Deep RL Project Arm Manipulation

Noriaki Ibe

Abstract—The forth project in term 2 of the Robotics Software Engineer Nanodegree Program requires students to create a DQN agent and define reward functions to teach a robotic arm to carry out two primary objectives:

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

Index Terms—Robot, IEEEtran, Udacity

1 Introduction

THE Google Deep Mind's project named AlphaGo has been showing the great improving of Artificial Intelligence ("A.I."). Behind the progress, there is the technical improvement in Reinforcement Learning ("RL") and it's combined with neural network named Deep Reinforcement Learning ("Deep RL").

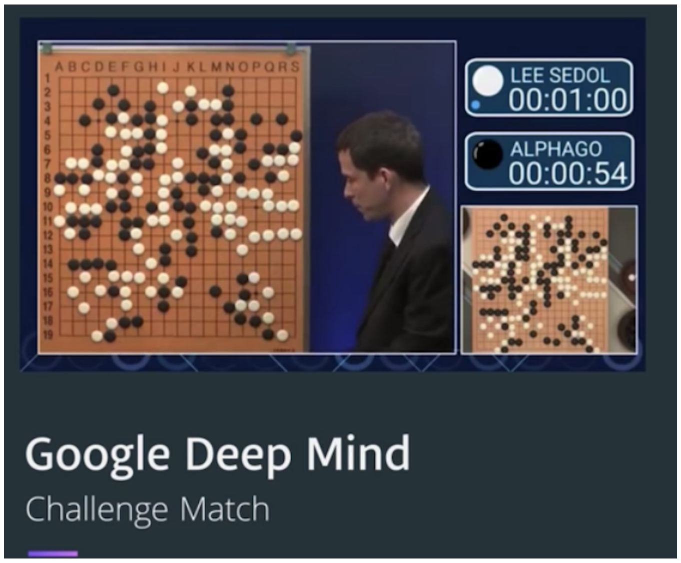


Fig. 1. AlphaGo

Especially, there is a paradigm shift in Deep RL for Robotics. The basic idea is to start with raw sensor input, define a goal for the robot, and let it figure out, through trial and error, the best way to achieve the goal. In this paradigm, perception, internal state, planning, control, and even sensor and measurement uncertainty, are not explicitly defined. All of those traditional steps between observations (the sensory input) and actionable output are learned by a neural network.

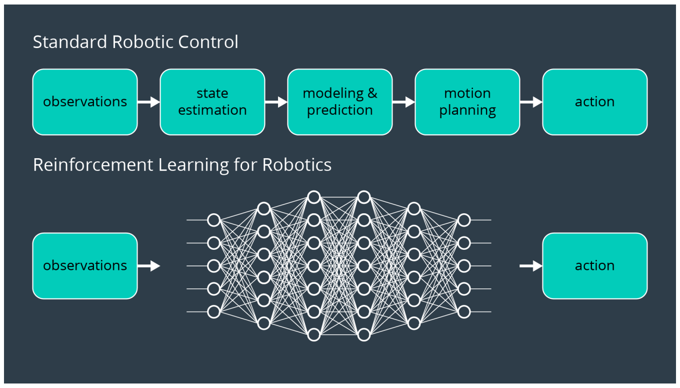


Fig. 2. Deep RL for Robotics

2 Background / Formulation

The Deep RL is constructed with several techniques as follows:

- RL basics
- Q-Learning
- Deep Q-Learning

2.1 RL basics

RL is close to the supervised learning a bit, but there no clear answers such as labels etc. The model sets the Agent and the Agent is taking action to maximize the rewards under the Environment.

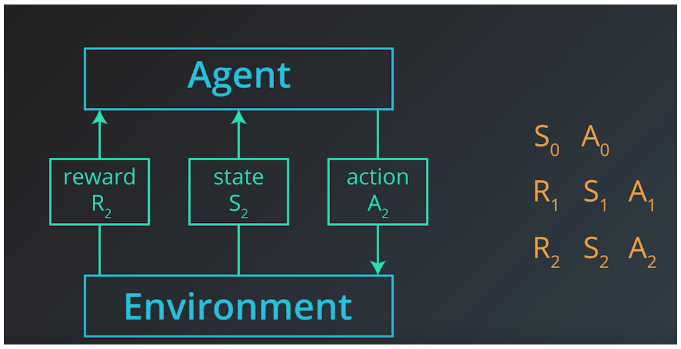


Fig. 3. Reinforcement Learning Model

2.1.1 Markov Decision Process ("MDP")

Markov Decision Process("MDP") defines the model to be solved as follows:

$States : S$
$Model : T(s,a,s')(= P(s'|s,a))$
$Actions : A(s), A$
$Reward : R(s), R(s,a), R(s,a,s')$
$Policy : (s) -> a$

The above, Policy function shall return the action under the certain State. Additionally, the followings are much important to optimize the policy in the model.

- Maximizing the rewards. (Minimizing penalty)
- Applying Discount rate for time

The above can be formulated as follows:
$U(s) = E[{}_t{}^t R(s_t)|, s_0 = s]$
$(s) = argmax_{a s} T(s,a,s)U(s)$
U is the sum of rewards and r means a discount rate and the above can be re-written with out as follows:
$U(s) = R(s) + max_s T(s,a,s)U(s)$
It's called " Bellman equation" and it's model to calculate the rewards from the Environment regardless policy.

2.1.2 Value Iteration

Value Iteration is the approach to optimize the action to maximize rewards With using Bellman equation under the environment. The steps are follows:

1) set rewards
2) calculate the rewards based on the action under the state $T(s,a,s)U(s)T(s,a,s)U(s)$
3) calculate the sum of the rewards (Bellman equation)$U(s) = R(s) + max_s T(s,a,s)U(s)$
4) reach the optimal point

as the above, the model could get the map of rewards under the environment. But, it's less productive if the model must check all of possible actions.

2.1.3 Policy Iteration

Policy Iteration is taking the following steps:

1) set ${}_0$ randomly
2) calculate $U^{(t)}(s)$
3) update ${}_t$ as ${}_{t+1}$ with ${}_{t+1} = argmax_a T(s,a,s)U^t(s)$
4) loop until ${}_{t+1 t}$

So as the above, the model shall find the optimal solution with Value Iteration and Policy Iteration. But as the above formula shows, $T(s,a,s)$ must be known clearly to define the model.

2.2 Q-Learning

Q-learning which is called "Model-free learning" is an another approach to solve problem from Value Iteration and Policy Iteration. Even in case of $T(s,a,s)$ is less known, the model starts as try. It's having an advantage such as we don't need to know the model in advance, but of course it's trade-off that we must learn a lot to know model. $Q(s,a)R(s,a)+maxaE[Q(s,a)]$ There is $E[Q(s,a)]$, which means Expected value instead of $T(s,a,s)$. Additionally, the process of learning with learning rate is as follows:
$Q(s,a) = Q(s,a) + (R(s,a) + max_a E[Q(s,a)]$
Q-learning is called as one of Temporal Difference learning, it's because it is learning from difference between the actual and the expected. Q-learning could know the details of the rewards if each state and action, which is called "Q-table". Then, there are several approaches to find optimal $Q(s,a)$ such as -greedy etc. But, the paradigm shift is coming from Deep neural network.

2.3 Deep Q-Learning

Then, it's time to apply Deep learning concepts into Q-Learning. Back Propagation is the one of main concepts in Deep learning. As the above, Temporal Difference is defined as:
$Q(s,a) = Q(s,a) + (R(s,a) + max_a E[Q(s,a)]$
And Loss could be formulated as follows:
$L = E[1/2 * (R(s,a) + max_{a'} Q_{i-1}(s,a)Q(s,a))^2]$
and divination could be leaded as follows:
$L({}_i) = E[(R(s,a) + maxQ_{i-1}(s,a)Q_i(s,a))Q_i(s,a)]$

2.3.1 Experience Replay

It's common that it's correlated due to time series data and an Experience Replay is the approach to reduce the effect of time series. So it's separated between experience and learning, then it's using random sampling when it's learning.

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\left[\left(r+\gamma\max_{a'}Q(s',a',w) - Q(s,a,w)\right)^2\right]$$

Fig. 4. Experience Replay

As the above, sampling shall be done with Experience(Red) and learn randomly(Blue).

2.3.2 Fixed Target Q-Network

$Q_{i1}(s,a)$ means the expected value and it works as the labels in supervised learning. But it depends on the weight of ${}_{i1}$ mainly and it's updated often. To avoid such frequent update, weight is fixed in sampling batch. When the Loss is calculated , w (red) was fixed and the expected value (blue) shall be stable. Then, learning process is going to go to the next batch after learning and updating w from w^-

$$\mathcal{L}(w) = \mathbb{E}_{s,a,r,s'\sim\mathcal{D}}\left[\left(r+\gamma\max_{a'}Q(s',a',w^-) - Q(s,a,w)\right)^2\right]$$

Fig. 5. Fixed Target Q-Network

3 ENVIRONMENT, CONFIGRATION AND TUNING

3.1 Environment

This project is based on the Nvidia open source project "jetson-reinforcement" developed by "Dustin Franklin". In the project "repo", the student can locate the gazebo-arm.world file in /gazebo/. There are three main components to this gazebo file, which define the environment. To launch the project for the first time, run the following in the terminal of the desktop gui:

```
$ cd /home/workspace/RoboND-DeepRL-Project/
    build/x86_64/bin
$ ./gazebo-arm.sh
```

- The robotic arm with a gripper attached to it.
- A camera sensor, to capture images to feed into the DQN.
- A cylindrical object or prop.

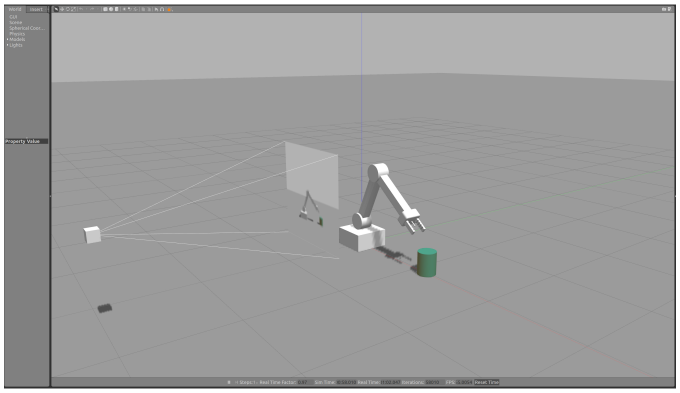


Fig. 6. Project Environment

The API provides an interface to the Python code written with PyTorch, but the wrappers use Pythons low-level C to pass memory objects between the users application and Torch without extra copies. By using a compiled language (C/C++) instead of an interpreted one, performance is improved, and speeded up even more when GPU acceleration is leveraged.

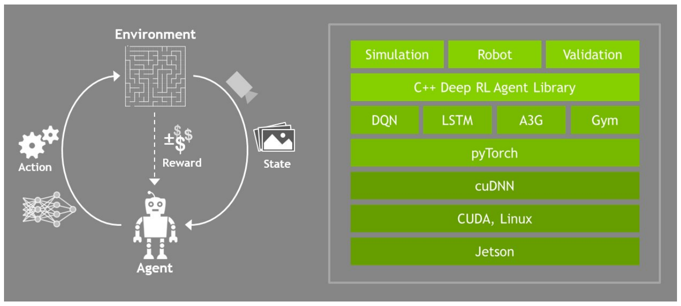


Fig. 7. API stack for Deep RL (from Nvidia repo)

3.2 Configuration and Tuning

The project model is built on a digital workspace provided from Udacity. The ArmPlugin.cpp file has specific TODOs or tasks listed out for the Project. The students have to complete each of those tasks in the following order. To make it clear, the ArmPlugin.cpp has number of tasks listed with specific TODOs.

3.2.1 1. Subscribe to camera and collision topics.

The project create the subscribers in the ArmPlugin::Load() function based on the following specifications:

For the camera node -

- Node - cameraNode
- Topic Name - /gazebo/arm_world/camera/link/camera/image
- Callback Function - ArmPlugin::onCameraMsg (this should be a reference parameter)
- Class Instance - refer to the same class, using the this pointer or keyword.

```
cameraSub = cameraNode->Subscribe("/gazebo/
    arm_world/camera/link/camera/image", &
    ArmPlugin::onCameraMsg, this);
```

For the contact/collision node -

- Node - collisionNode
- Topic Name - /gazebo/arm_world/tube/tube_/my_contact
- Callback Function - ArmPlugin::onCollisionMsg (this should be a reference parameter)
- Class Instance - refer to the same class, using the this pointer or keyword.

```
collisionSub = collisionNode->Subscribe("/
    gazebo/arm_world/tube/tube_link/
    my_contact", &ArmPlugin::onCollisionMsg,
      this);
```

3.2.2 2. Create the DQN Agent

The project referred to the constructor in "project_folder/c/dqnAgent.cpp" for the same.

3.2.3 3. Velocity or position based control of arm joints

In ArmPlugin::updateAgent(), there are two existing approaches to control the joint movements and there is discussion about it on slack channel on udacity robotics too, and it suggests to use position based control.

3.2.4 4. Reward for robot gripper hitting the ground

In Gazebos API, there is a function called GetBoundingBox() which returns the minimum and maximum values of a box that defines that particular object/model corresponding to the x, y, and z axes.

Using the above, the project can check if the gripper is hitting the ground or not, and assign an appropriate reward.

3.2.5 5. Issue an interim reward based on the distance to the object

In ArmPlugin.cpp a function called BoxDistance() calculates the distance between two bounding boxes. Using this function, calculate the distance between the arm and the object.

3.2.6 6. Issue a reward based on collision between the arm and the object.

In the callback function onCollisionMsg, you can check for certain collisions. Specifically, you will define a check condition to compare if particular links of the arm with their defined collision elements are colliding with the COLLISION_ITEM or not. So each bool collisionCheck shall be use for each objects.

```
/Have any part of the robot arm touch the
    object of interest, with at least a 90%
    accuracy.
//[90%]
bool collisionCheck = ((strcmp(contacts->
    contact(i).collision1().c_str(),
   COLLISION_ITEM) == 0) || (strcmp(
    contacts->contact(i).collision2().c_str
    (), COLLISION_ARM) == 0) || (strcmp(
    contacts->contact(i).collision2().c_str
    (), COLLISION_POINT) == 0) || (strcmp(
    contacts->contact(i).collision2().c_str
    (), COLLISION_MIDDLE) == 0)) ? true :
    false;

/Have only the gripper base of the robot arm
     touch the object, with at least a 80%
    accuracy.
//[80%]
bool collisionCheck = (strcmp(contacts->
    contact(i).collision2().c_str(),
   COLLISION_POINT) == 0) ? true : false;
```

3.2.7 7. Tuning the hyperparameters

At first, the project focused on to pass the 1st task.

- Have any part of the robot arm touch the object of interest, with at least a 90% accuracy.

After, the model perform well for task 1 the project set the task with changing the object as mentioned 6. Issue a reward based on collision between the arm and the object. But the model didn't work well so that some tuning was needed.

So there are two sets of parameters as below:

```
#define EPS_START 0.8f // 80% 0.8f : 90% 0.9
    f
#define EPS_END 0.01f // 80% 0.01f : 90%
    0.05f
#define EPS_DECAY 300 // 80% 300 : 90% 200
#define LEARNING_RATE 0.15f // 80% 0.15f :
   90% 0.2f
#define BATCH_SIZE 32 // 80% 32 : 90% 16
```

"PyTorch tutorials" explain that select_action as follows:

- - will select an action accordingly to an epsilon greedy policy. Simply put, well sometimes use our model for choosing the action, and sometimes well just sample one uniformly. The probability of choosing a random action will start at EPS_START and will decay exponentially towards EPS_END. EPS_DECAY controls the rate of the decay.

Additionally, the model with param 90% could perform around 60 70% accuracy as later shown. It means the model could learn until certain points so that tuning was focusing of learning slowly rather than the 1st task.

After tuning as the above, the project run following.

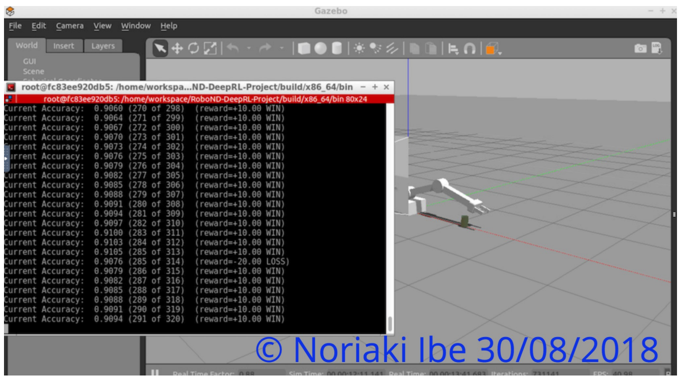
```
$ cd build
$ make
$ cd x86_64/bin
$ ./gazebo-arm.sh
```

4 Results

This section shows results for each tasks.

4.1 Task 1:

The model have any part of the robot arm touch the object of interest, with at least a 90 accuracy. It looks nice and works well.

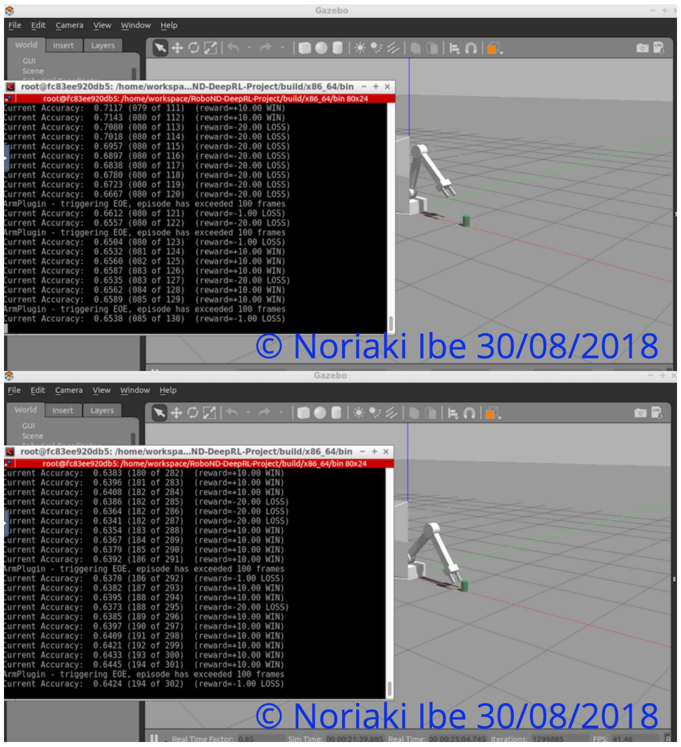


```
root@fc83ee920db5: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@fc83ee920db5: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.9060 (270 of 298)  (reward=+10.00 WIN)
Current Accuracy:  0.9064 (271 of 299)  (reward=+10.00 WIN)
Current Accuracy:  0.9067 (272 of 300)  (reward=+10.00 WIN)
Current Accuracy:  0.9070 (273 of 301)  (reward=+10.00 WIN)
Current Accuracy:  0.9073 (274 of 302)  (reward=+10.00 WIN)
Current Accuracy:  0.9076 (275 of 303)  (reward=+10.00 WIN)
Current Accuracy:  0.9079 (276 of 304)  (reward=+10.00 WIN)
Current Accuracy:  0.9082 (277 of 305)  (reward=+10.00 WIN)
Current Accuracy:  0.9085 (278 of 306)  (reward=+10.00 WIN)
Current Accuracy:  0.9088 (279 of 307)  (reward=+10.00 WIN)
Current Accuracy:  0.9091 (280 of 308)  (reward=+10.00 WIN)
Current Accuracy:  0.9094 (281 of 309)  (reward=+10.00 WIN)
Current Accuracy:  0.9097 (282 of 310)  (reward=+10.00 WIN)
Current Accuracy:  0.9100 (283 of 311)  (reward=+10.00 WIN)
Current Accuracy:  0.9103 (284 of 312)  (reward=+10.00 WIN)
Current Accuracy:  0.9105 (285 of 313)  (reward=+10.00 WIN)
Current Accuracy:  0.9076 (285 of 314)  (reward=-20.00 LOSS)
Current Accuracy:  0.9079 (286 of 315)  (reward=+10.00 WIN)
Current Accuracy:  0.9082 (287 of 316)  (reward=+10.00 WIN)
Current Accuracy:  0.9085 (288 of 317)  (reward=+10.00 WIN)
Current Accuracy:  0.9088 (289 of 318)  (reward=+10.00 WIN)
Current Accuracy:  0.9091 (290 of 319)  (reward=+10.00 WIN)
Current Accuracy:  0.9094 (291 of 320)  (reward=+10.00 WIN)
```

Fig. 8. Task 1

4.2 Task 2 with Param Task 1

The model had only the gripper base of the robot arm touch the object, with at least a 70% accuracy with Param Task 1 and accuracy decreased even the model learned longer period.



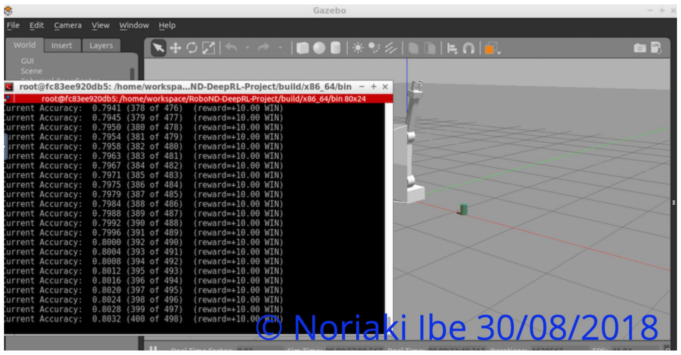
```
root@fc83ee920db5: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@fc83ee920db5: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.7117 (079 of 111)  (reward=+10.00 WIN)
Current Accuracy:  0.7143 (080 of 112)  (reward=+10.00 WIN)
Current Accuracy:  0.7080 (080 of 113)  (reward=-20.00 LOSS)
Current Accuracy:  0.7018 (080 of 114)  (reward=-20.00 LOSS)
Current Accuracy:  0.6957 (080 of 115)  (reward=-20.00 LOSS)
Current Accuracy:  0.6897 (080 of 116)  (reward=-20.00 LOSS)
Current Accuracy:  0.6838 (080 of 117)  (reward=-20.00 LOSS)
Current Accuracy:  0.6780 (080 of 118)  (reward=-20.00 LOSS)
Current Accuracy:  0.6723 (080 of 119)  (reward=-20.00 LOSS)
Current Accuracy:  0.6667 (080 of 120)  (reward=-20.00 LOSS)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.6612 (080 of 121)  (reward=-1.00 LOSS)
Current Accuracy:  0.6557 (080 of 122)  (reward=-20.00 LOSS)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.6504 (080 of 123)  (reward=-1.00 LOSS)
Current Accuracy:  0.6532 (081 of 124)  (reward=+10.00 WIN)
Current Accuracy:  0.6560 (082 of 125)  (reward=+10.00 WIN)
Current Accuracy:  0.6587 (083 of 126)  (reward=+10.00 WIN)
Current Accuracy:  0.6535 (083 of 127)  (reward=-20.00 LOSS)
Current Accuracy:  0.6562 (084 of 128)  (reward=+10.00 WIN)
Current Accuracy:  0.6589 (085 of 129)  (reward=+10.00 WIN)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.6538 (085 of 130)  (reward=-1.00 LOSS)
```

```
root@fc83ee920db5: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@fc83ee920db5: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.6383 (180 of 282)  (reward=+10.00 WIN)
Current Accuracy:  0.6396 (181 of 283)  (reward=+10.00 WIN)
Current Accuracy:  0.6408 (182 of 284)  (reward=+10.00 WIN)
Current Accuracy:  0.6386 (182 of 285)  (reward=-20.00 LOSS)
Current Accuracy:  0.6364 (182 of 286)  (reward=-20.00 LOSS)
Current Accuracy:  0.6341 (182 of 287)  (reward=-20.00 LOSS)
Current Accuracy:  0.6354 (183 of 288)  (reward=+10.00 WIN)
Current Accuracy:  0.6367 (184 of 289)  (reward=+10.00 WIN)
Current Accuracy:  0.6379 (185 of 290)  (reward=+10.00 WIN)
Current Accuracy:  0.6392 (186 of 291)  (reward=+10.00 WIN)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.6370 (186 of 292)  (reward=-1.00 LOSS)
Current Accuracy:  0.6382 (187 of 293)  (reward=+10.00 WIN)
Current Accuracy:  0.6395 (188 of 294)  (reward=+10.00 WIN)
Current Accuracy:  0.6373 (188 of 295)  (reward=-20.00 LOSS)
Current Accuracy:  0.6385 (189 of 296)  (reward=+10.00 WIN)
Current Accuracy:  0.6397 (190 of 297)  (reward=+10.00 WIN)
Current Accuracy:  0.6409 (191 of 298)  (reward=+10.00 WIN)
Current Accuracy:  0.6421 (192 of 299)  (reward=+10.00 WIN)
Current Accuracy:  0.6433 (193 of 300)  (reward=+10.00 WIN)
Current Accuracy:  0.6445 (194 of 301)  (reward=+10.00 WIN)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.6424 (194 of 302)  (reward=-1.00 LOSS)
```

Fig. 9. Task 2 with Param Task 1

4.3 Task 2 with Tuning

The model had only the gripper base of the robot arm touch the object, with at least a 80% accuracy with Tuning. It looks fine and performed well.



```
root@fc83ee920db5: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
urrent Accuracy:  0.7941 (378 of 476)  (reward=+10.00 WIN)
urrent Accuracy:  0.7945 (379 of 477)  (reward=+10.00 WIN)
urrent Accuracy:  0.7950 (380 of 478)  (reward=+10.00 WIN)
urrent Accuracy:  0.7954 (381 of 479)  (reward=+10.00 WIN)
urrent Accuracy:  0.7958 (382 of 480)  (reward=+10.00 WIN)
urrent Accuracy:  0.7963 (383 of 481)  (reward=+10.00 WIN)
urrent Accuracy:  0.7967 (384 of 482)  (reward=+10.00 WIN)
urrent Accuracy:  0.7971 (385 of 483)  (reward=+10.00 WIN)
urrent Accuracy:  0.7975 (386 of 484)  (reward=+10.00 WIN)
urrent Accuracy:  0.7979 (387 of 485)  (reward=+10.00 WIN)
urrent Accuracy:  0.7984 (388 of 486)  (reward=+10.00 WIN)
urrent Accuracy:  0.7988 (389 of 487)  (reward=+10.00 WIN)
urrent Accuracy:  0.7992 (390 of 488)  (reward=+10.00 WIN)
urrent Accuracy:  0.7996 (391 of 489)  (reward=+10.00 WIN)
urrent Accuracy:  0.8000 (392 of 490)  (reward=+10.00 WIN)
urrent Accuracy:  0.8004 (393 of 491)  (reward=+10.00 WIN)
urrent Accuracy:  0.8008 (394 of 492)  (reward=+10.00 WIN)
urrent Accuracy:  0.8012 (395 of 493)  (reward=+10.00 WIN)
urrent Accuracy:  0.8016 (396 of 494)  (reward=+10.00 WIN)
urrent Accuracy:  0.8020 (397 of 495)  (reward=+10.00 WIN)
urrent Accuracy:  0.8024 (398 of 496)  (reward=+10.00 WIN)
urrent Accuracy:  0.8028 (399 of 497)  (reward=+10.00 WIN)
urrent Accuracy:  0.8032 (400 of 498)  (reward=+10.00 WIN)
```

Fig. 10. Task 2 with Tuning

5 Discussion

There was difficulty in Tuning a bit. But it would be great example that Parameters must be adjusted when the target (or even environment) was different.

6 Conclusion / Future work

The project could perform at least as the requirements stated on Background / Formulation. Additionally, it would be great time to learn again about Reinforcement Learning through Deep Q-Learning. As mentioned the above, the project avoid applying Velocity based control and results were fine fortunately. But the model could face any problem in case of difficulty tasks rather than the project. So, it would be the next step to use Jetson-TX2 as real-world to grip something, which shall need Velocity based control, too.