Udacity RoboND Project

Deep Reinforcement-Learning Arm Manipulation

Final Project Report

Lutong Zhang

Table of Content

1. Abstract

This is the report for the Deep RL Arm Manipulation project of Udacity Robotics Nanodegree. The background of Deep RL is discussed. The implementations and the results are presented in the following chapters.

2. Introduction

Robots should behave both automatically and beneficial. This requires the observation of the environment and then action based on the states. This process can be pre-programmed, or can be learnt. Traditionally, the functions that maps the observation to actions are predefined. As the fast grown of Deep Learning, the robot now can abstract information from observations and then make decisions based on the neural network. This project is an example of this process.

3. Background

3.1 A Brief Introduction of Reinforcement Learning

One fundamental desire for robotics automation is to let the robot make decisions and act based on the conditions of environment as well as its own condition. The behavior of the robot should be helpful or beneficial, in an optimized manner. In the context of reinforcement learning, this robot is named as an agent.

Fig. 1 Interaction between Agent and Environment

The conditions of environment together with its own is named as state of the agent. The agent make action based on the state. Each action will lead to a result. The agent will be rewarded if the result is good, or other way around. A state of an agent is represented as s_n, and its action can be represented as a_n, where n stands for index of time-point for discrete system. The reward of a_n is usually written as r_{n+1}.

The mechanism that the actions are chosen at different states is called a policy.

$$s_n \xrightarrow{a_n} r_{n+1}, s_{n+1}$$

It's nature that the goal of the agent is to maximize the cumulative reward it receives in the long run. The discounted version of the expected return at time n can be written as

$$G_n = r_{n+1} + \gamma r_{n+1} + \gamma^{n+1} r_{n+2} + \cdots$$

where γ is a parameter, and it is valued between 0 and 1. It is called a discount rate. This discounted return function can be written in a recursive manner

$$\begin{aligned} G_n &= r_{n+1} + \gamma r_{n+1} + \gamma^{n+1} r_{n+2} + \cdots \\ &= r_{n+1} + \gamma(r_{n+1} + \gamma^n r_{n+2} + \cdots) \\ &= r_{n+1} + \gamma G_{n+1} \end{aligned}$$

A value function is defined as a function of states that estimate how good it is for the agent to be in a given state. Accordingly, value functions are defined with respect to particular policies. The state-value function for policy π is

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

Here E_π denotes the expected value of random variable given that the agent follow policy π. We can also define the action-value function for policy π as

$$p_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

These value functions can also be written recursively as

$$v_\pi(s) = E_\pi[R_{n+1} + \gamma G_{n+1} | S_n = s]$$

This function is also named as Bellman equation for v_π.

The optimal policy is a policy that maximize the action-value function

$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

which can be written recursively as

$$q_*(s, a) = E_\pi[R_{n+1} + \gamma v_*(S_{n+1}) | S_n = s, A_n = a]$$

There are several different ways to solve this optimization problem. However, in the real world the problem is there always exist infinitely many combinations of states. The optimization methods such as dynamic programming may fail for this kind of problem because of the complexity. This is where the Deep-Learning kicks in.

3.2 Deep-Learning in Reinforcement Learning

The agent can sometimes be regard as a function which takes a state as an input and outputs a choice of action. The agent to understand the situation by processing the information in the state, and Deep-Learning is a good tool for processing this kind of task.

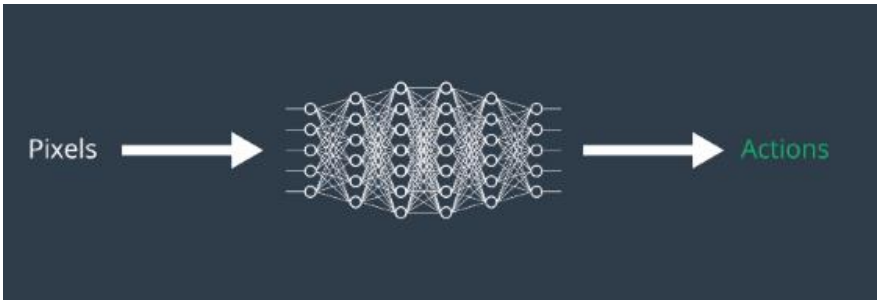

Fig. 2 Deep-Learning for Reinforcement Learning

The state-value function is formed by training the deep neural network (DNN). The cost function of the DNN is

$$\Delta w = \alpha[\left(R_{n+1} + \gamma \max_{a} q_*(s', a', w)\right) - q_*(s, a, w)]$$

3.3 Tasks of the Project

In this project, the input of the DNN is the raw image from the Gazebo simulator. The input image is firstly passed though several CNN layers, which are followed by a fully connected layer. The construction and training of the DNN are handled by an API developed by Nvidia.

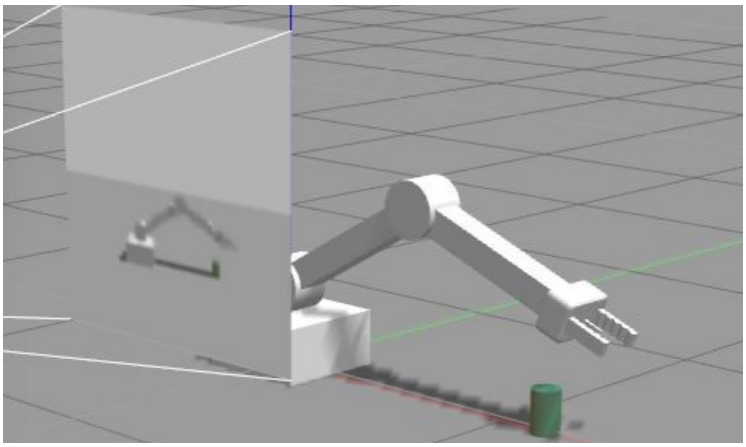

Fig. 3 The agent and the environment of this project

In this project, the agent is a robot arm, which has two revolve joints. The goal of

the agent is to touch the green tube by its body/gripper.

The first task of this project is to define rewards for the agent under different conditions. Secondly the control method of the robot arm should be defined. On the other point of view, this also defines the possible actions that the agent can take. The final task is to fine tune the hyperparameters and train the DNN to get good results. The detailed implementation will be explained in the next part of this report.

4. Configurations and Implementations

4.1 Configurations of Reward Functions

This robot-touching problem can be regard as an episodic problem. There are three conditions under which one episode should be terminated:

- Robot arm touch the tube
- Robot arm touch ground
- Timeout

If the robot arm touches the ground, the mission is failed. The reward for this condition is set to be negative. The reward is set in such a way that if the gripper is closer to the tube, a higher negative is returned to the robot.

If the robot arm touches the tube, the mission is successful. The reward for this condition is positive. An additional modification of this reward is that if the robot touches the tube earlier, a higher reward will be returned.

If the robot fails to touch the tube till the end of the episode, the mission is failed. A negative reward will be returned to the agent.

There are two objectives to the project. The first one is that any part of the robot arm should touch the object with at least an accuracy of 90%. The second one is that only the gripper base of the robot arm should touch the object with at least an accuracy of 80%. The rewards for these to objectives are slightly different.

4.2 Control Methods

For this robot arm, there are two ways to control. The robot arm can be controlled by speed, or by position. This choice can be implemented by *#define velocity_control*

to true or false.

The control reference signal is given by the function of *ArmPlugin::NextAction()*. Under velocity-control, the velocity is added or subtracted by an amount of value defined as *actionVelDelta*. In the same way, under position-control, the position is added or subtracted by *actionPosDelta* based on the *action* value. For the first objective, the velocity-control is implemented. For the second objective, the position-control is implemented.

4.3 Hyperparameters

Table 1 Choice of Hyperparameters for both Objective 1 and Objective 2

Parameter	Description	Value for Obj 1	Value for Obj 2
INPUT_WIDTH	Width of the input image	256	64
INPUT_HIGHT	Hight of the input image	256	64
OPTIMIZER	Choice of optimization function	RMSprop	Adam
LEARNING_RATE	Rate to determine how fast the DNN learns	0.8	0.08
REPLAY_MEMORY	Size fo the Replay Memory	2000	10000
BATCH_SIZE	Input batch size	128	256
USE_LSTM	Whether to use LSTM	TRUE	TRUE
LSTM_SIZE	Number of LSTM nodes to use in DQN	64	128

5. Results and Discussion

5.1 Objective 1

The goal of this objective is to let the robot arm touch the cylinder with any part. The training process of this mission is progressive. The final result is shown in Fig. 4 with accuracy above 90%.

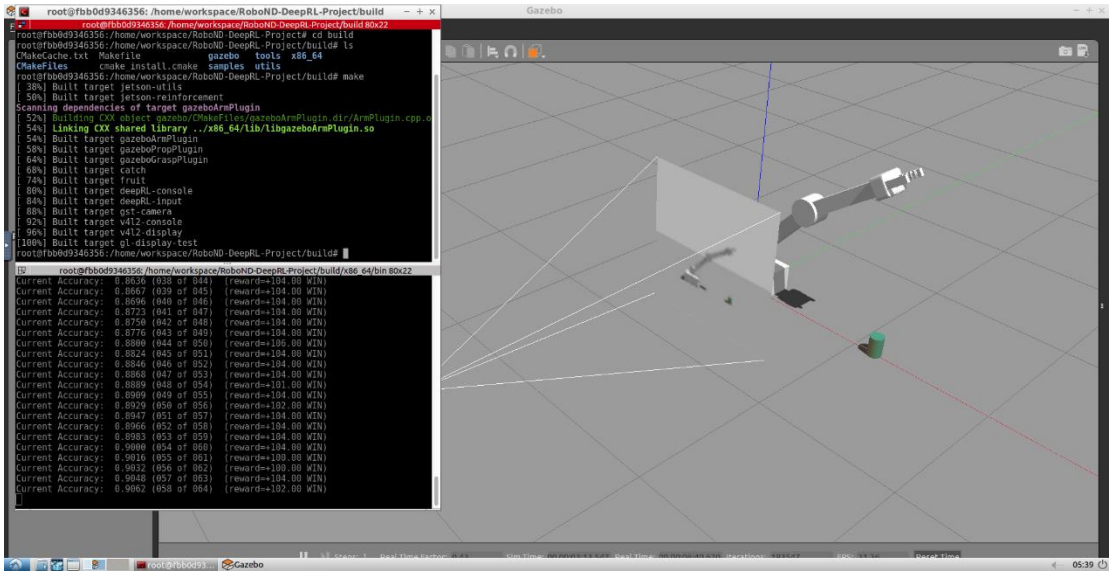

Fig. 4 Arm touching cylinder

5.2 Objective 2

The goal for objective 2 is to let the robot touch the cylinder only by its gripper. The result of this objective is shown in the figure below.

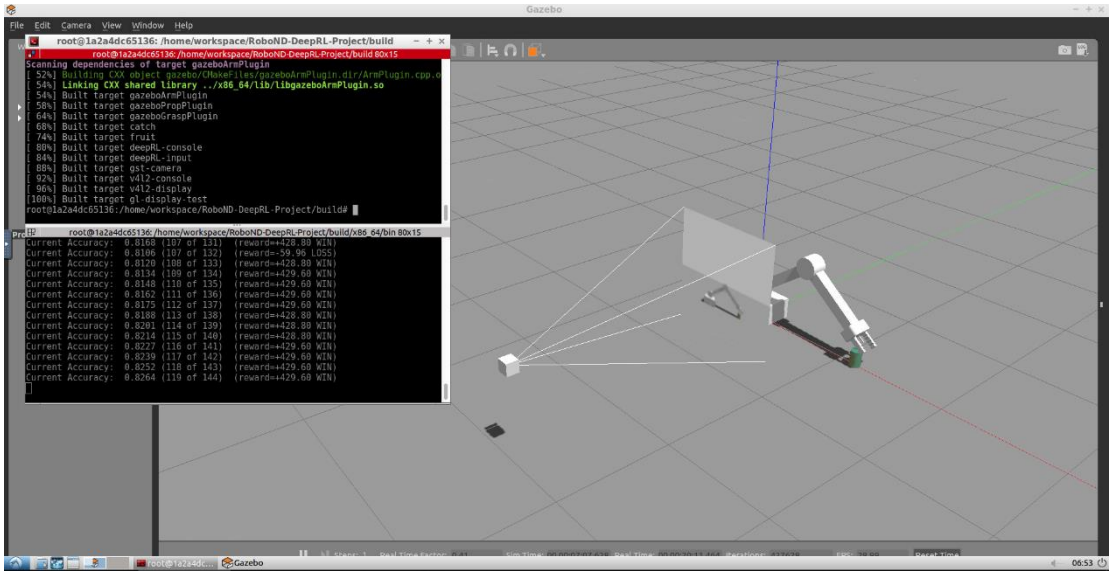

Fig.5 Gripper touching the cylinder

5.3 Discussion

With both objective accomplished, this DQN agent is provided to be effective for this kind of tasks. For both two tasks, at the beginning of the training process the agent seams to exploring the environment randomly. After a few episode the agent seams to find a correct direction. In the first task, the revolution of the elbow joint is small.

Nearly only the shoulder joint is turning to touch the cylinder. This behavior is beneficial because the task can be done quickly in this way. The second task seems to be more sensitive to randomness. Thus, the control method is turned from velocity-control to position-control. The training process is based on GPU provided Udacity. This makes the task faster and easier.

6. Future Work

The performance of the DQN maybe improved as follows. Other hyperparameters other the ones mentioned above can be tuned, such as the size of the replay memory, epsilon decay rate and so on. Additionally, I am planning to run this project on Jetson TX2 with a real-world robot arm instead of simulation environment.

Reference:

[1] R. S Sutton, A. G. Barto, Reinforcement Learning: An Introduction, The MIT Press, 2018