Deep RL Arm Manipulation

Alfonso Sanchez

Udacity Robotics Program

January 29, 2019

Abstract

The use of Deep Reinforcement Learning in robotics allows the training of robots to implement specific tasks while reducing the costs of programming robots to perform different actions. Reinforcement learning helps the robot, or its agent, to determine which actions to implement based on environment observations and positive and negative rewards. The use of deep learning techniques helps the agent to find the best way to solve the reinforcement learning problem based on experiences. In this work is presented the implementation of deep reinforcement learning for the control of an arm manipulator. The arm manipulator consists of 3 degrees of freedom and can implement six different state actions. Deep reinforcement learning is used to train the robot to reach a specific object. The objective of this article is to provide the reader with an introduction to the use of deep reinforcement learning for the control of robots.

I. Introduction

In reinforcement learning, the agent observes the environment to implement specific actions to obtain rewards depending on the actions taken. One of the main ideas in reinforcement learning is that the optimal action-value q_* helps to determine the best policy π_* to follow; to find the best way to solve the reinforcement learning problem. This is possible using Q-learning algorithms, which are reinforcement learning techniques that find the policy that tells the agent which actions to take under specific circumstances. Some of the essential characteristics of Q-learning method is that are Temporal Difference approaches (TD) [1], the environments do not need to be fully defined, and action-values are estimated from observations.

Deep reinforcement learning uses a deep neural network to represent the agent in reinforcement learning; the network learns the best way to solve the problem based on experiences, instead of on feedbacks to the agent that let it know how well the problem was solved. The Deep Reinforcement Learning or Deep-Q-learning (DQN) [2] algorithm uses a constitutional neural network to combine Deep Learning with Reinforcement Learning. In DQN, the deep neural network act as an approximation function, generating a vector of possible actions as output to determine which action to take depending on the scenario presented, learned based on experience.

Reinforcement learning is used in robotics [3, 4, 5], where the previously mentioned Q-learning and DQN techniques are used. In the DQN case, it is used to learn optimal policies from high dimension sensory inputs. In the following sections is presented more information about the Q and DQN algorithms, a DQN implementation for the control of a robotic arm manipulator, a discussion about the obtained results and future work related with the developed work.

II. Methods

i. Q-learning

Q-learning is a model-free algorithm; it can explore environments that are not fully defined. It is considered as a temporal difference algorithm, given that a change of states is learned under the assumption that the states are sequential or time-based. Then, the values for each state-action pair are estimated considering observations obtained from a specific environment. The following equation represents this algorithm:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) * Q(s_t, a_t) + \alpha * lv$$

where,

$$lv = r_t + \gamma + max_a Q(s_{t+1}, a)$$

In the equation above $Q(s_t, a_t)$ represents the old and updated Q-values, α represents the learning rate, and lv the learned value. Additionally, r_t represents the reward, γ the discount factor and $max_a Q(s_{t+1}, a)$ the optimal feature-value estimated. The learning rate and the discount factor are values between 0 and 1. The learning rate is used to express the portion of new information considered to calculate the next q-value on each state, while the discount factor is used to represent the portion of the future reward that influences the new Q-value at each time step.

ii. DQN

A difference between the Q-learning and DQN algorithms is that DQN generates a Q-value for all possible actions to take, in contrast with one Q-value generated at each time obtained from the Q-learning algorithm. This characteristic for the DQN algorithm is possible because of the use of a constitutional neural network that acts like he agent. In the simplest way, the objective of DQN is to train a policy that maximizes the discounted commutative rewards, such as:

$$R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$$

As mentioned previously, in the Q-learning algorithm the main idea is to construct an optimal policy that maximises the rewards:

$$\pi^*(s) = argmax_a Q^*(s, a)$$

where Q^* is a function that maps the product between the actions and states to a space composed by real numbers:

$$Q^* : states \times action \rightarrow \Re$$

However, the Q^* function is unknown, given the uncertainties generated by the limited available information from the world. Nonetheless, the use of neural networks as approximations helps to train a function that resembles Q^* []. For the training update rule it is used the fact that for every Q function for some policy follows the *Bellman* equation:

$$Q^\pi(s, a) = r + \gamma Q^\pi(s', \pi(s'))$$

The difference between the two sides of the previous equation is the error to minimize the training of the DQN algorithm:

$$\delta = Q(s, a) - (r + \gamma max_a Q(s', a))$$

III. Results

In this work the DQN algorithm was implemented in a robotic arm manipulator with 3 DOF and with six possible state actions to perform, as presented in figure 2, using a camera to obtain images for the actual robot position and a specific node to detect collision between a particular object and the robotic arm.

Figure 1: *Robot arm manipulator used in this project.*

In the figure above the camera used to obtain images from the robot arm manipulator is represented by a cube, while a green cylinder represents the object of interest. The arm manipulator is composed of 3 joints, two links and the gripper, as presented in the next figure. Furthermore, the robotic arm is capable of performing 6 different movements.

The training of the robot to find the location of an object of interest was implemented for two different cases. The first case was designed for any part of the robot arm manipulator to make contact with the green tube, while the second case was implemented for a contact of the gripper-base with the object of interest. The DQN algorithm was performed based on a Pytorch convolutional neural network. The

Figure 2: *Robot arm manipulator used in this project.*

rewards and hyperparameters used on each of the previously mentioned cases are presented next.

For the first case, rewards were based on the collision between the second link and the object of interest; the link between the second joint and the gripper-base and the tube used to represent the object of interest. If a collision happens, the agent gets a positive reward of 10; otherwise, the agent receives a negative reward of -5. On the other hand, for the second case, the reward given was a positive reward of 10 if a collision happens between the gripper-base and the green tube; otherwise, a negative reward of -5 was given.

For both mentioned cases, there was a given inner reward, based on the distance between the object of interest and the middle of the gripper. The amount of compensation given was based on the moving average of the mentioned distance, as presented in the next equation:

$$avDist = (avgDist * \alpha) + (Dist * (1 - \alpha))$$

where *avDist* is the moving average distance, *Dist* represents the distance between the gripper and the tube and α represents the degree of weighting decrease, a constant smoothing factor between 0 and 1.

Moreover, a negative reward of -5.8 was assigned if the gripper of the robotic arm manipulator reaches the ground. This amount of reward was allocated to distinguish when the robotic arm reached the ground from the case when there is not a collision between the desired part of the robotic arm and the object of interest. The hyperparameters used in the mentioned cases are presented in table 1.

Table 1: *Hyperparameters used.*

Parameter	**Case 1**	**Case 2**
INPUT_WIDTH	512	512
TINPUT_HEIGHT	512	512
OPTIMIZER	RMSprop	RMSprop
LEARNING_RATE	0.001	0.002
REPLAY_MEMORY	1000	1000
BATCH_SIZE	32	32
USE_LSTM	true	true
LSTM_SIZE	256	256
α	0.5	0.5
REWARD_WIN	10	10
REWARD_LOSS	-5	-5

The small values used for the learning rate hyperparameters were selected considering the available GPU time in the Udacity virtual environment since the most significant learning rate takes more time to train the robot. Thus, since for the first case, a better performance regarding the accuracy was required, the smallest learning rate was used when compared with the second case, where it was expected a small accuracy. The rest of the hyperparameters were selected based on the example presented in the Robotics Software Nanodegree program. The results obtained for each of the mentioned cases with the presented hyperparameters are shown in figures 3 and 4, for cases 1 and 2, respectively.

Lastly, velocities and movements of the arm joints were considered for the control of the robotic manipulator. The control could be implemented in two different ways; If actions taken were even, velocities and joint positions could increase for each of their corresponding DOF; otherwise, velocities and joint positions could decrease. However, velocities and joint positions were maintained within a predefined range. Increments in velocities and joints positions were possible using vector arrays to store these parameters for each of the DOF of the robot arm.

In this project, the robotic arm manipulator was controlled by changing the arm joints positions, as explained in the previous paragraph. The amount of change, positive or negative as mentioned before, was defined by the actionJointDelta variable.

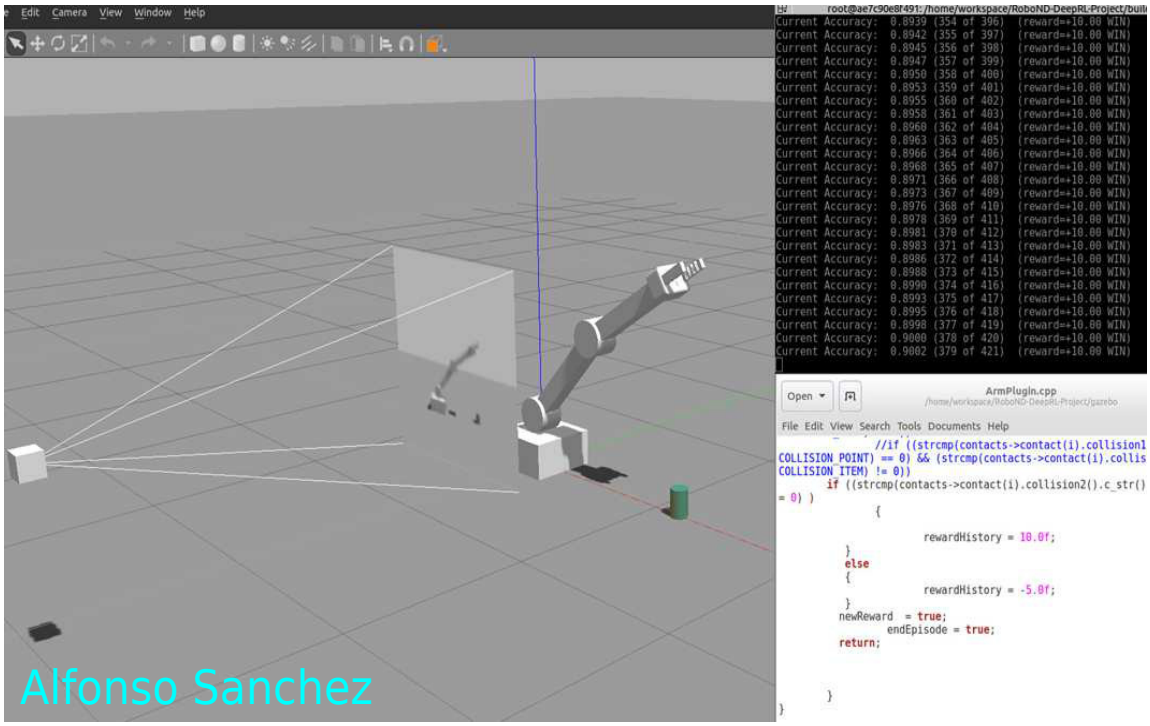

Figure 3: *Results obtained for the first presented case.*

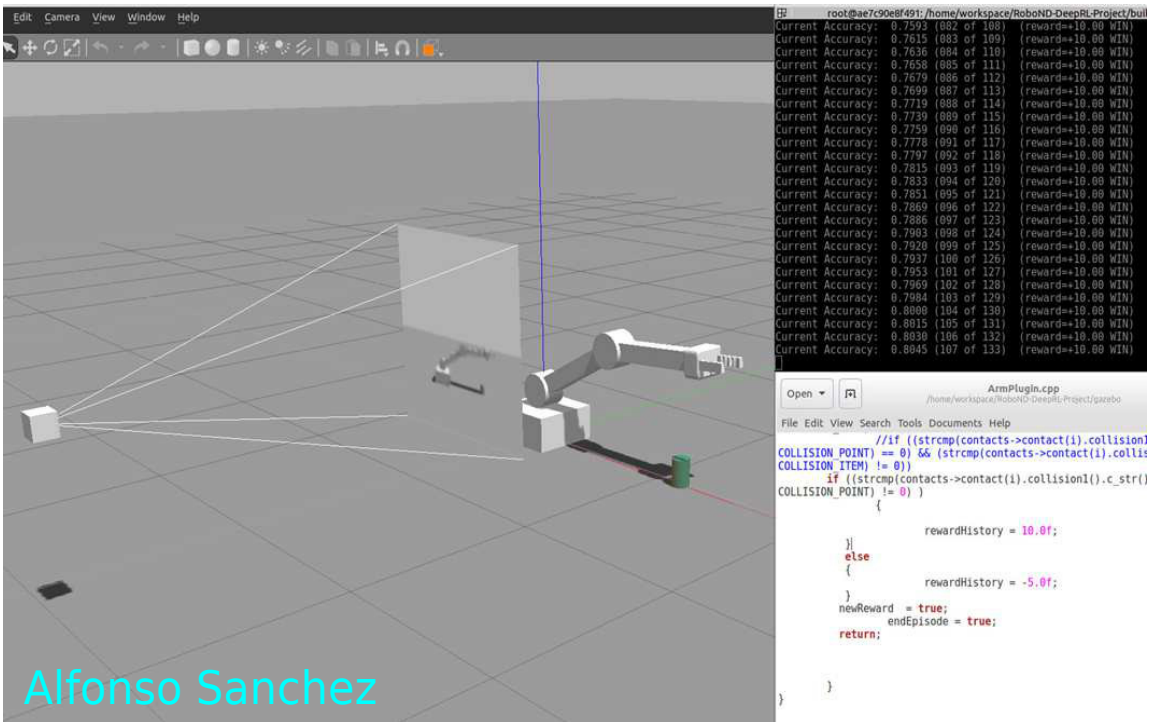

Figure 4: *Results obtained for the second presented case.*

IV. Discussion

The use of DQN to train a robot arm manipulator to learn new actions based on specific positive and negative rewards is possible and very practical. However, it can consume a lot of time if the hyperparameters are not tuned correctly. Furthermore, it consumes a lot of memory when compared with other methods, especially when compared with the manual programming of the robots to perform the desired actions. Nevertheless, the balance of positive and negative characteristics is favorable to the use of DQN, as proved with the presented cases. In both cases, the robotic arm manipulator was able to achieve the desired goal after an average of 300 runs.

In the presented results the time used for the robot to achieve the desired accuracy was less for the second case, even considering the desired accuracy to reach. It was expected for the performance of the robotic arm manipulator to be better in the first case, since the goal was for any part of the arm to touch the object of interest, when compared with the second case when only the collision between the base of the gripper and the green tube was considered for a positive reward.

V. Future work

It is important to realize that even when the results obtained were favorable; they are not good enough. The training of the robot can be improved considerably, especially for the first presented case where the collision of any part of the robot with the object of interest is considered as a positive reward. Therefore, it is required to implement a more exhaustive training of the robotic arm to achieve the desired goal with a better performance. For this, a better tuning of the hyperparameters is necessary, considering the effects of not thoroughly tested hyperparameters such as the size of the acquired image, the batch size, the α value, among others.

References

[1] Harm van Seijen. "Effective multi-step temporal-difference learning for non-linear function approximation". In: *arXiv preprint arXiv:1608.05151* (2016).

[2] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[3] Fangyi Zhang et al. "Towards vision-based deep reinforcement learning for robotic motion control". In: *arXiv preprint arXiv:1511.03791* (2015).

[4] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[5] Stephen James and Edward Johns. "3D simulation for robot arm control with deep Q-learning". In: *arXiv preprint arXiv:1609.03759* (2016).