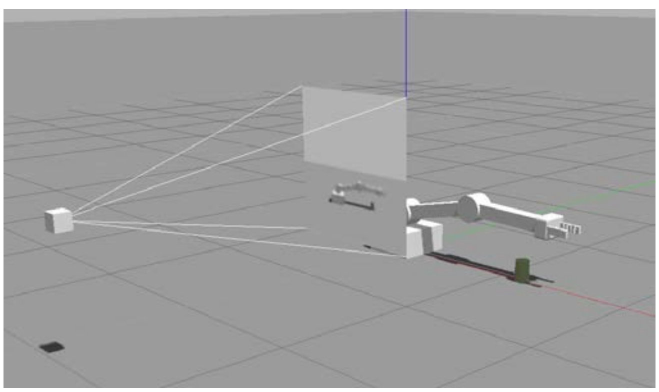
Deep RL Arm Manipulation for Udacity's Robotics Software Engineer Program

Leroy Friesenhahn

INTRODUCTION

Watching a child learn to walk can be a trying but rewarding time for parents. The failed attempts, occasional successes and finally the learned ability to walk is a natural ways humans acquire a skill. The Deep RL Arm Manipulation project provides the robot with the ability to learn the process of touching a cylinder with a high degree of accuracy. Just as a child receives encouragement for successes (rewards) and a bump to their bottom for failures(penalties), the RL Arm uses success and failure to learn its task in a way that it can repeat its task with a high degree of accuracy.



Robot Arm attempting to touch the object

Two objectives specified for this project.

1. Robot arm touches the object without the gripper touching the ground with greater than 90% accuracy for at least 100 attempts
2. Only the gripper is allow to touch the object without touching the ground with at least 80% accuracy for at least 100 attempts

REWARD FUNCTIONS

A system of rewards and penalties is used to train the Deep RL Arm DQN agent. The reward/penalty function can be found in the ArmPlugin.cpp program. This module provided a reward of +20 for touching the object with the robot arm in objective 1 and only the gripper in objective 2.

```
ArmPlugin.cpp
#define INPUT_CHANNELS 3   // originally 3
#define ALLOW_RANDOM true
#define DEBUG_DQN false
#define GAMMA 0.9f
#define EPS_START 0.7f
#define EPS_END 0.02f
#define EPS_DECAY 200

/*
/ TODO - Tune the following hyper parameters
/
*/

#define INPUT_WIDTH   64
#define INPUT_HEIGHT  64
//#define OPTIMIZER "RMSprop"
#define OPTIMIZER "Adam"
#define LEARNING_RATE 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 256
#define USE_LSTM true
#define LSTM_SIZE 256

/*
/ TODO - Define Reward Parameters
/
*/

#define REWARD_WIN  20.0f    // start with a value of 20
#define REWARD_LOSS -20.0f   // start with a value of -20
#define INTERIM_REWARD 4.0f
#define INTERIM_OFFSET 0.3f
#define ALPHA 0.4f          // start with a value of 0.4f
```


ARMPlugin.cpp parameters

A penalty of -20 was given if the robot touched the ground before completing its objective. This would discourage the robot arm from repeating the previous action. A penalty of -20 was issued if the robot was unable to complete its objective within 100 frames. Although the first objective is a binary task (touch the object or not) as compared to the 2^{nd} objective of only touching the object with the gripper, both objectives used the same reward system. This is most likely the reason the 2^{nd} process took longer to achieve the > 80% requirement.

An interim reward was given based on the current position of the robot arm. The method used is the Smoothed Moving Average (SMMA) of the delta of the distance to the goal.

avgGoalDelta = (avgGoalDelta * ALPHA) + (distDelta * (1.0f – ALPHA)

Where:

avgGoalDelta – is the resulting reward function to allow the agent to determine if it is getting closer to the object

distDelta - Last distance to the goal – current distance to the goal

ALPHA - A constant between 0 and 1 used to weight the recent delta. This allows for the most current delta to have a greater impact on the results without removing the previous delta's impact. This results in a smoother feedback to the agent.

The SMMA assist in eliminating fluctuations and provides the agent a roadmap to prevailing paths to maximize its rewards,

The agent rewards are based on distance and position of the robot (or gripper) rather than velocity. Joint control is used as the means of controlling the robot's position.

HYPER PARAMETERS

THE hyperparameters used by the agent shapes the ability to determine the speed, accuracy and the method for learning the objective.

Adjusting the hyperparameter is a job of skill, experience and many times trial and error. The initial change was to reduce the input size to 64 by 64. A square image reduces the difficulty of matrix operations and optimizing it for the GPU. After experimenting with RMSprop and Adam, the decision was to use Adam. It appears to provide a faster developing solution.

Selecting a learning rate was the next challenge. After trying rate from 0.01 to 0.1, the learning rate of 0.1 was used to obtain both objectives. Although a learning rate of 0.01 would improve the speed at which the correct path was learned, an attempt to use the same hyperparameters to complete both task was attempted. To accommodate this learning rate the replay_memory was set to 10000.

LSTM was used with a LSTM_SIZE of 256. LSTM is used to keep track of both long and short term memory. This will allow the agent to use both past and present camera frames in training the agent.

Experimenting with the ALPHA value (final value of 0.4f) showed some signs of improvement

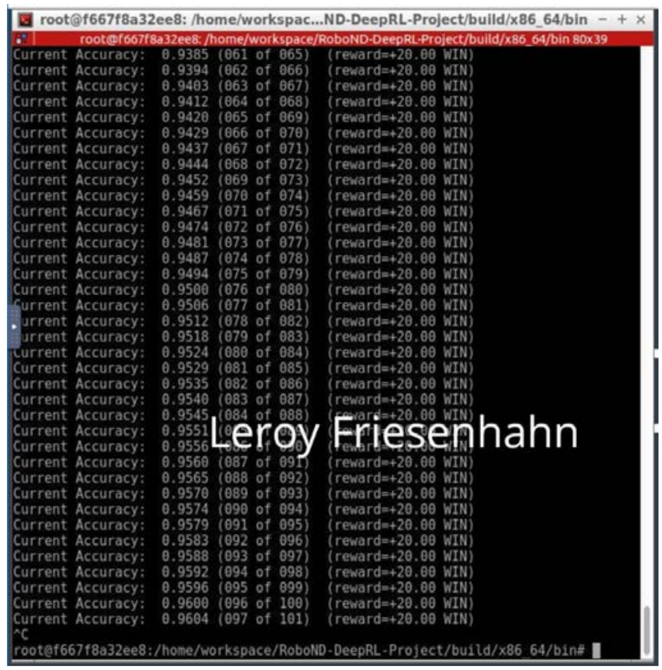
```
ArmPlugin::ArmPlugin()
ArmPlugin::Load('arm')
PropPlugin::Load('tube')
[deepRL]  use_cuda:        True
[deepRL]  use_lstm:        1
[deepRL]  lstm_size:       256
[deepRL]  input_width:     64
[deepRL]  input_height:    64
[deepRL]  input_channels: 3
[deepRL]  num_actions:     6
[deepRL]  optimizer:       Adam
[deepRL]  learning rate:   0.1
[deepRL]  replay_memory:   10000
[deepRL]  batch_size:      256
[deepRL]  gamma:           0.9
[deepRL]  epsilon_start:   0.7
[deepRL]  epsilon_end:     0.02
[deepRL]  epsilon_decay:   200.0
[deepRL]  allow_random:    1
[deepRL]  debug_mode:      0
[deepRL]  creating DQN model instance
[deepRL]  DRQN::__init__()
[deepRL]  LSTM (hx, cx) size = 256
[deepRL]  DQN model instance created
[deepRL]  DQN script done init
[cuda]  cudaAllocMapped 49152 bytes, CPU 0x2049a0000 GPU 0x2049a0000
[deepRL]  pyTorch THCState  0xACE06100
[cuda]  cudaAllocMapped 12288 bytes, CPU 0x204aa0000 GPU 0x204aa0000
ArmPlugin - allocated camera img buffer 64x64  24 bpp  12288 bytes
[deepRL]  nn.Conv2d() output size = 800
```


List of hyperparameters from ArmPlugin.cpp

.

RESULTS

Objective 1 – Any part of the robot touching the object with at least 90% accuracy for a minimum of 100 runs without touching the ground.



```
root@f667f8a32ee8: /home/workspac...ND-DeepRL-Project/build/x86_64/bin
root@f667f8a32ee8: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x39
Current Accuracy:  0.9385 (061 of 065)  (reward=+20.00 WIN)
Current Accuracy:  0.9394 (062 of 066)  (reward=+20.00 WIN)
Current Accuracy:  0.9403 (063 of 067)  (reward=+20.00 WIN)
Current Accuracy:  0.9412 (064 of 068)  (reward=+20.00 WIN)
Current Accuracy:  0.9420 (065 of 069)  (reward=+20.00 WIN)
Current Accuracy:  0.9429 (066 of 070)  (reward=+20.00 WIN)
Current Accuracy:  0.9437 (067 of 071)  (reward=+20.00 WIN)
Current Accuracy:  0.9444 (068 of 072)  (reward=+20.00 WIN)
Current Accuracy:  0.9452 (069 of 073)  (reward=+20.00 WIN)
Current Accuracy:  0.9459 (070 of 074)  (reward=+20.00 WIN)
Current Accuracy:  0.9467 (071 of 075)  (reward=+20.00 WIN)
Current Accuracy:  0.9474 (072 of 076)  (reward=+20.00 WIN)
Current Accuracy:  0.9481 (073 of 077)  (reward=+20.00 WIN)
Current Accuracy:  0.9487 (074 of 078)  (reward=+20.00 WIN)
Current Accuracy:  0.9494 (075 of 079)  (reward=+20.00 WIN)
Current Accuracy:  0.9500 (076 of 080)  (reward=+20.00 WIN)
Current Accuracy:  0.9506 (077 of 081)  (reward=+20.00 WIN)
Current Accuracy:  0.9512 (078 of 082)  (reward=+20.00 WIN)
Current Accuracy:  0.9518 (079 of 083)  (reward=+20.00 WIN)
Current Accuracy:  0.9524 (080 of 084)  (reward=+20.00 WIN)
Current Accuracy:  0.9529 (081 of 085)  (reward=+20.00 WIN)
Current Accuracy:  0.9535 (082 of 086)  (reward=+20.00 WIN)
Current Accuracy:  0.9540 (083 of 087)  (reward=+20.00 WIN)
Current Accuracy:  0.9545 (084 of 088)  (reward=+20.00 WIN)
Current Accuracy:  0.9551
Current Accuracy:  0.9556
Current Accuracy:  0.9560 (087 of 091)  (reward=+20.00 WIN)
Current Accuracy:  0.9565 (088 of 092)  (reward=+20.00 WIN)
Current Accuracy:  0.9570 (089 of 093)  (reward=+20.00 WIN)
Current Accuracy:  0.9574 (090 of 094)  (reward=+20.00 WIN)
Current Accuracy:  0.9579 (091 of 095)  (reward=+20.00 WIN)
Current Accuracy:  0.9583 (092 of 096)  (reward=+20.00 WIN)
Current Accuracy:  0.9588 (093 of 097)  (reward=+20.00 WIN)
Current Accuracy:  0.9592 (094 of 098)  (reward=+20.00 WIN)
Current Accuracy:  0.9596 (095 of 099)  (reward=+20.00 WIN)
Current Accuracy:  0.9600 (096 of 100)  (reward=+20.00 WIN)
Current Accuracy:  0.9604 (097 of 101)  (reward=+20.00 WIN)
^C
root@f667f8a32ee8:/home/workspace/RoboND-DeepRL-Project/build/x86_64/bin#
```


Result for objective 1 > 90%

Objective 2 – The gripper of the robot arm touching the object with at least 80% accuracy for a minimum of 100 runs without touching the ground.

Results of objective 2 > 80%

Conclusion/Future Work

The project is complete. Like watching a child learning to walk, I found myself encouraging the robot to continue to find its path (as if it could see me and respond to my verbal commands). Using similar hyperparameters for the robot to meet both objectives, the robot took considerable more attempts to meet the > 80% objective.

The most interesting observation in both run was in the first 30 attempts. Executing the process multiple times it with the same hyperparameters, it was found to produce different learning rates. If the robot achieved multiple successes within the first 30 attempts, the specified objective was achieved in a reasonably time. If within the first 30 attempts, the achievement rate was very low, it took the robot a larger amount of attempts to achieve the specified goals. This is an area that could use additional investigation. In the interest of time, the project was completed when the objectives were achieved.

I wish to thank to my fellow classmates for assisting me in completing this project including those on the Slack Channel.