DeepRL Arm Project: Nick Zuiker

Introduction

This project aims to create a Deep Q-Learning Network (DQN) to train a robotic arm to meet certain objectives. The Robotic Arm used for this project is simulated within Gazebo and runs in Udacity's online workspace.

This project generally 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:

1. The student should complete all tasks specified in the Classroom, with the end objective of the robot arm touching the object with at least a 90% accuracy for a minimum of 100 runs.
2. The student should complete all tasks specified in the Classroom, with the end objective of the arm's gripper base touching the object with at least a 80% accuracy for a minimum of 100 runs.

Reward Functions

The reward functions are mostly the same for both objectives, but there are a few differences that will be considered.

Objective 1

For objective 1, the approach 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 -10 will be issued upon ground collision. However, if the arm collided with the target object, it would issue a REWARD_WIN of 10, meeting the objective and resulting in a "win" for that run. Additional 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 -10 would be issued. This would result in a "loss" for that run.

Since velocity is being used to control the arm to complete the first objective, the reward function setup for the process in between to control the robotic arm 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 dial 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 proportionately as it gets closer 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.

Objective 2

For objective 2 (robotic arm collision with the target object must only occur with the gripper), velocity control was initially tested. After many attempts, it was decided to use “position control” instead of velocity. Since the arm is attempting to touch the target solely with the gripper, more finesse and joint control seemed to be the focus. 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 -10 is issued upon ground collision. However, if the gripper collided with the target object, it would issue a REWARD_WIN of 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 -10 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 last RL 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 is 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).

Hyperparameters

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

Objective 1

Hyperparameter	Value	Reason/Intuition/Comments
INPUT_WIDTH	64	Reduced width - anything higher seemed unnecessary
INPUT_HEIGHT	64	Reduced height - anything higher seemed unnecessary
OPTIMIZER	RMSprop	Initially tried Adam, but Slack channel suggested to try RMSprop
LEARNING_RATE	0.2f	Iterated from 0.35f to 0.2f
REPLAY_MEMORY	10000	Stayed the same as the lesson suggested
BATCH_SIZE	16	8 seemed too low, but anything higher seemed unnecessary
USE_LSTM	true	Improves learning from past experiences
LSTM_SIZE	256	64 was not working, so 128 and 256 were tried before settling

Objective 2

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	0.15f	Iterated from 0.25f to 0.15f

REPLAY_MEMORY	10000	Same used as for Objective 1
BATCH_SIZE	32	Bumped this up from 16 in attempt to increase learning
USE_LSTM	true	Same used as for Objective 1
LSTM_SIZE	256	Same used as for Objective 1

DQN API Settings

These settings were changed to decrease exploration and increase exploitation of previously found solutions. These settings decreased initial exploration probability and decreased that probability more quickly over time compared to the first objective.

Parameter	Value
EPS_START	0.8f
EPS_END	0.01f
EPS_DECAY	300

Results

Objective 1

Objective 1 was surprisingly difficult to train given the high accuracy that was required. The robotic arm intuitively began learning faster as hyperparameters were tuned and the reward values were increased. For each iteration of testing, at the beginning (before learning the objective), the arm started colliding with the ground, constantly receiving losses. Increasing the negative rewards for ground collisions didn't dramatically improve this behavior. However, this behavior did not occur as often in the final iteration.

The robot arm often tried to explore other ways to solve the problem, obtaining multiple solution approaches along the way. Ultimately, the final iteration for objective 1 managed to get to 90% accuracy after a few hundred episodes, but it was kept running until just over 800 episodes.

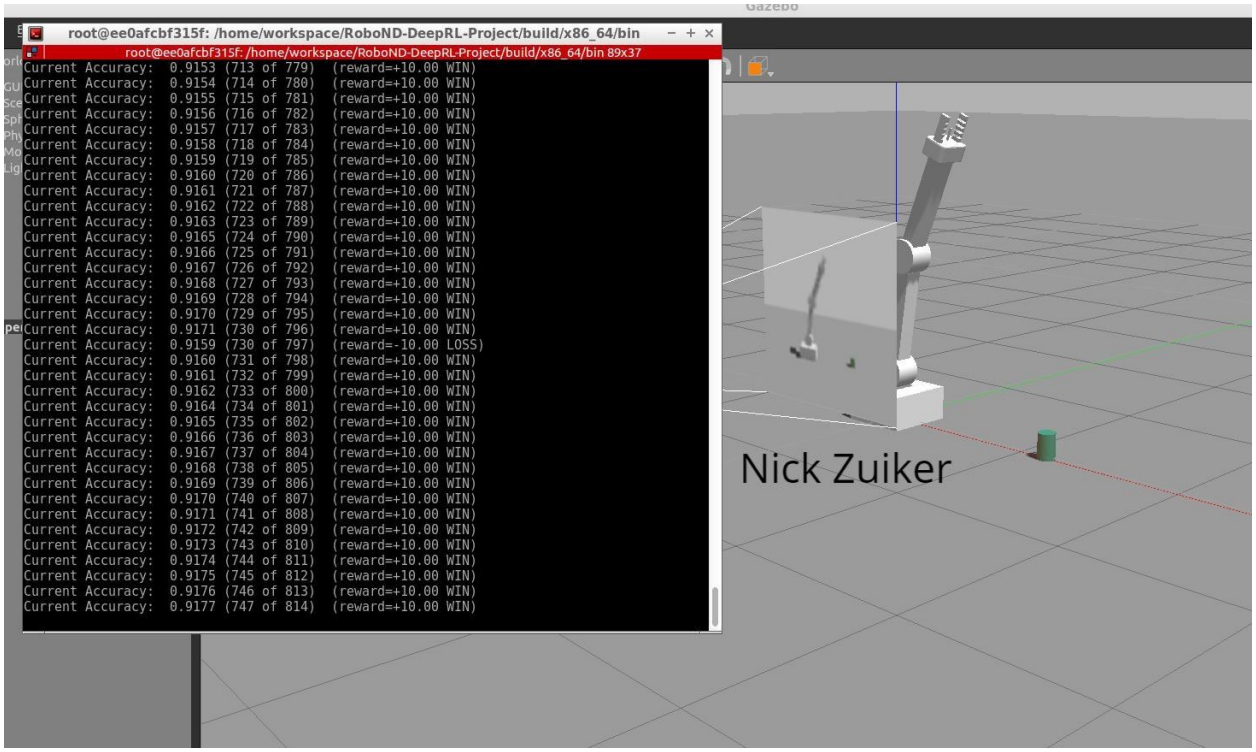

Figure 1: Objective 1 Accuracy

Objective 2

Naturally, objective 2 required more planning in its behavior than objective 1. After iterating through many sessions of hyperparameter tuning, it was found that the agent was trying to explore too much. Therefore, 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. The fact that the learning rate was decreased and the batch size increased seemed to help the agent to train and learn more precisely than in the first objective.

As with objective 1, the agent was shown contacting the ground hard enough to sometimes break, at least in the initial stages of the iteration. This a problem that would require much greater attention if this were a real physical system.

The robotic arm exceeded the minimum accuracy of 80% reaching up to 86% accuracy after exactly 100 runs. This seemed to be lucky outlier. Most attempts needed a few hundred episodes to get close to the required accuracy. Nonetheless, the result still serves as a proof of concept for the hyperparameters chosen.

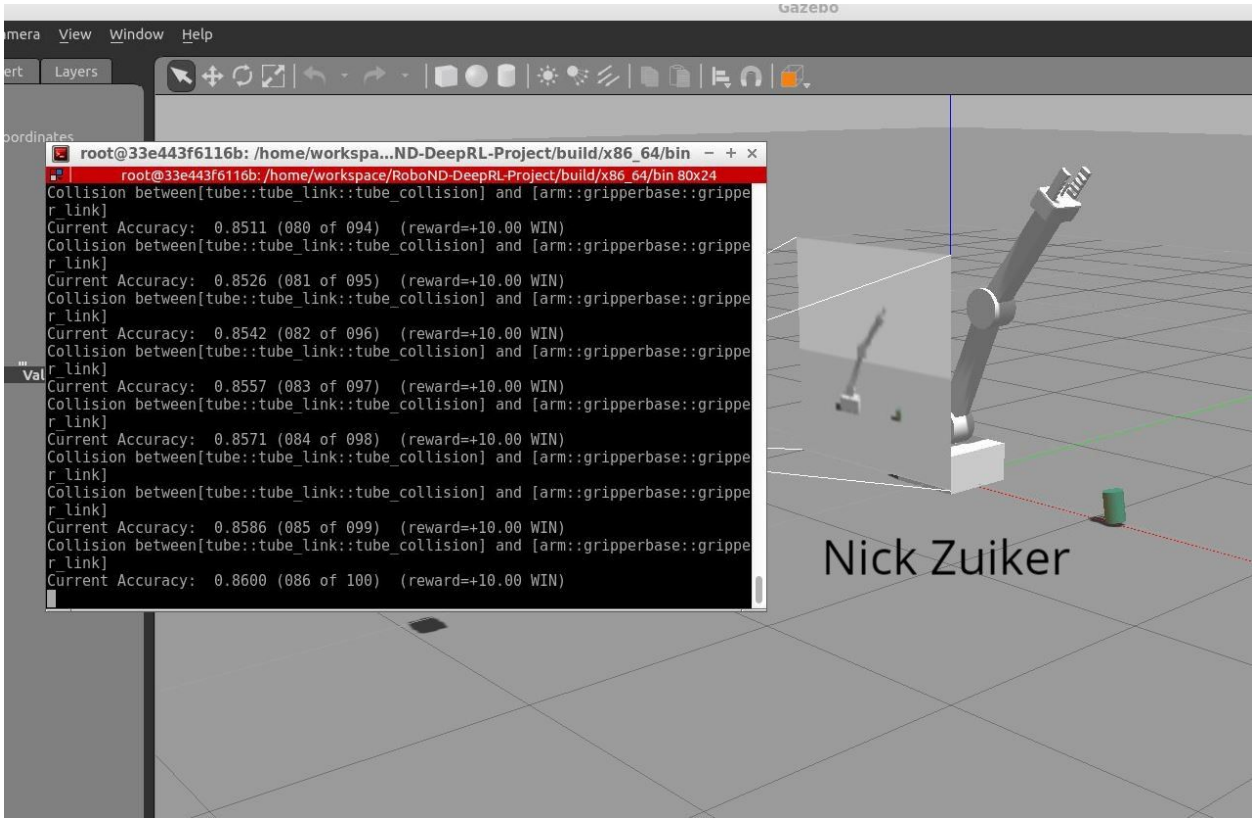

Figure 2: Objective 2 Accuracy

As expected, the second objective was harder to reach because training the robotic arm to have finesse and precision required finer tuning of hyperparameters and (in this case) DQN API settings. Ultimately, both agents were trained to meet the objectives presented.

Future Work

For future work, improvement on the hyperparameter tuning for the first objective could yield close to 100% accuracy. It seemed like the arm should have figured out that it could simply straighten fully and bring the arm down on the can for objective 1, but this behavior was never fully realized in this project. Effort could also be spent on the reward function to train the arm much faster. In true industry, having to wait through lengthy periods of time for robots to learn how to perform tasks may not be feasible. On that note, ground collision mitigation would also be an extremely important aspect of bringing these robots to industry. Forceful collisions while carrying valuable items could be catastrophic. This could likely be mitigated by using positional control and limiting the state space that the robot was allowed to enter.