Deep RL Arm Manipulation

Muhammad Khalaf

Abstract—Deep Reinforcement Learning is a machine learning method that extends reinforcement learning to solve problems of how agents can learn to take the best actions on the environment to get the maximum cumulative reward over time [1]. In this project, a robotic arm, the agent, is trying to learn how to reach a prop placed in a gaazebo environment, using a C++ API for a fast real-time response to the environment.

1 Reward Functions

1.1 Objective #1

In the first objective, it is required to have any part of the robot arm touch the object of interest, with at least a 90% accuracy for a minimum of 100 runs. The rewards given to the robotic arm were changed according to the behavior of the arm. But mainly there are three important cases.

1.1.1 touching the object

When the arm touches the object, that means its job is done. Hence, it is given a reward with value = 10, i.e. REWARD_WIN = 10.

1.1.2 hitting the ground or timeout

When the arm hits the ground or more than 100 frames have passed, it is given a penalty with value = -100, i.e. REWARD_LOSS = -100.
This value is much bigger than REWARD_WIN because the arm was hitting the ground very often, so it was a high penalty to discourage the arm from doing the same thing again with this high rate.

1.1.3 distance to object

An interim reward is given to the arm based on its distance to the object. The reward is calculated using smoothed moving average of the delta of the distance to the goal.

```
const float distDelta = lastGoalDistance -
    curGoalDistance; // if arm gets farther
    from goal, distDelta => -ive
// compute the smoothed moving average of
    the delta of the distance to the goal
float alpha = 0.2;
avgGoalDelta = (avgGoalDelta * alpha) +
    (distDelta * (1 - alpha));
reward = avgGoalDelta;
```

As a result the arm is urged to get closer to the goal. However, sometimes the arm didn't move, and this happens occasionally after it hits the ground. It seems that the arm is somehow afraid of getting too much closer to the ground, and hence it is not getting closer to the object also. So it is stuck, like in the next figure. The reward value is written on the left, and it shows that the robot is not moving, neither towards the object, nor away from it.
So to solve this problem, two solutions were introduced.

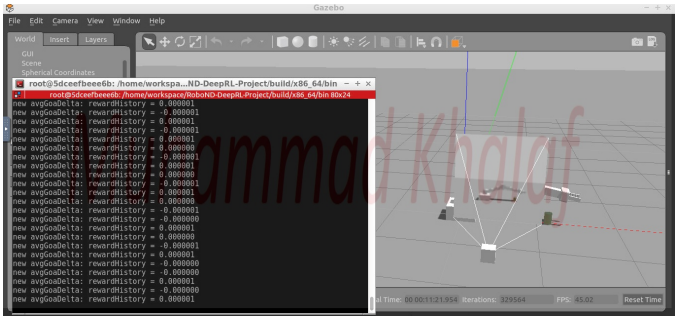

Fig. 1. the arm is not moving, afraid of hitting the ground

Firstly, the reward is multiplied by 25, to add a bigger weight to move towards the goal.

```
reward = 25*avgGoalDelta;
```

Secondly, when the delta of the distance to the goal is very small, a penalty of -1 is issued.

```
if (abs(distDelta) < 0.00001)
  reward = -1;
```

1.2 Objective #2

The second objective is to have only the gripper base of the robot arm touch the object, with at least a 80% accuracy for a minimum of 100 runs. Rewards were changed again to accommodate to the new objective.

1.2.1 touching the object

If the gripper touches the object, the reward is +100. REWARD_WIN is multiplied by 10 to encourage the arm to do the same action again, as it happens rarely at the beginning. While it is given a penalty = -5, if it touches the object with any part other than the gripper. It is a relatively small value, so as not to make the arm move away from the object in the next episodes.

1.2.2 hitting the ground or timeout

The penalty given here was only -10. Again, a relatively small value to encourage the gripper to reach the object.

1.2.3 distance to object

The same approach used in the first objective is used here also.

2 HYPERPARAMETERS

2.1 Objective #1

2.1.1 Input Height & Width

At first the INPUT_HEIGHT & INPUT_WIDTH were set to be 64*64. These small square image dimensions make calculations a lot faster for the GPU. When the arm was hitting the ground so often, the resolution is changed to be 256*256. But this actually made things worse,it was harder for the arm to learn from this huge input space. So it is fixed with 128*128, and other techniques were followed to fix this problem, like the above mentioned solutions.

2.1.2 Optimizer

The optimizer used was "RMSprop", which is a very good optimizer for recurrent and LSTM networks. RMSprop is an adaptive learning rate optimization algorithm, and it has a good technique to compute gradients so it is very useful to speed up neural networks [2].

2.1.3 Learning Rate

The learning rate used here is set to 0.01, this value was chosen using trials and errors with lr values of 0.1, 0.01 and 0.001.

2.1.4 Other Hyperparameters

The REPLAY_MEMORY size is set to 10000, BATCH_SIZE = 256 and LSTM is used with a size of 256.
With these values the arm performed well and very fast. The chosen sizes did not affect the speed of GPU.

2.2 Objective #2

2.2.1 Input Height & Width

Input size is canged from 128*128 to 64*64, to make it easier for the arm to reach the object fast.

2.2.2 Optimizer

The RMSprop optimizer is also used here.

2.2.3 Learning Rate

The same value of 0.01 is used.

2.2.4 Other Hyperparameters

The REPLAY_MEMORY size is set to 20000, BATCH_SIZE = 128 and LSTM is used with a size of 256. these values were chosen again using trials and errors.
The arm was sticking with the actions with the maximum rewards. It sometimes tries different bad actions but it returns immediately and chooses the right ones.

3 RESULTS

3.1 Objective #1

The position control method was adopted. In the first objective, the arm performed very well, with an accuracy of 97%. The task was very easy to the arm once it hits the object for the first time and guarantees a +10 reward.

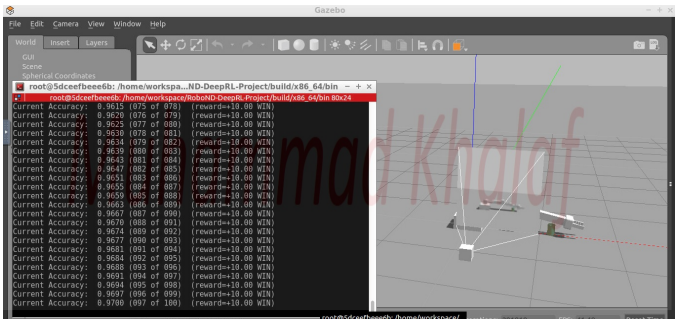

Fig. 2. Objective #1 accuracy. The arm is heading towards the object.

3.2 Objective #2

Here it was a lot more difficult for the arm to touch the object with its gripper. It usually hits the ground or touches the object with other links like it does in the first objective. So the motion was tweaked a little bit. The actionJointDelta, which is the delta of position of a joint, was changed from 0.15 to 0.075, It makes the arm motion very smooth without hitting the ground very often.
After many trials the arm was able to reach the object with its gripper, and achieved an accuracy of 82%.

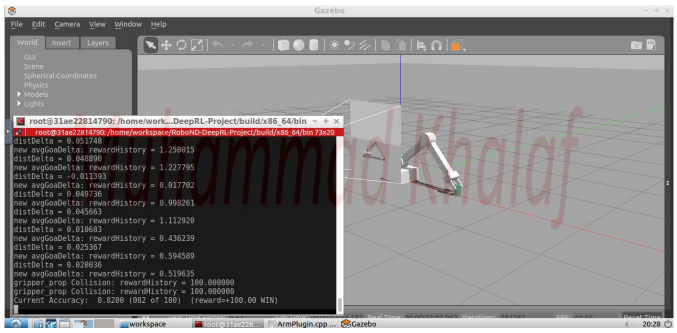

Fig. 3. Objective #2 accuracy. And the gripper is touching the object.

3.3 Future Work

For the second objective, there are a lot more to do to improve its results. For example, different optimizers with different hyperparameter values could perform better.
It is noticed that the accuracy is affected badly because of hitting the ground. From the beginning of the first episode, until the arm reaches the target for the first time, say in episode number n, most of the episodes from episode_1 to episode_n ends because the arm is hitting the ground.
So another technique that could have been introduced to solve this problem of hitting the ground very often, is to add a new penalty if the gripper approaches the ground, without hitting it. And in this situation the arm could be penalized with say 0.5*REWARD_LOSS, and keeps the end of episode flag to false as the arm hasn't hit the ground yet.

REFERENCES

[1] Wikipedia, "Reinforcement learning." https://en.wikipedia.org/wiki/Reinforcement$_learning$.
[2] Deeplearning.ai, "Rmsprop (c2w2l07)." https://www.youtube.com/watch?v=$_e-LFe_ignopbjreload = 10$.