

**Udacity Robotics Nanodegree**

**Deep Reinforcement Learning Project**

**Arm Manipulation**

**Chris Rowe**

**rwbotx@gmail.com**

**Project Repo: <https://github.com/rwbot/RoboND-DeepRL-Project>**

## Setup

In the method `ArmPlugin::Load()`, subscriber nodes were defined to facilitate camera communication and collision detection through Gazebo:

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

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

In the method `ArmPlugin::Create()`, the DQN Agent was created using the parameters defined at the top of **Armplugin.cpp**:

```
agent = dqnAgent::Create(INPUT_WIDTH, INPUT_HEIGHT, INPUT_CHANNELS,
    DOF*2, OPTIMIZER, LEARNING_RATE, REPLAY_MEMORY, BATCH_SIZE, GAMMA,
    EPS_START, EPS_END, EPS_DECAY, USE_LSTM, LSTM_SIZE, ALLOW_RANDOM,
    DEBUG_DQN);
```

The output of the DQN agent is mapped to particular actions of the robotic arm. The method `ArmPlugin::updateAgent()` is responsible for executing the action selected by the DQN agent. Two types of joint control were implemented: velocity control and position control. Both are dependent upon whether the issued action is an odd or even number. Odd-numbered actions decrease the joint position/velocity, while even-numbered actions increase the joint position/velocity. Additionally, the current delta of the position/values is used:

```
// Evaluating if action is even or odd
const int actionSign = 1 - 2 * (action % 2);
// Set joint position
float joint = ref[action/2] + actionSign * actionJointDelta;
// Set joint velocity
const float velocity = vel[action/2] + actionSign * actionVelDelta;
```

## Reward Function

The reward function is a crucial component in guiding the DQN agent through the process of learning to manipulate the arm. After every end of episode (EOE), a particular reward is issued, depending on the triggering event.

**REWARD\_LOSS is issued when the current episode exceeds 100 steps or when the arm contacts the ground.**

The **REWARD\_LOSS** issued when the arm contacts the ground is detected using the `GetBoundingBox()` function from the Gazebo API which provides the min/max XYZ values of the arm. Comparing this with the Z value of the ground, collisions can be detected.

```

const bool checkGroundContact = (gripBBox.min.z <= groundContact ||
gripBBox.max.z <= groundContact);

if(checkGroundContact){
    rewardHistory = REWARD_LOSS;
    ...
}

```

**REWARD\_WIN** is issued when any part of the arm contacts the object (Objective 1) and when only the gripper contacts the object (Objective 2).

The reward function for arm and object collision first checks for a collision between any part of the arm and the object. Depending on which objective was being attempted, the boolean GRIP\_ONLY controls whether the arm-object collision is a win or loss.

If GRIP\_ONLY == 0, the rules of Objective 1 are applied, and the arm-object collision is considered a win. Additionally, if that part happens to be the gripper, the reward is multiplied.

If GRIP\_ONLY == 1, the rules of Objective 2 are applied, and the arm-object collision is considered a loss by default. However, if that part happens to be the gripper, the collision is issued a multiplied reward.

```

// Define whether ARM or GRIPPER only is considered a win
#define GRIP_ONLY 0

// Check if the collision is between any part of the arm and the object
bool collisionItemCheck = ( strcmp(contacts->contact(i).collision1().c_str(),
COLLISION_ITEM) == 0 );

if (collisionItemCheck){
    // Reward for any collision with arm given only when GRIP_ONLY == 0
    rewardHistory = GRIP_ONLY ? REWARD_LOSS : REWARD_WIN;

    // Check if the collision is between only the gripper and the object
    bool collisionPointCheck = ( strcmp(contacts-
>contact(i).collision2().c_str(), COLLISION_POINT) == 0 );
    if (collisionPointCheck)
        rewardHistory = rewardHistory * 5.0f;

    // Update state
    newReward = true;
    endEpisode = true;
    return;
}

```

## Interim Reward

To reduce the necessary training time, an interim reward function was implemented which would help guide the arm towards the object. The reward is proportional to the change in distance between the arm and the object. If the arm is moving away from the object, the interim reward is negative. Conversely, if the arm is moving towards the object, the interim reward is positive. The raw values cause the arm to have a jerking movement, so they are smoothed by taking a moving average. Finally, due to the agent's tendency to remain still, a time penalty was introduced to force the agent to finish as quickly as possible.

```
// The current distance to the goal
const float distGoal = BoxDistance(gripBBox,propBBox);
// The current change in distance from the last position
const float distDelta = lastGoalDistance - distGoal;
// Smoothing the values using a moving average
avgGoalDelta = (avgGoalDelta * ALPHA) + (distDelta * (1.0f - ALPHA));
// Time penalty to reduce stand-still and completion time
rewardHistory = REWARD_INTERIM * avgGoalDelta - TIME_PENALTY;
```

## Hyperparameters

Below are all the relevant parameters used. Between both objectives, only 3 parameters were changed.

In Q-Learning, epsilon-Greedy is a typical exploration method used to encourage an agent to explore more of the state-action space. The larger the value, the more frequently a random action is chosen instead of one with the highest q-value. EPS\_START defines the initial value. **0.9f** ensures the agent is exposed to many state-action spaces. EPS\_END defines the final value. **0.0f** ensures that only learned actions with the highest q-value is used once it has been sufficiently exposed. EPS\_DECAY defines the rate by which the value decays from initial to final over time. For objective 2, this was increased to **250**. Touching the gripper to the object was less likely to happen than the whole arm. Preferably, the agent needed enough exploration to encounter that situation, but not too much that the agent falls into a "best loss" policy.

```
// 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 0.0f
#define EPS_DECAY 250 // Obj1: 200
```

The input dimensions were set to match that of the image captured by the camera: 64 x 64 pixels. "RMSprop" was used as the optimizer. LSTM was set to **true** to enable the agent to consider previously encountered states. LSTM\_SIZE was set to **256**, increasing the agent's ability to consider more complex moves. A LEARNING\_RATE of **0.1f** was sufficient for the first objective, but not for the second as it took much too long learn. It was increased to **0.9f**

because it was high enough that the time reduced dramatically and low enough to ensure it would actually converge. Similarly, the BATCH\_SIZE was increased from **64** to **128**.

```
// Learning Hyperparameters
#define INPUT_WIDTH 64
#define INPUT_HEIGHT 64
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.9f // Obj1: 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 128 // Obj1: 64
#define USE_LSTM true
#define LSTM_SIZE 256
```

REWARD\_INTERIM was the reward incrementally issued to lead the agent towards the object. ALPHA was used to calculate a moving average of the delta distances. To deter the agent from standing still, a TIME\_PENALTY was introduced. All three of these were tuned concurrently to find the values that issued appropriate interim rewards.

```
// Reward Parameters
#define REWARD_WIN 30.0f
#define REWARD_LOSS -30.0f
#define REWARD_INTERIM 5.0f
#define ALPHA 0.6f
#define TIME_PENALTY 0.4f
```

## Results

The DQN was able to successfully complete both objectives. Objective 1 parameters were consistent in results of 90% within the range of 100 to 120 episodes. Objective 2 parameters were less consistent. The initial episodes significantly influenced the convergence time. On less ideal runs, it would take the agent 40 episodes before receiving a single win. On more ideal runs, the agent would receive a win within the first 15 episodes. This puts the result of 80% consistent within the range of 100 to 300 episodes.

- 
- The screenshot shows a Gazebo simulation environment. In the center, a white robotic arm with two joints is positioned on a white base. It has a gripper at the end. To the right of the arm, there is a small green cylindrical object. The ground is a light gray plane with red and green grid lines. In the background, there are some faint blue structures. On the left side, a black terminal window is open, displaying a list of performance metrics. At the top of the terminal, the path "/root/.gazebo/" is visible. The metrics include "Current Accuracy:" followed by three numbers in parentheses and a reward value. The values generally increase from top to bottom, indicating improvement over time.
- /root/.gazebo/
- Current Accuracy: 0.8333 (845 of 954) (reward=+30.00 WIN)
- Current Accuracy: 0.8364 (846 of 955) (reward=+30.00 WIN)
- Current Accuracy: 0.8393 (847 of 956) (reward=+30.00 WIN)
- Current Accuracy: 0.8421 (848 of 957) (reward=+30.00 WIN)
- Current Accuracy: 0.8448 (849 of 958) (reward=+30.00 WIN)
- Current Accuracy: 0.8475 (850 of 959) (reward=+30.00 WIN)
- Current Accuracy: 0.8506 (851 of 960) (reward=+30.00 WIN)
- Current Accuracy: 0.8525 (852 of 961) (reward=+30.00 WIN)
- Current Accuracy: 0.8548 (853 of 962) (reward=+30.00 WIN)
- Current Accuracy: 0.8571 (854 of 963) (reward=+30.00 WIN)
- Current Accuracy: 0.8594 (855 of 964) (reward=+30.00 WIN)
- Current Accuracy: 0.8615 (856 of 965) (reward=+30.00 WIN)
- Current Accuracy: 0.8636 (857 of 966) (reward=+30.00 WIN)
- Current Accuracy: 0.8657 (858 of 967) (reward=+30.00 WIN)
- Current Accuracy: 0.8676 (859 of 968) (reward=+30.00 WIN)
- Current Accuracy: 0.8696 (860 of 969) (reward=+30.00 WIN)
- Current Accuracy: 0.8714 (861 of 970) (reward=+30.00 WIN)
- Current Accuracy: 0.8732 (862 of 971) (reward=+30.00 WIN)
- Current Accuracy: 0.8750 (863 of 972) (reward=+30.00 WIN)
- Current Accuracy: 0.8767 (864 of 973) (reward=+30.00 WIN)
- Current Accuracy: 0.8784 (865 of 974) (reward=+30.00 WIN)
- Current Accuracy: 0.8808 (866 of 975) (reward=+30.00 WIN)
- Current Accuracy: 0.8816 (867 of 976) (reward=+30.00 WIN)
- Current Accuracy: 0.8831 (868 of 977) (reward=+30.00 WIN)
- Current Accuracy: 0.8846 (869 of 978) (reward=+30.00 WIN)
- Current Accuracy: 0.8861 (870 of 979) (reward=+30.00 WIN)
- Current Accuracy: 0.8875 (871 of 980) (reward=+30.00 WIN)
- Current Accuracy: 0.8889 (872 of 981) (reward=+30.00 WIN)
- Current Accuracy: 0.8902 (873 of 982) (reward=+30.00 WIN)
- Current Accuracy: 0.8916 (874 of 983) (reward=+30.00 WIN)
- Current Accuracy: 0.8929 (875 of 984) (reward=+30.00 WIN)
- Current Accuracy: 0.8941 (876 of 985) (reward=+30.00 WIN)
- Current Accuracy: 0.8953 (877 of 986) (reward=+30.00 WIN)
- Current Accuracy: 0.8966 (878 of 987) (reward=+30.00 WIN)
- Current Accuracy: 0.8977 (879 of 988) (reward=+30.00 WIN)
- Current Accuracy: 0.8989 (880 of 989) (reward=+30.00 WIN)
- Current Accuracy: 0.9009 (881 of 990) (reward=+30.00 WIN)
- Current Accuracy: 0.9011 (882 of 991) (reward=+30.00 WIN)
- Current Accuracy: 0.9022 (883 of 992) (reward=+30.00 WIN)
- Current Accuracy: 0.9032 (884 of 993) (reward=+30.00 WIN)
- Current Accuracy: 0.9043 (885 of 994) (reward=+30.00 WIN)
- Current Accuracy: 0.9053 (886 of 995) (reward=+30.00 WIN)
- /root/.gazebo/

- Terminal Output:

Current Accuracy	reward
0.6800 (034 of 050)	(reward=-27800.00 LOSS)
0.6667 (034 of 051)	(reward=+900.00 WIN)
0.6731 (035 of 052)	(reward=+900.00 WIN)
0.6792 (036 of 053)	(reward=+900.00 WIN)
0.6852 (037 of 054)	(reward=+900.00 WIN)
0.6909 (038 of 055)	(reward=+900.00 WIN)
0.6964 (039 of 056)	(reward=+900.00 WIN)
0.7018 (040 of 057)	(reward=+900.00 WIN)
0.7069 (041 of 058)	(reward=+900.00 WIN)
0.7119 (042 of 059)	(reward=+900.00 WIN)
0.7167 (043 of 060)	(reward=+900.00 WIN)
0.7213 (044 of 061)	(reward=+900.00 WIN)
0.7258 (045 of 062)	(reward=+900.00 WIN)
0.7302 (046 of 063)	(reward=+900.00 WIN)
0.7344 (047 of 064)	(reward=+900.00 WIN)
0.7385 (048 of 065)	(reward=+900.00 WIN)
0.7424 (049 of 066)	(reward=+900.00 WIN)
0.7463 (050 of 067)	(reward=+900.00 WIN)
0.7500 (051 of 068)	(reward=+900.00 WIN)
0.7536 (052 of 069)	(reward=+900.00 WIN)
0.7571 (053 of 070)	(reward=+900.00 WIN)
0.7606 (054 of 071)	(reward=+900.00 WIN)
0.7508 (054 of 072)	(reward=-27800.00 LOSS)
0.7534 (055 of 073)	(reward=+900.00 WIN)
0.7566 (056 of 074)	(reward=+900.00 WIN)
0.7600 (057 of 075)	(reward=+900.00 WIN)
0.7632 (058 of 076)	(reward=+900.00 WIN)
0.7662 (059 of 077)	(reward=+900.00 WIN)
0.7692 (060 of 078)	(reward=+900.00 WIN)
0.7722 (061 of 079)	(reward=+900.00 WIN)
0.7750 (062 of 080)	(reward=+900.00 WIN)
0.7770 (063 of 081)	(reward=+900.00 WIN)
0.7805 (064 of 082)	(reward=+900.00 WIN)
0.7831 (065 of 083)	(reward=+900.00 WIN)
0.7857 (066 of 084)	(reward=+900.00 WIN)
0.7882 (067 of 085)	(reward=+900.00 WIN)
0.7907 (068 of 086)	(reward=+900.00 WIN)
0.7933 (069 of 087)	(reward=+900.00 WIN)
0.7955 (070 of 088)	(reward=+900.00 WIN)
0.7974 (071 of 089)	(reward=+900.00 WIN)
0.8000 (072 of 090)	(reward=+900.00 WIN)
0.8022 (073 of 091)	(reward=+900.00 WIN)
0.8043 (074 of 092)	(reward=+900.00 WIN)
0.7957 (074 of 093)	(reward=-27800.00 LOSS)
0.7979 (075 of 094)	(reward=+900.00 WIN)
0.8000 (076 of 095)	(reward=+900.00 WIN)

With enough GPU time, the performance of the DQN agent could be improved by employing a more extensive approach to tuning. Each set of parameters could be run multiple times to decrease the effect of varying initial episodes.