Mountain Car Game using Reinforcement Learning

Surya Teja Chavali, Charan Tej Kandavalli, Sugash T M,
Department of Computer Science and Engineering, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, India.

E-mail: bl.en.u4aie19014@bl.students.amrita.edu, bl.en.u4aie19013@bl.students.amrita.edu, bl.en.u4aie19062@bl.students.amrita.edu

***Abstract*— Reinforcement learning has gained a lot of attention recently, and we need to understand its principles. In this report, we have discussed the approach to building a reinforcement learning agent for a game called Mountain Car. The ideology of the game is defined as a car trying to reach a mountain top, using its potential energy against the force of gravity.**

In this case, the car is imagined to be in a U – shaped valley and aims to reach the rightmost top end. Since the gravitational force is assumed to be dominating the engine's power in this scenario, it has to gain momentum and reach the mountain top. We used Q – learning to achieve this process and have used the OpenAI gym Mountain Car environment (Python Library), for the agent to interact with it.

Keywords—Reinforcement Learning, Q – learning, Open AI

I. Introduction

One of AI's main goals is to create truly automated agents that interact with their surroundings to learn optimal behaviors and improve over time through trial and error. Reinforcement learning deal with the above statement in a very systematic manner. It is a subset of machine learning wherein an AI agent strives to maximize its rewards in its surroundings by taking behaviors that maximize its rewards. An RL agent (reinforcement learning agent) is the heart of any RL application. It simulates the experience of being a player in a traditional game that is played frequently to better one's strategy over time.

Deep learning allows RL to scale and optimize decision-making problems that were previously difficult to understand and are intractable, i.e., settings with high-dimensional state and action spaces. Though, RL has been a very interesting application of Machine Learning algorithms in training the agent, due to a dearth of intriguing and demanding situations in which to experiment, RL has typically been difficult to get started with. The solve the above problem, OpenAI has come with a set of RL environments spread diversely from simple black/white 'toy' based ones to the advanced simulated robotic and 3D video gaming environments.

In the Mountain Car game, we consider that the car is set to be the RL agent. The environment we have used in our project is the "OpenAI Gym Mountain Car environment". Getting back to our problem statement, the car starts in between the hilltops and aims to gain speed (momentum) as it reaches closer and closer to the goal point on the top.

II. Literature review

In this section, let us iterate through some of the previously executed works on RL formulations for this particular use case and games similar to this.

In [1], the authors have discussed a generative flow of the RPCL algorithm in the Open AI gym environment and tested that algorithm on three of the gym environments, namely – *inverted pendulum with discrete control input*, ***mountain car problem with discrete/continuous control inputs****, a bipedal walker with a 4-dimensional continuous control input.* They were also able to successfully visualize the performance metrics of the demonstrator and their agent and have estimated that the reward factor for major states is 0 to complete the task as quickly as possible. They have also proved that Continuous episodic tasks are better performing than the discrete ones, in Behaviour cloning.

The authors have modelled a TAMER framework in [2], by replacing the MDP reward signal with a human reward signal, TAMER avoids the MDP reward signal's limited and delayed characteristics. They have shown that SARSA (λ), along with linear model-based gradient descent is known to perform better in Mountain car RL agent formulation. The authors have mainly focussed on integrating different RL-based approaches such as modelling Q – function and TAMER-based learning.

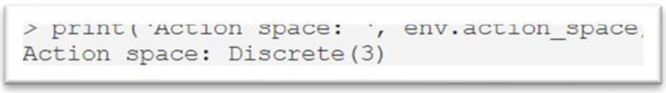
III. Methodology

A. Exploring the environment

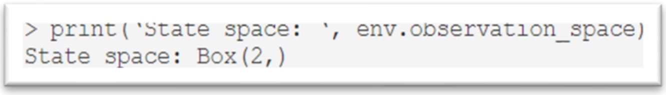
The environment used here is straightforward, and the user has to make sure of two things about the car at any given time: its location and speed. The above-mentioned parameters act as arguments for our environment. Since we are dealing with a 2D problem here, we will only be having a 2- dimensional state space. At each given time, the state of the car is determined by a vector comprising its horizontal location and speed.

At any one time, the car can only take one of three actions: accelerate ahead, accelerate backward, or do nothing. The environment will provide a new state every moment the agent acts (a position and speed) and the episode ends when either the car reaches the top position. By default, the three actions are represented by the integers 0, 1, and 2. Below is a sample piece of code, which explains how the state and action spaces are distributed.

As suggested by the above output, the state space is visualized as a 2 – dimensional box, and the action space is a linear array of three discrete actions. The environment has some pre-defined (default) range values for the state vectors and speeds vectors of the car – [1.2 – 0.6] and [-0.07 – 0.07] respectively.



```
> print('Action space: ', env.action_space
Action space: Discrete(3)
```

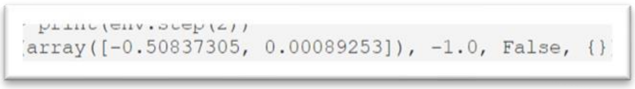


```
> print('State space: ', env.observation_space)
State space: Box(2,)
```

B. Values and Rewards

Rewards act as a primary step in deciding whether our agent is moving in the right direction or not. To achieve this, we assign our agent a reward when it shows positive reinforcement and punishes it when it shows the property of negative reinforcement.

The ideology followed here is that whenever the car reaches the top position, it is going to receive a big reward. But we don't want to reward the model simply when it reaches the top position, as this would make learning harder for it. Therefore, at each stage, the model may be rewarded appropriately to the quantity of energy it accumulates. The car must first accumulate enough kinetic energy (speed) before converting it to potential energy (height). Rewarding the car for the amount of energy it accumulates each time it reaches the top position, will help it learn to reach it.



```
print(env.step(2))
array([-0.50837305, 0.00089253]), -1.0, False, {}
```

Values are the basic units of measuring how good is the action considered by the car to reach an optimal point from that state. The purpose of reinforcement learning techniques is to approximate this value function as closely as possible. As a result, the car can consider all of its options (accelerating front and backward, or not moving at all) at any time. After that, it may calculate the predicted value of future states and choose the optimal course of action.

To achieve the above-mentioned process, the agent has to undergo rigorous training, which is done using the help of Deep Q Networks, which is discussed in the next section.

IV. System Architecture

A. Q – Learning algorithm

The Q - learning algorithm, is an off-policy algorithm, which means that it analyses and improves a target policy that is not the same as the observational policy that was used to create the data.

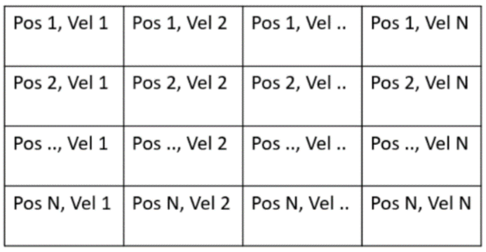
The learning algorithm is self–paced and solely based on building a Q – table (s, a), which denoted the values for probabilities for action to take place, given a state value. This technique is generally used in developing Q Networks for 3D simulated games. However, using a powerful technique like on the simple 2D game might overfit our model and not train the agent properly. This is because our state space is comprised of continuous values, implying that there exists an infinite number of states. Assume our location is between -20 and 20, with 0 being the car's beginning location. There are an endless number of places between -20 and 20 degrees. To overcome this hurdle, we follow the process of discretization.

In discretization, we split up our state space into chunks. One simple approach to do this is to round the first number and a second number of the state vector to the nearest 0.1 and 0.01, respectively, and then multiply the first number by 10 and the second one by 100.

We need techniques to observe the present state, take an action, then observe the effects of that action to implement Q-learning in Open AI Gym. Below is a small snippet of code through which we can accomplish this.

B. Tabular discretization method

Q-learning is a learning algorithm for finding optimal solutions in discrete state-action space for Markov Decision Processes. Discretizing the environment in order to decrease the state-action space is one technique to apply Q-learning to continuous state-action space. The objective of tabular discretization technique is to learn the value for all possible states in a table format. All conceivable states will be included in the table. The state space of our project consists of position and velocity which are continuous, implying that there exist an infinite number of states. In our project the position of the car is bounded between -1.2 and 0.6, where 0.521 is the car's starting point



Pos 1, Vel 1	Pos 1, Vel 2	Pos 1, Vel ..	Pos 1, Vel N
Pos 2, Vel 1	Pos 2, Vel 2	Pos 2, Vel ..	Pos 2, Vel N
Pos .., Vel 1	Pos .., Vel 2	Pos .., Vel ..	Pos .., Vel N
Pos N, Vel 1	Pos N, Vel 2	Pos N, Vel ..	Pos N, Vel N

Between -1.2 and 0.6, there are an infinite number of possible positions. We can't describe our state space with an arbitrarily huge table, so we break it down into parts and this job is done by tabular discretization method. As an example, the discretized space of the mountain car problem is shown in tabular form in Fig(1), in this figure the resolution is considered as N so there are N possible values for velocity, and N possible values for velocity. So a total of N^2 possible states. In the implementation of our project, we considered N as 12, which means there 12 tables with each table consists of 12 rows and 3 columns (length of the action space is 3).

C. Implementation

The environment that we have considered for our project is from OpenAI Gym. OpenAI Gym is a toolbox that lets you create a number of different simulated environments. In the initial part of implementation, the environment has been explored well in order have a clear idea on required parameters. While exploring it is found that the range of position is bounded between -1.2 and 0.6, range of velocity is bounded between -0.07 and 0.07 and the reward threshold is -110. Reward threshold of an environment is nothing but a reward value that an agent must earn before the task gets completed. Further in order to perform discretization, the discrete_state( ) function has been defined which takes observation as its input argument and it returns discretized state space table. The Q – learning algorithm had been implemented in which the programmatic form of Bellman's equation was incorporated. The train( ) function had been defined in order to train an agent in the environment by using discretized Q – learning method. Finally, we have performed Hyper parameter tuning to gain a knowledge on setting the optimal value for our hyper parameters and additionally the graph had been plotted in order to visualize the convergence of the algorithm on performing hyper parameter tuning.

V. Conclusion

In the past few years, reinforcement learning studies have made significant progress; yet, due to the complexity of the real world, many challenges need to be studied and solved. In this report, we have attempted to design a reinforcement learning agent to tackle the Mountain Car problem. To summarize the above, we mentioned the working and usage of Open AI's gym library for building and training an RL agent for the game 'Mountain Car'. We have also discretized our state space and made some minor changes to the Q-learning algorithm to tackle this problem.

The proposed model and architecture are still under development, that is, we are still trying to deal with algorithms and approaches such as TD3 and PPO. We are developing an efficient code for improving the model's performance when interacting with the Open AI's gym environment.

To the best of our knowledge, we have presented and put forward an idea of tackling the particular Mountain Car with one of the most used approaches to make an agent – Q learning. When an agent trains from a bunch of experiences, deep Q-learning takes advantage of experience replay. They also portray better self-learning abilities and scalability as far as distribution of code is concerned

References

[1] arXiv:2009.09577

[2] Knox, W. & Setapen, Adam & Stone, Peter. (2011). Reinforcement Learning with Human Feedback in Mountain Car. AAAI Spring Symposium - Technical Report.

[3] https://gym.openai.com/envs/MountainCar-v0/

[4] https://en.wikipedia.org/wiki/Mountain_car_problem

[5] https://towardsdatascience.com/reinforcement-learning-applied-tothe-mountain-car-problem-1c4fb16729ba

[6] https://link.springer.com/content/pdf/10.1007/BF00114726.pdf
.

[7] https://www.researchgate.net/figure/The-Mountain-Car-problem-is-atypical-benchmark-in-reinforcement-learning-The-goal-isto_fig4_337581742