Deep RL: Arm Manipulation

Juil O (horagong@gmail.com)

Abstract—The Deep Q Networkd(DQN) agent was trained to make the robotic arm to touch the object. The robot has the camera sensor and arm with three movable joints. The reward is given based on the distance between the gripper and the target object. It could touch the target object with at least the given accuracy.

Index Terms—Deep RL, Arm Manipulation

1 Introduction

This project uses DQN agent to control the robot arm and has two objective.

- Have any part of the robot arm touch the object of interest, with at least a 90% accuracy for a minimum of 100 runs.
- Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy for a minimum of 100 runs.

Q-Learning is a model-free algorithm, which means it can explore an environment which may not be fully defined. The values for each state-action pair are estimated based on observations of that environment. More specifically Q-Learning is a TD or Temporal Difference learning approach, because state changes are learned with the assumption that they are sequential, or time-based.

The Q-Learning algorithm is represented with an iterative equation that includes a learning rate(α), and a discount factor(γ). The learning rate is a value between 0 and 1 and represents the portion of new information that is incorporated into the q-value at each time step. The discount factor is also a value between 0 and 1 and represents the portion of the future rewards that influence the new qvalue at each time step.

As the agent explores the environment and acquires experience from different state-action pairs, it converges on a policy of action for any given state it observes.

The typical exploration method in Q-Learning is the epsilon-Greedy exploration method. The larger the epsilon value, the more frequently the agent chooses a random action instead of the action with the highest q-value in its table. This improves the chances that a large part of the state-action space is explored and tested. Once convergence has been achieved, the trained agent can be run without the exploration method.

A key difference between RL and Deep RL is the use of a deep neural network. The collection of value-action pairs that define what actions an agent should take in any situation can be thought as a function of the observations that the agent receives from its environment. A neural network can be used to approximate this function because through its large quantity of parameters that can be learned through trial and error. [1]

2 Simulation

The hyperparameters used are as followings.

- INPUT_WIDTH, INPUT_HEIGHT: 64
  - the same as the size of the camera image
- USE_LSTM: true
- LSTM_SIZE: 256
  - to memorize more moves
- OPTIMIZER: "Adam"
- LEARNING_RATE: 0.005f
- BATCH_SIZE 64
- REPLAY_MEMORY: 10000

Two type of joint control method was tested in the updateAgent(). The control for the position gave the better result.

- control for the velocity of the robot joint

```
velocity = vel[action/2] + actionVelDelta
    * ((action % 2 == 0) ? 1.0f : -1.0f);
```

- control for the position of the robot joint

```
joint = ref[action/2] + actionJointDelta
    * ((action % 2 == 0) ? 1.0f : -1.0f);
```

The reward parameters are like followings.

- REWARD_WIN: 0.1f
- REWARD_LOSS: -0.1f
- ALPHA: 0.7f
  - smoothing factor to the average distance.

The reward functions give REWARD_WIN or REWARD_LOSS like this.

- If the robot touches the target, REWARD_WIN * 10 (objective 1), REWARD_WIN * 20 (objective 2)
- If the robot hits the ground, REWARD_LOSS * 20
- When it exceeds the number of iteration per episode, REWARD_LOSS * 20
- According to the smoothed moving average of the delta of the distance to the goal,

```
avgGoalDelta  = avgGoalDelta * ALPHA
    + distDelta * (1.0f - ALPHA);
rewardHistory = avgGoalDelta > 0
    ? REWARD_WIN
    : (REWARD_LOSS * 20 * distGoal);
```

3 Results

The objetive 1 of the robot arm touching the object was completed at least a 90% accuracy for a minimum of 100 runs.

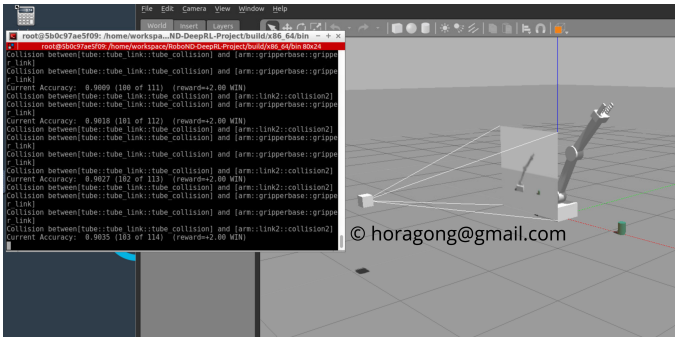

Fig. 1. Objective 1

The objetive 2 of the arm's gripper base touching the object was also completed at least a 80% accuracy for a minimum of 100 runs.

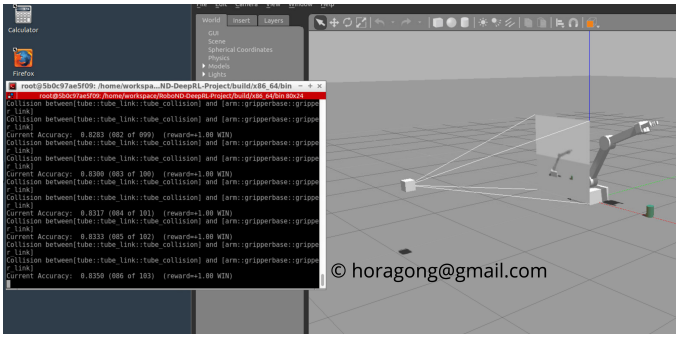

Fig. 2. Objective 2

It was the LEARNING_RATE hyperparameter that had a significant effect on the performance. At first, the simulation started with 0.1f but the accuracy results seemed to be somewhat oscillated. Other values, 0.01f, 0.005f, were tried and the 0.005f was selected.

The REWARD_WIN parameter is selected differently depending on the objectives. In the objective 1, when the any part of the robot touches the target object, the reward is assigned to REWARD_WIN * 10. But it had to be changed to other value in the objective 2. In that case, the reward was assigned to REWARD_WIN * 20. The objective 2 requires only the robot arm to touch the target arm and the chance to get the reward for touching the target object would be smaller than the objective 1. So the twice of the reward of the objective 1 seems to be more balanced and gave the better result.

In the simulation the robot sometimes moved slowly or didn't moved. So the interim reward was adjusted for that case. The interim reward was assigned to REWARD_LOSS as well as REWARD_WIN depending on the moving average of the distance between the arm and the target object. REWARD_LOSS * 20 * distance was appropriate as the negative reward.

4 Future work

This project was done on the simulator so it would be good to deploy on the real robot board.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press Cambridge, Massachusetts London, England, 2018.