Deep Reinforcement Learning for Breakout and Hopper: Using DQN and DDPG

Li Kezhi, 520021911013

***Abstract*—This paper explores the application of model-free reinforcement learning (RL) techniques in solving two problems: "Breakout," a classic arcade game, and "Hopper," a dynamic locomotion task. The Deep Q-Network (DQN) algorithm is used for "Breakout," while the Deep Deterministic Policy Gradient (DDPG) algorithm is employed for "Hopper." Through these examples, we demonstrate the potential of RL in addressing diverse challenges and showcase the adaptability and versatility of RL algorithms. This project contributes to a deeper understanding of RL and its impact on the development of intelligent autonomous systems.**

***Index Terms*—Reinforcement learning, model-free RL, DQN, DDPG, Breakout, Hopper.**

I. Introduction

Artificial Intelligence (AI) has revolutionized numerous fields by enabling machines to exhibit intelligent behavior and make autonomous decisions [1]. One prominent branch of AI is reinforcement learning (RL), which empowers agents to learn and adapt through interactions with their environment. RL has proven to be a powerful paradigm for training autonomous systems in various domains, including robotics, game playing, recommendation systems, and autonomous vehicles.

Reinforcement learning is a subset of machine learning that emphasizes sequential decision-making processes. Unlike supervised learning, which relies on labeled data, and unsupervised learning, which seeks to discover patterns in unlabeled data, reinforcement learning operates on a feedback mechanism based on rewards and punishments. By maximizing cumulative rewards over time, RL algorithms train agents to make optimal decisions in complex and dynamic environments.

In this paper, we delve into the domain of AI and specifically focus on reinforcement learning techniques, with an emphasis on model-free RL. Model-free RL refers to the class of RL algorithms that learn directly from experience without requiring explicit knowledge of the environment dynamics. This allows agents to generalize their learning across different environments and adapt to new situations.

Two key categories within model-free reinforcement learning are value-based RL and policy-based RL. Value-based RL involves estimating the value of different actions or state-action pairs in order to make decisions. It utilizes methods such as Q-learning and the Deep Q-Network (DQN) algorithm, which leverage deep neural networks to approximate the action-value function. This approach allows agents to learn and update their value estimates iteratively, ultimately leading to improved decision-making abilities.

On the other hand, policy-based RL directly learns the policy, which is a mapping from states to actions. Instead of estimating the value function, policy-based methods focus on searching for the best policy directly. Algorithms like the Deep Deterministic Policy Gradient (DDPG) algorithm utilize deep neural networks to parameterize the policy and optimize it through gradient descent. Policy-based RL enables agents to handle continuous action spaces and has shown remarkable success in a wide range of applications.

Our goal is to apply model-free RL to solve two different problems: "Breakout," a classic arcade game [2], and "Hopper," a dynamic locomotion task [3]. We will use the DQN algorithm for "Breakout" and the DDPG algorithm for "Hopper." By using these algorithms, we hope to show how RL can be used to solve diverse challenges and make machines smarter.

This paper provides a comprehensive exploration of AI and RL, with a focus on model-free RL. We will explain the concepts and methods behind RL in a simple and understandable way. Through our project examples, we aim to demonstrate the exciting potential of RL in solving real-world problems.

Overall, this paper is an opportunity for us to learn and showcase how AI and RL can make machines more intelligent. By exploring "Breakout" and "Hopper," we hope to inspire further exploration and understanding of RL algorithms and their applications.

II. Related Works

Reinforcement learning (RL) has been widely studied and applied in various domains. In this section, we review some key works related to our research on using model-free RL algorithms, specifically the Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG), to solve different problems.

A. Deep Q-Network (DQN)

The DQN algorithm, proposed by Mnih et al. [4], introduced a breakthrough in RL by combining Q-learning with deep neural networks. The DQN leverages a deep neural network, represented as $Q(s, a; \theta)$, to approximate the action-value function. The network takes the state s and action a as inputs and outputs the estimated value. The parameters θ are learned by minimizing the mean squared TD error using stochastic gradient descent,

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(r(s_t, a_t) + \gamma \max_a \hat{Q}(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)\right)^2\right]$$

, where r is the reward and γ is the discount factor that determines the importance of the future reward, and θ^- represents the parameters of the target network.

Specially, the DQN algorithm utilizes a technique called experience replay, where transitions (s, a, r, s') observed during interaction with the environment are stored in a replay memory. During training, a minibatch of transitions is sampled randomly from the replay memory to decorrelate the data and improve learning stability.

Algorithm 1 DQN algorithm

```
Initialize replay memory $D$ with capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for each episode $j$ do
    Initialize state $s_1$
    for $t = 1$ to $T$ do
        Select action $a_t$ using $\epsilon$-greedy policy based on $Q$ and $s_t$
        Execute action $a_t$, observe reward $r_t$ and next state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
        Sample random minibatch of transitions $(s_j, a_j, r_j, s_{j+1})$ from $D$
        Set target values:
        $y_j = r_j$ for terminal state $s_{j+1}$
        $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$ otherwise
        Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to $\theta$
        Every $C$ steps, update target network:
        $\theta^- = \theta$
    end
end
```

Since its introduction, DQN has achieved remarkable successes in various domains, including playing Atari 2600 games [2]. It demonstrated the ability to learn directly from raw pixel inputs and surpass human-level performance in several games.

B. Deep Deterministic Policy Gradient (DDPG)

DDPG, proposed by Lillicrap et al. [5], addresses RL problems with continuous action spaces. It combines the strengths of policy-based and value-based methods. DDPG utilizes an actor-critic architecture, where the actor network, denoted as $\mu(s; \theta^\mu)$, learns the policy by mapping states s to actions. The critic network, represented as $Q(s, a; \theta^Q)$, estimates the action-value function.

To update the actor and critic networks, DDPG employs the deterministic policy gradient algorithm, which computes the gradients of the expected return with respect to the policy parameters. The networks are updated using target networks and experience replay, allowing stable and efficient learning.

DDPG has demonstrated promising performance in various continuous control tasks, including robotic manipulation and locomotion [5]. Its ability to handle high-dimensional state spaces and continuous actions makes it well-suited for complex real-world applications.

Algorithm 2 DDPG algorithm

```
Initialize actor network $\mu$ with random weights $\theta^\mu$
Initialize critic network $Q$ with random weights $\theta^Q$
Initialize target networks $\hat{\mu}$ with weights $\theta^{\hat{\mu}} = \theta^\mu$ and $\hat{Q}$ with weights $\theta^{\hat{Q}} = \theta^Q$
Initialize replay memory $D$ with capacity $N$
Initialize exploration noise $\mathcal{N}$
Initialize batch size $B$ and learning rate $\alpha$
Initialize discount factor $\gamma$ and soft update factor $\tau$
Initialize episode count $E$ and maximum time steps $T$
for $e = 1$ to $E$ do
    Initialize a random process $\mathcal{N}$ for action exploration
    Initialize the episode's cumulative reward $R = 0$
    Initialize the episode's time step $t = 0$
    Initialize the initial state $s_0$
    while $t < T$ do
        Select an action $a_t = \mu(s_t; \theta^\mu) + \mathcal{N}t$ according to the current policy and exploration noise
        Execute action $a_t$, observe reward $r_t$ and next state $s_{t+1}$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay memory $D$
        Sample a random minibatch of $B$ transitions $(s_j, a_j, r_j, s_{j+1})$ from replay memory $D$
        Set $y_j = r_j + \gamma \hat{Q}(s_{j+1}, \hat{\mu}(s_{j+1}; \theta^{\hat{\mu}}); \theta^{\hat{Q}})$
        Update critic network by minimizing the mean squared Bellman error:
        $\theta^Q \leftarrow \theta^Q - \alpha \frac{1}{B} \sum_j \left(y_j - Q(s_j, a_j; \theta^Q)\right)^2$
        Update actor network using the sampled policy gradient:
        $\nabla_{\theta^\mu} J \approx \frac{1}{B} \sum_j \nabla_a Q(s, a; \theta^Q)|_{s=s_j, a=\mu(s_j)} \nabla_{\theta^\mu} \mu(s; \theta^\mu)|_{s_j}$
        $\theta^\mu \leftarrow \theta^\mu + \alpha \nabla_{\theta^\mu} J$
        Update the target networks with soft updates:
        $\theta^{\hat{Q}} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{\hat{Q}}$
        $\theta^{\hat{\mu}} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\hat{\mu}}$
        $R \leftarrow R + r_t$
        $t \leftarrow t + 1$
    end
end
```

C. Comparison and Advancements

Both DQN and DDPG have made significant contributions to RL. DQN excels in discrete action spaces and has shown exceptional performance in playing Atari games. DDPG, on the other hand, is effective in continuous action spaces and has been successfully applied in various robotic control tasks.

In our project, we apply DQN to solve the ”Breakout” game, showcasing the power of value-based RL in discrete action spaces. For the ”Hopper” locomotion task, we utilize DDPG to demonstrate the effectiveness of policy-based RL in continuous action spaces.

By combining these two algorithms, we are able to explore the strengths and limitations of each approach and provide insights into their applications in different problem domains.

III. Experiment

A. Breakout

1) Environment and Game Characteristics: "Breakout" is a classic arcade game released by Atari in 1976. It is a single-player game where the player controls a paddle at the bottom of the screen and tries to break a wall of bricks by bouncing a ball off the paddle. The objective of the game is to break as many bricks as possible while preventing the ball from falling off the screen.

The observation space of Breakout is a 210×160 RGB image. The action space consists of four discrete values, where **0** for "NOOP", **1** for "FIRE", **2** for "RIGHT" and **3** for "LEFT". The agent shoud be rewarded if the ball breaks any brick.

2) Preprocessing: As the image size is too large to store, we first down-sampling the image to 84×84 (aligned with [4]), and convert it to greyscale, see Fig. 1. We use "uint8" to store the image in the replay buffer and convert it to "float32" by normalising it between $[0, 1]$ when training.

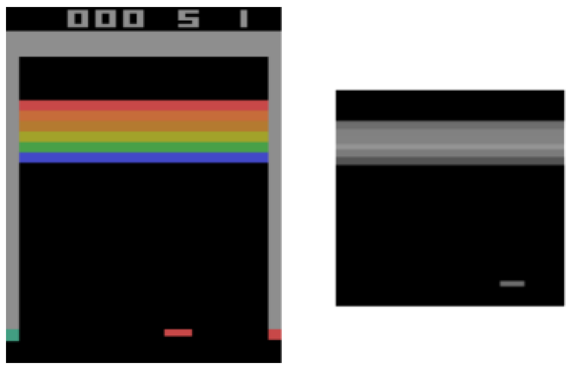

Fig. 1. The preprocessing filter applied to a frame in Breakout.

Once preprocessed, we stack four consecutive frames to construct the state $s_t = (x_{t-3}, x_{t-2}, x_{t-1}, x_t)$, which I assume to fulfill the Markov property since it captures both direction and speed of in-game objects.

3) Deep Q-Network Framework: The deep Q-network estimates the value of the action when inputs the state. Fig. 2 shows the deep neural network we used for this game. We first extract the features from image by putting three convolutional layers with different kernel size and stride width. Then, we directly put two fully connected layers to transform the features to the values of the four actions.

4) Parameters: As tuning parameters is always a tough work for reinforcement learning, the parameters baseline chosen in this game is shown in Tab. I. Then, comparisons of different choices of key hyper-parameters are presented in next part.

It should be noted that limited by the computation resource and time, we only train the model for 20000 episodes, so the final performance is not that good as we want, but we can still compare the effects of different parameters.

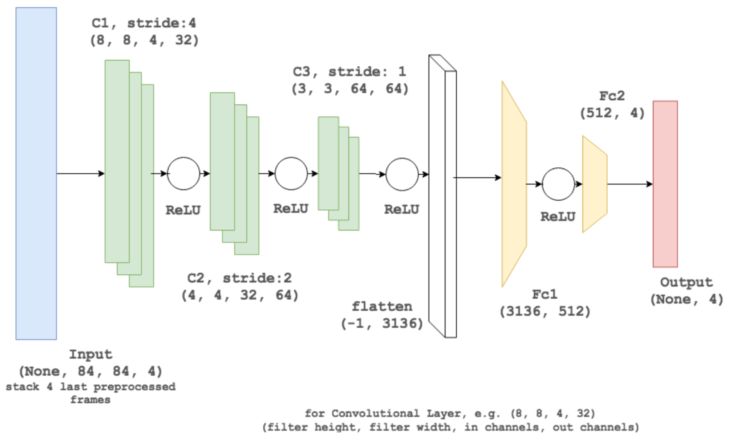

Fig. 2. The structure of deep neural network used for DQN.

TABLE I
BASELINE PARAMETERS FOR DQN

Parameter	Value
learning rate	2.5×10^{-4}
γ	0.99
episodes	20000
explore steps	800k
ϵ	$1 \rightarrow 0.1$
memory size	200k
batch size	32

5) Result and Comparisons: The results for baseline are shown in Fig. 3. From the figure, the reward is increased to around 2 and the length is fluctuated around 350, which means the agent has successfully found some strategies to play this game and can break 2 bricks per episode. The loss kept increasing to a maximal and finally converged downward around 0. The loss curve well explained the progress from explore stage to converge stage, where the increasing period of the loss means the agent explored many unfamiliar observations and had a large scale update toward its network parameters. The decreasing stage of the loss reflects that the explore rate had been small at that time and the agent tried to follow the best action route it has explored.

The discount factor γ represents how the future steps affect the current state, which plays an important role in TD method. Fig. 4 and Fig. 5 show the reward curve and loss curve during the training process of different choices of γ = 0.99, 0.9 and 0.8. It is intriguing that the choices of $\gamma = 0.9$ and $\gamma = 0.8$ both fail to find any strategy to play the game. The reward and loss both keep almost static during the training process, which indicates that the γ should be as close to 1 as possible to maintain a good performance during training.

Fig. 6 and Fig. 7 show the results of different choices of the memory buffer size. The memory buffer stores the transitions the agent once explored and should be sampled for the network training. The memory size should be large enough to store the hardly explored successful action of the agent, otherwise the agent will lose the precious information and have to explore how to move the wall again. The two figures proved that as the memory size becomes smaller, the final performance of

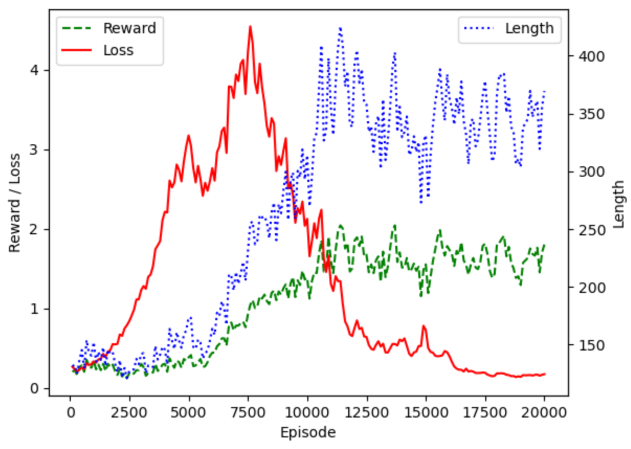

Fig. 3. Baseline Results.

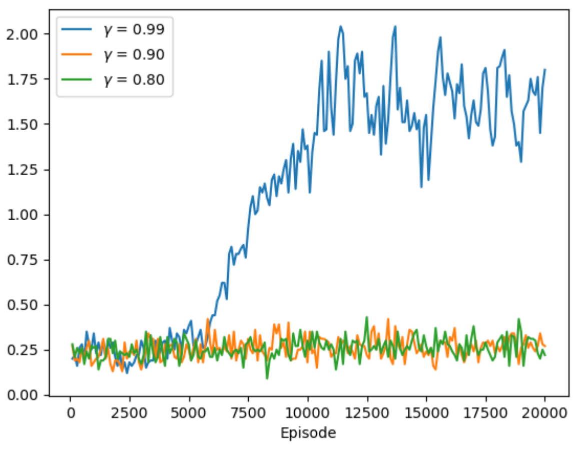

Fig. 4. Reward during Training Process by Different Gamma.

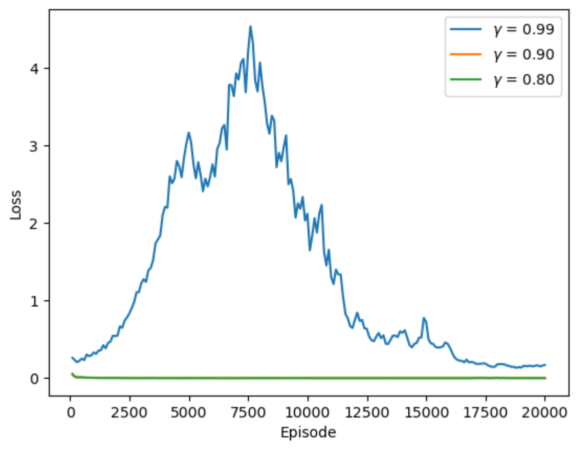

Fig. 5. Loss during Training Process by Different Gamma.

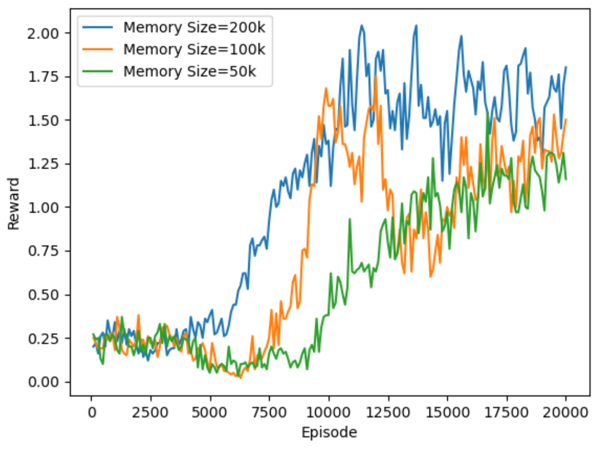

Fig. 6. Reward during Training Process by Different Memory Size.

the agents becomes worse. However, if we want to store the explored transitions as large as possible, a extremely large memory of the buffer should be allocated, which comes into a trade-off between the storage cost and the final performance.

The exploration step defines how many steps the agent need to randomly explore. Fig. 8 shows the epsilon decay process choosing different explore steps. It should be noted that it makes little difference if reduce the explore steps by 200k. Therefore, the reward curves shown in Fig. 9 are similar, which indicates that the 800k exploration steps are enough for this total episode number.

B. Hopper

1) Environment and Game Characteristics: The ”Hopper” benchmark in Mujoco is a popular reinforcement learning environment used to evaluate algorithms and agents. It simulates a one-legged robot or grasshopper-like creature and requires the agent to control its movements to hop forward while maintaining balance.

In this environment, the agent receives observations that capture the hopper’s body position, velocity, angle, and joint angular velocities. The agent’s action is to apply continuous forces to the hopper’s joints, coordinating its movements to generate propulsion and stability during the hopping motion. The agent’s reward combines the forward progress reward, the balance and stability reward and the energy efficiency reward.

2) The Actor and Critic Network: Fig. 10 shows the framework of actor and critic network used for this game. The actor network consists of two fully connected layers with one hidden layer between the input and output. The input of the actor is the state, which is a vector of 11 dimensions in this game specifically, and the output is the values of all actions, which indicates the force applied on the hopper. The critic has similar framework with a large hidden layer. The critic takes the state and action together as the input, and outputs the value it estimates for this state and action.

3) Parameters: The baseline parameters for DDPG are shown in Fig. II. The total number of episodes is chosen to be 5k limited by the computation resources.

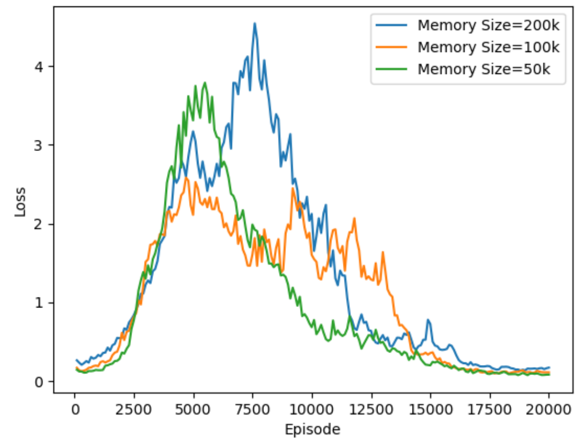

Fig. 7. Loss during Training Process by Different Memory Size.

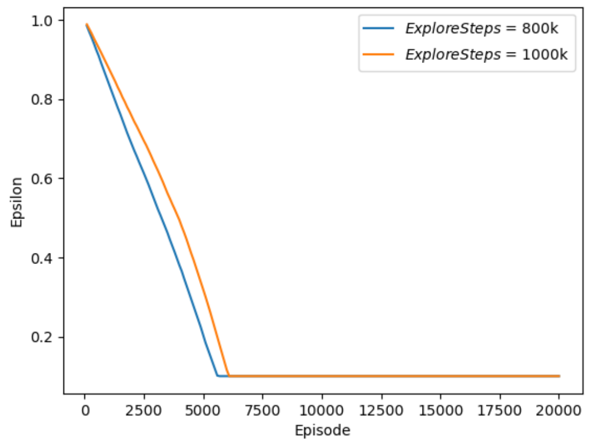

Fig. 8. Epsilon Decay of Different Exploration Steps.

4) Result and Comparisons: In DDPG, we are still curious about the effects of memory buffer size and the value of γ on the final performance of the agent. Fig. 11 and Fig. 12 show the reward and loss curves during training. Compared to the training period of DQN, the reward variance of DDPG is kind of large. However, seeing from the loss figure, the performance of $\gamma = 0.8$ is still worse than the other two choices of γ.

Finally, researching on the effects of memory size, Fig. 13 and Fig. 14 show that if the memory size is tool small, the agent is hard to learn anything from the environment, which

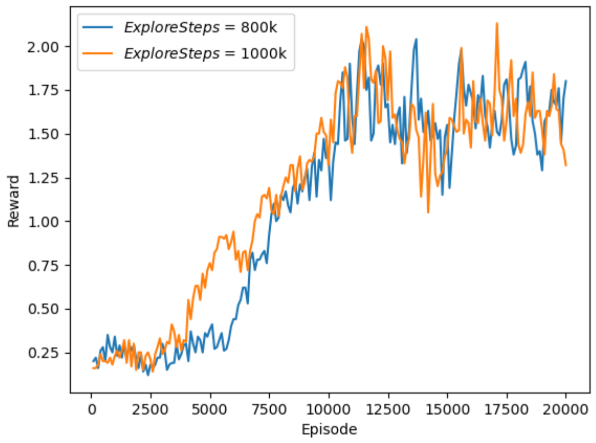

Fig. 9. Reward during Training Process by Different Explore Steps.

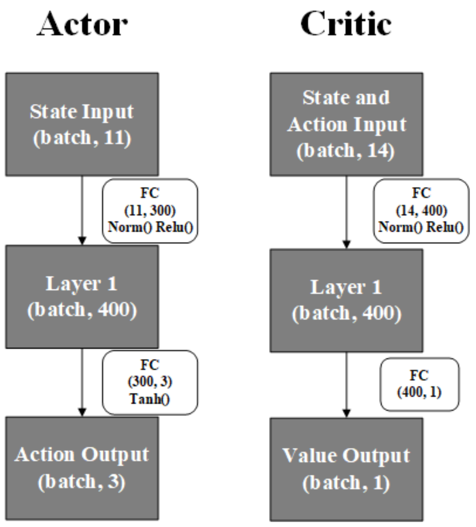

Fig. 10. Actor and Critic Network used fo DDPG.

reflects the significant importance of the memory size.

IV. Conclusion

In conclusion, this paper investigated the effects of different discount factors and memory sizes on the performance of DQN and DDPG algorithms in solving the "Breakout" and "Hopper" problems, respectively. Our findings provide valuable insights into the importance of these parameters in reinforcement learning. We observed that the choice of discount factor influenced the balance between short-term and long-term rewards, while larger memory sizes improved the learning stability of the algorithms. These results contribute to a better understanding of parameter selection and can guide future research in optimizing reinforcement learning algorithms for various tasks.

TABLE II
Baseline Parameters for DDPG

Parameter	Value
learning rate	1×10^{-3}
γ	0.99
episodes	5k
τ	0.001
memory size	200k
batch size	32

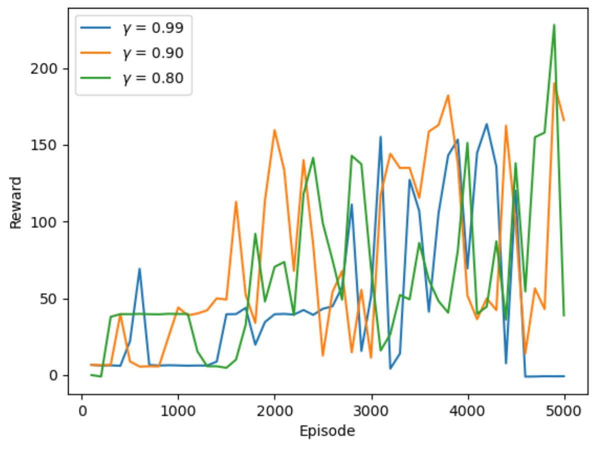

Fig. 11. Reward during Training Process by Different Gamma.

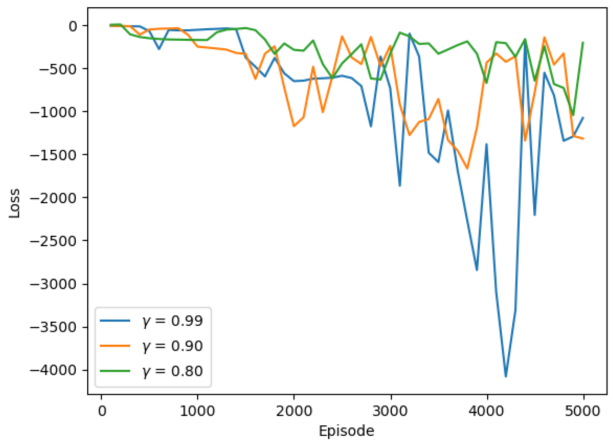

Fig. 12. Loss during Training Process by Different Explore Steps.

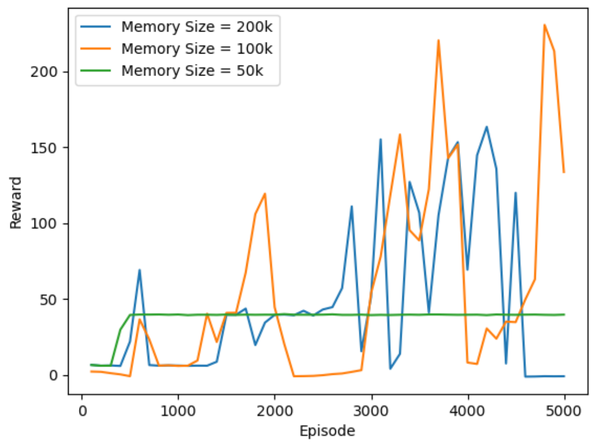

Fig. 13. Reward during Training Process by Different Memory Size.

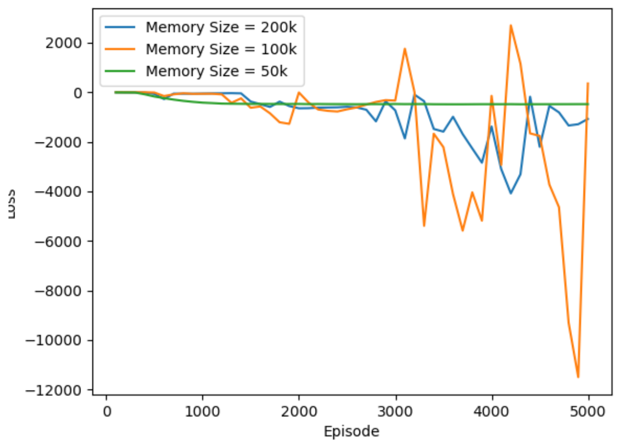

Fig. 14. Loss during Training Process by Different Memory Size.

References

[1] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto *et al.*, "The ai index 2021 annual report," *arXiv preprint arXiv:2103.06312*, 2021.
[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.
[3] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.
[4] 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.
[5] 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.