Deep RL Arm Manipulation Project

Taariq Hassan

Introduction

The goal of the Deep RL project was to create a Deep Q-Learning Network (DQN) to teach a robotic arm to achieve a number of objectives. The project was based on the Robotic Arm simulated environment from the Nvidia Jetson TX2 Deep Reinforcement Learning repository found here: https://github.com/dusty-nv/jetson-reinforcement

For this project, a DQN agent was created, and reward functions defined, in order to teach a simulated robotic arm to carry out the following 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 an 80% accuracy.

The robot arm was simulated in Gazebo.

Reward Functions

The rewards issued for different conditions will be discussed in further detail below.

For Task 1, if there was a collision between the robot arm and the target object, a reward of **2000** was issued. For Task 2, the reward of **2000** was issued when only the gripper of the robot touched the object and a reward of **-2000** if any other part of the robot touched the object. In both cases, the episode is ended since the goal state has been reached.

If the robot's gripper collides with the ground, a reward was issued based on how close the collision was to the target. The function used was simply $REWARD\_LOSS * distGoal$ where $REWARD\_LOSS = -2000$ and $distGoal$ represents the distance between the gripper and the target. It can be seen that if the gripper hits the ground close to the object, a small negative reward will be issued, and if the collision is further away, a large negative reward will be issued. This encourages the robot to move closer to the target, but the reward remains negative which means the robot won't settle for only getting close. This reward function was used for both tasks.

An interim reward was also issued while the robot was in motion. The reward function used was a smoothed moving average of the distance to the goal, and was the recommended reward function to use.

$$avgGoalDelta = avgGoalDelta * REWARD\_ALPHA + distDelta * (1 - REWARD\_ALPHA)$$

The $REWARD\_ALPHA$ parameter set the weight that new readings had on the result and was set to **0.2**. This reward function was used for both tasks.

Finally, a reward of **-2000** was issued if the episode reached its maximum length, which in this case was set to 100 frames. This reward function was used for both tasks.

Joint Control

For this project, two methods for controlling the robot's joints were explored, namely Velocity Control and Position Control. As their names suggest, velocity control control's the velocity of the robot arm's joints while position control control's the joint positions.

Initially, velocity control was used, but after changing to position control, the performance of the robot arm increased considerably. Therefore, position control was used for both tasks.

Hyperparameters

The hyperparameters used were the same for both tasks. The choice of hyperparameters are discussed in further detail below.

Firstly, the size of the input image, $INPUT\_WIDTH$ and $INPUT\_HEIGHT$ were both reduced from **512** to **64** so as to reduce memory usage and increase speed.

The Optimizer was first set to **RMSprop** with the intention of changing to a different optimizer if necessary however, the objectives were completed successfully with **RMSprop**. The $LEARNING\_RATE$ was initially set to **0.01** however, it was found that the arm took long to converge on a solution. Increasing it to **0.1** resulted in much faster learning.

$REPLAY\_MEMORY$ was left at **10000** however, USE_LSTM was set to **true** in order to enable the use of Long-Short-Term Memory. Through trial and error, $BATCH\_SIZE$ and $LSTM\_SIZE$ were set to **64** and **256** respectively. Both were increased from their original values of **8** and **32** since no memory issues were encountered.

As mentioned previously, $REWARD\_ALPHA$ was set to **0.2**. This was increased from the initial value of **0.1** to increase the smoothing of the averaged distance to the target.

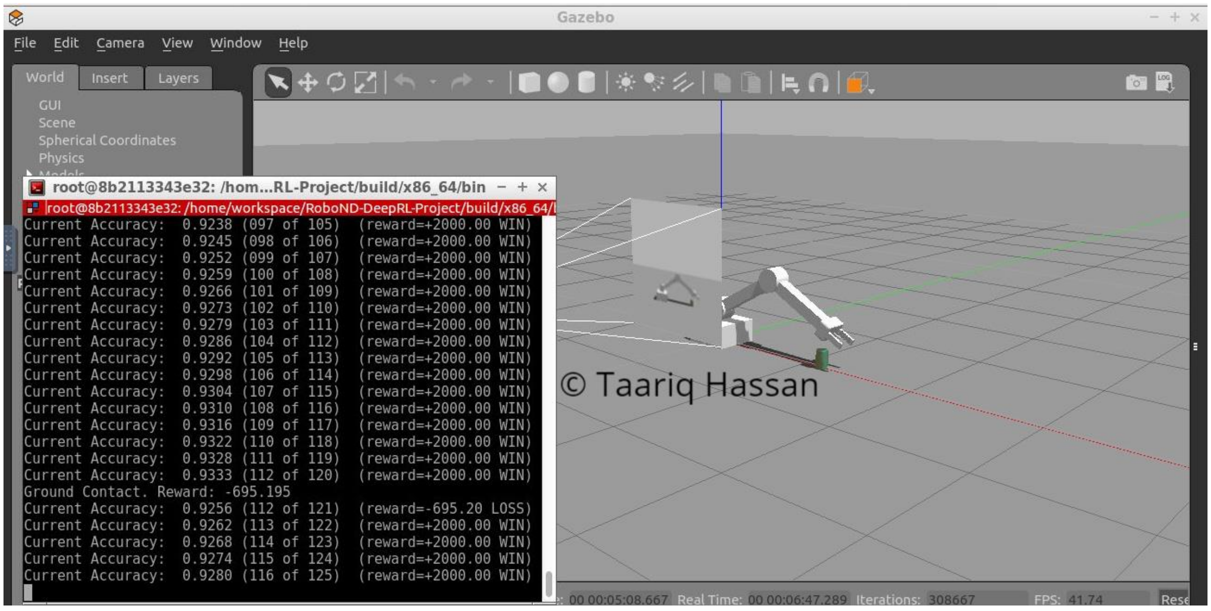
Finally, EPS_END was decreased from **0.05** to **0.01** to decrease the probability of the robot performing a random action once the optimal action has been learned.

Results

Task 1

For Task 1, any part of the robot needed to touch the target with an accuracy of 90%. Over a number of attempts, it was found that the robot would learn one of two solutions, either the configuration shown in the image below with the elbow at an upward angle, or with the elbow down, touching the ground. While both of these solutions are valid, it was observed that with the latter, upon impact with the target the robot gripper would detach from the arm. The cause of this behaviour was not investigated however it was still seen as a successful attempt.

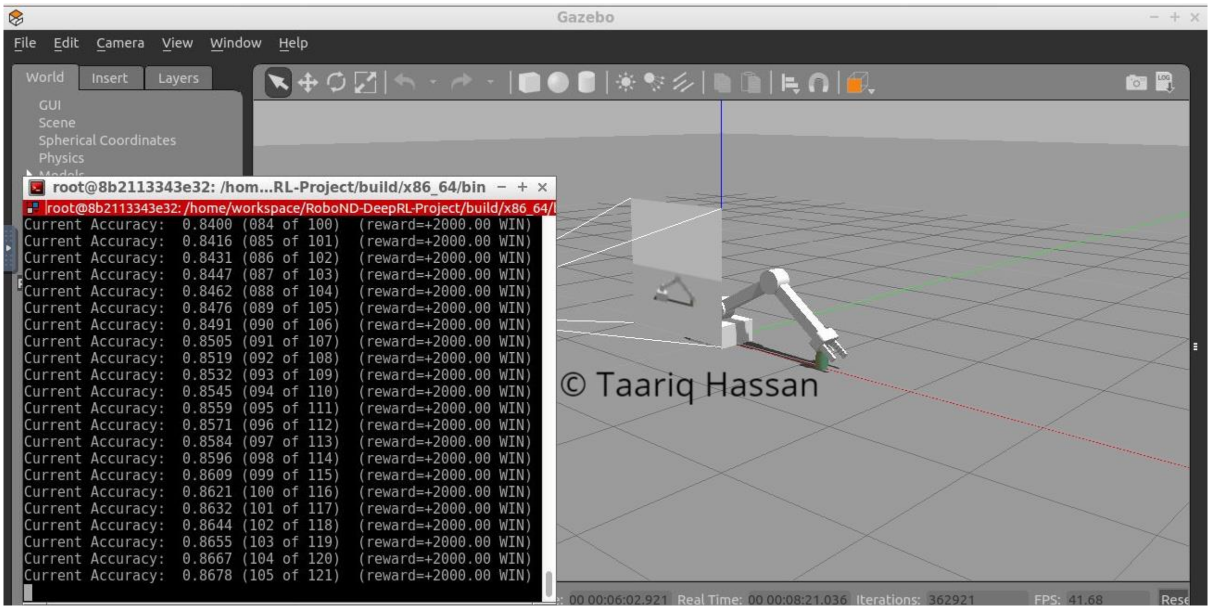
With the final set of parameters, task 1 was successfully completed, with the robot learning the correct solution in less than 100 episodes. The accuracy after 100 episodes was approximately **92%** and still climbing, as can be seen in the figure below.


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
root@8b2113343e32: /hom...RL-Project/build/x86_64/bin
root@8b2113343e32: /home/workspace/RoboND-DeepRL-Project/build/x86_64/
© Taariq Hassan
00 00:05:08.667 Real Time: 00 00:06:47.289 Iterations: 308667 FPS: 41.74

```
Current Accuracy:  0.9238 (097 of 105)  (reward=+2000.00 WIN)
Current Accuracy:  0.9245 (098 of 106)  (reward=+2000.00 WIN)
Current Accuracy:  0.9252 (099 of 107)  (reward=+2000.00 WIN)
Current Accuracy:  0.9259 (100 of 108)  (reward=+2000.00 WIN)
Current Accuracy:  0.9266 (101 of 109)  (reward=+2000.00 WIN)
Current Accuracy:  0.9273 (102 of 110)  (reward=+2000.00 WIN)
Current Accuracy:  0.9279 (103 of 111)  (reward=+2000.00 WIN)
Current Accuracy:  0.9286 (104 of 112)  (reward=+2000.00 WIN)
Current Accuracy:  0.9292 (105 of 113)  (reward=+2000.00 WIN)
Current Accuracy:  0.9298 (106 of 114)  (reward=+2000.00 WIN)
Current Accuracy:  0.9304 (107 of 115)  (reward=+2000.00 WIN)
Current Accuracy:  0.9310 (108 of 116)  (reward=+2000.00 WIN)
Current Accuracy:  0.9316 (109 of 117)  (reward=+2000.00 WIN)
Current Accuracy:  0.9322 (110 of 118)  (reward=+2000.00 WIN)
Current Accuracy:  0.9328 (111 of 119)  (reward=+2000.00 WIN)
Current Accuracy:  0.9333 (112 of 120)  (reward=+2000.00 WIN)
Ground Contact. Reward: -695.195
Current Accuracy:  0.9256 (112 of 121)  (reward=-695.20 LOSS)
Current Accuracy:  0.9262 (113 of 122)  (reward=+2000.00 WIN)
Current Accuracy:  0.9268 (114 of 123)  (reward=+2000.00 WIN)
Current Accuracy:  0.9274 (115 of 124)  (reward=+2000.00 WIN)
Current Accuracy:  0.9280 (116 of 125)  (reward=+2000.00 WIN)
```

Task 2

Task 2 required only the robot's gripper base to touch the target with an accuracy of at least 80%. This was a harder task for the robot to learn but it was still able to meet the objective in under 100 episodes. After 100 episodes, the accuracy was **84%** and climbing, as can be seen in the figure below. A similar joint configuration as Task 1 was learnt, with the elbow at an upward angle.


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
root@8b2113343e32: /hom...RL-Project/build/x86_64/bin
root@8b2113343e32: /home/workspace/RoboND-DeepRL-Project/build/x86_64/
© Taariq Hassan
00 00:06:02.921 Real Time: 00 00:08:21.036 Iterations: 362921 FPS: 41.68

```
Current Accuracy:  0.8400 (084 of 100)  (reward=+2000.00 WIN)
Current Accuracy:  0.8416 (085 of 101)  (reward=+2000.00 WIN)
Current Accuracy:  0.8431 (086 of 102)  (reward=+2000.00 WIN)
Current Accuracy:  0.8447 (087 of 103)  (reward=+2000.00 WIN)
Current Accuracy:  0.8462 (088 of 104)  (reward=+2000.00 WIN)
Current Accuracy:  0.8476 (089 of 105)  (reward=+2000.00 WIN)
Current Accuracy:  0.8491 (090 of 106)  (reward=+2000.00 WIN)
Current Accuracy:  0.8505 (091 of 107)  (reward=+2000.00 WIN)
Current Accuracy:  0.8519 (092 of 108)  (reward=+2000.00 WIN)
Current Accuracy:  0.8532 (093 of 109)  (reward=+2000.00 WIN)
Current Accuracy:  0.8545 (094 of 110)  (reward=+2000.00 WIN)
Current Accuracy:  0.8559 (095 of 111)  (reward=+2000.00 WIN)
Current Accuracy:  0.8571 (096 of 112)  (reward=+2000.00 WIN)
Current Accuracy:  0.8584 (097 of 113)  (reward=+2000.00 WIN)
Current Accuracy:  0.8596 (098 of 114)  (reward=+2000.00 WIN)
Current Accuracy:  0.8609 (099 of 115)  (reward=+2000.00 WIN)
Current Accuracy:  0.8621 (100 of 116)  (reward=+2000.00 WIN)
Current Accuracy:  0.8632 (101 of 117)  (reward=+2000.00 WIN)
Current Accuracy:  0.8644 (102 of 118)  (reward=+2000.00 WIN)
Current Accuracy:  0.8655 (103 of 119)  (reward=+2000.00 WIN)
Current Accuracy:  0.8667 (104 of 120)  (reward=+2000.00 WIN)
Current Accuracy:  0.8678 (105 of 121)  (reward=+2000.00 WIN)
```

It should be noted that the above results were obtained after many attempts. In some cases, the robot would take far more than 100 episodes to meet the objective. In most cases the simulation was stopped if the objective was not reached within 300 episodes. This was done in order to save on GPU hours in the provided Udacity workspace since it would take much more episodes to reach the required accuracy.

Future Work

Seeing as though the project is based on a tutorial for the Jetson TX2, the next step would be to run the simulation and training on the TX2 itself.

Further improvements could also be made to the reward functions. One possible addition would be to incorporate the frame count into the reward, encouraging the robot to learn to touch the target as quickly as possible. It was observed during testing that in some cases the arm would remain stationary or make very small adjustments forwards then backwards and only start moving towards the target just before the episode timed out. This result was issued the same reward as if the robot had moved for the target immediately. By reducing the reward with time, solutions which reached the target earlier would be favoured.

Some more time could also be spent further tuning the hyperparameters. One possible avenue to explore would be to decrease `EPS_DECAY`, the rate at which epsilon is reduced. As discussed previously, in some cases, the robot would take significantly more episodes to attain a reasonable accuracy. This may be due to the agent being too exploratory. By adjusting the `EPS_DECAY` parameter, the agent could adopt a more exploitatory approach, in the hopes of a solution being found more consistently. The effects of this remains to be explored.