Deep RL Arm Manipulation

Bahadir Ozkan

Abstract—Deep reinforcement learning is being applied to variety of robotics problems. The nature of reinforcement learning (RL) is much like the human learning comes from interacting with the environment. In RL the agent with the positive or negative rewards learns from its environment. The goal of this paper is to describe how a DQN agent with its reward functions is created for teaching a robotic arm to carry out two objectives.Challenges of creating the RL agent for these tasks are discussed. Structure of the reward function and the hyper-parameters that are used for the agent to achieve the objectives are described.

Index Terms—Robotic arm manipulation, RL, deep learning, DQN agent.

1 Introduction

Reinforcement learning is different from supervised or unsupervised learning in a way that it is not dependent to the labeled data and since it's goal is not to explore the structure of unlabeled data. In reinforcement learning, an agent interacts with its environment with the help of a reward function and tries to discover the actions that will maximize the positive rewards. This notion of RL provides great flexibility to apply it to numerous problems. In this project, a Deep Q-Learning Network (DQN) was used to teach a robotic arm to execute given tasks. A camera feed is provided as input to the agent and the collisions of the robotic arm was captured to evaluate the success rate. The two task that are given to the agent are:

1) Have any part of the robot arm touch the object of interest, with at least a 90% accuracy.
2) Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy.

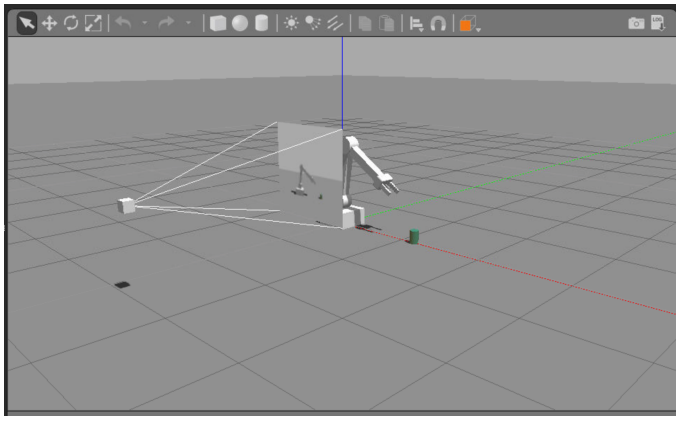

Fig. 1. Udacity, Deep RL Arm Manipulation Project

2 Reward Functions

Udacity Project Workspace with enabled GPU was used to perform the training and simulation. The reward function that penalizes ground collision or when the episode exceeds 100 step time, and gives a reward when the tube or gripper collision is achieved was used for both tasks. However, issuing an interim reward based on the distance to the object was more complicated for the second task. Average delta, which is smoothed moving average of the delta of the distance to the goal was used to calculate the distance between the arm and the object.

```
average_delta = (average_delta * alpha) +
    (dist * (1 - alpha))
```

The interim reward function is based on the distance of the arm to the object. If the distance to the object, **avgGoalDelta** is positive, for task 1 the **rewardHistory** is updated as REWARD_WIN multiplied with the **avgGoalDelta** which spurs the arm to get closer or touch to the object. Additionally, arm was rewarded if it is very close to the goal or when it contacts the goal. Task 1 was achieved quite easily after setting the hyper-parameters therefore a simple reward function was pat.

```
if(avgGoalDelta>0 || distGoal < 0.1)
{
  //rewardHistory = REWARD_WIN *
      avgGoalDelta; //Task 1
  rewardHistory = (1 -
      pow(distGoal,0.4f)); //Task 2
}
else if (distGoal == 0)
{
  rewardHistory = REWARD_WIN * 20;
}
else
{
  rewardHistory = REWARD_LOSS * distGoal;
}
```

Interim reward function for the second task is more sophisticated function due to remedy sparse reward issues. If the reward function then agent is not rewarded very often. Modified reward function redueces agent's learning time due to the gradual feedback it provides. Exponantial of the distGoal was subtracted from one to achieve a gradual reward. For further information see Medium post of Bonsai.

3 HYPER-PARAMETERS

The following hyper-parameters were tuned to achieve the tasks:

- **INPUT_WIDTH and INPUT_HEIGHT:** Input width and height were chosen as 64 x 64 to reduce memory usage.
- **OPTIMIZER:** RMSprop was used for Task 1 and Adam for Task 2 however no significant difference was observed and both Adam and RMSprop could be used for both tasks that are given.
- **LEARNING_RATE:** 0.01 was used for the first task. For the second task the learning rate was needed to set to 0.02 to achieve a better result.
- **OTHERS:** LSTM was used in both tasks with doubled size in Task 2. Velocity control was used for the first task, position control for the second. Position control provided far better results than the velocity control on the second task.

4 RESULTS

DQN agent could not perform the tasks without elaborate reward functions and fine tuning in Task 2. Task 1 was achieved with less effort. An accuracy of 97 percent was reached in the first task in 100 episodes. On the second task %80 thresholds was exceeded on the 80th episode and the accuracy increased over %90 before it was terminated on episode 160.

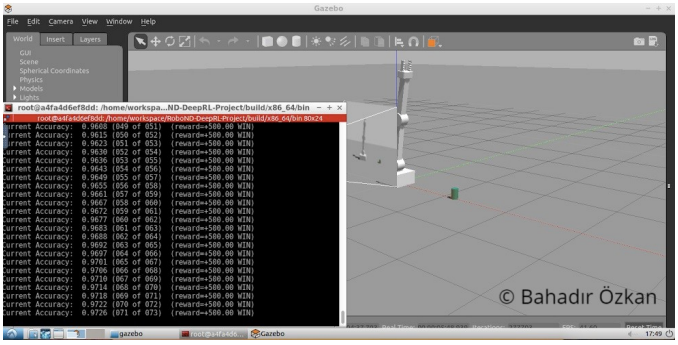

Fig. 2. Task 1

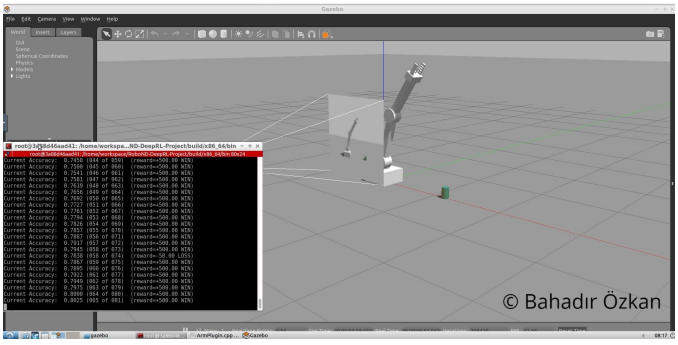

Fig. 3. Task 2

5 FUTURE WORK

Reward functions and hyper-parameters were modified to achieve second task after successfully completing the first task. Convolutional Neural Networks (CNNs) or similar networks can be adapted to support the vision based robot control. An agent trained with a CNN is expected to be more flexible and reusable in given tasks.