Udacity Robotics Nanodegree Term 2

DeepRL Project

Robert Aleck | github.com/mnbf9rca | 16th August 2019

Introduction

A Deep Q-Learning Network was implemented against a framework for a robotic arm with 3 DOF simulated within the Gazebo environment. The author had to:

- Subscribe to the relevant topics (camera imagery, collisions)
- Instantiate a DQN Agent
- Choose between velocity or position based arm joint manipulation, and implement in code
- Design an appropriate reward function, covering successful (collision between defined objects) and unsuccessful actions, including episode terminal states and interim rewards
- Tune hyperparameters and reward function for two scenarios:
    - At least 90% collision accuracy between any part of the arm and the collision object over at least 100 episodes, and
    - At least 80% accuracy in collisions between the gripper base of the robot arm and the collision object over at least 100 episodes

Reward functions

The robot arm control was selected as "position", using default values.

At first, a large (>100 or even >500) REWARD_WIN and REWARD_LOSS value was used, with smaller/fractional rewards issued for interim states, however, the model failed to train effectively. Smaller values were then chosen, with additional multipliers providing greater weighting to episode-terminal (collision) events, as follows:

```
#define REWARD_WIN 0.15f
#define REWARD_LOSS -0.15f
#define REWARD_COLISSION_GROUND 10          // multiplier - it the ground
#define REWARD_COLLISION_CORRECT_PART 20          // multiplier - hit the correct item
#define REWARD_COLLISION_WRONG_PART 10          // multiplier - hit the wrong item
#define MIN_DISTANCE_TO_MOVE_WITHOUT_PENALTY 0.05f // how far must the gripper have moved compared to last frame to avoid a penalty
```

The reward function was unchanged when training to hit the gripper base. The selection of collision target is changed with the REWARD_ANY_COLLISION parameter – when set to false, only collisions between COLLISION_ITEM and COLLISION_POINT were rewarded, otherwise all collisions with COLLISION_ITEM resulted in a reward.

```
#define REWARD_ANY_COLLISION false          // reward for hitting any part of the arm on the tube
```

The following rewards were issued for episode-terminal states:

Ground contact	REWARD_LOSS * REWARD_COLISSION_GROUND
Collision between COLLISION_ITEM and either any other part, or with COLLISION_POINT	REWARD_WIN * REWARD_COLLISION_CORRECT_PART
Collision between COLLISION_ITEM and any part other than COLLISION_POINT	REWARD_LOSS * REWARD_COLLISION_WRONG_PART

To encourage the arm to move towards the target, interim rewards were issued. A positive movement towards the goal received REWARD_WIN * timePenalty (where timePenalty is a value trending from 1 to 0 proportional to the number of frames completed), whereas movement away received REWARD_LOSS. Additionally, as it was noticed that the arm spent a long time in a relatively stable position, REWARD_LOSS is added to rewardHistory at each frame that the gripper failed to move, in a smoothed, moving average, at least MIN_DISTANCE_TO_MOVE_WITHOUT_PENALTY since the previous frame.

Hyperparameters

The following hyperparameters were chosen and used for both collision models:

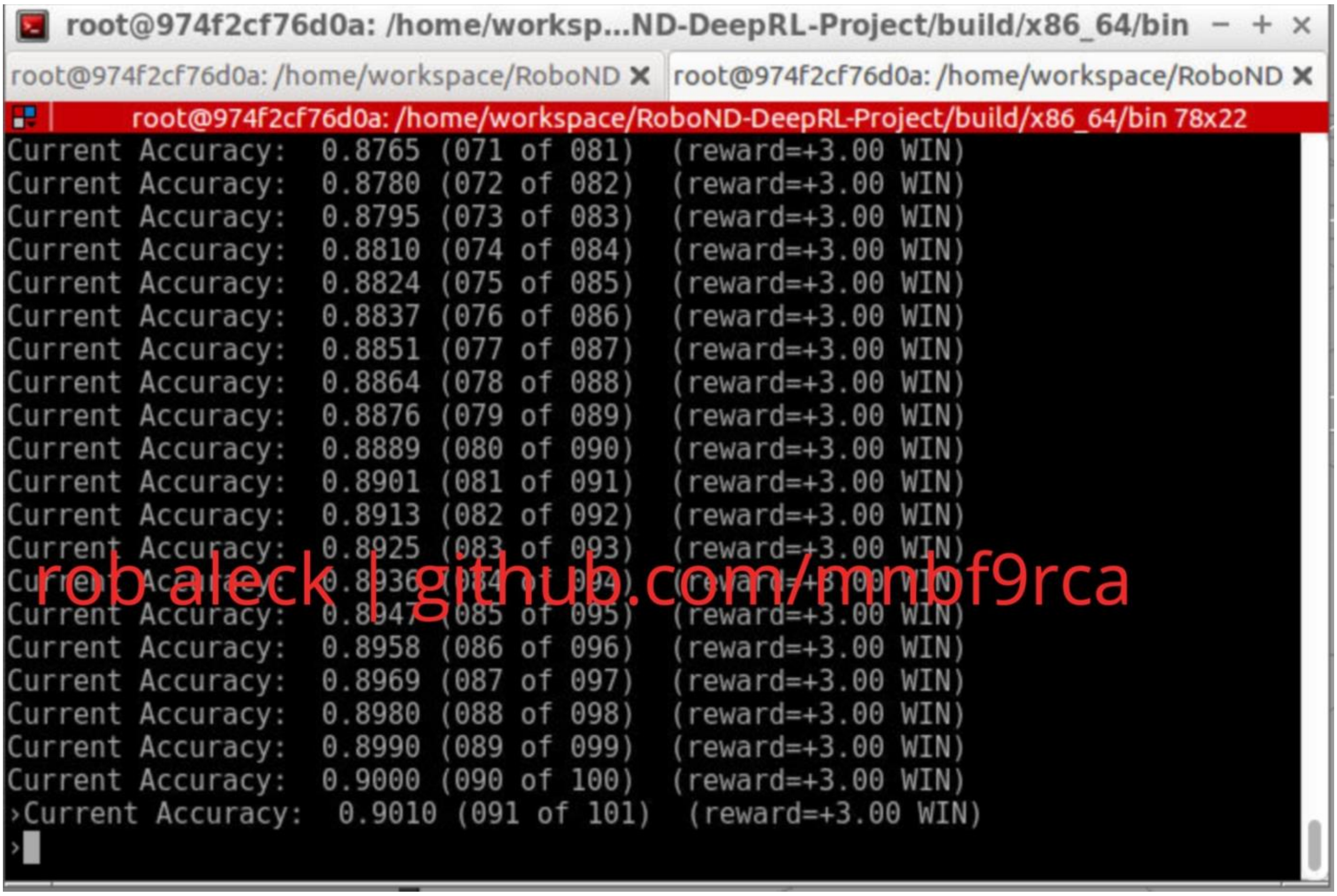
```
#define INPUT_WIDTH 64
#define INPUT_HEIGHT 64
#define OPTIMIZER "Adam"
#define LEARNING_RATE 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 512
#define USE_LSTM true
#define LSTM_SIZE 256
```

The input image size was observed to be 64x64, so this was selected for the model. Adam was selected for the optimizer based on previous experience with reinforcement learning, where the author has found it to converge quicker for relatively simple models. The learning rate was initially 0.01, but was increased to 0.1 to speed up learning – no evidence of overfitting was found. Replay memory was left as default. Batch size was increased to take advantage of the large memory available on the server, and the LSTM size increased experimentally – higher than 256 failed to fit, lower than 128 slowed learning

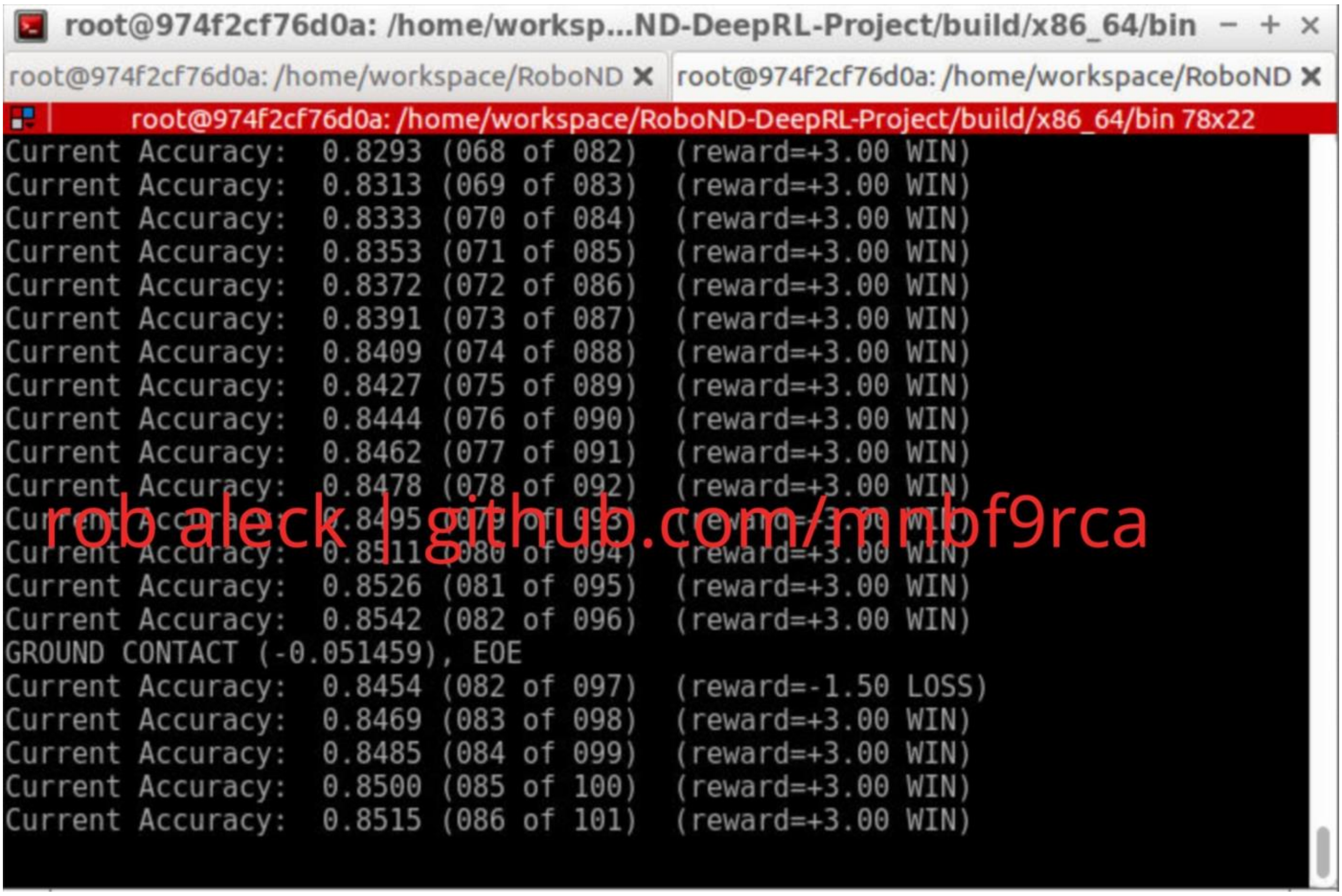
Results

In general, both objectives were met relatively quickly. If the arm initially trained away from the collision item, learning took longer, but it almost always achieved at least one WIN within the first 10 episodes. In some cases, after apparently operating correctly for 20-30 episides, the arm would “overstretch” and hit the floor – this can be seen in one of the examples below at episode 82.

Have any part of the robot arm touch the object of interest, with at least a 90% accuracy for a minimum of 100 runs.



Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy for a minimum of 100 runs.



Future work

Future work could include:

- Training the model to locate the object between the gripper arms, close the gripper when in place, and move the object
- Randomising the location of the object
- The LSTM has not been optimised (using only defaults the author has used successfully in the past).
- Adjusting other hyperparameters, such as the DISTANCE_DECAY_FACTOR, which was left at 0.90 throughout.