Deep Reinforcement Learning Project
Mohamed Ewis

1 Introduction:

In this project, a Deep Q-learning Network (DQN) is created and it aims to train a robotic arm to perform certain actions and meet certain objectives.
This project mainly leverages an existing DQN that gets instantiated with specific parameters to run the Robotic Arm. For Deep Reinforcement Learning to occur, reward functions and hyperparameters must be defined.
To test the capabilities of DQN, two objectives were established:

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.
2. 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:

In this project, there were two objectives. Based on the objective, the reward function was modified to suit the objective's need.

The two objectives were as follows:

- **Objective 1:** This objective was to make any point of the robotic arm touch the object of interest. In order to complete this objective, the approach taken was to use “velocity control” instead of position. To control velocity, a simple check was performed to observe whether the action was even or odd. Depending on the action check, actionVelDelta was either added or subtracted.
  The robotic arm must never collide with the ground, therefore, a REWARD_LOSS of 1 * 20 will be issued upon ground collision. However, if the arm collided with the target object, it would issue a REWARD_WIN of 1 * 10, meeting the objective and resulting in a “win” for that run. Another criteria was added, to encourage the arm to accomplish its objective before the episode was over (frame span). If the run exceeded the maximum number of frames per episode (maxEpisodeLength), which was set to 100, a REWARD_LOSS of 1 * 1000 would be issued. This would result in a “loss” for that run. Since it’s using velocity control to complete the first objective, the reward function setup for the process in between to control the robotic arm’s movement will defer to the one that will be used for the second objective. As part of the generic solution, the average delta (avgGoalDelta) was calculated just as stated in the tasks lesson:
  avgGoalDelta = (avgGoalDelta * alpha) + (distDelta * (1.0f - alpha));
  The reward was issued as follows:

```
if (avgGoalDelta > 0)
{
        if(distGoal > 0.001f){
                rewardHistory = REWARD_WIN / distGoal;
        }
        else if (distGoal < 0.001f || distGoal == 0.0f){
                rewardHistory = REWARD_WIN / 2000.0f;
        }
}
```

This approach encourages the arm to quickly reach the goal by increasing the reward (inversely proportional) as it gets close to the goal. However, once this arm gets to a certain point (close enough), the reward stops increasing and only issues a small reward, which should encourage the arm to slow down as it reaches the target.

- **Objective 2:** The goal of this objective was to make the robotic arm touch the object, but only with its gripper part. For this objective, "position control" was used instead of velocity. To control position, a simple check was performed to observe whether the action was even or odd. Depending on the action check, actionJointDelta was either added or subtracted. The robotic arm must never collide with the ground, therefore, a REWARD_LOSS of 1 * 20 will be issued upon ground collision. However, if the gripper collided with the target object, it would issue a REWARD_WIN of 1 * 100, meeting the objective and resulting in a “win” for that run. For this objective, the emphasis was to get it to touch with the gripper (solely), which is why the reward is higher than for the previous objective. Also, another limitation was added, to encourage the arm to accomplish its objective before the episode was over (frame span). If the run exceeded the maximum number of frames per episode (maxEpisodeLength), which was set to 100, a REWARD_LOSS of 1 would be issued. This would result in a “loss” for that run. Although the objective is still to collide with the target within one run/episode, some emphasis was taken away from this limitation given that velocity control has been replaced by position control.

  However, since position control will be used to achieve objective 2, the reward function setup for the process will be different this time around. As part of the generic solution, the average delta (avgGoalDelta) was calculated just as stated in the tasks lesson (the same as for objective 1):

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

  However, this time around instead of using the distance from the goal (distGoal) to drive the rewards, the smoothed moving average will be used, as follows:

```
if (avgGoalDelta > 0) {
        if(distGoal > 0.0f){
                rewardHistory = REWARD_WIN * avgGoalDelta;
        }
        else if (distGoal == 0.0f){
                rewardHistory = REWARD_WIN * 10.0f;
        }
}
else {
        rewardHistory = REWARD_LOSS * distGoal;
}
```

  This approach focuses on the movement, encouraging the robotic arm to move more. The more it moves towards the goal, the more rewards it will get (proportional to avgGoalDelta). However, once the arm at 0 distance away from the it would provide an extra reward. Furthermore, if not moving towards the goal, it would issue a REWARD_LOSS proportional to the distance to the goal (distGoal). At least that’s the theory behind this approach.

3 Hyperparameters:

For both objectives, tuning the hyperparameters was an iterative and trial and error process.

- **Objective 1:**

Hyperparameter	Value	Comments
INPUT_WIDTH	64	Reduced width.
INPUT_HEIGHT	64	Reduced height.
OPTIMIZER	"Adam"	Used Adam as it is a better optimizer than "RMSProp".
LEARNING_RATE	0.2f	Trial and error to get the best result.
REPLAY_MEMORY	10000	Did not change this parameter.
BATCH_SIZE	16	More learning was required and the memory permitted this batch size.
USE_LSTM	True	Improved learning from past experiences.
LSTM_SIZE	256	Trial and error in iteration to get the optimum value.

- **Objective 2:**

Hyperparameter	Value	Comments
INPUT_WIDTH	64	Reduced width.
INPUT_HEIGHT	64	Reduced height.
OPTIMIZER	"Adam"	Used Adam as it is a better optimizer than "RMSProp".
LEARNING_RATE	0.1f	Requires a slower learning as it is a much more specific objective.
REPLAY_MEMORY	10000	Did not change this parameter.
BATCH_SIZE	32	Deeper learning was required than objective 1 and the memory permitted this batch size.
USE_LSTM	True	Improved learning from past experiences.
LSTM_SIZE	256	Trial and error in iteration to get the optimum value.

Also the following values were changed in the DQN API for objective 2. These values were changed in order to facilitate a quicker exploitation and reduce the exploration in order to achieve a faster convergence.

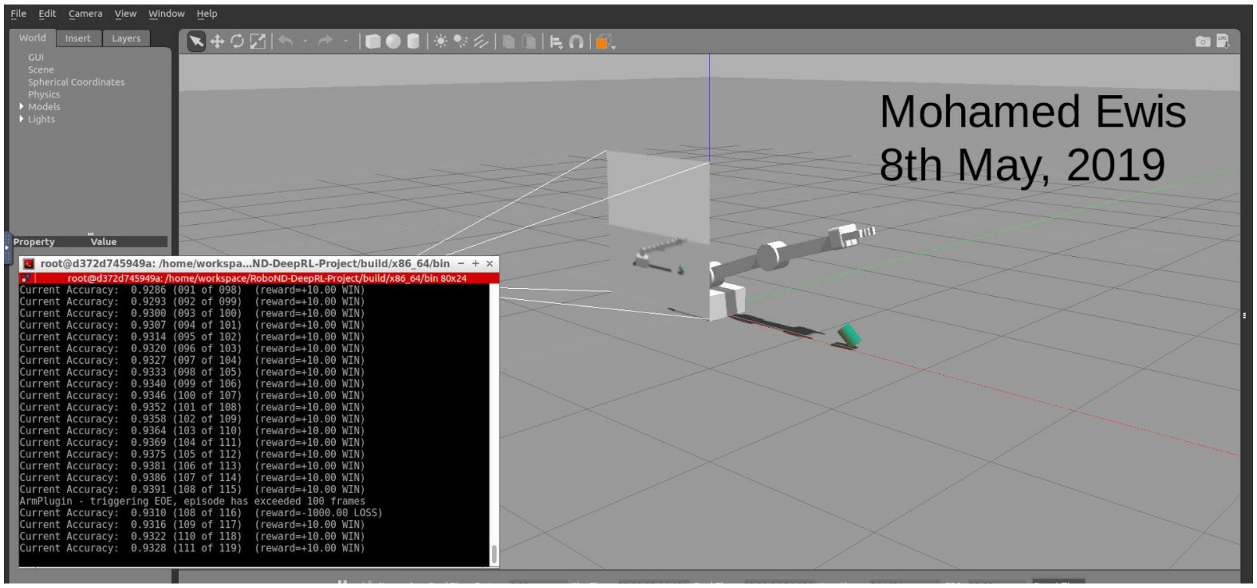
Parameter	Value
EPS_START	0.8f
EPS_END	0.01f
EPS_DECAY	300

4 Results:

- Objective 1: This objective, was relatively simple to train, the robotic arm intuitively started learning better as hyperparameters were tuned and the reward values were increased. The arm learned relatively quickly. It was noted that it kept trying to explore other ways to tackle the problem, obtaining multiple solution approaches along the way. Ultimately, the latest version of the code for objective 1 managed to get to 95% accuracy almost right after 100 runs.

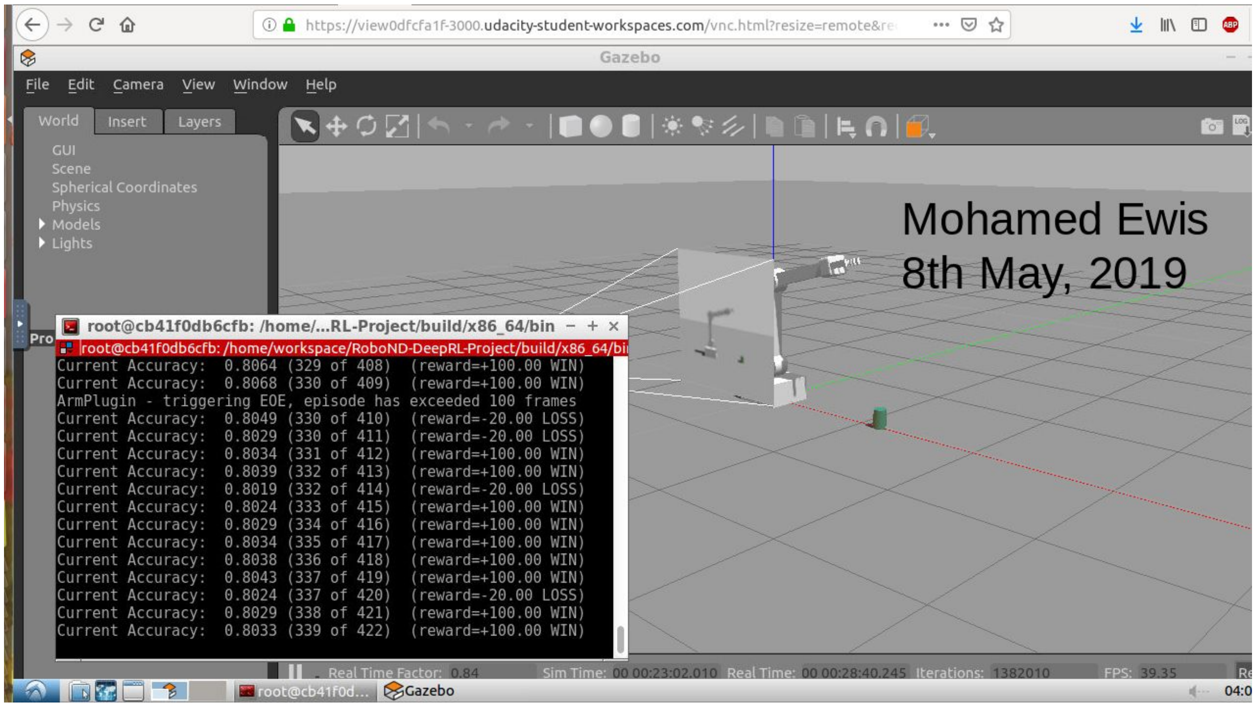
- Objective 2: This objective required more accuracy as the end objective was more nuanced and sophisticated and there was little room for error. After iterating and testing many times, by tuning hyperparameters, it was noticed that the agent was trying to explore too much. However, there are so many approaches that can be taken to touch the target with just one point. For that reason, the DQN API Settings were changed to decrease exploration slightly compared to objective 1. This helped on teaching the robotic arm agent to perform a solution approach quicker. Also, the fact that the learning rate was decreased and the batch size increased, seemed to have helped the agent to train and learn deeper than objective 1. However, as with objective 1, the agent was also displayed hitting the ground hard enough to sometimes break, at least in the initial stages of the iteration. This a problem that could be catastrophic for a real robot (which normally costs a lot of money to make). This should be explored further. Nonetheless, for the purpose of this project, the robotic arm exceeded the minimum accuracy of 80% reaching up to 85% accuracy after almost 160 runs. The second objective was harder to reach, as training the robotic arm to have finesse and be more precise, required finer tuning of hyperparameters and, as mentioned previously, DQN API settings as well. Nonetheless, both agents were optimally trained to meet the objectives presented, with greater than expected accuracy for the second objective once it was finally tuned properly.

Figure 1 and 2 depicts that both objectives were satisfied. For the first objective, an accuracy of 95% was achieved and for the 2nd objective, an accuracy of 85% was achieved.



```
root@d372d745949a: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@d372d745949a: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.9286 (091 of 098)  (reward=+10.00 WIN)
Current Accuracy:  0.9293 (092 of 099)  (reward=+10.00 WIN)
Current Accuracy:  0.9300 (093 of 100)  (reward=+10.00 WIN)
Current Accuracy:  0.9307 (094 of 101)  (reward=+10.00 WIN)
Current Accuracy:  0.9314 (095 of 102)  (reward=+10.00 WIN)
Current Accuracy:  0.9320 (096 of 103)  (reward=+10.00 WIN)
Current Accuracy:  0.9327 (097 of 104)  (reward=+10.00 WIN)
Current Accuracy:  0.9333 (098 of 105)  (reward=+10.00 WIN)
Current Accuracy:  0.9340 (099 of 106)  (reward=+10.00 WIN)
Current Accuracy:  0.9346 (100 of 107)  (reward=+10.00 WIN)
Current Accuracy:  0.9352 (101 of 108)  (reward=+10.00 WIN)
Current Accuracy:  0.9358 (102 of 109)  (reward=+10.00 WIN)
Current Accuracy:  0.9364 (103 of 110)  (reward=+10.00 WIN)
Current Accuracy:  0.9369 (104 of 111)  (reward=+10.00 WIN)
Current Accuracy:  0.9375 (105 of 112)  (reward=+10.00 WIN)
Current Accuracy:  0.9381 (106 of 113)  (reward=+10.00 WIN)
Current Accuracy:  0.9386 (107 of 114)  (reward=+10.00 WIN)
Current Accuracy:  0.9391 (108 of 115)  (reward=+10.00 WIN)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.9310 (108 of 116)  (reward=-1000.00 LOSS)
Current Accuracy:  0.9316 (109 of 117)  (reward=+10.00 WIN)
Current Accuracy:  0.9322 (110 of 118)  (reward=+10.00 WIN)
Current Accuracy:  0.9328 (111 of 119)  (reward=+10.00 WIN)
```

Figure 1: Objective 1 result



```
root@cb41f0db6cfb: /home/...RL-Project/build/x86_64/bin
root@cb41f0db6cfb: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bi
Current Accuracy:  0.8064 (329 of 408)  (reward=+100.00 WIN)
Current Accuracy:  0.8068 (330 of 409)  (reward=+100.00 WIN)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy:  0.8049 (330 of 410)  (reward=-20.00 LOSS)
Current Accuracy:  0.8029 (330 of 411)  (reward=-20.00 LOSS)
Current Accuracy:  0.8034 (331 of 412)  (reward=+100.00 WIN)
Current Accuracy:  0.8039 (332 of 413)  (reward=+100.00 WIN)
Current Accuracy:  0.8019 (332 of 414)  (reward=-20.00 LOSS)
Current Accuracy:  0.8024 (333 of 415)  (reward=+100.00 WIN)
Current Accuracy:  0.8029 (334 of 416)  (reward=+100.00 WIN)
Current Accuracy:  0.8034 (335 of 417)  (reward=+100.00 WIN)
Current Accuracy:  0.8038 (336 of 418)  (reward=+100.00 WIN)
Current Accuracy:  0.8043 (337 of 419)  (reward=+100.00 WIN)
Current Accuracy:  0.8024 (337 of 420)  (reward=-20.00 LOSS)
Current Accuracy:  0.8029 (338 of 421)  (reward=+100.00 WIN)
Current Accuracy:  0.8033 (339 of 422)  (reward=+100.00 WIN)
```

Figure 2: Objective 2 result

5 Future work:

For future work, it would be great to improve on the hyperparameter tuning for the first objective to achieve almost 100% accuracy results. Obtaining a higher accuracy and faster might be another improvement to investigate, as in real-life, having to wait long periods for robots to learn how to perform tasks might be slightly more inconvenient than in a simulation. Furthermore, cannot emphasize enough the ground collision issue that must be mitigated for this to be a useful real-life solution. It must never collide with the ground, or at least not with that amount of force as that might cause damage to the arm. Perhaps a constraint can be developed and implemented to shut down the arm right before colliding with the ground.