Report: Deep RL Arm Manipulation

Xuan He

Abstract—In this project, Deep-Q-Networks algorithm was applied to train the robotic arm to touch the object under constraints in simulated environment. There are two goals: (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. Those two goals were successfully achieved using Gazebo simulated environment with a C++ plug-in interfacing between Gazebo Simulated Robot and the DQN agent.

Index Terms—Robotic arm manipulation, Deep Reinforcement Learning,

1 Introduction

In robotics area, reinforcement learning is a classical research area. Reinforcement learning (RL) enables a robot to autonomously discover an optimal behavior through trial-and-error interactions with its environment.
Consider, for example, attempting to train a robot to return a table tennis ball over the net. In this case, the robot might make an observations of dynamic variables specifying ball position and velocity and the internal dynamics of the joint position and velocity. This might capture well the states of the system providing a complete statistic for predicting future observations. The actions available to the robot might be the torque sent to motors or the desired accelerations sent to an inverse dynamics control system. A function is called the policy that generates the motor commands (i.e., the actions) based on the incoming ball and current internal arm observations (i.e., the state) [1].
A reinforcement learning problem is to find a policy that optimizes the long term sum of rewards; a reinforcement learning algorithm is one designed to find such a (near)-optimal policy.

2 Background

In reinforcement learning, an agent tries to maximize the accumulated reward over its life-time. In an episodic setting, where the task is restarted after each end of an episode, the objective is to maximize the total reward per episode. If the task is ongoing without a clear beginning and end, either the average reward over the whole life-time or a discounted return (i.e., a weighted average where distant rewards have less influence) can be optimized. In such reinforcement learning problems, the agent and its environment may be modeled being in a state s S and can perform actions a A, each of which may be members of either discrete or continuous sets and can be multidimensional [1]. Classical reinforcement learning approaches are based on the assumption that we have a Markov Decision Process (MDP) consisting of the set of states S, set of actions A, the rewards R and transition probabilities T that capture the dynamics of a system. Transition probabilities (or densities in the continuous state case) T(s,a,s) = P(s—s,a) describe the effects of the actions on the state. Transition probabilities generalize the notion of deterministic dynamics to allow for modeling outcomes are uncertain even given full state.
DQN uses experience replay to keep training the neural networks and find the approximation of policy function. In this project, the power of DQN is leveraged with a classical arm manipulation problem. The objectives of the project consist of two parts:

- Objective 1: Have any part of the robot arm touch the object, wtih at lease a 90% accuracy for a minimum of 100 runs.
- Objective 2: Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy for a minimum of 100 runs.

3 Reward Functions

The output of the Deep Q-Network (DQN) is usually mapped to a particular action, which is the control of each joint for the simulated robotic arm in this project. Control of the joint movements can be through velocity, position, or a mix of both. In case of this project position control approach was selected.

The reward system was designed to train the robot to have any part of the robot arm touch the object of interest in one attempt then have the gripper base of the robot arm touch the object in a second attempt. Positive and negative rewards are defined as numeric constants *REWARD_WIN* and *REWARD_LOSS*. A win is issued when task is accomplished. A loss is issued when the arm when the arm touches the ground or when the length of the episode exceeds 100 steps in time. The occurrence of any of these events will end in the episode.

Since positive rewards from touching the prop by the gripper link are very sparse, an interim reward function was designed. Its purpose is to guide the robot arm towards the target. The relevant measure is the distance δ_t between the gripper and target object. For consecutive steps $t-1$ and t, the difference is

$$\delta_t = d_t - d_{t-1}$$

In order to obtain a smooth reward function, the moving average of the recorded delta values was computed, where

the parameter $ALPHA$ should be tuned between the value 0 to 1.

$$\begin{aligned} distDelta &= lastGoalDistance - distGoal \\ avgGoal &= (avgGoalDelta * ALPHA) \\ &+ (distDelta * (1.0f - ALPHA)) \end{aligned} \tag{1}$$

A linear function of the $avgGoalDelta$ function is proportional to the average speed towards the goal and is a good candidate for the interim reward function.

$$\begin{aligned} rewardHistory &= (INTERIM\_REWARD * \\ avgGoalDelta) &- INTERIM\_OFFSET; \end{aligned} \tag{2}$$

The parameter $INTERIM\_REWARD$ and $INTERIM\_OFFSET$ are also constant parameters that are tuned in the experiment. From experiments, the reward function works both for objective 1 and objective 2.

4 Hyperparameters

The size of the input was set to 64 x 64 the same size as the camera image. The long-short-term memory architecture was enabled. The RMSprop and Adam optimizer were experimented. RMSprop was used during the simulation runs for objective 1. Adam was used in the reward function design experiments for objective 2. From objective 1 to objective 2, the batch size was increased from 32 to 256 to improve training result. The LSTM size was increased from 64 to 256, which should allow the agent to memorize more information.

During the simulation runs for objective 2, the start value of the exploration parameter EPS_START was tuned between 0.9 and 0.7. The end value of the exploration parameter EPS_END was tuned. The learning rate was tuned between the values 0.08 and 0.1. The performance of each configuration was determined using the accuracy output during the simulations. The final configurations for both objective simulation are listed below.

	Objective 1	Objective 2
EPS_START	0.7	0.7
EPS_END	0.02	0.02
EPS_DECAY	200	200
INPUT_WIDTH	64	64
INPUT_HEIGHT	64	64
LEARNING_RATE	0.1	0.1
BATCH_SIZE	32	256
OPTIMIZER	RMSprop	Adam
LSTM_SIZE	64	256
REWARD_WIN	20	20
REWARD_LOSS	-20	-20
INTERIM_REWARD	4	4
INTERIM_OFFSET	0.3	0.3
ALPHA	0.4	0.4

5 Result and Discussion

Both objectives were achieved. The best performance runs were observed with the linear reward function. For objective 1, the final accuracy is around 0.95 and for objective 2, the final accuracy is around 0.90.

The largest positive effect on the agents performance was observed when lowering the value of moving average

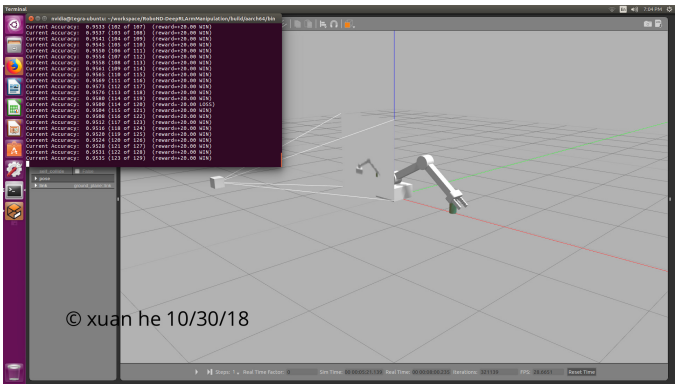

Fig. 1. Objective 1 with training output and Gazebo output

ALPHA parameter from 0.9 to 0.4. This finding emphasizes the importance of designing a smooth reward function.

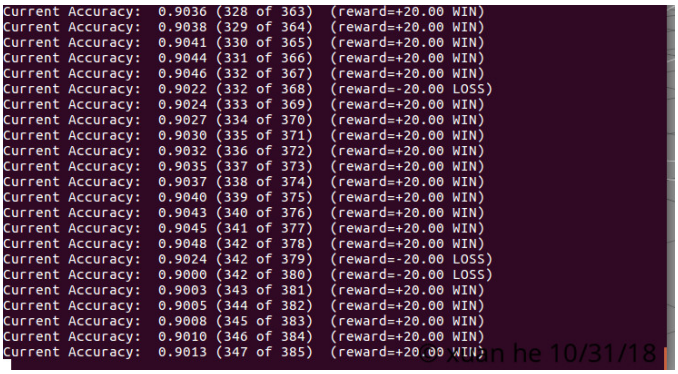

Fig. 2. Objective 2 with training output

The best performing configurations for each objective are shown in the table. The number of episodes it took the agent to reach the threshold accuracy varied between 200 and 700, depending on the initial conditions of the simulation run. During the simulation runs, the accuracy improved slowly over time.

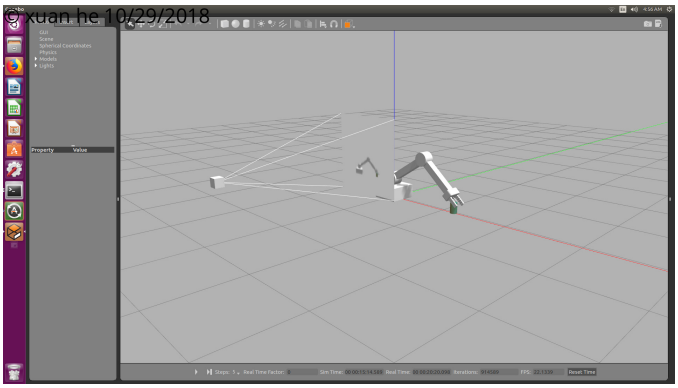

Fig. 3. Objective 2 with Gazebo output

6 Conclusion / Future work

The two objectives of the project were achieved. To further improve the reinforcement-learning framework, we

can keep tuning the remaining DQN hyper-parameters like the discount factor, size of replay memory and epsilon decay rate. The algorithm can also be implemented on real hardware such as Jetson TX2 combined with robotic arm. However, retraining is necessary for the system to learn from the real environment.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st ed., 1998.