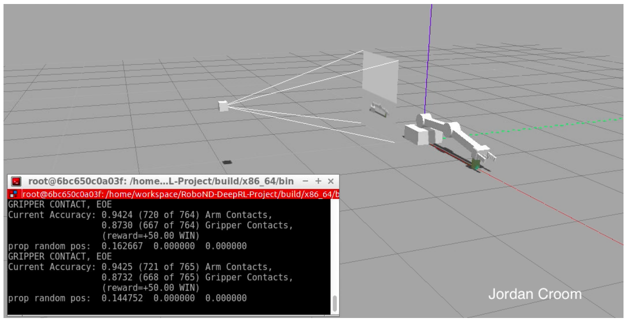
Deep Reinforcement Learning Project

Jordan Croom


root@6bc650c0a03f: /home...L-Project/build/x86_64/bin
root@6bc650c0a03f: /home/workspace/RoboND-DeepRL-Project/build/x86_64/b
Jordan Croom

```
GRIPPER CONTACT, EOE
Current Accuracy: 0.9424 (720 of 764) Arm Contacts,
                  0.8730 (667 of 764) Gripper Contacts,
                  (reward=+50.00 WIN)
prop random pos:  0.162667  0.000000  0.000000
GRIPPER CONTACT, EOE
Current Accuracy: 0.9425 (721 of 765) Arm Contacts,
                  0.8732 (668 of 765) Gripper Contacts,
                  (reward=+50.00 WIN)
prop random pos:  0.144752  0.000000  0.000000
```

Fig. 1: Agent in Test Environment

1 Introduction

THIS writeup summarizes my results for the Udacity Robotics Nanodegree deep reinforcement learning project. In this project, I used a 4 degree of freedom robot simulated within Gazebo to demonstrate the effectiveness of deep Q learning in robotic grasping applications. The simulated environment reported the following information:

1) **Joint State**: The position of each joint (and velocity using approximate numerical differentiation)
2) **Target Object Bounding Box**: The pose/dimensions of the bounding box of object to be grasped
3) **Gripper Bounding Box**: The pose/dimensions of the bounding box surrounding the end effector
4) **Collisions**: Collisions between the ground, elements of the robot arm, and the target object.

The baseline project deliverable was to first touch the target with part of the any part of the arm (including the gripper) in 90% of all simulations, then, as a second deliverable, contact the target with only the gripper in 80% of all simulations. For these simulations, the target, a small tube, would normally be spawned in the same location each time. I chose instead to spawn the tube in a different randomized location along the x axis in each simulation iteration. Because the robot only needed to move along the x axis, its azimuth rotation joint was locked in place.

2 Reward Functions

Reward functions were separated into two classes: interim rewards and end of episode rewards. The interim rewards served to guide the robot closer to the target during a simulation run. End of episode rewards served as the final assessment of the effectiveness of a given set of weights.

2.1 Interim Reward

The interim reward function was defined for iteration i as:

$$R_i = (1-\alpha)(d_{i-1} - d_i) + \frac{\alpha}{n}\sum_{n=0}^{i-1} R_n$$

Where d_i is the distance from the gripper to the target at iteration i. The parameter α was tuned and set to 0.3.

This function serves to smooth out interim rewards between iterations based on a weighted moving average of rewards from all previous iterations.

I also experimented with additional reward functions, including the Huber Loss and a reward function augmented with an additional term representing the distance to the ground in z with a small offset. The former did not improve performance noticeably. With the latter, I intended to incentivize a different behavior or first moving the gripper close to the ground, then sweeping the gripper toward the target until it made contact. Unfortunately, this more complicated interim reward function actually worsened robot performance even after some experimentation with weights.

2.2 End of Episode Reward

Each episode is ended in one of four scenarios:

1) **Arm Contact**: The robot arm makes contact with the target
2) **Gripper Contact**: The gripper makes contact with the target
3) **Ground Contact**: Any part of the arm comes into contact with the ground
4) **Timeout**: The arm/gripper fails to contact the ground or the target within the specified number of iterations

I chose to combine the two goals of the project into a single combined goal for simplicity. Specifically stated, the robot's arm (including the gripper) should contact the target a minimum of 90% of episodes, and the gripper should contact the target a minimum of 80% of episodes over at least 100 episodes. Any contact with the ground or a timeout was considered a failed episode.

I used the following end of episode rewards to incentivize the desired performance:

1) **Arm Contact (Excluding Gripper) Reward**: 5.0
2) **Gripper Contact Reward**: 50.0
3) **Ground Contact Reward**: -10.0
4) **Timeout**: -1.0

In theory, a ground contact and timeout are equally undesirable, but I chose to implement a higher penalty for ground contact. While this doesn't have a technical justification, if this were a real robot, I'd want to highly disincentivize ground contact since it could damage the robot, the floor, or both.

Gripper contact was highly rewarded since this counts for both arm contact and gripper contact. Arm contact excluding the gripper was rewarded at 1/10th the rate. Though this is technically a successful episode, it is not as desirable as gripper contact. This combination of weights resulted in relatively quick learning during testing.

3 Robot Control

Based on the reward from each episode, the DQN network returned a single action. The number of options was twice the number of degrees of freedom. Essentially, the agent could choose to increment a single joint in either the positive or negative direction (position control) or to increment the velocity of each joint in the positive or negative direction (velocity control).

After some experimentation, I chose to use position control. While I initially assumed velocity control would result in smoother motions since joint velocity could be maintained between iterations until altered by the DQN network, it actually resulted in more unstable behavior. Position control resulted in more conservative, less erratic motions that servoed the gripper reliably toward the target.

While I did not have a chance to try additional control strategies, it would be interesting to see what sort of results are obtainable with a multidimensional network output that allows the robot to increase or decrease the velocity or position of all joints during each episode.

4 Hyperparameters

I used the following hyperparameters to successfully meet the accuracy requirements with a tube placed at a random position along the x axis:

- Input Images
  - Input Width: 64
  - Input Height: 64
- DQN Action Selection
  - γ: 0.9
  - ε_{start}: 0.9
  - ε_{end}: 0.05
  - ε_{decay}: 100
- LSTM Settings
  - Use LSTM: True
  - Replay Memory: 1000
  - LSTM Size: 64
- Training and Optimization
  - Batch Size: 16
  - Optimizer: RMS Prop
  - Learning Rate: 0.005

By downsampling the observed image fed to the DQN agent, I was able to significantly increase the evaluation time. With an image size of 512x512 (the project default), the Udacity cloud workspace actually ran out of memory and failed to execute the DQN agent.

The DQN action selection parameters control the probability with which the DQN agent will select a random action rather than the predicted best action based on the current network weights and input. This is particularly important early on in the training when the DQN agent has not yet settled on a reliably successful policy. I found that a reduced epsilon decay time constant of 100 (rather than the default of 200) was still sufficient to converge on a successful DQN agent. This reduction also helped to reduce the number of early losses that had to be made up to meet the accuracy requirements.

While it was possible to achieve adequate performance without activating the LSTM for the static target object, I observed much improved performance for the randomly placed object after activating the LSTM. I chose to increase the LSTM cell size from 32 to 64 to increase the number of previous iterations remaining in short term memory.

Based on previous experience, I lowered the optimizer learning rate to 0.005 to avoid overfitting or weighting early random tube placements too strongly. I chose a batch size of 16 (versus the default of 8) to increase the accuracy of the gradient calculation in the optimizer while still keeping the working memory load fairly low.

5 Results

A video of a successful agent test run (run 2 from Figure 2 can be found here.

As can be seen in Figure 2, choosing the appropriate parameters did not guarantee repeatable performance. In the first run, while the agent successfully exceeded the minimum required accuracy of 90% for arm contacts, it failed to achieve 80% accuracy for gripper contacts. With the same hyperparameters, the agent achieved nearly 90% accuracy in the second run.

TABLE 1: Agent Performance Summary

Run	1	2
Episodes	817	785
Peak Arm Accuracy	94.0% @ Episode 798	94.4% @ Episode 785
Peak Gripper Accuracy	71.4% @ Episode 21	87.5% @ Episode 775

Qualitatively, performance appeared to depend heavily on the number of contacts of the gripper on the target during the early training episodes. In run 1, the agent was not particularly lucky to have many gripper contacts, and it settled into a ”strategy” of repeating non-gripper arm contacts for a smaller reward. In run 2, the agent had a number of early gripper contacts and as a result had a greater proportion of successful outcomes over the long run.

6 Future Work

I'd like to revisit the challenge of grasping the target randomly placed in x and y (not just x). My early attempts at this had limited success. Some changes to the hyper parameters will likely be required, including increasing the short term memory buffer length and increasing the epsilon decay time constant. I'd also like to experiment with increasing the image height and width to allow for more discernible target depth (i.e. size of tube in frame) information.

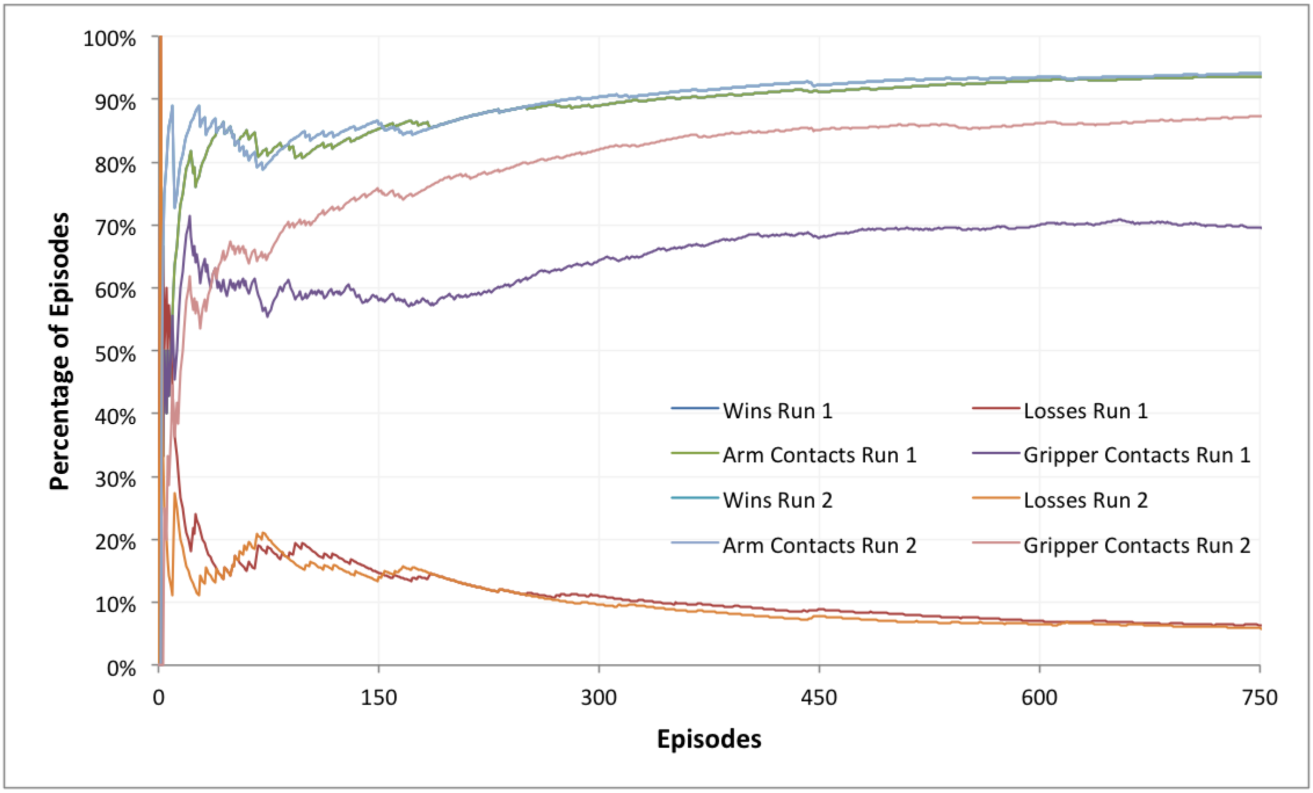


Fig. 2: Agent Performance for 2 Runs with Target at Random X Position

For a real robot, a simplistic reward policy such as was utilized in this project would not be sufficient. While it's great to successfully train the simulated robot to touch the tube with only the gripper, in a real-world application, the robot would be required to slowly approach the tube without knocking it over, come close to contacting it with the middle of the gripper, close the left and right gripper fingers around the object, and lift it off the ground. This clearly demands a more complex reward policy as well as increased granularity in the information fed to the agent. It's quite possible, though, that such a feat would be impossible with just a side view image of the workspace.