Deep RL Arm Manipulation project

By : Hamza CHETTOUR

i. Introduction :

The project's goal is to create a DQN agent to make a robot arm perform the following tasks :

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

To achieve them, the reward functions and control logic must be created too and hyperparameters tuned.

We will only discuss the reward functions, control logic and parameters tuning as the creation of the agent is pretty straightforward.

ii. Reward functions :

There are many strategies to teach the arm to touch the object and ultimately touch it with only the gripper base.

Our strategy was to split the goal to several small ones and have some rewards and punishments to teach the arm to achieve them. If all these small goals are achieve the arm would touch the object.

We can say it like this :

1 – We want the arm to never touch the ground.
2 – We want the arm to learn to move the gripper closer to the object.
3 – We want the arm to touch the object
4 – We want the arm to touch the object with the gripper base.

In the following, we will present the reward functions to teach the robot these small goals respecting this order of priority.

1) " We want the arm to never touch the ground "

This is our most forbidden deed. To teach the robot to never touch the ground we punish him every time it touches the ground. The severity of the punishment should be correlated to the distance to the object. This insures that if the arm keeps only touching the ground (a very unlucky series of exploration actions), it will get close to the objects and ultimately touch it. Actually, this alone could be a reward function that can do the work but would take a long time to converge.

The reward chosen is : rewardHistory = - 1000*dist;

where dist is the distance between the gripper and the object.

2) "We want the arm to learn to move the gripper closer to the object. "

Now the robot is punished for touching the ground, it should learn to get the gripper base closer to the object.

We set a reward for touching the ground. To keep the robot gripper in air and get it closer to the object we should give a reward more encouraging than the one given for touching the ground, hence the importance of the order of priority presented above.

Each time the robot makes an action we reward it without ending the episode with a reward correlated to the distance to the grippe :

rewardHistory = avgGoalDelta;

Where avgGoalDelta is a smoothed moving average of the delta of the distance to the goal :

avegGoalDelta=(avgGoalDelta*alpha)+(distDelta*(1-alpha));

Now unless the robot touches the ground or the object we don't end the episode and the reward is more encouraging because it's positive.

Now we have to teach the robot that there is a limit to the episode which is 100.
We do that by punishing the robot if the limit is reached and the object is not touched.
The punishment should be correlated to the distance to the object to ensure that at worst if the robot keeps missing it an reaching the limit of the episode, it will get better and finish closer the next time:

rewardHistory = -100*dist;

The reward is more encouraging than touching the ground in most cases because the factor is 100 and not 1000. The only times touching the ground is more encouraged are when the robot touches the ground very close to the object. Said in other words : any unit of distance closer to the object in air without touching the ground is rewarded 10 times better than the same unit closer to object touching the ground.

3) "We want the arm to touch the object"

When a collision is detected this means that the arm has touched the object because it's the only object present in the environment. Even though, we test if one part of the collision is the object and if so we issue a reward:

rewardHistory = REWARD_WIN;

Where REWARD_WIN = 100;

Now, this is sufficient for task 1. The robot will not touch the ground, try to get closer to the object. and with this reward learn to touch the object with any part of the arm.

What we want now is to teach it to touch it with gripper base. What we should keep in mind is that until now it has done a great job learning to touch it. We want to keep this learned skill and encourage touching the object with a specific part by building upon it. Giving a reward for example only if this condition is met would erase all these informations and would need a lot of effort in exploration to find the series of actions that would insure that. In contrast, if we encourage the robot little by little to get closer, any explored action would be useful. We do this by subtracting from the reward the distance between the contact point and the gripper base :

rewardHistory = REWARD_WIN-100000*dist2;

Where dist2 is the distance between the contact point and the gripper base.
When this distance is 0 the gripper base is touching the object and the more this contact point is distant the reward gets smaller.
The factor of 100000 was set empirically. We want more precision when we get close so we amplify the error. In other word we subtract REWARD_WIN for every 1/10 000 distance unit between the contact point and the gripper base.

All these 3 reward function have as effect to teach the robot to touch object with any part an then try each time to touch it with a part closer the gripper base until learning to touching it with only the gripper base.

4)" We want the arm to touch the object with the gripper base"

Once the gripper base touches the object we give the most encouraging reward :

rewardHistory = 10*REWARD_WIN;

iii. **Action control :**

There are two options for the control logic : by joint position or by joint velocity. Ultimately, the arm would learn to reach the object no matter what the type of the control logic is used. But if one uses the velocity, the arm would explore more values of velocity first and maybe would get higher velocities, while if one chooses position control he would get more precision first because it would explore a broader position values.

Now the value of the action would affect the results if not done properly. A simple adding/subtracting from the previous value is a good function to use :

float joint = ref[action/2]+pow(-1,action % 2)*(actionJointDelta);

We test if the action is even or odd and then add/or subtract a fixed action step: actionJointDelta=0.15 from the previous joint position.

iv. **Hyper-parameters tuning :**

This was the most challenging part. Having many hyper-parameters to tune , one would go for days tuning them, without getting closer to a good results. A good understanding of the effect of each one them is essential to get an optimal tuning. Although we don't understand yet all their effects we grasped how to tune most of them which helped getting a good result :

INPUT_WIDTH and INPUT_HEIGHT :
The resolution of the frames captured should be sufficient to get all the details needed to make a decision. Having a big resolution would result of an over fitting and thus a bad learning and also hardware memory consumption. The value chosen is 64.

LEARNING_RATE :
The learning rate should be lowered to solve over fitting error but not too low to not cause slow learning from episode to episode. The value chosen is 0.05.

Discussion on the previous parameters and how to set EPS_END and EPS_DECAY :

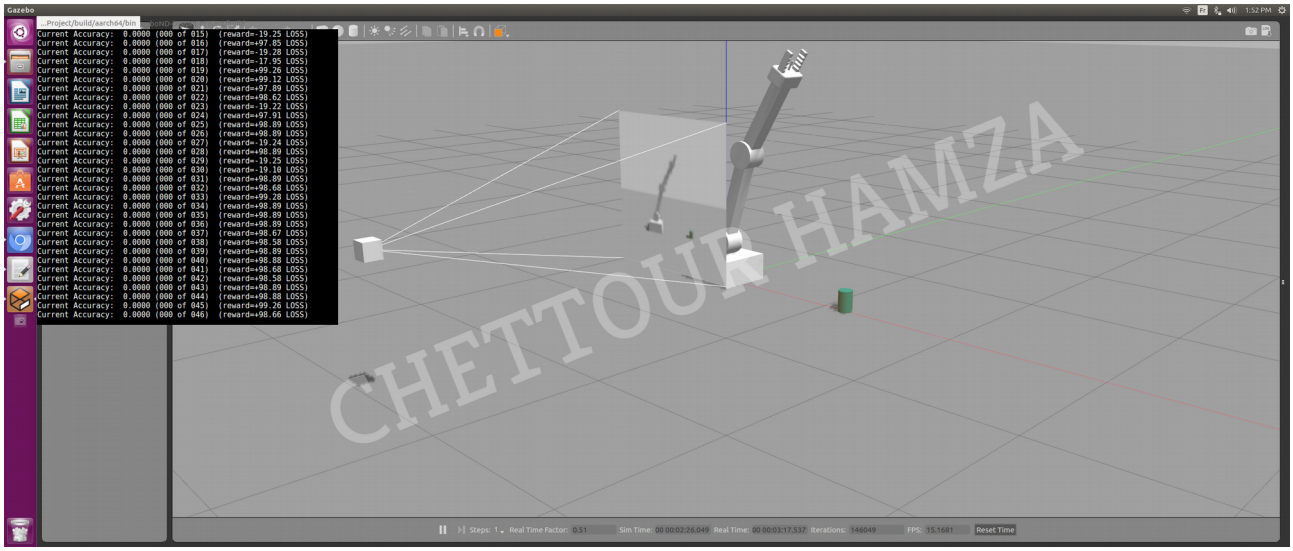
Many trials an errors showed that the real struggle is the compromise between exploration and exploitation.
The project's goal is to attain a certain accuracy over many episodes. This means that the robot should attain a certain consistency in touching the object. This could be translated this way : once the robot learn how to touch the object it should keeps doing it accurately.
This means that no exploration should exist by then.

On the other hand, the arm should explore at first until finding the best series of actions to take.

The INPUT_WIDTH and INPUT_HEIGHT and LEARNING_RATE gives the arm a learning speed.
The EPS_DECAY sets the number of episodes the exploration takes to get to the low level EPS_END.

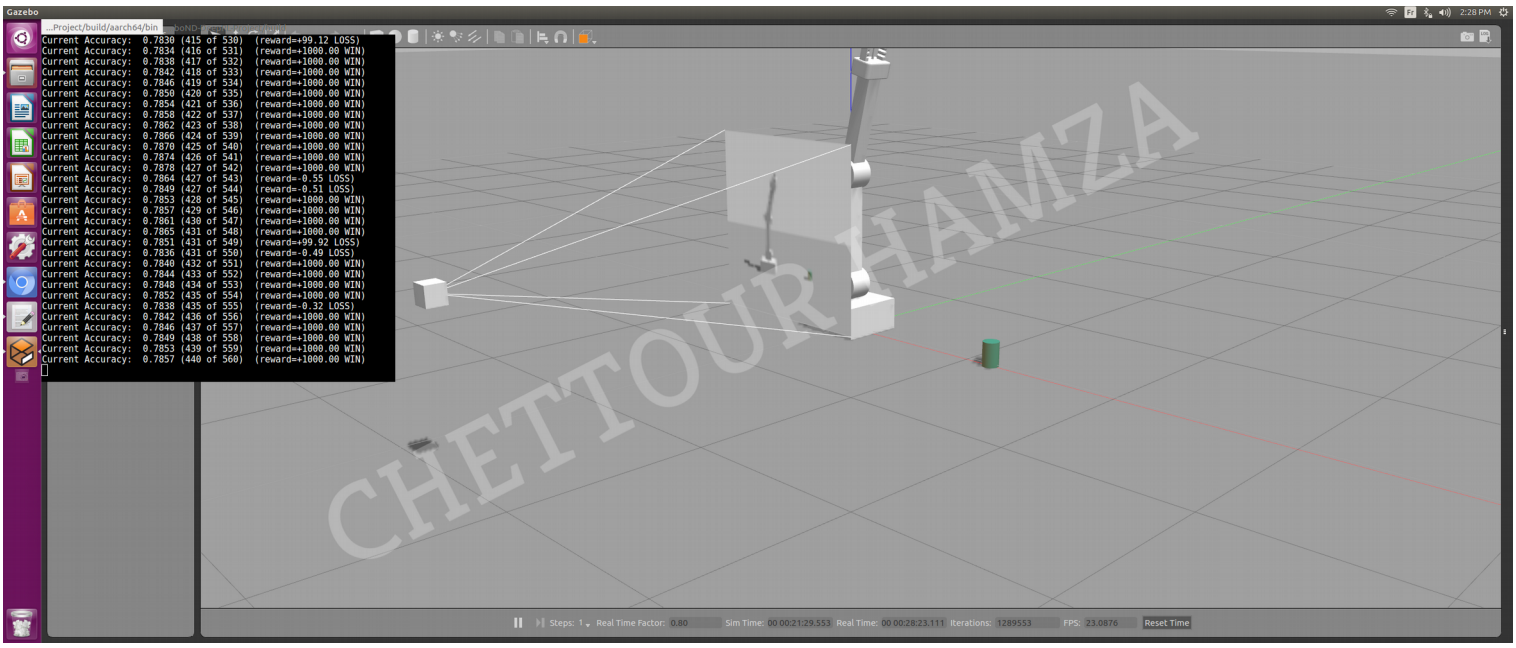
This is where the tuning gets trick.

If exploration decay doesn't give time to the arm to discover a series of actions that touches the object , touching the object would be less and less probable as episodes pass :



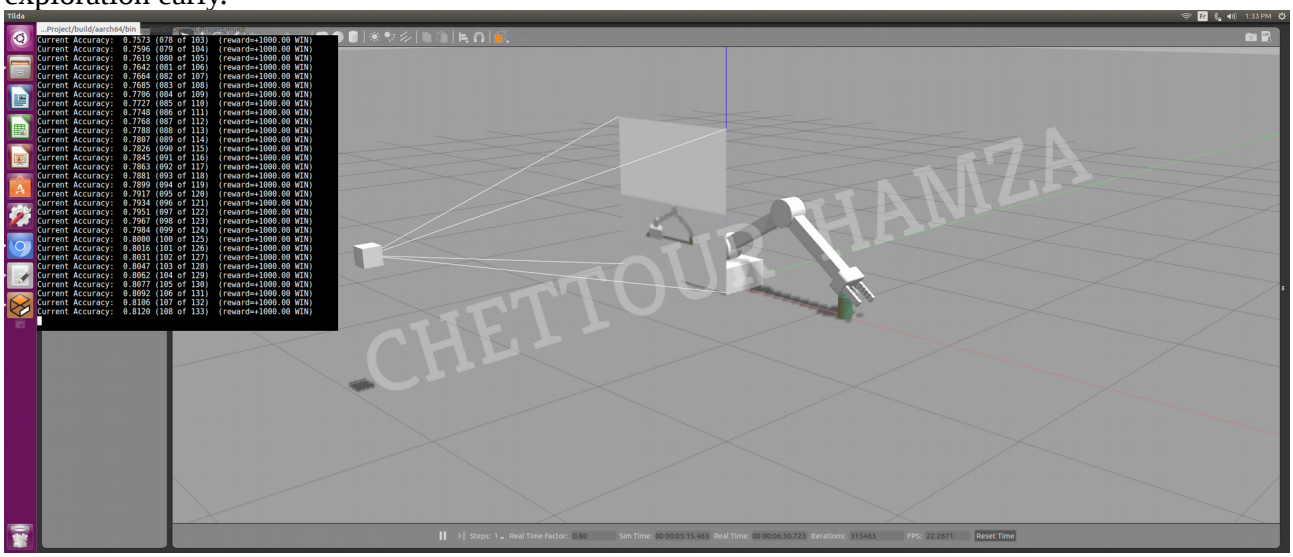
The decay was set to 50 and EPS_END to 0. In the attempt shown in the image, the exploration was not lucky to draw a series of actions that touches the object. Although getting very close by the end of the 50 episodes the robot didn't learn to touch the object.

The other compromise is a about the fading exploration. If the EPS_DECAY is set too high, we would have random actions even when the robot has already encountered the series of actions to perform the task , and thus we would loose the consistency needed to achieved the accuracy demanded.



The image above resulted from and attempt after setting EPS_DECAY to 500. As the picture shows, the arm has already learned how to touch the object, but we still see some failed attempts due to the fact that the discovery is still on.

One strategy was to gamble on an early appearance of the a series of actions and turn off the exploration early.



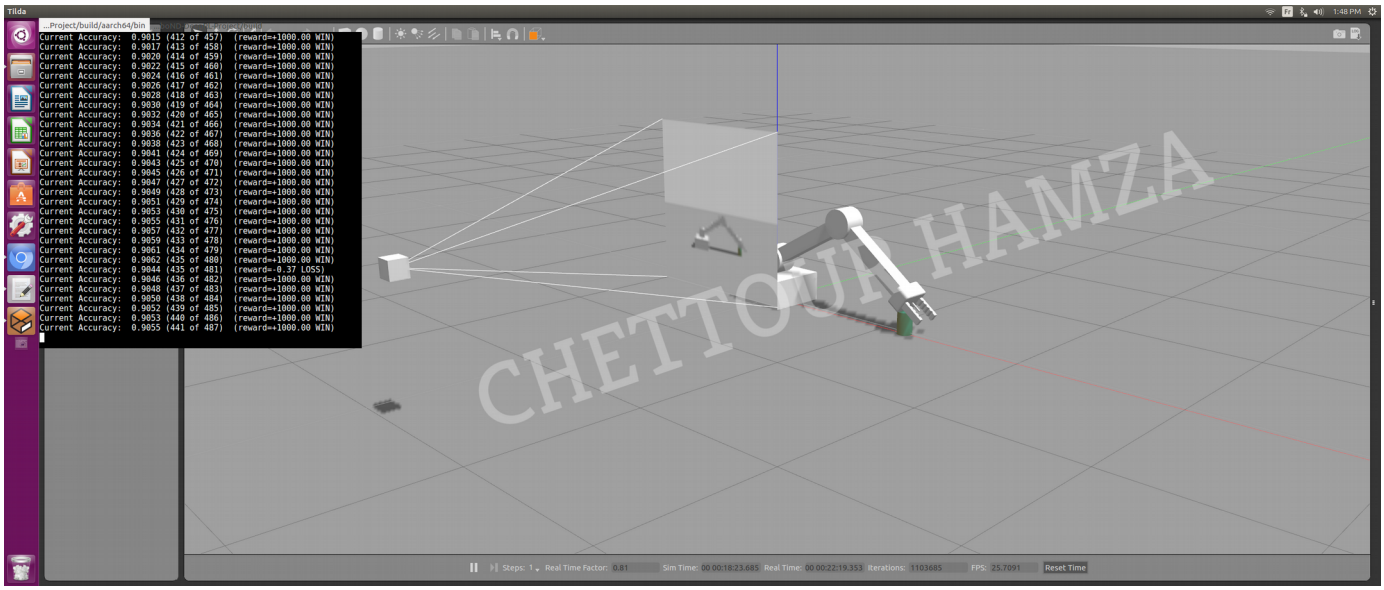
The image above was obtained after a series of attempt following the gambling strategy. We set the EPS_DECAY to 50 and EPS_END to 0.
We launch the simulation and if no actions that touches the object appears in the first 50 episodes we stop the attempt and launch again. It's gambling. But when a successful touch appears the robot learns very quickly and keeps touching the object consistently. The image shows achieving 80% accuracy after only 100 episodes.

Although it gives a great result when successful, this strategy has two major draw backs :

1 – Gambling on the appearance of successful attempt is not very elegant and is not sure. For this project the action space is not very vast but for another case It could be, making the probability of the appearance of a successful attempt at the beginning very low.

2- Once the robots sets on a successful series of actions after exploration ends (50 episodes) it keeps repeating them. It hasn't explored all the action space. And if because of an actuation uncertainty or calculation error the arm ends in a new position the robot wouldn't know what to do. This is why exploration probability should be small but not zero allowing the robot to keep learning to master it's environment.

Said in other words, we would trade off getting to the asked accuracy quickly for getting to it longer. The robot would achieve the accuracy target after learning enough and understanding better it's environment to keep doing the task in a stable manner.



Following this strategy, we set the EPS_DECAY to 300 and EPS_END to 0,0001. As the image shows, we get to 90% accuracy for task 2 after 450 episodes. We see that once in a while we get an exploration (episode 481).

v. Future work :

As discussed before, an optimal parameter tuning can't be done just with a try and error method but needs an understanding of what role each parameter plays exactly in the learning an what's its effect. For future work, this mastery of the parameters needs to be achieved.

For reward functions, we must add a reward correlated to the time the robots touches the object at to encourage the series of actions that get the results quicker.

Also a weird results we got when we set a non linear correlation between the reward and the distances, for example :

rewardHistory=-exp(10*dist)

The exponential would amplify the error thus the punishment on far distance to the object and lower it on closer distance, but weirdly enough linear functions like the ones used above worked better for learning. This needs to be understood better and clarified.

Finally, the effect of the deep neural network architecture and constitution need to be revised and studied.