Modelling a Reinforcement Learning Agent For Mountain Car Problem Using Q – Learning With Tabular Discretization

Surya Teja Chavali[1], Charan Tej Kandavalli[2], Sugash T M[3], Amudha J.[4]
[1 – 4]Department of Computer Science and Engineering, Amrita School of Engineering, Bengaluru, Amrita Vishwa Vidyapeetham, India.
bl.en.u4aie19014@bl.students.amrita.edu, bl.en.u4aie19013@bl.students.amrita.edu, bl.en.u4aie19062@bl.students.amrita.edu, j_amudha@blr.amrita.edu

2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon) | 978-1-6654-9790-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/MysuruCon55714.2022.9972352

***Abstract*— The advancement of Reinforcement Learning (RL) algorithms has shown great success in the field of adaptive and responsive video games. Open AI gym models are open-source visual interfaces for representative environments for modelling robust reinforcement learning tasks. This work aims at solving the mountain car problem involving the "MountainCar-v0" environment used from the OpenAI gym collection framework. Using the Q - learning algorithm, we present a novel algorithm using Tabular-Discretization for solving the considered problem. The gym environment we have considered has a default reward threshold, which is the reward value that an agent must earn before the task gets completed. The proposed model achieved a value close to the threshold parameter set for the experiment. Hyperparameter tuning has been carried out in the latter, to better understand the convergence of the model in terms of the variance of all 4 hyperparameters taken into account- learning rate, decay rate, epsilon, and the discount factor. This proposed method of Tabular Q-Learning is both memory and time efficient in terms of other existing algorithms such as SARSA and Online Q-Learning.**

Keywords— RL, OpenAI gym, Discretization, Q – learning, Hyperparameters, Mountain Car Problem

I. INTRODUCTION

One of AI's key goals is to construct fully automated agents that interact with and interpret situations to learn optimal and efficient behaviours 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 behaviours that maximize its rewards. An RL agent (reinforcement learning agent) [1] 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, Open AI [2] 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 used in this work is the "Open AI Gym Mountain Car environment"[1] [3]. This problem statement states that when 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.

The organization of this paper is as follows: In section II, we iterate on the existing systems and works developed on solving a real-time use case using reinforcement learning algorithms. In sections III and IV, we detail the system architecture and the terminologies taken into account in the formulation of the RL agent. Section V discusses the experimental results obtained and the inferences gained. Lastly, we are going to conclude and mention the future improvements in the final section.

II. LITERATURE REVIEW

In this section, we iterate through some of the previously executed works on RL formulations for different use cases and games similar to this.

In [4], 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 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 discrete ones, in behaviour cloning.

Apart from this, many different researchers have modelled agents mainly for detection tasks using advanced HMM algorithms such as Viterbi (an explanation of this can be seen in [5]) and Forward-Backward algorithms, such as in [6]. J Amudha and R Deepalakshmi in [7] have discussed a similar reward updating policy method called temporal feature extraction. These set algorithms were further discussed and portrayed in [8] and [9].

The authors have modeled a TAMER framework in [10], 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 focused on integrating different RL-

based approaches such as modeling Q – function and TAMER-based learning.

The use cases of RL have been extended to various domains such as object detection. In [11], the authors also modelled an RL agent to tackle the task of gaze behaviour prediction using fixational qualitative scores.

III. System Architecture

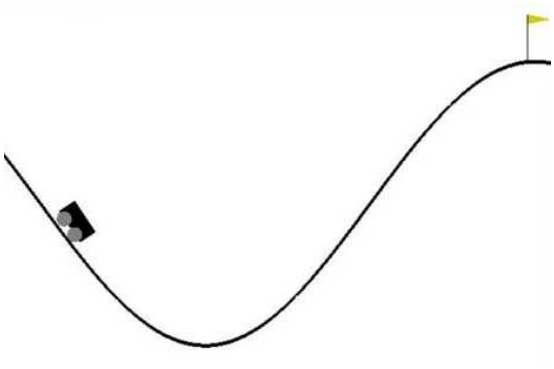

Fig. 1. glimpse of the mountain car environment

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 this 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.

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 speed vectors of the car – (1.2 – 0.6) and (-0.07 – 0.07) respectively.

B. *Values and Rewards*

Rewards act as a primary step in deciding whether the agent is moving in the right direction or not. To achieve this, we assign the 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 for 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. Values are the basic units of measuring how good the action is 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.

IV. Proposed Methodologies

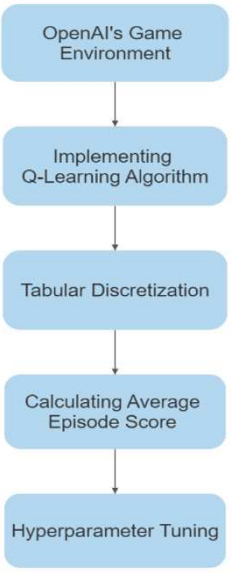

Fig. 2. A flow of the tools and algorithms employed

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 [13] for 3D simulated games. However, using a powerful technique like the simple 2D game might overfit the model and not train the agent properly. This is because the state space here, is comprised of continuous values, implying that there exists an infinite number of states. For instance, assume a location, which 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 the 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.

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 [12]. The objective of the tabular discretization technique is to learn the value of all possible states in a table format. All conceivable states will be included in the table. The state space of this work consists of position and velocity which are continuous, implying that there exist an infinite number of states. In this paper, the position of the car is bounded between -1.2 and 0.6, where 0.521 is the car's starting point.

TABLE I
STATE SPACE OF THE MOUNTAIN CAR PROBLEM

P 1; V 1	P 1; V 2	P 1; V …	P 1; V n
P 2; V 1	P 2; V 2	P 2; V …	P 2; V n
…….	…….	…….	…….
P n; V 1	P n; V 2	P n; V ….	P n; V n

Between -1.2 and 0.6, there are an infinite number of possible positions. We can't describe the state space with an arbitrarily huge table, so we break it down into parts and this job is done by the tabular discretization method. As an example, the discretized space of the mountain car problem is shown in Table I, for which the scope of consideration is N, so, there are N possible values for positions and N possible values for velocity. So, a total of N^2 possible states. While implementing this work, the value of N has been considered as 12, which indicates there are 12 tables with each table consisting of 12 rows and 3 columns (the length of the action space is 3).

For instance, let's assume that the vehicle's position is bounded between -10 to 10, in this space, there are infinite Q – tables possible and we can't just visualize an infinitely large state space table to achieve this. Therefore, we choose to discretize and split it into sub-parts. In Table I, 'P' denotes the Position of the car on the slope and 'V' denotes the velocity (a vector of speed) of the car on the slope.

Also, a higher learning rate (α), during the algorithm training period indicates that your Q settings are being updated gradually in larger amounts of values. Therefore, we should decrement this while the agent is learning to stabilize the output of your model, by incorporating the decay factor, which eventually converges into an ideal course of action.

C. Implementation

The environment that considered here for this work, is from the Open AI Gym collection of interactive environments. Open AI Gym is a toolbox that lets you create several different simulated environments. In the initial part of the implementation, the environment has been explored well in order to have a clear idea of the required parameters. While exploring it is found that the range of position is bounded between -1.2 and 0.6, the range of velocity is bounded between -0.07 and 0.07 and the reward threshold is -110. The reward threshold of an environment is nothing but a reward value that an agent must earn before the task gets completed.

One of the most important hyperparameters in model optimization is the epsilon factor, which is used to choose particular measures based on the Q values we currently have. Let's consider an illustration, where we choose the pure greedy technique (epsilon = 0), we will always choose the state with the highest q value out of all the states.

Further, in order to perform discretization, the discrete_state() function has been employed 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. This presents a problem for exploration because it is simple to become locked at a local optimum. So, we use epsilon to introduce randomization. For instance, regardless of the real q value, if epsilon = 0.3, we will choose random actions with a 0.3 probability.

$$Q(s, a) = r + \gamma \, max_{a'}[Q(s', a')] \qquad (1)$$

Equation (1) represents a Bellman Equation, where r and γ denote reward and discount factors respectively, and a state's value function under a policy 'π' is denoted by the symbol v(s). The equation illustrates the connection between such a state's value and the values of its succeeding states. The train() function had been defined in order to train an agent in the environment by using discretized Q – learning method. Finally, hyperparameter tuning has been performed to gain a piece of knowledge on setting the optimal value for the chosen hyperparameters, and additionally, the graph had been plotted to visualize the convergence of the algorithm post hyperparameter tuning.

V. Experimental Results And Analyses

The entire Q – learning process and model training part are performed in Jupyter Notebook on a machine, running on 16 GB of RAM with an Intel i7 processor for which, execution time was about 1.5 days. The entire hyperparameter tuning has been split up into 4 different divisions with various experiments involving:

a) An increase in learning rate (α) from 0.008 to 0.05
b) A decrement in discount factor (γ) from 0.99 to 0.55
c) A reduction in decay rate (a) from 100 to 10
d) An increment in epsilon value (ϵ) from 0.01 to 0.1

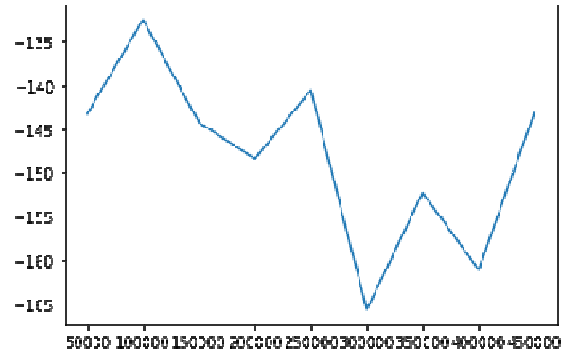

Fig. 3. Graphical result of Average value vs Number of Episodes

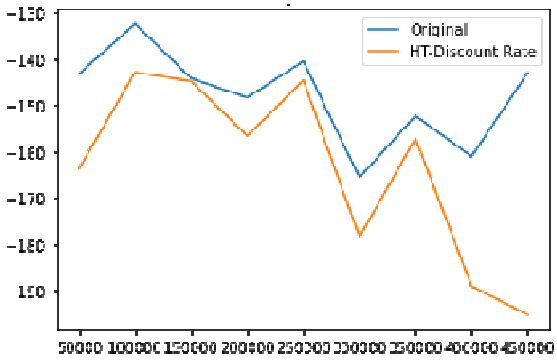

Fig. 4. Visualization of Average value for Agent with initial parameters and Hyper-parameter tuned (Discount Rate) Agent.

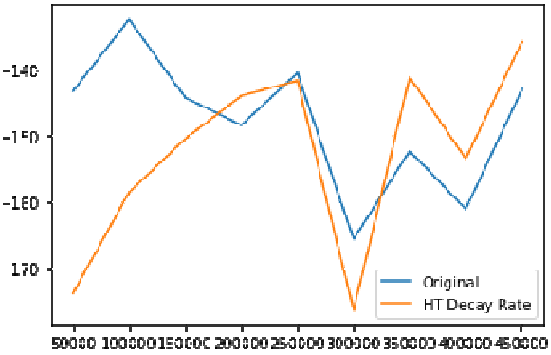

Fig. 5. Visualization of Average value for Agent with initial parameters and Hyper-parameter tuned (Decay Rate) Agent

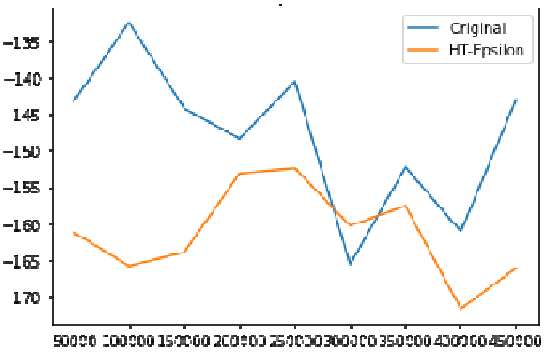

Fig. 6. Visualization of Average value for Agent with initial parameters and Hyper-parameter tuned (Epsilon) Agent

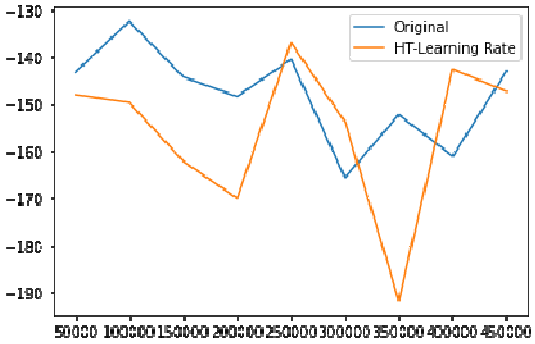

Fig. 7. Visualization of Average value for Agent with initial parameters and Hyper-parameter tuned (Learning Rate) Agent

TABLE II
VARIOUS SCORE VALUES OBTAINED WHEN THE HYPERPARAMETERS WERE TUNED

Learning Rate	Decay	Epsilon	Discount Factor	Score	Training Time
0.008	100	0.01	0.55	-200.0	2 hr 5 mins.
0.008	100	0.1	0.99	-166.0	3 hr 3 mins.
0.05	100	0.01	0.99	-164.0	1 hr 59 mins.
0.008	100	0.01	0.99	-108.0	8 hr 46 mins.
0.008	**10**	**0.01**	**0.99**	**-109.0**	**1 hr 42 mins.**

TABLE III
PERFORMANCE COMPARISON

Metric	Tabular Q – Learning	Q - Learning	SARSA (online)
Memory Efficiency	Low	High	High
Training Speed	Fast	Slow	Slow
Model Convergence	Fast	Fast	Fast
Validation of agent	High	Poor	High

Fig. 3-11 portray the average number of episodes (vs) the score values along with performance comparisons of the original agent as well as the agent when the parameters have been tuned. Table II describes all the hyper-tuned parameters and the respective training time along with the score values achieved. We notice that the optimal score achieved is when the epsilon and decay are the lowest. This is due to the fact that low decay rates and fewer epsilon values lead to higher rewards and helps the agent in choosing the best action-value pair. In Table III, a glimpse of comparison between the existing techniques and Tabular Discretization has been provided.

VI. CONCLUSION AND FUTURE WORK

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. When an agent trains from a bunch of experiences, deep Q-learning takes advantage of experience replay [16]. They also portray better self-learning abilities and scalability as far as the distribution of code is concerned. 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 the state space and performed various experimental changes to the Q-learning algorithm to tackle this problem.

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 – Tabular Discretization with Q learning, which performs in an optimal fashion as compared to the Online Q – learning or the Online SARSA methods. The proposed can further be extended to deal with algorithms and approaches such as Temporal Difference – 3 (TD3) and Proximal Policy Optimization (PPO) [17] can execute them on the Mountain Car problem and analyze the resultant optimal score (reward) values.

REFERENCES

[1] M. Naeem, S. T. H. Rizvi and A. Coronato, "A Gentle Introduction to Reinforcement Learning and its Application in Different Fields," in IEEE Access, vol. 8, pp. 209320-209344, 2020, doi: 10.1109/ACCESS.2020.3038605.

[2] Brockman, Greg, et al. "Openai gym." arXiv preprint arXiv:1606.01540 (2016).

[3] Ms.S.Manju, & Dr.Ms.M.Punithavalli,. (2011). An Analysis of Q-Learning Algorithms with Strategies of Reward Function. International Journal on Computer Science and Engineering. 3.

[4] Tao, Feng and Yongcan Cao. "Learn to Exceed: Stereo Inverse Reinforcement Learning with Concurrent Policy Optimization." ArXiv abs/2009.09577 (2020): n. pag.

[5] S. T. Chavali, C. T. Kandavalli, S. T M and S. R, "Grammar Detection for Sentiment Analysis through Improved Viterbi Algorithm," 2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), 2022, pp. 1-6, doi: 10.1109/ACCAI53970.2022.9752551

[6] N. Shlezinger, Y. C. Eldar, N. Farsad and A. J. Goldsmith, "ViterbiNet: Symbol Detection Using a Deep Learning Based Viterbi Algorithm," 2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), 2019, pp. 1-5, doi: 10.1109/SPAWC.2019.8815457

[7] R. Deepalakshmi and J. Amudha, "A Reinforcement Learning based Eye-Gaze Behavior Tracking," 2021 2nd Global Conference for Advancement in Technology (GCAT), 2021, pp. 1-7, doi: 10.1109/GCAT52182.2021.9587

[8] S. Krishnan, J. Amudha and S. Tejwani, "Gaze Exploration Index (GE i)-Explainable Detection Model for Glaucoma," in IEEE Access, vol. 10, pp. 74334-74350, 2022, doi: 10.1109/ACCESS.2022.3188987

[9] Chandrika, K.R., Amudha, J., Sudarsan, S.D. (2020). Identification and Classification of Expertise Using Eye Gaze—Industrial Use Case Study with Software Engineers. In: Bansal, J., Gupta, M., Sharma, H., Agarwal, B. (eds) Communication and Intelligent Systems. ICCIS 2019. Lecture Notes in Networks and Systems, vol 120. Springer, Singapore. https://doi.org/10.1007/978-981-15-3325-9_30

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

[11] C. Jyotsna and J. Amudha, "Eye Gaze as an Indicator for Stress Level Analysis in Students," 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2018, pp. 1588-1593, doi: 10.1109/ICACCI.2018.8554715

[12] B. Jang, M. Kim, G. Harerimana and J. W. Kim, "Q-Learning Algorithms: A Comprehensive Classification and Applications," in IEEE Access, vol. 7, pp. 133653-133667, 2019, doi: 10.1109/ACCESS.2019.2941229

[13] Fawaz, Hassan, et al. "Graph Convolutional Reinforcement Learning for Collaborative Queuing Agents." arXiv preprint arXiv:2205.12009 (2022)

[14] Nayak, S., Ravindran, B. (2020). Reinforcement Learning for Improving Object Detection. In: Bartoli, A., Fusiello, A. (eds) Computer Vision – ECCV 2020 Workshops. ECCV 2020. Lecture Notes in Computer Science(), vol 12539. Springer, Cham. https://doi.org/10.1007/978-3-030-68238-5_12

[15] C. K. Ramachandra and A. Joseph, "IEyeGASE: An Intelligent Eye Gaze-Based Assessment System for Deeper Insights into Learner Performance," Sensors, vol. 21, no. 20, p. 6783, Oct. 2021, doi: 10.3390/s21206783.

[16] I. T. Nicholaus and D.-K. Kang, "Robust experience replay sampling for multi-agent reinforcement learning," Pattern Recognit. Lett., vol. 155, pp. 135–142, 2022

[17] Pathmakumar T, Elara MR, Gómez BF, Ramalingam B. A Reinforcement Learning Based Dirt-Exploration for Cleaning-Auditing Robot. Sensors (Basel). 2021 Dec 13;21(24):8331. doi: 10.3390/s21248331. PMID: 34960425; PMCID: PMC8706451.