Deep RL Arm Manipulation

Objectives of Project

There are two objectives to this project. The first part is to have the robot arm touch the object with 90% accuracy for a minimum of 100 runs and the second objective is to have the arm's gripper base touch the object with at least 80% accuracy. This was done by creating a DQN Agent with position based control of its arm joints. Awards were assigned based on the robots collision with the ground and object and distance from gripper to object. Hyperparameters were tuned as well to accomplish this.

Reward Functions

By far the most important part to meet the objectives for this project was the assigning of appropriate rewards. For both objectives, the same reward function was used using the position based control of its arm joints. There were just some minor differences in hyperparameter tuning discussed in the next section. Also, a definition switch in the code labelled OBJECTIVE1 determines whether objective 1 or 2 is being tested for. OBJECTIVE1 is true if objective 1 is being tested for otherwise it's false if objective 2 is being tested for. The switch just determines which part of the arm is tested against the collision with the object for a positive reward.

The three rewards are for the gripper hitting the ground, an interim reward based on the distance to the object and a reward for a collision between either the arm and the object or a collision between the gripper base and the object. In general REWARD_WIN is set at 10 and REWARD_LOSS IS set at -10.

The reward for the gripper hitting the ground was maxEpisodeLength * REWARD_LOSS. Since the max episode length is 100, the loss for the gripper hitting the ground was set at a discouraging large -1000.

The interim reward based on the distance to the object was the same for both objectives. It is defined as:

rewardHistory = avgGoalDelta * REWARD_WIN * exp(-distGoal)

where:

avgGoalDelta = (avgGoalDelta * ALPHA) + (lastGoalDistance – distGoal) * (1 – ALPHA)

avgGoalDelta is essentially a smoothing function. It weights the importance of the past avgGoalDelta by a factor of ALPHA and the new delta distance between the distance to the goal now and the previous distance to the goal by a factor of 1 - ALPHA. This helps stabilize the movement of the arm. If ALPHA is too small, the arm is much more unstable (moving back and forth) in its pursuit of the objective.

The rewardHistory takes this smoothed delta distance and multiplies it by REWARD_WIN and an exponential function gives higher values to shorter distances to the goal encouraging the optimizer to move toward the goal quicker.

The reward for the collision between the arm and the object or the gripper base and the object (determined by the tested objective) is given by:

(maxEpisodeLength – episodeFrames) * REWARD_WIN

This reward encourages the optimizer to find a path in the shortest period of time as the less number of frames to accomplish the objective corresponds to a higher reward.

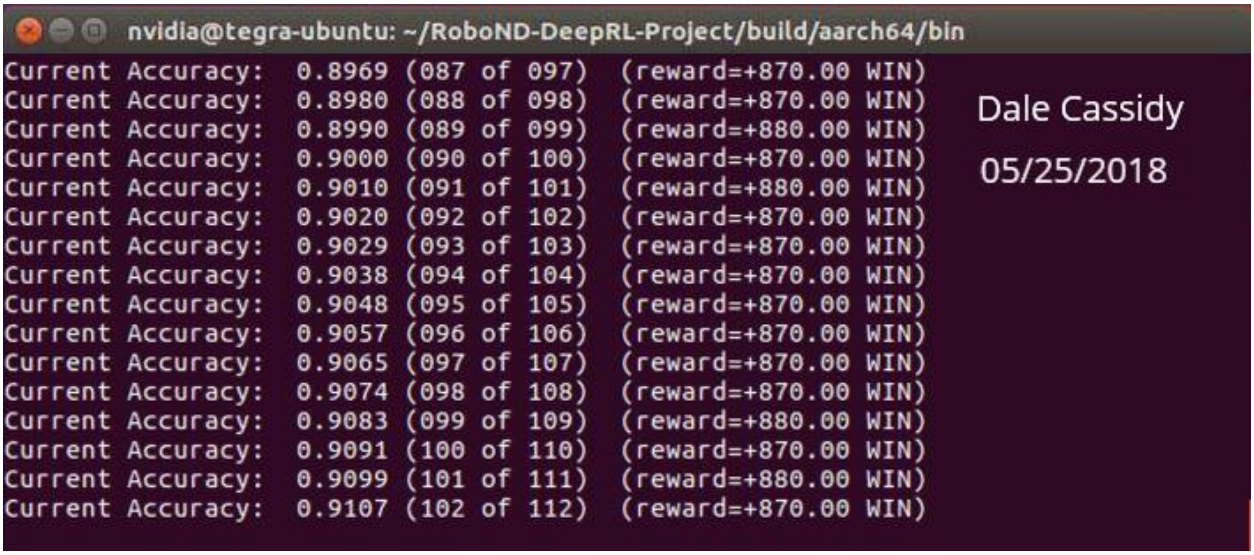
Hyperparameters

The following is a list of hyperparameters and why they were chosen:

- **OPTIMIZER "Adam"** – Chosen by trial and error. The other one tested was RMSprop but Adam performed better.
- **EPS_END 0.01f**- This helps determine how fast the learning trails off towards the end of the episode. A smaller number was chosen from trial and error because most of the learning that made the arm successful was done towards the beginning. A higher number at the end caused the arm to try and learn more when it was not needed which affected the accuracy negatively.
- **EPS_DECAY 70**- This number was also made smaller similar to the reason done so for EPS_END. A smaller decay number allowed the optimizer to learn more towards the beginning of the episode before trailing off quicker towards the end. This was done because most of the learning successfully contributing to good arm performance was done toward the beginning.
- **NUM_ACTIONS 6**- just the number of actions the optimizer could give which was two for each of the 3 joints.
- **BATCH_SIZE 8-** This number was kept as presented. Changing this experimentally didn't appear to add any extra benefits.
- **INPUT_WIDTH AND INPUT_HEIGHT 64**- these numbers were made smaller because a smaller resolution was all that was needed and used less memory and computational power.
- **LEARNING_RATE 0.03f**- This number was increased from 0.01. It seemed to perform better with more learning chance at the beginning of the episode.
- **REPLAY_MEMORY 10000 for objective 1 and 50000 for objective 2**- The original 10000 worked fine experimentally for objective 1, but for objective 2 the cyclic replay buffer was increased by a factor of 5 because it was harder for the arm to hit a smaller target (the base) to the object and therefore when it did it needed to be stored in the cyclic buffer to be sampled from again for a longer period of time. Otherwise, it may not get a chance to be resampled as easily.
- **USE_LSTM true and LSTM_SIZE 256**- this was used only for objective 1. When using LSTM, if a good path is found to the goal, LSTM allows the arm to repeatedly follow the same successful behavior. LSTM doesn't work as well if a good path is not found early on because LSTM will often repeat a less than optimal path again instead of exploring a new solution. Using LSTM did not work well experimentally for objective 2.
- **ALPHA 0.8 for objective 1 and 0.75 for objective 2-** having a slightly higher ALPHA for objective 1 than objective 2 seemed to work a little better. For objective 1 the path was found more consistently and for objective 2 it seemed to give the arm a little bit more room to move around and find an optimal solution.

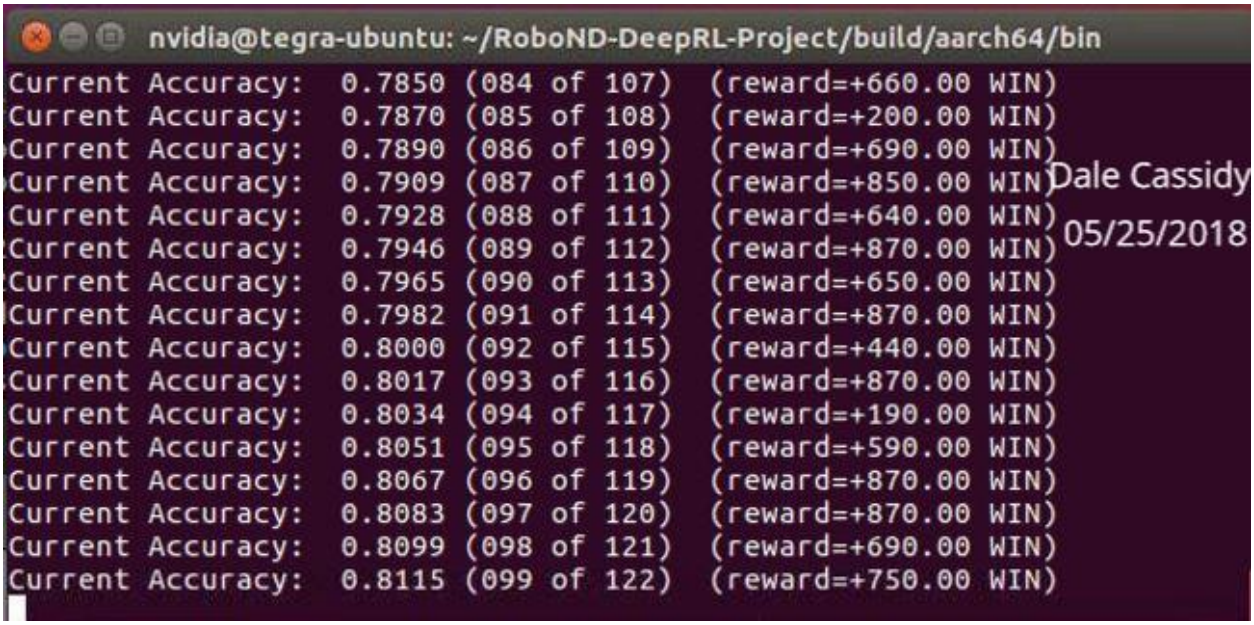
Results

The following is a screenshot of the terminal results for objective 1:



A 90% win rate was achieved at the first 100 iterations and continued to increase in accuracy. Using LSTM here was beneficial to keep the arm choosing the same optimal path.

The following are the results for the second objective:



An 80% win rate was achieved on the 115th iteration and continued to increase in accuracy. Increasing the REPLAY_MEMORY was very beneficial for this objective.

The only real difficulty noted in the performance of the DQN agent was when one of its first random paths chosen resulted in the top joint having an angle closer to 90 than 180. The incremental reward function was not as effective in these cases. Often it would get stuck very close to the object on the near side but not actually touch the object. The training took much longer in these cases to overcome initial unsuccessful attempts. This was especially true with objective 1, but was true to some degree for object 2 as well.

Future Work

As mentioned above in the previous section, there was an issue sometimes when an early path chosen put the arm on a path on the near side of the tube instead of coming in and hitting the tube from the top. Given more time, a way to control for this may be to reward the robot when its top joint is straighter (closer to 180 than 90 degrees) to encourage it to more efficiently find an optimal path.

Also given more time, it would be helpful to read research papers to see how others came up with more effective interim reward functions for this type of problem.