Udacity Deep Reinforcement Learning

Patrick Mockridge

Abstract—Reinforcement learning has a long history in the field of robotics and is used commercially today.The aim of this, the penultimate, Udacity Robotic Software Engineering Nanodegree project is to create a Deep Q-Learning network with the capability to guide a robotic arm to touch a cylindrical shape, in two specified ways, each with their own desired accuracy, in a simulated Gazebo environment. The first task was for the robot arm to touch the cylinder with any part of the arm at least 100 times with greater than 90 percent accuracy. The second task was for only the gripper base to touch touch the cylinder with at least 80 percent accuracy. Both objectives were successfully achieved.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localisation.

1 Introduction

Reinforcement learning (RL) is a field of study within robotics with much heritage. RL draws its origins from psychology and neuroscience regarding how animals learn from their environment.

Recently, with the acceleration of the development of Graphic Processing Units and, thus, highly performant real world Deep Learning applications, the merging of Deep Learning with Reinforcement Learning gave rise to the Deep Q-Learning Network (DQN) in 2015. This architecture is applicable across a wide array of potential applications and is thus of interest to those organisations pursuing Artificial General Intelligence.

The aim of this project was to create a DQN to allow a robotic arm to perform a task without knowing either the kinematic model nor any previous path planning.

This project used a robotic arm with three degrees of freedom simulated in Gazebo and shown in Figure 1.

The robot arm was incentivised to touch the cylinder in various ways. The input to DQN model training is the camera with a profile view of the robot arm's actions in a 2D scene.

The DQN is therefore trained using information from the 2D camera view recording in real time, maximising reward by modifying the position of the joints.

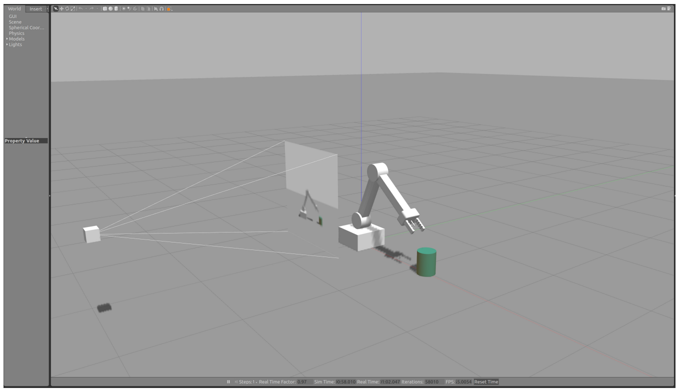
The following project goals were met:

1) Have any part of the robot arm touch the object of interest with 90 percent accuracy
2) Have only the gripper base touch the object of interest with 80 percent accuracy

2 Shared Settings

The following settings were shared across both tasks:

- GAMMA 0.9f: this value was left as default
- EPS_START 0.9f: The chance of randomisation at the beginning of each run was left at the the default value, the higher the value, the more the total phase space of possible actions can be explored.
- EPS_END 0.01f: reduced from default to limit exploitation behaviour.



Steps:1 Real Time Factor: 0.97 Sim Time: 0:58.010 Real Time: 1:02.047 Iterations: 58010 FPS: 5.0054

Fig. 1. Provided Udacity Simulated Project Setup in Gazebo

- EPS_DECAY 150: reduced somewhat in order to achieve accuracy quickly and thus meet project specifications which punish early inaccuracy and phase space exploration, leading to greater GPU workspace resource use, which is a limited commodity for the Nanodegree.

2.1 Joint Control

Position control was decided upon for both tasks. This was inferred from perusing the Robotic Engineering Nanodegree Slack channel.

actionJointDelta was set to 0.12 in order to induce smoother movements from the joint.

2.2 Hyper-parameters

The following hyper-parameters were shared between both tasks:

- INPUT_WIDTH: 64
- INPUT_HEIGHT: 64
- REPLAY_MEMORY: 10000

2.3 Reward Function

Both reward functions were similar.

2.3.1 *Reward Parameters*

- REWARD_WIN: 0.1f
- REWARD_LOSS: -0.1f

The value if these parameters was chosen using the code snippet below, allowing parameters to be tweaked by inspection:

```
if (DEBUG) {
  printf(
  "ArmPlugin - issuing reward
    \%f, EOE=\%s \%s\n",
    (rewardHistory > 0.1f) ? "POS+" :
    (rewardHistory > 0.0f) ? "POS" :
    (rewardHistory < 0.0f) ? "NEG"
    : "ZERO");
  }
  agent->NextReward(rewardHistory,
  endEpisode);
}
```

2.3.2 *Episode Timeout*

If the maximum episode length exceeded 100 frames, a negative reward was issued and the episode is stopped.

2.3.3 *Ground Contact*

In the event of ground contact, a large negative reward (10x base negative reward) is issued, and the episode is stopped.

2.3.4 *Distance to Object*

The following code snippet was used to reward the robot based upon its distance to the object.

```
const float distDelta = lastGoalDistance
                        - distGoal
// distDelta is positive if arm is closer
const float alpha = 0.7
// reducing sensivity to current delta

/* calculate a smoothed out moving average
of the delta of the distance to the goal */
float avg_delta = (avg_delta * alpha) +
                  (distDelta * (1.0 - alpha));
avgGoalDelta = avg_delta

if avgGoalDelta > 0.001f) {
    // +ive reward for getting closer
    if (distGoal > 0.0) {
    // rewardHistory = REWARD_WIN;
    }
    else if (dist Goal < 0.001
            || distGoal == 0.0)
    {
    rewardHistory = REWARD_WIN * 20;
    }
}
else if (avgGoalDelta < 0.0f) {
    //-ive reward when further away
    rewardHistory = REWARD_LOSS * distGoal;
}
else {
    /* cumulative punishment if arm
    is not moving */
    rewardHistory += REWARD_LOSS
}
```

3 Task 1 Settings

It was necessary for Task 1 for the robot to touch the object of interest with any part of itself at least 90 percent of the time after at least 100 repetitions.

3.1 Collision Check

```
bool collisionCheck = (strcmp(
    contacts->contact(i).collision1().c_str(),
    COLLISION_ITEM) == 0) ? true : false;

if (collisionCheck) {
    rewardHistory = REWARD_WIN * 25.0f
    newReward = true;
    endEpisode = true;
    return;
}
```

3.2 Hyperparameters

A simple model was chosen, reflecting the simplicity of the task and the need for the task to achieve success early, not spending many episodes exploring the phase space.

- OPTIMIZER "RMSProp": a common optimiser found to perform well from inspection.
- LEARNING_RATE 0.2f: Was found to be most optimal for achieving necessary accuracy quickly.
- BATCH_SIZE 32: value found to be adequate
- USE_LTSM true: from experimentation this was found to be necessary. LTSM use improved performance.
- LTSM_SIZE 128: adequate for achieving desired accuracy

4 Task 2 Settings

Task 2 was both intended to be and found to be more of an challenge to complete than Task 1. The robot is tasked with touching the item of interest 80 percent of the time over a period of at least 100 episodes.

4.1 Collision Check

Positive rewards are given when the arm gripper touches the item of interest. The faster the collision takes place within each episode, the greater the reward.

```
bool collisionCheck = (strcmp(
    contacts->contact(i).collision1().c_str(),
    COLLISION_ITEM) == 0) ? true : false;

bool collisionCheck_G = (strcmp(
    contacts->contact(i).collision2().c_str(),
    COLLISION_POINT) == 0) ? true : false
if (collisionCheck)
{
    if (collisionCheck_G) {
        rewardHistory = (100 - episodeFrames)
```

```
                                   * REWARD_WIN
        newReward = true;
        endEpisode = true;
        return;
}
```

subsectionHyperparameters

Determining hyperparameters required many iterative experiments, tweaking each value by inepction until optimal values were achieved.

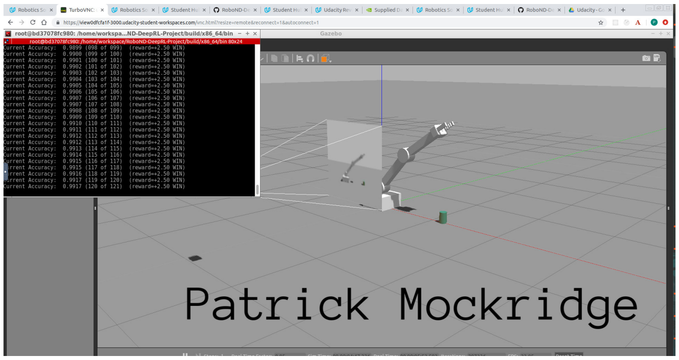
- OPTIMIZER ”Adam: Adam is typically considered to be the best and most versatile optimiser for deep learning.
- LEARNING_RATE 0.1f: As the action of the arm needs to be more precise in Task 2 than in Task 1, so the exploration of the action space needs more precision. Thus the learning rate is lower, but still high enough to achieve necessary action quickly.
- BATCH_SIZE 128: A greater batch size was found to produce smoother, more regular arm movements.
- : LTSM_ITEM 256: As the arm movement was required to be more precise, a larger LSTM stores more information about past movements, hopefully leading to better accuracy.

5 Results

Figures 2 and 3 demonstrates completion of the task.

5.1 Task 1

Task one was completed quickly with little tweaking of parameters.

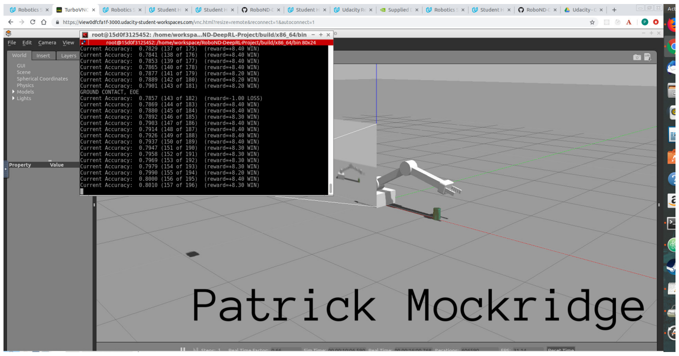


```
root@bd37078fc980: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.9899 (098 of 099)  (reward=+2.50 WIN)
Current Accuracy:  0.9900 (099 of 100)  (reward=+2.50 WIN)
Current Accuracy:  0.9901 (100 of 101)  (reward=+2.50 WIN)
Current Accuracy:  0.9902 (101 of 102)  (reward=+2.50 WIN)
Current Accuracy:  0.9903 (102 of 103)  (reward=+2.50 WIN)
Current Accuracy:  0.9904 (103 of 104)  (reward=+2.50 WIN)
Current Accuracy:  0.9905 (104 of 105)  (reward=+2.50 WIN)
Current Accuracy:  0.9906 (105 of 106)  (reward=+2.50 WIN)
Current Accuracy:  0.9907 (106 of 107)  (reward=+2.50 WIN)
Current Accuracy:  0.9907 (107 of 108)  (reward=+2.50 WIN)
Current Accuracy:  0.9908 (108 of 109)  (reward=+2.50 WIN)
Current Accuracy:  0.9909 (109 of 110)  (reward=+2.50 WIN)
Current Accuracy:  0.9910 (110 of 111)  (reward=+2.50 WIN)
Current Accuracy:  0.9911 (111 of 112)  (reward=+2.50 WIN)
Current Accuracy:  0.9912 (112 of 113)  (reward=+2.50 WIN)
Current Accuracy:  0.9912 (113 of 114)  (reward=+2.50 WIN)
Current Accuracy:  0.9913 (114 of 115)  (reward=+2.50 WIN)
Current Accuracy:  0.9914 (115 of 116)  (reward=+2.50 WIN)
Current Accuracy:  0.9915 (116 of 117)  (reward=+2.50 WIN)
Current Accuracy:  0.9915 (117 of 118)  (reward=+2.50 WIN)
Current Accuracy:  0.9916 (118 of 119)  (reward=+2.50 WIN)
Current Accuracy:  0.9917 (119 of 120)  (reward=+2.50 WIN)
Current Accuracy:  0.9917 (120 of 121)  (reward=+2.50 WIN)
```

Fig. 2. Task 1 completed. Showing 120 succesful episodes from 121: 99.17 percent accuracy

5.2 Task 2

Task two proved more of a challenege, especially if the arm was slow to learn in the beginning, it could later not achieve sufficient accuracy later to make up for early losses. After many attempts to optimise, 157 successful episodes were achieved from 196 in total.



```
root@15d0f3125452: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.7829 (137 of 175)  (reward=+8.40 WIN)
Current Accuracy:  0.7841 (138 of 176)  (reward=+8.40 WIN)
Current Accuracy:  0.7853 (139 of 177)  (reward=+8.40 WIN)
Current Accuracy:  0.7865 (140 of 178)  (reward=+8.40 WIN)
Current Accuracy:  0.7877 (141 of 179)  (reward=+8.20 WIN)
Current Accuracy:  0.7889 (142 of 180)  (reward=+8.20 WIN)
Current Accuracy:  0.7901 (143 of 181)  (reward=+8.20 WIN)
GROUND CONTACT, EOE
Current Accuracy:  0.7857 (143 of 182)  (reward=-1.00 LOSS)
Current Accuracy:  0.7869 (144 of 183)  (reward=+8.40 WIN)
Current Accuracy:  0.7880 (145 of 184)  (reward=+8.40 WIN)
Current Accuracy:  0.7892 (146 of 185)  (reward=+8.30 WIN)
Current Accuracy:  0.7903 (147 of 186)  (reward=+8.40 WIN)
Current Accuracy:  0.7914 (148 of 187)  (reward=+8.40 WIN)
Current Accuracy:  0.7926 (149 of 188)  (reward=+8.40 WIN)
Current Accuracy:  0.7937 (150 of 189)  (reward=+8.40 WIN)
Current Accuracy:  0.7947 (151 of 190)  (reward=+8.30 WIN)
Current Accuracy:  0.7958 (152 of 191)  (reward=+8.30 WIN)
Current Accuracy:  0.7969 (153 of 192)  (reward=+8.30 WIN)
Current Accuracy:  0.7979 (154 of 193)  (reward=+8.30 WIN)
Current Accuracy:  0.7990 (155 of 194)  (reward=+8.20 WIN)
Current Accuracy:  0.8000 (156 of 195)  (reward=+8.40 WIN)
Current Accuracy:  0.8010 (157 of 196)  (reward=+8.30 WIN)
```

Fig. 3. Task 2 completed. Showing 157 successful episodes from 196: 80.10 percent accuracy

6 Discussion

This project was not easy to complete. A lot of new demanding material and ideas, a good understanding of C++ and much iterative tweaking of parameters were necessary.

Deep Q Learning clearly has vast potential in industry, but it is not possible to tweak parameters manually and go through this process in a commercial setting every time. Thus some kind of meta-Q-Learning is probably required, selecting different hyperparameters and optimisation functions according to heuristics.

Training multiple robots in simulation in parallel with some kind of genetic algorithm, sharing best outcomes across all bots would also speed up learning, making it industrially performant, also limiting the potentially chaotic behaviour of LTSM cells.

Using a tool such as Tensorboard to help visualise the network would also help to debug and optimise the network.

Also, in real life, some actions by the robot would simply be off limits, not possible. Hitting the ground or surrounding machinery, walls et cetera would be extremely costly. These would need to be hard coded in, perhaps designed into the hardware itself also, using a high integrity safety system.

7 Conclusion

Both tasks were completed to the desired level of accuracy and valuable experience was gained in learning the benefits and drawbacks of Deep Q Learning.

In the near future it would be good to find a real world use case for DQN and use it with the new Jetson Xavier Development kit and the Isaac development stack, with parallel DQL via Isaac cloud.

In a real commercial or public setting, it is not safe enough to allow the DQN to assume full control, and safety instrumented functions and intrinsic hardware limitations would be required.