RoboND – Deep RL Arm Manipulation Project

For this project, the goal is to create a DQN agent and define reward functions to teach a robotic arm to carry out two primary objectives:

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 a 80% accuracy for a minimum of 100 runs.

Reward functions:

The reward values used to accomplish each objective are set as:

REWARD_WIN = 1

REWARD_LOSS = -1

For both objectives joint position control was implemented to control the movement of each joint for the robotic arm. It is believed that the problem of teaching the robot to touch the object is a position problem, where the position of the robot is more important than its speed. Controlling the position of each point in order to have the robot move towards the object and touch it was required.

First objective rewards:

1- Assign a reward for the robot gripper hitting the ground.

The robot arm must never collide with the ground, therefore, a REWARD_LOSS of -1*20 will be issued upon ground collision.

2- Issue an interim reward based on the distance to the object.

An Interim reward is issued to encourage the robot arm to accomplish its objective before the episode terminates.

The following formula was used to compute the smoothed moving average of the delta of the distance to the goal:

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

where,

distDelta = lastGoalDistance – distGoal

alpha was set as 0.2.

The above formula is used such that when the avgGoalDelta value is positive, it means the robot is moving closer towards the object, and vice versa.

When the robot is moving towards the goal a REWARD_WIN of 1 is issue, else, a REWARD_LOSS of -1 * distGoal is issued. That is to encourage the robot to move towards the object during each episode.

Also, since it is required for the robot to keep moving during each episode a REWARD_LOSS of -1 is added to the total rewards collected during the episode every time the robot is found standing still.

3- Issue a reward when the robot exceeds 100 frames without indecision

A REWARD_LOSS of -1*100 is issued when 100 frames are exceeded and no action is taken from the robot.

4- Issue a reward based on collision between the arm and the object.

A REWARD_WIN of 1*10 will be issued when any part of the robot arm collides with the object as that results in the first objective being met.

Second objective rewards:

The same reward functions where applied to achieve the second objective, except for the last reward function which was modified as the following:

4- Issue a reward based on collision between the arm and the object.

For this object since the collision is required between the gripper base and the object, a REWARD_WIN of 1*100 is issued so that the robot knows for certain that it touching the object with the base gripper only is the goal.

If the robot's gripper middle touches the object a small REWARD_WIN of 1*10 is issued to ensure that it learns that action is almost good.

Else if any other part of the arm touches the object, a REWARD_LOSS of -15 is issued.

Hyperparameters:

First objective:

The following parameters were used to achieve the first objective:

```
ArmPlugin::ArmPlugin()
ArmPlugin::Load('arm')
PropPlugin::Load('tube')
[deepRL]  use_cuda:       True
[deepRL]  use_lstm:       1
[deepRL]  lstm_size:      256
[deepRL]  input_width:    128
[deepRL]  input_height:   128
[deepRL]  input_channels: 3
[deepRL]  num_actions:    6
[deepRL]  optimizer:      RMSprop
[deepRL]  learning_rate:  0.01
[deepRL]  replay_memory:  10000
[deepRL]  batch_size:     64
[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
```


- Environment width and height were reduced to 128 to reduce computational power, save up memory and increase the computational speeds.
- “RMSprop” Optimizer was used as it was seen to be recommended by the forums to speed up learning.
- Decreasing the LEARNING_RATE to 0.01 helped with reaching 90% accuracy faster.
- REPLAY_MEMORY stayed the same as the lessons suggested. It is set to a high value to keep information of all the episodes it will perform.
- BATCH_SIZE set 64 because increasing the batch size gives better estimates of gradient decent in neural networks it was made sure to not increase it severely to reduce computational overhead.
- USE_LSTM set to true to improve learning from past experiences.
- LSTM_SIZE was set to 256 as 32 didn’t seem to be working.

Second objective:

The following parameters were used to achieve the first objective:

```
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:  0.01
[deepRL]  replay_memory:  10000
[deepRL]  batch_size:     512
[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
```

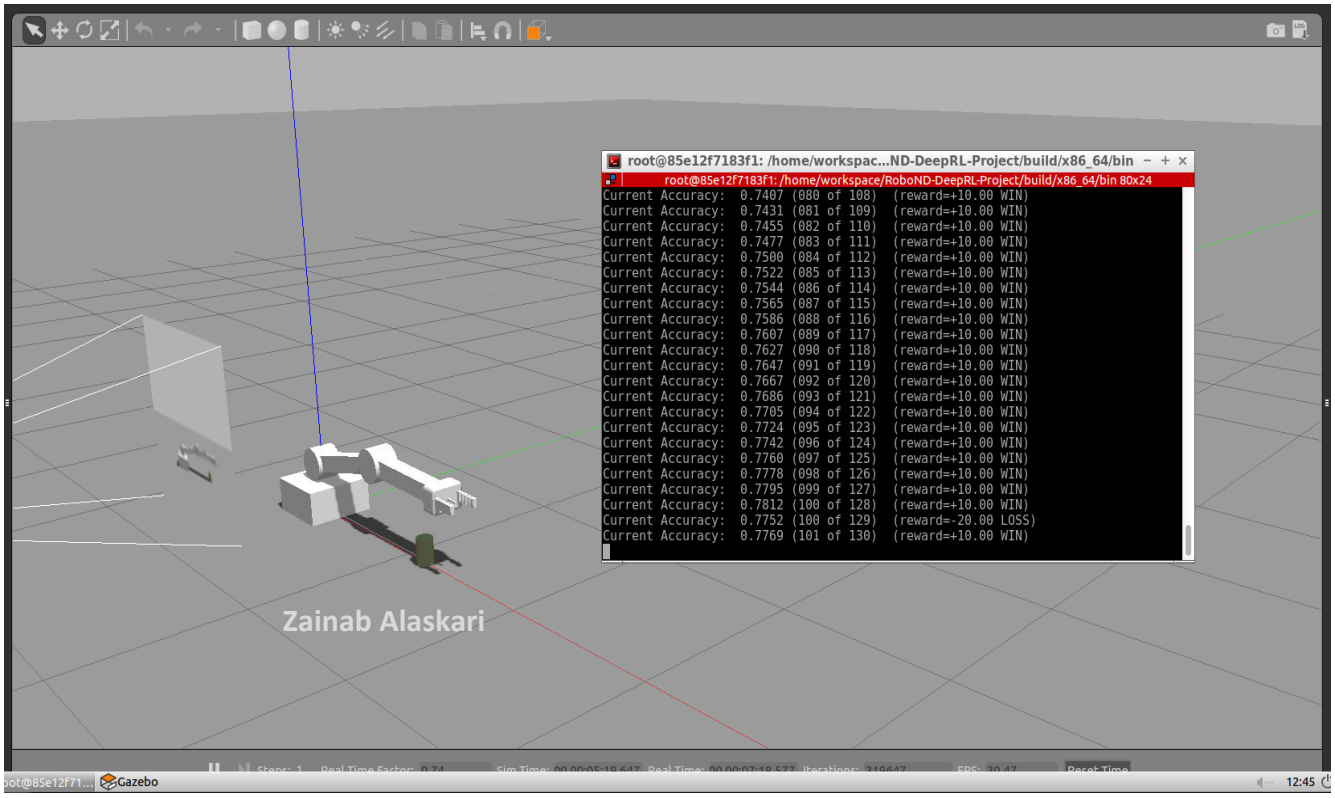

- Environment width and height were reduced even further to 64 to reduce computational overhead and system complexity where it was needed to compensate for the increase of the used batch size for this objective.
- “RMSprop” Optimizer was tested at first with varying the learning rate value between (0.1 , 0.01 , 0.001), varying image size and varying batch sizes. All the testing with this optimizer didn’t give good results, therefore the “Adam” optimizer was tested, and it was found to be working well for the specified goal.
- It seemed that keeping the LEARNING_RATE at 0.01 helped achieved the required 80% accuracy fast.
- REPLAY_MEMORY stayed the same. It is set to a high value to keep information of all the episodes it will perform.
- BATCH_SIZE was increased massively to 512 because better estimates of gradient decent in neural networks was needed to achieve the 80% accuracy within 100 runs for the second objective. As this task was more difficult than the first one because a small surface area was needed to achieve a collision with the object.
- USE_LSTM set to true to improve learning from past experiences.
- LSTM_SIZE was kept the same at 256.

Results:

First objective:

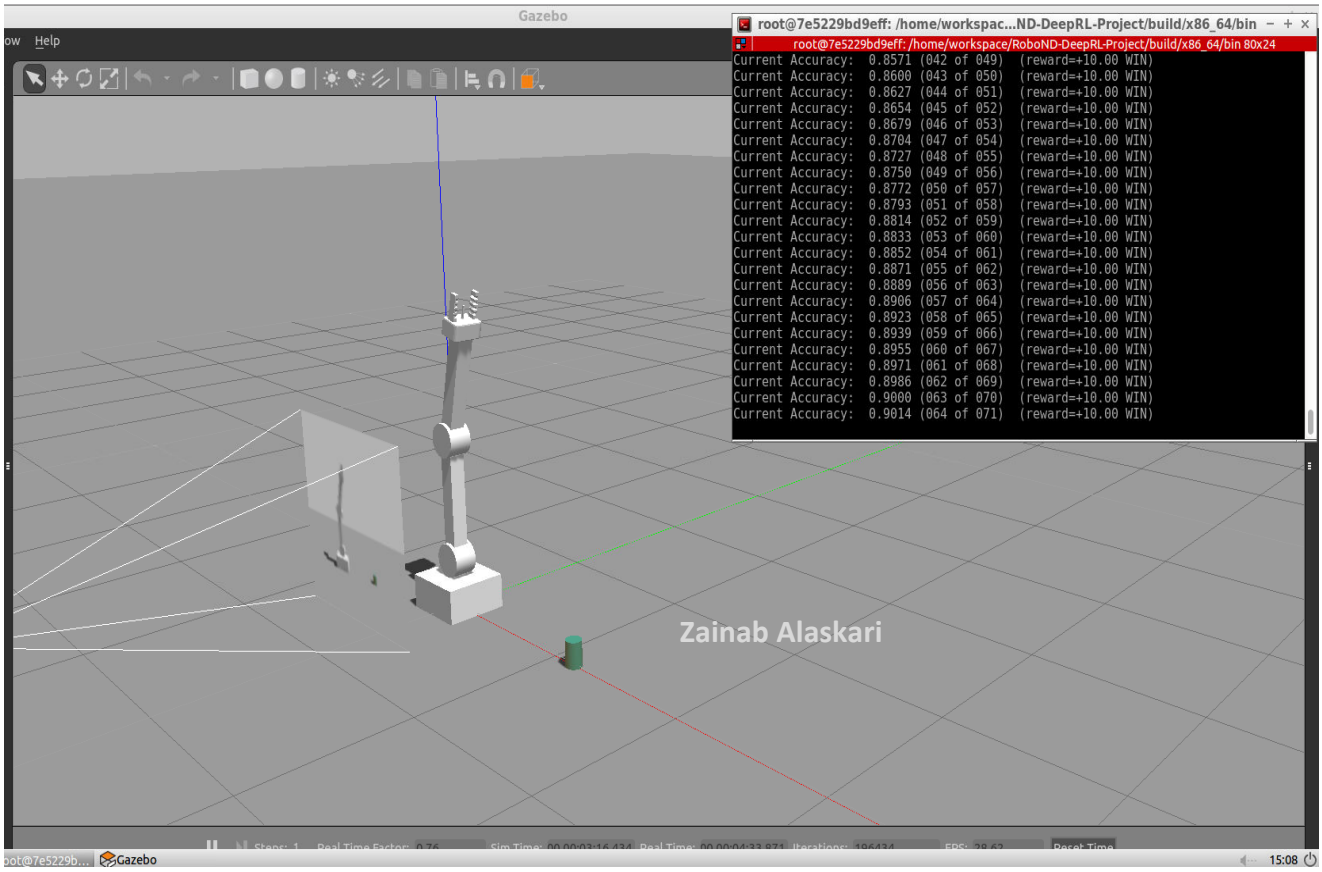
For the first objective many learning rates were tested, and it was found that the higher the learning rate the faster the learning procedure gets, however it doesn't mean the better the achieved accuracy gets.

The below figure shows that setting the learning rate to 0.2 does not achieve 90% accuracy within 100 runs. The accuracy at the 100th run was found to be almost 74%.

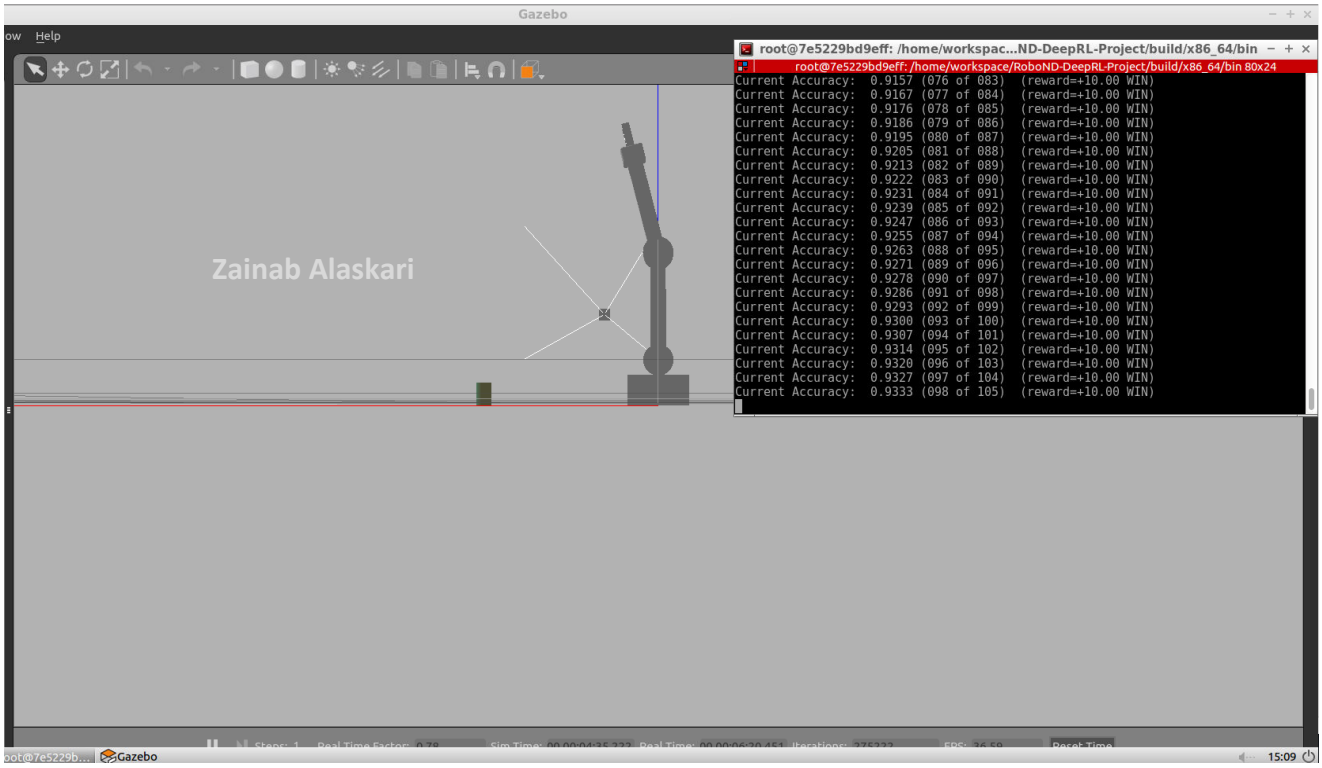


Only decreasing the learning rate to 0.01 helped achieve the first goal of having any part of the robot touch the object at 90% within the first 100 epoxides.

This figure shows the DQN agent reaching a 90% accuracy after 70 episodes of training have passed which achieves the first objective of the project successfully.



At the 100th run the DQN agent achieves 93% accuracy.

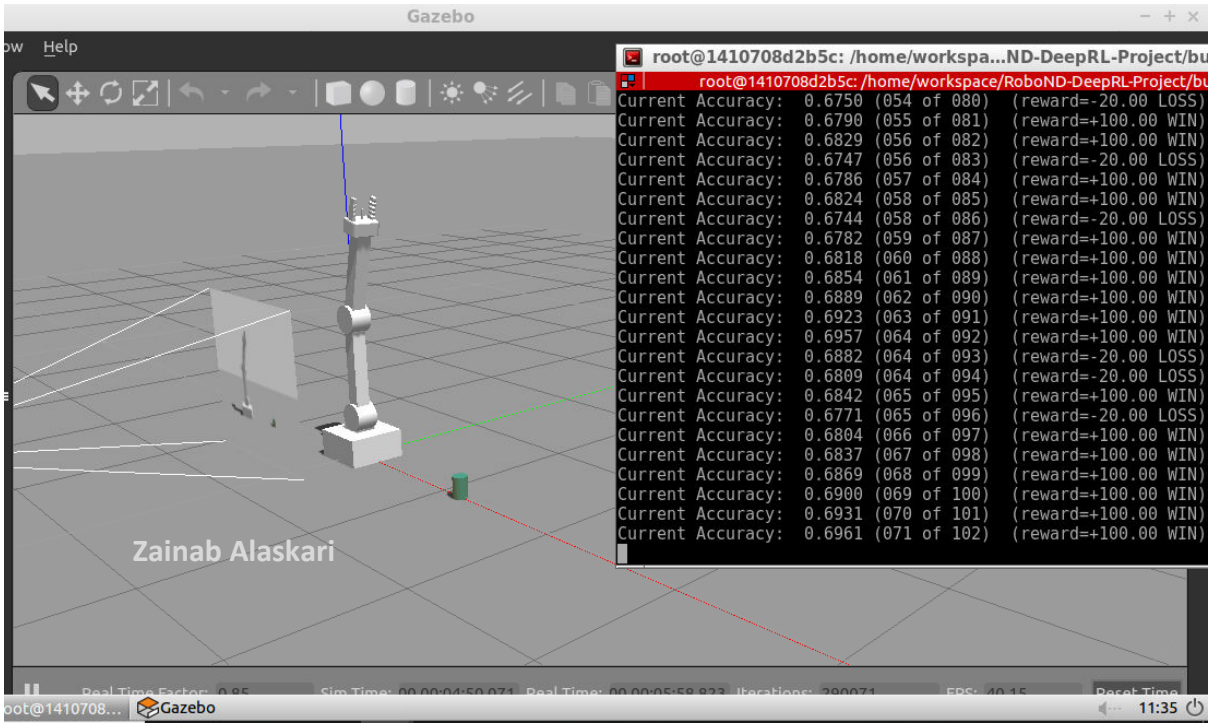


The results show that the reward functions work greatly for training the DQN agent.

Second Objective:

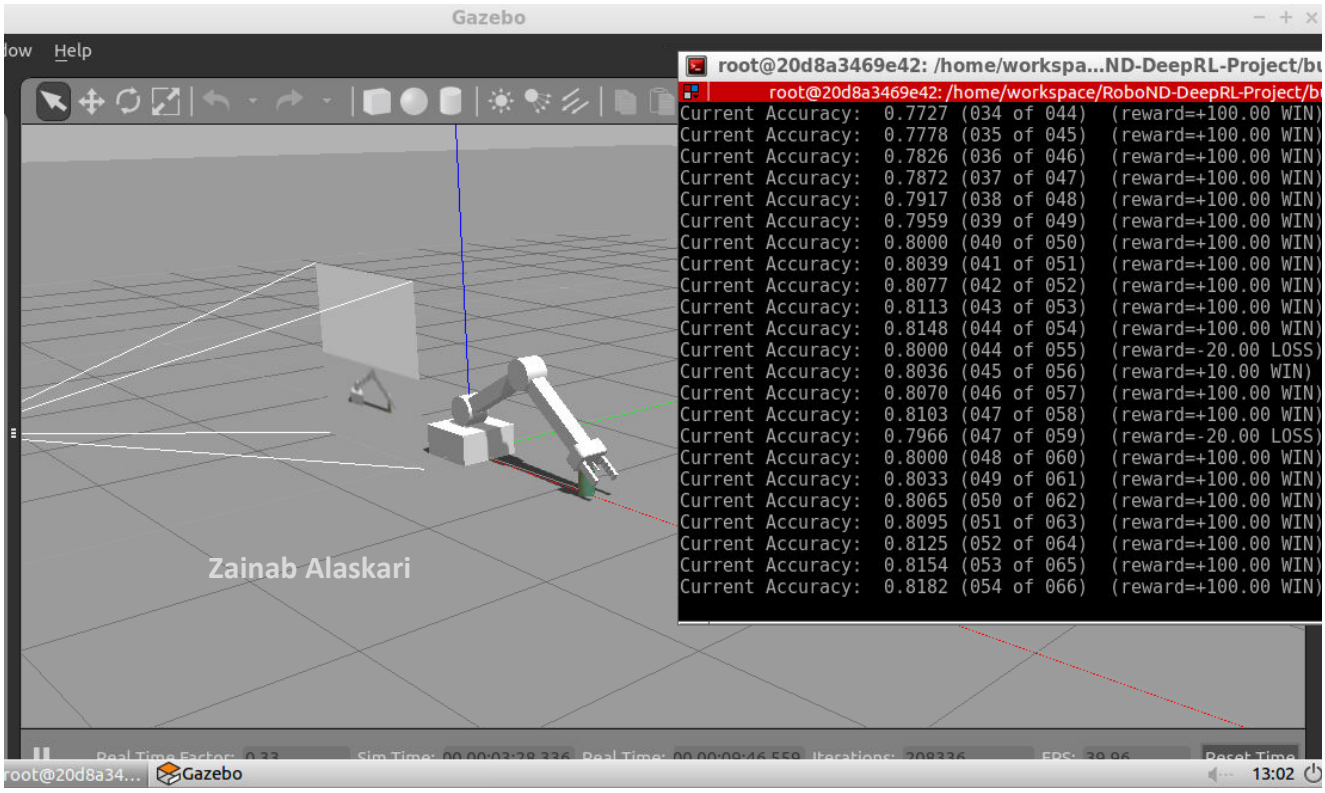
To have the gripper base touch the object with 80% accuracy within the first 100 runs was a challenge and required many trial and errors before converging to the right hyperparameters values.

Initially “RMSprop” Optimizer was tested with varying the learning rate value between 0.1 and 0.001, varying image size and varying batch sizes. All the testing with this optimizer didn’t give good results as seen from the below figure.

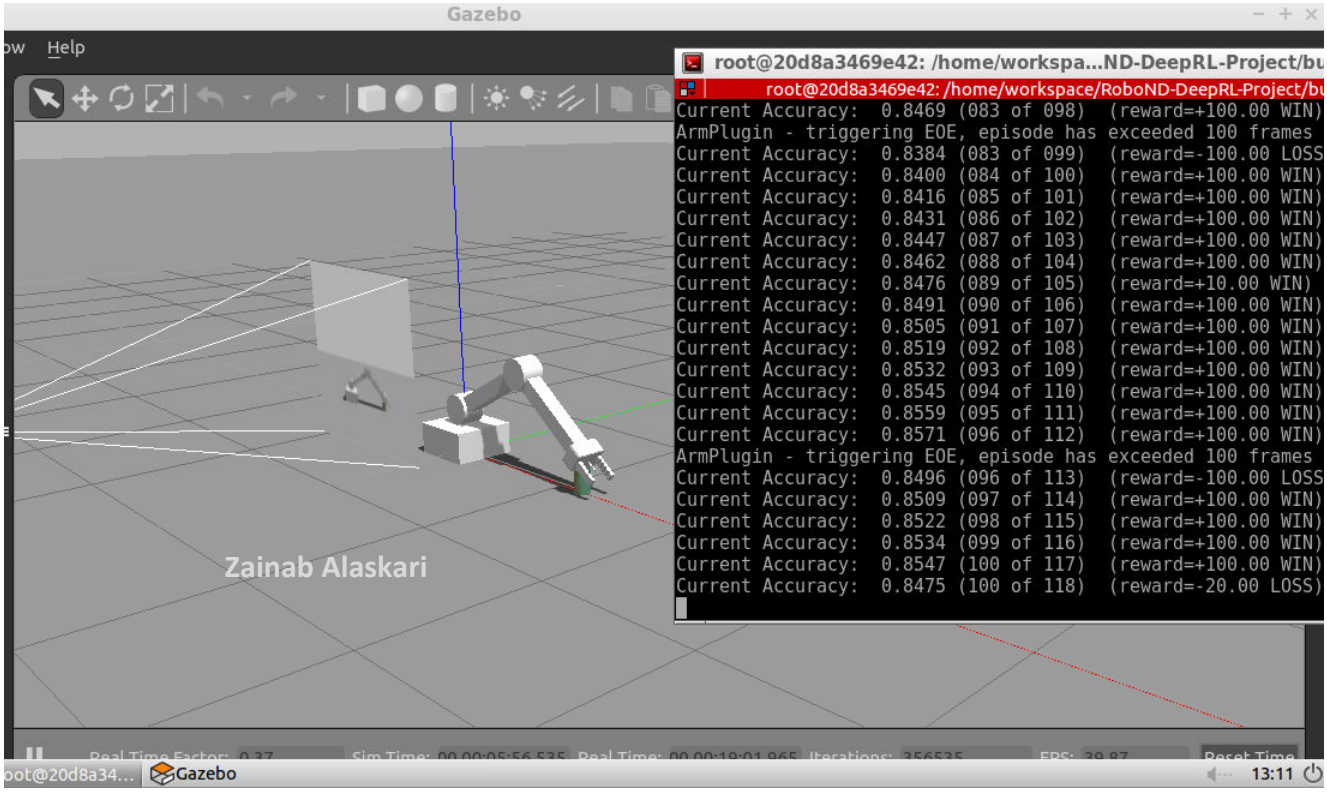


Changing the optimizer to “Adam” and increasing the batch size to 512 helped achieve the required accuracy faster and achieve the second objective successfully.

This figure shows the DQN agent reaching 80% accuracy after only 50 episodes of training have passed!



At the 100th run the DQN agent achieves 84% accuracy.



The results show that the reward functions work greatly for training the DQN agent for the second objective.

Discussion:

For achieving both objectives, the agent always started with a rough learning curve, where it would start moving very far off from the object and would collide with the ground many times. With the passing of each episode the agent learned through its reward functions what actions to take to score the highest and hence achieve its goal. The speed and accuracy at which the learning took always depended on the values of the hyperparameters, having a good set of hyperparameters was very crucial for achieving the two objectives. The results also showed that using the “Adam” optimizer achieved high accuracy very fast.

Future Work:

For future work, I would definitely test the “Adam” optimizer with the first task to see if it would achieve an accuracy of 90% faster. Also, I would test other optimizers with both tasks to test how they affect the training of the DQN agent.

Also, to achieve better results in the first task I would have increased the reward function value for when any part of the robot arm touches the object to REWARD_WIN of 1*100 instead on only 10, to test how the speed of training would increase.

I would also test using speed joint control for both tasks to see how the speed of the agent affects the learning process and how much would it smooth the robot’s motion.

Overall, any set of hyperparameters can be tuned more with more trial and errors to achieve great accuracies within the most minimum number of episodes.