Optimal Robot Grasp Planning Using Deep Reinforcement Learning Techniques

Hamid Manouchehri (hmanouch 50547895), Christ Joe Maria Anantharaj (christjo 50600405)

April 10, 2025

1 Abstract

In this project, we are utilizing the Pybullet simulator [**?**] for the task of finding the proper point of grasp for an unknown object. We have a 7 DOF robotic manipulator (KUKA LBR iiwa) and a cuboid object with an unevenly distributed mass. When we start the environment figure(1).
Initially, we bring the end effector of the robotic manipulator on top of the object, basically the location of the object is known but the shape is not. The idea is in each episode, the robot approaches the object and grip it from a random point, then it lifts the object to some extent and it measures the tilt angle with respect to horizontal plane, after that it puts back the object and based on the reward that the robot receives, it moves the end effector further.
We have started with a simple scenario where the robot arm looks for the best grasping point along the x-axis, the actions would be right or left. This project aims to find the optimal Deep Q-Learning Network algorithm from the family of DQN algorithms to train the robot agent to be able to grasp an object efficiently without the risk of dropping it

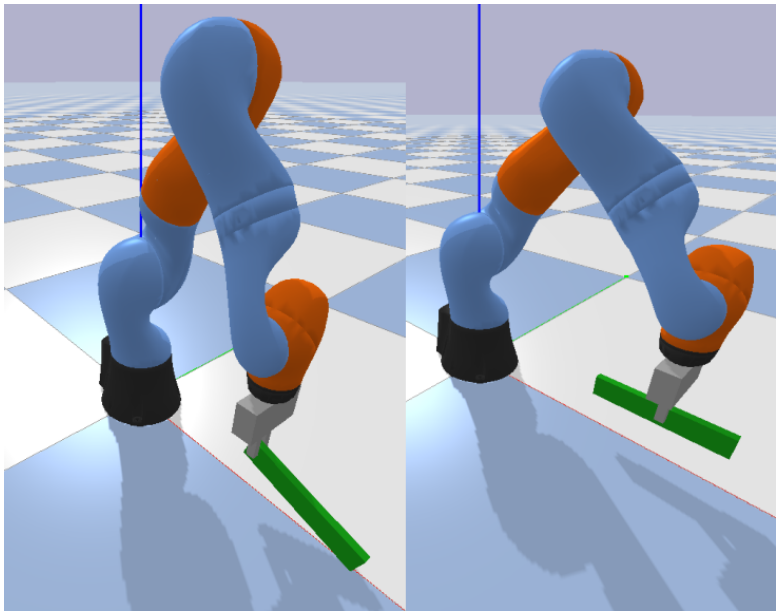

Figure 1: Pybullet simulator, grasping an unknown in-homogeneous object

Figure 2 shows the initial attempt of the robot trying to grasp the object without any external training factor. Without learning the orientation of the given object, the robot struggles to be able to hold on the the object efficiently. The underlying reward system is further elaborated in this report.

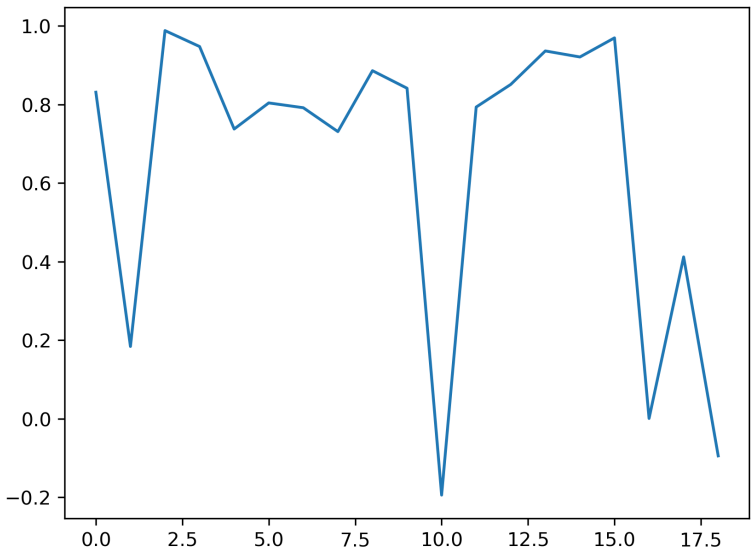

Figure 2: Reward per episode

2 Simulation Setup

This project utilizes the PyBullet simulation to be able to make the environment with the agent and the object which the agent will interact with. PyBullet depends on the physical characteristics like Euler and Quaternion coordinates and Kinematics for motion to make the agent move and interact with the environment and the object.

2.1 Agent

The agent used is a robot arm with prismatic gripper with 7 joints that help the robot arm move around. The grippers contain 2 clasps using with the robot arm grips an object. The agent and the inhomogenous object are placed on a plane. The agent and the object are calibrated such that the robot can grasp the object, but without the right training, the object can be dropped or not grasped efficiently which can lead the object to be tilted or even fall from the grip.

2.2 Observation Space

One of the main components of a Reinforcement Learning problem is the definition of the observation space which will be used to help the agent understand its current state with respect to the environment. The observation space for this problem is defined continously using the coordinate points along the x-axis of the object and the tilt angle measured after the object has been lifted by the robot arm. This helps the agent, ie the robot arm learn whether its taking the right action or not.

2.3 Action Space

The other important component of Reinforcement learning problem is defining the action space. This helps the user as well as the agent understand the set of actions the agent can take to interact with the environment and its elements. For this problem, the robot arm can take 2 actions, left or right along the x-axis of the object. Due to the limitations of the PyBullet simulation and the lack of mechanical components to be able to vary the action space, the actions for this problem statement have been defined to be discrete set with fixed movements. Nevertheless, for different objects with different center of masses, it can get complicated for the agent to learn the optimal point on the object to grasp it efficiently.

2.4 Reward Function

For every action the agent takes, a reward is provided to help the agent know whether its on the right track or not. Based on the outcome of this action, the reward can either be a penalty for an incorrect action or a

positive reward for a good action. For this problem statement, the rewards are based on how good the robot arm picks up the object without the scenario of it dropping or tilting the object too much. The threshold for the tilt angle is set to be around 0.5 radians, and hence if the the angle exceeds this in either orientation or axis, the agent is given a penalty. But if the tilt angle is less than the threshold, its given a positive reward.

3 Deep Q-Learning

DQN is part of a vast range of deep reinforcement learning algorithms that involves a function approximator like a neural network to estimate the optimal Q-Value using which the agent selects the optimal action to achieve its goal state. In this project, we aim to implement and find the optimal Deep Q-Learning algorithm for robotic applications like this which can help perform complex tasks.

3.1 Vanilla DQN

Vanilla DQN is a simple and considered the most basic implementation of the Deep Q-learning algorithms in the DQN family. Nonetheless, even today it finds itself useful in various applications, like this very problem. In this project, we have started with a simple implementation of the DQN algorithm involving a neural network that takes in observations of dimension 2 and gives an output of dimension 2 for the actions.

Every DQN algorithm involves these components:

- Target and Policy networks
- Replay Buffer

The target network is used to overcome the 'moving target' problem and replay buffers are used to store the past experiences to train the policy network on. The target network is then periodically updated to learn the optimal weights.

Using this framework, the robot arm was trained for 100 episodes and 20 timesteps each. The rewards per episode were recorded and plotted.

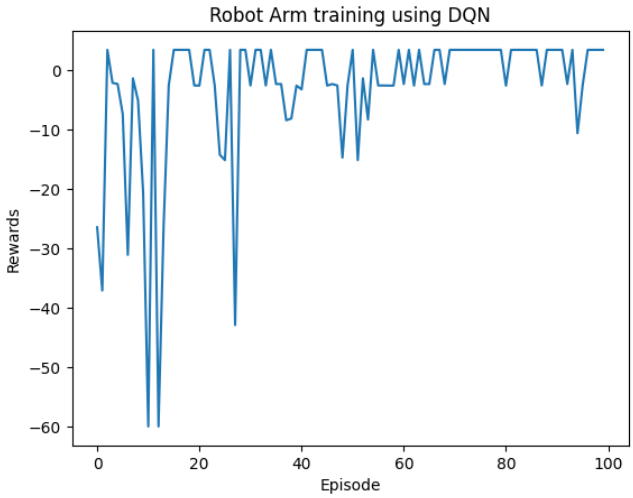

Figure 3: Rewards per Episode plot - Vanilla DQN

From this plot we can infer that the agent starts of slowly choosing greedy actions choses using the epsilon-greedy policy and gradually choosing the optimal actions that were trained using the policy network. But even towards the decay of epsilon value, the agent seems to kind of struggle to maintain higher rewards, showing this problem can definitely be improved.

3.2 Double DQN

Unlike vanilla DQN where the optimal actions are chosen using only a single approximator, Double DQN uses two approximators to estimate the optimal Q-value, which resolves the problem of overestimation bias. In many cases, Double DQN has proven to achieve optimal results sooner than vanilla DQN, and hence we have tried to implement this strategy to verify if the agent can perform better given this constraint.

The agent was trained for 100 episodes and 20 timesteps each. The rewards at the end of each episode were recorded and plotted.

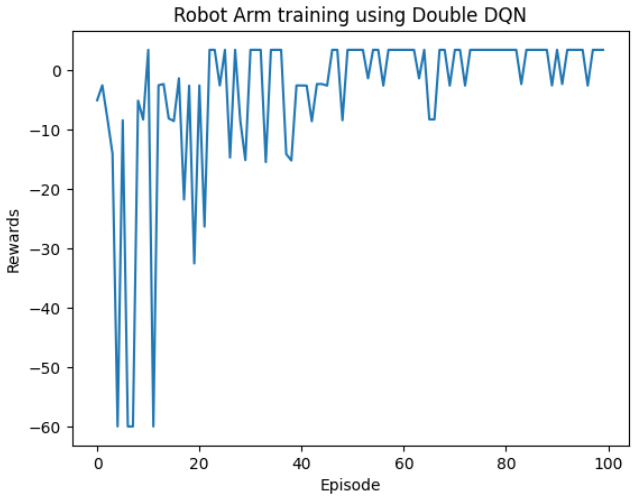

Figure 4: Rewards per Episode plot - Double DQN

3.3 Double DQN with Prioritized Experience Replay

The replay buffer in DQN plays a very vital role in storing the past experience trajectories from which a subset is sampled and are used to train the neural network to learn and differentiate the good actions for the given state. But a simple list with randomly sampled experiences do not always ensure consistent or optimal learning for the agent. Hence, the prioritized experience replay (PER) is introduced. Using a PER helps introduce a priority factor to these trajectories which help in the sampling process. Initially, all the experiences are given maximum priority using the temporal difference errors to ensure they are sampled atleast once. These experiences are sampled with some probability value, which gradually changes over time based on their assigned importance weights. As the agent is trained, the priorities are updated helping the important experiences providing more value to the training process.

Using the PER, the agent was trained for 100 episodes and 20 timesteps each. The rewards obtained for each episode was recorded and plotted.

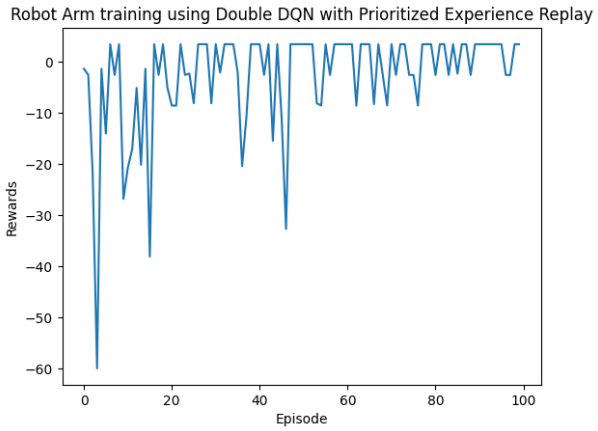

Figure 5: Rewards per Episode plot - Double DQN with Prioritized Experience Replay

The rewards now have shown a positive improvement showing how sampling important past experiences and learning from them has helped the agent learn the optimal policy to select the actions. This can however be further improved by making some modifications to the function approximator or the neural network used to estimate the Q-value.

3.4 Dueling DQN

Dueling DQN is an improvement of the DQN where the network estimates Q-values for each action, but also estimates the value of each state. In most cases, the choice of action for the state matters too. In this case, the neural network consists of two streams, the value stream $V(s)$ that estimates the value of that state, and the advantage stream $A(s, a)$ that calculates how good the selected action in that state is.

The benefits of using Dueling DQN lies in the agent learning state values more robustly, especially when an action does not significantly change the outcome. Using the dueling DQN, the agent was trained for 100 episodes and 20 timesteps each. The rewards for each episode were recorded and plotted.

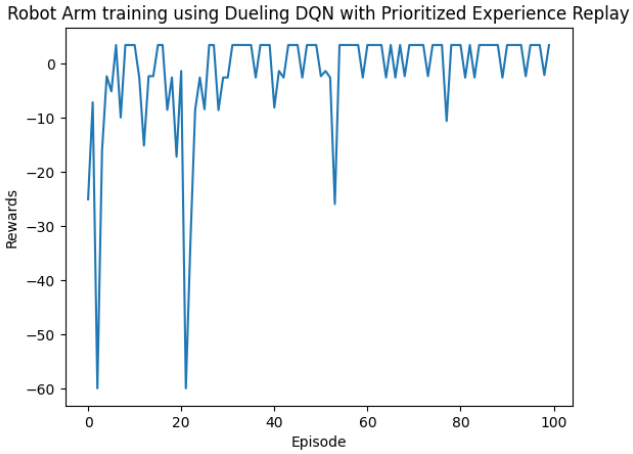

Figure 6: Rewards per Episode plot - Dueling DQN

From this plot we can infer that the agent is now learning to be more consistent in the positive region and selecting the optimal actions sooner.

4 Results

With the above implementation of the different algorithms from the DQN family, various observations were made that helped the robot arm learn to optimally grasp an object without tilting or dropping it. But the key contribution of this work lies in analyzing the best algorithm that helps in solving problem statements such as this and possible various other robotic applications with probably even more complex problem statements.

The results are compared using the rewards obtained during the training of the robot arm agent. Following is a plot containing all the rewards per episode during training.

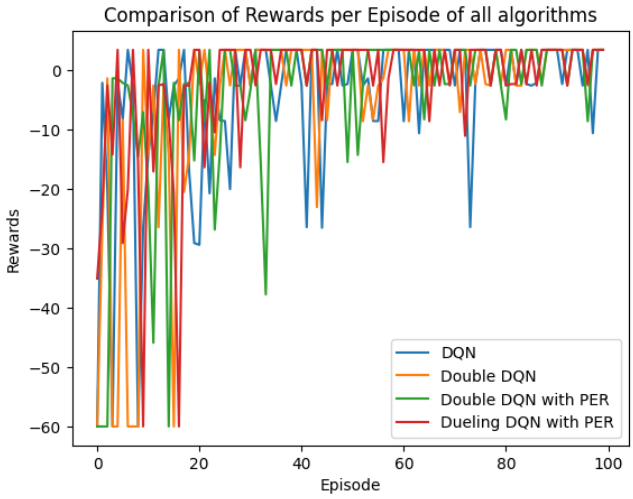

Figure 7: A comparison of performance of all the DQN algorithms

From figure 7 we can see how the Dueling DQN with PER algorithm has helped the agent achieve a higher reward sooner and maintain it consistently as compared to the other algorithms. This behavior can be attributed to the neural network architecture of Dueling DQN and prioritizing past experiences that provided more information in learning.

On a further analysis it was observed that the average rewards for all the 4 models do show significant differences that further complement the advantage of using Dueling DQN. On an average, the vanilla DQN, Double DQN, Double DQN with PER and Dueling DQN with PER have and average reward of -4.3, -4.8, -3.9 and -2.3 respectively. There is a higher variance in the rewards obtained using Dueling DQN as compared to the other algorithms, further proving to be a more optimal algorithm for this problem statement.

References

[1] https://pybullet.org/wordpress/

[2] https://gymnasium.farama.org/index.html

[3] https://arxiv.org/abs/2005.12729

[4] https://medium.com/@samina.amin/deep-q-learning-dqn-71c109586bae

[5] https://danieltakeshi.github.io/2019/07/14/per/

[6] https://medium.com/free-code-camp/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682

[7] https://medium.com/data-science/rainbow-the-colorful-evolution-of-deep-q-networks-37e662ab99b2