2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS) | 978-1-6654-0610-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/CSITSS54238.2021.9682970

Solving the Lunar Lander Problem using Reinforcement Learning

Rohit Sachin Sadavarte
Computer Science and Engineering
R.V College of Engineering
Bangalore
rohitsachins.cs18@rvce.edu.in

Rishab Raj
Computer Science and Engineering
R.V College of Engineering
Bangalore
rishabrajp.cs18@rvce.edu.in

B Sathish Babu
Computer Science and Engineering
R.V College of Engineering
Bangalore
bsbabu@rvce.edu.in

***Abstract*—Reinforcement Learning is an area of machine learning concerned with enabling an agent to solve a problem with feedback with the end goal to maximize some form of cumulative long-term reward. In this paper, two different Reinforcement Learning techniques from the value-based technique and policy gradient based method headers are implemented and analyzed. The algorithms chosen under these headers are Deep Q Learning and Policy Gradient respectively. The environment in which the comparison is done is OpenAI Gym's LunarLander environment. A comparative analysis of the two techniques is then performed in order to understand the differences in a deterministic episodic state space. Both of these algorithms are model free, that is, they can be applied irrespective of the environment and do not need to have any knowledge about the exact details of the environment itself, hence the comparison can be extended to any other environment that shares these characteristics.**

***Index Terms*—Reinforcement learning, Deep Q-Networks, OpenAI Gym, Lunar Lander, Neural Networks, Machine Learning**

I. Introduction

REINFORCEMENT learning is a branch of machine learning and artificial intelligence that has seen promise, especially over the last few years. Being distinctively different in application, usage and implementation from Convolutional neural networks and Recurrent Neural Networks, it deserves a category of its own. It is an important part of the field of Machine Learning, with some of the most important milestones such as by OpenAI [1] coming from it. The uniqueness of the field, along with the discord in the development community owing to incomparable results that have occurred due to the absence of benchmarks, has been observed to be a major problem in the field, especially by OpenAI in [2]. This resulted in the inception of the gym framework.

The working of reinforcement learning, as shown in Fig 1 is based on the environment receiving its actions from the agent and the actions being executed to get some form of reward associated with the result for the same. The environment goes from state to state over time based on these actions it takes in these states. This results in a need for the algorithms to incorporate rewards that may be gained several states down the line.

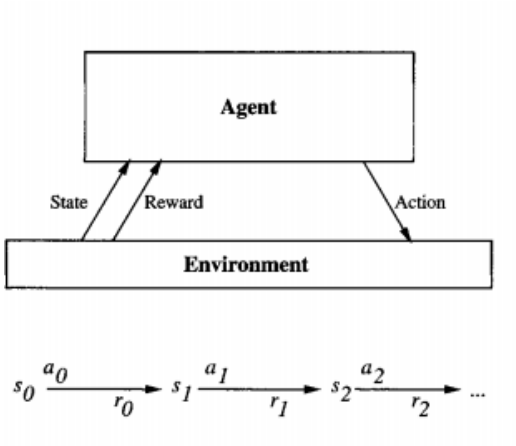

Fig. 1. The working of the Reinforcement Learning Model [3]

A. Q Learning

To understand the working of Deep Q learning, Vanilla Q learning needs to be looked at first. The algorithm of Q Learning is based on the construction of a Q Table [3].

Q Learning is based on a mapping of states to actions with rewards [4]. The highest Q value is the one that is always preferred. This is the best way to go in a deterministic environment since the solution to such a deterministic problem is bound to be the same on each iteration. The construction of the Q Table is the main aim of the algorithm. This table is used to keep track of the states, actions and the rewards expected on taking those actions. This creates a state action pair to Q value mapping, where the Q value could be considered a representation of the reward that is gained by the agent on taking said action.

The training of the Q Learning algorithm is based on populating the Q Table. Initially, the whole table can be initialized to zeros, or randomly initialized, the work presented here follows the latter.

B. Deep Q Learning

The core difference between Deep Q learning and Q Table based Q learning, or Vanilla Q Learning is how the Q Table is implemented. To explain in simplest terms, the Deep Learning

978-1-6654-0610-9/21/$31.00 ©2021 IEEE

approach implements the Q Table in the form of a neural network [5]. Thus, rather than a direct mapping, there is a function that connects the two in a way as to provide the reward in each case.

The implementation of Reinforcement learning networks is peculiar in a way due to the presence of the extra ambiguity that may not be present otherwise in the traditional models such as CNNs and RNNs. The peculiarity here is the way the code itself works, traditionally, there is only one model on which the forward and backward passes are made, whereas, in the case of Deep Q Learning, two models are needed. This is done to prevent losing out on the massive advantages that concepts such as batch normalization have on the training of a neural net, and the libraries that support the same in the same context.

C. Policy Gradient

Policy gradient methods are also based on model free reinforcement learning. This algorithm is very different from Deep Q learning, especially with respect to the equation involved in the training of the model. The basic idea behind policy gradients is that of the loss function, which provides direct updates to the policy itself, rather than certain values that need to be changed.

The idea behind this set of algorithms is that of function approximation [6], which helps in the derivation of the Policy gradient theorem. Among the set of algorithms that are based on the Policy gradient theorems, the REINFORCE algorithm [7] has been chosen for this work. This algorithm, since it is a part of the reinforcement learning suite, has the reward parameter, but differs in the fact that the decisions taken are not deterministic, but instead are stochastic. This makes any decision a probability-based decision rather than a fixed decision. This introduces randomness in the environment and thus makes results less reproducible.

D. Environment

The OpenAI gym [2] set of environments aimed at increasing the consistency of experiments in the Reinforcement learning field has been used here. Specifically, the Lunar Lander environment has been chosen due to its simplicity to understand while also adding enough states for the difference between the two algorithms to be visibly seen. Other envi- ronments such as the cartpole were also considered, but the simplicity of the environment demolished the possibility of any real comparison. An image of the same can be seen in Fig 2. The reward here depends on the distance from the landing pad, the velocity at which the landing occurs and the angular velocity of the craft, where the basic math behind this can be found at [8]

E. Proposed work

The lack of standard benchmarks being used in the Reinforcement learning field is a major problem that has already been identified in [2]. The same work also provided a solution to the same in the form of the OpenAI gym that aims at introducing a larger domain of the researchers in the field. The intention of this work is to address the above problem and compare the two algorithms with the above context.

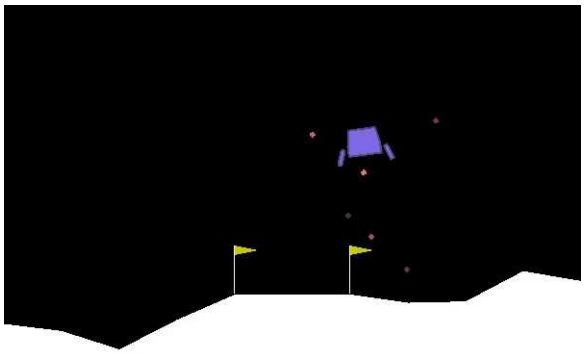

Fig. 2. The Lunar Lander environment from OpenAI

F. Organization of Paper

This paper first provides a background to the work done in the field previously under the Related works section. Following this, in the Implementation section, the working of algorithms has been explained along with necessary equations. The results and related graphs have been included under the Results section. Finally, the last section describes the conclusion that can be obtained from the work done.

II. Related Works

As the field progressed, the idea of Q learning also saw its inception and consecutive implementation in the traditional ta- ble like format. Q learning itself, today is a textbook algorithm that can be found easily in books such as [3] and [9]. The Deep Q learning model, however, being relatively challenging to implement can be seen more recently in [5]. The difference between the two as the challenges shall be discussed further. The effectiveness of Deep Q learning can be seen in [10] where a single model was able to beat most of the games on the Atari console, this was made possible by using a CNN that took the screen of the console as the input to the network and performed some action that was based on the Deep Q network described above. Followed by this, [11] showed how the Deep Q learning models can be used in 3D environments as well.

Policy Gradients were introduced in [6], and as previously mentioned, it works on the policy gradient method which is based on the policy gradient theorem as described there.

A comparison of Reinforcement Learning algorithms has been done before as well, in [12], using the same environment chosen in this work. The algorithms chosen in [12] were all model-free value-based algorithms, while the model-free policy based algorithms were sidelined. The paper also used the OpenAI Gym library introduced in [2].

The environments in Reinforcement learning have also been an area with lots of contributions. Algorithmic implementation of Reinforcement Learning algorithms has been around certain

978-1-6654-0610-9/21/$31.00 ©2021 IEEE

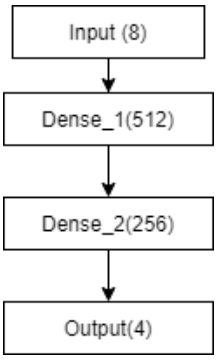

Fig. 3. The model that was used for Deep Q learning

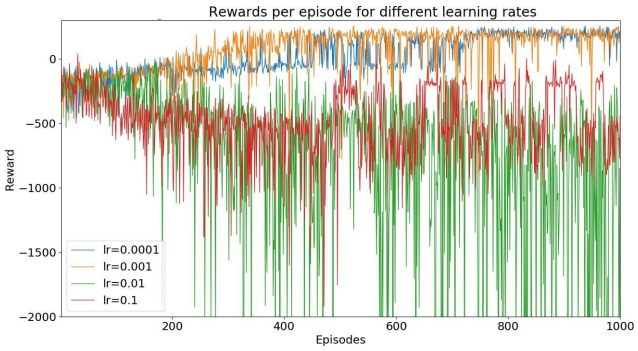

Fig. 4. The different learning rates and the graph of the rewards associated with them

games and other informal methods. The community has seen implementations to play Pong, Mario Kart, and almost every other popular game in the market while exhausting most of the retro games first. This creates incomparable results among the various implementations that have been done.

III. Implementation

The model has been implemented here using Keras and Tensorflow in Python. The environment is the Openai gym's Lunar Lander [2].

A. Deep Q learning

The implementation of Deep Q learning as shown in [5], needs one neural network that is updated every N steps, while also using a copy the network to have a prediction value that is needed. This is because the equation for the Q value is updated according to the bellman equation given below.

$$Q(s, a) = \sum_{s_t \in S, r \in R} p(s, r\, s, a)(r + \gamma \max_{t \in A(s_t)} Q(s, a)) \quad (1)$$

To explain the equation, each Q value depends on the set of Q values occurring after it, till the task is completed. This can be considered to be a chain of max Q values until the completion of the task where the reward on completion should be greater than the Q value part of the equation. This leads to a phenomenon similar to back-propagation, but up the Q Table. This chain can be considered real in the case of Vanilla Q learning, where there is a table that exists. The table being replaced by the neural network.

The important part of training any neural network is the tuning of the hyper-parameters [13]. The same has been described below, with a description of every parameter. The model that has been used in this work can be seen in Fig 3.

1) Learning rate: As in any traditional neural network, the Learning rate is an important part of the training of this neural network as well. The only use of this is in the fit method, which takes the Q value, expected Q value and fits over it with the mean squared error. Thus, a range of Learning rate values were tested with, to choose the best fit that worked with the data, as shown graphically in Fig 4

The graph in Fig 4 helps in the choice of the learning rate, with the most stable and highest rewards that are obtained. A lower reward as in the case of a learning rate of 0.0001 or 0.1 is not preferred, since it reflects the fact that the model is yet to solve the problem. Similarly, a higher variance that can be seen in the case of 0.001 is due to not enough exploration being done, and there being Q values calculated that are not the most optimal. Thus, 0.01 was chosen as the learning rate, since both of these problems are minimized from its plot, as a result, its model.

2) Epsilon Decay: Epsilon is a value that has been chosen to ensure that the model does not overfit over one particular deterministic route while training. Since this may hinder the progress and lead to the model getting stuck on local minima. The way in which epsilon values prevent the same is by adding a random chance to take an action at random rather than follow the Bellman equation (Eq 1). These are called exploitation and exploration, where exploring the environment is done due to the random actions taken, thus leaving as few nodes unvisited

978-1-6654-0610-9/21/$31.00 ©2021 IEEE

as possible, whereas exploiting is done using the Bellman equation. The randomness needs to reduce over the training period. This is done by taking another value called the epsilon decay less than 1 and multiplying it with the value of epsilon at every iteration. There are other ways to implement the same, but this is the one that was chosen here, drawing inspiration from some of the previous works.

The graph in Fig 5 describes the way in which the value of epsilon decay is chosen, a similar idea that was presented in the choice of the learning rate also applies here.

3) Gamma: Gamma is the value that takes care of how much the next state matters. This is seen even in the Bellman equation in the Bellman Equation. The reward is usually described for every state within every episode. Therefore, this value is what affects the change in Q values in the row of a table, or across an argmax layer, in the respective cases. This makes the value of gamma somewhat important, which leads to Fig 6.

978-1-6654-0610-9/21/$31.00 ©2021 IEEE

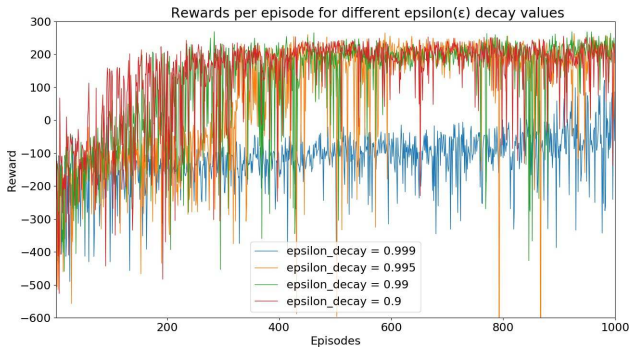

Fig. 5. The different values of epsilon decay and the graph of the rewards associated with them

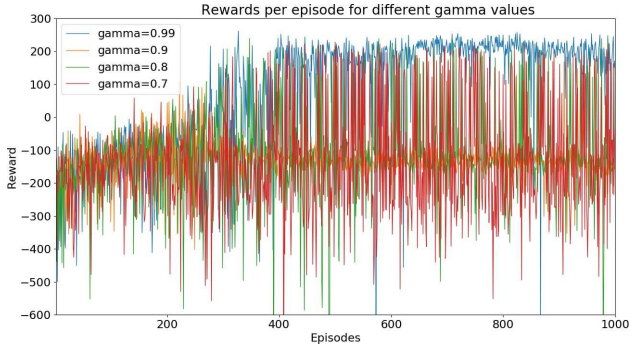

Fig. 6. The different values of gamma and the graph of the rewards associated with them

Again here as in the previous two cases, the value that is relatively stable, with better reward values can be chosen.

B. Policy Gradient algorithms

The algorithm here is based on the policy gradient theorem [6], the background of the algorithm lies in the area of it being possible to directly update the policy rather than update it using the values that come from it. This results in a long term effect that may result in changes that can be seen even before the episode ends.

The set of algorithms describes many simple and complex algorithms, among which, the most popular and most basic, is the REINFORCE algorithm [14]. The algorithms are based on the fact that the final result is based on a Value function V(θ) which gives a good measure of the performance measure. This can be a normalized form of the rewards obtained over the different states visited by the agent. The dependence of the value function based on the state of the environment is found. The state of the environment at various points is the variables in the environment. Here, they are the directional velocities and the position along the axes, along with the rotational angle. The policy function so described is now fit based on the gradient of the above loss, using gradient ascent [15]. Gradient ascent is a method where the total value of the function needs to be maximized as opposed to the usual gradient descent, where the same is minimized. The difference in equations is as shown here, with gradient descent being

$$\theta = \theta - \alpha(\delta\theta) \tag{2}$$

While the equation for gradient ascent is simply

$$\theta = \theta + \alpha(\delta\theta) \tag{3}$$

Since the modern libraries are not tuned to perform gradient ascent by default, the easy solution to this is to save the loss in its negative value rather than as the loss described by the equations. This enables using the libraries directly in place.

The advantage of this method over Deep Q learning is that the method is stochastic and not deterministic. This makes the system less reliable to always make the best decisions. The same has been shown experimentally as can be seen in the results of the work.

The hyper-parameters here only include the learning rate and the size of the model. This has simply been checked experimentally for the learning rate that gives the best reward rather than graphically. Since it was observed that the reward here was much lower than in the case of Deep Q learning in all the iterations, the size of the model was also fiddled with, with the best result being showcased below. The model, in this case, is similar to that in Fig 3, except the last layer is a softmax layer instead of a linear layer involving direct output activations.

IV. Results

Through the methods described above, the Deep Q learning model has a training accuracy as shown in Fig 7. The model gains a reward that is close to the final landing by epoch 500, with some exploratory paths leading to a lower loss. The model was seen to learn how to land at the positive peak of the graph.

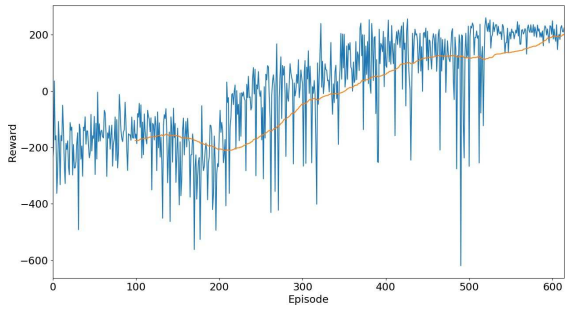

Fig. 7. The results for the Deep Q learning network on training

The REINFORCE model performs relatively worse on the same model. A graph has been chosen at about the same episode number as the DQN to see this difference visibly. However, the reward did not reach a stable value for at least 6

978-1-6654-0610-9/21/$31.00 ©2021 IEEE

times the length of this graph consistently. The variance of the result in the model is also quite extreme, which makes such a model unnecessary. This highly variant graph can be seen in Fig 8.

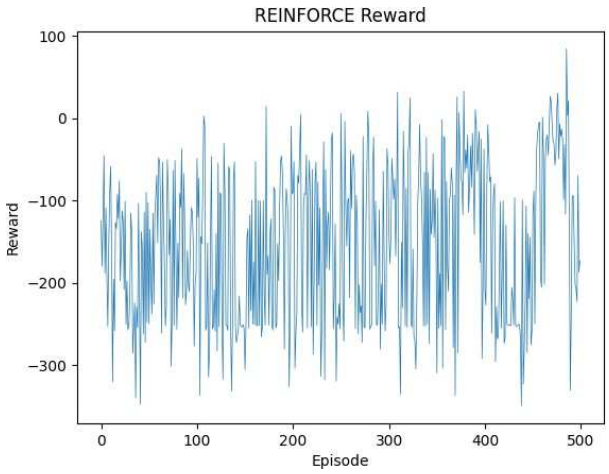

Fig. 8. The results for the REINFORCE model on training with the same model as Fig 3

However, on changing the architecture of the model itself, a somewhat better result was obtained, but when compared to the Deep Q Learning results, the results still have tremendous variance. The architecture of the model was changed so that the size of both hidden layers is 10 units, with the rest of the model architecture being just the same. This resulted in the result as shown in Fig 9.

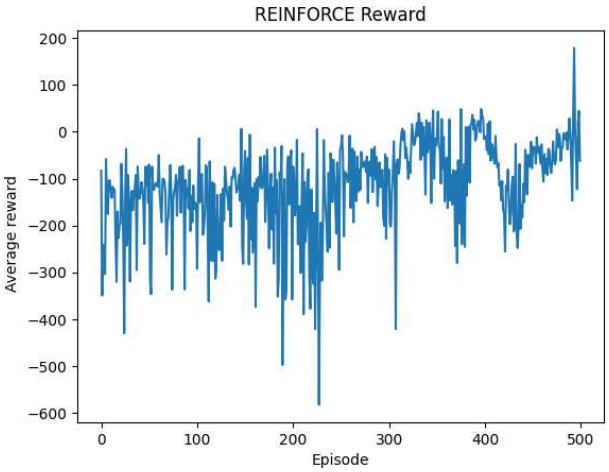

Fig. 9. The results for the REINFORCE model on training with a new model

V. Conclusion and Further Scope

Looking at the graphs and training, it seems pretty obvious that Deep Q learning although the older of the two algorithms is better off when it comes to episodic environments. The REINFORCE algorithm at its best barely corresponds to some of the graphs that the experiment plotted during the choice of the learning parameters. This is because an episodic environment such as LunarLander or any other game is likely to be deterministic rather than stochastic. This renders the stochastic policy gradient method disadvantageous since a deterministic environment is likely to have a better result with a non-stochastic policy. This being said, the validity of these results can only be limited to such environments. It could be possible that Value based methods will perform better than Policy Gradient based methods even in the real world under uncertainty. Since the same cannot be implemented in the deterministic environment, which cannot be covered under this experiment and is left for future work.

References

[1] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Jo´zefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," 2019.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[3] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[4] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning." *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *In Advances in Neural Information Processing Systems 12*. MIT Press, 2000, pp. 1057–1063.

[7] J. Peters, "Policy gradient methods," *Scholarpedia*, vol. 5, no. 11, p. 3698, 2010, revision #137199.

[8] M. S. Islam and I. M. Mehedi, "Landing trajectory generation and energy optimization for unmanned lunar mission," *Mathematical Problems in Engineering*, vol. 2021, p. 9902390, Jul 2021. [Online]. Available: https://doi.org/10.1155/2021/9902390

[9] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[11] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," *CoRR*, vol. abs/1609.05521, 2016. [Online]. Available: http://arxiv.org/abs/1609.05521

[12] S. Gadgil, Y. Xin, and C. Xu, "Solving the lunar lander problem under uncertainty using reinforcement learning," 2020.

[13] X. Yu, "Deep q-learning on lunar lander game," 05 2019.

[14] J. Zhang, J. Kim, B. O'Donoghue, and S. P. Boyd, "Sample efficient reinforcement learning with REINFORCE," *CoRR*, vol. abs/2010.11364, 2020. [Online]. Available: https://arxiv.org/abs/2010.11364

[15] H. Kimura and S. Kobayashi, "Reinforcement learning for continuous action using stochastic gradient ascent," 1998.

978-1-6654-0610-9/21/$31.00 ©2021 IEEE