Project 5: Reinforcement Learning

Dongjun Cho DCHO13@JH.EDU
Programming Project 5
605.649
7 May 2022

Abstract

This project is intended to implement different type of the reinforcement algorithm: Value Iteration, Q Learning, SARSA, for three types of racetrack dataset.

1. Introduction

The reinforcement learning algorithm is one of the algorithms for machine learning. Reinforcement is one of the ways to learn through trial and error. For example, a child does not know how to walk when they first walk but learning how to walk while interacting with the environment is called reinforcement learning. Based on this reinforcement, reinforcement learning is an algorithm that finds goals by learning through mistakes and rewards. Similar to traditional neural networks learning weights and biases from labeled data, they use the concept of Reward to learn weights and biases. The goal is to learn the best behavior or policy.

Reinforcement learning(RL) is a learning process in which agents and environments interact through information called the action, state, and reward, achieving a given goal. The purpose of the paper is to research and implement various reinforcement learning algorithms: Value iteration, Q-Learning, and SARSA. These RL algorithms are the methods of learning generalized tasks. In this report, 3 track datasets are used to evaluate RL algorithms. Each of the datasets represents different types of race car tracks, L-track, O-track, and R-track. Using these RL algorithms, it teaches race cars how to track from starting point to the finishing point without crashing into the walls. When the race car crashes into the walls, there are two methods to handle it, 'soft-crash', and 'harsh-crash'. The soft crash method is if the car crashes into the walls, the car is placed at the nearest position on the track to the place where it crashed and set the speed/velocity to zero. The hard crash method is if the car crashes into the walls, the car returns to starting points and set the speed to zero. Since the harsh crash method resets the position of the car whenever it crashes into the wall, it would require more trial and error than a soft crash. So, it would take more time to execute because it is more difficult to train due to more trial and error. Hypothesis 1: We hypothesize that the harsh crash method would take more steps than the soft-crash method. To test this hypothesis, we compare the number of steps that need to finish the race to find the optimal policy for these algorithms using 3 racetrack datasets. Since algorithms take too much time to train, it would be difficult to find the optimal policy for all of the tracks.

In Section 2, we describe how each algorithm works, in section 3, we describe the experimental approach to the algorithm, in section 4, we present the result of our experiment. We discussed the behavior of the algorithm in section 5, and then we conclude the discussion with the result and possible future works in section 6.

2. Description of Algorithms

2.1 Components

There are some components of notation that construct the reinforcement learning algorithm. S is the set of states in the environment. The racetrack dataset is the environment of this project. A is the set of actions available to the agent. T is the state transition function.

$$S \times \ A \rightarrow \pi(S)$$
$$T(s, a, \ s\prime) = P(s\prime|s, a)$$

The probability of transitioning to state S prime, given that the agent is currently in state S and applies action a. So, $\pi(s)$ is the policy. It keeps the progress that the agents determine their actions based on the current state. R is a reward function.

$$S \times \ A \rightarrow \ R$$

It receives the reward when we take action A while we are in the state S. γ is the discount factor. It explains the amount by which feature reward is discounted.

2.2 Markov Decision Process

Markov Decision Process is invented by Richard Bellman in the late 1950s. Given policy π, it determines how well that policy performs, and it finds out that the value function $V^{\pi}(s)$ for all of the states in the problem. Using greedy policy, it determines the optimal action based on the current state.

$$\pi(s) = argmax_{a \in A}[r(s, a) + \gamma \sum_{s\prime \in S} T(s, a, s\prime)V^{\pi}(s\prime)]$$

2.3 Bellman's Line Generation Equation

For this project, it uses Bellman's equation as a wall detector. In this equation, the agent at some time explores that environment and tries to find the path to the goal. Based on the starting point and velocity, the agent knows the endpoints. Using starting points and endpoints, it generates the path between two points to determine the state of the track.

$$V(s) = \max_a(R(s, a) + \gamma V(s\prime))$$

2.4 Value Iteration

Unlike Q-learning and SARSA, Value Iteration actually isn't a learning algorithm. It's a dynamic programming algorithm that solves a Markov decision process. It computes the value function V_t using $Q_t(s, a)$. It estimates the value of taking action in the state and acting from that point on in accordance with the greedy policy derived from the current value function. It tries to find Q(s, a) by updating Q, which is the sum of the current value and discounted value (γ) from each action.

For the value iteration, it first initializes the value for all of the states to 0. It originally set the goal state as 0, but since it takes too much time to iterate/execute, I decided to increase the final goal to a positive number slightly. So, I set the final goal to 1. Then, it calculate the $Q_t(s, a)$ for each state.

$$Q_t(s, a) = R(s, a) + \gamma \sum_{s \in S} T(s, a, s')V_{t-1}$$

Then, it finds the optimal action for each states.

$$\pi_t(s) = argmax_{a \in A} Q_t(s, a)$$
$$V_t(s) = Q_t(s, \pi_t(s)$$

Then, it keeps incrementing the learning procedure steps and repeats the process until it reaches convergence.

2.5 Q-Learning

Unlike Value Iteration, Q-learning is the learning algorithm for reinforcement learning. Just like Value Iteration, Q-learning tries to find optimal Q(s, a). It is the objective of learning the underlying Q-function. The agent in Q-learning learns based on the learning time difference in prediction for successive time steps. For search strategies, it uses Epsilon (ϵ)-Greedy policy to find the optimal balance between exploration and exploitation.

$$\pi(x) = \begin{cases} argmax_{a \in A} Q(s, a) & \text{with probability (1 - } \epsilon\text{).} \\ randomaction & \text{with probability } \epsilon. \end{cases}$$

For each state, ϵ-greedy policy tries to find optimal action with the probability of the greedy policy method. It applies what the optimal action would have been according to the current estimate of our Q-function. It uses an off-policy algorithm. The off-policy algorithm is the policy that includes the exploration step and does not necessarily follow the greedy choice. The update is based on the greedy choice. But, the learning doesn't necessarily conform to the policy applied.

For Q-learning, it initializes the Q table to 0 just like value iteration. It randomly selects the starting states. Then, it chooses the action with the highest Q value from the Q table for each state. It chooses action via ϵ-greedy policy. Then, it takes action a, then observes r and s'. Then it updates the Q(s, a) using the formula below.

$$Q(s, a) = Q(s, a) + \eta(r + \gamma max_{a'} Q(s', a') - Q(s, a))$$

It updates the current state based on the Q(s, a). Then, it repeats the process until it reaches the terminal state.

2.6 SARSA

SARSA is another learning algorithm for reinforcement learning. It is a mnemonic for the steps taken in learning. SARSA stands for State-Action-Reward-State-Action. It is very similar to the Q-learning algorithm. It uses ϵ-Greedy policy to find optimal Q value. It updates the Q based on the immediate experience following the Q value ($Q_t(s', a')$). Unlike Q-learning, SARSA uses an on-policy algorithm. It approximates the Q-function for the current policy that is being used. It runs as an online algorithm.

For the SARSA algorithm, it works just like the Q-learning algorithm. The only difference is how it updates the Q function. In SARSA, it updates the Q value using this formula below,

$$Q(s, a) = Q(s, a) + \eta(r + \gamma Q(s', a') - Q(s, a))$$

It updates the current state and current actions based on the Q(s, a). Then, it repeats the process until it reaches the terminal state.

3. Experimental Approach

For this project, I couldn't apply tunning process because it took too much time to run a single iteration. For parameters, several parameters needed to consider for value iteration, Q-learning, and SARSA. In value iteration, it has a learning rate (γ), and error (ϵ). So, the learning rate set to 0.9 and 0.01 for error. For Q-learning and SARSA, it has an epsilon greedy policy (ϵ), learning rate(γ), total iteration, and alpha. For Q-learning and SARSA, it sets the total iteration as 500, 0.1 for an alpha, and 0.1 for epsilon greedy policy. The learning rate and error set the same value from Value Iteration.

4. Results of Experiments

There are 3 racetrack datasets: L-track, O-track, and R-track. For each racetrack, it compared the number of steps for each of the type of crash. For the number of steps, it ran 10 experiments for each track and then took the average number of steps to get reasonable statistics. There are two types of crashes: soft crashes, and harsh crashes. For value iteration, it demonstrated the graph to visualize how the delta decreases and number of steps until it converged. For Q learning and SARSA, it demonstrated the graph to visualize the relationship between the number of steps and learning iterations.

4.1 Crash-Type: Soft Crash vs. Harsh Crash

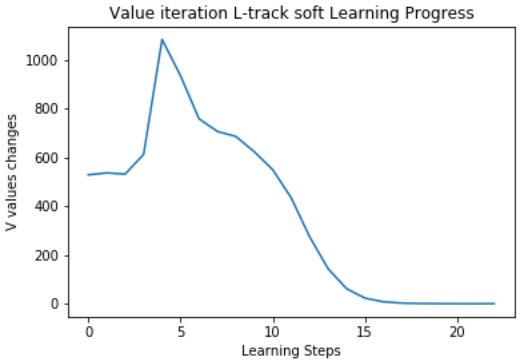
Soft Crash type is the method that the car placed at the nearest position on the track to the place where it crashed if it crashes into a wall. Harsh Crash type is the method that the car placed at the starting point if it crashes into a wall. It compared the number of average steps for each of the learning algorithms for soft crash type and harsh crash type.

Table 1: Average number of steps for both soft and harsh crash type.

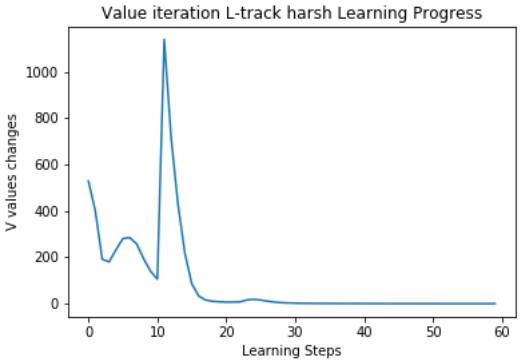
	Value Iteration		Q Learning		SARSA	
Crash Type / Track Type	Soft	Harsh	Soft	Harsh	Soft	Harsh
L-track	11	15	159	760	238	1698
O-track	25	28	669	1645	769	2402
R-track	25	32	271	1177	441	1702

4.2 Graphs: Value Iteration

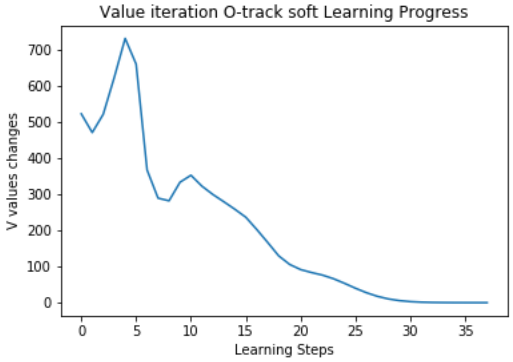
It demonstrated the relationship between v value changes and learning steps for value iteration. It also compared the learning steps between soft and harsh crash types.



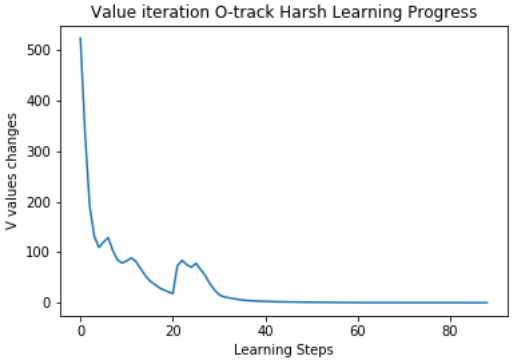
(a) Value Iteration L-track Soft Crash



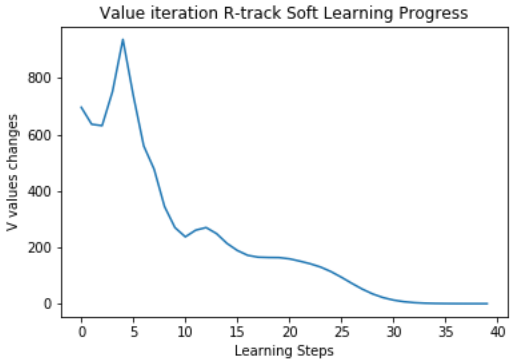
(b) Value Iteration L-track Harsh Crash



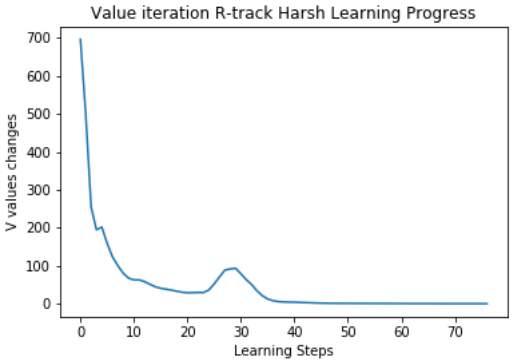
(c) Value Iteration O-track Soft Crash



(d) Value Iteration O-track Harsh Crash



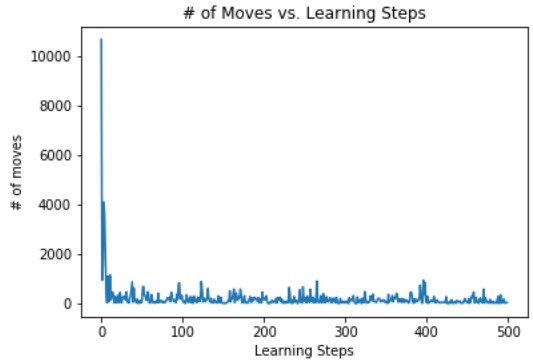
(e) Value Iteration R-track Soft Crash



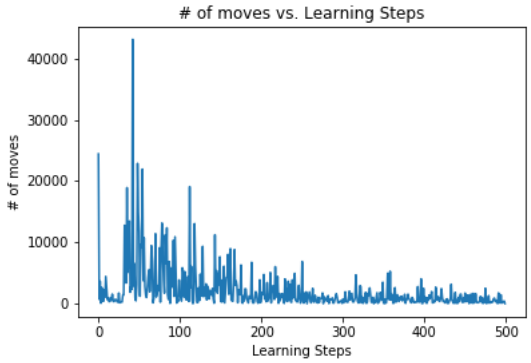
(f) Value Iteration R-track Harsh Crash

4.3 Graphs: Q Learning

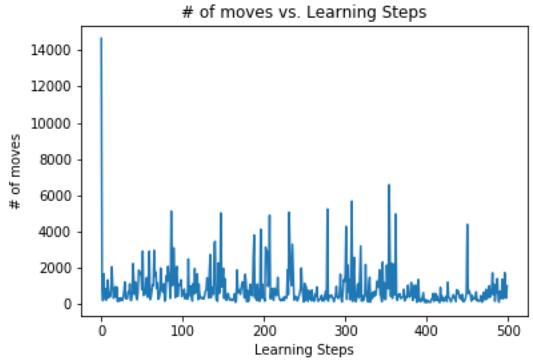
It demonstrated the relationship between the number of steps and learning for Q-Learning. It also compared the learning steps between soft and harsh crash types.



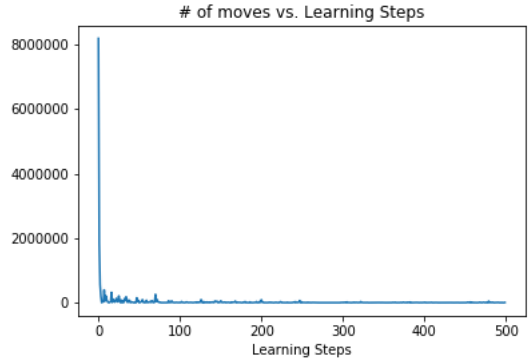
(g) Q Learning L-track Soft Crash



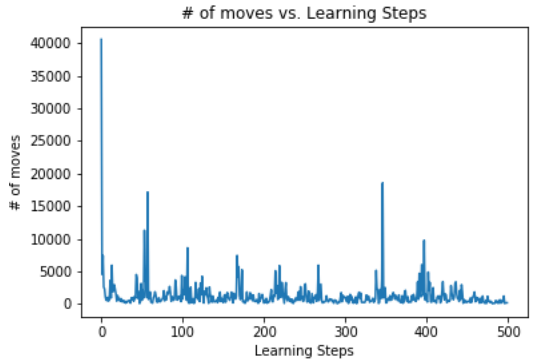
(h) Q Learning L-track Harsh Crash



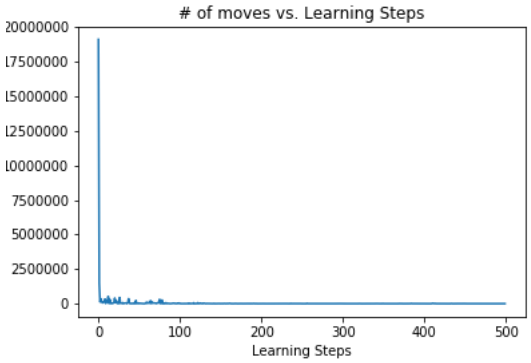
(i) Q Learning O-track Soft Crash



(j) Q Learning O-track Harsh Crash



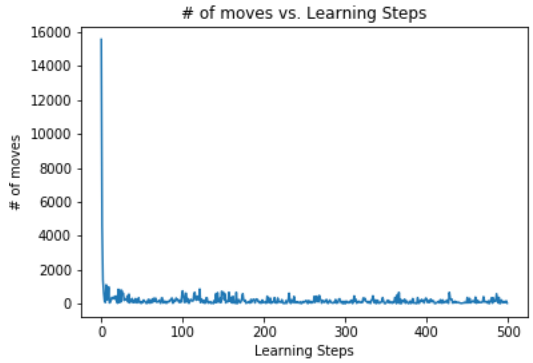
(k) Q Learning R-track Soft Crash



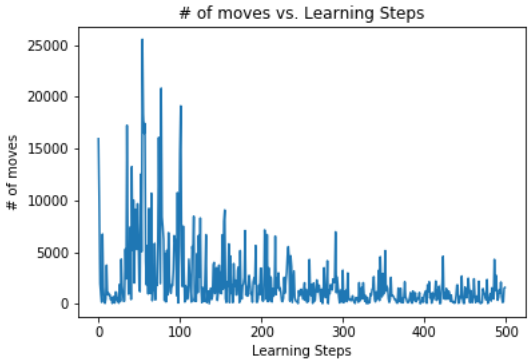
(l) Q Learning R-track Harsh Crash

4.4 Graphs: SARSA

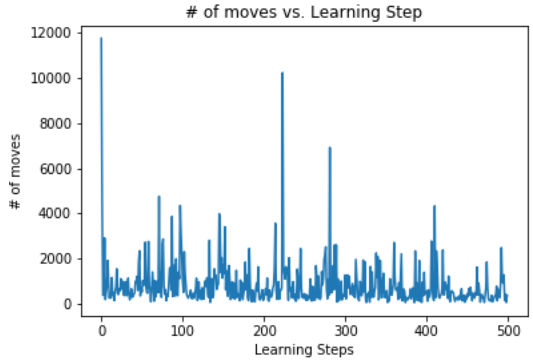
It demonstrated the relationship between the number of steps and learning for SARSA. It also compared the learning steps between soft and harsh crash types.



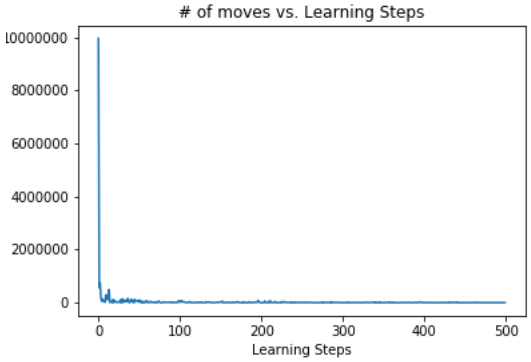
(m) SARSA L-track Soft Crash



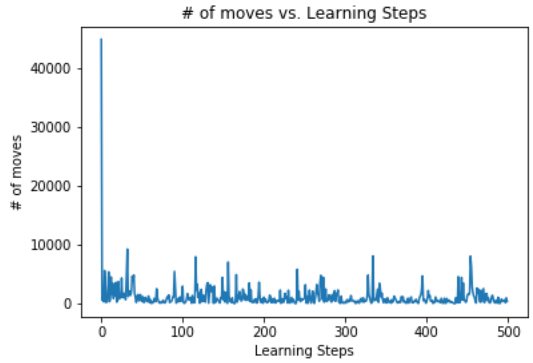
(n) SARSA L-track Harsh Crash



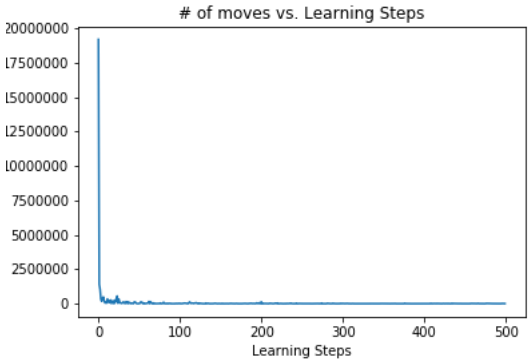
(o) SARSA O-track Soft Crash



(p) SARSA O-track Harsh Crash



(q) SARSA R-track Soft Crash



(r) SARSA R-track Harsh Crash

5. Behavior of Algorithms

The comparison of the number of steps between soft crash type and harsh crash type was expected. Hypothesis 1 stated that the harsh crash method would take more steps than the soft-crash method. Based on Table 1, it showed that the soft crash type performs better than the harsh crash type for all three algorithms. It showed that value iteration performs much better than Q-learning and SARSA. So, it showed that Q-learning and SARSA have not fully converged on a certain track. For this experiment, the maximum number of episodes (learning steps) set to 500. By increasing the maximum number of learning steps could improve the performance.

For value iteration graph, it showed how the v value decreased as the learning proceed until it reaches the threshold value. Based on the learning curve graph for value iteration, it showed that the soft crash type took smaller learning steps to converge than the harsh crash type. So, We can see that the harsh crash type is more difficult to learn than the soft crash type. In terms of execution time, the simple track(L-track) converged more quickly than the complex track(O-track and R-track) in value iteration. For a complex track with a harsh crash type, it took almost 16-19 hours to finish the convergence process.

For learning curves for Q-learning and SARSA, it is very interesting that the learning steps of the first iteration take much longer than other iterations. It is also interesting that the number of steps to reach the finish points gets smaller as the iteration goes on. In terms of execution time, the simple track(L-track) took fewer steps than the complex track. So, the complex track took more time to execute than the simple track. For complex track (O-track, R-track) with soft crash type, it took almost 6-8 hours to finish the first iteration. However, the complex track with harsh crash type took almost 11-13 hours to finish the first iteration.

6. Conclusion

This paper implemented the value iteration, Q-Learning, and SARSA. Throughout this paper, I was able to learn how the reinforcement learning algorithm learns. For reinforcement learning (Q-learning, SARSA), the performance of the algorithm would improve by increasing the maximum number of learning steps. For Future analysis and implementation, it would be better to implement the enhanced reinforcement learning algorithm to speed up the execution time, so I can increase the maximum number of learning steps.