Deep RL

Deep RL Arm Manipulation Project

The goal of this project is to develop a Deep RL agent controlling a robot arm.

The two primary project objectives are:

- 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 touch the object of interest, with at least a 80% accuracy for a minimum of 100 runs.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Basic Requirements

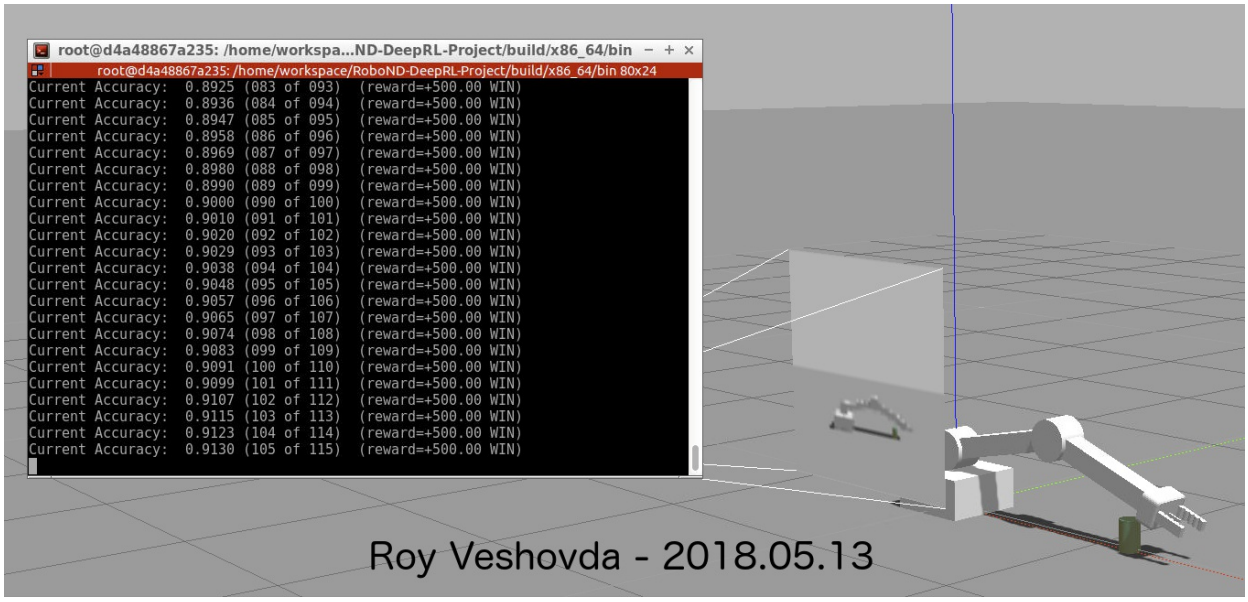
1. Include in your project submission your write-up (PDF format), supporting images or videos, and the entire project folder minus the build folder.

This is fulfilled in the submitted Github repository.

Objectives

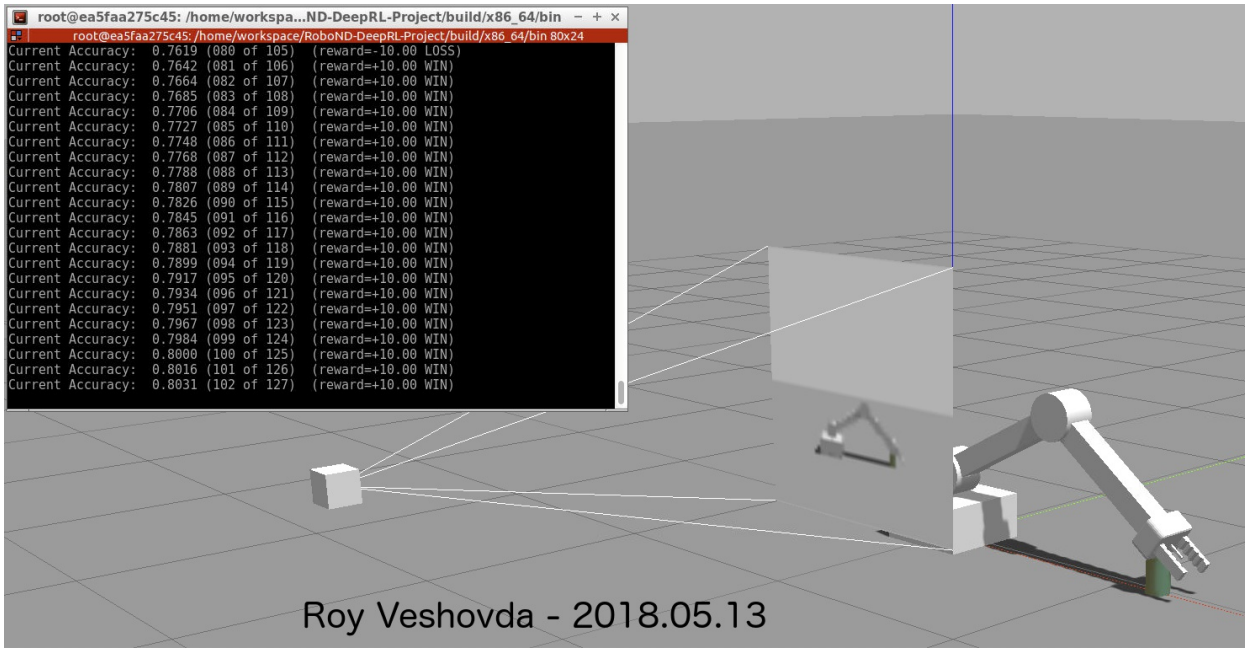
2. Have any part of the robot arm touch the object of interest, with at least a 90% accuracy for a minimum of 100 runs.

The image below shows the result for this objective. Accuracy of 91,3% over 115 runs.



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

The image below shows the result for this objective. Accuracy of 80,31% over 127 runs.



Writeup Requirements

4. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You should submit your writeup as pdf.

You are reading the writeup.

5. Reward Functions: Explain the reward functions that you created.

The reward functions are implemented in `ArmPlugin.cpp`, and consists of the following parts:

- Reward for collision (in method `onCollisionMsg`). For objective 1 this reward was for any part of the robot arm. For objective 2 this was only for the gripper base.
- Reward (penalty) for touching the ground (in method `OnUpdate`). Same for both parts.
- Reward for moving toward object (in method `OnUpdate`). Same for both parts. Positive value if moving toward. Negative if moving away.
- Reward (penalty) for not moving (in method `OnUpdate`). This was introduced for objective 2 to improve performance.

The scaling of these rewards and penalties were revised between objective 1 and 2, as the images of the results shows. This was done as part of the tuning of reward functions and hyperparameters.

For joint control, the direct position control was chosen and implemented. This was done by giving the DQN agent double number of actions related to directions of freedom (DOF) for the robot arm. And letting two actions affect each joint, one increasing value, and the other decreasing the value.

6. Hyperparameters: Specify the hyperparameters that you selected for each objective, and explain the reasoning behind the selection.

Image dimensions was set to the same as the size of the input (64x64), since upscaling the input images would not gain any more information to the DQN agent (`INPUT_WIDTH` and `INPUT_HEIGHT`).

The optimizer (`OPTIMIZER`) was set to `Adam` as the `Adam` optimizer generally performs better than `RMSprop`.

For the first objective the learing rate (`LEARNING_RATE`) was set to 0.1, while it was decreased to 0.01 for objective 2. The reduction was a result of trial and error and on the assumption that the agent should learn a bit slower to get better overall performance.

For objective 1 the replay memory (`REPLAY_MEMORY`) was set to 10000, but increasd to 20000 for objective 2. This was done to allow the agent to have more previous experiences to learn from.

Batch size (`BATCH_SIZE`) was set as high as 512 to allow the learning to perform as good as possible. Normally a large batch size i preferred, but need to be a tradeoff in relation to how much GPU memory is available.

I chose to use LTSM (`USE_LSTM` = true) and set the size (`LSTM_SIZE`) to 256. This was also a result of trial and error, but based on the assumption that I wanted the memory features of

LSTM, and a fairly large network, to allow the agent to remember a few steps back. The size of the network is again a tradeoff between performance and available GPU memory (in addition to overall latency performance).

7. Results: Explain the results obtained for both objectives. Include discussion on the DQN agent's performance for both objectives. Include watermarked images, or videos of your results.

The images for the results are shown above, and will not be repeated here.

For objective 1 the required accuracy was achieved fairly fast, and the agent was able to repeat the motion to touch the object over and over again. Once the winning motion was found the agent simply repeated the actions and got the results required.

As for objective 2 I spent a lot of time tuning and testing and tuning and testing more. I the beginning I had a reward function which punished the robot arm if any other parts than the gripper base touched the object. This was a bit too strict reward function, as the agent would be punished even if the grippers touched the object. I got much better results once I eased the restrictions for the grippers, and only gave reward if the gripper base touched the object, and no punishment for other parts touching. This gave the wanted behaviour for the agent, as shown in the image above.

8. Future Work: Briefly discuss how you can improve your current results.

In the beginning the agent could spend a few runs (sometimes as much as 10 runs) before it found the preferred trajectory. Interim rewards based on wanted trajectory could be explored to get the arm to faster find the best trajectory. This might not be a sustainable solution as it might limit the agent to explore other trajectories.

Velocity based control should also be explored and compared to position based control to find the best solution.

The base was fixed for this solution, so a future version should of course explore a non-fixed base. And the object always appeared at the same location. A random object placement should of course also be explored.