Deep RL Arm Manipulation

Antti Hietanen

May 12, 2018

1 Introduction

In this report a brief summary of the Deep RL Arm Manipulation project is described. The target of the project was to teach a simulated 3 DoF robotic arm to accomplish two primary tasks: 1) have any part of the robot arm to touch the object of interest; 2) have only the gripper base of the robot arm to touch the object. The accuracy of the robot performance has to be 90 % and 80 % in the tasks respectively.

During the project an intelligent agent (robot) was created which then interacted with the simulated environment, gathered information and changed its behavior according to the feedback (reward) it was given by the environment. Two sensors, imaging and contact sensor, were used in the project to acquire information about the current status of the robot and the environment and finally used to decide what kind of reward the robot will be receiving.

2 Reward Functions

In the project the agent receives reward when it arrives to the terminal states. In the project there are two different terminal states: 1) robot hits the tube; 2) robot hits the ground. In the former state the robot is given a positive reward while in the latter one the robot receives a negative reward. In addition, if the robot does not touch the tube in a certain time limit (100 episodes was used in the project) the robot is assigned a negative reward.

In addition to the terminal state reward the agent is given an interim reward all the time while the robot is interacting with the environment. The used interim reward function was smoothed moving average of the delta of the distance to the goal and it is defined as:

$$average\_delta = (average\_delta * moving\_avg) + (dist * (1 - moving\_avg)), \quad (1)$$

where *dist* is the distance between the robot gripper and tube, *average_delta* is the previous interim reward and *moving_avg* defines how much weight is given for the previous interim reward and how much for the distance between the arm and the tube. In both tasks velocity controller was used.

2.1 Task 1: Collision between the arm and the object

In the first task the reward for the terminal state was set to 500. Sign of the reward was positive for the desired terminal state and negative for the undesired

state. The terminal state reward was set to high so it will be the first priority of the robot and not be overcame by the cumulative (in task 2) interim reward. The *moving_avg* was set to zero which meant in the experiment that only the gripper distance to the tube was affecting the amount of interim reward the robot was given.

2.2 Task 2: Collision between the gripper and the object

For the task 2 the *moving_avg* was set to 0.1 which was noted to robustify the process of only touching the tube by the gripper. In addition receiving the positive terminal state reward was made more strict by only allowing the gripper ($arm :: gripperbase :: gripper\_link$) of the robot to touch the object.

3 Hyperparameters

In the project several hyperparameters had to be tuned to get the optimal results. The parameters for the Deep Q-Learning algorithm were:

Image size – The width and height of the image which the learning algorithm is going to process. *Default:* 512×512

Optimizer – Gradient decent optimizing method, such as RMSprop or Adam. *Default:* None

Learning rate – How much to modify the model weights based on the given gradient. If the parameter value is low the learning might be more reliable but the learning process takes more time. With too high value the process takes less time but the process will be more unreliable and the algorithm can easily "overshoot" over the optimal values. *Default:* 0.0

Replay memory – The replay memory buffer size. The experience replay memory of a learning algorithm stores the transitions that the agent observes, allowing the algorithm to reuse this data later. The replay memory should stabilizes and improves the for instance the DQN training process. *Default:* 10000

Batch size – Amount of samples taken from the total training set to propagated through the network in the training phase. *Default:* 8

Use LSTM – LSTM is a notation for Long Short Term Memory networks which are a special kind of RNN, capable of learning long-term dependencies. *Default:* False

LSTM Size – Number of inputs for the LSTM network. *Default:* 32

3.1 Task 1: Collision between the arm and the object

In the first task the image was reduced to 64×64. This was noted to reduce memory requirements of the GPU and speed up the whole training process without sacrificing too much from performance. The selected optimizer was *RMSprop*, which was used in many example codes and during the experiments it worked well. Learning rate was set to 0.1 which was pretty high but worked in the first task. Replay memory was kept as default. As with the image size the

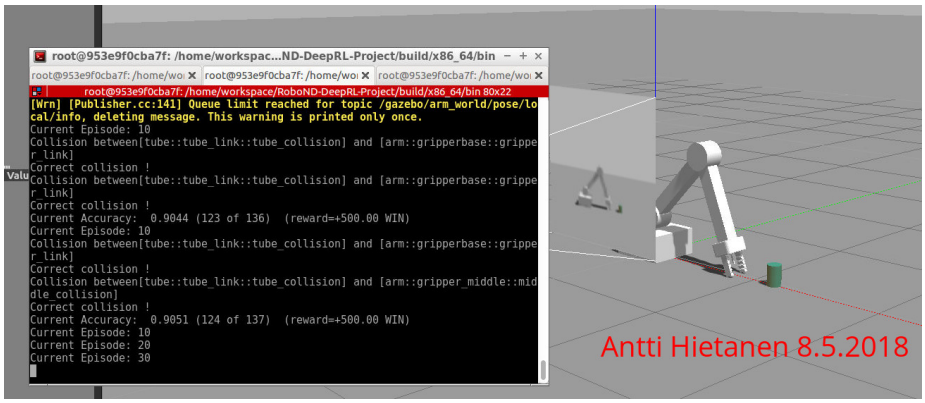

Figure 1: Accuracy of Task 1.

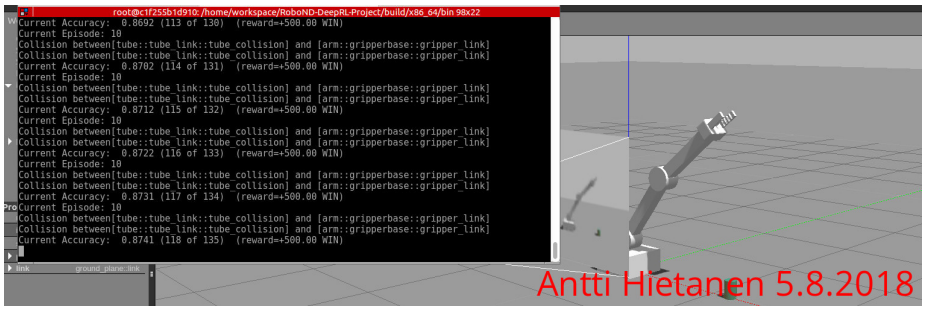

Figure 2: Accuracy of Task 2

batch size was increased to 32 to reduce memory footprint without any notable performance problems. The LSTM flag was set to $True$ and $lstm\_size$ to 128 which increased the performance a little bit.

3.2 Task 2: Collision between the gripper and the object

The only hyperparameter which was changed for the task 2 was the learning rate which was modified to 0.01. With the learning rate of 0.1 the most common action of the robot after 10 minutes of training was to overshoot the tube so that only the $link\_2$ of the robot arm touched the object of interest and consequently the episode was flagged as failed.

4 Results

In Figure 1 and Figure 2 are shown the results for Task 1 and Task 2 respectively. From the images we can see that the robot achieved the required performance level described in the manuscript by using the configurations explained in Section 2 and Section 3. During the project it was noted that the biggest factors affecting the robot accuracy was the proper design of the interim reward function and use of enough small learning rate.

5 Future work

The future work will include testing the learning capabilities of the robot in more difficult environments. This includes initializing the object of interest to a random position every time a new episode starts and making the robot more flexible by allowing it to rotate along its base and increasing the arm reach. To improve the existing results the future work will also include studying and experimenting different reward functions, testing the position controller and using different gradient optimization methods.