy take a value from the integer range [-5, 5]. We then iterate through all possible actions, which is the acceleration of the car in the y- and x-directions. The accelerations can take a value from the integer range [-1, 1]. While iterating through the actions, we perform the action, check whether we have finished and/or crashed. If we crashed, we reset in the respective manner. If we finished, we set our new v value to the previous state's v value. Then we calculate the expected value from this state based on the probability of taking the action (as there is a 20% chance of failure to take an action) and update the value table and q-table for the previous state. Lastly, we check for convergence.

In Q-learning, we do not have awareness of the whole track, and thus must change our approach. We loop through a given number of iterations, and for each loop, we define a new state at random and instantiate the car at that point. We then define an inner loop that continues until the loop finds the finish line. We select an action to take based on a ϵ-greedy policy, as well as using a 20% action-failure rate. We, again, check for finishing and for crashing, updating the position as appropriate. Our update rule for Q-learning is: $Q(s, a) = Q(s, a) + \alpha(r + \gamma \max_a Q(s', a))$, where α is the learning rate, γ is the discount factor, Q is our Q-table, s is the current state, a is the current action, and s' is the next state. After the inner loop concludes (we find the finish), we utilize a decay factor to alter ϵ and α to increase our speed to finish.

Lastly, we have our SARSA implementation. When training SARSA, we are not aware of the entire track, just as in Q-learning. We define an outer loop that iterates through a given number of epsidoes. At the beginning of each iteration, we define a starting point

at random and instantiate a car at that point and select an action to take based on an ϵ-greedy algorithm. We then enter an inner loop that iterates over a given number of iterations. We take an action (again, with a 20% failure rate), and check for a crash, updating the position as specified. Next, we find the next action to take through the ϵ-greedy strategy, this will be a'. Next, we update our Q-table by the following update rule: $Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$. The variables are defined the same as in Q-learning, and we have the addition of a' which is the next action taken.
When collecting run time statistics, we iterate through each method-track combination ten times in order to generate meaningful insights. The values of the hyperparameters used were found through repeated trials and observing whether the test found the finish line, as well as the number of steps taken.

3. Results

The values that we found to work the best overall out of the ranges tested varied by reinforcement learning method chosen. For Value iteration, we used a discount factor of .9 and a threshold value of .1.
For Q-learning, we chose a discount factor of .9, an ϵ of .6, a decay of .9999, an α of .8, and a max iteration of 10000.
Lastly, for SARSA (State-Action-Reward-State-Action), we used 10000 episodes, a discount factor of .9, an ϵ of .1, a decay of .99, and an α of .1. For the L- and O-tracks we used 1000 iterations per episode, while we used 10000 iterations per episode for the R-track. This reflects the difficulty of converging to an optimal solution on the R track.
The performance of our algorithms in training is shown in Figures 1-3, while the performance of our testing is reported in Table 1. It is important to pay attention to the y-axis scales in Figures 2 and 3, as the number of iterations and the cumulative reward, respectively, varies greatly between tracks, especially between the R-track and the other two.

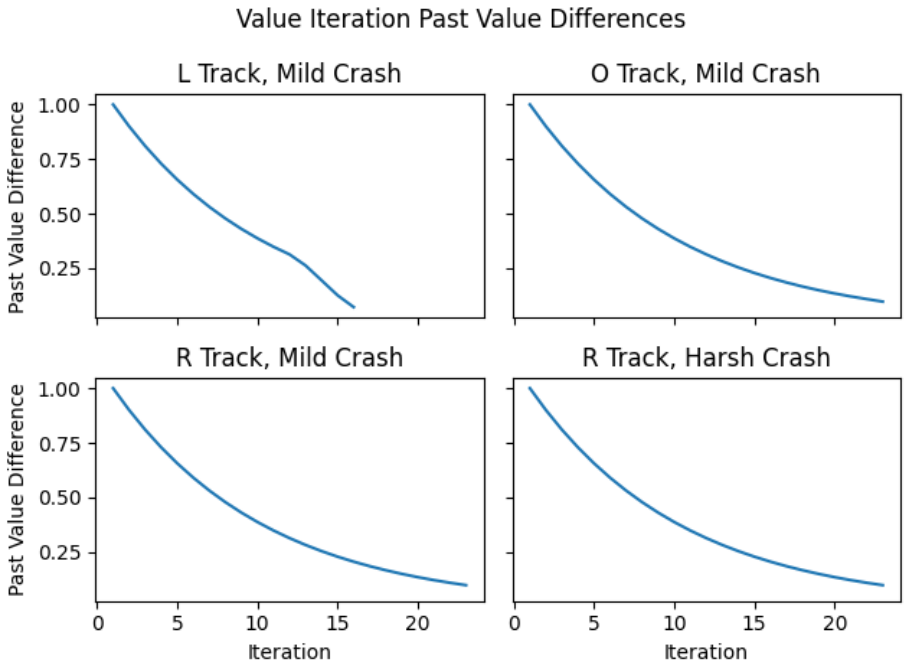

Figure 1: The progression of training (reflected in Past Value Difference) on the various tracks (L, O, and R) with various crash conditions (Mild vs Harsh) by performing value iteration.

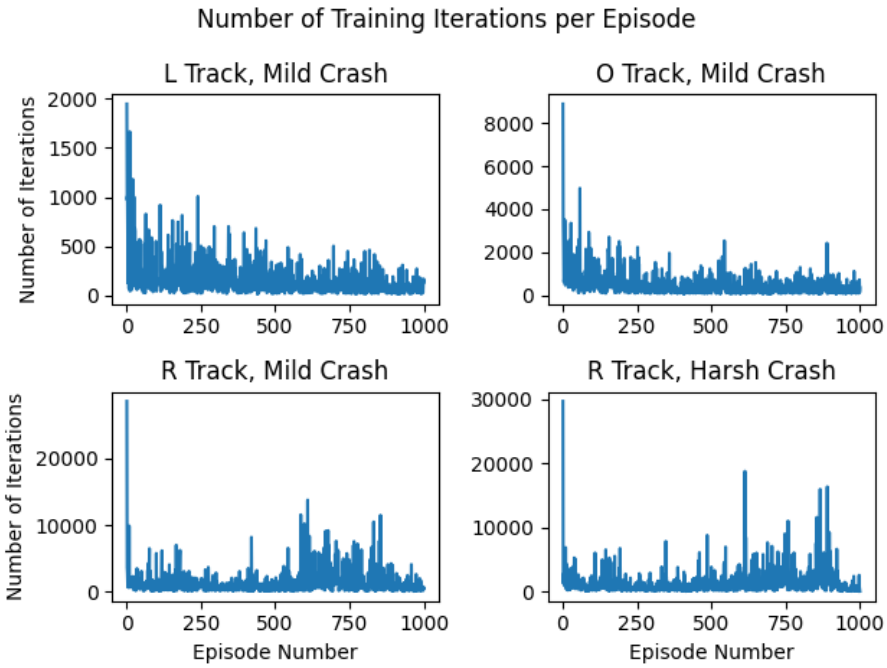

Figure 2: The number of iterations required to reach the finish in each episode of training on the various tracks (L, O, and R) with various crash conditions (Mild vs Harsh) by performing Q-learning.

Figure 3: The cumulative reward in each episode across the episodes performed on the various tracks (L, O, and R) with various crash conditions (Mild vs Harsh) by performing SARSA. The values are negative since we only include negative rewards and an absorbing state of the finish, which gives a reward of 0.

Track	Crash Type	Value Iteration	Q-learning	SARSA
L	Mild Crash	14.1	38.4	75
O	Mild Crash	27.8	134.33	100.75
R	Mild Crash	29.6	210	307.88
	Harsh Crash	28.4	192.75	234

Table 1: The average number of steps required for the car to go from the start to the finish in each case.

4. Discussion

When looking at the training data presented in Figures 1-3, we see that the L track with mild crash conditions continuously trains to optimal values faster, with less iterations, and with more reward than any other track-condition combination.
In value iteration, track L takes only 16 iterations to reach our threshold value, as opposed to the 23 required on the O- and R-tracks. This is due to the difference in size of the L-track, which is 11x37, while the O- and R-tracks are 25x25 and 28x30, respectively.
When performing Q-learning, Figure 2 shows that track L has comparatively low iterations

per episode, while track R has large iterations required at times, and requires, on average, more iterations to finish than either other track. In both the L- and O-tracks, there is a trend towards lower number of iterations, however, both R-track conditions begin trending to lower number of iterations, but are prone to perturbation, as shown in the random spiking around episode 600.
Lastly, Figure 3 shows the learning of SARSA through the cumulative reward in each episode. Here, we can appreciate the slope of learning as the episodes progress. By approaching 0 the method is finding more rewarding paths, thus increasing the cumulative reward in each episode.
Regarding our testing results, we are able to conclude that the value iteration method does indeed have the best results in terms of number of steps required for a test car to make it from the starting line to the finishing line. Value iteration is followed by Q-learning in step count, with SARSA taking the most number of steps on average. An interesting case arises on the O-track, where SARSA has a 25% shorter step count than Q-learning. This may be due to the design of the track and/or the hyperparameters being used for Q-learning were sub-optimal, or a local minima.
Another interesting case to observe is that in the harsh crash scenario on the R-track, all three reinforcement learning methods took fewer steps when compared to their mild crash counterparts. This may be due to the harsh crash forcing a premature restart within an episode and allowing for an "extra, mini" episode to occur. However, this does not explain the (albeit, small) difference in the value iteration method, which does not rely on episodes. Further tests would be required to determine whether that difference is meaningful.

5. Conclusion

After performing three different reinforcement learning methods: value iteration, Q-learning, and SARSA (State-Action-Reward-State-Action), on the commonly used "racetrack problem" across three different race tracks, with two crash types being tested on the R-track, we have found that the optimal strategy in terms of minimizing the number of steps taken to reach the finish line is value iteration.
Value iteration is not always a viable strategy, as it requires knowledge of the entire space being used in order to iterate. This is where Q-learning and SARSA begin to be relied on. Neither require knowledge of the space, which is a benefit in practical scenarios. The difference between the two is that Q-learning is an on-policy method, while SARSA is off-policy. In our use case, it appears that the off-policy strategy leads to better results than on-policy, as demonstrated by Table 1.

6. Appendix

6.1 Supplemental Tables