# Deep RL Arm Manipulation

Hytham Tag

**Abstract**—The goal of this project is train a 2 DOF manipulator using reinforcement learning and to optimize it's framework so that the arm can achieve 2 tasks the 1st is to make any part of the arm touch the can with 90% of accuracy at least & the 2nd it to make the gripper touch the can by at least 80%

**Index Terms**—Robot, IEEEtran, Udacity, LaTeX, deep learning.

## 1 Introduction

THE goal of this project is to create DQN agent and define reward system to teach a robotic arm to carry out two primary tasks:

- Have any part of the robot arm to touch the can , with at least 90%
- Have the gripper of the robot arm to touch to touch the can, with at least 80%

The idea here is to make the robot arm try over and over like a child trying to walk, in the beginning the child try to stand up to get a candy but he fails many times until he get close to the best way to stand up right and get the reward (candy), this is the same as the robot arm as the robot arm tries to get close to the target until it touches it and get a reward for touching it whether for task 1 or 2.

### 1.1 Reinforcement Learning

Reinforcement is a class of machine learning where an agent learns how to behave in the environment by performing actions and thereby drawing intuitions and seeing the results where the robot arm is trying to take the best action to get more postive reward

### 1.2 Q-learning

Q-learning is a model-free reinforcement learning algorithm. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model (model-free) of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.

### 1.3 DQN

Deep reinforcement learning is the combination of reinforcement learning (RL) and deep learning . it uses neural network to train the agent.It uses deep learning and reinforcement learning principles in order to create efficient algorithms that can be applied on areas like robotics Deep Learning + Reinforcement Learning = Deep-Q-Networks Instead of memorizing a different Q-value for each combination of pixels on the screen (theres billions!), we use ConvNets to generalize Q-values across similar states.

## 2 Reward Functions

A system of Rewards and penalties is used to train the Robot ARM DQN Agent. It can be found in Armpulgin.cpp. I have chosen Position control and bulit a reward penalty system using it. There Are 3 types of rewards as following:

- Postive Reward for successfully ending the episode
- Negative Reward for unsuccessfully ending the episode
  - reaching more than 100 frames
  - hitting the ground or colliding by it self
  - hitting the can by the arm not gripper (task 2)
- Reward based on distance to objectNegative Reward on time to make the arm reach the object faster
- Negative Reward on small steps (¡ 0.002) to prevent the arm from freezing

an interim reward function was designed. Its purpose is to guide the robot arm towards the target. It measure is the distance dt between gripper and target object at t -1 and t, the difference deta = d(t-1) -d(t) is calculated. In order to obtain a smooth reward function, and then the moving average is calculated as following

- (negativavgGoalDelta = (avgGoalDelta * alpha) + (distDelta * (1.0f - alpha))

If the moving average is less than 0.002 it get negative reward this prevent the robot arm from freezing

- if ( avgGoalDelta ¡ 0.002)
- rewardHistory = REWARD_LOSS;

also there is time penalty the faster the robot reaches the object the less the loss

- rewardHistory = REWARD_MUL * distGoal - time_penalty;

### 2.1 Hyperparameters

These values play a very important role in the training process for the neural network.

- the input width and height where decreased to 64 to lower computational power need also we don't need more than 64 because isn't complex and we don't need many details.

TABLE 1
Hyperparameters

| Parameter | Task 1 | Task 2 |
|---|---|---|
| INPUT_WIDTH | 64 | 64 |
| INPUT_HEIGHT | 64 | 64 |
| OPTIMIZER | Adam | Adam |
| LEARNING_RATE | 0.1 | 0.1 |
| REPLAY_MEMORY | 10000 | 20000 |
| BATCH_SIZE | 256 | 512 |
| USE_LSTM | true | true |
| LSTM_SIZE | 256 | 256 |
| Alpha | 0.9 | 0.4 |

- i have used eventually learning rate of 0.1 as i tried many more than 0.1 didn't achieved the desired accuracy as it reached saturation but smaller than 0.1 toke very long time which is also undesirable for this project..
- I have chosen Adam as optimizer it was working perfectly.
- i have chosen batch size of 256 for task 1 and 512 for task it because task 2 is harder and the larger the batch size the better estimates of gradient descent in neural network.
- i have chosen REPLAY_MEMORY of 10000 for task 1 and 20000 for task 2 as the neural network was able to learn from it's past episode experience through the replay memory as the network agent samples from it randomly int order to try different actions that given it positive reward in past episodes. therefore the replay memory was set to high to keep all the information

### 2.2 Task1

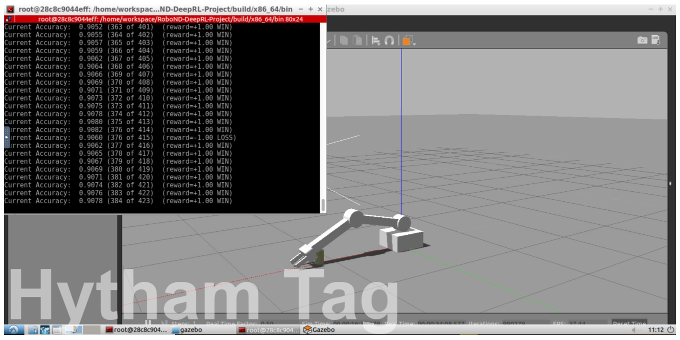
- A reward(WIN *10) is giving If any part of the arm touch
- the object.
- A reward(WIN) is giving if the arm come close to target
- A penalty(LOSS) is giving if any part of the robot touch
- the ground and the episode end.
- A penalty(LOST * distance to goal) is provided if the arm robot go away from the target
- Any collision ends the episode.

### 2.3 Task2

- A reward(WIN *5) is giving If any part of the arm touch
- the object.
- A penalty(LOSS *5) is giving if any part of the robot touch the object.
- A reward(WIN) is giving if the arm come close to target
- A penalty(LOSS) is giving if any part of the robot touch
- the ground and the episode end.
- A penalty(LOST * distance to goal) is provided if the arm robot go away from the target
- A penalty loss if the absolute average is less than 0.02
- A time penalty the longer the robot arm takes
- Any collision ends the episode.

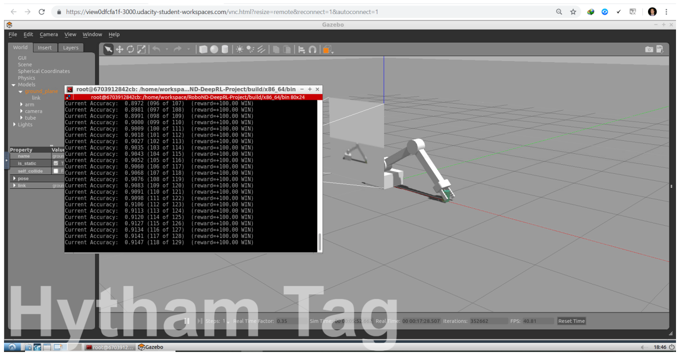
## 3 Results

I was able to achieve more than 90% in both tasks and the largest positive effect on the agents performance was observed when lowering the value of the moving average ALPHA parameter from 0.9 to 0.4. also a video showing what i have done https://www.youtube.com/watch?v=aA83qXZpdi4



```
root@28c8c9044eff: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.9052 (363 of 401)  (reward=+1.00 WIN)
Current Accuracy:  0.9055 (364 of 402)  (reward=+1.00 WIN)
Current Accuracy:  0.9057 (365 of 403)  (reward=+1.00 WIN)
Current Accuracy:  0.9059 (366 of 404)  (reward=+1.00 WIN)
Current Accuracy:  0.9062 (367 of 405)  (reward=+1.00 WIN)
Current Accuracy:  0.9064 (368 of 406)  (reward=+1.00 WIN)
Current Accuracy:  0.9066 (369 of 407)  (reward=+1.00 WIN)
Current Accuracy:  0.9069 (370 of 408)  (reward=+1.00 WIN)
Current Accuracy:  0.9071 (371 of 409)  (reward=+1.00 WIN)
Current Accuracy:  0.9073 (372 of 410)  (reward=+1.00 WIN)
Current Accuracy:  0.9075 (373 of 411)  (reward=+1.00 WIN)
Current Accuracy:  0.9078 (374 of 412)  (reward=+1.00 WIN)
Current Accuracy:  0.9080 (375 of 413)  (reward=+1.00 WIN)
urrent Accuracy:  0.9082 (376 of 414)  (reward=+1.00 WIN)
urrent Accuracy:  0.9060 (376 of 415)  (reward=-1.00 LOSS)
urrent Accuracy:  0.9062 (377 of 416)  (reward=+1.00 WIN)
Current Accuracy:  0.9065 (378 of 417)  (reward=+1.00 WIN)
Current Accuracy:  0.9067 (379 of 418)  (reward=+1.00 WIN)
Current Accuracy:  0.9069 (380 of 419)  (reward=+1.00 WIN)
Current Accuracy:  0.9071 (381 of 420)  (reward=+1.00 WIN)
Current Accuracy:  0.9074 (382 of 421)  (reward=+1.00 WIN)
Current Accuracy:  0.9076 (383 of 422)  (reward=+1.00 WIN)
Current Accuracy:  0.9078 (384 of 423)  (reward=+1.00 WIN)
```

Fig. 1. Task 1.



```
root@6703912842cb: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.8972 (096 of 107)  (reward=+100.00 WIN)
Current Accuracy:  0.8981 (097 of 108)  (reward=+100.00 WIN)
Current Accuracy:  0.8991 (098 of 109)  (reward=+100.00 WIN)
Current Accuracy:  0.9000 (099 of 110)  (reward=+100.00 WIN)
Current Accuracy:  0.9009 (100 of 111)  (reward=+100.00 WIN)
Current Accuracy:  0.9018 (101 of 112)  (reward=+100.00 WIN)
Current Accuracy:  0.9027 (102 of 113)  (reward=+100.00 WIN)
Current Accuracy:  0.9035 (103 of 114)  (reward=+100.00 WIN)
Current Accuracy:  0.9043 (104 of 115)  (reward=+100.00 WIN)
Current Accuracy:  0.9052 (105 of 116)  (reward=+100.00 WIN)
Current Accuracy:  0.9060 (106 of 117)  (reward=+100.00 WIN)
Current Accuracy:  0.9068 (107 of 118)  (reward=+100.00 WIN)
Current Accuracy:  0.9076 (108 of 119)  (reward=+100.00 WIN)
Current Accuracy:  0.9083 (109 of 120)  (reward=+100.00 WIN)
Current Accuracy:  0.9091 (110 of 121)  (reward=+100.00 WIN)
Current Accuracy:  0.9098 (111 of 122)  (reward=+100.00 WIN)
Current Accuracy:  0.9106 (112 of 123)  (reward=+100.00 WIN)
Current Accuracy:  0.9113 (113 of 124)  (reward=+100.00 WIN)
Current Accuracy:  0.9120 (114 of 125)  (reward=+100.00 WIN)
Current Accuracy:  0.9127 (115 of 126)  (reward=+100.00 WIN)
Current Accuracy:  0.9134 (116 of 127)  (reward=+100.00 WIN)
Current Accuracy:  0.9141 (117 of 128)  (reward=+100.00 WIN)
Current Accuracy:  0.9147 (118 of 129)  (reward=+100.00 WIN)
```

Fig. 2. Task 2.

## 4 Future Work

I was able to reach very satisfying result but i think we can use trajectory planning as a way to be like a reference and as the robot arm move on it it get higher reward this is better than the distance between the arm and the object also i want to try it on a real robot arm which i will do ti very soon.