# Rover-Arm

## 1. Abstract

The task of organizing and arranging objects in a room can be a tedious and time-consuming process. We present the RoverArm project, which aims to develop a robotic system capable of autonomously picking up and arranging objects using reinforcement learning techniques. The project is divided into multiple tasks, starting with the ability to pick up a single object, which is the goal of the first version of the project. A Gym Environment is created to facilitate training of the robotic arm and has been published as a PyPi package.

We explore various reinforcement learning algorithms, including DQN, DDPG, Advantage Actor Critic (A2C), and Proximal Policy Optimization (PPO). Additionally, we experiment with different neural network variations, such as simple Multi-Layer Perceptron (MLP), DQN-style architecture, Qt-Opt with numeric observations added layer-wise and concatenated, as well as Vision Transformer (VIT) models.

## 2. Environment and Task

The RoverArm project utilizes a simulated environment in the OpenAI Gym framework to train and evaluate the robotic arm's object manipulation capabilities. The task of the bot is to navigate close enough and pick up the object. The environment provides visual and numeric observations, including front and top views of the scene and tray, along with coordinates of rover, end effector, and object. If the bot gets close to the object it gets  a small positive reward (say 0.001) and if it gets far it gets a negative reward, once the object is picked it gets a large positive reward (+1). The action space is a six-dimensional box, ranging from -1 to 1, encompassing rover actions (throttle, steer) and arm actions (dx, dy, dz for end effector coordinates, and f for finger location).

## 3. Reinforcement Learning

### a. Algorithms

I have experimented with DQN, DDPG - Actor critic (base version), Advantage Actor Critic, and PPO. Since, A2C and PPO are expected to work better and due to training time constraints and GPU limitations, I have majorly trained on A2C and PPO.

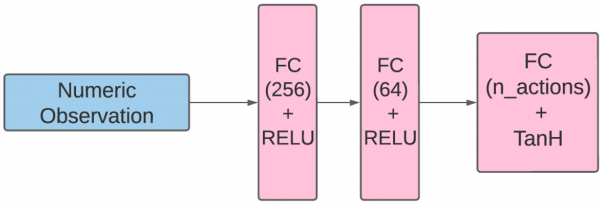
DQN - The output of neural network is n_actions * actions_per_dimension no. of values, where each action is discretized into actions_per_dimension categories and we take softmax across actions_per_dimension values to get n_actions maximum actions.

Actor Critic and PPO models - The outputs of the Actor network are two n_actions dimensional vectors, one for mean which comes from NN model, and one for standard deviation which is kept constant in Actor Critic and comes from model in PPO. We use the mean and standard deviation to sample the action. The Critic Network always outputs a single value, V function value of the state.

## b. Neural Network Architectures

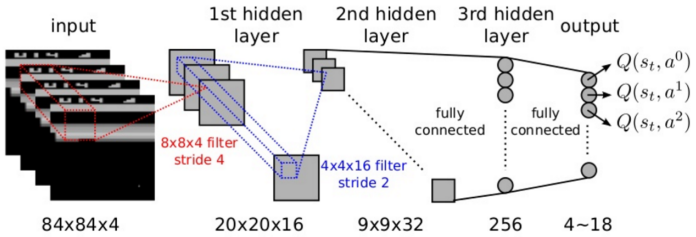
### I. Simple MLP

Numeric Observation is passed through a set of fully connected layers and RELU activation except for the final layer where Tanh is used for the activation layer.



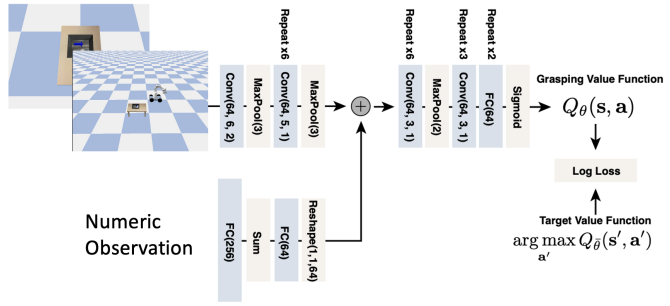

### II. DQN Style

Observation (only front view of scene) is taken, converted to gray scale image. The last 4 frames of the environment are joined and passed to the NN. A set of convolutional layers followed by fully connected layers are used.



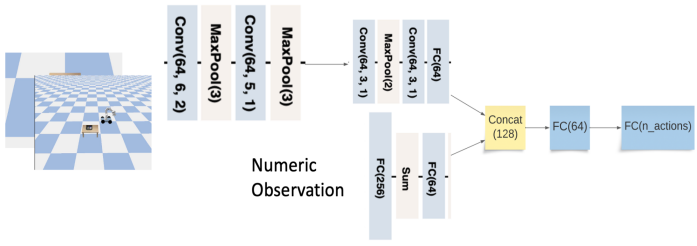

### III. Stand Qt-Opt Style

Observation (the front view of scene and top view of tray, 6 channels image) is passed through a set of conv + max pool layers and numeric observation is passed through FC + Relu layers and both the outputs are added layerwise and this vector is passed through a set conv + max pool layers and then FC + Relu and finally one FC + tanh.



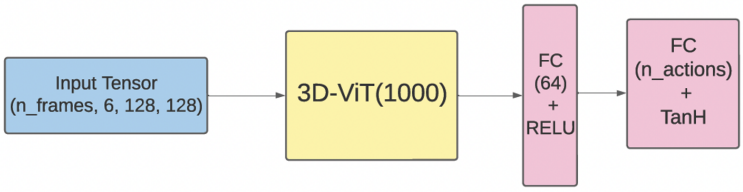

### IV. Modified Qt_Opt Style

Observation (the front view of scene and top view of tray, 6 channels image), goes through a similar process as above except that Numeric observation is concatenated instead of addition. Also, this model is altered to take n frames as input, so the input image of 6*n_frames channels is passed as the first conv layer's input.



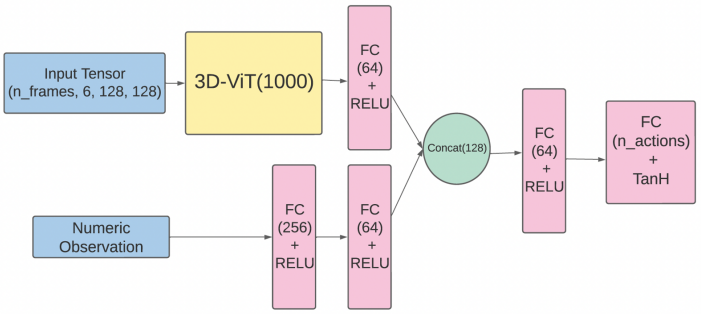

### V. 3D-ViT

Observation (n_frames of 6 channel image) is passed through the VIT block, which outputs an encoded vector of size 1000 and which is passed to FC + RELU and then FC + Tanh layers.



### VI. 3D-ViT + Numeric Observations

Observation (n_frames of 6 channel image) is passed through the VIT block, which outputs an encoded vector of size 1000 and which is passed to FC + RELU, the numeric observation goes through set of FC layers and then both are concatenated and this is passed to set of FC layers.



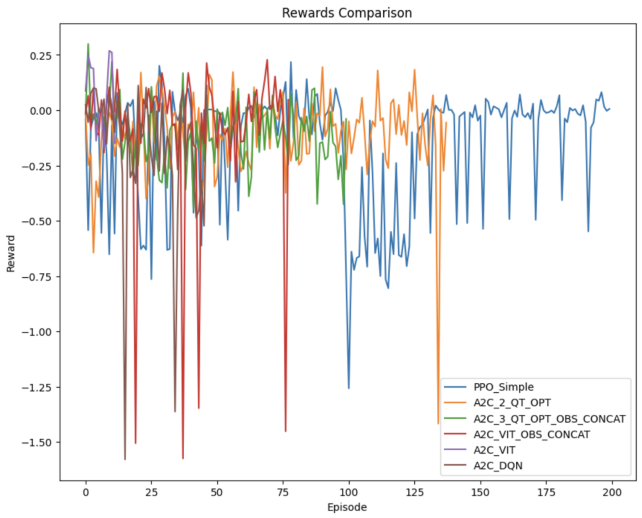

## 4. Training

The simplest task of the environment (navigating and picking the object from the tray) is quite complex because it takes at least 3000-5000 steps when the task is performed by a human. Therefore, it requires large amounts of data to be trained, and large models like ViT that support the retention of such information. These models take at least 40-50 hours to get trained for 100 episodes of 2000 steps. Detailed training times can be found in the table on the right.

| MODEL | EPISODES | STEPS | Observed TRAINING TIME | Expected training (100 episodes) |
|---|---|---|---|---|
| A2C_Simple | 200 | 2000 | 2HOURS | 1Hour |
| PPO_Simple | 200 | 2000 | 2HOURS | 1Hour |
| A2C_1_DQN | 50 | 2000 | 6HOURS | 12Hours |
| A2C_2 – QT OPT | 100 | 2000 | 22HOURS | 22HOURS |
| A2C_3 – QT OPT + OBS CONCAT | 100 | 2000 | 16HOURS | 16HOURS |
| A2C – VIT2 (OBS CONCAT) | 80 | 2000 | 32HOURS | 40HOURS |
| A2C – VIT1 | 10 | 2000 | 6HOURS | 60HOURS |

## 5. Results and Conclusion

A plot of Episode Reward vs Episode has been shown on the right. The poor performance is majorly due to the lack of training data. If the model is provided with hand crafted examples or expert trained data, it could perform better. Yet, we can observe that the rover navigates towards the object and the arm tries to get close to the object in some instances.




The environment and training code with other details can be found in this repository.
Saiphani724/RoverArm: A OpenAI Gym Env for Rover with Arm (github.com)

## 6. References


a. DQN - [1312.5602] Playing Atari with Deep Reinforcement Learning (arxiv.org)
b. DDPG - [1509.02971] Continuous control with deep reinforcement learning (arxiv.org)
b. A2C - [1602.01783] Asynchronous Methods for Deep Reinforcement Learning (arxiv.org)
c. Qt-Opt - [1806.10293] QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation (arxiv.org)
e. TorchRL PPO - Reinforcement Learning (PPO) with TorchRL Tutorial
f. ViT Pytorch Code - lucidrains/vit-pytorch: Implementation of Vision Transformer