Deep RL Arm Manipulation

Martin Papanek

Abstract—In this project a DQN reinforcement learning agent is connected to a robot arm simulated in gazebo. The agent learns to use the arm in order to find a specific object in the world, using only data 2D images taken by a camera in gazebo, by means of a pixels to actions approach. This paper will discuss the implementation of the rewards, the hyperparameters used by the Deep RL network and the translation from actions to robot arm motion.

Index Terms—Robot, Deep RL, IEEEtran, Udacity, LaTeX.

1 Introduction

The purpose of this project is to utilize Deep Reinforcement learning (Deep RL) to train a robot arm to touch an object that is placed close to it. Deep RL is a good choice for many robot tasks because the training can be performed using only sensor data that the robot collects as it interacts with the world. This kind of approach is called *pixels-to-actions*. In this project the robot arm is implemented in the Gazebo simulator which additionally includes a camera that is pointed at the robot arm. This camera continuously takes 2D images of the robot arm and the tube that needs to be picked up.

The type of Deep RL network used for this project is the Deep Q Network (DQN). The DQN algorithm uses a convolutional neural network to approximate the Q state-value function. The $Q(s, a)$ function gives the expected reward for choosing and action a when in state s. If a good approximation of the true Q function is available then the agent can use it to follow the best policy by always picking the action with the best Q-value in each state that it finds itself in.

To implement a DQN agent in this project, a Gazebo plugin implemented in c++ must be completed. This plugin interacts with the Gazebo simulator by subscribing to message topics. The agent requires information about collisions between the robot arm and the ground or the tube and it requires the images from the camera so it must subscribe to these topics.

2 Reward Functions

In reinforcement learning, the driving force for an agent to learn the task are the rewards. The choice of what rewards are giving to the agent by the environment and when these rewards are given is crucial, because a poor reward function can cause the agent to not learn the task correctly or to learn very slowly.

The task of the robot arm is to learn how to touch the tube object. Thus, first of all there should be a positive reward (1.0) when the robot succeeds in its task and a negative reward (-1.0) when the robot fails, for example if it touches the ground or if it doesn't manage to touch the tube within 100 frames.

Then to improve the learning speed, or in other words to make the DQN agent converge more quickly to a good approximation to the Q-function, an intermediary distance based reward R' is given to the robot after each step. This reward is calculated based on change in distance ($\Delta dist$) of the bounding box between the robot's gripper and the objects bounding box. The closer the robot is the bigger the reward should be, and this will be expressed by the delta distance which is bigger if the robot moves further from the object than he was before, and smaller if the robot moves closer to the object. Furthermore, to ensure that the robot moves smoothly towards the target object, the interim rewards are computed with a smoothed moving average of all the interim distances. The smoothing parameter α determines importance of individual distance samples. If $\alpha = 1$ then the robot does not use interim rewards at all. If $\alpha = 0$ then the robot's rewarded at each step with its current distance to the target. In the project an α of 0.5 was used.

$$R'(\Delta dist) = R'(dist) * \alpha + \Delta dist * (1 - \alpha) \quad (1)$$

The choice of the reward function depends heavily on the type of robot control that is employed. In this project there was a choice between position control versus velocity control. In position control the joint positions are incremented directly based on the action dictated by the DQN agent. In velocity control the joints moved by applying velocities and the directions of the velocities and their magnitudes are given by the DQN agents chosen action. For this robot, the velocity control was used because empirically it was performing better with much quicker convergence. However, an adjustment was made to velocity control. When incrementing velocity after each action, the arm would often crash too quickly. Instead, just setting the velocity either positive or negative based on action sign has worked much better.

3 Hyperparameters

Hyperparameter are parameters used to adjust the DQN network. These are different than the weights that the DQN learns in order to approximate the Q-function. For this project there are a number important parameters, which will be discussed in the next sections

3.1 Input size

The input size is the size of the image that is passed to the convolutional neural network. If this size is set too large then the network will have a lot of detail in the image which will slow down the convergence of the network. If the size is too small then there may not be enough detail.

The starting value in this case was 512 which was too much. It was changed to 128 and the speed to reach good accuracy of the network increased dramatically and prevent any out of memory errors in PyTorch.

3.2 Optimizer and Learning rate

This is the type of optimizer that the neural network will use during back propagation to decrease the cost function. In this case the RMSprop optimizer was chosen.

A learning rate of 0.01 was chosen, and it performed well, so there was no reason to change it.

3.3 Replay Memory, Batch Size and LSTM

The replay memory solves the issue of moving target for the DQN agent. In this case we use a replay buffer (experience buffer) of size 10000 episodes. The batch size controls the sizes of batches fed to the neural network. A value of 8 worked well.

The Long-Short-Term memory allows the Deep Q network to use historical data during training, in a way similar to a recurrent network. To improve the performance of the network, I turned on LSTM and set the history that it can remember to 32.

4 Results

If the agent losses a lot in the first few episodes then it will take longer to stabilize and start winning consistently. However, once the agent stabilizes, and the ϵ-greedy exploration rate decreases sufficiently, then the robot arm is almost always able to locate the object which is a very good result, considering that it only used 2D image data to learn what to do.

In the case where the object can be touched by any part of the arm the accuracy was very good as can be seen from the figure 1. Moreover, it doesn't take very long (around 100 episodes) for the agent to converge to a good Q-function which has above 90% accuracy.

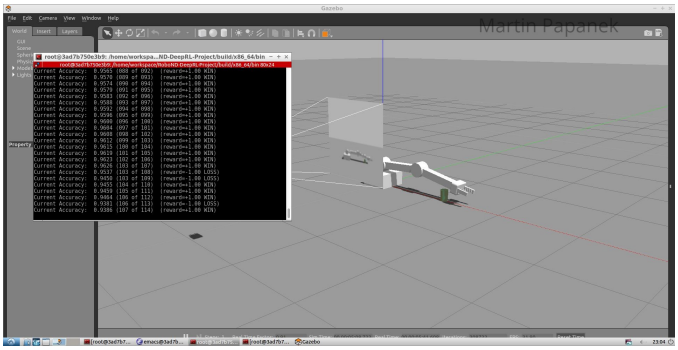

Fig. 1. 90% accuracy to touch with any arm part

The harder task, which is to touch the tube specifically with the gripper base of the arm, takes considerably longer to reach convergence. During testing it took more than 400 episodes to reach above 80% accuracy. The results for this task are shown in figure 2.

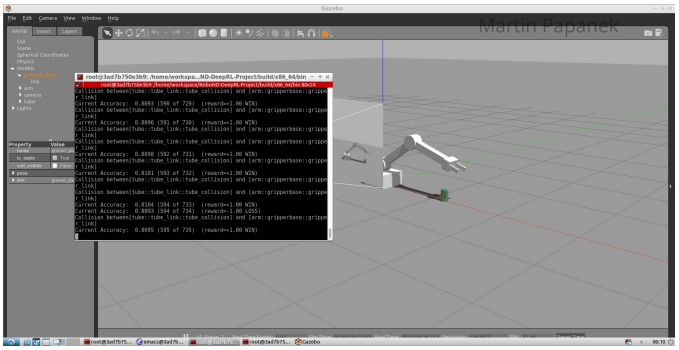

Fig. 2. 80% accuracy to touch with gripper base

5 Future Work

To further improve the results obtained, further hyperparameter adjustments could be done. Also a big factor in the performance of the agent is the reward function that is used. While the smoothed moving average interim reward function performed quite well to guide the robot arm towards the target object, I did not try any other interim functions. Tweaking the smoothing parameter is another avenue that could be explored to improve the results, specifically to improve the convergence speed.