Deep RL Arm Manipulation Project – Udacity: Deep Reinforcement Learning

Carl Sosa-Rivera

1. Introduction

This project aims to create a Deep Q-Learning Network (DQN) and, with it, train a robotic arm to meet certain objectives. The Robotic Arm used for this project is simulated on Gazebo and run and trains (learns) on the Nvidia Jetson TX2 Deep Reinforcement Learning platform.

This project mainly leverages an existing DQN that gets instantiated with specific parameters to run the Robotic Arm. For Deep Reinforcement Learning to occur, reward functions and hyperparameters must be defined. To test the capabilities of DQN, two objectives were established:

a. Have any part of the robot arm touch the object of interest, with at least a 90% accuracy for a minimum of 100 runs.
b. Have only the gripper base of the robot arm touch the object, with at least an 80% accuracy for a minimum of 100 runs.

2. Reward Functions:

To achieve both objectives, rewards functions were configured slightly different as you'll see in the following.

2.1. Objective 1

To meet objective 1, which is to have the robotic arm collide with the target object at any point, the approach taken was to use "velocity control" instead of position. To control velocity, a simple check was performed to observe whether the action was even or odd. Depending on the action check, actionVelDelta was either added or subtracted.

The robotic arm must never collide with the ground, therefore, a REWARD_LOSS of 1 * 20 will be issued upon ground collision. However, if the arm collided with the target object, it would issue a REWARD_WIN of 1 * 10, meeting the objective and resulting in a "win" for that run. Another criteria was added, to encourage the arm to accomplish its objective before the episode was over (frame span). If the run exceeded the maximum number of frames per episode (maxEpisodeLength), which was set to 100, a REWARD_LOSS of 1 * 1000 would be issued. This would result in a "loss" for that run.

Since it's using velocity control to complete the first objective, the reward function setup for the process in between to control the robotic arm's movement will defer to the one that will be used for the second objective. As part of the generic solution, the average delta (avgGoalDelta) was calculated just as stated in the tasks lesson:

```
avgGoalDelta  = (avgGoalDelta * alpha) + (distDelta * (1.0f - alpha));
```

However, to hone in on velocity control, the reward was issued as follows:

```
if (avgGoalDelta > 0)
{
        if(distGoal > 0.001f){
                rewardHistory = REWARD_WIN / distGoal;
        }
        else if (distGoal < 0.001f || distGoal == 0.0f){
                rewardHistory = REWARD_WIN / 2000.0f;
        }
}
```

This approach encourages the arm to quickly reach the goal by increasing the reward (inversely proportional) as it gets close to the goal. However, once this arm gets to a certain point (close enough), the reward stops increasing and only issues a small reward, which should encourage the arm to slow down as it reaches the target.

2.2. Objective 2

To meet objective 2, which is to have the robotic arm collide with the target object only with the gripper, velocity control was initially tested. After many attempts, it was decided to use "position control" instead of velocity. Since touching the target solely with the gripper, more finesse and joint control seemed to require more emphasis. To control position, a simple check was performed to observe whether the action was even or odd. Depending on the action check, actionJointDelta was either added or subtracted.

The robotic arm must never collide with the ground, therefore, a REWARD_LOSS of 1 * 20 will be issued upon ground collision. However, if the gripper collided with the target object, it would issue a REWARD_WIN of 1 * 100, meeting the objective and resulting in a "win" for that run. For this objective, the emphasis was to get it to touch with the gripper (solely), which is why the reward is higher than for the previous objective. Also, another limitation was added, to encourage the arm to accomplish its objective before the episode was over (frame span). If the run exceeded the maximum number of frames per episode (maxEpisodeLength), which was set to 100, a REWARD_LOSS of 1 would be issued. This would result in a "loss" for that run. Although the objective is still to collide with the target within one run/episode, some emphasis was taken away from this limitation given that velocity control has been replaced by position control.

However, since position control will be used to achieve objective 2, the reward function setup for the process will be different this time around. As part of the generic solution, the average delta (avgGoalDelta) was calculated just as stated in the tasks lesson (the same as for objective 1):

```
avgGoalDelta  = (avgGoalDelta * alpha) + (distDelta * (1.0f - alpha));
```

However, this time around instead of using the distance from the goal (distGoal) to drive the rewards, the smoothed moving average will be used, as follows:

```
if (avgGoalDelta > 0)
{
        if(distGoal > 0.0f){
                rewardHistory = REWARD_WIN * avgGoalDelta;
        }
        else if (distGoal == 0.0f){
                rewardHistory = REWARD_WIN * 10.0f;
        }
}
else {
        rewardHistory = REWARD_LOSS * distGoal;
}
```

This approach focuses on the movement, encouraging the robotic arm to move more. The more it moves towards the goal, the more rewards it will get (proportional to avgGoalDelta). However, once the arm at 0 distance away from the it would provide an extra reward. Furthermore, if not moving towards the goal, it would issue a REWARD_LOSS proportional to the distance to the goal (distGoal). At least that's the theory behind this approach.

3. Hyperparameters

For both objectives, tuning the hyperparameter was an iteration process. Nonetheless some thought was put into selecting them.

3.1. Objective 1

To achieve the first objective, to train the robotic arm to touch the target at any point, the following hyperparameters were used:

Hyperparameter	Value	Reason/Intuition/Comments
INPUT_WIDTH	64	Reduced width since didn't believe a higher one was required
INPUT_HEIGHT	64	Reduced height since didn't believe a higher one was required
OPTIMIZER	"RMSprop"	Started with Adam and SGD but looking at forums found that RMSprop also worked well. It did.
LEARNING_RATE	0.2f	Iterated from 0.5f to 0.2f
REPLAY_MEMORY	10000	Stayed the same as the lesson suggested
BATCH_SIZE	16	Thought it required deeper learning than 8 batches.
USE_LSTM	True	Improves learning from past experiences
LSTM_SIZE	256	Seemed like 32 was not working so iterated until 256.

3.2. Objective 2

To achieve the second objective, to train the robotic arm to touch the target with the gripper, the hyperparameters and DQN API settings were modified:

a. Hyperparameters

Hyperparameter	Value	Reason/Intuition/Comments
INPUT_WIDTH	64	Same used as for Objective 1
INPUT_HEIGHT	64	Same used as for Objective 1
OPTIMIZER	“RMSprop”	Same used as for Objective 1
LEARNING_RATE	0.1f	Requiring a more specific action, deeper learning was required for the second objective.
REPLAY_MEMORY	10000	Same used as for Objective 1
BATCH_SIZE	16	Deeper learning than for the first objective was required.
USE_LSTM	True	Same used as for Objective 1
LSTM_SIZE	256	Same used as for Objective 1

b. DQN API Settings

These settings were changed to decrease exploration and increase exploitation of a solution previously found. These settings decreased initial exploration probability, and decreased that probability quicker than for the first objective, to end in a lower exploration probability.

Parameter	Value
EPS_START	0.8f
EPS_END	0.01f
EPS_DECAY	300

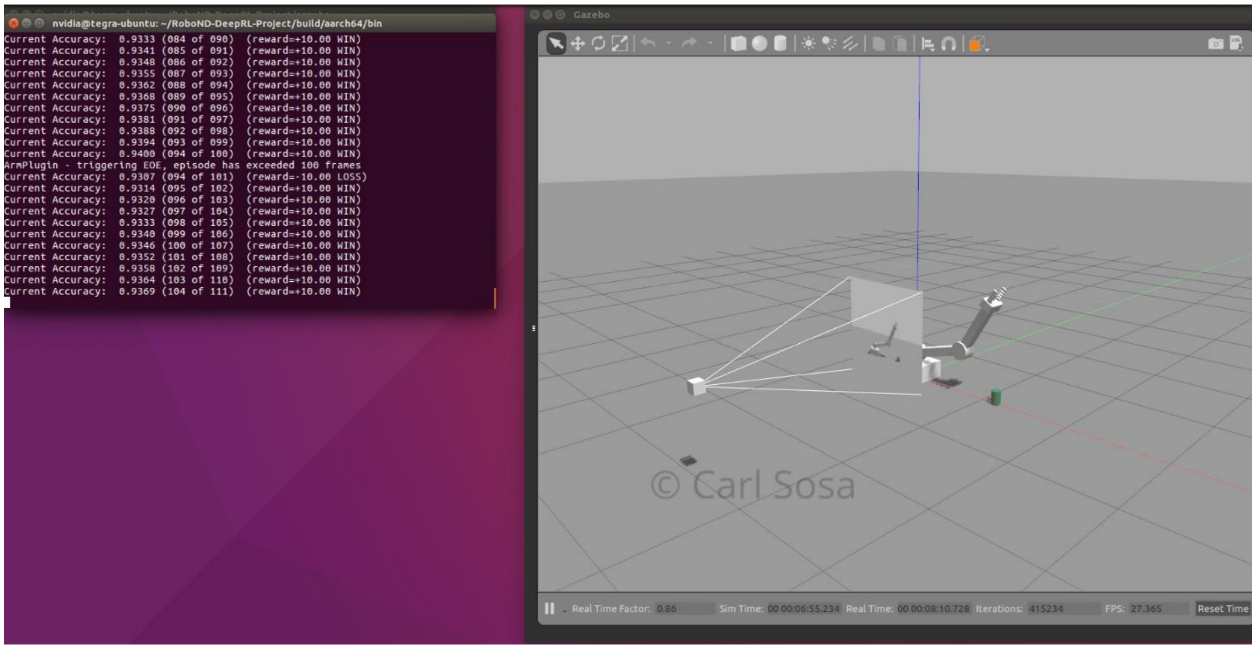
4. Results

As seen in (Fig. 1 & Fig 2), both objectives were achieved. Meeting percentage accuracy for both scenarios right after 100 runs were executed.

4.1. Objective 1

Objective 1, was relatively simple to train, the robotic arm intuitively started learning better as hyperparameters were tuned and the reward values were increased. At almost each iteration of testing, at the beginning before learning the objective, the arm started colliding with the ground, seeming to break multiple times. Increasing the negative rewards for ground collisions didn’t dramatically improve this behavior. This could definitely be a problem for real-life solution. However, this behavior did not happen as much on the last iteration, and learned fairly quickly, which is why it became the final version for this objective.

It was noted that it kept trying to explore other ways to tackle the problem, obtaining multiple solution approaches along the way. Ultimately, the latest version of the code for objective 1 managed to get to 90% accuracy almost right after 100 runs.



```
nvidia@tegra-ubuntu: ~/RoboND-DeepRL-Project/build/aarch64/bin
Current Accuracy:  0.9333 (084 of 090)  (reward=+10.00 WIN)
Current Accuracy:  0.9341 (085 of 091)  (reward=+10.00 WIN)
Current Accuracy:  0.9348 (086 of 092)  (reward=+10.00 WIN)
Current Accuracy:  0.9355 (087 of 093)  (reward=+10.00 WIN)
Current Accuracy:  0.9362 (088 of 094)  (reward=+10.00 WIN)
Current Accuracy:  0.9368 (089 of 095)  (reward=+10.00 WIN)
Current Accuracy:  0.9375 (090 of 096)  (reward=+10.00 WIN)
Current Accuracy:  0.9381 (091 of 097)  (reward=+10.00 WIN)
Current Accuracy:  0.9388 (092 of 098)  (reward=+10.00 WIN)
Current Accuracy:  0.9394 (093 of 099)  (reward=+10.00 WIN)
Current Accuracy:  0.9400 (094 of 100)  (reward=+10.00 WIN)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.9307 (094 of 101)  (reward=-10.00 LOSS)
Current Accuracy:  0.9314 (095 of 102)  (reward=+10.00 WIN)
Current Accuracy:  0.9320 (096 of 103)  (reward=+10.00 WIN)
Current Accuracy:  0.9327 (097 of 104)  (reward=+10.00 WIN)
Current Accuracy:  0.9333 (098 of 105)  (reward=+10.00 WIN)
Current Accuracy:  0.9340 (099 of 106)  (reward=+10.00 WIN)
Current Accuracy:  0.9346 (100 of 107)  (reward=+10.00 WIN)
Current Accuracy:  0.9352 (101 of 108)  (reward=+10.00 WIN)
Current Accuracy:  0.9358 (102 of 109)  (reward=+10.00 WIN)
Current Accuracy:  0.9364 (103 of 110)  (reward=+10.00 WIN)
Current Accuracy:  0.9369 (104 of 111)  (reward=+10.00 WIN)
```

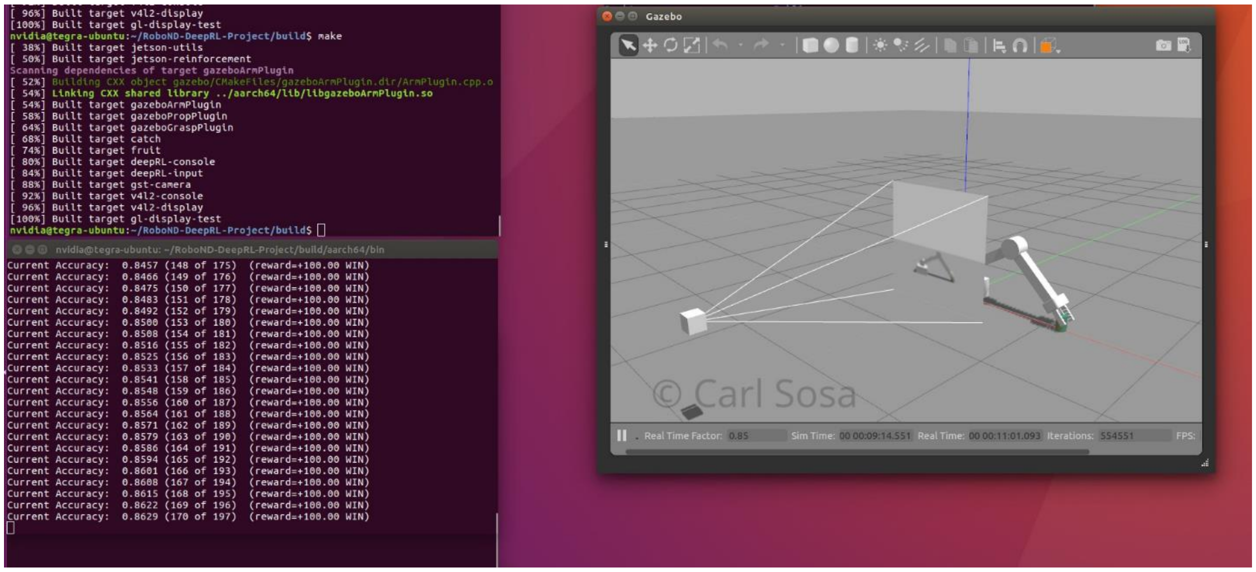
Figure 1: Arm collision with target.

4.2. Objective 2

Normally, for objective 2, required more accuracy in its behavior. This was dramatically harder to train than objective 1. After iterating and testing many times, by tuning hyperparameters, it was noticed that the agent was trying to explore too much. However, there are so many approaches that can be taken to touch the target with just one point. For that reason, the DQN API Settings were changed to decrease exploration slightly compared to objective 1. This helped on teaching the robotic arm agent to perform a solution approach quicker. Also, the fact that the learning rate was decreased and the batch size increased, seemed to have helped the agent to train and learn deeper than objective 1.

However, as with objective 1, the agent was also displayed hitting the ground hard enough to sometimes break, at least in the initial stages of the iteration. This a problem that could be catastrophic for a real robot (which normally costs a lot of money to make). This should be explored further.

Nonetheless, for the purpose of this project, the robotic arm exceeded the minimum accuracy of 80% reaching up to 86% accuracy after almost 200 runs.



```
[ 96%] Built target v4l2-display
[100%] Built target gl-display-test
nvidia@tegra-ubuntu:~/RoboND-DeepRL-Project/build$ make
[ 38%] Built target jetson-utils
[ 50%] Built target jetson-reinforcement
Scanning dependencies of target gazeboArmPlugin
[ 52%] Building CXX object gazebo/CMakeFiles/gazeboArmPlugin.dir/ArmPlugin.cpp.o
[ 54%] Linking CXX shared library ../aarch64/lib/libgazeboArmPlugin.so
[ 54%] Built target gazeboArmPlugin
[ 58%] Built target gazeboPropPlugin
[ 64%] Built target gazeboGraspPlugin
[ 68%] Built target catch
[ 74%] Built target fruit
[ 80%] Built target deepRL-console
[ 84%] Built target deepRL-input
[ 88%] Built target gst-camera
[ 92%] Built target v4l2-console
[ 96%] Built target v4l2-display
[100%] Built target gl-display-test
nvidia@tegra-ubuntu:~/RoboND-DeepRL-Project/build$
```

```
nvidia@tegra-ubuntu: ~/RoboND-DeepRL-Project/build/aarch64/bin
Current Accuracy:  0.8457 (148 of 175)  (reward=+100.00 WIN)
Current Accuracy:  0.8466 (149 of 176)  (reward=+100.00 WIN)
Current Accuracy:  0.8475 (150 of 177)  (reward=+100.00 WIN)
Current Accuracy:  0.8483 (151 of 178)  (reward=+100.00 WIN)
Current Accuracy:  0.8492 (152 of 179)  (reward=+100.00 WIN)
Current Accuracy:  0.8500 (153 of 180)  (reward=+100.00 WIN)
Current Accuracy:  0.8508 (154 of 181)  (reward=+100.00 WIN)
Current Accuracy:  0.8516 (155 of 182)  (reward=+100.00 WIN)
Current Accuracy:  0.8525 (156 of 183)  (reward=+100.00 WIN)
Current Accuracy:  0.8533 (157 of 184)  (reward=+100.00 WIN)
Current Accuracy:  0.8541 (158 of 185)  (reward=+100.00 WIN)
Current Accuracy:  0.8548 (159 of 186)  (reward=+100.00 WIN)
Current Accuracy:  0.8556 (160 of 187)  (reward=+100.00 WIN)
Current Accuracy:  0.8564 (161 of 188)  (reward=+100.00 WIN)
Current Accuracy:  0.8571 (162 of 189)  (reward=+100.00 WIN)
Current Accuracy:  0.8579 (163 of 190)  (reward=+100.00 WIN)
Current Accuracy:  0.8586 (164 of 191)  (reward=+100.00 WIN)
Current Accuracy:  0.8594 (165 of 192)  (reward=+100.00 WIN)
Current Accuracy:  0.8601 (166 of 193)  (reward=+100.00 WIN)
Current Accuracy:  0.8608 (167 of 194)  (reward=+100.00 WIN)
Current Accuracy:  0.8615 (168 of 195)  (reward=+100.00 WIN)
Current Accuracy:  0.8622 (169 of 196)  (reward=+100.00 WIN)
Current Accuracy:  0.8629 (170 of 197)  (reward=+100.00 WIN)
```

Figure 2: Gripper collision with target.

As expected, the second objective was harder to reach, as training the robotic arm to have finesse and be more precise, required finer tuning of hyperparameters and, as mentioned previously, DQN API settings as well.

Nonetheless, both agents were optimally trained to meet the objectives presented, with greater than expected accuracy for the second objective once it was finally tuned properly.

5. Future Work

For future work, would like to improve on the hyperparameter tuning for the first objective to achieve almost 100% accuracy results. Obtaining a higher accuracy and faster might be another improvement to investigate, as in real-life, having to wait long periods for robots to learn how to perform tasks might be slightly more inconvenient than in a simulation. Furthermore, cannot emphasize enough the ground collision issue that must be mitigated for this to be a useful real-life solution. It must never collide with the ground, or at least not with that amount of force. Perhaps a constraint can be developed and implemented to shut down the arm right before colliding with the ground.