

Deep Reinforcement Learning: Arm Manipulation

Beshari Jamal

Abstract—This outline aims to apply deep reinforcement learning principles as a replacement for traditional control methods. Deep Reinforcement Learning is a branch of deep learning and a new researched field. It has many different applications in robotics and other disciplines. In this project, a robot successfully learns by reward and punishment, a set of new behaviors to achieve a final goal with high accuracy and consistency, it learns not to make contact with the ground, and to only contact a specific object, as an initial phase of picking it up later.

Index Terms—Robot, IEEETran, Udacity, L^AT_EX, deep learning.

1 INTRODUCTION

REINFORCEMENT learning is an area of machine learning, beside supervised and unsupervised learning. It refers to goal-oriented algorithms, which can master achieving ambitious objectives, or maximize a “reward” over many iterations. During the iterations, an agent -an entity that attempts to optimize reward- gets rewarded episodically or immediately based on its performance, eventually to teach itself to achieve a task. Reinforcement algorithms combined with deep learning beat world title holders at the game of Go as well as experts at various Atari video games.

In this project, An agent teaches itself to touch an object of interest using deep reinforcement learning. The project requires to design an agent with reward functions that motivate him to reach an object of interest with any of its links with 90% accuracy and to develop another reward function to touch the same object with only the gripper 80% of the time.

2 BACKGROUND

2.1 Reinforcement Learning

Reinforcement comes originally from behavioral psychology, which means a reaction that strengthens an organism’s future behavior. In computer science and robotics, an agent can take an action (out of a set of possible actions) based on a specific instance of the environment, (also known as environment state). After an agent makes an action, that action causes a change in the environment (new state), and it receives a reward or a punishment accordingly.

After several actions or iterations, an agent attempts to find a correlation between the environment states and its actions to maximize its rewards. In Q-learning, this is performed by estimating the value of state-action pairs to rewards to be able to find the best policy; a policy in reinforcement learning is a function that appoints which action to take dealt a particular state of the environment. As the agent learns and maps its state-action with rewards, it attempts to find the best policy for the maximum reward.

2.2 Deep Reinforcement Learning

Deep reinforcement learning is when the agent is a neural network, specifically one with many deep layers, also named a deep neural network. Deep neural networks are sets of self-learning algorithms. They learn and develop with data and samples. Convolutional layers could be used and the agent would be known as a convolutional agent. Convolutional agents can process images as input due to their convolutional properties and then map them to actions. Here, a Deep Q Learning algorithm (DQN) is applied to map state-action pairs to rewards.

Reinforcement learning algorithms have the potential to learn so much. The real power of these algorithms is from their capacity to exist in parallel on multiple chips at the same time, and practice day and night without burnout. An algorithm trained on the game of chess will have played many more games than any human could hope to complete in many many lifetimes.

3 PROJECT SETUP

3.1 Initial Configuration

The project starts with a simple arm with two joints in Gazebo. The initial configuration is shown in 1. The DQN agent can choose out of six actions; increase, decrease or do nothing on either of the two joints. Python’s package Pytorch is used to apply tensor computation with strong GPU acceleration and to build Deep Neural Networks on a tape-based auto-differentiation system. A camera is shown in the image with four straight lines coming out of it. This camera is the agent’s method to sense the environment.

3.2 Reward Functions

Many cases are considered for rewards; the goal was to have the robot touch the cylinder. Therefore, touching the cylinder resulted in receiving a score of +20 while hitting the ground resulted in a punishment of -5. Finally, a timeout punishment if the agent does not make a decision that ends the episode in 100 frames. All the rewards mentioned earlier result in episode termination.

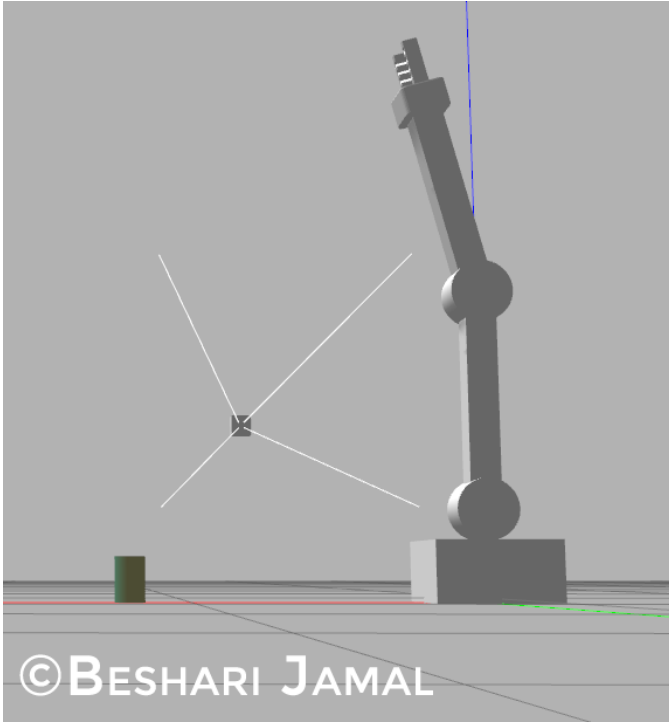


Fig. 1. Initial Environment and Agent Configurations

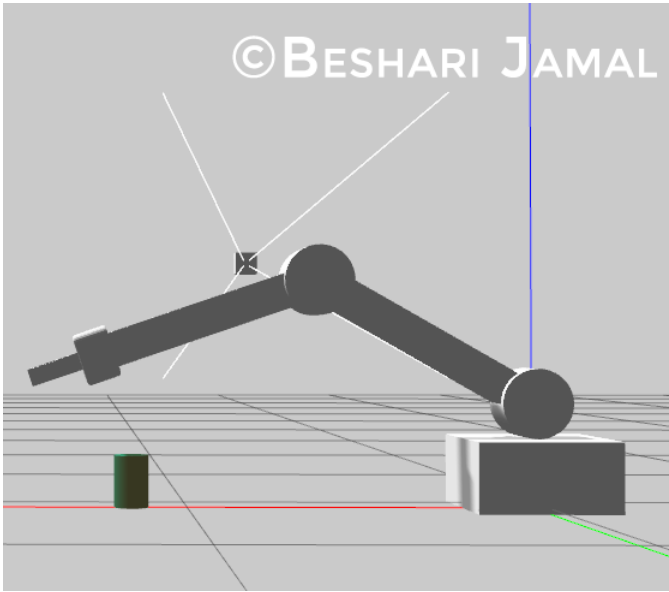


Fig. 2. Mid Environment and Agent Configurations

Additionally, an immediate reward was given defined by a function that gives a reward as the robot gets nearer to the cylinder and punishes it as gets further.

Finally for Challenge two while the robot or the agent is only allowed to touch the object of interest with the gripper, a punishment of -2 was given if the robot touched it with its second link.

The position control was implemented for both challenges, and it was found to be more accurate. Speed control had the attitude of a speed car racer smashing to the ground at best 25% of the time.

TABLE 1
Reward Parameters

Action	Reward
Touches ground	-5
Touches cylinder	20
Time-out	-50
Touches Cylinder with wrong link	-2

3.3 Hyperparameters

This agent model is full of hyperparameters that need to be tuned. The height and the width of images fed to the agents were both decreased to 64 to speed computations. Learning rate was started at 0.01 and reduced in every experiment till 0.0015 worked the best. Epsilon, the exploring percentage starts high at 90% and it decays to 0.1% after the robot learns its specific task, one would like the robot to keep its strategy more often and be more reliable. Different optimizers were used, and RMSprop optimizer proved to work the best in DRL with the current learning rate. Long Short-Term Memory networks (LSTM), a particular kind of recurrent neural networks (RRN) was used and allocated 512.

```
[deepRL] use_cuda:      True
[deepRL] use_lstm:      1
[deepRL] lstm_size:     512
[deepRL] input_width:   64
[deepRL] input_height:  64
[deepRL] input_channels: 3
[deepRL] num_actions:   3
[deepRL] optimizer:     RMSprop
[deepRL] learning_rate: 0.0015
[deepRL] replay_memory: 20000
[deepRL] batch_size:    32
[deepRL] gamma:         0.9
[deepRL] epsilon_start: 0.9
[deepRL] epsilon_end:   0.001
[deepRL] epsilon_decay: 200.0
[deepRL] allow_random:  1
[deepRL] debug_mode:    0
[deepRL] creating DQN model instance
[deepRL] DRQN: _init_()
[deepRL] LSTM (hx, cx) size = 512
[deepRL] DQN model instance created
[deepRL] DQN script done init
[cuda]   cudaAllocMapped 49152 bytes, CPU 0x101340000 GPU 0x101340000
[deepRL] pyTorch THCState 0x60FE63E0
[cuda]   cudaAllocMapped 12288 bytes, CPU 0x101540000 GPU 0x101540000
ArmPlugin - allocated camera img buffer 64x64 24 bpp 12288 bytes
[deepRL] nn.Conv2d() output size = 800
```

Fig. 3. Parameters values used for both Challenges

4 RESULTS

4.1 Challenge One

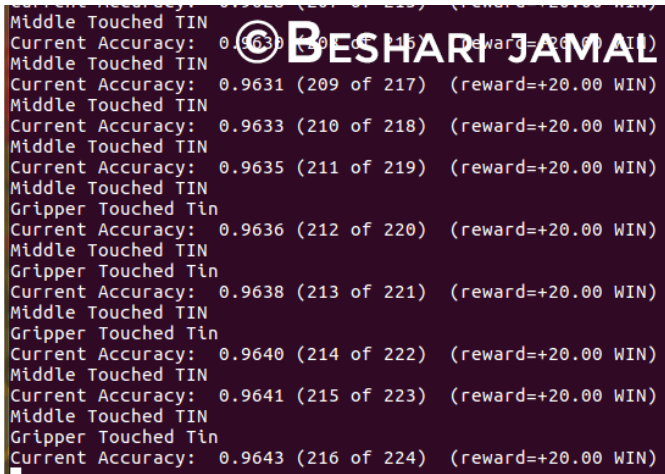
Challenge one was to touch the object of interest at least more than 100 touches with accuracy of 80%. as shown in figure 4 both requirements were met with 2811 episodes and 99.32%.

```
Touched Tin
Current Accuracy: 0.9932 (2799 of 2808) (reward=+20.00 WIN)
Touched Tin
Current Accuracy: 0.9932 (2790 of 2809) (reward=+20.00 WIN)
Touched Tin
Current Accuracy: 0.9932 (2791 of 2810) (reward=+20.00 WIN)
Touched Tin
Current Accuracy: 0.9932 (2792 of 2811) (reward=+20.00 WIN)
```

Fig. 4. Number of Episodes and Accuracy of Challenge One

4.2 Challenge two

Challenge two was to touch the object of interest at least 100 touches with an accuracy of 80%. as shown in figure 5 both requirements were met with 216 episodes and 96.43%.



```

Middle Touched TIN
Current Accuracy: 0.9630 (209 of 215) (reward=+20.00 WIN)
Middle Touched TIN
Current Accuracy: 0.9631 (209 of 217) (reward=+20.00 WIN)
Middle Touched TIN
Current Accuracy: 0.9633 (210 of 218) (reward=+20.00 WIN)
Middle Touched TIN
Current Accuracy: 0.9635 (211 of 219) (reward=+20.00 WIN)
Middle Touched TIN
Gripper Touched Tin
Current Accuracy: 0.9636 (212 of 220) (reward=+20.00 WIN)
Middle Touched TIN
Gripper Touched Tin
Current Accuracy: 0.9638 (213 of 221) (reward=+20.00 WIN)
Middle Touched TIN
Gripper Touched Tin
Current Accuracy: 0.9640 (214 of 222) (reward=+20.00 WIN)
Middle Touched TIN
Current Accuracy: 0.9641 (215 of 223) (reward=+20.00 WIN)
Middle Touched TIN
Gripper Touched Tin
Current Accuracy: 0.9643 (216 of 224) (reward=+20.00 WIN)

```

Fig. 5. Number of Episodes and Accuracy of Challenge Two

5 DISCUSSION

All objectives for both challenges were met. Nevertheless, it is important to note that in reality, even 99% accuracy is unacceptable, the agent should perform as designed 100% of the time. When the projects are started the agent seems to survey the space possibilities. This is mainly manifested in the initial five episodes as the agent is still initially and seems to explore the space by experimenting with all the possible action options. The immediate reward motivates the agent to bring the arm gripper closer to the object of interest every frame that passes. If the agent stays still, in place, it gets a 0 score, if it gets further is gets punished if closer, it gets positive reinforcement. It is noticed that having a motivating immediate reward and punishment function directs the robot smoothly to its episodic task. The arm had lots of shortcomings using the speed controller. Lots of the times the arm head to the cylinder too fast and also hit the ground and gets a negative reward of -5. If the speed is too fast, there may not be enough frames to slow down the speed, given the agent can only decrease by certain angular velocity each frame.

6 CONCLUSION / FUTURE WORK

Deep Reinforcement has shown great promise in control systems, especially end-to-end reinforcement learning, where a neural network controls the whole the process from sensors to actuators in a robot or agent.

This project did not take advantage of the experience of the agent in that every experiment, the agent had no experience from the previous one. Some improvements include that the weights of the neural network could be kept from experiment to experiment to get a highly accurate state-action pairs to value. The frames and steps could be finer for accurate control. Also, finally, the immediate reward could be correlated to a path rather than a goal. It is believed with

these improvements the robot could be reinforced even to pick up the object of interest. This project has achieved its objectives and paved the way for more fun experimentation.

REFERENCES

- [1] Yuxi Li *DEEP REINFORCEMENT LEARNING: AN OVERVIEW*. arXiv:1701.07274v5 [cs.LG] 15 Sep 2017 <https://arxiv.org/pdf/1701.07274.pdf> Accessed: July 5, 2018.
- [2] Udacity, 2017 *Robotics Software Engineer Nanodegree Program: Term2: DRL for Robotics*. <https://www.udacity.com/> Accessed: July 5, 2018.