Deep RL Project Writeup

Goal of the project is to set up and optimize a reinforcement learning task. A Deep Q-Network is used to inform the actions of a robotics arm in a simulated environment. The task is to touch a given object.

- ***Objective 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.
- ***Objective 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.

Training of the network along with the simulations were done using Udacity's Project Workspace with GPU enabled environment.

Following tasks were implemented before running the simulation environment and commencement of training.

- Subscription to camera and collision topics.
- Creation of DQN agent using learning hyper-parameters defined as constants in the file.
- The definition of collision checks to determine which robot arm link touched the target object.
- Both velocity control and position control of the robot were implemented.
- Criteria for ground contact of the arm were defined.
  The z coordinates of the gripper link bounding box were used.
  The threshold ground level was set at z = 0.05 meters, to accommodate for the inaccuracy of the distance calculation.

Reward Function

A reward system was set up in order to teach the robotic arm how to reach for the target object.
Positive and negative rewards are defined as numeric constants **REWARD WIN** and **REWARD LOSS**.
A win is issued for touching the prop with any link (objective 1) or the gripper base link (objective 2).
A loss is issued when the arm touches the ground or when the length of the episode exceeds 100 steps in time. The occurrence of any of these events will end the episode.

Since positive rewards from touching the prop by the gripper link are very sparse, an interim reward function was designed based on distance from the arm to the object. It's purpose is to guide the robot arm towards the target.
The relevant measure is the distance **dt** between gripper and target object.
For consecutive steps **t−1** and **t**, the difference **Δt = d[t-1] − dt** was calculated.
In order to obtain a smooth reward function, the moving average of the recorded **Δ** values was computed

```
const float distDelta  = lastGoalDistance - distGoal;

// compute the smoothed moving average of the delta of the distance to the goal
avgGoalDelta  = (avgGoalDelta * ALPHA) + (distDelta * (1.0f - ALPHA));
```

A linear function of the ***avgGoalDelta*** is proportional to the average speed towards the goal and should be a good candidate for an interim reward function.

Objective 1

Velocity control of the robot arm was enabled during simulation runs. Reward function is directly proportional to the moving average.

```
rewardHistory = INTERIM_REWARD * avgGoalDelta;
```

Objective 2

Reward parameters and agent control were adjusted according to suit the demands of this task.

- Velocity control was switched to position control
- Value of ALPHA was lowered from an initial guess of 0.9 to values in the range 0.4-0.5

The need to test out non-linear reward functions, led me to the following function.

```
rewardHistory = exp(-1.0f * distGoal) * INTERIM_REWARD * avgGoalDelta;
```

Exp function outputs are in the range - [0, 1]. The multiplication of distGoal with -1 ensures that lesser distances to the goal ensures a bigger reward.
The arm accelerated towards the target object, but failed to accomplish the task.

Calculation of the moving average was also modified to abandon the usage of the

constant ALPHA.
The intention was to give less weightage to distance deltas when the arm is closer to the object.
This should rectify the inaccuracy in the computation of small distance deltas.
Simulation runs showed the arm slowing down to a halt before reaching the target object and then failed the objective.

```
const float gamma = exp(-1.0f * distGoal);
avgGoalDelta = (avgGoalDelta * gamma) + (distDelta * (1 - gamma));
rewardHistory = 30.0f * avgGoalDelta;
```

Failure of the task under a non-linear function led me back to trying out a linear one.
Introduced a negative offset to punish a standstill or movement away from the object.
INTERIM_REWARD and INTERIM_OFFSET represent the slope and the offset, roughly speaking.

```
rewardHistory = INTERIM_REWARD * avgGoalDelta - INTERIM_OFFSSET;
```

DQN Parameter Tuning

The size of the input was set to 64 × 64. Long-short-term memory was enabled by setting USE_LSTM to true.
RMSprop and the Adam optimizer were utilized. RMSprop was used during the simulation runs for objective 1, Adam was used in the reward function design experiments for objective 2.
When going from objective 1 to objective 2, the batch size was increased from 32 to 256.
The LSTM size was increased from 64 to 256, which should allow the agent to memorize more complex moves.

During simulation runs for objective 2, the start value of the exploration parameter EPS_START was set between 0.7 <-> 0.9. The value of EPS_END were set between 0.02 <-> 0.05. Learning rate was set between 0.01 <-> 0.1. Performance of each configuration was determined by accuracy readings from the outputs.

```
// Define DQN API Settings

#define INPUT_CHANNELS 3
#define ALLOW_RANDOM true
#define DEBUG_DQN false
#define GAMMA 0.9f
#define EPS_START 0.9f
#define EPS_END 0.05f
#define EPS_DECAY 200


// Define Hyperparameters

#define INPUT_WIDTH   64
#define INPUT_HEIGHT  64
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 32
#define USE_LSTM true
#define LSTM_SIZE 64
```

```
// Define DQN API Settings

#define INPUT_CHANNELS 3
#define ALLOW_RANDOM true
#define DEBUG_DQN false
#define GAMMA 0.9f
#define EPS_START 0.9f
#define EPS_END 0.05f
#define EPS_DECAY 200


// Define Hyperparameters

#define INPUT_WIDTH   64
#define INPUT_HEIGHT  64
#define OPTIMIZER "Adam"
#define LEARNING_RATE 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 256
#define USE_LSTM true
#define LSTM_SIZE 256
```

DQN configurations used for objective 1(left) and objective 2(right).

Results

Both of the objectives were reached. Exponential reward function did not measure up to linear reward function.

The largest positive effect on the agents performance was observed when lowering the value of ALPHA parameter from 0.9 to 0.4.
This finding emphasizes the importance of designing a smooth reward function.
The number of episodes it took the agent to reach the threshold accuracy varied between ≈ 200 and ≈ 700, depending on the initial conditions of the simulation run.
During the simulation runs, the accuracy improved slowly over time.

	Objective 1	Objective 2
REWARD_WIN	10.0	20.0
REWARD_LOSS	-10.0	-20.0
INTERIM_REWARD	10.0	4.0
ALPHA	0.9	0.4

Accuracy/Episodes	0.98 / 100	0.82 / 228

Future Work

The reinforcement learning performance could possibly be improved by =>

- Further tuning of DQN parameters like discount factor, size of replay memory and epsilon decay rate.
- Further investigation of exponential dampening function to create a viable non-linear reward function.

It will also be interesting to see how well the framework generalizes for a robotic arm with more degrees of freedom.