Deep RL Arm Manipulation

The goal to the Deep RL Arm Manipulation project is to create a DQN agent and define reward functions to teach a robotic arm to carry out two primary objectives:

1. *Have any part of the robot arm touch the a designated object on the ground, with at least a 90% accuracy.*
2. *Have only the gripper base of the robot arm touch the designated object, with at least a 80% accuracy*

The template project is based on the Nvidia open source project "jetson-reinforcement" developed by Dustin Franklin.

Introduction

Deep Reinforcement Learning is an active area of research in robotics. This project is a challenging Deep RL implementation using a gazebo environment and an arm plugin (ArmPlugin.cpp) that assigns rewards, either **positive** or **negative**, based on simple rules. During repeated attempts by the robot to touch a cylinder in the gazebo world without touching the ground, the goal is to train the robot arm to reach the desired accuracy goal. The project was implemented on a Jetson TX2 board running Linux 16.04. The project sources reside in the git directory : https://github.com/mbufi/RoboND-DeepRL-Project

Reward functions

The reward functions are defined in ArmPlugin.cpp. The arm joints were updated using position control (default settings) instead of using velocity control. For each of the joints, there are two actions; increase or decrease the joint position.

- **`REWARD_WIN`** was set to `0.125` (`0.1` for the second objective) with **`REWARD_LOSS`** to `-0.125` (`-0.1` for the second objective).
- If the robot gripper hit the ground, a **`REWARD_LOSS * 10`** was given and the episode ended.

Within the episode, interim rewards were issued if there was no ground contact or 100 frames had not been exceeded.

The main interim reward of each episode was based on the distance goal delta between the gripper and the Cylinder prop. If a positive weighted average was derived then a **`REWARD_WIN`** was recorded otherwise **`REWARD_LOSS * distance_to_goal`** was issued. Thus the **`REWARD_LOSS`** would be higher the further the arm was from the goal.

An additional **`REWARD_LOSS`** was added in the second objective for the gripper base stating:

- if the absolute average goal delta was < `0.001` to penalize no movement.

For the first objective:

- If the robot arm hit the prop, a **`REWARD_WIN * 10`** was used

For the second objective:

- `REWARD_LOSS * 5` if the collision was not with the `gripper_middle`.
- `REWARD_WIN * 20` was issued if the collision point was `gripper_middle` .

For both objectives , **any collision ends the episode**.

Hyper Parameters

Image dimensions were set to the same size as the input. Training was performed on a the Jetson TX2, which required no fine tuning to improve performance. INPUT_WIDTH 64 x INPUT_HEIGHT 64

The OPTIMIZER was changed from none to Adam. Possible common options are Adam or RMSProp, which are instances of adaptive learning rates to use in gradient-based optimization of parameters. Adam slightly outperforms RMSprop, which in turn gave good results. The parameter use LSTM was set to true to enable the LSTM agent

For objective 1 the LEARNING_RATE was 0.1 with REPLAY_MEMORY at 1000 . The value was chosen via trail and error.

For objective 2 the LEARNING_RATE was decreased to 0.01 due to the higher REPLAY_MEMORY set at 20000 . The higher REPLAY_MEMORY was used for more discrete learning, due to the smaller surface area required to achieve a collision to meet objectives.

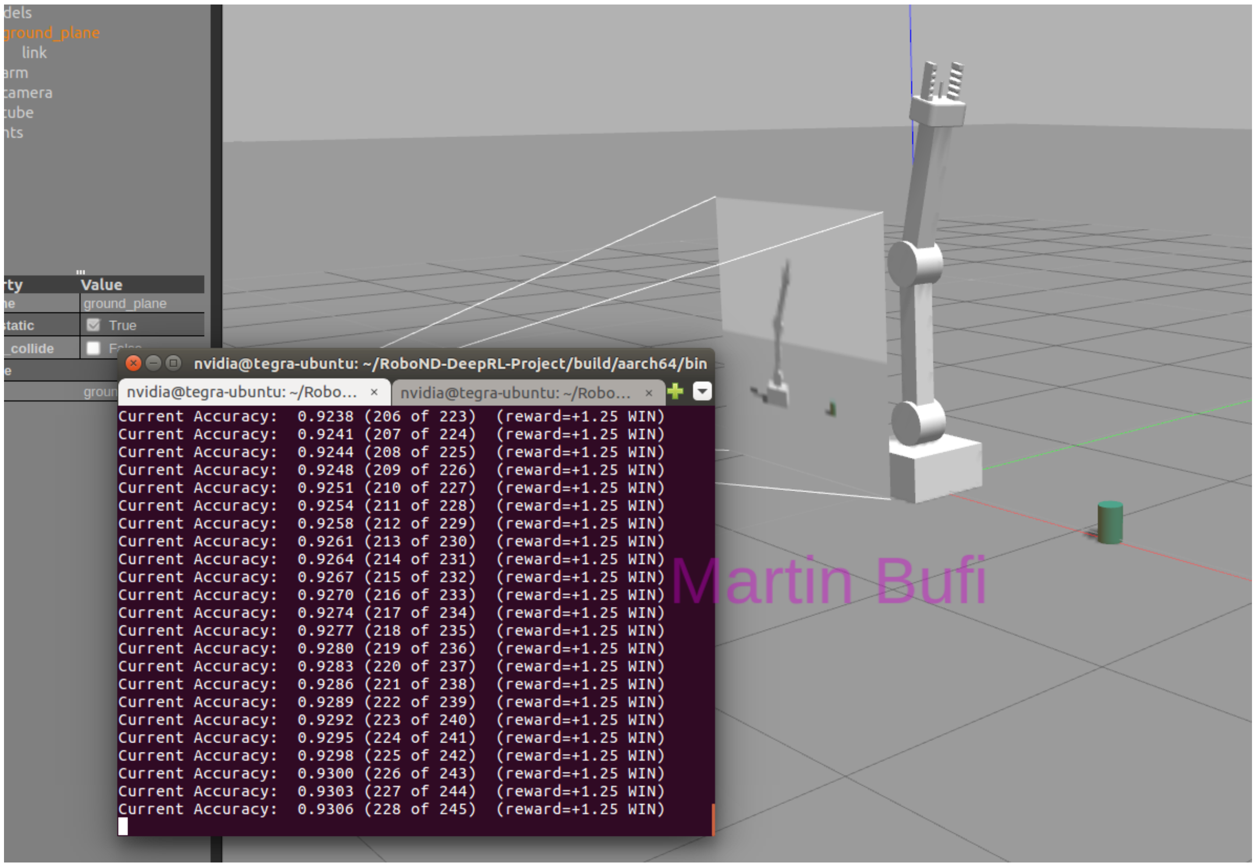
BATCH_SIZE was set to 512 (again sufficient memory on the TX2).

LSTM (Long short-term memory) was used USE_LSTM = true with a size of LSTM_SIZE = 256 which was set via trial and error.

In general, most parameters were set based on trail and error since I was using a TX2 and not a regular graphics card. TX2 is an experimental hardware platform, therefore optimization of parameters is extremely crucial for proper training.

Results

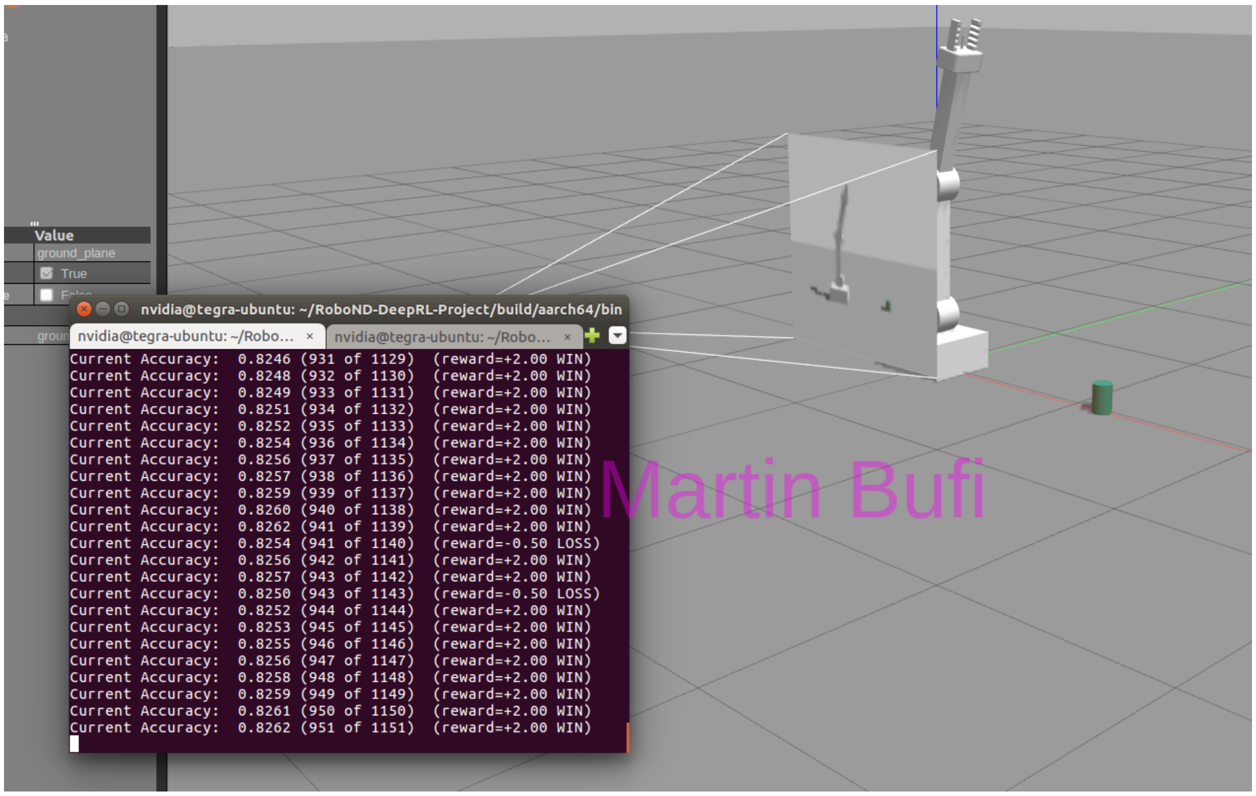
Objective 1 : Have any part of the robot arm touch the object of interest, with at least a 90% accuracy for a minimum of 100 runs.



```
nvidia@tegra-ubuntu: ~/RoboND-DeepRL-Project/build/aarch64/bin
Current Accuracy:  0.9238 (206 of 223)  (reward=+1.25 WIN)
Current Accuracy:  0.9241 (207 of 224)  (reward=+1.25 WIN)
Current Accuracy:  0.9244 (208 of 225)  (reward=+1.25 WIN)
Current Accuracy:  0.9248 (209 of 226)  (reward=+1.25 WIN)
Current Accuracy:  0.9251 (210 of 227)  (reward=+1.25 WIN)
Current Accuracy:  0.9254 (211 of 228)  (reward=+1.25 WIN)
Current Accuracy:  0.9258 (212 of 229)  (reward=+1.25 WIN)
Current Accuracy:  0.9261 (213 of 230)  (reward=+1.25 WIN)
Current Accuracy:  0.9264 (214 of 231)  (reward=+1.25 WIN)
Current Accuracy:  0.9267 (215 of 232)  (reward=+1.25 WIN)
Current Accuracy:  0.9270 (216 of 233)  (reward=+1.25 WIN)
Current Accuracy:  0.9274 (217 of 234)  (reward=+1.25 WIN)
Current Accuracy:  0.9277 (218 of 235)  (reward=+1.25 WIN)
Current Accuracy:  0.9280 (219 of 236)  (reward=+1.25 WIN)
Current Accuracy:  0.9283 (220 of 237)  (reward=+1.25 WIN)
Current Accuracy:  0.9286 (221 of 238)  (reward=+1.25 WIN)
Current Accuracy:  0.9289 (222 of 239)  (reward=+1.25 WIN)
Current Accuracy:  0.9292 (223 of 240)  (reward=+1.25 WIN)
Current Accuracy:  0.9295 (224 of 241)  (reward=+1.25 WIN)
Current Accuracy:  0.9298 (225 of 242)  (reward=+1.25 WIN)
Current Accuracy:  0.9300 (226 of 243)  (reward=+1.25 WIN)
Current Accuracy:  0.9303 (227 of 244)  (reward=+1.25 WIN)
Current Accuracy:  0.9306 (228 of 245)  (reward=+1.25 WIN)
```

For reasons unknown, the robotic arm struggled at first to hit the prop. After some trial and error, it began learning how to hit it properly and with a degree of accuracy in a repeatable fashion. On occasion the arm would try a different path and begin losing its training. But due to having memory, it would then go back to using the winning sequence to hit the prop. As shown above, the objective was achieved well within the criteria specified with an accuracy of 92% before I stopped the script. I am sure it would have gone higher, if left to run longer.

<u>Objective 2:</u> Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy for a minimum of 100 runs.


Value
ground_plane
True
nvidia@tegra-ubuntu: ~/RoboND-DeepRL-Project/build/aarch64/bin
nvidia@tegra-ubuntu: ~/Robo...
nvidia@tegra-ubuntu: ~/Robo...
Martin Bufi

```
Current Accuracy:  0.8246 (931 of 1129)  (reward=+2.00 WIN)
Current Accuracy:  0.8248 (932 of 1130)  (reward=+2.00 WIN)
Current Accuracy:  0.8249 (933 of 1131)  (reward=+2.00 WIN)
Current Accuracy:  0.8251 (934 of 1132)  (reward=+2.00 WIN)
Current Accuracy:  0.8252 (935 of 1133)  (reward=+2.00 WIN)
Current Accuracy:  0.8254 (936 of 1134)  (reward=+2.00 WIN)
Current Accuracy:  0.8256 (937 of 1135)  (reward=+2.00 WIN)
Current Accuracy:  0.8257 (938 of 1136)  (reward=+2.00 WIN)
Current Accuracy:  0.8259 (939 of 1137)  (reward=+2.00 WIN)
Current Accuracy:  0.8260 (940 of 1138)  (reward=+2.00 WIN)
Current Accuracy:  0.8262 (941 of 1139)  (reward=+2.00 WIN)
Current Accuracy:  0.8254 (941 of 1140)  (reward=-0.50 LOSS)
Current Accuracy:  0.8256 (942 of 1141)  (reward=+2.00 WIN)
Current Accuracy:  0.8257 (943 of 1142)  (reward=+2.00 WIN)
Current Accuracy:  0.8250 (943 of 1143)  (reward=-0.50 LOSS)
Current Accuracy:  0.8252 (944 of 1144)  (reward=+2.00 WIN)
Current Accuracy:  0.8253 (945 of 1145)  (reward=+2.00 WIN)
Current Accuracy:  0.8255 (946 of 1146)  (reward=+2.00 WIN)
Current Accuracy:  0.8256 (947 of 1147)  (reward=+2.00 WIN)
Current Accuracy:  0.8258 (948 of 1148)  (reward=+2.00 WIN)
Current Accuracy:  0.8259 (949 of 1149)  (reward=+2.00 WIN)
Current Accuracy:  0.8261 (950 of 1150)  (reward=+2.00 WIN)
Current Accuracy:  0.8262 (951 of 1151)  (reward=+2.00 WIN)
```

With the finer control required (only hitting the gripper based), and alteration to the interim reward system, this configuration would often hesitate before making a move. During the early stages of training, it learned how to get very close to the prop but never make it there since it was so hesitant. Once it learned how to effectively hit the prop with the middle of the gripper, it began experimenting other ways to do so. Such as extending past the prop and then coming back for it in an arch (as if it was digging), but this failed due to it usually hitting the prop with its arm joint. This whole process of learning the proper way to do it without hesitation took the arm over 100 iterations before becoming more confident. Decreasing the learning rate to 0.01 and increasing LSTM memory to 20000, were what allowed it to learn the precise movements to reliably complete the objective.

Discussion

Training a DeepRL agent is a time consuming task, often taking many hours of computer time. Tuning parameters seems like black magic. The variability from run to run seems to far exceed the effect of any parameter changes. Tuning parameters such as LSTM size can give different results, some of them amusing and unexpected. Perhaps using a different approach to the problem, such as using a genetic algorithm would allow for less manual labor of tuning hyper parameters.

Conclusion and Future Work

This C++ API was developed by Nvidia and optimized specifically for embedded platforms such as the TX2 performed extremely well on the native TX2 platform. The code took advantage of the Jetson CUDA architecture and ran the episodes at a rate of about one per second. Building the project on the Jetson platform was very simple and it performed well. With some fine tuning, one would not notice any performance degradation in such a small platform.

Due to the randomness of the algorithm in this project, one begins to wonder if this randomness could be reduced or channeled by looking at Genetic Algorithms for Deep RL. The idea is that populations of living organism (or in this case, the arm) learn to adapt to their environment by a means of natural selection. Perhaps something such as this could be used to give the DeepRL agent more guidance in choosing parameters to accomplish the goals. This would allow the user (myself) to start up the script and allow the generations of arms to learn how to adjust the hyper parameters themselves, leaving the developer to sit back and watch the magic unfold. This is something I would like to further research and perhaps implement on the TX2.