README.md

Udacity Robotics Nanodegree: Deep RL Arm Manipulation

Sebastian Castro, 2019

Introduction

In this project, a deep reinforcement learning (RL) agent is trained to move a simulated robot manipulator toward a target object.

- The **observation** is an image showing a 2D projection of the robot arm environment
- The **output** is the joint position or velocity of the robot arm joints

To solve this problem, a Deep-Q Network (DQN) agent is used. This means that the output of the agent is one of several possible discrete actions. Depending on the mode used (position or velocity control), the DQN agent will increase or decrease the position or velocity of an individual joint at each time step.

Most of the changes in this project were implemented in the source file `ArmPlugin.cpp`.

Setup

Network Architecture

A provided convolutional neural network (CNN) architecture is used, which is provided in the project. This consists of 3 2-D convolutional layers with kernel size of 5 and stride of 2, with batch normalization and ReLU activations. This architecture can be found in the `DQN.py` script.

The only change to the network was that the input image is scaled to 256-by-256 pixels. This reduced the network size and memory usage when training on a GPU, at the potential expense of accuracy loss. This decrease in resolution is especially noticed when attempting to randomize the initial object location.

Reward Function

The reward function consists of 3 components:

- Losing reward (if hitting the ground or timing out): -50
- Winning reward (if object is reached): +50
- Interim reward: `10*avgGoalDist - 1`

where `avgGoalDist = alpha*goalDist - (1-alpha)*avgGoalDist` and `alpha` = 0.5

The interim reward was designed such that moving towards the goal would increase reward, while every time step that the agent is active is penalized to encourage quickly moving towards the goal.

NOTE: For the case where only the gripper base should touch the object, a winning reward of +50 is received if the gripper base collides with the object, and a "neutral" reward of 0 is received if another link of the arm collides with the object. This is done because it is a preferable outcome to timing out or hitting the ground.

Hyperparameters

Training is done with the Adam optimizer, with an initial learning rate of 0.001. Replay memory consists of a buffer of 5000 experiences and training is done in mini-batches of 16 experiences.

Training parameter	Value
Optimizer	Adam
Learning rate	0.001
Experience buffer size	5000
Mini-batch size	16

The agent uses a discount factor of 0.95, meaning the estimated value is halved in approximately 13.5 steps.The agent explores using an epsilon-greedy approach, meaning it selects a random action with an initial probability of 0.9, which decays to a minimum of 0.05 after 200 training episodes.

Agent parameter	Value
Discount factor	0.95
Initial epsilon	0.9
Final epsilon	0.05
Decay episodes	200

Finally, we have chosen to use velocity control as the output, although this can be changed using the `VELOCITY_CONTROL` macro. At each time step, the velocity of one joint can be increased or decreased by 0.05 radians per time step. The joint position and velocity limits of the arm are also obeyed.

Control parameter	Value
Joint control type	Velocity
Velocity step (rad/step)	0.05
Max velocity (rad/step)	0.2
Min velocity (rad/step)	-0.2
Max joint angle (rad)	2.0
Min joint angle (rad)	-0.75

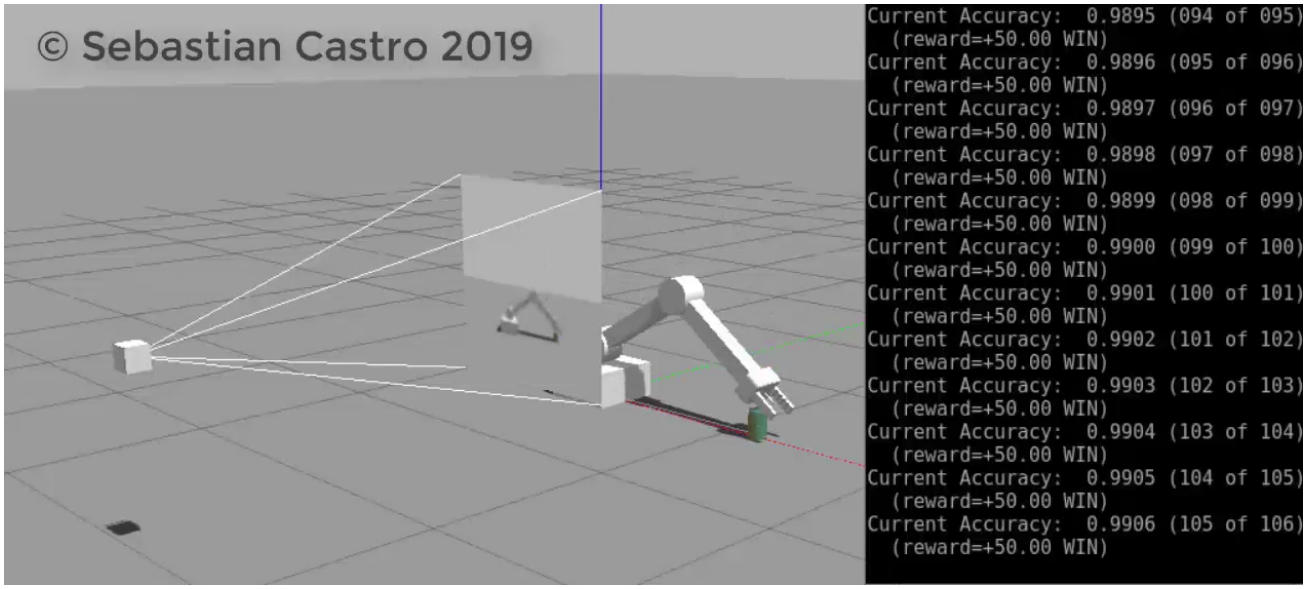
Results

Experiment 1: Any Part of Arm

In the first experiment, we allowed any part of the arm to collide with the object to consider the episode a win. After 100 training episodes, accuracy was 99% (with only 1 failure).

Experiment 1 can be enabled by setting the `GRIPPER_BASE_MODE` macro to `false` .

Refer to this video.

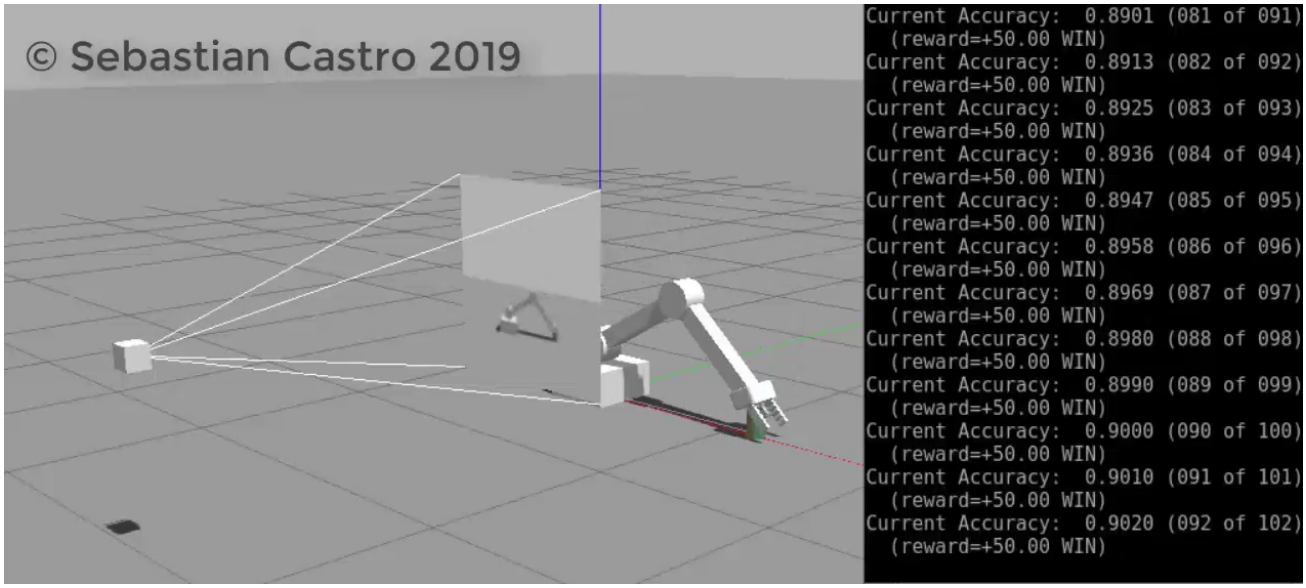


Experiment 2: Gripper Base Only

In the second experiment, we allowed only the gripper base link to collide with the object to consider the episode a win. However, a collision with other parts of the arm issued zero reward instead of the negative loss reward. This led to 90% accuracy after 100 episodes.

Experiment 2 can be enabled by setting the `GRIPPER_BASE_MODE` macro to `true`, which is the default setting in this repository.

Refer to this video.

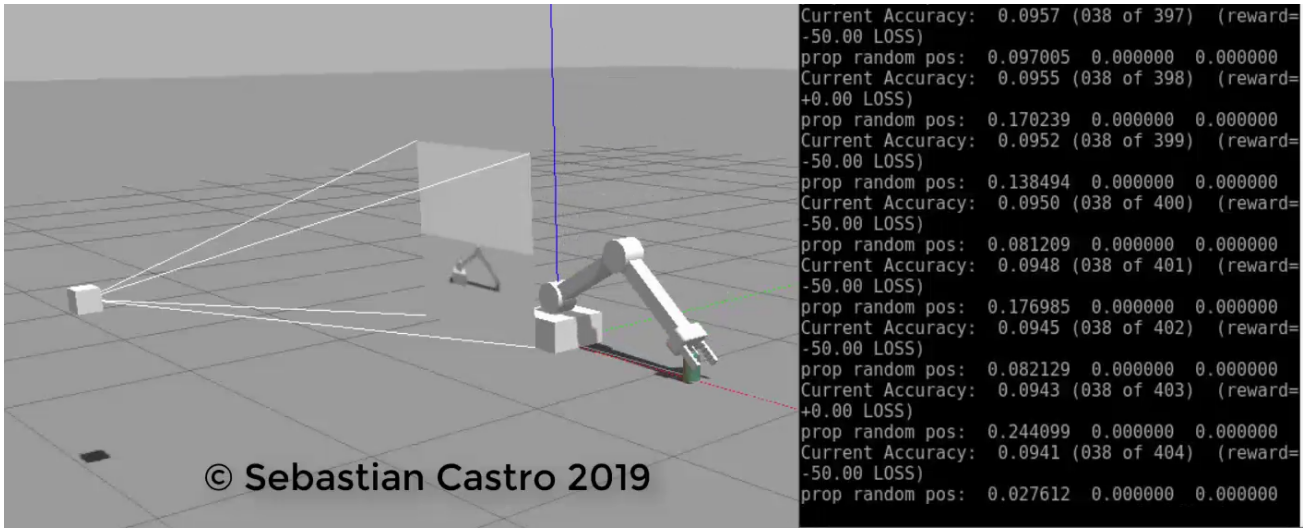


Experiment 3: Random Prop Location

We attempted randomizing the initial object location, which required more training. After 400 episodes, accuracy was still below 10%, so there is an opportunity for improvement. Refer to the next section for more details on these results.

Experiment 3 can be enabled by setting the `RANDOMIZE_PROP` macro to `true`. Notice that this conditionally increases the `EPS_DECAY` and `BATCH_SIZE` macros to allow training the RL agent for longer and on more data.

Refer to this video.



Future Work / Conclusion

Scaling Up: As seen in the results section above, the basic experiments with the 2 degree-of-freedom (DOF) arm and a static object succeeded with at least 90% accuracy. However, there is a future opportunity to train the object with random initial object location as well as moving to the full 3DOF arm case, where the arm can yaw around its base and the gripper can therefore access objects at nonzero Y locations. This will require more training steps, more data, and perhaps higher image resolution and a different neural network architecture.

Rearchitecting the DQN: Changing the default neural network structure could lead to more interesting results, especially with more complicated problems like random initial object location and higher DOF. This could mean changing the input image size, adding more convolutional layers and changing the filter parameters, and adding fully-connected layers after the convolutional layers. A bigger network in theory would lead to higher accuracy, but it would take longer to train, require more data, and potentially run into issues due to the larger memory usage.

Changing the action space: Discrete-action agents like DQN can work well for low-dimensional problems. However, a robot arm is inherently a continuous system, so discretizing leads to a loss in control precision for the arm. It would be interesting to explore other RL agents that support continuous outputs such as Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO).