Deep RL Arm Manipulation

chun pook

Abstract—Deep Reinforcement Learning can learn complex robotic skills from camera or other input sensory input. In this Project, using the camera raw image as input to the system. A vision based Reinforcement Learning system was first trained to allow the robotic arm to be able to touch the object of interest with 90% accuracy, and then the system was trained to have the gripper touching the object with 80% accuracy.

Index Terms—Robot, Deep RL, DQN Agent, Robotic ARM.

1 Introduction

Deep Reinforcement learning has been successfully applied to various domain such as video games. A Vision based Reinforcement Learning using image captured by camera as input can learn to manipulate robotic arm.

In this project, A Deep Q-Learning Network (DQN) was trained to control the robotic arm performing the following Two tasks in a simulated gazebo environment.

1) have any part of the robot arm touch the object of interest, with at least a 90% accuracy with a minimum of 100 run.
2) have only the gripper base of the robot arm touch the object, with at least a 80% accuracy with a minimum of 100 run.

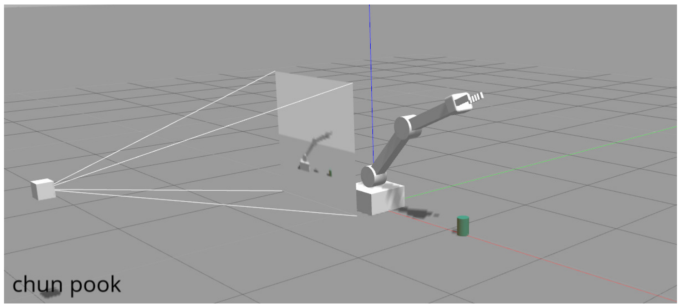


Fig. 1. Robotic arm running in gazebo environment

Robots needs to respond in real-time to environmental changes, so computational performance matters. In this project, a C/C++ API(Application Programming Interface) was used to allow Python code written in PyTorch to pass memory objects between the user's application and Torch efficiently, which improve the performance.

2 Background

The project only use position based arm control, velocity based is not being used. So VELOCITY_CONTROL was set to false.

2.1 Hyper Parameters

The image size was reduced from 512x512 to 64x64 to reduce complexity and the computation workload.

Adam (Adaptive Moment Estimation) is an adaptive learning rate method similar to RMSprop, but adds bias-correction and momentum to RMSprop.In this project, Adam is chosen as the optimizer with a learning rate set to be 0.01.

Replay memory stores the transitions that the agent observes, allowing algorithm to reuse this data later on. By Randomly sampling from this memory, the transitions that build up a batch are decorrelated. Tables of sufficient size should be used to store the data, picking 10000 does not consume too much of memory and allow the system to store sufficient data.

LSTM was also used as part of the network, which allows the network to take into account of multiple past frames from the camera sensor instead of a single frame. and a size of LSTM_SIZE set to 256 seems to work fine.

TABLE 1
hyper parameters

INPUT_WDITH	64
INPUT_HEIGHT	64
OPTIMIZER	Adam
LEARNING_RATE	0.01f
REPLAY_MEMORY	10000
BATCH_SIZE	32
USE_LSTM	true
LSTM_SIZE	256

2.2 reward

Reward was issued under different circumstances.

1) Robot arm touching the object of interest, for Task 1.

```
rewardHistory = REWARD_WIN * 10;
newReward = true;
endEpisode = true;
```

2) Gripper touching the object of interest. unlike the first task, there is a penalty for not having the Gripper touching the object.

```
if (strcmp(contacts->contact(i).collision2().c_str(),
           COLLISION_POINT) == 0) {
    rewardHistory = REWARD_WIN * 10;
} else {
    rewardHistory = REWARD_LOSS * 5;
}
newReward = true;
endEpisode = true;
```

3) Taking more than 100 steps reaching the object. There is a penalty for the system taking lot of steps to achieve the goal. In this case it is considered to be a failure when it takes more than 100 steps.

```
rewardHistory = REWARD_LOSS * 0.1;
newReward     = true;
endEpisode    = true;
```

4) Arm touching the ground. There is a penalty if the arm ever touches the ground.

```
rewardHistory = REWARD_LOSS;
newReward     = true;
endEpisode    = true;
```

5) Arm moving closer or away from the object.The system get rewards when arm move closer to the target, but get penalized if it is moving further apart. And to improve smoothness, a smoothed moving average was used to calculate the delta of the distance from the goal.

```
const float distDelta  = lastGoalDistance - distGoal;
const float alpha = 0.9f;

// compute the smoothed moving average of the delta
// of the distance to the goal
avgGoalDelta  = avgGoalDelta * alpha + distDelta * (1.0 - alpha);
if (avgGoalDelta > 0.0)
{
    //distance getting smaller
    rewardHistory = REWARD_WIN*0.1;
} else {
        rewardHistory =  0.1*REWARD_LOSS * distGoal;
}
```

6) Arm not moving, sometimes the arm is not moving. to keep it moving, a penalty was imposed if change in distance is too small.

```
if (fabs(avgGoalDelta) < 0.01)
    rewardHistory += 0.1 * REWARD_LOSS;
```

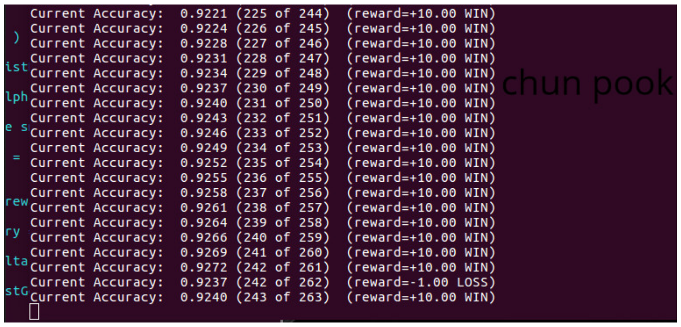
3 RESULTS

Both objectives were achieved, and the accuracy improves with number of runs.

However first task is a simpler task to learn, and it took smaller number of steps to achieve 90% as compared to achieving 80% for the second task.

3.1 First Objective

have any part of the robot arm touch the object of interest, with at least a 90% accuracy with a minimum of 100 run.

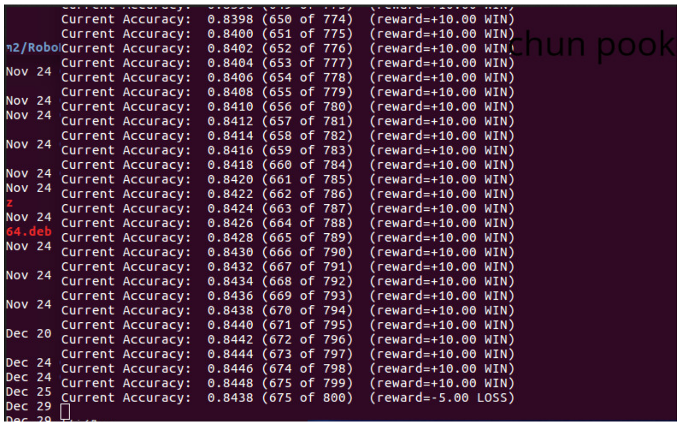


```
Current Accuracy:  0.9221 (225 of 244)  (reward=+10.00 WIN)
Current Accuracy:  0.9224 (226 of 245)  (reward=+10.00 WIN)
Current Accuracy:  0.9228 (227 of 246)  (reward=+10.00 WIN)
Current Accuracy:  0.9231 (228 of 247)  (reward=+10.00 WIN)
Current Accuracy:  0.9234 (229 of 248)  (reward=+10.00 WIN)
Current Accuracy:  0.9237 (230 of 249)  (reward=+10.00 WIN)
Current Accuracy:  0.9240 (231 of 250)  (reward=+10.00 WIN)
Current Accuracy:  0.9243 (232 of 251)  (reward=+10.00 WIN)
Current Accuracy:  0.9246 (233 of 252)  (reward=+10.00 WIN)
Current Accuracy:  0.9249 (234 of 253)  (reward=+10.00 WIN)
Current Accuracy:  0.9252 (235 of 254)  (reward=+10.00 WIN)
Current Accuracy:  0.9255 (236 of 255)  (reward=+10.00 WIN)
Current Accuracy:  0.9258 (237 of 256)  (reward=+10.00 WIN)
Current Accuracy:  0.9261 (238 of 257)  (reward=+10.00 WIN)
Current Accuracy:  0.9264 (239 of 258)  (reward=+10.00 WIN)
Current Accuracy:  0.9266 (240 of 259)  (reward=+10.00 WIN)
Current Accuracy:  0.9269 (241 of 260)  (reward=+10.00 WIN)
Current Accuracy:  0.9272 (242 of 261)  (reward=+10.00 WIN)
Current Accuracy:  0.9237 (242 of 262)  (reward=-1.00 LOSS)
Current Accuracy:  0.9240 (243 of 263)  (reward=+10.00 WIN)
```

Fig. 2. Any Part of the Robot Arm Touching the Object

3.2 Second Objective

have only the gripper base of the robot arm touch the object, with at least a 80% accuracy with a minimum of 100 run.



```
Current Accuracy:  0.8398 (650 of 774)  (reward=+10.00 WIN)
Current Accuracy:  0.8400 (651 of 775)  (reward=+10.00 WIN)
Current Accuracy:  0.8402 (652 of 776)  (reward=+10.00 WIN)
Current Accuracy:  0.8404 (653 of 777)  (reward=+10.00 WIN)
Current Accuracy:  0.8406 (654 of 778)  (reward=+10.00 WIN)
Current Accuracy:  0.8408 (655 of 779)  (reward=+10.00 WIN)
Current Accuracy:  0.8410 (656 of 780)  (reward=+10.00 WIN)
Current Accuracy:  0.8412 (657 of 781)  (reward=+10.00 WIN)
Current Accuracy:  0.8414 (658 of 782)  (reward=+10.00 WIN)
Current Accuracy:  0.8416 (659 of 783)  (reward=+10.00 WIN)
Current Accuracy:  0.8418 (660 of 784)  (reward=+10.00 WIN)
Current Accuracy:  0.8420 (661 of 785)  (reward=+10.00 WIN)
Current Accuracy:  0.8422 (662 of 786)  (reward=+10.00 WIN)
Current Accuracy:  0.8424 (663 of 787)  (reward=+10.00 WIN)
Current Accuracy:  0.8426 (664 of 788)  (reward=+10.00 WIN)
Current Accuracy:  0.8428 (665 of 789)  (reward=+10.00 WIN)
Current Accuracy:  0.8430 (666 of 790)  (reward=+10.00 WIN)
Current Accuracy:  0.8432 (667 of 791)  (reward=+10.00 WIN)
Current Accuracy:  0.8434 (668 of 792)  (reward=+10.00 WIN)
Current Accuracy:  0.8436 (669 of 793)  (reward=+10.00 WIN)
Current Accuracy:  0.8438 (670 of 794)  (reward=+10.00 WIN)
Current Accuracy:  0.8440 (671 of 795)  (reward=+10.00 WIN)
Current Accuracy:  0.8442 (672 of 796)  (reward=+10.00 WIN)
Current Accuracy:  0.8444 (673 of 797)  (reward=+10.00 WIN)
Current Accuracy:  0.8446 (674 of 798)  (reward=+10.00 WIN)
Current Accuracy:  0.8448 (675 of 799)  (reward=+10.00 WIN)
Current Accuracy:  0.8438 (675 of 800)  (reward=-5.00 LOSS)
```

Fig. 3. Only the Gripper Base of the Robot Arm Touching the Object

4 DISCUSSION

Objective 1 was relatively easy to achieve as compared to Objective 2, due to the nature of the problem.

It took significantly smaller number of runs to train the system to have 90% for First task as compared to achieving 80% for the second task.

5 CONCLUSION / FUTURE WORK

The project was ran on native ubuntu xenial x86_64 platform, and both objectives were achieved using position based control of arm joints.

However building the system in ubuntu xenial 16.04, is complicated by the fact that the torch code base seems to be outdated and incompatible, especially the Protobuf and torch version.

Further improvements will include

1) implements object randomization, increasing the arm reach as suggested in the project challenge section
2) implements velocity based, instead of position based control of arm joints
3) There is already a penalty if number of steps is over 100, it can be further refined by having the reward function take into account of the current episode frame number, for each step.

6 REFERENCE

- jetson reinforcement learning dusty-nv on github https://github.com/dusty-nv/jetson-reinforcement.
- An Overview of gradient descent optimization algorithms, http://ruder.io/optimizing-gradient-descent/index.html