Deep RL Arm Manipulation

Jiangdong Chen

Abstract—The goal of this project is to create a Deep Q-learning Network (DQN) agent and define reward functions to teach a 3 DOF robotic arm to touch the object. Through tuning hyperparameters and training the DQN agent, the robotic arm fulfils the requirements of having any part of the robot arm touch the object of interest with at least 90% accuracy, and having only the gripper base touch the object with at least 89% accuracy.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, deep learning.

1 Introduction

Reinforcement learning methods have been applied to range of robotic control tasks, from locomotion to manipulation and autonomous vehicle control [1]. Reinforcement learning enables a robot to autonomously discover an optimal behavior through trial-and-error interactions with its environment. Instead of explicitly detailing the solution to a problem. In reinforcement learning the designer of a control task provides feedback in terms of a scalar objective function that measures the one-step performance of the robot [2].

In reinforcement learning, an agent tries to maximize the accumulated reward over its lifitime. Reinforcement learning aims to find a mapping from states to actions, called policy π, that picks actions a in given states s maximizing the cumulative expected reward. In general, the agent needs to learn and discover the relations between states, actions, and rewards.

However, reinforcement learning is generally a hard problem, especially in robotics field. Problems in robotics are often best represented with high-dimensional, continous states and actions, and it is often unrealistic to assume that the true state is completely observable and noise-free.

2 Background

In this project, the problem is to build a DQN agent that uses image data from a fixed camera sensor and reward values as input, then produce appropriate actions. For every frame that the camera receives, the agent needs to take an appropriate action. This output (action value) can then be mapped to a specific action - controlling the arm joints. The robot arm has three degree of freedom, and is required to perform the following task:

1) Touch the object with any part of the robot arm with 90% accuracy;
2) Touch the object with the gripper base of the robot arm, with 80% accuracy.

The simulation is implemented in a Gazebo environment. There is a tube object, fixed camera sensor, and 3DOF arm, as Fig.1 shown.

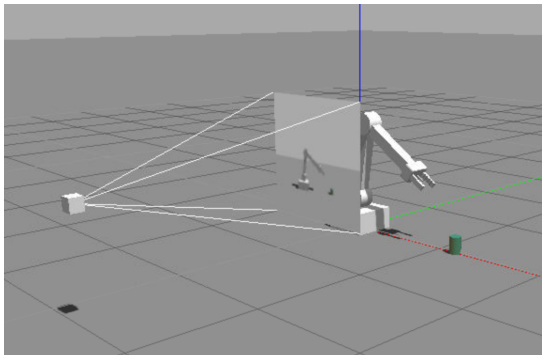

Fig. 1: Gazebo Simulation Environment.

2.1 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative $reward$ [3]. Basic reinforcement learning is generally modeled as a Markov decision process, which can be represented as a diagram as Fig.2 shown [4]. An agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent.

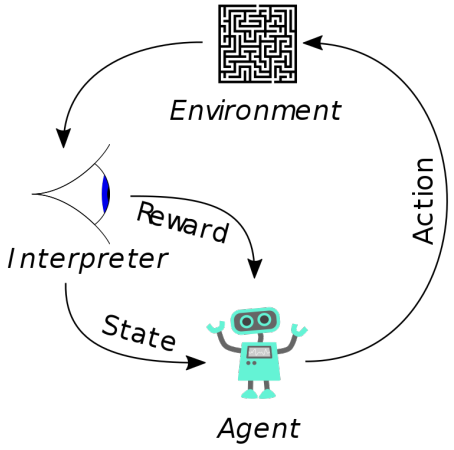

Fig. 2: Reinforcement Learning Diagram.

2.2 Q-Learning

Q-learning is a commonly used model free approach. It builds a table to store rewards for each state and action pair, and updates these values at every time step. Q-learning can be defined by [5]:

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

The value update rule is the core of the Q-learning algorithm.

2.3 Deep Reinforcement Learning and Deep Q-Network

Deep reinforcement learning can be considered as the combination of deep learning and reinforcement learning, because it uses artificial neural networks to train the agent. Deep Q-Network (DQN) is the ignition of the deep RL field. DQN stabilize the training of Q action value function approximation with CNN using experience replay and target network [6]. DQN uses image pixels and the rewards as inputs, thus is a end-to-end reinforcement learning approach.

3 Simulations

3.1 Rewards Definition

The common rewards of Task 1 and Task 2 include reward for the robot gripper hitting the ground, and an interim reward based on the distance to the object.

The hitting ground reward definition is shown in Fig.3.

```
bool checkGroundContact = gripBBox.min.z <= groundContact ? true : false;

if(checkGroundContact)
{

    if(DEBUG){printf("GROUND CONTACT, EOE\n");}

    rewardHistory = REWARD_LOSS;
    newReward     = true;
    endEpisode    = true;
}
                                                                    @Chen
```

Fig. 3: Definition of Hitting Ground Reward.

The interim reward definition is shown in Fig.4.

```
if(!checkGroundContact)
{
    const float distGoal = BoxDistance(gripBBox, propBBox); // compute the reward from distance to the goal

    if(DEBUG){printf("distance('%s', '%s') = %f\n", gripper->GetName().c_str(), prop->model->GetName().c_str(), distGoal);}


    if( episodeFrames > 1 )
    {
        const float distDelta  = lastGoalDistance - distGoal;

        // compute the smoothed moving average of the delta of the distance to the goal
        avgGoalDelta = avgGoalDelta * ALPHA + (distDelta * (1.0 - ALPHA));
        rewardHistory = INTERIM_REWARD * avgGoalDelta - INTERIM_OFFSET;
        newReward     = true;
    }

    lastGoalDistance = distGoal;
}
                                                                    @Chen
```

Fig. 4: Definition of Interim Reward.

Compare Task 1 and Task 2, there is a difference in reward definition. In Task 2, it is only valid if the collision position between the manipulator and the object is on gripper. So a reward based on collision between the arm's gripper base and the object is issued, and the reward based on collision between the arm and the object in Task 1 is replaced. The definition of collision rewards for Task 1 and Task 2 is shown in Fig.5.

```
/* Task 1: Check if there is collision between the arm and object, then issue learning reward */
//bool collisionCheck = strcmp(contacts->contact(i).collision1().c_str(), COLLISION_ITEM) == 0 ? true : false;


/* Task 2: Check if there is collision between the gripper and object, then issue learning reward */
bool collisionCheck = (strcmp(contacts->contact(i).collision1().c_str(), COLLISION_ITEM) == 0) &&
                      (strcmp(contacts->contact(i).collision2().c_str(), COLLISION_POINT) == 0) ? true : false;


if (collisionCheck)
{
    rewardHistory = collisionCheck ? REWARD_WIN : REWARD_LOSS;

    newReward  = true;
    endEpisode = true;

    return;
}
                                                                    @Chen
```

Fig. 5: Definition of Collision Reward.

3.2 Arm Control

Both Task 1 and Task 2 implement position control but not velocity control, namely, the three joint angles are control directly. There are two outputs for every action - increase or decrease in the joint angles. These outputs is based on whether the action is even or odd, as Fig.6 shown.

```
// TODO - Set joint position based on whether action is even or odd.
float joint = ref[action/2] + (1 - 2 * (action % 2)) * actionJointDelta;
                                                                    @Chen
```

Fig. 6: Position Control.

3.3 Hyperparameters Tuning

DQN agents in both tasks are created with the same hyperparameter, as shown in Table 1. The input size is decreased from 512 to 64 for efficiency concern, but is sufficient to perform the required results. The increase of LSTM_SIZE can improve DQN stability facilitating the training procedure. A small batch size may lead to non-convergence of training process and insufficient smoothness of training curve. On the contrary, a large batch size can lead to local convergence, and batch size of 32 is proved to be a good value in this project.

The Adam optimizer is an extension to SGD. It is computationally efficient, and typically require little tuning.The learning rate is set to a relatively lower number of 0.01, which can make the training process maintain a lower update range, and prevent damage to previously learned features. Replay memory lead to more efficient use of previous experience, and the setting value of 10000 provide sufficient information for the model to train on.

Hyperparameters related to the definition of reward or penalty is determined by experiments.

TABLE 1: Choice of Hyparameters

Hyperparameters	Value
INPUT_*	64
OPTIMIZER	Adam
LEARNING_RATE	0.01
REPLAY_MEMORY	10000
BATCH_SIZE	32
USE_LSTM	true
LSTM_SIZE	256
REWARD_WIN	1.0
REWARD_LOSS	-1.0
INTERIM_REWARD	3.0
INTERIM_OFFSET	0.5
ALPHA	0.3

4 Results

4.1 Task 1

The DQN agent is able to teach the arm to touch the object using any part of the arm with an accuracy of 91%, as Fig.7 shown.

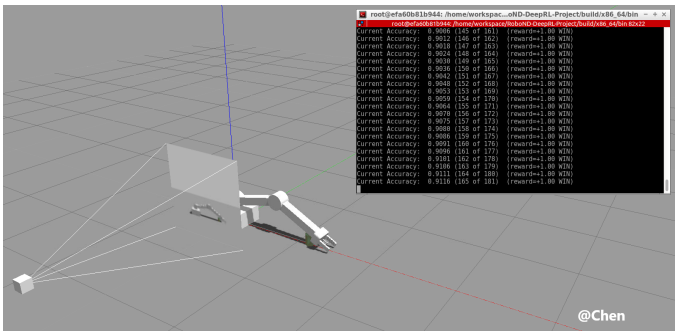

Fig. 7: Task 1 Result.

4.2 Task 2

The DQN agent is able to teach the arm to touch the object using gripper base with an accuracy of 85%, as Fig.8 shown.

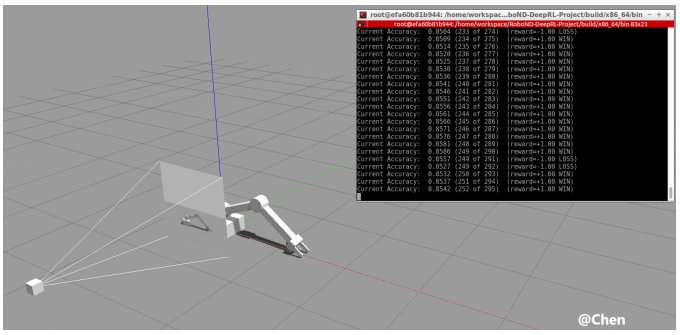

Fig. 8: Task 2 Result.

5 Discussion

The hyperparameters setting in two task2 are the same, and both tasks achieve an acceptable result. However, in practical applications where the DQN agent can be used, the accuracy of 90% is insufficient. By further tuning the above hyperparameters, the result can be improved. For example, appropriately increasing input size of the images, or batch size can provide good feature quality, increasing LSTM size can stablize the DQN training procedure and improve the accuracy. However, large hyperparameters can lead to a vast computational resource, and put forward higher requirements for hardware.

The API used in this project is developed with C++, which is considered as one of the most efficient languages. Especially when C++ application is deployed in embedded system, the advantage is more obvious. C++ is of great significance to real-world robotic systems.

6 Conclusion / Future work

A DQN agent for the robotic arm is created and several reward functions are defined. By tuning essential hyperparameters of the agent in $ArmPlugin.cpp$ file in C++ API and training the DQN, robotic arm achieves the two requirements.

Future works include utilize the framework to control a robotic arm with enhanced capabilities like picking objects, and control robotic arm with more degrees of freedom.

References

[1] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3389–3396, IEEE, 2017.

[2] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[3] WIKIPEDIA, "Reinforcement learning." https://en.wikipedia.org/wiki/Reinforcement_learning.

[4] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[5] WIKIPEDIA, "Reinforcement learning." https://en.wikipedia.org/wiki/Q-learning.

[6] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.