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 faciliate camera communication and collision detection through Gazebo:

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

```
// Collision Message subscriber
collisionSub = collisionNode-
>Subscribe("/gazebo/arm_world/tube/tube_link/my_contact",&ArmPlugin::onCollis
ionMsg, 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 a 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 makes contact with the ground.

The `REWARD_LOSS` issued when the arm makes contact with 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 makes contact with the object (Objective 1) and when only the gripper makes contact with 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 adequately 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
```

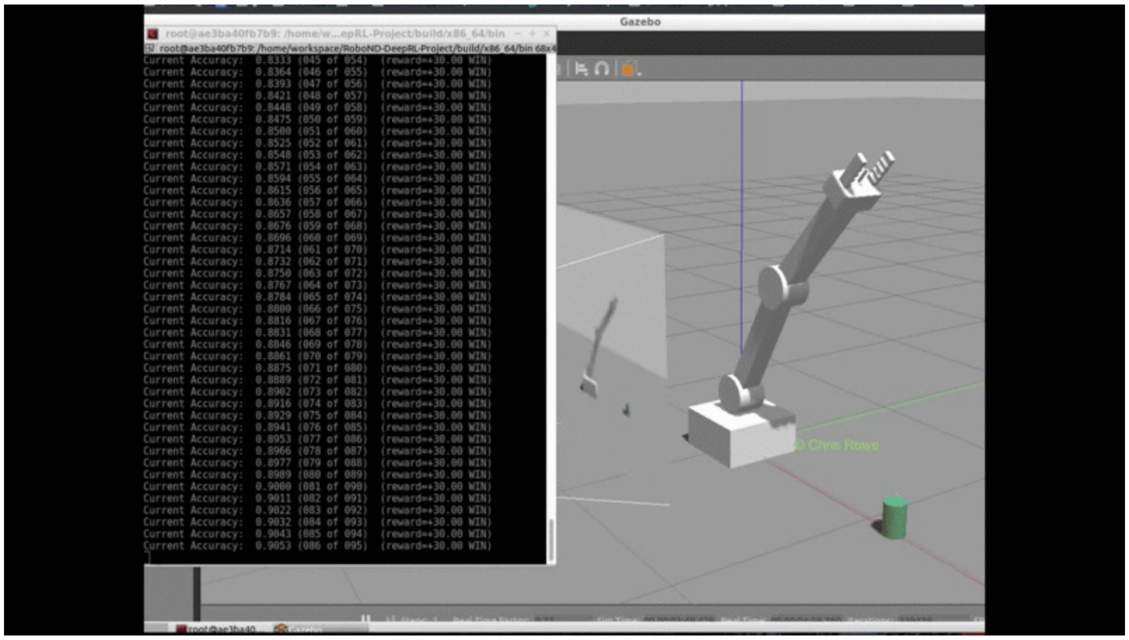
REWARD_INTERIM was the reward incrementally issued to lead the agent towards the object. ALPHA was used to calculate a moving average of the delta distances. To deter the agent from standing still, a TIME_PENALTY was introduced. All three of these were tuned concurrently to find the values that issued 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

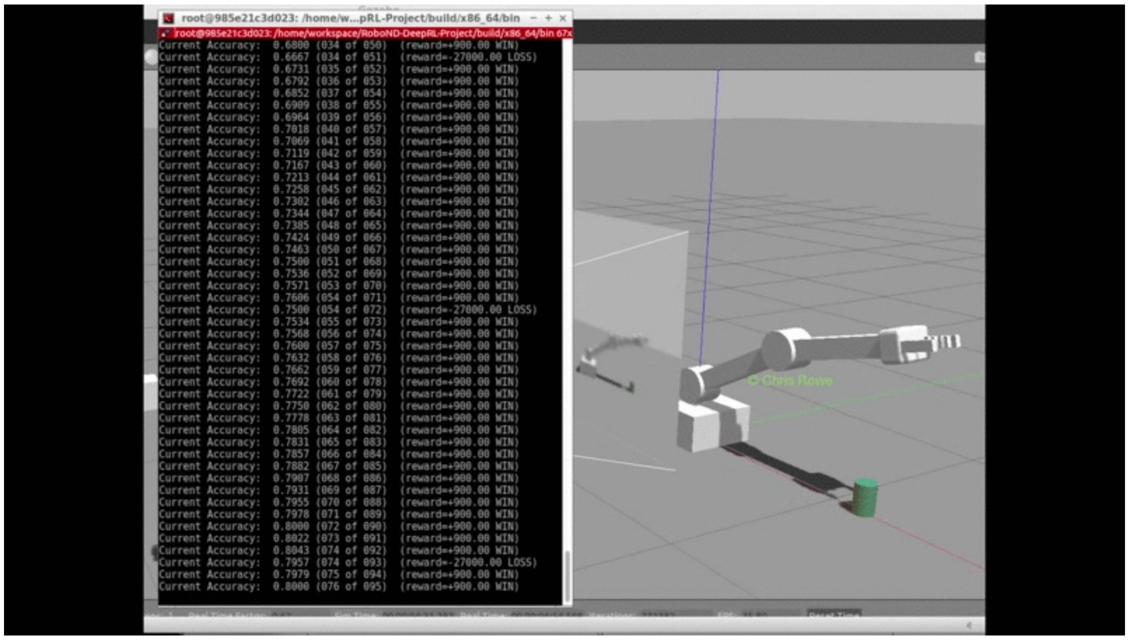
The DQN was able to successfully complete both objectives. Objective 1 parameters were consistent in results of 90% in a range from 100 to 120 episodes. Objective 2 parameters were less consistent. The initial episodes greatly influenced the convergence time. On less ideal runs, it would take the agent 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 results of 80% in a range from 100 to 300 episodes.

- Objective 1 - 91% at 100th episode



```
root@ae3ba40fb7b9: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 68x4
Current Accuracy:  0.8333 (045 of 054)  (reward=+30.00 WIN)
Current Accuracy:  0.8364 (046 of 055)  (reward=+30.00 WIN)
Current Accuracy:  0.8393 (047 of 056)  (reward=+30.00 WIN)
Current Accuracy:  0.8421 (048 of 057)  (reward=+30.00 WIN)
Current Accuracy:  0.8448 (049 of 058)  (reward=+30.00 WIN)
Current Accuracy:  0.8475 (050 of 059)  (reward=+30.00 WIN)
Current Accuracy:  0.8500 (051 of 060)  (reward=+30.00 WIN)
Current Accuracy:  0.8525 (052 of 061)  (reward=+30.00 WIN)
Current Accuracy:  0.8548 (053 of 062)  (reward=+30.00 WIN)
Current Accuracy:  0.8571 (054 of 063)  (reward=+30.00 WIN)
Current Accuracy:  0.8594 (055 of 064)  (reward=+30.00 WIN)
Current Accuracy:  0.8615 (056 of 065)  (reward=+30.00 WIN)
Current Accuracy:  0.8636 (057 of 066)  (reward=+30.00 WIN)
Current Accuracy:  0.8657 (058 of 067)  (reward=+30.00 WIN)
Current Accuracy:  0.8676 (059 of 068)  (reward=+30.00 WIN)
Current Accuracy:  0.8696 (060 of 069)  (reward=+30.00 WIN)
Current Accuracy:  0.8714 (061 of 070)  (reward=+30.00 WIN)
Current Accuracy:  0.8732 (062 of 071)  (reward=+30.00 WIN)
Current Accuracy:  0.8750 (063 of 072)  (reward=+30.00 WIN)
Current Accuracy:  0.8767 (064 of 073)  (reward=+30.00 WIN)
Current Accuracy:  0.8784 (065 of 074)  (reward=+30.00 WIN)
Current Accuracy:  0.8800 (066 of 075)  (reward=+30.00 WIN)
Current Accuracy:  0.8816 (067 of 076)  (reward=+30.00 WIN)
Current Accuracy:  0.8831 (068 of 077)  (reward=+30.00 WIN)
Current Accuracy:  0.8846 (069 of 078)  (reward=+30.00 WIN)
Current Accuracy:  0.8861 (070 of 079)  (reward=+30.00 WIN)
Current Accuracy:  0.8875 (071 of 080)  (reward=+30.00 WIN)
Current Accuracy:  0.8889 (072 of 081)  (reward=+30.00 WIN)
Current Accuracy:  0.8902 (073 of 082)  (reward=+30.00 WIN)
Current Accuracy:  0.8916 (074 of 083)  (reward=+30.00 WIN)
Current Accuracy:  0.8929 (075 of 084)  (reward=+30.00 WIN)
Current Accuracy:  0.8941 (076 of 085)  (reward=+30.00 WIN)
Current Accuracy:  0.8953 (077 of 086)  (reward=+30.00 WIN)
Current Accuracy:  0.8966 (078 of 087)  (reward=+30.00 WIN)
Current Accuracy:  0.8977 (079 of 088)  (reward=+30.00 WIN)
Current Accuracy:  0.8989 (080 of 089)  (reward=+30.00 WIN)
Current Accuracy:  0.9000 (081 of 090)  (reward=+30.00 WIN)
Current Accuracy:  0.9011 (082 of 091)  (reward=+30.00 WIN)
Current Accuracy:  0.9022 (083 of 092)  (reward=+30.00 WIN)
Current Accuracy:  0.9032 (084 of 093)  (reward=+30.00 WIN)
Current Accuracy:  0.9043 (085 of 094)  (reward=+30.00 WIN)
Current Accuracy:  0.9053 (086 of 095)  (reward=+30.00 WIN)
```

- Objective 2: 81% at 100th episode



```
root@985e21c3d023: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 67x
Current Accuracy:  0.6800 (034 of 050)  (reward=+900.00 WIN)
Current Accuracy:  0.6667 (034 of 051)  (reward=-27000.00 LOSS)
Current Accuracy:  0.6731 (035 of 052)  (reward=+900.00 WIN)
Current Accuracy:  0.6792 (036 of 053)  (reward=+900.00 WIN)
Current Accuracy:  0.6852 (037 of 054)  (reward=+900.00 WIN)
Current Accuracy:  0.6909 (038 of 055)  (reward=+900.00 WIN)
Current Accuracy:  0.6964 (039 of 056)  (reward=+900.00 WIN)
Current Accuracy:  0.7018 (040 of 057)  (reward=+900.00 WIN)
Current Accuracy:  0.7069 (041 of 058)  (reward=+900.00 WIN)
Current Accuracy:  0.7119 (042 of 059)  (reward=+900.00 WIN)
Current Accuracy:  0.7167 (043 of 060)  (reward=+900.00 WIN)
Current Accuracy:  0.7213 (044 of 061)  (reward=+900.00 WIN)
Current Accuracy:  0.7258 (045 of 062)  (reward=+900.00 WIN)
Current Accuracy:  0.7302 (046 of 063)  (reward=+900.00 WIN)
Current Accuracy:  0.7344 (047 of 064)  (reward=+900.00 WIN)
Current Accuracy:  0.7385 (048 of 065)  (reward=+900.00 WIN)
Current Accuracy:  0.7424 (049 of 066)  (reward=+900.00 WIN)
Current Accuracy:  0.7463 (050 of 067)  (reward=+900.00 WIN)
Current Accuracy:  0.7500 (051 of 068)  (reward=+900.00 WIN)
Current Accuracy:  0.7536 (052 of 069)  (reward=+900.00 WIN)
Current Accuracy:  0.7571 (053 of 070)  (reward=+900.00 WIN)
Current Accuracy:  0.7606 (054 of 071)  (reward=+900.00 WIN)
Current Accuracy:  0.7500 (054 of 072)  (reward=-27000.00 LOSS)
Current Accuracy:  0.7534 (055 of 073)  (reward=+900.00 WIN)
Current Accuracy:  0.7568 (056 of 074)  (reward=+900.00 WIN)
Current Accuracy:  0.7600 (057 of 075)  (reward=+900.00 WIN)
Current Accuracy:  0.7632 (058 of 076)  (reward=+900.00 WIN)
Current Accuracy:  0.7662 (059 of 077)  (reward=+900.00 WIN)
Current Accuracy:  0.7692 (060 of 078)  (reward=+900.00 WIN)
Current Accuracy:  0.7722 (061 of 079)  (reward=+900.00 WIN)
Current Accuracy:  0.7750 (062 of 080)  (reward=+900.00 WIN)
Current Accuracy:  0.7778 (063 of 081)  (reward=+900.00 WIN)
Current Accuracy:  0.7805 (064 of 082)  (reward=+900.00 WIN)
Current Accuracy:  0.7831 (065 of 083)  (reward=+900.00 WIN)
Current Accuracy:  0.7857 (066 of 084)  (reward=+900.00 WIN)
Current Accuracy:  0.7882 (067 of 085)  (reward=+900.00 WIN)
Current Accuracy:  0.7907 (068 of 086)  (reward=+900.00 WIN)
Current Accuracy:  0.7931 (069 of 087)  (reward=+900.00 WIN)
Current Accuracy:  0.7955 (070 of 088)  (reward=+900.00 WIN)
Current Accuracy:  0.7978 (071 of 089)  (reward=+900.00 WIN)
Current Accuracy:  0.8000 (072 of 090)  (reward=+900.00 WIN)
Current Accuracy:  0.8022 (073 of 091)  (reward=+900.00 WIN)
Current Accuracy:  0.8043 (074 of 092)  (reward=+900.00 WIN)
Current Accuracy:  0.7957 (074 of 093)  (reward=-27000.00 LOSS)
Current Accuracy:  0.7979 (075 of 094)  (reward=+900.00 WIN)
Current Accuracy:  0.8000 (076 of 095)  (reward=+900.00 WIN)
```

Future Work

Given more 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.