Snake Game and Robotic Arm Object Manipulation Implementation Using Reinforcement Learning

Aadhar Bansal, Pranav K Nair

Northeastern University, Boston, MA

Abstract

The project deals mainly with the two tasks. First, To conduct experiments towards different approaches to solving the famous 2D Snake Game. It has been observed that Reinforcement Learning provides better learning outcomes compared to traditional Machine Learning algorithms. Our approach includes Deep Q-Learning and Double Deep Q-Learning. Secondly, The Robotic Arm object manipulation has to do tasks that are ubiquitous to our environment but this requires smooth robotic joints angle transition and object-end effector distance accuracy. We employ the PPO algorithm for this task. An important feature of the PPO algorithm is that the changes made to the policy are compared so as to make sure there is a controlled amount of change. This is quite important in real-world scenarios, especially in the medical field.

I. Introduction

A. Snake

Any game to play efficiently needs experience as they are challenging but these games are easy to formalize. This has been a popular field for the Artificial Intelligence and Machine Learning research community. The developing community has tried to create an agent which can efficiently learn to traverse through the game after gathering some experience like a real human player and to do this Reinforcement Learning has been a prominent choice. Reinforcement Learning for game solving became popular after the Defeat of the World's best Go player by Deep Mind's AlphaGo. There can be different techniques and algorithms to solve these games like Dynamic Programming but it could only solve for the small number of states but not for the games which have a large complex structure and a large number of states.

The Nokia 6110 mobile phone popularized the Snake game all over the world. It is a relatively simple game played by a single player who controls the direction of movement of the snake that tries to eat the food items (typically an apple) by running into them head-on. The "apple" would emerge in a random location on the screen. After eating an item snake's grows a unit of length. This maneuvering becomes progressively more difficult.

This report implements two different techniques Deep Q-Learning and Double Deep Q-Learning on the Snake Game environment which has been implemented using the PyGame library and the results of the Game Scores have been analyzed.

B. Robotic Arm

Dynamic work conditions should not be a problem for contemporary robotic operations. They ought to be capable of independently adapting to new tasks, movements, environmental changes, and disturbances. The robots have challenges from smooth task completion in real time to adjusting to a new environment and being able to work autonomously with accuracy. They should be able to work on different platforms and environments without complex tuning every time. This makes even a simple task quite challenging and tedious to achieve.

The high number of degrees of freedom of modern robots leads to large dimensional state spaces, which are difficult to learn: example demonstrations must often be provided to initialize the policy and mitigate safety concerns during training. Moreover, when performing dimensionality reduction, not all the dimensions can be fully modeled: an appropriate representation for the policy or value function must be provided in order to achieve training times that are practical for physical hardware.

In the Robot Arm environment being used, there exists a robot arm positioned near a table. This arm is allowed to move freely within the three-dimensional space above the table so as to reach its target position. The robot arm has a gripper at its end and its goal is to come in contact with an object. While this may seem like a simple task, precision is of utmost importance when taking into consideration the potential applications. For example, in robotic surgery, the precision of an incision or any such act is imperative.

This paper implements the Proximal Policy Optimization, or PPO, is a policy gradient method for reinforcement learning for robotic arm manipulation.

II. Background

In this Section, we explore the prerequisite knowledge required to understand the implementation, results, and analysis.

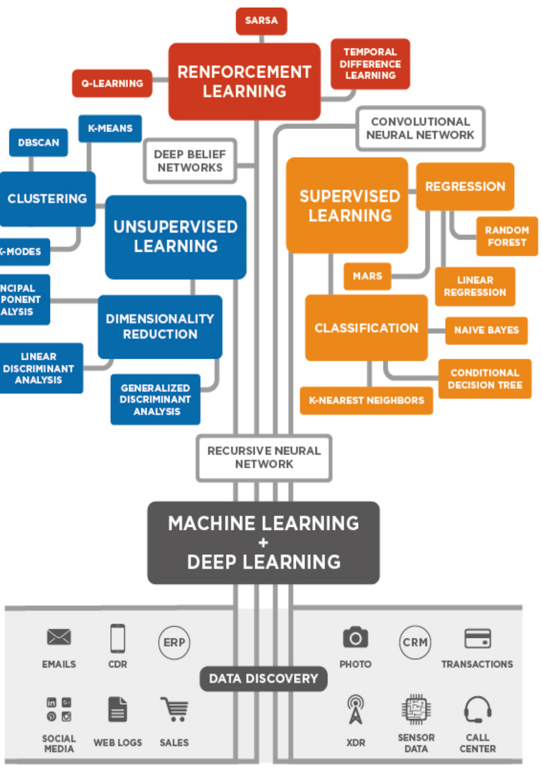


Figure 1: Machine Learning can be classified into three subcategories, Supervised Learning, Unsupervised Learning and Reinforcement Learning.

.

A. Markov Decision Process

In the Reinforcement Learning problem, there is a decision-maker called an agent who interacts with the environment. In return, the environment responds to the agent with a reward and a new state based on the actions of the agent. In RL we teach the agent to interact based on the reward which could be negative or positive and to formalize this we need MDP.

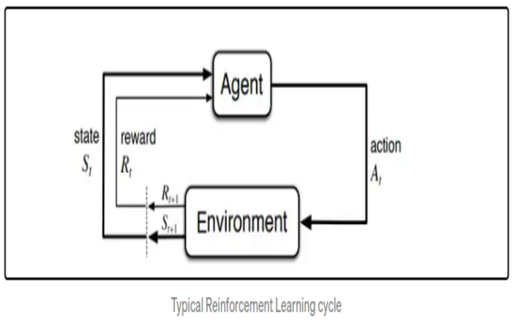


Figure 2: MDP Process

B. Episodic and Continuous Tasks

Episodic Tasks: These are the tasks that have a terminal state (end state).We can say they have finite states. For example, in racing games, we start the game (start the race) and play it until the game is over (race ends!). This is called an episode. Once we restart the game it will start from an initial state and hence, every episode is independent.

Continuous Tasks : These are the tasks that have no ends i.e. they don't have any terminal state.These types of tasks will never end.For example, Learning how to code!

C. Model-Free and Model-Based RL

Model-based RL uses experience to construct an internal model of the transitions and immediate outcomes in the environment. Appropriate actions are then chosen by searching or planning in this world model.

Model-free RL, on the other hand, uses experience to learn directly one or both of two simpler quantities (state/ action values or policies) which can achieve the same optimal behavior but without estimation or use of a world model. Given a policy, a state has a value, defined in terms of the future utility that is expected to accrue starting from that state.

Model-free methods are statistically less efficient than model-based methods, because information from the environment is combined with previous, and possibly erroneous, estimates or beliefs about state values, rather than being used directly.

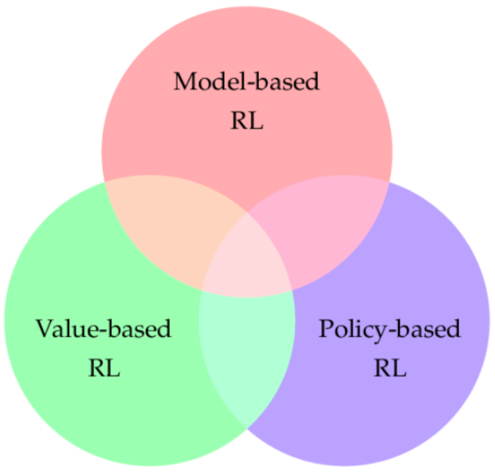


Figure 3: Reinforcement Learning Classification

D. Neural Network

A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain. It is a type of machine learning process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain. It creates an adaptive system that computers use to learn from their mistakes and improve continuously. Thus, artificial neural networks attempt to solve complicated problems, like summarizing documents or recognizing faces, with greater accuracy.

Each node, or artificial neuron in an artificial neural network, connects to another and forms a network of neurons and this has an associated weight and threshold. A node in the Artificial Neural Network is said to be activated if its value is above the specified threshold value and an activated node transmits the data to its immediate next layer, and if it's not activated, no data is transmitted to the adjacent layer of the network. The representation of a simple Artificial Neural Network can be seen in Figure 1. In an Artificial Neural Network (ANN), for each input x i, we multiply the input value x i with weight w i and sum all the multiplied values.

$$\sum = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

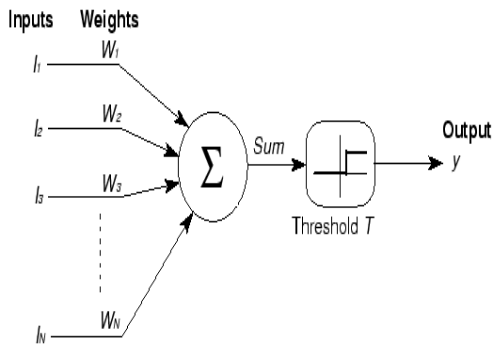


Figure 4: Artificial Neuron

D.1. Layers of Neural Network

A layer is made of perceptrons which are linked to the perceptrons of the previous and the next layers if such do happen to exist. Every layer defines it's own functionality and therefore serves its own purpose. Neural networks consist of an input layer(takes the initial data), an output layer(returns the overall result of the network), and hidden layers.

In order for the network to be able to learn and produce results each layer has to implement two functions — forward propagation and backward propagation.

Forward propagation is only responsible for running the input through a function and return the result. No learning, only calculations. Backpropagation does two operations: Update the parameters of the layer in order to improve the accuracy of the forward propagation method and Implement the derivative of the forward propagation function and return the result.

D.2. Cost Function

cost functions are used to estimate how badly models are performing. Put simply, a cost function is a measure of how

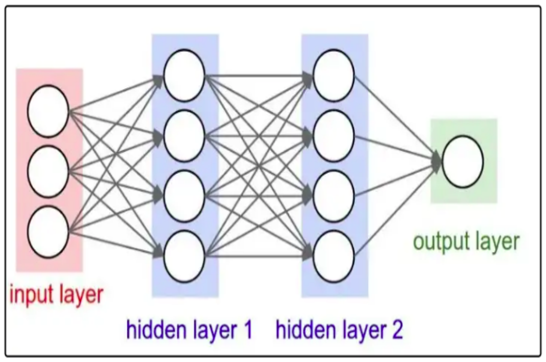


Figure 5: Layers in Neural Network

wrong the model is in terms of its ability to estimate the relationship between X and y. This is typically expressed as a difference or distance between the predicted value and the actual value. The cost function can be estimated by iteratively running the model to compare estimated predictions against true values. Gradient descent is an efficient optimization algorithm that attempts to find a local or global minima of a function.

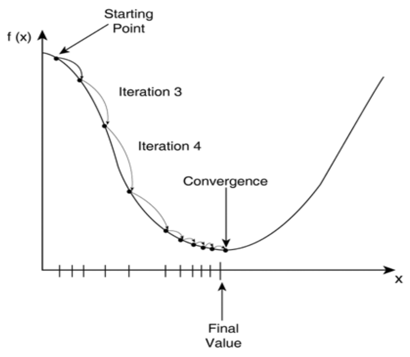


Figure 6: Gradient Descent

E. Robotic Arm

Robot arms are ideal for operations which are repetitive, consistent and require a very high degree of accuracy, as well as for applications in which a human worker might struggle to perform safely.

Robotic arms are fast, accurate and reliable, and can collectively be programmed to perform an almost infinite range of different operations. The dramatic reduction in buy-in costs for industrial robotic arms over the past decade has seen them rise to far more widespread use today than ever before - whether desktop-mounted or installed as part of a high-volume production line, robotic arms are now commonly found across a broad range of industries and sectors

F. Inverse Kinematics

Kinematics is the science of motion. In a two-joint robotic arm, given the angles of the joints, the kinematics equations give the location of the tip of the arm. Inverse kinematics refers to the reverse process. Given a desired location for the tip of the robotic arm, what should the angles of the joints be so as to locate the tip of the arm at the desired location. There is usually more than one solution and can at times be a difficult problem to solve.

This is a typical problem in robotics that needs to be solved to control a robotic arm to perform tasks it is designated to do. In a 2-dimensional input space, with a two-joint robotic arm and given the desired coordinate, the problem reduces to finding the two angles involved. The first angle is between the first arm and the ground (or whatever it is attached to). The second angle is between the first arm and the second arm.

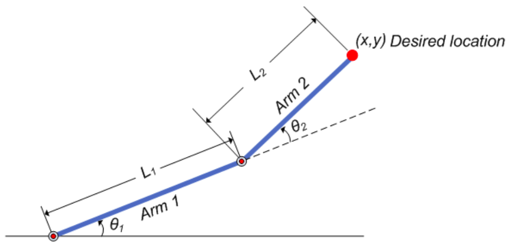


Figure 7: Two Joint Robotic Arm

III. Methodology

A. Snake

The base approach we chose is to use the Deep Q-Networks as they have been proved efficient for playing Atari games(Mnih and Kavukcuoglu, 2013 [1]) and other games such as Cartpole (Roibu, 2019 [2]). We have also implemented the Double Deep Q-Network Algorithm to compare with the DQN results. All the experiments are conducted on a custom environment created using the 'PyGame' package.

A.1 Mathematical Representation of the Snake

- Abstractly, each step of the Snake game could be considered as finding a self-avoiding walk (SAW) of a lattice in R^1 A Snake game can mathematically represented as
  - The m × n game board is represented as G = (V, E), where V = vi is the set of vertices, where each vertex corresponding to a square in the board, and E = {eij - vi is adjacent to vj}.
  - The snake is represented as a path {a1, · · · , ak}, where a1, ak are the head and tail, respectively.
- (Viglietta(2013))proved that any game involving collectible item, location traversal and oneway path is NP-hard. Snake is a game involving traversing a one-way path, i.e., the snake cannot cross itself. Thus, if we have no prior information about the sequence of food appearance, picking every single shortest SAW for each step in an episode is NP-hard.

A.2 Reduction and Selection of State and Action Space

For successful implementation and analysis of the reinforcement Learning algorithms to learn to play this game. One of the main obstacles is the huge state space. If we take the state space of all the game screen cell it would become n*n. Naively, using the exact position of the snake and food, each cell could be parameterized by one of the four conditions: contains food, contains the head of the snake, contains the body of the snake, blank. By simple counting principle the size of the state space (S) will become huge and immense when n is large and learning in state space of such size is infeasible for efficient computation. We chose the 11 states space as the input for the implementation as follows.

- Information about immediate danger any of the directions, [danger right, danger forward, danger left].
- the direction of the snake relative to the game at a particular point of time, [direction left, direction right, direction up, direction down].
- Relative position of the snake with respect to food, [food left, food right, food up, food down]
- State Space : { danger right, danger forward, danger left, direction left, direction right, direction up, direction down, food left, food right, food up, food down }.
- Action Space: { Left, Right, Straight or Forward }
- Rewards:
  - Ate Food: +10
  - Terminate or game Over: -10
  - else: 0
- Figure 8 can be defined as [0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0].

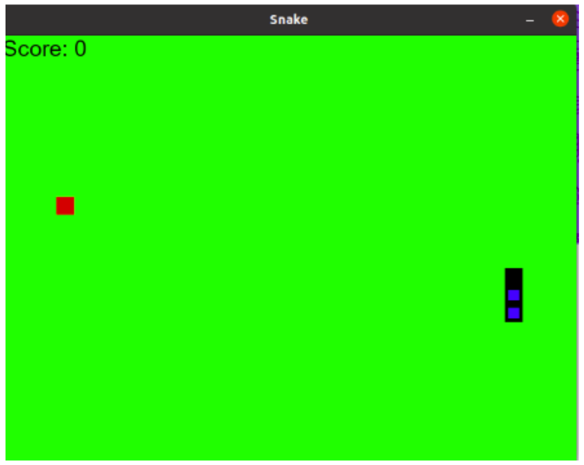


Figure 8: Snake Environment at a particular State space

A.3 Reinforcemt Learning algorithms: Q-Learning

In Q-learning, an agent tries to learn the optimal policy from its history of interaction with the environment. A history is defined as a sequence of state-action-rewards:

$$< s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \cdots >$$

For the Snake game, the process is indeed a Markov Decision Process, and the agent only needs to remember the last state information. We can define the Snake game as in (state, reward and next action .....).

In Q-learning, which is off-policy, we use the Bellman equation as an iterative update

$$Q_{i+1}(s, a) = E_{s' \sim \epsilon}\{r + \gamma \max_{a'} Q_i(s', a' | s, a)\}$$

Since the distribution and transition probability is unknown to the agent, in our approach we use a neural network to approximate the value of the Q-function. This is done by using the temporal difference formula to update each iteration's estimate as:

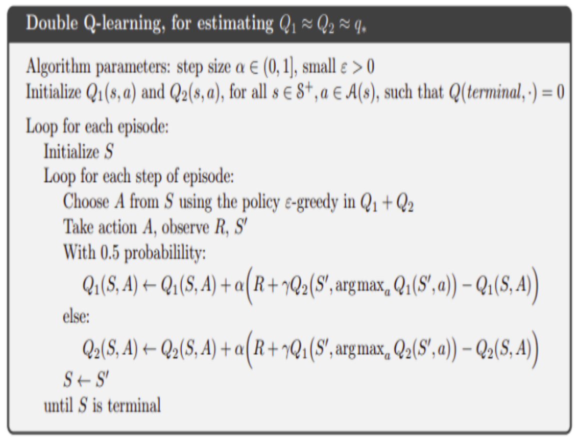
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \max_{a'} Q(s',a') - Q(s,a))$$

A.4 Deep Q-Leaning Network

DQN is a model-free Reinforcement Learning algorithm that uses Deep Learning. It is an algorithm that uses Q-Learning to learn what is the best action to take in a given state and it uses Deep Learning to estimate the Q value function. For environments with large and complex state spaces, simple Q-Learning cannot be used as it is too much information to compute. Hence we include Neural Networks to assist with the process. To break it down more simply, in Q-learning, a memory table Q[s, a] is made to store Q-values for all the possible combinations of s and a (i.e. the state and action). The agent learns a Q-Value function, which determines the expected total return in a given state-action pair. The agent thus has to act in a way that maximizes this Q-Value function.

A.5 Double Q-Learning

According to the optimal policy in Deep Q-Learning, the agent tends to take the non-optimal action in any given state only because it has the maximum Q-value. Such problem is called the overestimations of the action value (Q-value). To prevent this, Double Q-Learning can be used. Double Q-Learning uses two different actionvalue functions, Q and Q‘, as estimators. Even if Q and Q‘ are noisy, these noises can be viewed as uniform distribution. Q function is for selecting the best action a with maximum Q-value of next state. Q‘ function is for calculating expected Q-value by using the selected action a from Q. Double Q-Learning implementation with Deep Neural Networks is called Double Deep Q-Networks (Double DQN). Double DQN uses two different Deep Neural Networks: Deep Q-Network (responsible to calculate Q) and Target Network (responsible to calculate Q‘). This method is particularly useful to avoid those situations where the discrepancy between the neural networks causes them to recommend totally different actions given the same state, thus, reducing harmful overestimations and improving the performance



```
Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q_1(s,a)$ and $Q_2(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize $S$
  Loop for each step of episode:
    Choose $A$ from $S$ using the policy $\varepsilon$-greedy in $Q_1 + Q_2$
    Take action $A$, observe $R$, $S'$
    With 0.5 probabilility:
      $Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$
    else:
      $Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$
    $S \leftarrow S'$
  until $S$ is terminal
```

B. Robotic Arm

RL formulated around the basic MDP appears to be intuitive and straightforward;MDP appears to be intuitive and straightforward. However, robotic training has the difficulties which are high dimensionality, continuous action space, long training time.

As the Robotic arm has 6 Degrees of Freedom, which is the requirement to reach the point in a space but this freedom of movement also increases the complexity of the system as this requires increasing parameters to control like velocity of each joint. The Robot also operates in a continuous complex environment. This allows the agent to have continuous State space with an indefinite number of states rather than deterministic state space. The choice to limit the state space and action choice are done on the account of the results needed.

Many algorithms have been development for the simple agents and environment but robotic arm manipulation is more complex and complex reward function for the smooth transitioning and to reduce training time.

- State Space is defined as the Coordinates (x1,y1,z1) of end-effector, coordinate of object (x2, y2, z2)
- state space: { x1, y1, z1, x2, y2, z2}
- Reward :{ difference of (Equilidean distance between the current endeffector and object position) and (Equilidean distance between the previous end-effector position and object position)}
- object position does not change while in an episode.
- Action: step to reach near to the object position.

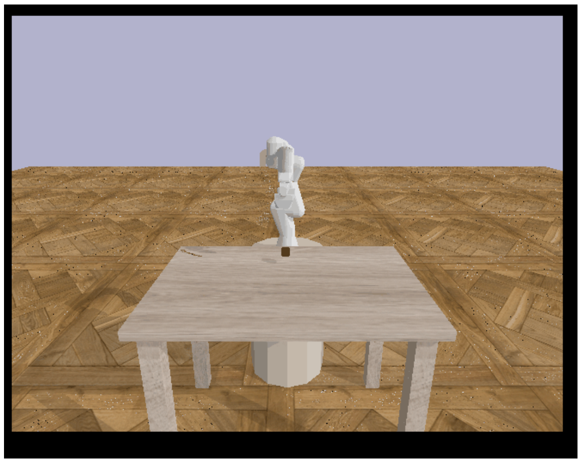


Figure 9: Robotic Arm with object

B.1. Policy Gradient Methods

Policy gradient is a type of Reinforcement Learning technique that relies upon optimizing parametrized policies with respect to the expected return by gradient descent. The policy gradient methods target modeling and optimizing the policy directly. The policy is usually modeled with a parameterized function with respect to θ, π_θ(a/s). The value of the reward function depends on this policy and then various algorithms can be applied to optimize θ for the best reward.

The most common used Gradient estimator has the form:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

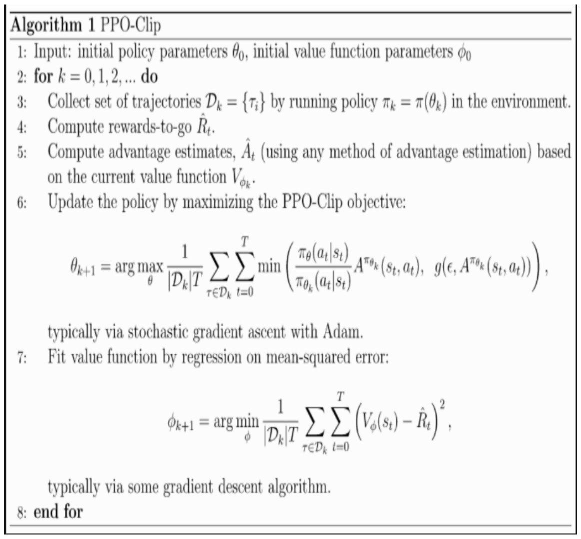
where π_θ is a stochastic policy and $\hat{A}_t$ is an estimator of the advantage function at timestep t. Here, the expectation $\hat{E}_t$ [. . .] indicates the empirical average over a finite batch of samples, in an algorithm that alternates between sampling and optimization. Implementations that use automaticdifferentiation software work by constructing an objective function whose gradient is the policy gradient estimator; the estimator $\hat{g}$ is obtained by differentiating the objective.

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right].$$

While it is appealing to perform multiple steps of optimization on this loss L^{PG} using the same trajectory, doing so is not well-justified, and empirically it often leads to destructively large policy updates

B.2. Proximal Policy Optimization

PPO is a policy gradient method for reinforcement learning. The motivation behind this algorithm was to have one with the data efficiency and reliable performance of TRPO while using only first-order optimization. In this algorithm, there exist 2 policies: an old policy and a current policy. The main idea is that after an update, the current policy should not be too far from the old policy. This makes sure we don't make large changes to our policy. This is important when it comes to performing tasks such as surgery wherein the robot must move with some degree of stability and must not make any sudden movements.


Algorithm 1 PPO-Clip

```
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, ...$ do
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
       on the current value function $V_{\phi_k}$.
6:     Update the policy by maximizing the PPO-Clip objective:
       $\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$
       typically via stochastic gradient ascent with Adam.
7:     Fit value function by regression on mean-squared error:
       $\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$
       typically via some gradient descent algorithm.
8: end for
```

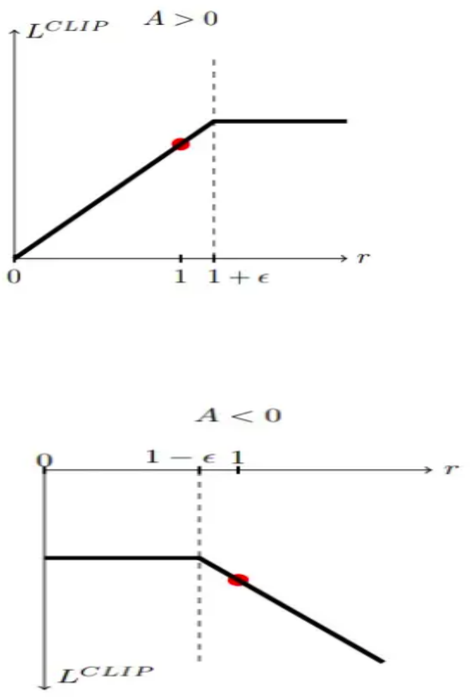


Figure 10: Clip Parameters

IV. RESULTS

A. Snake

We have implemented the DQN and Double DQN, trained and executed the model for more than 600 Games and compared them for analysis basis on the Max Score recieved and Mean Score received. for the hyperparameters the γ is 0.9 and the learning rate α is 0.1. Results comparison can be seen in Table 1 and 2.

Episode Range	DQN	Double DQN
0-100	4	1
100-300	28	22
300-600	33	40

TABLE I: Mean score in Games

Episode Range	DQN	Double DQN
0-100	19	4
100-300	72	73
300-600	70	74

TABLE II: Max score in Games

For the first 100 Games, the DQN algorithm performs better than the Double DQN algorithm in both the max score and mean score but as the model starts to gain more experience, the Double DQN algorithm starts to show better results. For the range of 100 to 300 Episodes, DQN has an mean score of 28 and Double DQN gets 22 but the highest score achieved is better for Double DQN, which is 73.

Double DQN starts to show overall better results after the 300 episodes. For the 300 to 600 episode period, the Mean and Maximum score for the Double DQN is better than the DQN. The Double DQN mean score is 40 and the DQN score is 33. The Maximum score is 74 for the Double DQN whereas for DQN it is 70.

From figures 11 and 12 we can observe that DQN initially produces a higher score after a few episodes. After around 100 episodes it gave out a score of 40 but this also shows the bias.

To counter this Double DQN is used. Double DQN starts slowly but after it gains enough experience it tends to give better and more consistent results. It can be seen overall that Double DQN reaches the upper quartile of the score achieved more than DQN.

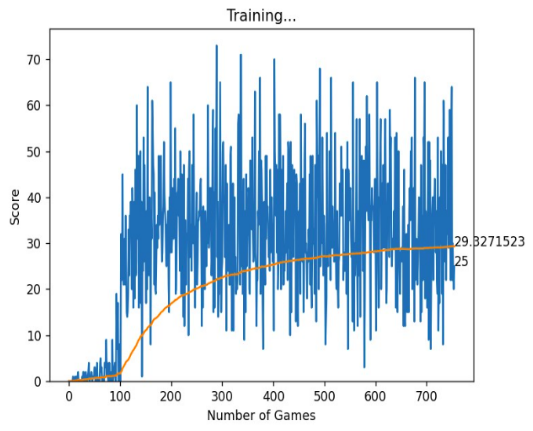


Figure 11: DQN results for snake Game

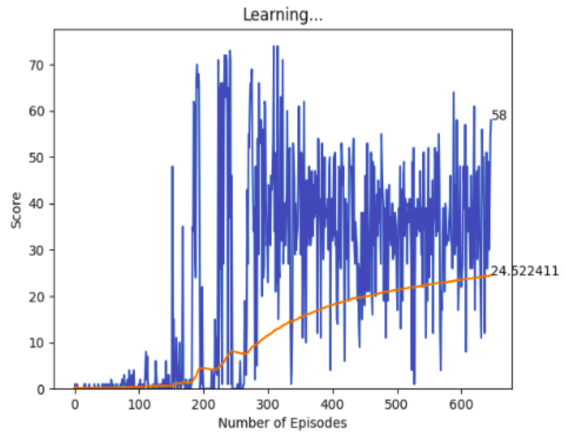


Figure 12: Double DQN results for snake Game

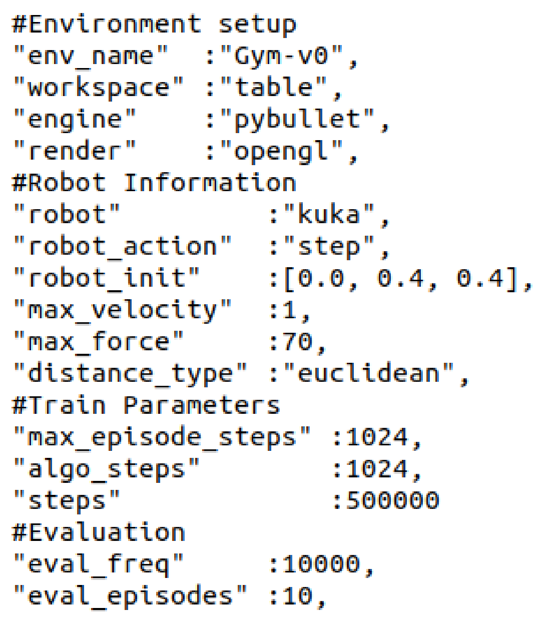
B. Robotic Arm

We have Implemented the PPO algorithm for the Robotic arm to reach to the object we have use the PyBullet environment and received results after training it 500,000 steps.

The robotic arm tries to reach the goal which is the object position and doing that it receives the reward which is the equilidean distance difference between the object and end effector position respect to previous state observation.

From Figure 14, It can be observed that there is a high density of the positive Episode rewards when the model has been trained for more than 100,000 timesteps, this shows us that the Robotic arm is moving towards the position of the object more number of times. This shows that the PPO algorithm has been training the model satisfactorily.

It can be seen observed as the number of training increases, the success rate of the grasping operation is also increasing and the avergae number of steps used reduces.



```
#Environment setup
"env_name"   :"Gym-v0",
"workspace" :"table",
"engine"     :"pybullet",
"render"     :"opengl",
#Robot Information
"robot"            :"kuka",
"robot_action"     :"step",
"robot_init"      :[0.0, 0.4, 0.4],
"max_velocity"    :1,
"max_force"       :70,
"distance_type"  :"euclidean",
#Train Parameters
"max_episode_steps" :1024,
"algo_steps"          :1024,
"steps"               :500000
#Evaluation
"eval_freq"       :10000,
"eval_episodes"  :10,
```

Figure 13: Parameters for the Robotic arm

This shows with the learning timesteps the adaptibilty of the robot with the environment improves and shows the effect of the algorithm on the agent and the system.

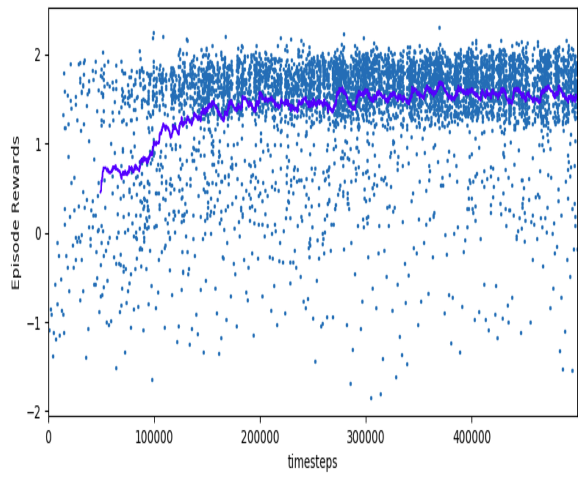


Figure 14: Episode returns w.r.t. the timesteps

From Figure 15, We can observe that the robot arm learns the model the Episode distance reduces and there is high density at the lower distance value. This shows that the robot efficiently goes to the object and performs the reach function much smoothly, the algorithm also shows its strength with the increasing timesteps.

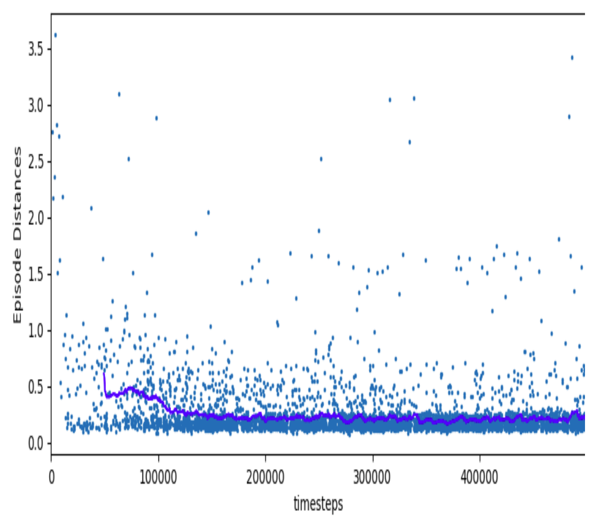


Figure 15: Episode distance w.r.t. the timesteps

From Figure 14 and Figure 15, We can conclude that the robotic arm has learned to reach the object with the minimum distance possible.

V. Conclusion

In this project, we have shown the implementation of several algorithms of Reinforcement Learning, i.e. DQN, Double DQN, and PPO. We anticipated that the difference between the performances of DQN and Double DQN might become apparent after the long training period, and the outcome verified our expectations. While we were able to get great results with the Snake game, when comparing it with a know optimal solution obtained by manually playing the game, it was found that the algorithms could not perform to that standard.

The PPO algorithm on the robot arm initially faltered but with 500,000 training steps, the module was able to perform very well and give great results. Towards the end, the arm was able to move in the three-dimensional space with ease and reach its target location and as the model trains and learned the robot received better cumulative reward and the episode distance also reduces.

VI. Acknowledgement

The Implementation of the algorithms can be found in the following GitHub repository: https://github.com/aadhar01/ReinforcementLearningProject

VII. References

[1]. D. S. A. G. I. A. D. W. M. R. Volodymyr Mnih, Koray Kavukcuoglu, (https://arxiv.org/pdf/1312.5602.pdf)

[2]. A.-C. Roibu, (https://arxiv.org/pdf/1905.04127.pdf)

[3]. Giovanni Viglietta. 2013. Gaming is a hard job, but someone has to do it!

[4]. Risto Miikkulainen, Bobby Bryant, Ryan Cornelius, Igor Karpov, Kenneth Stanley, and Chern Han Yong. Computational Intelligence in Games.URL:ftp://ftp.cs.utexas.edu/pub/neural-nets/papers/miikkulainen.wcci06.pdf

[5]. Van Seijen, H., van Hasselt, H., Whiteson, S. and Wiering, M. (2009). A theoretical and empirical analysis of Expected Sarsa, 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning pp. 177184. URL: http://goo.gl/Oo1lu

[6]. Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M.2013. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research 47: 253–279.
[7]. Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert,M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. OpenAI Baselines. https://github.com/ openai/baselines.

[8]. Delhaisse, B.; Rozo, L.; and Caldwell, D. G. 2020. Pyrobolearn: A Python framework for robot learning practitioners. In Conference on Robot Learning, 1348–1358.

[9]. J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. "Trust region policy optimization". In: CoRR, abs/1502.05477 (2015).

[10]. Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. "Sample Efficient Actor-Critic with Experience Replay". In: arXiv preprint arXiv:1611.01224 (2016).