Deep RL Arm Manipulation

Lorenzo Bermillo

I. Rewards

Training an agent is no different from training a child. Most parents use a reward system when teaching a child. The child gets rewarded for doing something right and gets punished if they did something wrong. This reward system can also be used to train the DQN agent for this project. To do this, the agent's rewards must first be defined.

There are two tasks in this project:

1. Have any part of the robot arm touch the object in front of it while preventing the gripper from touching the ground.
2. Have ONLY the gripper of the robot arm touch the object in front of it while preventing the gripper from touching the ground.

For both tasks, the agent is penalized for hitting the ground by some constant value, this value had to be the lowest value that the agent can earn to discourage the agent from repeatedly performing this action. The value chosen for this was -100. The robot is also penalized after 100 frames to discourage it from indecision. When part of the robot arm touches the object, the reward system differs for each task. For the first task, because the task is binary (touch the object and get rewarded or touch the ground and get punished), the reward was just the positive value of the constant used to penalize the robot for touching the ground, in this case the value used was +100. Because the second task isn't binary like the first, a few things were considered. The overall goal was to touch the object with the gripper, which is more of a difficult task than just touching the object. For this, the agent is encouraged by rewarding it with a very high reward value, in this case +1000, for accomplishing the goal. However, the robot arm can't avoid touching the object with another part other than its gripper. One may want to discourage the agent from touching the object with parts of the arm that is not the gripper and issue the loss reward to it, but the agent shouldn't be discouraged from touching the object for this is part of the goal. So instead of punishing the agent completely, the agent is only penalized by a smaller amount than the loss reward. In the project, the penalty for the robot arm touching the object other than its gripper was 50% of the loss reward.

So far, the only rewards discussed are the terminal rewards, meaning once the conditions have been met, the agent is given the reward and the episode terminates and starts over. But how does the agent know what to do during the episode? Another reward needs to be defined so that the it will encourage the agent to accomplish its task. This short-term reward is responsible for guiding the agent to its goal, so for this project, a reward based on the distance of the gripper to the object is a great start. The reward function chosen for this is the Smoothed Moving Average, or SMMA, of the delta of the distance to the goal. The function is defined as follows:

```
average_delta = (average_delta * alpha) + (dist * (1 - alpha));
```

By using SMMA, it shapes the reward function and gives the agent gradual feedback to let it know that it's doing better as it gets closer to the object of interest. This function helps eliminate fluctuations and allows for agent to follow prevailing trend to maximize its rewards, allowing the agent to learn faster. This is achieved by setting *alpha* closer to 0 (*alpha* is a constant between 0 and 1), doing so adds more weight to the recent delta as the average of the deltas are computed. Thus, the older computed deltas are not removed completely, but only have a minimal impact on the moving average, resulting to a smoother feedback.

Lastly, because the rewards are based on distance and position of the gripper rather than velocity, the joint control used for this project is position control.

II. Hyperparameters

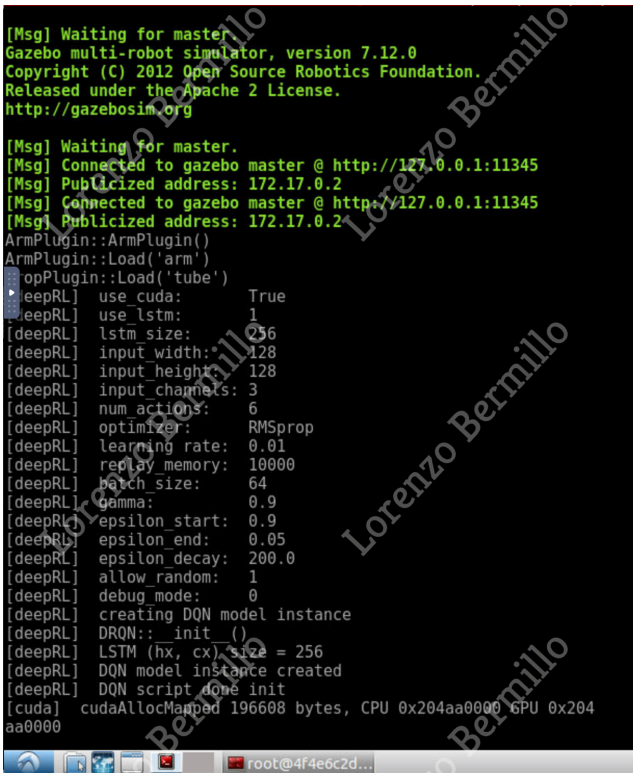

Figure 1: Objective 1 parameters

The first thing that came to mind when tuning the parameters was to condense the input size to reduce the complexity, so the input size was reduced from the default 512x512 to smaller square image of 128x128. Keeping the square image also reduces the difficulty of matrix operations thus optimizing it for the GPU. To further optimize it and speed up the operations, the batch size was also reduced to 64.

Because RMSprop was the only optimizer where the agent accomplished the goal for both tasks after testing other two

optimizers (SGD and Adam), this was elected to be the optimizer of choice. Although the other two optimizers may have worked with proper parameters. Next was setting the learning rate. Setting up the learning rate was an iterative process of decreasing an initial rate by small increments until convergence. Starting from 0.1, the learning rate was decreased by 0.02. The model started converging at 0.01, and so this was the learning rate chosen for task 1.

Finally, since the network is taking inputs of a sequence of images, LSTM is set to true to keep track of both long and short-term memory. Using LSTMs as part of that network, can train the network better by taking into consideration several past frames from the camera instead of a single frame. Because the data set is not very large, the size of the LSTM was set to 256.

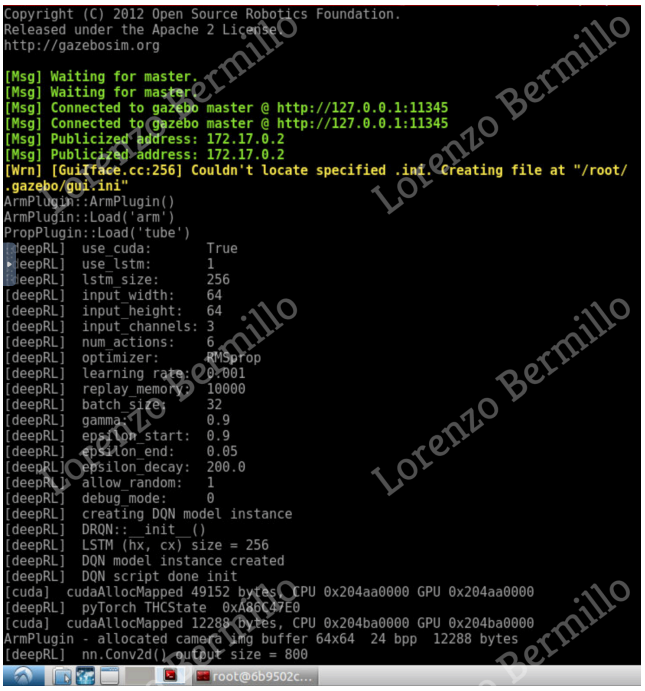

Figure 2: Objective 2 parameters

There weren't many changes made to the parameters for the second task. To speed up GPU operations, the input size was further reduced to 64x64 and the batch size to 32. The learning rate was also reduced using the same method from the first objective and the final learning rate was 0.001.

III. Results

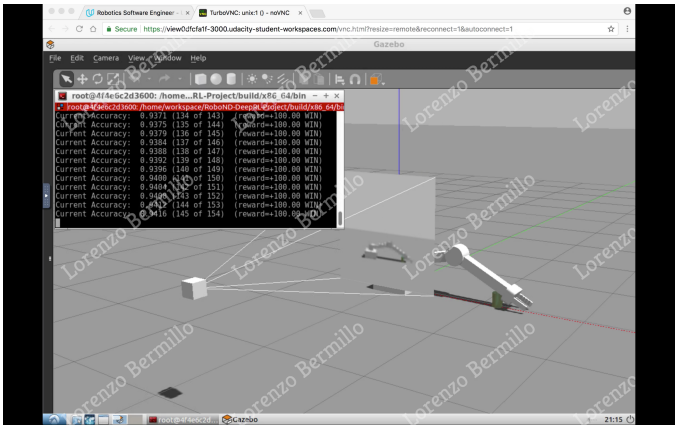

Figure 3: Robot arm touching the object for objective 1 after 154 episodes

The goal for both objectives were met after many episodes. In the first task, after 166 episodes, the agent yielded to 94.58% accuracy while for the second task, after 250 episodes, the agent was able to yield 88.4% accuracy.

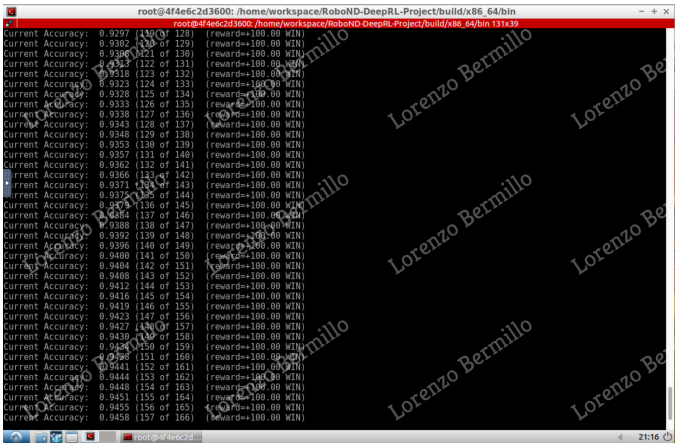

Figure 4: Objective 1 accuracy

It was no surprise that the first objective would yield to a better accuracy due to having a binary task. Overall, both agents performed well but only after the goal was found. However, in several of the iterations, especially for the second objective, if the agent doesn't learn about the goal soon enough, it doesn't seem to ever converge, and the simulation needs to start over. Another shortcoming for the second objective was the agent kept getting stuck in a local minimum, where the agent knows to touch the object but it's not touching it with the gripper. Nonetheless, once the agent learned about what it's supposed to do, it didn't take long to start winning consecutively.

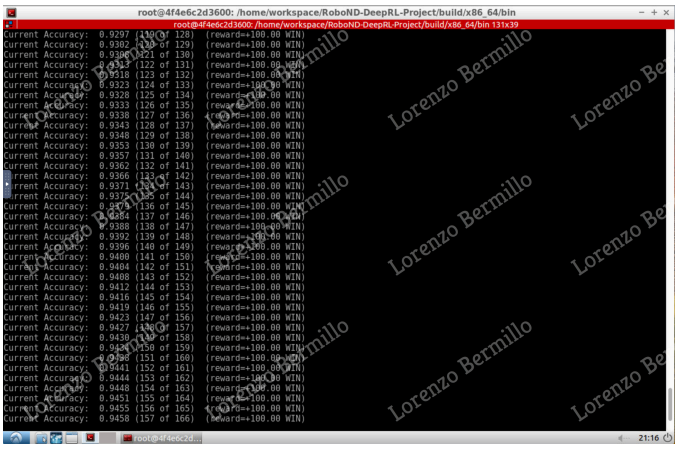

Figure 5: Objective 2 accuracy

It seems best that the agent learns about the goal as early as possible to converge quickly and not get stuck in a local minimum.

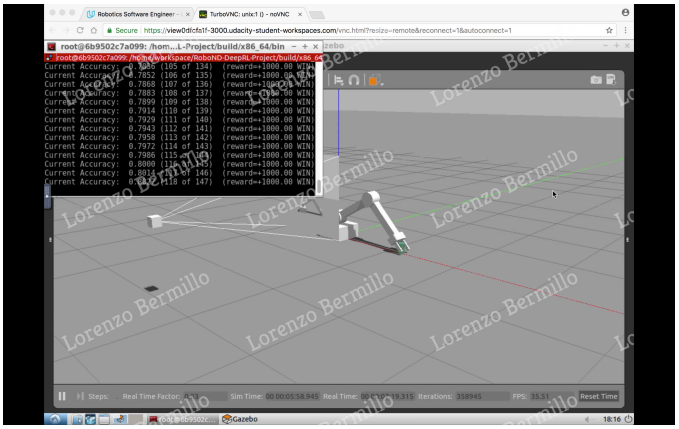

Figure 6: Gripper touching the object for objective 2 after 147 episodes

IV. Improvements

Although both objectives were achieved, there are some improvements the agent can use to learn better. For instance, a well-engineered reward system could have prevented the agent in the second objective from getting stuck in a local minimum, the Exponential Moving Average (EMA) could replace the SMMA as the interim reward. Another is using a different optimizer such as the Adam and tuning the agent to this optimizer to speed up the learning process greatly and converge faster. Or simply just perfecting the tune for the current optimizer may just do the trick.

V. Works Cited

Bonsai. *Deep Reinforcement Learning Models: Tips & Tricks for Writing Reward Functions*. n.d. 2018.

Brownlee, Jason. *CNN Long Short-Term Memory Networks*. 21 August 2017. 2018.

Udacity. *Deep RL*. n.d. 2018.