An Ablation Study of Deep Deterministic Policy Gradient

Rohan Chandra
EN.525.637.81 Foundations of Reinforcement Learning
Johns Hopkins University
Fall 2025

***Abstract* — DQN, or Deep Q Networks, effectively used deep neural networks as large, non-linear function approximators for reinforcement learning problems with continuous state spaces. However, the algorithm does not perform well on problems that have a continuous action space as well, like most robotic control problems. Lillicrap et. al developed the Deep Deterministic Policy Gradient, or DDPG, algorithm to solve this problem. In this project, I recreated DDPG and ran an ablation study on two strategies the algorithm employed: experience replay buffers and target networks. I was able to replicate similar results on two out of the three environments I tested and confirm some of the same conclusions the original authors came to, while also conducting the study successfully. Some concessions had to be made due to time and resource constraints.**

I. Introduction

DDPG, or Deep Deterministic Policy Gradient, was developed in 2015 by Lillicrap et. al to create algorithms for reinforcement learning problems with continuous action spaces [1]. They used Silver et. al's DPG, or Deterministic Policy Gradient [6], as a base, and then incorporated strategies from other papers, like Batch Normalization [7] and Experience Replay Buffers [5]. The authors achieved good results on a wide variety of reinforcement learning problems, while also confirming that deep neural networks could effectively be used as non-linear function approximators. The prevailing thought at the time was that the networks were too unstable to ever be useful, but DQN and DDPG showed otherwise [5] [1].

II. Recreation and Ablation Study Setup

A. Environment

I used three environments from OpenAI Gymnasium, simulated using the MuJoCo physics engine [2] [9]. These environments were Pendulum-v1, Reacher-v5, and Hopper-v5. Each environment was chosen for its ease of use and fast runtime, and the specific sequence of environments would provide increasing difficulty for the algorithm. A comparison of each environment is shown in Table I.

There is a considerable difference in not only the state and action dimensionality of each environment, but also in the difficulty of the task itself.

TABLE I: Comparison of Gymnasium environments

Env.	State / Action	Task Description	Difficulty
Pendulum-v1	3 / 1	Swing up and balance a single pendulum in the upright position	Low
Reacher-v5	11 / 2	Control a two-link arm to reach a randomly placed target in the plane	Medium
Hopper-v5	11 / 3	Learn one-legged forward locomotion without falling	High

B. Algorithm Recreation Setup

For the purposes of this project, a successful recreation of the algorithm meant being able to achieve similar results on the majority of the three environments compared to the low-dimensional implementation of DDPG. This version of the algorithm used low-dimensional observations, like cartesian coordinates or joint angles, as input data to the neural networks.

Each test was structured as an episodic task, where the episode finished either after failing the task, or reaching the maximum step limit, which was set to 1000 steps for this project. Each seed was run for 1000 episodes, making the total maximum step count for each seed 1000000. Five seeds were used for each environment. The average raw score and the best raw score were taken from each seed and averaged in order to get the final average raw score and average best raw score across the five seeds.

Finally, these raw scores were normalized using two baselines. The first was a random policy, and the second was from scores generated from a planner algorithm that had full access to the inner workings and dynamics of each environment. The former was marked 0, and the latter 1. This means that if DDPG achieved a score less than 0, it did worse than a random policy, and if it achieved a score higher than 1, it did better than this omniscient planner. Equation 1 shows the formula used to normalize the raw scores with the baselines.

$$R_{\text{norm}} = \frac{R_{\text{agent}} - R_{\text{random}}}{R_{\text{expert}} - R_{\text{random}}} \tag{1}$$

C. Ablation Study

The ablation study was conducted using the Reacher-v5 environment. I wanted the results of the study to be dependent

on the algorithm itself, and not be too heavily influenced by the choice of environment, hence why the medium difficulty one was chosen. The study was conducted on the use of the Experience Replay Buffer (or just Replay Buffer) and the Target Networks. The authors cite the latter as being crucial to the success of the algorithm, something I wanted to test myself [1]. The former was chosen because I wanted to see how important having independently and identically distributed, or i.i.d, data was to the algorithm's success.

For each ablation configuration, I ran ten seeds. Like with the algorithm recreation, each seed was run for 1000 episodes, with a maximum step count of 1000. For each seed, the raw score for each episode was stored. Then, across the ten seeds, the raw scores were averaged, returning the final 1000 values representing the average raw reward per episode. These 1000 raw rewards were then normalized using Equation 1 to get the final deliverable: 1000 values, representing the average normalized reward for each episode across the 10 seeds.

D. Limitations

Due to time and resource constraints, I made some concessions in both recreating the results from the original algorithm and running the ablation study. First, the authors originally used 27 environments to test DDPG, running 5 seeds for each environment for a maximum step count of 2.5 million. I did not have access to enough GPU power to make this much testing feasible in the time I had for this project, therefore, I chose the environments already mentioned above, and ran each test for a maximum step count of 1 million. For two of the environments I used, Reacher and Hopper, I had to use the modern versions of the environments because the versions the authors used have since been deprecated.

The authors developed both low-dimensional and high-dimensional versions of DDPG [1]. The low-dimensional version uses low-level features, like joint angles or coordinates, as input to the models, whereas the high-level version uses screenshots from the simulations themselves (additional convolutional layers are added to process these screenshots). For both the recreation and the ablation study, I only used the low-dimensional version.

In the original papers, the authors used an iLQG planner [4] to get the top baseline scores for each environment. Instead I used benchmark scores obtained from other sources. [3]

Batch normalization was used extensively in the low-dimensional algorithm to normalize each of the features to have unit mean and variance. However, in my testing I found that it was having a severely negative effect on training. It is possible that this had to do with having to use the newer environments for testing. Therefore, I did not use batch normalization in any of my tests. I did not see a large negative impact on performance by omitting batch normalization.

III. Method and Results

A. DDPG

The DDPG algorithm is derived from Silver et. al's DPG algorithm, while also incorporating strategies from other papers to enable it to develop policies for continuous action space problems [1]. It uses two neural networks, an actor and a critic, to represent the policy and the Q-Learning Bellman Equation respectively. During each training step, a mini-batch of samples is drawn from the experience replay buffer. The actor receives a state and outputs what it believes is the maximizing action, based off its current policy. Then, the algorithm uses the target networks to calculate the target value, while the original critic calculates its best estimate for what the action-value function should be. Equation 2 shows the calculation of the target value.

$$y_i = r_i + \gamma\, Q'\Big(s_{i+1}, \mu'\Big(s_{i+1} \mid \theta^{\mu'}\Big) \mid \theta^{Q'}\Big) \tag{2}$$

The mean square error between these two values, the target and the estimate, becomes the loss the critic performs backpropagation against, shown in Equation 3.

$$L(\theta^Q) = \frac{1}{N}\sum_{i=1}^{N}\big(y_i - Q(s_i, a_i \mid \theta^Q)\big)^2 \tag{3}$$

This critic is then used to update the actor, shifting its weights towards what the critic believes will output the true maximizing action for the next step. This is known as the Policy Gradient, and was first developed by Silver et. al [6]. It is shown in Equation 4.

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s, a \mid \theta^Q)|_{s=s_i,\, a=\mu(s_i)}\ \nabla_{\theta^\mu}\mu(s \mid \theta^\mu)|_{s_i} \tag{4}$$

Finally, the target network weights are updated manually using the soft-update hyperparameter τ, shown in Equation 5.

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{5}$$

Each of the strategies DDPG uses are summarized below.

1) : Actor-Critic Architecture/Deep Neural Networks. The use of the neural networks and actor-critic architecture are vital to the success of the algorithm, otherwise it is impossible for Q-Learning to work on continuous action spaces.

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \tag{6}$$

$$Q \leftarrow Q + \alpha\, \delta_t \tag{7}$$

As demonstrated in Equations 6 and 7, the Bellman Equation has to find the action that maximizes the Q-function, but in a continuous setting this becomes impossible to do. Instead, we can have the actor approximate the maximizing action, a critic to evaluate the actor, and then use backpropagation to update the weights for both networks so that they can learn better behavior.

2) : Target Networks. Q-Learning with neural networks is prone to error. Without target networks, the mean square error is minimized to try and update the weights, but then the target values are also dependent on the weights, so both the weights and targets are moving in the same direction, and the optimizer has nothing to stabilize against. This causes instability, which leads to divergence. The solution is to use copies of the original networks, called target networks, to calculate the target values instead. Then, when it is time to update the weights, the weights for the original networks are updated via traditional backpropagation, while the target network weights are updated manually. Equation 5 shows how this update is done.

τ is a tunable hyperparameter that is typically less than 1. It limits the rate at which the target networks can learn, while still having them "track" the knowledge the original networks have gained. While this does slow down the training, it greatly increases the stability, ensuring the algorithm can learn properly.

3) : Experience Replay Buffer. Traditional neural networks operate on the assumption that the data it is receiving is independent and identically distributed, or i.i.d. This assumption does not hold for most reinforcement learning problems, as oftentimes one experience is dependent on what happened before it, creating correlation within the data. To solve this, the authors take inspiration from Minh et. al's Deep Q Networks, and use an Experience Replay Buffer [5]. The buffer stores all the experiences the algorithm encounters while training, typically stored as tuples of the form (state, action, reward, next state). To train the neural networks, mini-batches of random samples are taken from the buffer, mimicking i.i.d data.

4) : Batch Normalization. This strategy was specifically used for the low-dimensional version of the algorithm to ensure that the low-level features were scaled the same. In traditional neural network learning, if one feature has larger values than another, it can skew training. However, using batch normalization, each feature is scaled to have unit mean and variance, ensuring that this doesn't become an issue [7].

5) : Noise. Noise was introduced into the algorithm to encourage exploration. When the actor chooses an action, noise is sampled from a predefined noise policy and added to the chosen action. The authors used Ornstein-Uhlenbeck noise, which mimics Brownian motion. However, they also specified that the noise could be chosen to fit the environment [8].

The results from my recreation experiments are shown below.

TABLE II: Comparison of average and best returns between my implementation and the original results

Environment	R_{av} (mine)	R_{best} (mine)	R_{av} (orig)	R_{best} (orig)
Pendulum-v1	0.861	1.196	0.946	1.021
Reacher-v5	0.847	1.023	0.720	0.987
Hopper-v5	0.02	0.565	0.676	0.936

I was able to achieve similar or better results on the easy and medium level environments, Pendulum-v1 and Reacher-v5, but was not able to match the performance on the hardest environment, Hopper-v5. Hopper-v5 is more unstable compared to the other two environments, and the task itself is harder, as explained in Table I. In the original paper, the authors achieved an average score, for the low dimensional case, of 0.676 on Hopper-v5. This is much lower compared to the environments with easier tasks, like Pendulum, where the average score was 0.946.

Based on the results, DDPG is still a strong algorithm to use for problems with continuous action spaces, even with modern environments.

B. Ablation: Target Networks

For this portion of the ablation study, five configurations were tested. They consisted of no target networks, and $\tau = 0.001, 0.005, 0.01, 0.05$, with the authors originally using 0.001. The results of the study are shown in Figure 1 and Figure 2. For better visualization, the curves are smoothed by averaging over 5 iterations.

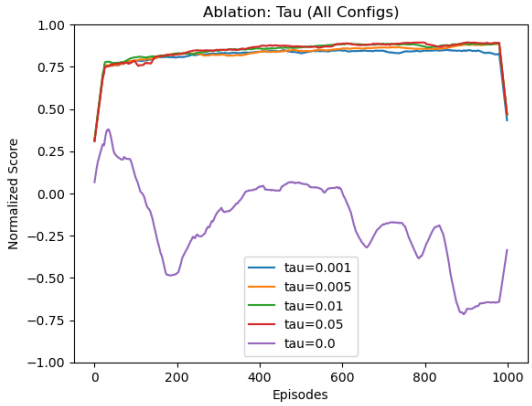


Fig. 1: Tau Ablation, All Configurations

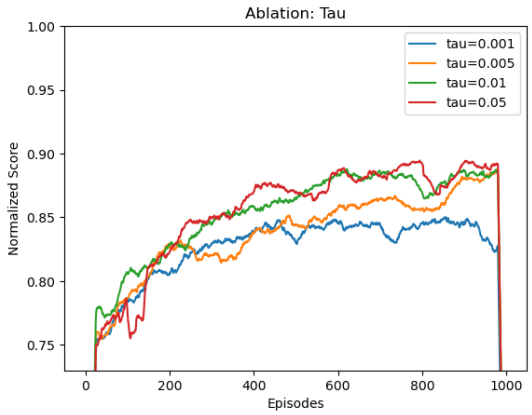


Fig. 2: Tau Ablation, Only Configurations with Target Networks

Figure 1 shows the result of running all the configurations. The purple line represents the case where the target networks

were removed. Removing the target networks severely impacted the performance of the model, with the scores being almost always less than that of the random policy. This confirms a conclusion that the DDPG authors came to. In Figure 2 of the paper, they showed the target networks were "crucial" for the algorithm to perform well [1]. I was able to come to the same conclusion, highlighting the need for target networks to ensure that the Q-Learning can be done properly.

Figure 2 is the same as Figure 1, but without the "no target networks" configuration. $\tau = 0.001$, the original value, performed the worst out of the four, with the normalized score increasing as τ increased. This suggests that while target networks are necessary for the algorithm to work, they can be used more aggressively than the authors previously assumed, and doing so can lead to better performance overall.

C. Ablation: Experience Replay Buffer

The experience replay buffer was tested using the following configurations: no buffer, and buffers of size 500000, 1000000, and 2000000. The original size of the experience replay buffer was 1000000.

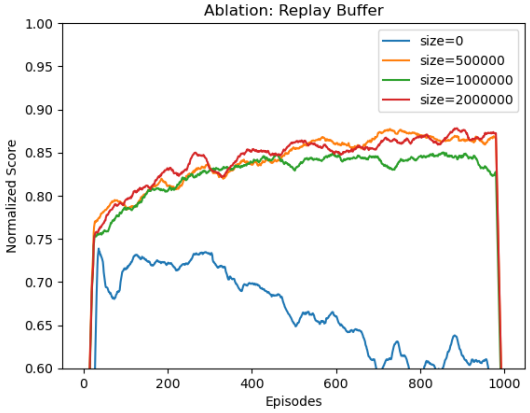


Fig. 3: Replay Buffer Ablation, All Configurations

Figure 3 shows that similar to the target networks, not having the experience replay buffer negatively impacted the performance of the model (though not as much as it did with the target networks). However, regarding the other configurations, there was not much difference between them, despite the fact that the experience replay buffer size was both halved and doubled. This suggests that while a experience replay buffer is necessary to have the algorithm perform well, the size of the buffer does not make as much of an impact on performance as changing τ for the target networks did.

IV. Conclusion

Testing was done with the original DDPG algorithm on three environments of increasing difficulty. Similar results to those reported in the original paper were achieved for two of the three environments, with the last environment, Hopper-v5, showing significantly worse performance. This can be attributed to the task itself, having a one legged robot jump forward as fast as it can, being considerably harder than the others.

Through the ablation study, I was able to determine the impact that the target networks and experience replay buffer have on the algorithm. The target networks are crucial for the algorithm to perform well, which the authors also reported in the original DDPG paper. The soft update hyperparameter, τ, plays an important role in the algorithm's performance, and through testing, it was shown that increasing tau led to a higher normalized score. This suggests that DDPG users can be more aggressive with this hyperparameter without running the risk of the algorithm diverging.

The experience replay buffer is also necessary for the algorithm to perform well, although the drop in performance was not as severe as it was when removing the target networks. However, unlike the target networks, affecting the size of the associated hyperparameter (the size of the buffer) did not have a significant impact on performance. This suggests that while the buffer is necessary for the algorithm to perform well, the size of the buffer itself is not as important, so long as it is around 1000000.

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. [Online]. Available: https://arxiv.org/abs/1509.02971

[2] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 5026–5033. [Online]. Available: https://ieeexplore.ieee.org/document/6386109

[3] A. Raffin, "RL Baselines3 Zoo," 2021. [Online]. Available: https://github.com/DLR-RM/rl-baselines3-zoo

[4] Y. Tassa, T. Erez, and E. Todorov, "Control-limited differential dynamic programming," *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1168–1175. [Online]. Available: https://ieeexplore.ieee.org/document/6907001

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://www.nature.com/articles/nature14236

[6] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," *International Conference on Machine Learning (ICML)*, 2014, pp. 387–395. [Online]. Available: https://arxiv.org/abs/1509.02971

[7] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *International Conference on Machine Learning (ICML)*, 2015, pp. 448–456. [Online]. Available: https://arxiv.org/abs/1502.03167

[8] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Physical Review*, vol. 36, no. 5, pp. 823–841, 1930. [Online]. Available: https://journals.aps.org/pr/abstract/10.1103/PhysRev.36.823

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016. [Online]. Available: https://arxiv.org/abs/1606.01540