Deep Reinforcement Learning Arm Manipulation

Gassó Loncan, J.C.

Abstract—Reinforcement learning is old well known area of machine learning with several years of history with robotics applications. It is presented the deep reinforcement learning project. The aim of this project is to create a Deep Q-Learning Network to guide a robotic arm to touch a cylindrical shape on a simulated environment. Two task were asked: (i) have any part of the robot arm touch the object of interest, with at least a 90% accuracy, (ii) Have only the gripper base of the robot arm touch the object, with at least an 80%. accuracy. Both objectives were achieved. It is explained the logic, reasoning and methods applied to met the project requirements. A valuable experience was gained on this type of machine learning algorithms.

Index Terms—Udacity, Robotic Nanodegree, Reinforcement learning, Deep learning.

1 Introduction

Reinforcement learning is old well known area of machine learning with several years of history with robotics applications. The main idea of RL has its origins on psychological and neuroscientific perspectives on animal behaviour of how agents learn from their environment. Mayor breakthroughs have been reached in the last two decades from the fusion of the RL with deep learning (DL). In 2015, the Deep Q-Learning Network (DQN) algorithm was introduced, which is responsible for taking off from the field of the deep RL [1]–[3].

The aim of this project is to create a Deep Q-Learning Network (DQN) and, with it, to enable a robotic arm to learn to perform a specific task without even knowing the kinematic model or the previous planning of the path. The robotic arm used for this project has three degrees of freedom and is simulated in Gazebo as it shown in Fig. 1. The main task is to touch a cylindrical shape on the stage. The entrance to the learning model is provided by a camera inside the scene that films the robot from the side. Since the DQN model is pre-provided, the reward functions and hyperparameters must be defined and adjusted.

Therefore, the DQN model will use the information obtained by the constant recording of the camera, trying to generate a sequence of motion by modifying the angular position of the joints to maximise the cumulative reward.

The specifications that must be met are:

1) Have any part of the robot arm touch the object of interest, with at least a 90% accuracy.
2) Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy.

The project was based on the Robotic Arm simulated environment from the Nvidia Jetson TX2 Deep Reinforcement Learning repository found at https://github.com/dusty-nv/jetson-reinforcement. The development of the porject was done in the online Udacity workspace.

2 Shared Settings

Here there were explained settings and configurations that are shared for both task.

2.1 DQN API Settings

- **GAMMA** 0.9f: was left as default because the task that the robot need to achieve it will compose by a sequence of actions before the episode ends, so the relative distant future takes relevance.
- **EPS_START** 0.9f: was left as default. A 90% of chance to make a random move at begging for *exploration*.
- **EPS_END** 0.01f: was reduced a little so the *exploitation* domain the behaviour at the end.
- **EPS_DECAY** 150: was reduced a little so the *exploitation* is reached earlier. The idea was to met the accuracy specification as fast as possible.

2.2 Joint Control

For both tasks, position control was selected. In the first instance, it is only by mere intuition that position control is considered simpler than speed control, since the latter can be defined as the derivative of the former. But as a second

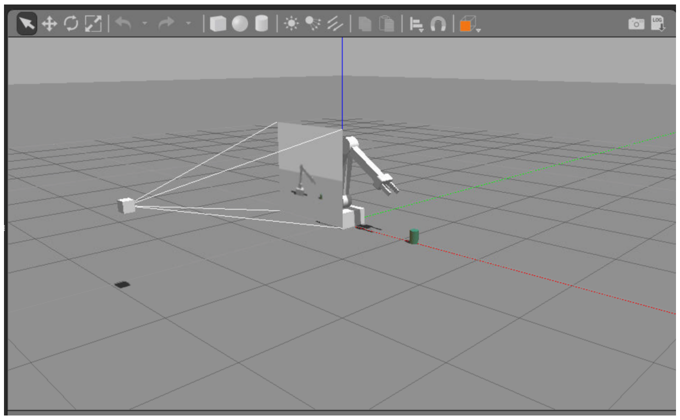


Fig. 1. Gazebo scene with the robotic arm and the cylindrical shape. ©Udacity Robotic Nanodegree.

argument, the slack forum was used and several comments were observed where they had obtained better results with the position control, so it was decided to start working with this type of control. After running some experiments it was decided to reduce a little the `actionJointDelta` and it was set as 0.12, aiming to made smother movements.

2.3 Hyper-parameters

As the second task proved to be far more difficult than the first, the only hyperparameters shared for both tasks was the input image shape and the replay memory. The image shape was reduced to 64x64 as it is enough resolution to identify the arm shape and its links positions, so on the cylinder position. In addition, higher resolution will have a considerable impact on training time.

- **INPUT_WIDTH** 64
- **INPUT_HEIGHT** 64
- **REPLAY_MEMORY** 10000

For the replay memory it was considered enough memory for the problem hence it was kept the suggested value.

2.4 Reward function

Both reward functions were quite the same. There were only a minor change on the reward function from task one to task two on the collision check.

2.4.1 Reward parameters

- **REWARD_WIN** 0.1f
- **REWARD_LOSS** -0.1f

The value of this parameters were chosen following the below code line hence it could get the all the tags `"POS"`,`"POS+"`,`"NEG"` and `"ZERO"` during debug.

```
if(DEBUG){
    printf(
    "ArmPlugin - issuing reward %f, EOE=%s  %s\n",
    rewardHistory, endEpisode ? "true" : "false",
    (rewardHistory > 0.1f) ? "POS+" :
    (rewardHistory > 0.0f) ? "POS" :
    (rewardHistory < 0.0f) ? "NEG" : "ZERO");
    }
    agent->NextReward(rewardHistory, endEpisode);
```

2.4.2 Episode timeout

If episode exceeded `maxEpisodeLength` (setted at 100), a negative reward will be issued, and the episode is ended.

- `rewardHistory = REWARD_LOSS;`
- `newReward = true;`
- `endEpisode = true;`

2.4.3 Ground contact

As the arm must not hit the ground, a large negative reward will be issued, and the episode is ended. The `groundContact` threshold was modified because the simulation tended to be a little susceptible to detect contact with the ground even after touching the cylinder.

- `groundContact = 0.01f;`
- `rewardHistory = REWARD_LOSS * 10;`
- `newReward = true;`
- `endEpisode = true;`

2.4.4 Distance to object

This part of the function was perhaps the most complex, but after several attempts it ended up being quite simple, as the more complex functions ended up having bad results. It was tested, for example, by implementing a sigmoid function to increase the reward to a limited extent until the target was reached or simply by dividing the reward by the distance and it did not work either. So it was decided to formulate something simple and clear, which finally worked. The following code snippet explains it self.

```
const float distDelta = lastGoalDistance - distGoal;
//if is positive the arm is getting closer
const float alpha = 0.7; // less sensitive to
    current delta

/* compute the smoothed moving average of
the delta of the distance to the goal*/
float average_delta  = (average_delta * alpha) +
                    (distDelta * (1.0 - alpha));
avgGoalDelta  = average_delta;

if (avgGoalDelta > 0.001f){
    // positive reward when it is getting closer
    if (distGoal > 0.0){
        rewardHistory = REWARD_WIN;
    }else if (distGoal < 0.001 || distGoal == 0.0){
    // get a bonus when it is very close
        rewardHistory = REWARD_WIN * 20;
    }
}else if (avgGoalDelta < 0.0f){
    // more punishment the further away from the
    object
    rewardHistory = REWARD_LOSS * distGoal;
}else{
    // accumulative punish if it is not moving
    rewardHistory += REWARD_LOSS;
}
```

3 TASK 1 SETTINGS

To reach objective one, it was necessary to get any part of the robot arm to touch the object of interest, with at least 90% accuracy, after at least 100 repetitions.

3.1 Collision check

As the arm must hit the cylinder with any part, there will be a large positive reward when it is touched and the episode is set as over.

```
// Cylinder collision flag
bool collisionCheck = (strcmp(
    contacts->contact(i).collision1().c_str(),
    COLLISION_ITEM) == 0) ? true : false;

if (collisionCheck){
    rewardHistory = REWARD_WIN * 25.0f;
    newReward   = true;
    endEpisode = true;
    return;
}
```

3.2 Hyperparameters

As this task was not considered difficult, and also was the first one, it was decided to choose a simple model. Luckily this task was reached in the first run.

- **OPTIMIZER** ”RMSprop”: it is one of most common optimiser and also the one used int the fruit examples provided.

- **LEARNING_RATE** 0.2f: it was decided to start with a high value and reduce it if necessary.
- **BATCH_SIZE** 32: it is a quite standard value and work.
- **USE_LSTM** true: long term memory was considered to be important.
- **LSTM_SIZE** 128: it was thought to be more than enough to this problem and work.

4 Task 2 Settings

Being more difficult than task 1, for task 2 objectives it was necessary to get only the gripper base of the robot arm touch the object, with at least a 80% accuracy, after at least 100 repetitions.

4.1 Collision check

There will be a positive reward when the arm touch only with the gripper and the episode is set as over. The greater the reward, the faster the collision is detected.

```
// Cylinder collision flag
bool collisionCheck = (strcmp(
    contacts->contact(i).collision1().c_str(),
    COLLISION_ITEM) == 0) ? true : false;
// Gripper collision flag
bool collisionCheck_G = (strcmp(
    contacts->contact(i).collision2().c_str(),
    COLLISION_POINT) == 0) ? true : false;

if (collisionCheck)
{
    if (collisionCheck_G){
        rewardHistory = (100 - episodeFrames) *
                        REWARD_WIN;
        newReward  = true;
        endEpisode = true;
        return;
    }
```

4.2 Hyperparameters

This task required several attempts and most of the hyper-parameters were chosen by trial and error. It was needed to tune almost all them as follow:

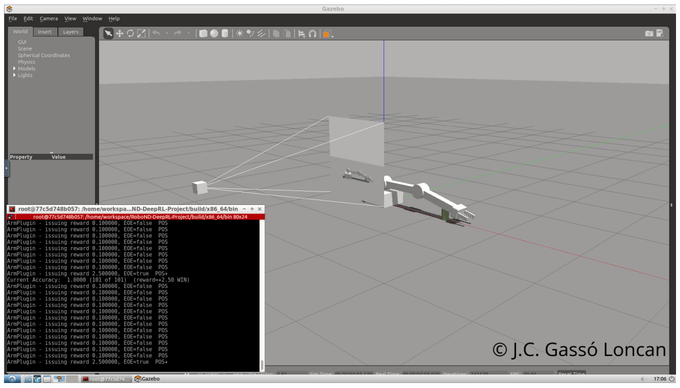
- **OPTIMIZER** "Adam": After a little research, it was decide to try Adam optimiser following [4].
- **LEARNING_RATE** 0.1f: At first was chosen 0.01 but the model don not reach the accuracy after 500 iterations, so it was set to 0.1 hoping to learn a faster, knowing the risk of falling on a local minimum.
- **BATCH_SIZE** 128: 32 samples per batch resulted in irregular movements and the arm hitting the ground. It was therefore decided to give more examples before the network weight update, hoping that the model could learn more by individual episode.
- **LSTM_SIZE** 256: As the arm movement required to be more precise it was given more LSTM units hoping the model could process more past actions and got a better prediction.

5 Results

As you can see from Fig. 2 and Fig. 3 both tasks have been completed.

5.1 Task 1

The Task 1 specifications were met on the first run right from the beginning. In this video can be seen the arm hit the object almost completely stretched out.

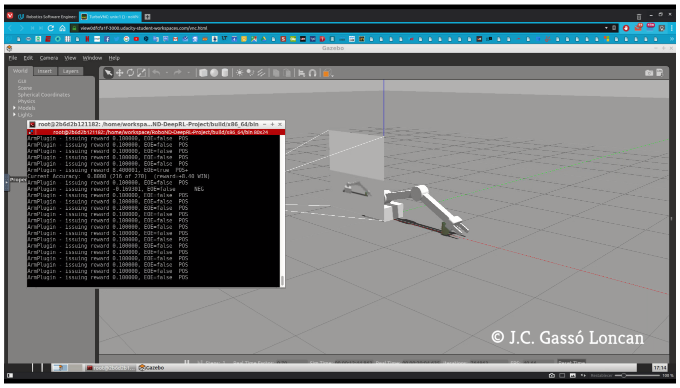


```
root@77c5d748b057: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@77c5d748b057: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 2.500000, EOE=true  POS+
Current Accuracy:  1.0000 (101 of 101)  (reward=+2.50 WIN)
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 2.500000, EOE=true  POS+
```

Fig. 2. Task 1 accuracy after 100 iterations was 100%

5.2 Task 2

Unlike Task 1, Task 2 required many attempts and time to fine-tune the hyper-parameters and reward functions. It took 270 iterations to reach 80% of the required accuracy, but in this video it could be seen that it was overcome and grows slowly. A more delicate movement can also be seen.



```
root@2b6d2b121182: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@2b6d2b121182: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 8.400001, EOE=true  POS+
Current Accuracy:  0.8000 (216 of 270)  (reward=+8.40 WIN)
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward -0.169301, EOE=false      NEG
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
ArmPlugin - issuing reward 0.100000, EOE=false  POS
```

Fig. 3. Task 2 accuracy after 270 iterations was 80%

6 Discussion

This project was one of the most time demanding of the whole the course. It shown that as like as any deep learning algorithm, DQL is very powerful and has a huge potential, but it could be tricky and hard to find the best hyperparameter tuning and a good reward function definition, that it is almost the heart of this technique.

However, even after much thinking, trial and error was the main method to achieve the goals and any revision or control of the development of the deep network taring was done. May be using tools such tensorboard could help to find easier the model settings [5]. Furthermore, LSTM are powerful networks units but could have sometimes a chaotic behaviour, so one need to be careful [6].

In addition, in both task the agent was shown to be hitting the ground hard enough to break the arm apart, at least in the early stages of the iteration. This is a problem that could be accepted for a real robot, which usually are expensive. Anyway dividing learning in two steps, first in simulation environment and then in a real robot could be a good approach.

7 Conclusion

Both specifications of the task were met. As expected, the second task was difficult to accomplish. A valuable experience was gained on this type of machine learning algorithms, having a notion of both its potential and its complexity with a simple example like the one provided.

There were many problems to compile the project both in Jetson and on a notebook, both with the udacity project and the official Jetson base repository. As the project is based on a tutorial for the Jetson TX2, the next step would be to run the simulation and training on the TX2 itself.

For future work, improve the tuning of hyper parameters and/or design a better reward function for the second target to achieve accuracy results of almost 100%. In real life, having to wait long periods of time for robots to learn how to perform tasks can be a little more inconvenient than in a simulation. In addition, it should never hit the ground or at least not hard enough to break the robot. Perhaps some limitations could be defined to avoid hitting the ground. Finally, analyzing other types of control, such as velocity, may be more complex but may give smoother movements.

References

[1] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.
[2] L.-J. Lin, "Reinforcement learning for robots using neural networks," tech. rep., Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.
[3] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.
[4] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.
[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.
[6] J. Brownlee, "On the suitability of long short-term memory networks for time series forecasting." https://machinelearningmastery.com/suitability-long-short-term-memory-networks-time-series-forecasting/. Last update: May 26, 2017.