Udacity Project: Deep Reinforcement Learning for Robotic Arm Manipulation

Brad Huang

August 3, 2018

1 Introduction

Deep Reinforcement Learning (Deep RL) is a powerful tool that recently gained public interests due to the popular AlphaGo event, where the AlphaGo program used deep RL to learn how to play Go and eventually was able to beat the best Go players in the world. The core concept of deep RL is simple - given input controls and appropriate way to reward and penalize behaviors and outcome, a deep RL agent can learn to use neural networks to optimize its decision making process and complete tasks that it was not explicitly taught to do. In this project, I attempted to use deep RL to train a robotic arm to touch a target by videos of its actions and rewards of its states in Gazebo.

2 Deep Q Network Agent

2.1 Agent Setup

The robotic arm contains three degrees of freedom - one revolute joint from the base of the robot to the first link, another revolute joint from the first link to the second link, and one last revolute joint from the second link to the gripper. Since the robot can control each joint's degree of rotation from its respective reference frame, I devised the deep Q-network (DQN) agent to have six possible actions - increasing or decreasing each joint at a time. The DQN agent receives image sequences of its motion in the world, and also rewards based on its distance to the target cylinder and contacts with the ground or the target. The DQN agent then used these information to come up with an appropriate policy moving its arm and gripper towards the target through training episodes. I hope to achieve two tasks with the DQN agent - (1) any part of the robotic arm is in contact with the target cylinder, and (2) only the gripper of the robotic arm is in contact with the target cylinder. The agent should be able to consistently reach the first goal 90% of the episodes and reach the second goal 80% of the episodes.

2.2 Reward Functions

The reward scheme of the DQN agent contains three parts, interim rewards, contact rewards and end of episode (EOE) rewards. Interim rewards are temporary rewards for the agent before it contacts any other objects, while the collision rewards are assigned when it touches another object. Each episode of the learning agent ends immediately when there is a contact, but if the robot has not gotten into contact with anything after a certain amount of frames, the EOE reward would be assigned to end the episode.

2.2.1 Interim Reward

The interim rewards attempt to guide the robotic arm as close to the target as possible. At each frame, I would compute the distance between the gripper and the target, and the delta between the previous distance and the current distance. The delta distance is smoothed using a smoothing and moving average equation to reduce noises. When the delta distance is positive, the robotic arm has moved closer to the target, and vice versa. As a result, the interim rewards is simply assigned to be the smoothed delta amplified by a certain amount to be on the same order with other rewards.

2.2.2 Collision Reward

Three types of collisions can occur between the robot arm and the other objects. First, the robotic arm can collide with the ground, making it potentially unusable. Thus, the reward for hitting the ground is highly negative for the robot. Second, the robotic arm can collide with the target. In the first goal, it does not matter which part of the arm collides with the target, so a positive reward is given whenever the collision is detected. In the second goal, the robot would need to contact the target with its gripper, so a positive reward is only given when the collision is between the gripper links and the target, while a negative reward is given when the collision is between the other parts of the arm. To encourage the robot to move closer to the target, the penalty in the second part is not as heavy as the ground collision penalty.

2.2.3 End of Episode Reward

In the first task, the robot would be penalized if it made no contact within 100 frames. This allows ample time for it to explore and move around in the space. On the other hand, in the second task, the robot is only allowed 80 frames before it needs to make contact. The lower maximum frame number forces the robot arm to search for a good and simple solution, as opposed to falling back on complicated and unstable solutions. This also helped accelerating the training process during the second task, since the agent was often trapped in undesirable states until the EOE.

3 Hyperparameters

3.1 Goal 1

Few hyperparameter changes were required to complete the first task. The most import changes are:

- The `"Adam"` optimizer seemed to performed the best from trial and error compared to `"RMSprop"` and `"SGD"`. Learning rate of 0.005f was also selected from trial and error.
- I reduced the input image dimension to 128x128, which is sufficient for the application and reduces the complexity of the model.
- I enabled LSTM after verifying that the reward functions are working properly. Enabling LSTM sometimes made the robot's behavior more erratic, but the robot can often find a solution sooner with the LSTM.
- I increased the batch size to 128, making the computation time lower without significant drawbacks

3.2 Goal 2

To complete the second task, I tried many combinations of hyperparameters, and settled with the following changes from task 1:

- Instead of the `"Adam"` optimizer, the `"RMSprop"` optimizer seemed to performed the best. I was able to use a higher learning rate of 0.1f with the `"RMSprop"` optimizer.
- I increased the multiplication factor for the interim reward from 10.0f in task 1 to 100.0f in task 2. Higher interim rewards force the robot arm to move as close to the target as possible before any contacts.
- After verifying the reward function to be appropriate, I reduced the `EPS_DECAY` parameter from 200 to 100. This allows the learned solution to be performed more consistently and the agent would turn to exploitation sooner.
- I decreased the maximum number of episodes to reach EOE from 100 to 80. This forced the robot to learn simpler solutions, while also made each episode much shorter and the entire training process faster.
- I increased the LSTM size to 256, which did not seem to have immediate impact but did allow the robot to carry out complicated solutions at times.

4 Results

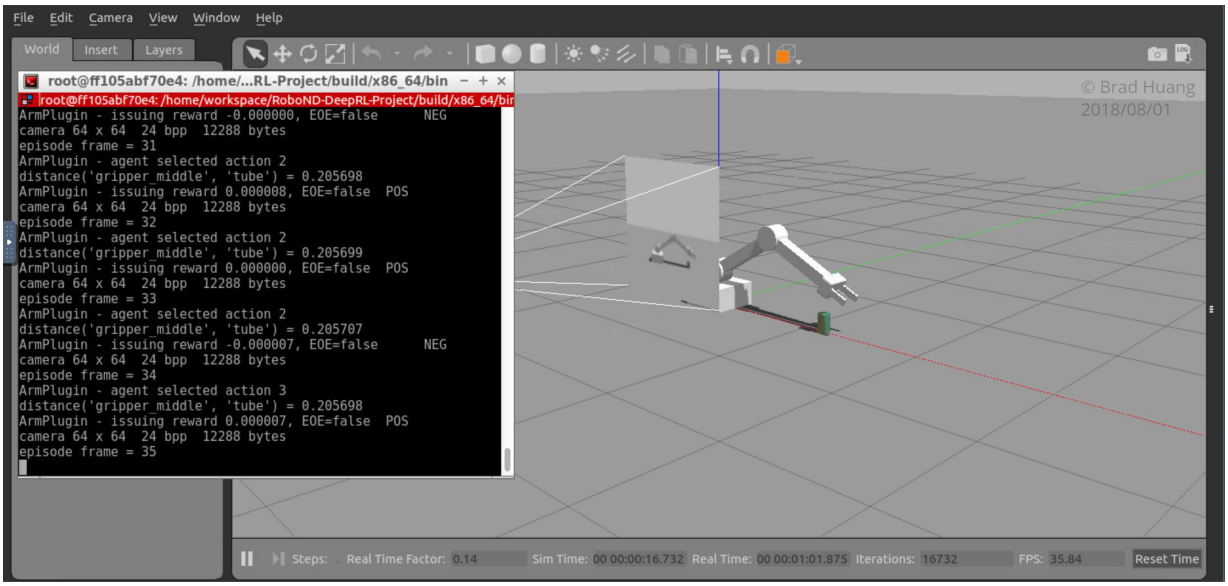

Figure 1. The robot settled in a position with small penalty but no positive rewards.

I had some troubles setting up the correct rewards functions and state updates at first. I attempted to use the distance to the goal in the interim rewards, but the results were worse than simply using the smoothed delta distances. In addition, I discovered that my robot stayed in a position until EOE a lot. This way, it would not have to suffer from the large penalty of hitting the ground, but also could not find the steps to reach the cylinder. I found that the problem had to do with my state updates, where I did not determine the odd- and evenness of the action correctly and were causing the joint positions to be unstable. The problem was fixed soon after I corrected the state update bug.

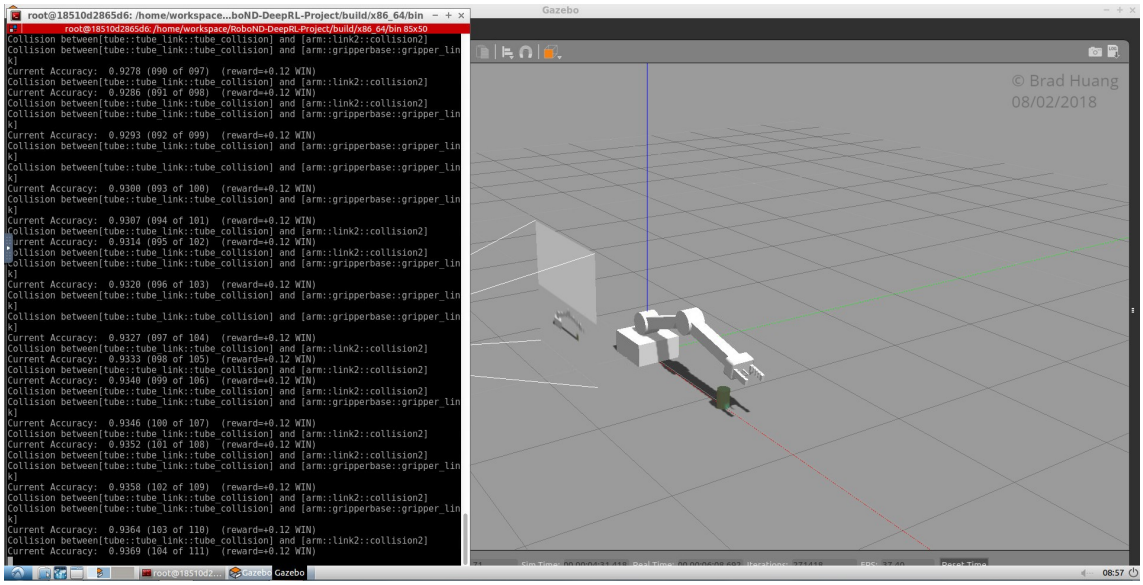

Figure 2. Any part of the robot collides with the target cylinder 93.7% of 111 episodes.

The robot was able to reach the first goal efficiently after the DQN agent was set up correctly. It failed the first few episodes while exploring the state space, and had many failed attempts bending backwards or hitting the ground. However, after it experienced a good solution by simply rotating the base joint and a little bit of the arm joint to smash down the cylinder, it was able to repeat the solution consecutively to reach the target 90% accuracy after 100 episodes.

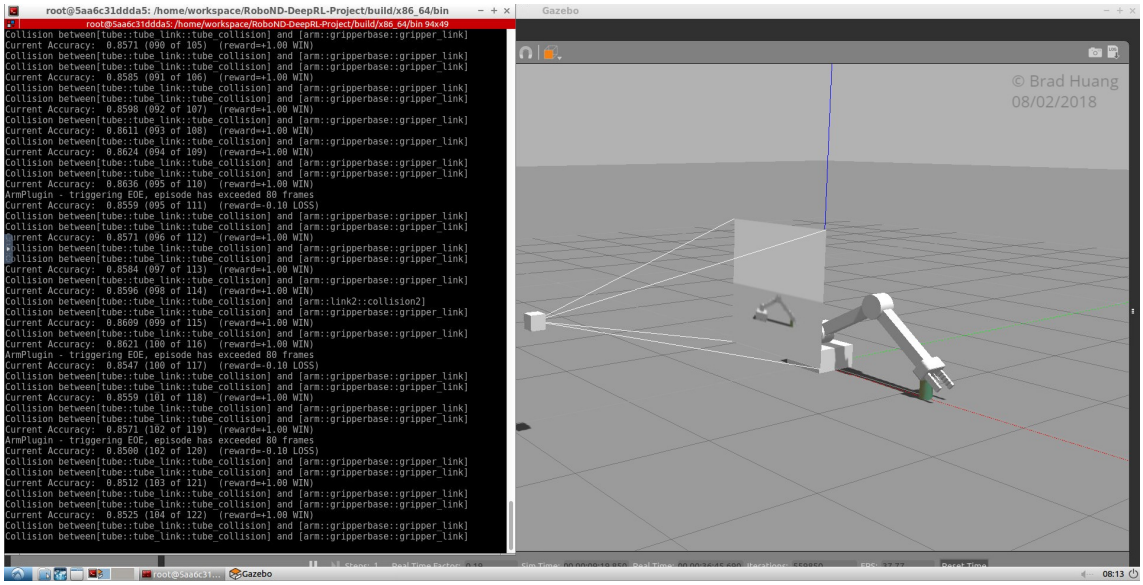

Figure 3. The gripper of the robot collides with the target cylinder 85.3% of 122 episodes.

The robot reached the first goal with little tuning efforts, while reaching the second goal required lots of hyperparameter tuning and trials. The figure above shows the most successful run I had after around 20 runs, learning to efficiently touch the cylinder with the gripper in 10 episodes.

5 Discussions

The main problem with the agent is its randomness. Every experiment run would learn the solution differently. It is much more troublesome in the second task. There are many more paths to complete the first task than the second task in state space, because only a few joint configurations can lead to the robot touching the cylinder with only its gripper. As a result, during the second task, Some runs took a few dozen episodes to start repeatedly performing the solution, while some runs took only a few episodes to do the same. I was able to speed up learning by reducing the EOE number to 80, and speed up exploitation by reducing `EPS_DECAY`. The combined effect of the two changes made the robot learn efficient solution as opposed to complicated and delicate solution that it may not be able to reproduce.

6 Future Work

The agent can definitely be improved in task 2 since the results were not always promising even with the final configuration I had. I tried devising more complicated reward functions, such as incorporating the distance between the gripper and the cylinder when the robot touched the target with other parts of the arm, but it was difficult to evaluate whether the changes were improving the agent or not. I believe the better approach next time would be reducing the variability of the learning process first, by disabling random steps and LSTM, to figure out the best reward functions, and then try to tune the other hyperparameters related to the variabilities.