Deep Reinforcement Learning Arm Manipulation

Carlos E. Sanoja

Abstract—The high dimensionality of the control space of robots makes training with real-world experiments with end-to-end solutions impractical. In this project is analyzed the behavior of the control arm using deep reinforcement learning to learn from raw sensor data (to detect collisions) through simulation. Finally, it is shown the effect of hyper-parameters in the Q-Network and the function reward design to successfully teach the arm to control joints to achieve the goal.
. We also present preliminary results in direct transfer of policies over to a real robot, without any further training.

Index Terms—Robot, Udacity, Control, Manipulator, DQN.

1 Introduction

TRADITIONALLY, robot arm control has been solved by hand-crafting solutions in a modular fashion, for example: 3D reconstruction, scene segmentation, object recognition, object pose estimation, robot pose estimation, and finally trajectory planning. However, this approach can result in loss of information between each of the modules resulting in accumulation of error, and it is not flexible for learning a range of tasks, due to the need for prior knowledge [1].

Implementation of deep learning have allowed to apply control being learned just from images in an end-to-end way. This is an approach that allow robots to learn from complex data from sensor avoiding complex algorithms programming. However, deep learning solutions needs large amount of training data to deal with high dimensionality policies search.

In this project is presented an approach that uses a given deep Q-learning agent and simulation in Gazebo to control the position of the joints of a robotic platform. The controller accepts images of the environment as its only input, and outputs motor actions for the task of touching a object, over a range of initial configurations. To encourage efficient learning, a structured reward function is designed with intermediate rewards. (Fig.1).

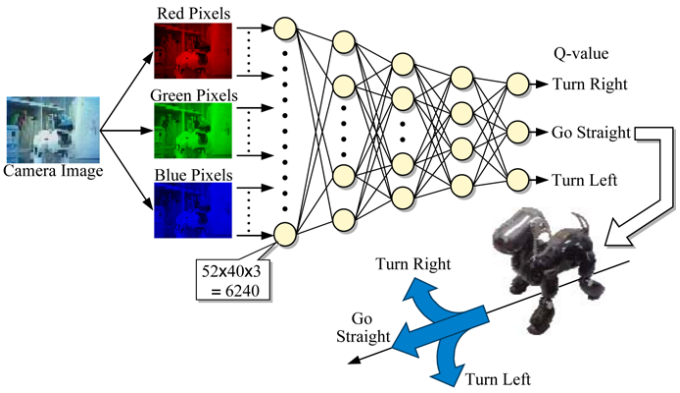

Fig. 1. Learning structure for Deep Q-learning approach.

To achieve the objectives the following task are required:

- Subscribe to topics published by Gazebo.
- Create the DQN Agent.
- Create position control function.
- Create a collision checked function.
- Intermediate rewards based on distance between arm and object.
- Created reward function for collisions.
- Tunning the hyper-parameters.

2 Reward Functions

As was mentioned before, Deep Q-Network output is mapped to a set of actions, which, for this project, are the position of the joints for the simulated arm.

A reward system was set up in order to train the robotic arm to touch the target object. A win is set for touching the prop with any link (objective 1) or the gripper base link (objective 2). A loss is returned when the arm touches the ground or when the length of the episode exceeds the maximum steps in time. Intermediate reward will be issued while robot arm is moving based on the distance from the object of interest. When collision happens with object a win reward is issued and episode is ended.

The reward function approach was designed to follow a linear function proportional to the moving average.

$$RewardHistory = RewardInterXavgGoalDelta - offset$$

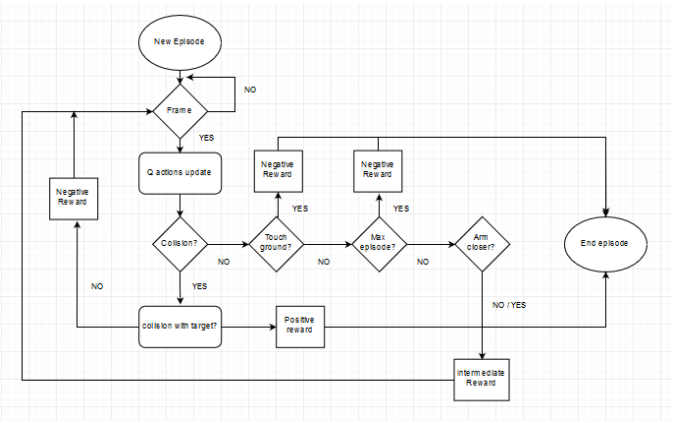

Fig. 2. Flowchart for Reward function algorithm.

Where intermediate reward is issue based on a moving average of the delta of the distance from the robot and the target object.

Set the collision check properly is crucial to obtain the desire behavior of the arm. For the objective one, it is checked that any part of the arm can collide with the target object and it is returned a positive reward. However, for the objective two, it must be checked that just the gripper collide with the target. To ensure a properly behavior, if the gripper collide it is returned a full positive reward. But, if other part of the arm have collided it is return a half negative reward.

2.1 Reward Values

- **Win Reward:** it is the number of points that will be assigned if the robot touch the target object.
- **Loss Reward:** it is the number of points that will be assigned if the robot don't touch the target object. It is reduce to the following situations: the robot hits the ground or exceed the number of iterations per episode.
- **Intermediate Reward :** it is the number of points that will be assigned based in distance from the robot to the object.
- **Alpha:** this is a smoothing factor to control average distance. Initially was set to 0.9 but, the performing of the network wasn't optimal. After try the same network for three different values, this value was set to 0.3
- **Offset:** It was added to give a range of uncertainty to the linear function. As the delta of distance can be mapped to be in [0.2, 1.8], this value aims to be a regularized factor.

Parameter	Objective 1	Objective 2
Win Reward	4	4
Loss Reward	-4	-4
Intermediate Reward	4	1.7
Alpha	0.3	0.3
Offset	0.14	0.14

Fig. 3. Reward parameters.

3 Hyper-parameters tunning

The first time the network was used, its performance was poor (0.2/1). To start to perceived better results, the following parameters were adjusted.

- **Input_Widht X Input_Height:** every camera frame is fed into the DQN agent to make the prediction. These parameter decided the size of the inputs. The initial values (512 x 512) were to big and they were memory inefficient.
- **Optimizer:** When the cost function for a neural network is selected, the goal is to minimize this cost function. For this optimization problem, it is used gradient descent or other variants of it where the model parameters (here weights and biases in the network) are updated in a way to decrease the cost function. For this project were tested Adam and RMSprop. Adam was choose because learns fastest and is more stable (doesn't suffer any major decrease in accuracy) than RMSprop.
- **Learning_rate:** In simple words learning rate determines how fast weights (in case of a neural network) change. If learning rate is too high derivative may miss the 0 slope point or learning rate is too low then it may take forever to reach that point.
- **Num_Actions:** For this project and as was selected to control the robot through joint positions, there are two possible actions: increase or decrease the joint position. The robot have 3-DOF, thus the number of actions are six.
- **Replay Memory:** DNN is easily overfitting current episodes. Once DNN is overfitted, its hard to produce various experiences. To solve this problem, Experience Replay stores experiences including state transitions, rewards and actions, which are necessary data to perform Q learning, and makes mini-batches to update neural networks [2]. This number was selected to be 12000 which means it will be possible to store 375 states that can be reused randomly.
- **Batch Size:** refers to the number of training examples utilized in one iteration. Which means, bigger batch size reduced the number of iterations needed. However, apply this techniques is to become more memory efficient, bigger sizes mean more memory and computing cost. For this project it was selected to be 32.
- **Use LSTM:** Enabling this parameter allow training the network by taking into consideration multiple past frames from the camera sensor instead of a single frame. For this project is required to be enabled.
- **LSTM Size:** This value may cause memory problem if high values are selected. For this project was selected 256 and it achieved the goal correctly.

Parameter	Objective 1	Objective 2
Input Widht	64	64
Input Height	64	64
Optimizer	Adam	Adam
Learning rate	0.01	0.01
Num Actions	6	6
Replay Memory	12000	12000
Batch Size	32	32
Use LSTM	TRUE	TRUE
LSTM Size	256	256

Fig. 4. Q-Network hyper-parameters.

4 Results

Both objectives were achieved. It is important to note that both agents were trained with the same hyper-parameters.

The only parameter that was changed is *Intermediate reward* who affect directly how the arm reach the target object while it is in the same episode based in the distance between them.

The agents were tested in 200 cycles and both of them accomplish the goal satisfactorily as shown in Fig. 5 y Fig. 6.

- Objective one: 97% accuracy
- Objective one: 83% accuracy

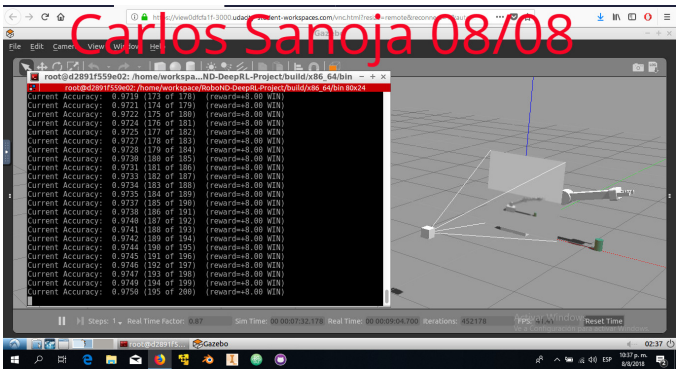

Fig. 5. DQN performance for objective 1.

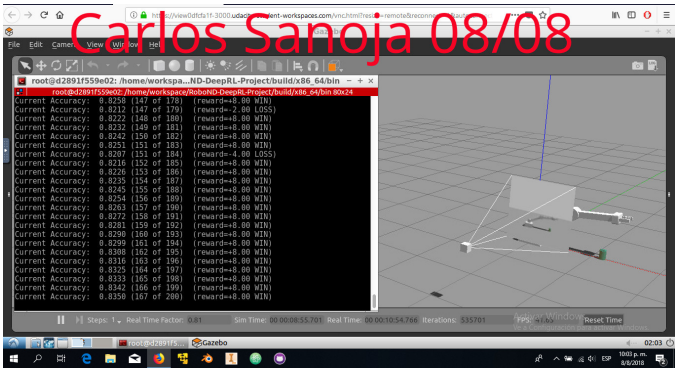

Fig. 6. DQN performance for objective 2.

Finally, the manipulator can be observed while training in the following path: ***https://youtu.be/x4nqyPY7mwI***

5 Future Work

The deep reinforcement learning framework can be improved in several ways:

- Tuning the other hyper-parameters of the network
- Analyzing another type of reward functions based on their properties. EX: polynomials or exponential functions.
- Identify the learning time of the combination of parameters
- Use search algorithm like Genetic algorithms to find a proper combination of parameters. Maybe can be used a DQN to feed another DQN with the values of the parameters as the Q values output.

6 References

1) 3D Simulation for Robot Arm Control with Deep Q-Learning . Stephen James, Edward Johns
2) Reinforcement Learning for Robots Using Neural Networks, 1993
3) https://www.quora.com/What-is-the-learning-rate-in-neural-networks
4) https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2
5) https://towardsdatascience.com/welcome-to-deep-reinforcement-learning-part-1-dqn-c3cab4d41b6b