Balancing an Inverted Pendulum using Reinforcement Learning

Sai Prajwal Kotamraju[#1], Rakshith Subramanyam[#1], Sai Pratyusha Gutti[#1]

EEE, Arizona State University

[1]skotamra@asu.edu, [1]rsubra17@asu.edu, [1]sgutti@asu.edu,

***Abstract*— In this project, an inverted pendulum is balanced using Reinforcement learning. Various Reinforcement Learning algorithms were studied out of which three algorithms have been implemented to solve the given problem statement. Firstly, the Policy Gradient method in OpenAI environment is illustrated for noise-free and noisy cases. For the same noise cases, Q-learning has been implemented in Matlab environment. Lastly, the cart-pole model was built in Robotic Operating System (ROS) to achieve real-time implementation of a policy gradient learning algorithm. Implementations show that algorithms give 100 percent success rate with trials much less than 1000. The results have been elucidated and the corresponding tables with required trials have been included.**

***Keywords*— Reinforcement Learning, Policy Gradient Method, Q-learning, OpenAI, Matlab, Robotic Operating System, Competing for category 3, 4, 5 (ROS and Matlab)**

I. Introduction

Reinforcement Learning (RL), one of the fields of Machine Learning has been around since 1950s producing interesting applications over the years but was not popular enough until 2013. This was when the startup, DeepMind, demonstrated the possibility of outperforming humans in playing any Atari game from scratch. Without any prior knowledge of rules of the game, the objective was achieved by using raw image as inputs. From this high dimensional sensory input, control policies were learnt directly using RL. Hence from [1], Reinforcement Learning is defined as a software agent making observations and taking actions within an environment for which it receives rewards in return. The learning is carried out in a way that maximises the expected long-term rewards. The algorithm used by the software agent to determine its action is its policy.

Use of exploration and exploitation in RL achieves a balance in the learning unlike in standard supervised learning. Few of the RL techniques such as Q-learning do not require model to find an optimal action-selection policy. Reinforcement learning enjoys multiple advantages in Machine Learning in addition to the ones mentioned above. However, the challenges faced by RL include learning from a signal that is frequently sparse, noisy and delayed, dealing with sequences of highly correlated states, changing of data distribution as the algorithm learns new behaviours etc [2]. The limit cycle on the control of the inverted pendulum might converge the Q-value to zero and destroy the stabilization of the optimal control policy. RL finds many applications such as game theory, control theory, genetic algorithms, information theory, statistics, swarm intelligence, etc. [3].

The problem statement of the project is elucidated as follows. A cart-pole setup with the given parameters in [4] is to be built and the pendulum/pole is to be balanced within the required boundary conditions for 600,000 time steps where each time setup is 0.02 seconds within trials for 100 runs. The problem statement has been explained in detail along with parameters' definition under Section III. The cart-pole setup during the learning process in Matlab environment is as shown in figure 1.

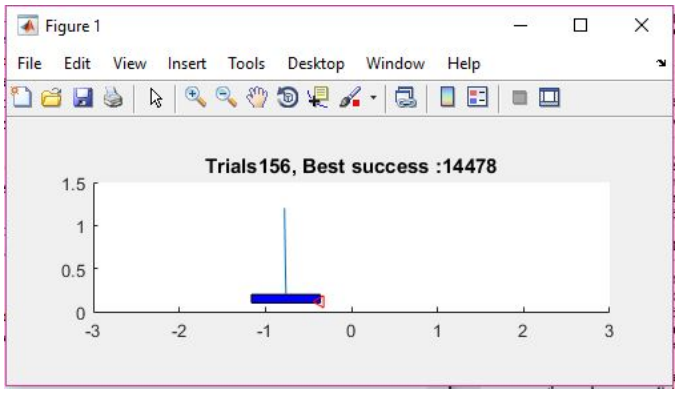

Fig. 1. Cart-pole setup in Matlab

The organisation of this report is as follows. Section II describes the approach used for the experimentation in OpenAI, Matlab, ROS environment. Section III covers the simulation procedure and the results obtained for each method. Section IV concludes the project report.

II. Approach

Three approaches were implemented for balancing the inverted pendulum using Reinforcement learning. The first approach is by using Policy gradients with Neural Network Policy in OpenAI environment. The second approach was to implement Q-learning algorithm using Matlab. Another approach was to build the model on Robotic Operating System (ROS) and train it using Policy Gradient Method. The

[#1] All the authors did equal amount of work

implementation of RL in ROS model is in progress as it's involving real time constraints.

A. Policy Gradients using Neural Network Policy

Policy Gradient algorithms optimize the parameters of the policy by calculating and following the gradients towards higher rewards. One of the popular variants of the Policy Gradient algorithms are Reinforce algorithms, introduced by Ronald Williams in 1992 [5]. The steps involved in this method are as follows:

- Neural Network Policy plays the game several times and at each step the gradients are computed which would make the chosen action even more likely.
- Once several episodes are run, the action scores are computed.
- If the action score is positive, the gradients which have been computed are applied in order to make the chosen action even more likely in future. If the action score is negative, opposite gradients are applied to make the corresponding action less likely in future. The solution is to apply each gradient vector with corresponding action's score.
- Finally, the mean of all the resultant gradient vectors is computed and is used to perform a gradient descent step.

In order to implement the policy gradients method, a policy maker is required which, in our case, is a Multi-Layer Neural Network with the following architecture.

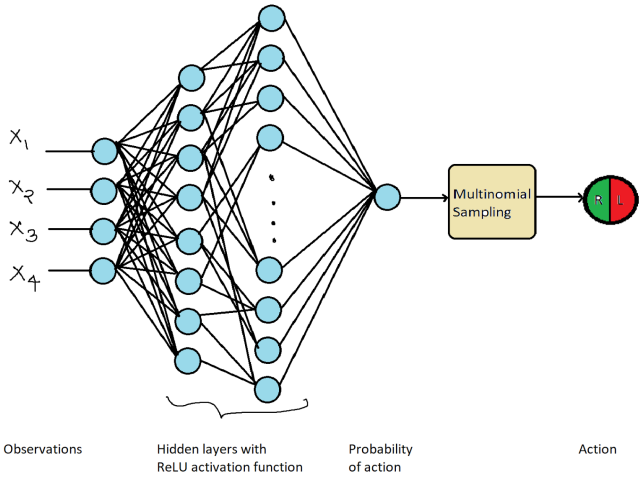

Fig. 2. Multi-Layer Neural Network used as a Policy maker with two hidden layers with 8 and 16 neurons respectively.

As shown in the above figure, Fig.2., a Multi-Layer NN with observation vector (position, linear velocity, angle, angular velocity) as input is used as a policy maker in order to apply the Policy Gradient variant discussed above. Note that the probability of action at the output neuron (due to sigmoid activation function) is fed to the Multinomial sampling block instead of hard thresholding to perform the action. This helps the Neural Network policy to explore the policy space once in awhile depending upon the probability of action. For example, if the probability of action is 0.96, the multinomial sampling block given an output 1 with a probability of 0.96. So, it has a 0.04 probability to output 0 so that the agent could explore the policy space.

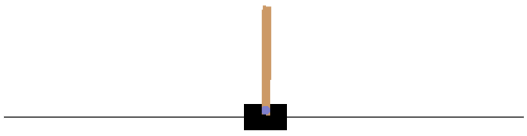

Fig. 3. Cart pole simulation in OpenAI environment.

B. Q-Learning Algorithm

Q-learning is one of the important reinforcement algorithms based on instantaneous strategy and model-free technique. Without the requirement for adaptation, this technique can handle problems with stochastic transitions and rewards. It is an algorithm used to estimate the optimal state-action values which are commonly referred to as the Q-values. The Q-value is the sum of future discounted rewards expected by the agent on an average after it reaches the state s and chooses an action a. This happens before it sees the outcome of this action, assuming that it acts optimally after that action. This is described in equation (1) from [1]

$$Q_{k+1}(s,a) = \sum_{\hat{s}} T(s,a,\hat{s}) \left[R(s,a,\hat{s}) + \gamma . max_{\hat{a}} Q_k \ (\hat{s},\hat{a}) \right] \ \forall \ (s,a) \quad (1)$$

where $T(s,a,\hat{s})$ defines the transition probabilities from state s to $\hat{s}$ given that the agent chose action a, $R(s,a,\hat{s})$ defines the reward that the agent gets when it transitions from state s to $\hat{s}$ given that the agent chose action a, γ is the discount rate. It is required to take into account that the tuning of parameters is important to achieve better results.

The Q-values are initialized to zero which are updated using the above equation. Each state has to be experienced multiple times to find the transition probabilities and at least once to know the rewards. One of the available exploration policies like the ϵ- greedy policy is used to explore the Markov

decision process. Once the optimal Q-values are obtained with the optimal policy, the action with the highest Q-value is chosen. It is known that the agent has only partial knowledge of the Markov Decision Process (MDP). Because of this, the algorithm keeps track of the rewards it expects along with the running average of the rewards r for every state-action pair (s, a) .As the target policy acts optimally, hence the maximum of the Q-value is selected for the next state. The Q-learning algorithm. Thus the Q-values can be updated using the below equation

$$Q_{k+1}(s,a) = (1-\alpha)\,Q_k(s,a) + \alpha(r + \gamma.max_{\hat{a}}\,Q_k \quad (\hat{s},\hat{a})) \quad (2)$$

where α is the learning rate, r is the reward.

$\alpha = 0$ will result in the agent not learning anything and $\alpha = 1$ implies that the agent considers only the most recent information. $\gamma = 0$ will give importance only to the present rewards whereas γ close to 1 will take into account long-term high rewards.

C. Policy Gradients using Neural Network Policy in ROS

ROS is a robotics middleware developed in Stanford University. The platform enables cross coupling between robots in real world with robots in simulations. These robots are created with physics constraints which makes them a very close approximation of real world model. For this project we thought of using this platform to collect training data as the program written for the virtual environment is the same as that of a real world one so it will be very analogous to a real world training data.

We created an environment emulating the cart pole in ROS and visualized it in gazebo. An URDF(Universal Robot Description format) file was written for the cart pole complying with the parameters mentioned in [4]. This was then exported by a launch file which opened a new GUI. Then an python program was developed to interface the data that was given by the cart pole in the virtual environment with the user. This program takes in action and environment state as input parameters and would return the position of cart, velocity of cart, angle of pole, angular velocity of pole and a status string. If the cart and the pole does not comply with the constraints as set by [4] the string would assume the value 'Done' and program would reset the environment.

III. Simulation and Results

The cart-pole system used in the current study is same as the one mentioned in [4]. The governing equations for the cart-pole problem are as follows:

$$\frac{d^2\theta}{dt^2} = \frac{g\sin\theta + \cos\theta[-F - ml\dot{\theta}^2\sin\theta + \mu_c\,\mathrm{sgn}(\dot{x})] - \frac{\mu_p\dot{\theta}}{ml}}{l\left(\frac{4}{3} - \frac{m\cos^2\theta}{m_c+m}\right)} \quad (3)$$

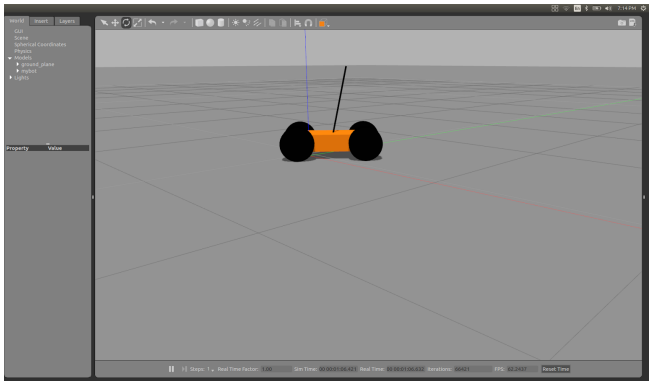

Fig.4. ROS implementation of cart pole.

$$\frac{d^2x}{dt^2} = \frac{F + ml[\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta] - \mu_c\mathrm{sgn}(\dot{x})}{m_c+m} \quad (4)$$

Where,
$g = 9.8$ m/s^2 ; Acceleration due to gravity
$m_c = 1.0$ Kg ; Mass of the cart
$m = 0.1$ Kg ; Mass of the pole
$l = 0.5$ m ; Half-pole length
$\mu_c = 0.0005$; Coefficient of friction on track
$\mu_p = 0.000002$; Coefficient of friction of pole on track
$F = \pm 10$ N ; Force applied to cart's center of mass

The angular velocity of the pole and the angle made by the pole with respect to vertical axis are calculated using the linear approximations from the angular acceleration using the time step value of 0.02 seconds. The same applies to the calculation of linear velocity and the displacement of the cart. They are calculated by linear approximations from linear acceleration of the cart. According to [4], 0.02 sec was the time step and a trial is defined as a complete process from start to fall. When the pole falls outside the range of [-12 deg, 12 deg] and beyond [-2.4 m, 2.4 m] in reference to the central position on the track, it is considered as end of the trial. A run consists of maximum 1000 trials and it is considered successful if it reaches 60000 time steps within the 1000th trial.

A. OpenAI Environment

The policy gradient algorithm was trained with a learning rate of 0.1 and a discount rate of 0.97, the model was trained on twenty runs to balance the pole for 60,000 time steps. The testing was carried out for noise free and required noise cases. The results, after the tuning the parameters, are as documented in Table 1. The folder with the code and the game environment have been attached along.

Noise Type	Success Rate	No of Trials (20 runs)
Noise free	100 %	35.65
Uniform 5 % Actuator Noise	100 %	34.6
Uniform 10 % Actuator Noise	100 %	49.6
Uniform 5 % Sensor Noise	100 %	32.5
Uniform 10 % Sensor Noise	100 %	44.1
0.1 Variance Gaussian sensor Noise	100 %	40.6
0.2 Variance Gaussian sensor Noise	100 %	106.9

Table I. Average results.Cart pole being balanced for 60000 time steps for 20 trials

B. Matlab Environment

Using the approach elucidated in Section II. B., the required model was designed. According to the equations (3) and (4), the model was built and for each iteration the Q-values were updated. For the noise-free case, with parameters of learning rate = 0.3 and discount factor = 0.9, the success rate was 100% for average number of trials being 42 for 60000 time steps. The results for every run is as shown in figure 5 and tabulated in table II.

Noise Type	Success Rate	No of Trials (20 runs)
Noise free	100%	42

Table II. Average results.Cart pole being balanced for 60000 time steps for 20 trials

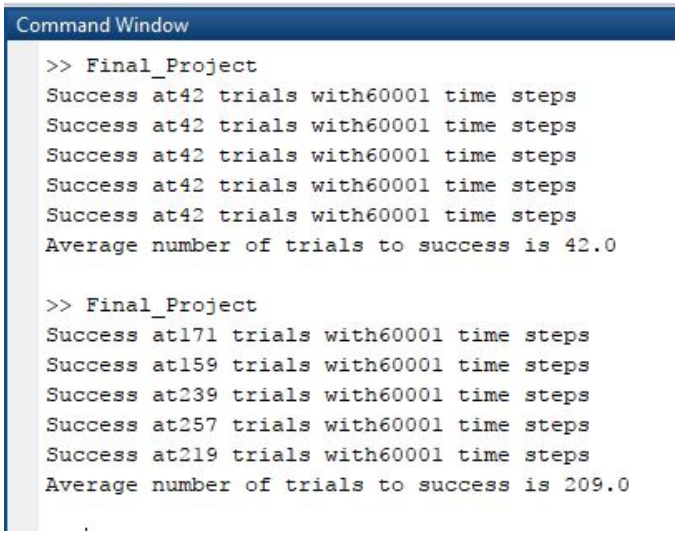

Fig.5. Matlab results for noise-free case

C. ROS Environment

Training the policy gradient algorithm is computationally very heavy and time consuming as the training happens in real time and the rendering of the graphics slows down the training. We are currently working on making the environment train much faster and use less computation.

IV. Conclusion

Various RL algorithms were studied and analyzed as to which algorithm is to be implemented to achieve the project objective of balancing the inverted pendulum for 60000 time steps. It was inferred to implement Policy Gradient method and Q-learning technique and the results are tabulated. It took 35 trials on an average to balance the inverted pendulum in a noise free case in OpenAI environment. In addition, the implementation was also done in Matlab using Q-learning and ROS environment. It was found that it took 42 trials on an average for a success rate of 100%. Implementation of RL in ROS environment is in progress.

References

[1] Aurélien Géronn, “Hands-On Machine Learning with Scikit-Learn & TensorFlow”, 1st Edition, California, O’Reilly, 2017, pp. 441 – 442.

[2] V. Mnih et al (2013, Dec 19). “Playing Atari with Deep Reinforcement Learning” [Online]. Available: https://arxiv.org/pdf/1312.5602v1.pdf

[3] Wikipedia, Reinforcement learning [Online]. Available: https://en.wikipedia.org/wiki/Reinforcement_learning

[4] Jennie Si, Yu-Tsung Wang, “On-Line Learning Control by Association and Reinforcement”, IEEE TRANSACTIONS ON NEURAL NETWORKS, Vol. 12, No. 2, March 2001.

[5] R. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning", 1992.