Contents

Project 6: Reinforcement Learning

Jaudat Raza JAUDAT.RAZA@JHU.EDU
Master of Computer Science
John Hopkins University
Baltimore, MD 21218

Editor: Jaudat Raza

Abstract

This document will implement a reinforcement learner and apply it to the racetrack problem. The resource used will be linked below in the resource section of the document. The document will follow a simple pattern for which a less than 5-minute video will be submitted as well. The Jupyter Notebook is used to implement the algorithms and the result will be displayed under the code with explanation of the output in the video.

1 Problem Statement

The purpose of this assignment is to implement reinforcement learner and apply it to the racetrack problem. The racetrack problem is a popular control problem, often used in reinforcement learning assignments. The goal is to control the movement of a race car along a pre-defined racetrack. You want your race car to get from the starting line to the finish line in a minimum amount of time. In this version of the problem, your agent will be the only racer on the track, so it is more like a time trial than a full competitive race.

We will experiment with two different versions of how "bad" it is to crash. The first variant says that, if the car crashes into a wall, it is placed at the nearest position on the track to the place where it crashed, and its velocity is set to (0, 0). The second, harsher variant says that when a car crashes, its position is set back to the original starting position, as well as zeroing its velocity. Essentially, in this latter case if you crash, the agent must start over from the beginning. Both cases of the crash will be implemented but the results from only the R shaped will be mentioned in the report per the project description. For this assignment, the project specifies that for any attempt to accelerate, there is a 20% chance that the attempt will simply fail, and the velocity will remain unchanged at the next timestep.

The three Algorithm we will run the racetrack problem on are Value Iteration, Q-Learning, and SARSA algorithm. We will use the provided tracks, which are L-track, 0-track, and R-Track. I hypothesize that the algorithm will be able to finish the racetrack. Value iteration will learn faster than Q learning because of the reward function. It has info of the bigger picture compare to Q-Learning and SARSA. The third hypothesize is regarding the crash. In which I think the later case will be worse than the first one. Starting from the beginning each time will impact the learning rate and the result from the R shaped Track will show that return to nearest open position after the crash will perform better. One more thing I am hypothesize is based on the complexity of the racetrack. L might be able to be solved faster compare O and the R should be slowest out of the three

2 Algorithm Studied

In this project we are covering Value Iteration, Q-Learning and State Action Reward State Action (SARSA).

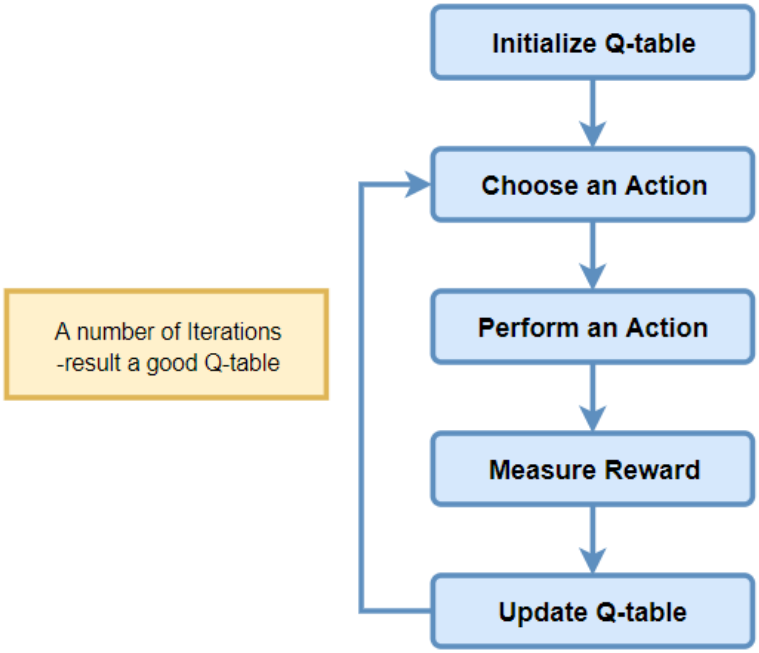
2.1 Q Learning Algorithm

In most real-world cases, it is not efficient to build a complete model. In these cases, the transition and reward are unknown. To solve this issue, model free reinforcement learning algorithm like Q Learning were created. Q-learning is an off-policy learner. Means it learns the value of the optimal policy independently of the agent's actions.

2.1.1 Q-Function

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s', a^g) - Q(s,a))]$$

2.1.2 Q Learning Algorithm Process



2.1.2.1 Initialize the Q-Table

First the Q-table must be built. There are n columns, where n = number of actions. There are m rows, where m = number of states.

2.1.2.2 Chose an Action

There are four action you can pick from, which are up, down, right, and left.

2.1.2.3 Perform an Action

Now we basically keep performing the step and updating the table for a set of defined number and keep updating the table. So, at the beginning, the epsilon is high, but as the agent explore the environment, the epsilon decreases, and the agent is able to exploit the environment. Each action it performs, it updates the state under the action takes.

2.1.2.4 Measure Reward

Now after taking the action, we measure the reward. Based on which state you end up being based on action

2.1.2.5 Evaluate

This process is repeated till the learning is stopped. In this way the Q-Table is been updated and the value function Q is maximized. Here the Q (state, action) returns the expected future reward of that action at that state.

2.2 SARSA

. SARSA is almost the same thing as Q learning the differences is that the SARSA chooses another action following the same current policy and using $r + \gamma Q(s', a')$ as target. SARSA is called on-policy learning because new action a′ is chosen using the same epsilon-greedy policy as the action a, the one that generated s′.

2.2.1 SARSA Equation

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a))]$$

2.3 Value Iteration

Value iteration is known as a model-based reinforcement learning technique because it tries to learn a model of the environment where the model has two components, which are transition function and reward function. In short, in value iteration, the race car knows the transition probabilities and reward function (i.e. the model of the environment) and uses that information to govern how the race car should act when in a particular state. Being able to look ahead and calculate the expected rewards of a potential action gives value iteration a distinct advantage over other reinforcement learning algorithms when the set of states is small.

2.3.1 Value Iteration Equation

This is picked up from UC Berkley Machine Learning Course.

$$V_{i+1}^{*}(s) \leftarrow \max_{a} \sum_{s'} T(s,a,s') \left[R(s,a,s') + V_i^{*}(s')\right]$$

3 Tuning

To compare the performance of the algorithm for the different crash scenarios, the principal hyperparameters were kept the same for all runs. These hyperparameters are listed below.

3.1 Discount Rate

The Discount Rate was set to 0.9 in all of the algorithms.

3.2 Learning Rate

The learning rate was set to 0.2, it really doesn't impact much just the timing.

3.3 Exploration Probability

The exploration probability was set to 0.2 as suggested by the project description. To give the algorithm a little challenge.

3.4 Number of Training Iteration

The number of iterations is set to 50. The number of cycles or tries it can make per iteration is set to 100.

4 Result

For Q learning and SARSA the number of iterations had to be really high for it to actually learn the track. For Q learning, I set the algo to start from 500000 steps and go up all the way to 6000000. For SARSA it goes all the way upto 2000

4.1 R-Track

4.1.1 Value Iteration

Number of Iteration	Near Position Crash	Return to Start Crash
5	100	100
10	100	100
15	46.2	100
20	38.3	100
25	24.2	37.4
30	29.8	42.6
35	23.2	32.1
40	22.1	29.5
45	26.6	36.6
50	23.1	34.3

4.1.2 Q Learning

Number of Iteration	Return To Start Crash	Near Position Crash
500000	500	500
1000000	500	251.3
2000000	500	44.3
3000000	435.9	28.3
4000000	398.3	72.3
5000000	238.9	76.2
6000000	37.6	25.3

4.1.3 SARSA

Number of Iteration	Return To Start Crash	Near Position Crash
200000	500	500
400000	500	500
600000	500	500
800000	500	350.4
1000000	420.8	220.3
1200000	302.8	180.3
1400000	233.4	90.2

4.2 L-Track

4.2.1 Value Iteration

Number of Iteration	Near Position Crash
5	100
10	12.4
15	11.5
20	11.9

25	11.5
30	10.7
35	10.9
40	10.8
45	11.9
50	11.2

4.2.2 Q Learning

Number of Iteration	Near Position Crash
500000	15.2
1000000	12
2000000	11.1
3000000	11.2
4000000	11.5
5000000	10.8
6000000	10.2

4.2.3 SARSA

Number of Iteration	Near Position Crash
200000	125.2
400000	92.2
600000	70.1
800000	85.2
1000000	38.5
1200000	32.8
1400000	24.2

4.3 O-Track

4.3.1 Value Iteration

Number of Iteration	Near Position Crash
5	6.1
10	6.0
15	5.5
20	4.9
25	4.5
30	5.4
35	4.3
40	4.5
45	4.1
50	4.2

4.3.2 Q Learning

Number of Iteration	Near Position Crash
500000	4.2
1000000	3.9
2000000	53.8
3000000	5.7
4000000	5.0

5000000	4.5
6000000	4.3

4.3.3 SARSA

Number of Iteration	Near Position Crash
200000	45.2
400000	32
600000	20.1
800000	11.2
1000000	10.5
1200000	8.8
1400000	15.2

5 Conclusion

So now for some of the things we see from the results is that all 3 algorithms do learn the track and are able to finish the track as I hypothesized at the beginning of the paper. Now they were all able to finish the track, but the complication of each track did also play a role. Like for example as our results show that all the algorithms took longer time or more steps to complete the R-Track, where they were able to perform better in the O and L track. But the interesting thing that I saw was that SARSA and Q learning were able to perform better the value iteration in O Track compare to e L Track. I thought R would be the hardest to learn, but then, it would be O and then L.

The hypothesize again which will learn faster was very accurate and the results clearly showed it. For Value iteration I only had to max out around 50 iteration where in Q learning and SARSA 50 would not even learn the track. I had to maximize the learning by starting at 500000 and go up to 10000000. It took so long that I had to let the algorithm run over night to get the results for R , L and O Track. I was only using quarter of my computer resources but even for that to run overnight was impressive. Basically, what I noticed that what the average step Value iteration was able to achieve in 50 iterations or so took Q learning almost 600000, which is a very huge difference.

The third hypothesis that we made was regarding the crash, which one would take longer? This was only applied to R Track. The running process for any of the tracks was very long, so the conclusion on this hypothesis is just based on R Track and as we see the Crash where the agent had to start back from start took drastically more steps then starting from the next safe point on the track.

6 Resources

- StatQuest with Joh Starmer. “Gradient Descent, Step-by-Step.” YouTube, 05 Feb. 2019, https://www.youtube.com/watch?v=sDv4f4s2SB8.
- Srinivasan, Aishwarya V. “Stochastic Gradient Descent - Clearly Explained !!” *Medium*, Towards Data Science, 7 Sept. 2019, towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.
- “7 Types of Activation Functions in Neural Networks: How to Choose?” *MissingLink.ai*, missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.