Deep Reinforcement Learning Project - Deep RL Arm Manipulation

Navid Safaeian

1 Introduction

For this project, your goal is to create a DQN (Deep Q-Learning Network) agent and define reward functions to teach a robotic arm to meet certain objectives. The Robotic Arm used for this project is simulated on Gazebo with C++ API. This project basically leverages an existing DQN that gets instantiated with specific parameters to run the Robotic Arm. To implement Deep Reinforcement Learning, Reward functions and hyperparameters must be defined. There are two tasks in this project:

- Have any part of the robot arm touch the object of interest, with at least a 90% accuracy for a minimum of 100 runs.
- Have only the gripper base of the robot arm touch the object, with at least an 80% accuracy for a minimum of 100 runs.

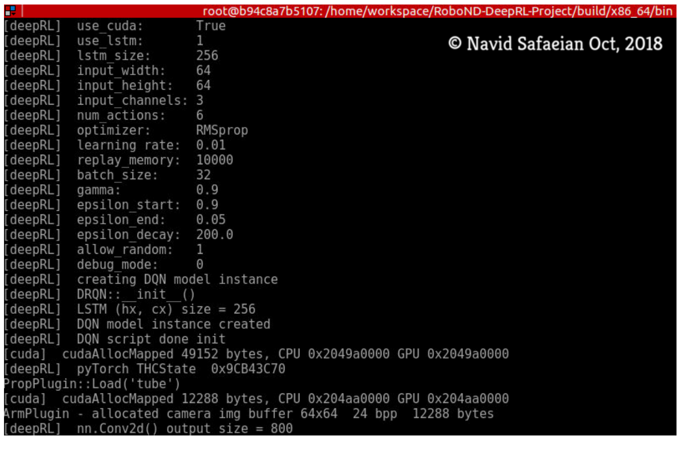
2 Reward Functions

For this project, Deep Q-Network (DQN) output is the control of each joint for the simulated robotic arm. Control of the joint movements can be based on velocity, position, or a mix of both. Position control approach was selected in case of this project.
Reward system was designed to train the robot to have any part of the robot arm touch the object of interest in one attempt (task 1) then have the gripper base of the robot arm touch the object in a second attempt (task 2).
The robotic arm/gripper must never collide with the ground, therefore, a **Reward-Loss** will be issued upon ground collision (-300 Reward-Loss). However, if the arm/gripper collided with the target object, it would issue a **Reward-Win**, meeting the objective and resulting in a win for that (+300 Reward-Win) run. Another criteria was added, to encourage the arm to accomplish its objective before the episode was over (frame span). Each episode is limited to a certain number of attempts, penalty (Reward-Loss of -300) will be issued if maximum length of the episode was reached without winning. A penalty will be issued and episode will be ended if robot arm touched ground. Interim reward or interim penalty will be issued while robot arm is moving based on the distance from the object of interest. A win reward is issued and episode is ended when collision happens with the object. In case of task 2 (robotic gripper base), 10% of the -300 (or -30) will also be deducted if collision of the robot with the target object was not with robot gripper base.
Interim rewards are obtained based on a smoothed moving average of the delta of the distance from the robot arm/gripper to the object of interest. The calculation would be as following:

$$dist = last\_distance\_to\_goal - current\_distanced\_to\_goal$$
$$average\_delta = (average\_delta * alpha) + (dist * (1 - alpha))$$

Then interim reward is computed based on the Average Delta. In this project, *alpha* is a smoothing factor to control average distance. *alpha* was chosen to be 0.3 based on what was mentioned in project details.
Another reward parameter (**Reward-Mult**) as a multiplier was used to control the amount of points given in each interim reward (loss or win) based in distance from object of interest. In this project multiplier was chosen to be 200 since the delta of distance was ranging from 0.4 to 1.4 so the rewards was ranging from 80 to 280. This can be changed based on the distance of the target from the robot arm.
Considering the fact that this project tasks are Episodic; the above rewards parameters was chosen so that goal of touching the target object can be framed as the maximization of (expected) cumulative reward.

3 DQN Hyper-parameter Tuning



```
root@b94c8a7b5107: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin
[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:      RMSprop
[deepRL]  learning rate:  0.01
[deepRL]  replay_memory:  10000
[deepRL]  batch_size:     32
[deepRL]  gamma:          0.9
[deepRL]  epsilon_start:  0.9
[deepRL]  epsilon_end:    0.05
[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  0x9CB43C70
PropPlugin::Load('tube')
[cuda]  cudaAllocMapped 12288 bytes, CPU 0x204aa0000 GPU 0x204aa0000
ArmPlugin - allocated camera img buffer 64x64  24 bpp  12288 bytes
[deepRL]  nn.Conv2d() output size = 800
```

Fig. 1. Objective 2 Parameters

Tuning the hyperparameter was an iteration process for both objectives. Fig. 1 shows the parameters used in DQN RL algorithms for objective 2 (task 2). Following parameters were adjusted to get good outcomes from the DQN agent.

INPUT WIDTH x INPUT HEIGHT : At every simulation iteration, every camera frame is fed into the DQN agent which leads to a prediction and an appropriate action by the agent. Size of the input or the dimension of the camera frame is decided by these two parameters. The first thing that came to mind when tuning the parameters was to condense the input size to reduce the complexity, so the input size was reduced from the default 512x512 to smaller square image of 128x128. Keeping the square image also reduces the difficulty of matrix operations thus optimizing it for the GPU. To further optimize it and speed up the operations, the batch size was also reduced to 64.

OPTIMIZER: There are many gradient descent to optimize the DQN algorithm: Adam, RMSprop, Adagrad, etc. In this project RMSprop and Adam were tested and it produced similar results. However, RMSprop was the only optimizer where the agent accomplished the goal for both tasks, so this was elected to be the optimizer of choice.

LEARNING_RATE: Setting up the learning rate was an iterative process of decreasing an initial rate by small increments until convergence. This parameter tells the optimizer how far to move the weights in the direction of the gradient for a batch. The low learning rate leads to a more reliable optimization, but it will take a lot of time because steps towards the minimum of the loss function are tiny. In opposite, in a high learning rate, training may not converge or even diverge. For task 1, 0.3 was worse than 0.2, 0.1 was too slow to converge however with value of 0.2 it was possible to reach the targeted accuracy. for Task 2 0.2 did not work well 0.01 was required to reach the target accuracy.

REPLAY_MEMORY: A cyclic buffer that stores the transitions that the DQN agent observes for later reuse by sampling from it randomly. the transitions that build up a batch are decorrelated. It has been shown that this greatly stabilizes and improves the DQN training procedure. This number was selected to be 10000 which means it will be possible to store 10000/Batch SIZE or 312 states that can be reused randomly. if number of joints are more and it is required to store more variety of states, then this number can be further increased.

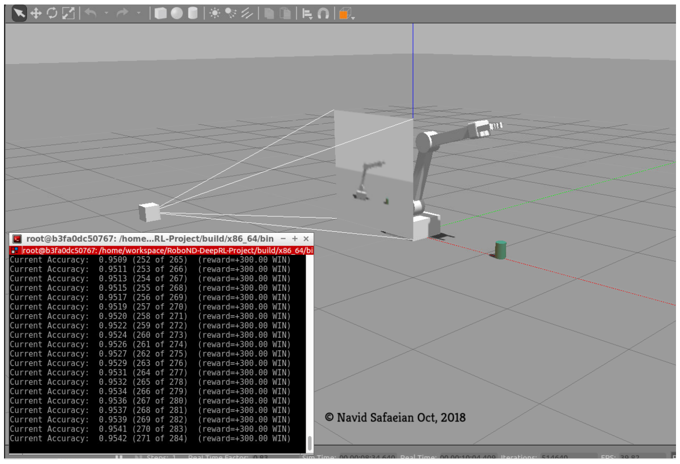
BATCH_SIZE: The number of training examples is equal to the batch size multiply by the number of iterations. The bigger batch size will require the less number of iterations, whereas it will need more memory. 128, 64, and 32 was tested and the Batch_Size of 32 was found the best fit based on computer performance and available memory. This number will also impact REPLAY MEMORY to store states as a positive correlation.

USE_LSTM: Since the network is taking inputs of a sequence of images, LSTM is set to true to keep track of both long and short-term memory. Using LSTM (long Short Term Memory) as part of DQN network will allow training the network by taking into consideration multiple past frames from the camera sensor instead of a single frame. This will much improve the accuracy of learning. In this project LSTM was required and it was enabled to reach the required accuracy.

LSTM_SIZE: The bigger the size leads to the more computing power. Because the data set is not very large, the size of the LSTM was set to 256. Although, 512 was tested but it was causing memory errors.

4 Results

The robotic arm/gripper intuitively started learning better as hyperparameters were tuned and the right reward system was implemented. At almost each iteration of testing, at the beginning before learning the objective, the arm started colliding with the ground, seeming to break multiple times. In inappropriate parameters, DQN network was taking longer time to reach the objective, accuracy was very low, and in many cases system was stopping with memory related errors. After the proper fine tuning of the hyperparameters (section 3), the accuracy for objective 1 reached to more than 95% within less than 300 cycles (Fig. 2 and Fig. 3).

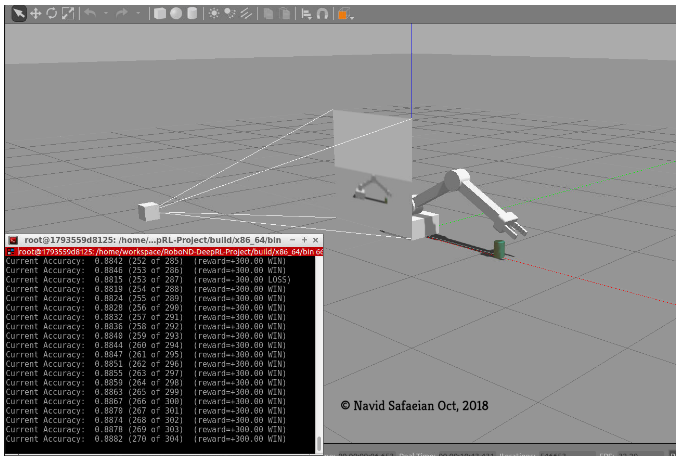

root@b3fa0dc50767: /home...RL-Project/build/x86_64/bin

```
root@b3fa0dc50767: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bi
Current Accuracy:  0.9509 (252 of 265)  (reward=+300.00 WIN)
Current Accuracy:  0.9511 (253 of 266)  (reward=+300.00 WIN)
Current Accuracy:  0.9513 (254 of 267)  (reward=+300.00 WIN)
Current Accuracy:  0.9515 (255 of 268)  (reward=+300.00 WIN)
Current Accuracy:  0.9517 (256 of 269)  (reward=+300.00 WIN)
Current Accuracy:  0.9519 (257 of 270)  (reward=+300.00 WIN)
Current Accuracy:  0.9520 (258 of 271)  (reward=+300.00 WIN)
Current Accuracy:  0.9522 (259 of 272)  (reward=+300.00 WIN)
Current Accuracy:  0.9524 (260 of 273)  (reward=+300.00 WIN)
Current Accuracy:  0.9526 (261 of 274)  (reward=+300.00 WIN)
Current Accuracy:  0.9527 (262 of 275)  (reward=+300.00 WIN)
Current Accuracy:  0.9529 (263 of 276)  (reward=+300.00 WIN)
Current Accuracy:  0.9531 (264 of 277)  (reward=+300.00 WIN)
Current Accuracy:  0.9532 (265 of 278)  (reward=+300.00 WIN)
Current Accuracy:  0.9534 (266 of 279)  (reward=+300.00 WIN)
Current Accuracy:  0.9536 (267 of 280)  (reward=+300.00 WIN)
Current Accuracy:  0.9537 (268 of 281)  (reward=+300.00 WIN)
Current Accuracy:  0.9539 (269 of 282)  (reward=+300.00 WIN)
Current Accuracy:  0.9541 (270 of 283)  (reward=+300.00 WIN)
Current Accuracy:  0.9542 (271 of 284)  (reward=+300.00 WIN)
```

© Navid Safaeian Oct, 2018

Fig. 2. Robot arm touching the object for task 1 after 284 episodes

For objective 2 required more accuracy in its behavior and this was harder to train than objective 1. However, the DQN agent was well learned and performed the objective 2 with more than 88% within less than 300 cycles. For both objectives (tasks), the accuracy of performance by DQN agent were still increasing with more episode cycles. The similar performance was obtained in every run which confirms that system was well tuned and stable.
After setting the final tuned parameters, there was no need to re-tune the parameters after changing the objective from ”arm touching the object” to ”gripper base touching the object”. By changing the Learning Rate to 0.01, the DQN performance was excellent and stable to handle both cases.


root@1793559d8125: /home/...pRL-Project/build/x86_64/bin
root@1793559d8125: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin
© Navid Safaeian Oct, 2018

```
Current Accuracy:  0.8842 (252 of 285)  (reward=+300.00 WIN)
Current Accuracy:  0.8846 (253 of 286)  (reward=+300.00 WIN)
Current Accuracy:  0.8815 (253 of 287)  (reward=-300.00 LOSS)
Current Accuracy:  0.8819 (254 of 288)  (reward=+300.00 WIN)
Current Accuracy:  0.8824 (255 of 289)  (reward=+300.00 WIN)
Current Accuracy:  0.8828 (256 of 290)  (reward=+300.00 WIN)
Current Accuracy:  0.8832 (257 of 291)  (reward=+300.00 WIN)
Current Accuracy:  0.8836 (258 of 292)  (reward=+300.00 WIN)
Current Accuracy:  0.8840 (259 of 293)  (reward=+300.00 WIN)
Current Accuracy:  0.8844 (260 of 294)  (reward=+300.00 WIN)
Current Accuracy:  0.8847 (261 of 295)  (reward=+300.00 WIN)
Current Accuracy:  0.8851 (262 of 296)  (reward=+300.00 WIN)
Current Accuracy:  0.8855 (263 of 297)  (reward=+300.00 WIN)
Current Accuracy:  0.8859 (264 of 298)  (reward=+300.00 WIN)
Current Accuracy:  0.8863 (265 of 299)  (reward=+300.00 WIN)
Current Accuracy:  0.8867 (266 of 300)  (reward=+300.00 WIN)
Current Accuracy:  0.8870 (267 of 301)  (reward=+300.00 WIN)
Current Accuracy:  0.8874 (268 of 302)  (reward=+300.00 WIN)
Current Accuracy:  0.8878 (269 of 303)  (reward=+300.00 WIN)
Current Accuracy:  0.8882 (270 of 304)  (reward=+300.00 WIN)
```

Fig. 3. Robot gripper touching the object for task 1 after 300 episodes

5 Future Work

- There is still room to improve the fin-tuning of the hyper-parameters by comprehensively analyzing their changes and their individually effects on the DQN system by a proper approach like graphing the accuracy progress for every change in parameters and finding the maximum achievable accuracy.
- If there is an access to high memory and computing power, train DQN agent with higher input number for example 512x512 for other DQN applications such as complicated games.
- Furthermore, the plan is to let the project run on the Jetson TX2, and would also be interesting to investigate how well the framework generalizes to controlling other type of robotic arm with enhanced capabilities and more degrees of freedom.