Training a Robotic Arm using Deep Q-Network

Peng Xu

Abstract—A Deep Q-Network is used to train a 2 degree Robotic Arm to grasp the tube.

Index Terms—Robotic Arm, Deep Reinforcement Learning, Deep Q-Network, Gazebo.

1 Introduction

THE project goal is to create a DQN (Deep Q-Network) agent and define reward functions to teach a robotic arm to carry out two primary objectives:

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

Reinforcement Learning is a type of machine learning that allows you to create AI agents that learn from the environment by interacting with it. Just like how we learn to ride a bicycle, this kind of AI learns by trial and error. As seen in Fig. 1, the robot represents the AI agent, which acts on the environment. After each action, the agent receives the feedback. The feedback consists of the reward and next state of the environment. The reward is usually defined by a human. If we use the analogy of the bicycle, we can define reward as the distance from the original starting point.

With algorithms such as DQN, one can learn by choosing the actions and observing their results directly in the environment.

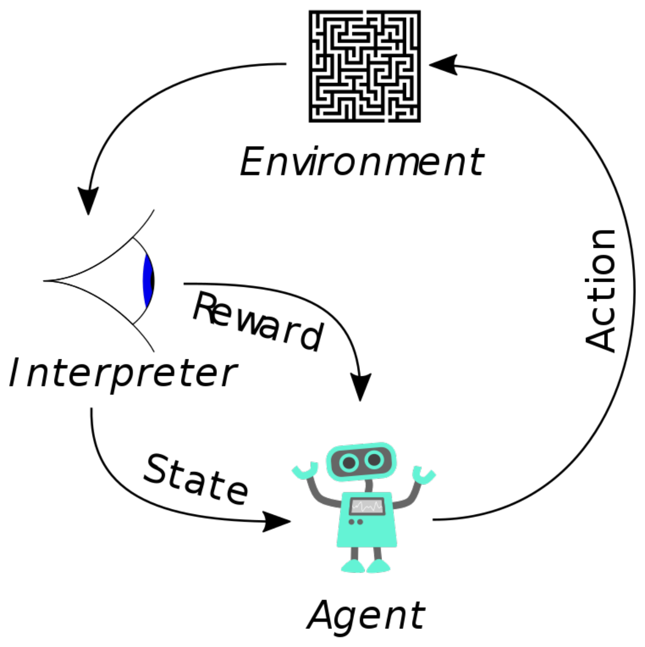


Fig. 1. How an agent interacts with the environment.

2 Formulation

2.1 Reward Functions

The reward function is a crucial component in guiding the DQN agent through the process of learning to manipulate the arm. After every end of episode (EOE), a particular reward is issued, depending on the triggering event.

2.1.1 Reward for Timeout

REWARD_LOSS is issued when the current episode exceeds 100 steps.

2.1.2 Reward for Robot Gripper Hitting the Ground

REWARD_LOSS is issued when the arm contacts the ground.

Whether the arm contacts the ground is detected using the GetBoundingBox() function from the Gazebo API which provides the min/max XYZ values of the arm. Collisions can be thought to happen when the Z value of the ground is below the one of the arm.

```
bool checkGroundContact = (gripBBox.max.z <=
    groundContact) && (gripBBox.min.z <=
    groundContact);

if(checkGroundContact)
{

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

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

2.1.3 Issue an interim reward based on the distance to the object

In most steps, the robot arm cannot end the current episode by the desired contact. But it should be always either moving closer or further to the goal. An interim reward function was in this concept implemented by measuring how far away from the desired contact position which would help guide the arm towards the object and reduce the training time. The reward is proportional to the change in distance between the arm and the object. If the arm is moving away from the object, the interim reward will be negative. Else, the interim reward will be positive. The raw values cause the arm to have a jerking movement, so they are smoothed by taking a moving average. Finally, due to the agent's tendency to remain still, a time penalty was introduced to encourage the agent to finish as quickly as possible.

```
const float distGoal = BoxDistance(propBBox,
    gripBBox); // 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 * (1.0f - ALPHA));

if(avgGoalDelta > 0)
  rewardHistory = REWARD_WIN;
else
  rewardHistory = REWARD_LOSS * distGoal;

// we want to be moving
if (abs(avgGoalDelta) < .001f)
  rewardHistory += REWARD_LOSS;

newReward  = true;
}

lastGoalDistance = distGoal;
```

2.1.4 *Issue a reward based on collision between the arm and the object*

REWARD_WIN is issued when any part of the arm contacts the object in Task 1 or when only the gripper base contacts the object in Task 2.

For task 1:

```
bool collisionCheckArm =
(strcmp(contacts->contact(i).collision1()
.c_str(), COLLISION_ITEM) == 0);
```

For task 2:

```
bool collisionCheckArm =
(strcmp(contacts->contact(i).collision1()
.c_str(), COLLISION_ITEM) == 0) &&
(strcmp(contacts->contact(i).collision2()
.c_str(), COLLISION_POINT) == 0);
```

If the disired contact is detected, a positive reward, REWARD_WIN, would be issued and end the current episode. Else, a small punishment or a negative reward should be assigned to the reward function for the time consuming.

```
if (collisionCheckArm)
{
  rewardHistory = REWARD_WIN;
  newReward = true;
  endEpisode = true;
  return;
} else {
  rewardHistory = REWARD_LOSS * 0.1f;
  newReward = true;
  endEpisode = false;
}
```

2.2 Hyperparameters

In Q-Learning, epsilon-Greedy is a typical exploration method used to encourage an agent to explore more of the state-action space. The larger the value, the more frequently a random action is chosen instead of one with the highest q-value. EPS_START defines the initial value. 0.9f ensures the agent is exposed to many state-action spaces. EPS_END defines the final value. 0.0f ensures that only learned actions with the highest q-value is used once it has been sufficiently exposed. EPS_DECAY defines the rate by which the value decays from initial to final over time.

```
// Define DQN API Settings

#define INPUT_CHANNELS 3
#define ALLOW_RANDOM true
#define DEBUG_DQN false
#define GAMMA 0.9f
#define EPS_START 0.9f
#define EPS_END 0.05f
#define EPS_DECAY 200
```

The input dimensions were set to match that of the image captured by the camera: 64 x 64 pixels. "Adam" was adopted as the optimizer. LSTM was set to true to enable the agent to consider previously encountered states. LSTM_SIZE was set to 256 as default, strengthening the agent's ability to consider more complex moves. A LEARNING_RATE of 0.1f was sufficient for Task 1, but not for Task 2 as it took much too long learn. For better ensurance of convergence, the value was decreased to 0.01f. The BATCH_SIZE was decreased from 64 to 32 considering the computing power.

```
#define INPUT_WIDTH 64
#define INPUT_HEIGHT 64
#define OPTIMIZER "Adam"
#define LEARNING_RATE 0.01f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 32
#define USE_LSTM true
#define LSTM_SIZE 256
```

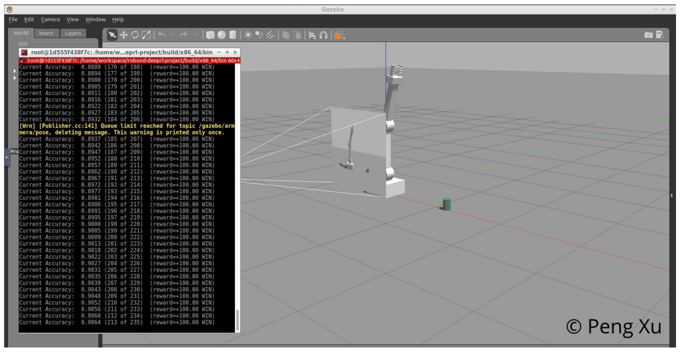
REWARD_INTERIM was the reward incrementally issued to lead the agent towards the object. ALPHA was used to calculate a moving average of the delta distances. To deter the agent from standing still, a TIME_PENALTY was introduced. All three of these were tuned concurrently to find the values that issued appropriate interim rewards.

```
// Reward Parameters
#define REWARD_WIN 30.0f
#define REWARD_LOSS -30.0f
#define REWARD_INTERIM 5.0f
#define ALPHA 0.6f
#define TIME_PENALTY 0.4f
```

3 Results

3.1 Task 1: Touch Any Part of the Robot Arm

A screen shot is shown as in Fig. 2. The required accuracy 90% was reached after about 220 episodes, which is quite efficient.

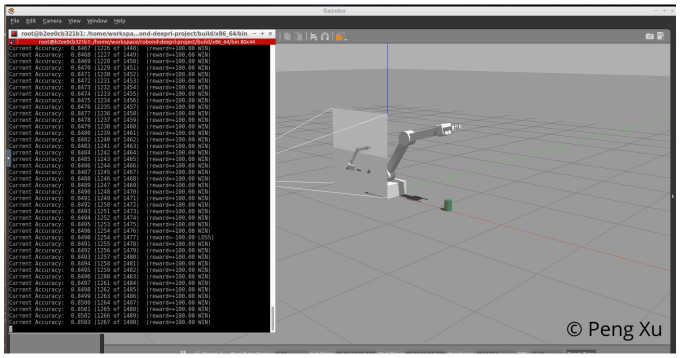


```
root@1d555f438f7c: /home/w...eprl-project/build/x86_64/bin
root@1d555f438f7c: /home/workspace/robond-deeprl-project/build/x86_64/bin 66x4
Current Accuracy:  0.8889 (176 of 198)  (reward=+100.00 WIN)
Current Accuracy:  0.8894 (177 of 199)  (reward=+100.00 WIN)
Current Accuracy:  0.8900 (178 of 200)  (reward=+100.00 WIN)
Current Accuracy:  0.8905 (179 of 201)  (reward=+100.00 WIN)
Current Accuracy:  0.8911 (180 of 202)  (reward=+100.00 WIN)
Current Accuracy:  0.8916 (181 of 203)  (reward=+100.00 WIN)
Current Accuracy:  0.8922 (182 of 204)  (reward=+100.00 WIN)
Current Accuracy:  0.8927 (183 of 205)  (reward=+100.00 WIN)
Current Accuracy:  0.8932 (184 of 206)  (reward=+100.00 WIN)
[Wrn] [Publisher.cc:141] Queue limit reached for topic /gazebo/arm
mera/pose, deleting message. This warning is printed only once.
Current Accuracy:  0.8937 (185 of 207)  (reward=+100.00 WIN)
Current Accuracy:  0.8942 (186 of 208)  (reward=+100.00 WIN)
Current Accuracy:  0.8947 (187 of 209)  (reward=+100.00 WIN)
Current Accuracy:  0.8952 (188 of 210)  (reward=+100.00 WIN)
Current Accuracy:  0.8957 (189 of 211)  (reward=+100.00 WIN)
Current Accuracy:  0.8962 (190 of 212)  (reward=+100.00 WIN)
Current Accuracy:  0.8967 (191 of 213)  (reward=+100.00 WIN)
Current Accuracy:  0.8972 (192 of 214)  (reward=+100.00 WIN)
Current Accuracy:  0.8977 (193 of 215)  (reward=+100.00 WIN)
Current Accuracy:  0.8981 (194 of 216)  (reward=+100.00 WIN)
Current Accuracy:  0.8986 (195 of 217)  (reward=+100.00 WIN)
Current Accuracy:  0.8991 (196 of 218)  (reward=+100.00 WIN)
Current Accuracy:  0.8995 (197 of 219)  (reward=+100.00 WIN)
Current Accuracy:  0.9000 (198 of 220)  (reward=+100.00 WIN)
Current Accuracy:  0.9005 (199 of 221)  (reward=+100.00 WIN)
Current Accuracy:  0.9009 (200 of 222)  (reward=+100.00 WIN)
Current Accuracy:  0.9013 (201 of 223)  (reward=+100.00 WIN)
Current Accuracy:  0.9018 (202 of 224)  (reward=+100.00 WIN)
Current Accuracy:  0.9022 (203 of 225)  (reward=+100.00 WIN)
Current Accuracy:  0.9027 (204 of 226)  (reward=+100.00 WIN)
Current Accuracy:  0.9031 (205 of 227)  (reward=+100.00 WIN)
Current Accuracy:  0.9035 (206 of 228)  (reward=+100.00 WIN)
Current Accuracy:  0.9039 (207 of 229)  (reward=+100.00 WIN)
Current Accuracy:  0.9043 (208 of 230)  (reward=+100.00 WIN)
Current Accuracy:  0.9048 (209 of 231)  (reward=+100.00 WIN)
Current Accuracy:  0.9052 (210 of 232)  (reward=+100.00 WIN)
Current Accuracy:  0.9056 (211 of 233)  (reward=+100.00 WIN)
Current Accuracy:  0.9060 (212 of 234)  (reward=+100.00 WIN)
Current Accuracy:  0.9064 (213 of 235)  (reward=+100.00 WIN)
```

Fig. 2. Task 1: Touch Any Part of the Robot Arm.

3.2 Task 2: Touch the Gripper Base of the Robot Arm

A screen shot is shown as in Fig. 3. The required accuracy 80% was reached after around 1000 episodes.



```
root@b2ee0cb321b1: /home/workspa...ond-deeprl-project/build/x86_64/bin
root@b2ee0cb321b1: /home/workspace/robond-deeprl-project/build/x86_64/bin 80x44
Current Accuracy:  0.8467 (1226 of 1448)  (reward=+100.00 WIN)
Current Accuracy:  0.8468 (1227 of 1449)  (reward=+100.00 WIN)
Current Accuracy:  0.8469 (1228 of 1450)  (reward=+100.00 WIN)
Current Accuracy:  0.8470 (1229 of 1451)  (reward=+100.00 WIN)
Current Accuracy:  0.8471 (1230 of 1452)  (reward=+100.00 WIN)
Current Accuracy:  0.8472 (1231 of 1453)  (reward=+100.00 WIN)
Current Accuracy:  0.8473 (1232 of 1454)  (reward=+100.00 WIN)
Current Accuracy:  0.8474 (1233 of 1455)  (reward=+100.00 WIN)
Current Accuracy:  0.8475 (1234 of 1456)  (reward=+100.00 WIN)
Current Accuracy:  0.8476 (1235 of 1457)  (reward=+100.00 WIN)
Current Accuracy:  0.8477 (1236 of 1458)  (reward=+100.00 WIN)
Current Accuracy:  0.8478 (1237 of 1459)  (reward=+100.00 WIN)
Current Accuracy:  0.8479 (1238 of 1460)  (reward=+100.00 WIN)
Current Accuracy:  0.8480 (1239 of 1461)  (reward=+100.00 WIN)
Current Accuracy:  0.8482 (1240 of 1462)  (reward=+100.00 WIN)
Current Accuracy:  0.8483 (1241 of 1463)  (reward=+100.00 WIN)
Current Accuracy:  0.8484 (1242 of 1464)  (reward=+100.00 WIN)
Current Accuracy:  0.8485 (1243 of 1465)  (reward=+100.00 WIN)
Current Accuracy:  0.8486 (1244 of 1466)  (reward=+100.00 WIN)
Current Accuracy:  0.8487 (1245 of 1467)  (reward=+100.00 WIN)
Current Accuracy:  0.8488 (1246 of 1468)  (reward=+100.00 WIN)
Current Accuracy:  0.8489 (1247 of 1469)  (reward=+100.00 WIN)
Current Accuracy:  0.8490 (1248 of 1470)  (reward=+100.00 WIN)
Current Accuracy:  0.8491 (1249 of 1471)  (reward=+100.00 WIN)
Current Accuracy:  0.8492 (1250 of 1472)  (reward=+100.00 WIN)
Current Accuracy:  0.8493 (1251 of 1473)  (reward=+100.00 WIN)
Current Accuracy:  0.8494 (1252 of 1474)  (reward=+100.00 WIN)
Current Accuracy:  0.8495 (1253 of 1475)  (reward=+100.00 WIN)
Current Accuracy:  0.8496 (1254 of 1476)  (reward=+100.00 WIN)
Current Accuracy:  0.8490 (1254 of 1477)  (reward=-100.00 LOSS)
Current Accuracy:  0.8491 (1255 of 1478)  (reward=+100.00 WIN)
Current Accuracy:  0.8492 (1256 of 1479)  (reward=+100.00 WIN)
Current Accuracy:  0.8493 (1257 of 1480)  (reward=+100.00 WIN)
Current Accuracy:  0.8494 (1258 of 1481)  (reward=+100.00 WIN)
Current Accuracy:  0.8495 (1259 of 1482)  (reward=+100.00 WIN)
Current Accuracy:  0.8496 (1260 of 1483)  (reward=+100.00 WIN)
Current Accuracy:  0.8497 (1261 of 1484)  (reward=+100.00 WIN)
Current Accuracy:  0.8498 (1262 of 1485)  (reward=+100.00 WIN)
Current Accuracy:  0.8499 (1263 of 1486)  (reward=+100.00 WIN)
Current Accuracy:  0.8500 (1264 of 1487)  (reward=+100.00 WIN)
Current Accuracy:  0.8501 (1265 of 1488)  (reward=+100.00 WIN)
Current Accuracy:  0.8502 (1266 of 1489)  (reward=+100.00 WIN)
Current Accuracy:  0.8503 (1267 of 1490)  (reward=+100.00 WIN)
```

Fig. 3. Task 2: Touch the Gripper Base of the Robot Arm.

4 Discussion

The DQN was able to successfully complete both tasks. Task 1 parameters were consistent in results of 90% within the range of 250 episodes. Task 2 parameters were less consistent which due to the higher requirement of the reward functions. The beginning stage significantly influenced the convergence time. If the beginning episodes happen to perform well it would be more likely to reach the success rate in a shorter time.

5 Future work

Deep Reinforcement Learning methods are promising on training robots. There are several aspects that can be dived deeper, such as

- High dimensional motion control.
- Other more efficient Deep Reinforcement Learning methods.
- More realistic simulation environments prepared for transfer learning.

Those above experiments relys on heavy computation ability namely power GPUs and careful hyper-parameter tuning.