RoboND Project: Deep RL Arm Manipulation

Milton Wong

Abstract—This project focus on building a DQN agent and define reward functions to teach a robotic arm to the robot arm touch the object of interest The task of this project was controlling robotic arm with three joints in the Gazebo simulator with help of the Deep Q Network. That has been implemented in C++ programming language by accessing DQN API with the help of $dqnAgent$ class.

Index Terms—Deep Reinforcement Learning, Q Learning, Arm Manipulation.

1 Introduction

This report summarizes the results for *Deep Reinforcement Learning* (DRL) [1] based Arm Manipulation task. In this project, Four degrees of freedom (DoF) robot is simulated within Gazebo to demonstrate the effectiveness of deep Q learning method in robotic grasping applications. The simulated environment reported the following information:

1) Joint State: The position of each joint (and velocity using approximate numerical differentiation)
2) Target Object Bounding Box: The pose/dimensions of the bounding box of object to be grasped
3) Collisions: Collisions between the ground, elements of the robot arm, and the target object.
4) Gripper Bounding Box: The pose/dimensions of the bounding box surrounding the end effector

The baseline project task was to first touch the target with part of the any part of the arm (including the gripper) in 90% of all simulations, then, as a second task, contact the target with only the gripper in 80% of all simulations. For these simulations, the small tube target, would normally be spawned in the same location each time.

2 Reward Function

Both task are using the same reward function and params except the episode end condition. Win and loss reward values was selected as following:

```
#define REWARD_WIN 15.0f
#define REWARD_LOSS -15.0f
#define REWARD_ALPHA 0.5f
```

$REWARD\_ALPHA$ is a smoothing parameter that determine the contribution of a running average delta distance to the goal $avgGoalDelta$. The value 0.5 was determined experimentally and works great for both tasks.

Transitional reward consists of $avgGoalDelta$ with a multiplier of 4 - selected experimentally to balance $REWARD\_WIN/REWWARD\_LOSS$ values.

And the second term -0.2 is a penalty for each move that prevents the agent to just hold the position and stop moving at all.

The same reward function and values are used for both tasks. Joint control was selected as positional for both tasks. Also quick experiments show that velocity control works good as well. However the final params wasnt tested with the velocity control and may require further tuning of reward function values and DQN parameters.

2.1 Gripper hitting the ground

Assigned reward function is always in large negative value because gripper hitting the ground is critical for the task. The reward becomes better when gripper position is closer to a target. Additionally a threshold value and action joint interval are decreased from default value to avoid the collision.

```
actionJointDelta = 0.10f;
groundContact = 0.01f;
checkGroundContact
      = ( gripBBox.min.z < groundContact );
rewardHistory
        = 10*REWARD_LOSS
        + 500*(gripBBox.min.x - propBBox.min.x);
newReward = true;
endEpisode = true;
```

2.2 Collision between the Arm and the Object

Touching the object with the arm is goal of the Task1. Therefore, reward becomes large positive value. And condition is provided as below:

```
collisionCheck = ( collision1 ==
        "tube::tube_link::tube_collision" );
if (collisionCheck){
        rewardHistory = 100*REWARD_WIN;
        newReward = true;
        endEpisode = true;
}
```

2.3 Collision between the Gripper Base and the Object

Touching the object with the arms gripper base is the goal of the Task2. The reward function is provided as below:

```
collisionCheck1 = ( collision1 ==
        "tube::tube_link::tube_collision");
collisionCheck2 = ( collision2 ==
        "arm::gripperbase::gripper_link") ||
        ( collision2 ==
        "arm::gripper_middle::middle_collision");
if( collisionCheck1 ){
```

```
if (collisionCheck2){
        rewardHistory = 100*
                        REWARD_WIN;
        newReward = true;
        endEpisode = true;
        }
else{
        rewardHistory = REWARD_LOSS
                        / 5.0;
        newReward = true;
        endEpisode = true;
        }
}
```

3 HYPERPARAMETERS

Hyperparameters for the DQN Agent was selected as following:

```
#define INPUT_WIDTH 64
#define INPUT_HEIGHT 64
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.01f //Task1 = 0.01f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 32
#define USE_LSTM true
#define LSTM_SIZE 256
```

During such backward tests were determined that lower $LEARNING\_RATE$ of 0.01f works better for Task1. And higher $LEARNING\_RATE$ of 0.1f is more important for Task2. During parameters tuning, higher $LSTM\_SIZE$ showed better results thus it applies 256, which means that agent can learn more complex feature in its state and track them during the subsequent actions. $BATCH\_SIZE$ higher than 32 showed worse results and wasnt been able to learn past 56% of accuracy for Task2. ADAM [2] optimizer showed worse results than RMSProp [3]. As a result we have one set of hyperparameters that works for both objectives with the only change in $LEARNING\_RATE$ between them.

4 RESULTS

The training environment is Udacity Project Workspace with enabled GPU.

4.1 Task 1

Tests of the selected parameters on Task1 showed good results in episode round 230 with the achieved accuracy 91% for the last 100 episodes as shown in Fig. 1.

Continue the training process for the next 1000 episodes, it showed (Fig. 2) the further improvements up to the accuracy 100% of the last 100 episodes.

To achieve this result, some hyper-parameters tuning are done. After iterations, convergence behavior is affected by initial random state a little. In other words, movement of arm slightly changes for each run.

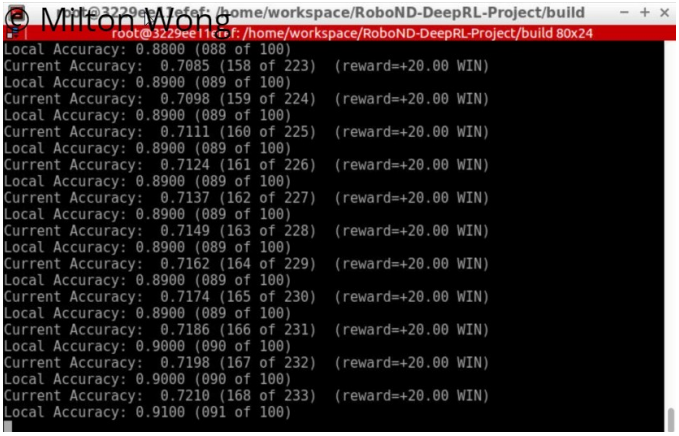

Fig. 1. Accuracy in episode 233 for Task 1.

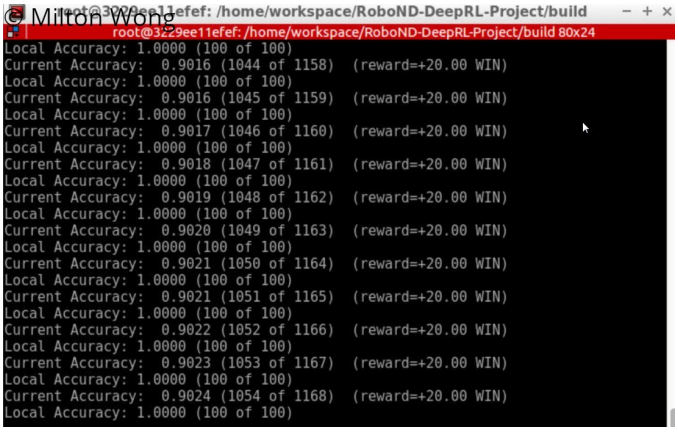

Fig. 2. Accuracy in episode 1168 for Task 1.

4.2 Task 2

For the second task, it was much harder to find the right balance of the hyperparameters but after around 20 hours of tests on GPU, it was converged showing good enough results. In just 267 episodes, local accuracy of 95% on the last 100 episodes were achieved for Task 2. (Fig. 3)

To achieve this result, a lot of hyper-parameter tuning are required compared to Task 1 done. The behavior of arm is very sensitive to reward functions and hyper-parameters. For example, arm tends to avoid collision with ground with hyper-parameter of Task1. In Task2, however, arm should not only avoid collision with ground but also bend its joint to have its gripper with the object. To have only the gripper base of the robot arm touch the object, reward function is adjusted to avoid touch with middle parts of the arm and the object by adding reward loss depend on the distance between the gripper and the object. And the arms behavior largely depends on its initial state.

5 FUTURE WORK

How could we achieve the best result in the shortest amount of episodes with the same parameters of DQN Agent and reward function for both task1 and task2 simultaneously?

It's promising to joint the training of the agent on both tasks simultaneously.

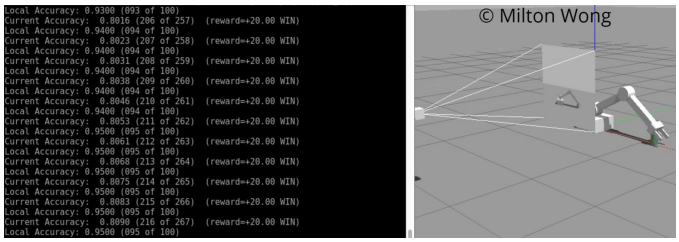

Fig. 3. Accuracy in episode 267 for Task 2.

The other interesting task is to try the trained agent on different arm configurations (link length, joint, shapes, etc) and observe how transferable capability our agent achieves between different arms.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[3] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.