Deep RL: Manipulator

Shilpaj Bhalerao

July 29, 2019

Abstract: *Deep Reinforcement Learning has achieved a great deal of achievements. AlphaGo used Deep RL to win game against world's best Go player. This paper shows use of Deep RL by a robotic arm manipulator to touch the object.*

Keywords: ROS, Gazebo, Deep Learning, Reinforcement Learning, Deep RL, etc.

1. Introduction

Deep Reinforcement learning is a field of robotics in which raw image is taken from a camera and fed to Deep Neural Network. The output of the DNN is passed to an agent which gets trained on the data and then applies control value to the actuators. In this paper, a camera is placed in front of the robotic arm. The camera input is passed to the DRL agent.

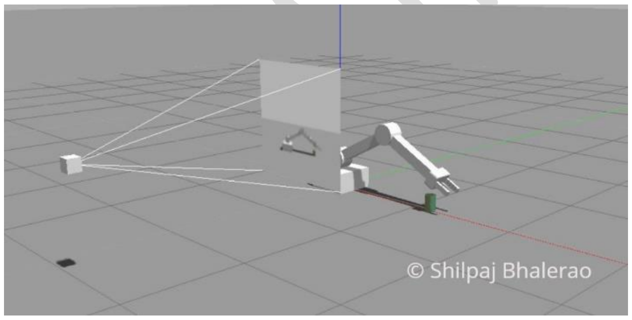


Fig 1: Robotic ARM manipulator with camera

The DRL agent has to achieve following tasks:

- 90% accuracy when arm collides with prop
- 80% accuracy when gripper collides with prop

2. Reward Function

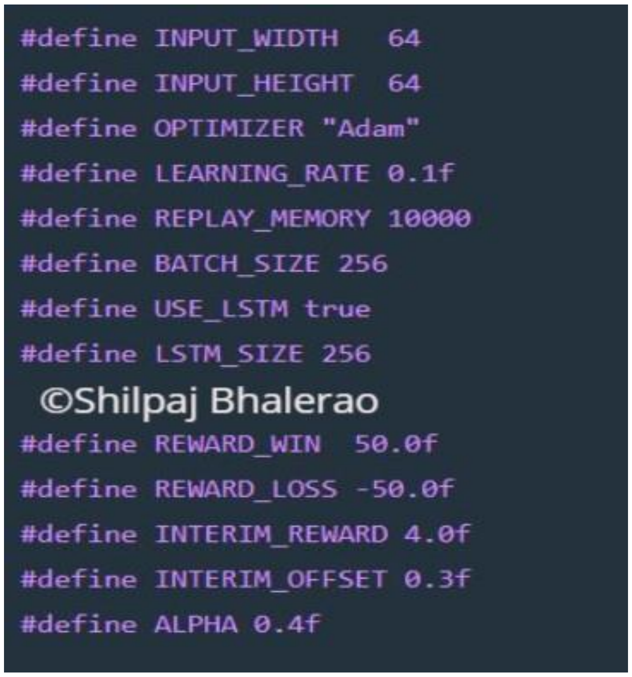
The reward function is defined in ArmPlugin.cpp in the repository. The objective of the project is to touch the prop. The episode will end if the robotic arm touches the ground or the object.

To achieve this task, the arm is punished when it touches the ground or it fails to complete the task within the selected frames.

The reward function is designed to reward the arm 50 points when it touches the object and penalizes it 50 points if it fails to complete the task or touches the ground.

An interim reward was given based on the current position of the robotic arm. The method used is a smooth moving average based on delta value which is a distance of arm from the object. The reward function also contains alpha term which decides how much to rely on the previous delta value. This reward function eliminates the fluctuations and helps to maximize the reward.

3. Hyperparameters



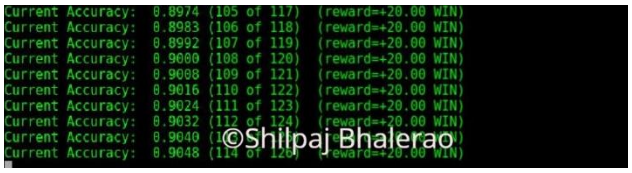
```
#define INPUT_WIDTH   64
#define INPUT_HEIGHT  64
#define OPTIMIZER "Adam"
#define LEARNING_RATE 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 256
#define USE_LSTM true
#define LSTM_SIZE 256
#define REWARD_WIN  50.0f
#define REWARD_LOSS -50.0f
#define INTERIM_REWARD 4.0f
#define INTERIM_OFFSET 0.3f
#define ALPHA 0.4f
```

Fig 2: Hyperparameters

Above figure shows the hyperparameters used for the project. The input size is reduced from 256 to 64 in order to reduce the processing time. Adam optimizer is used for the DRL network. Usually, 0.01 is good initial value of learning rate but in this case, it wasn't helping so decided to use 0.1. Batch size of 256 is used along with the LSTM of 256 units. The value of alpha which decides how to rely of previous value is 40%.

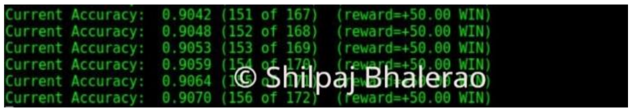
4. Results

Objective-1 of achieving task with more than 90% accuracy when arm touches the prop is tried with two different reward values of 20 and 50. The objective is achieved earlier in case of 20 points compared to the 50 points.



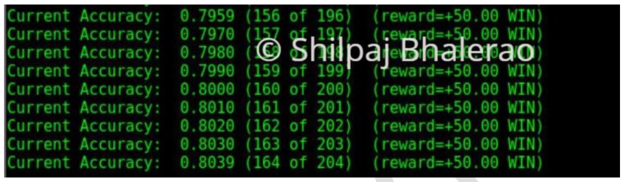
```
Current Accuracy:  0.8974 (105 of 117)  (reward=+20.00 WIN)
Current Accuracy:  0.8983 (106 of 118)  (reward=+20.00 WIN)
Current Accuracy:  0.8992 (107 of 119)  (reward=+20.00 WIN)
Current Accuracy:  0.9000 (108 of 120)  (reward=+20.00 WIN)
Current Accuracy:  0.9008 (109 of 121)  (reward=+20.00 WIN)
Current Accuracy:  0.9016 (110 of 122)  (reward=+20.00 WIN)
Current Accuracy:  0.9024 (111 of 123)  (reward=+20.00 WIN)
Current Accuracy:  0.9032 (112 of 124)  (reward=+20.00 WIN)
Current Accuracy:  0.9040 [illegible]
Current Accuracy:  0.9048 (114 of 126)  (reward=+20.00 WIN)
```

Fig 3: Objective-1 with more than 90% accuracy



```
Current Accuracy:  0.9042 (151 of 167)  (reward=+50.00 WIN)
Current Accuracy:  0.9048 (152 of 168)  (reward=+50.00 WIN)
Current Accuracy:  0.9053 (153 of 169)  (reward=+50.00 WIN)
Current Accuracy:  0.9059 (154 of 170)  (reward=+50.00 WIN)
Current Accuracy:  0.9064 [illegible]
Current Accuracy:  0.9070 (156 of 172)  (reward=+50.00 WIN)
```

Fig 4: Objective-1 with different reward function



```
Current Accuracy:  0.7959 (156 of 196)  (reward=+50.00 WIN)
Current Accuracy:  0.7970 (157 of 197)  (reward=+50.00 WIN)
Current Accuracy:  0.7980 [illegible]
Current Accuracy:  0.7990 (159 of 199)  (reward=+50.00 WIN)
Current Accuracy:  0.8000 (160 of 200)  (reward=+50.00 WIN)
Current Accuracy:  0.8010 (161 of 201)  (reward=+50.00 WIN)
Current Accuracy:  0.8020 (162 of 202)  (reward=+50.00 WIN)
Current Accuracy:  0.8030 (163 of 203)  (reward=+50.00 WIN)
Current Accuracy:  0.8039 (164 of 204)  (reward=+50.00 WIN)
```

Fig 5: Objective-2 with more than 80% accuracy

As the agents keeps training for a greater number of epochs, it keeps on improving its performance. Initially, it couldn't even figure out in which direction to move. As the time passes, it becomes more and more consistent.

To achieve the goal of more than 90% accuracy, agent is rewarded if there is a collision between arm and the prop. Similarly, in case of 80% accuracy, agent is only rewarded when gripper collides with prop.

The agent has learned to perform its objective but more fine tuning on system with higher processing capability will help to improve the results.

5. Future Work

To improve the project results, one can try using a deeper neural network. The network can be optimized using Batch Normalization, dropout, etc. One can also try making the network wider to achieve more accuracy.

I want to try placing 2 cameras perpendicular to each other in the gazebo environment which will allow one more degree of freedom to the robot. Then the robot can be trained to pick up objects in 3D world.