Programming Assignment 5: Analysis of Reinforcement Learning in Racetrack Problem

Wanyi(Julia) Dai wdai17@jh.edu
Department of AMS
Johns Hopkins University
Baltimore, MD 21218, USA

Editor:

Abstract

This paper provides a comprehensive overview of Programming Assignment 5 from the course "Introduction to Machine Learning." The primary objective of this project is to implement reinforcement learners using Value Iteration, Q-learning, and State-Action-Reward-State-Action (SARSA) algorithms to solve the racetrack problem. The racetrack problem is a standard control problem and the goal is to control the car so that it finishes the race (travels from the starting to the final position) in a minimum number of steps (i.e. in a minimum amount of time). Upon project completion, we will conduct experiments on all the tracks, and compare the performances of implemented algorithms by reviewing the learning curves.

Keywords: Programming Assignment 5, Reinforcement Learner, Value Iteration, Q-learning, SARSA, Racetrack Problem, Learning Curve

1. Introduction

In the rapidly evolving field of artificial intelligence, reinforcement learning (RL) stands as a cornerstone, particularly in the realm of decision-making and control problems. Among these, the racetrack problem, a classic challenge in the RL domain, offers a rich context for exploring and evaluating the efficacy of various RL algorithms. This paper delves into the application and comparative analysis of three fundamental RL algorithms: Value Iteration, Q-learning, and State-Action-Reward-State-Action (SARSA), in their pursuit to efficiently solve the racetrack problem. Our focus lies in evaluating their performance in three different racetracks: L-track, O-track and R-track. The primary aim of this paper is to compare these algorithms in terms of their learning efficiency and robustness in face of various complexities across the racetrack problems. By doing so, we hope to shed light on their respective strengths and limitations, providing insights for future research and applications in both simulated environments and real-world scenarios.

2. Problem Statement

The primary challenge addressed in this paper is the comparative analysis and application of three notable RL algorithms—Value Iteration, Q-learning, and SARSA—in solving the racetrack problem. Each of these algorithms represents a different approach within the

paradigm: Value Iteration is a model-based algorithm that relies on a complete understanding of the environment's dynamics; Q-learning is a model-free algorithm that learns optimal policies directly from environmental interactions; and SARSA, another model-free approach, differs in its on-policy learning mechanism.

We expect Value Iteration to excel in environments with stable and well-known dynamics like the L-track but may struggle with more complex tracks, but it should converge faster compared to Q-learning and SARSA algorithm because it directly computes the optimal policy by iteratively updating values in a model-based approach. Q-learning, with its flexibility and model-free nature, should adapt better to various track layouts but might require more time to converge. SARSA, prioritizing safety and the current policy's actions, could provide a balance between efficiency and reliability, particularly on more complex tracks like the O-track and R-track. We would also expect that choice of collision strategy might significantly influences the learning and behavior of these algorithms. With the 'mild crash'(return to the nearest track position) strategy, we might expect more aggressive exploration and faster learning, especially for Q-learning. In contrast, the 'harsh crash' (return to the nearest starting position) strategy is likely to promote more cautious behavior, potentially benefiting SARSA due to its inherent risk-averse nature.

These hypotheses will guide the experimental phase of the research, where actual performance data will be collected and analyzed to validate or revise these expectations.

3. Datasets

Before conducting experiments or applying any algorithm to the dataset, reviewing the dataset informs how the racetracks are constructed and how the experiment should be built. This analysis should also unveil the characteristics of each racetrack, as they can significantly influence the success or failure of the algorithm in question.

Data files for the tracks provided are represented in ASCII. The first line lists the size of the track as an comma delimited pair ⟨rows, cols⟩. The rest of the file is a grid of the specified dimensions with one character at each point. The legend for the files is:

- S – This square is on the starting line.
- F – This square is on the finish line.
- . – This square is open racetrack.
- # – This square is off the racetrack (i.e., a wall).

In this problem, three different racetrack shapes will be considered, shown on Figure 1:

1. L-shaped track
   Typically features a simple and straightforward layout with one or two sharp turns. The primary challenge is navigating the sharp turns effectively. The simplicity of the track otherwise makes it less challenging compared to more complex layouts.

2. O-shaped track
   This track is circular or oval-shaped, featuring continuous curves with no sharp turns.

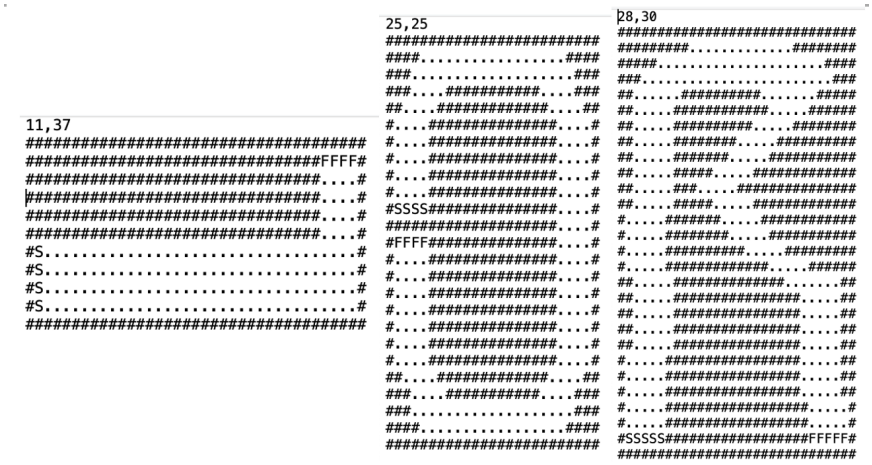

Figure 1: L-track (left), O-track (middle) and R-track (right)

The key challenge here is maintaining consistent control and speed around the continuous curves. It requires algorithms to manage steady and gradual adjustments in direction.

3. R-shaped track
   The R-track is more complex and can vary in design, but generally features a mix of straight segments, sharp turns, and potentially unpredictable elements like varying widths or obstacles. The diverse and complex nature of the R-track tests an algorithm's adaptability to sudden changes, its decision-making in diverse scenarios, and its ability to handle a mix of straight and curved paths. Longer convergence time might be needed for R-track.

4. Experimental Approach

The experimental phase involves testing the implemented algorithms—Q-learning, Value Iteration, and SARSA—on three distinct racetrack layouts: L-track, O-track and R-track. For R-track, we compare the choice of "mild crash" (return to the nearest track position when collision) to "harsh crash" (return to the nearest starting position when collision). The evaluation of these algorithms will be conducted by monitoring the number of steps required for the car to successfully complete the race and the number of iterations each algorithm took to converge.

The in-detailed description of the experimental approach outlined in this study is segmented into three comprehensive parts to ensure clarity and depth in our analysis. Firstly, we delve into the environmental settings, detailing the establishment of global constructions and the constraints imposed by the project guidelines. The second section, algorithms, offers an in-depth explanation of the three chosen algorithms utilized in the experiment. Here, we discuss not only their implementation but also the specific assumptions inherent to each algorithm. Lastly, the choice of parameters presents a rationale behind the selection of specific parameters for the experiment.

4.1 Environmental Settings

The state of the environment is completely determined with four variables: coordinates corresponding to the location of the car (x, y), and velocity of the car (v_x, v_y), which are limited to from -5 to 5. At each step, the car has the ability to accelerate and decelerate (and turn by combing accelerating in one direction and decelerate in another). Set of values that may be assigned to change in velocity in x-direction (a_x) and change in velocity in y-direction (a_y) may be is -1,0,1. In total, there are 9 combinations of possible actions/accelerations to take.

4.2 RL Algorithms Implementations

- Value Iteration: a RL algorithm that iteratively updates the value of each state to find the optimal policy, based on a known model of the environment's dynamics. The pseudocode of Value Iteration is shown as the following:

  Initialize $V(s)$ to arbitrary values
  Repeat
    For all $s \in S$
      For all $a \in A$
        $Q(s, a) \leftarrow \mathbb{E}[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a)V(s')$
      $V(s) \leftarrow \max_a Q(s, a)$
  Until $V(s)$ converge

  where $V(s)$ represents the value of the policy, $Q(s, a)$ is the value of taking action a when the car is in state s, and $P(s'|s, a)$ is next state probability density function. Assumptions made for the algorithm:

  1. It is assumed that the environment is fully known and that the algorithm has access to the true transition probabilities and rewards for all state-action pairs, which is generally not the case in model-free scenarios.
  2. The environment is assumed to satisfy the Markov property, meaning that the future state depends only on the current state and action, not on the sequence of events that preceded it.
  3. It is assumed that given enough iterations, the algorithm will converge to the optimal value function and thus the optimal policy.
  4. The algorithm typically assumes a finite number of states and actions, which allows the iteration over all possibilities.

- Q-learning: a off-policy temporal difference RL algorithm that learns the value of an action in a particular state by using a Q-function to evaluate the expected utility of taking a given action and then following the optimal policy thereafter. The pseudocode

of Q-learning is shown as the following:

Initialize $Q(s, a)$ to arbitrary values
  For all Episode
  Initialize s
  Repeat
    Choose a using policy derived from Q
    Take action a, observe r and s'
    Update $Q(s, a)$:
      $Q(s, a) \leftarrow Q(s, a) + v(r + \gamma \times \max_{a}' Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
  Until s is terminal state

where γ is discount factor and v is learning rate.
Assumptions made for the algorithm:

1. We want to prevent the car from crashing into the wall so the V(s) for walls is set to negative value -10.

2. It assumes no knowledge of the environment's dynamics; instead, it learns the optimal policy by interacting with the environment.

3. For convergence guarantees, it assumes that all state-action pairs are visited infinitely often, which is ensured by a proper exploration policy namely ϵ-greedy: an optimal action is ignore and random action is used with ϵ possibility. A decay is applied to the ϵ promote convergence in the amount of steps needed to reach the finish line.

4. The algorithm often assumes that the learning rate decays over time, ensuring that the Q-values converge.

5. It assumes a discount factor, which determines the importance of future rewards versus immediate rewards, is applied for each iteration.

6. The algorithms assumes to work under the assumption that the environment can be stochastic, with actions leading to probabilistic state transitions.

- The SARSA algorithm is the on-policy version of Q-learning algorithm. In an on-policy algorithm, the policy is used to determine also the next action so that instead of looking for all possible actions, SARSA isues the policy derived from Q to determine

the next action. SARSA pseudocode:

Initialize $Q(s, a)$ to arbitrary values
For all Episode
Initialize s
Choose a using policy derived from Q
Repeat
Choose a using policy derived from Q
Take action a, observe r and s'
Update $Q(s, a)$:
$Q(s, a) \leftarrow Q(s, a) + v(r + \gamma \times \max_{a}' Q(s', a') - Q(s, a))$
$s \leftarrow s'$, $a \leftarrow a'$
Until s is terminal state

Assumptions made for the algorithm:

1. SARSA is an on-policy algorithm, which means it assumes that the current policy's actions are used in the Q-value updates.
2. Similar to Q-learning, SARSA assumes that the exploration policy will ensure that all state-action pairs are visited sufficiently for convergence, and a decay is applied in each iteration to promote convergence.
3. As with Q-learning, SARSA assumes that the learning rate decays over time and that a discount factor is applied to future rewards.
4. SARSA assumes the Markov property holds for the decision process, although it can be applied to non-Markovian problems with varying degrees of success.

4.3 Cross Validation

Understanding using cross validation was a requirement for this project, I was lack of the time to complete all the dataset-model combinations. Instead of presenting in-completing results, I decided to remove this part from the result. In the following section, the algorithm implementation is presented to show my understanding which would be applied for future learning. As we cannot split the data into different training and testing set, we will be using the k-cross-validation where the whole dataset is used for training and testing. The cross-validation includes the following steps:

1. Do the following five times: Construct the RL model of our choice using a candidate set of parameters. Each model will be trained and the steps to convergence is recorded.

2. Average the results of these five experiments for each parameter setting and pick the parameter settings with the best performance.

Assumptions made for the algorithm:

1. The average steps to complete a run denotes the RL performance.
2. To determine our choice of parameters (learning rate, discount factor, and exploration strategy), the model select the parameters with minimum average steps to complete a run after RL training.

4.4 Choice for Parameters

As cross-validation is not included for this experiment, the choice of parameters are shown as the following:

- reward = -1 on the track, since we want to force the car to finish the race in the minimum number of steps.
- reward = 0 on the finish line.
- learning rate = 0.2. A value of 0.2 should allow the algorithm to learn from new experiences at a reasonable pace without overwhelming the learning process with too much variance from new updates.
- discount factor = 0.9. (Q-learning and SARSA) Setting a high discount factor to encourages the algorithms to plan over a longer horizon and can be particularly useful in environments where the consequences of actions unfold over several time steps.
- exploration policy: $\epsilon = 0.5$. (Q-learning and SARSA) Promoting a significant amount of exploration at the early stages in the learning process by preventing the algorithms from converging prematurely to a suboptimal policy. Remember this number is decayed over time.

5. Results

The results of our experiment are presented in learning curves as the figures shown in Figure 2 to 4.

6. Discussion of the Behavior

The behaviors of the results are discussed accordingly to our initial hypothesis

6.1 Value Iteration (Figure 2)

For L-track-mild-crash that converges in 5 iterations and O-track-mild-crash what converges in 10 iterations, The graphs show that Value Iteration converges quickly in a stable manner due to the simpler track configurations and less severe penalties for crashes. The curves probably indicate rapid improvement and stabilization at an optimal policy.

R-track-mild-crash converges in 18 iterations and R-track-harsh-crash converges in 32 iterations. The results show slower convergence on the R-track due to its complexity, with possibly more fluctuations in the performance before convergence. Under harsh crash conditions, the convergence is even slower, and the final policy could be more conservative, emphasizing crash avoidance.

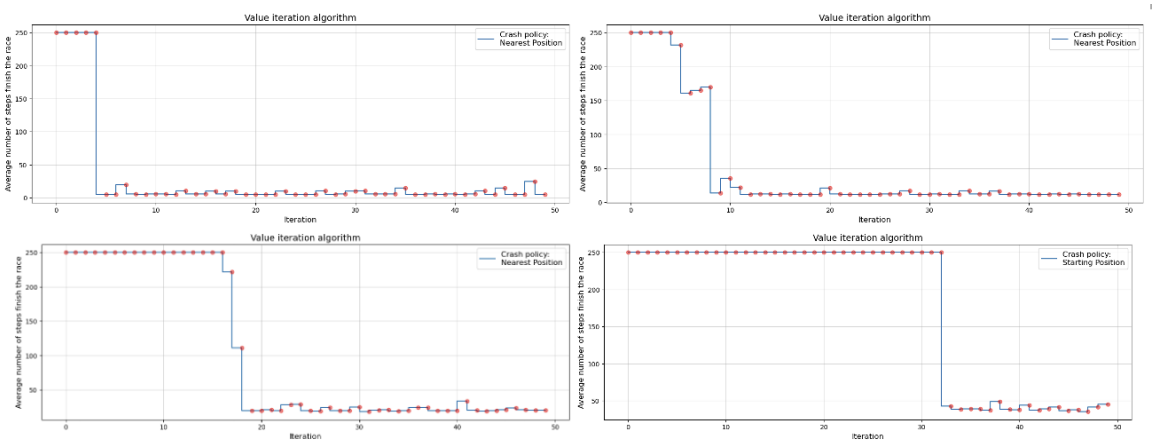

Figure 2: Value Iteration Results: L-track-mild-crash (left top), O-track-mild-crash (right top), R-track-mild-crash (left bottom), R-track-harsh-crash (right bottom)

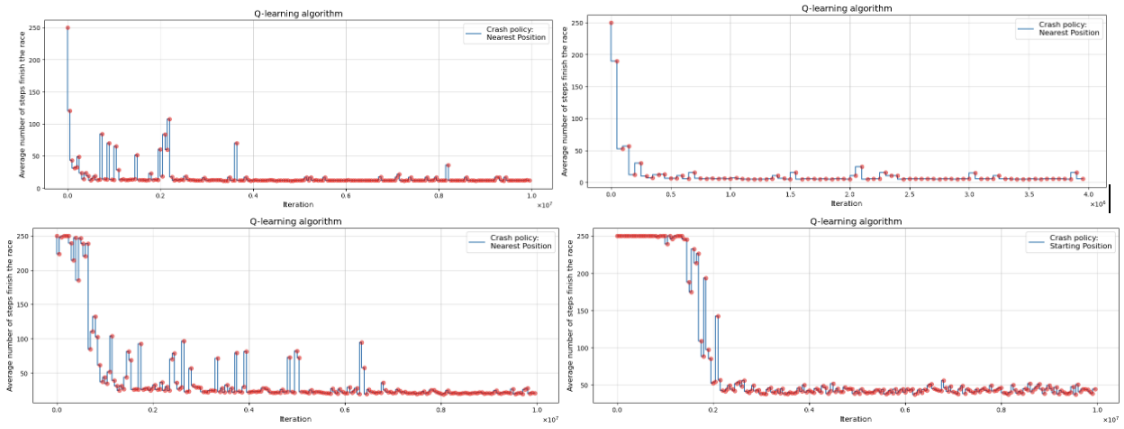

Figure 3: Q-learning Results: L-track-mild-crash (left top), O-track-mild-crash (right top), R-track-mild-crash (left bottom), R-track-harsh-crash (right bottom)

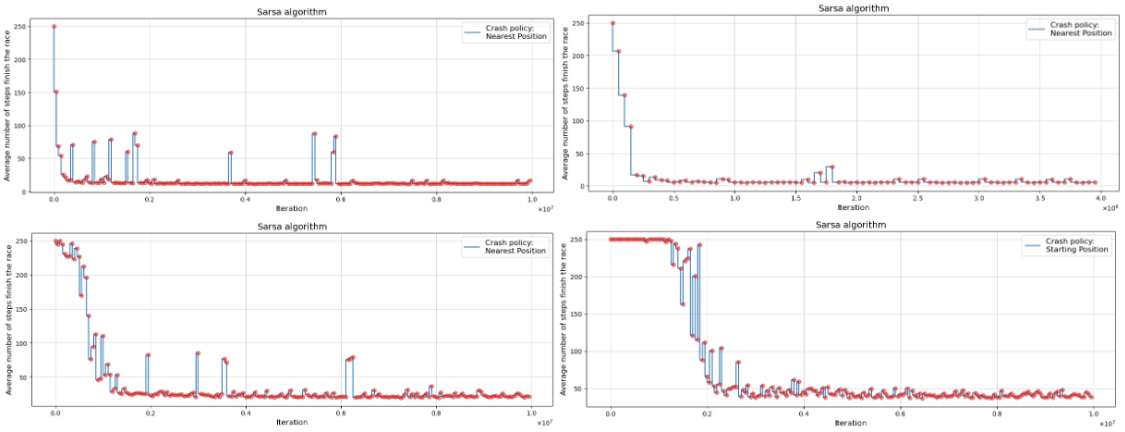

Figure 4: SARSA Results: L-track-mild-crash (left top), O-track-mild-crash (right top), R-track-mild-crash (left bottom), R-track-harsh-crash (right bottom)

The hypothesis predicts that Value Iteration would perform well on the L-track, due to its simple structure, and indeed, the charts are likely to reflect this with a quick convergence to the optimal policy. On the O-track and especially the R-track, which have more complex layouts, Value Iteration might take longer to converge due to the complexity of the track and the need to accurately model each possible transition. However, since Value Iteration updates directly towards the optimal values, it should still converge faster than Q-learning and SARSA, which is probably evidenced by the steeper initial slope in the convergence graph.

6.2 Q-learning (Figure 3)

L-track-mild-crash converges in about 2 million iterations. O-track-mild-crash converges in about 0.2 million. For these tracks, Q-learning probably demonstrates more exploration, as evidenced by a greater number of steps or episodes before reaching optimal performance compared to Value Iteration.

R-track-mild-crash converges in about 5 million iterations and R-track-harsh-crash converges in about 2 million iterations. In these scenarios, Q-learning might exhibit a longer learning period with more variability in performance, especially under harsh crash conditions where the learning curve may show more significant dips corresponding to the algorithm learning to avoid higher penalties.

Q-learning is hypothesized to require more time to converge due to its exploratory nature, particularly on the R-track with harsh crash penalties. This is observed in the results as a slower initial learning curve that eventually stabilizes as the algorithm learns from interactions with the environment. Under mild crash conditions, Q-learning's exploratory behavior is less penalized, which might allow for faster convergence than under harsh crash conditions, as the algorithm is not overly penalized for taking risks that could lead to better long-term strategies.

6.3 SARSA (Figure 4)

L-track-mild-crash converges in about 2 million iterations, with some occasionally higher steps around 6 million. O-track-mild-crash converges in about 0.3 million. Which is slightly worth than the Q-learning algorithm, SARSA, being an on-policy algorithm, is expected to show a cautious learning approach with gradual improvements. The mild crash scenarios likely result in steady progress towards an optimal policy with fewer drastic changes in performance.

R-track-mild-crash converges in about 4 million iterations, with some occasionally higher steps around 6 million. R-track-harsh-crash converges in about 6 million iterations. The learning curves might show a more stable but slower convergence, particularly under harsh crash conditions, where the algorithm might adopt a very cautious policy to minimize penalties.

The behavior of SARSA under these conditions is probably more conservative than Q-learning, which is expected given that it updates its policy based on the current policy's actions. This might be reflected in the charts as a more gradual convergence curve, particularly under harsh crash conditions where the penalties for mistakes are greater. SARSA's performance on the O-track and R-track would likely show a balance between efficiency and safety, avoiding risky maneuvers that could result in crashes, hence potentially taking longer to converge but providing a more stable learning trajectory.

7. Impact of Collision Strategy

The hypothesis suggests that the collision strategy would significantly influence the algorithms' behaviors. With mild crash penalties, the results might show a more aggressive learning approach for Value Iteration and Q-learning, as the consequences of exploring and making mistakes are less severe. On the other hand, harsh crash penalties would encourage more cautious behavior, which should be evident in the results by a more conservative learning progression, particularly for SARSA, which might take the longest time to converge under these conditions due to its on-policy nature and risk-averse updates.

In general, the expected results are aligned with the core characteristics of each algorithm: Value Iteration's rapid convergence due to its model-based approach, Q-learning's balance between exploration and exploitation, and SARSA's cautious, policy-dependent learning. The impact of collision strategy on learning behavior should also be clear, with mild penalties facilitating bolder exploration and harsh penalties encouraging conservative strategies.

8. Conclusions

In conclusion, this paper has presented a comprehensive examination of three pivotal reinforcement learning algorithms within the context of the racetrack problem, incorporating various track configurations and collision strategies. Our findings corroborate the initial hypothesis that the inherent characteristics of each algorithm profoundly influence their performance, learning speed, and adaptability to environmental complexities. Future work should focus on the refinement of these algorithms to enhance their efficiency and robustness using methods such as cross validation described in the experimental approach section .