Final Project Text

Yiwen Liang, Kerry Ye

May 2024

In this project, we aim to introduce a reinforcement learning methodology paper titled "Continuous Control with Deep Reinforcement Learning" by Lillicrap, Timothy P., et al., through both text and videos. Given the diverse audience levels, we will begin by providing a broad overview of reinforcement learning. Subsequently, we will delve into various sub-topics within reinforcement learning, gradually narrowing down to the proposed algorithm in the paper, which is the **Deep Deterministic Policy Gradient (DDPG)** algorithm. Both text summary and the video presentation are available in our GitHub post, https://github.com/yiwen-liang/PHP_2650_Final_Project.

1 Reinforcement Learning

Reinforcement learning (RL) is a machine learning method where an agent learns to make decisions by navigating an environment to achieve an optimal outcome. Unlike supervised learning, which relies on labeled data, and unsupervised learning, which uncovers patterns in unlabeled data, reinforcement learning hinges on a trial-and-error learning process. The goal of the agent is to learn a policy, which is a mapping from states to actions, in a way that maximizes the cumulative reward it receives over time.

The key concepts in RL are as follows:

Agent: The ML algorithm that learns to interact with the environment. It makes decisions based on the state of the environment and the rewards it receives.

Environment: The problem that the agent interacts with. It provides feedback to the agent in the form of rewards or penalties based on the actions taken by the agent.

State: The current situation of the environment at a particular time.

Action: The decision or choice made by the agent at a given state.

Reward: The feedback provided by the environment to the agent after it takes an action. It indicates how favorable or unfavorable the action taken by the agent.

A reinforcement learning model can be represented by:

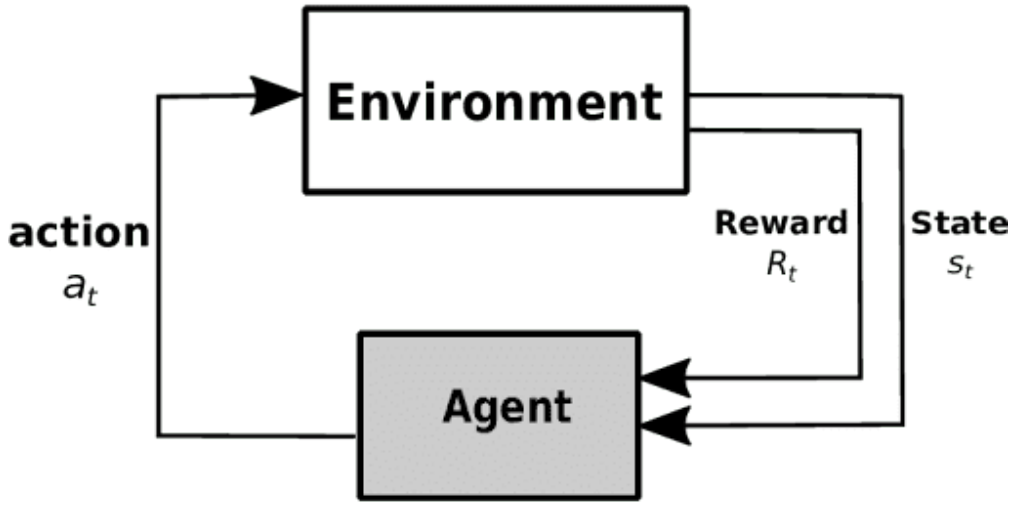

Figure 1: Graphic Representation of Reinforcement Learning Model

In RL, the agent at a specific state (S_t) interacts with the environment by taking actions (a_t). Based on these actions, the environment provides feedback in the form of rewards or penalties to the agent (R_t).

The key challenge in reinforcement learning is the trade-off between exploitation and exploration. Exploitation involves taking actions that the agent believes will lead to the maximum immediate reward based on its existing knowledge. Exploration, on the other hand, refers to the process of the agent trying out different actions to discover potentially beneficial actions. To maximize its rewards, the agent must prefer the known actions and find the successful one in producing reward. But to discover such actions, it has to try actions that it has not yet explored. Too much exploration may result in inefficiency, while too much exploitation

may lead to the agent missing out on potentially better actions or failing to adapt to new environment. Therefore, balancing exploitation and exploration is essential in reinforcement learning.

The representation of the reinforcement learning algorithm can be shown as:

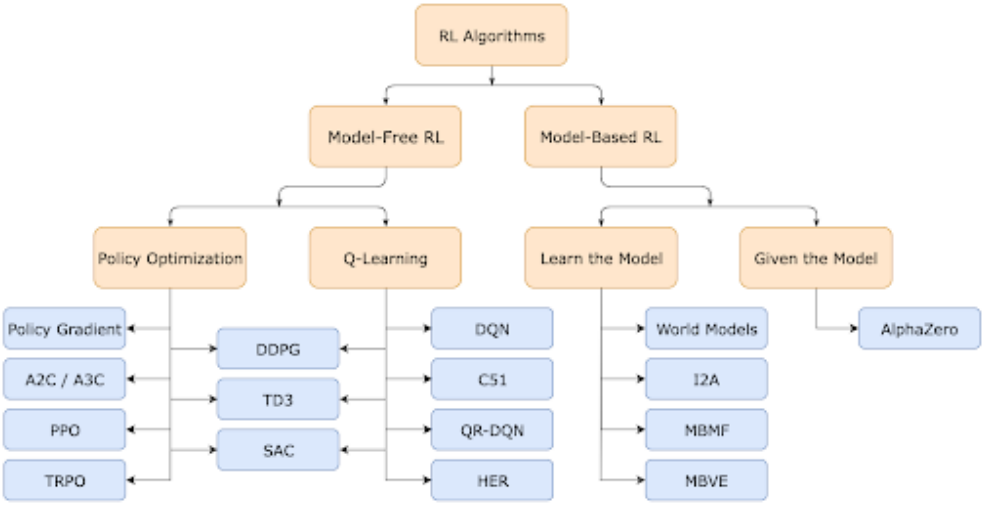

Figure 2: Graphic Representation of Reinforcement Learning Algorithm

The reinforcement learning algorithm can be categorized into two types: Model-Free and Model-Based algorithms. Specifically, for Model-Free algorithm, it can be divided into Policy Optimization and Q-Learning.

Our introduced method, DDPG, is a Model-Free algorithm that combines ideas from both policy optimization and Q-learning.

2 Model-Free v.s. Model-Based Algorithm

Model-free and model-based reinforcement learning (RL) are two fundamental approaches used by agents to learn optimal behavior in environments.

In model-free reinforcement learning (RL), agents learn from direct interactions with the environment without constructing a detailed model of its dynamics. They use trial-and-error methods to learn either a policy or a value function, adjusting their decisions based

on the rewards received and the states observed. Some common model-free algorithms are Q-learning, SARSA, and Policy Gradient methods like REINFORCE.

Conversely, model-based RL involves agents creating an explicit model of the environment, which includes transition probabilities and expected rewards. Using this model, agents can simulate future trajectories and make decisions that maximize expected rewards. Techniques in model-based RL include Dyna-Q, Monte Carlo Tree Search (MCTS), and Model Predictive Control (MPC).

In this project, we will focus on model-free RL, as the paper's DDPG algorithm is a model-free, off-policy actor-critic algorithm.

3 Policy Optimization

Policy optimization is a class of reinforcement learning algorithms that directly optimize the policy, which maps from states to actions, to maximize the expected return. Instead of estimating the value function or the dynamics of the environment, policy optimization algorithms focus on directly learning a good policy.

One of the ideas that the DDPG algorithm used is Deterministic Policy Gradient (DPG).

The DPG is one type of policy optimization algorithm that aims to directly learn deterministic policies, which map states to actions, in continuous action spaces. Its objective is to maximize the expected return over time. The DPG algorithm computes the gradient of the expected return with respect to the parameters of the policy and updates the policy parameters using gradient ascent.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s) Q^{\pi_\theta}(s, a) \right]$$

Where:

- $\nabla_\theta J(\theta)$ is the gradient of the expected return $J(\theta)$ with respect to the policy parameters θ.

- $\pi_\theta(s)$ is the deterministic policy parameterized by θ, representing the action selected in state s.

- $Q^{\pi_\theta}(s, a)$ is the state-action value function under policy π_θ, indicating the expected return when taking action a in state s.

The actor-critic method is a technique that combines aspects of both value-based and policy-based methods. In actor-critic algorithms, there are two main components: the actor and the critic. The actor is responsible for selecting actions while the critic is responsible for evaluating the effectiveness of the actions selected by the actor.

The DPG is a specific type of actor-critic algorithm. In actor-critic methods like DPG, there are two main components: the actor and the critic.

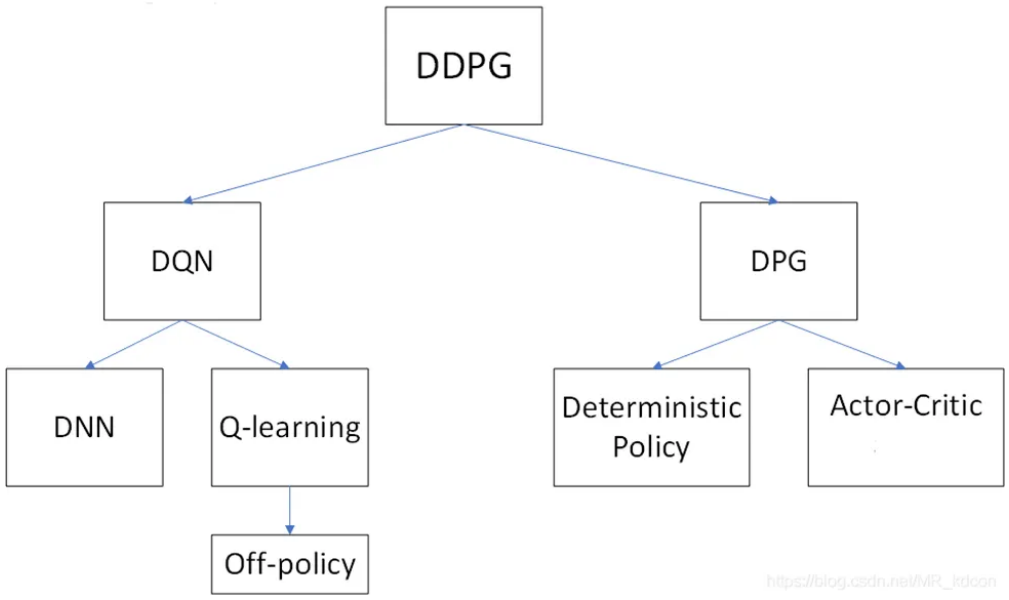

Figure 3: Relationship Between DPG and Actor-Critic

Actor (Deterministic Policy): The actor in DPG learns a deterministic policy, which directly maps states to actions without considering probability distributions. The goal of the deterministic policy is to select the action that maximizes the expected return.

Critic (Value Function): The critic in DPG learns the value function, which estimates the expected return for state-action pairs. It evaluates the effectiveness of the actions selected by the actor by estimating the value of those actions. The critic provides feedback to the

actor by estimating how effective the chosen actions are.

4 Q Learning

Q-learning is a model-free reinforcement learning algorithm utilized to obtain the optimal action-selection policy for a given Markov decision process (MDP). It focuses on learning a state-action value function, commonly represented as $Q(s, a)$, which estimates the expected return from taking action a in state s.

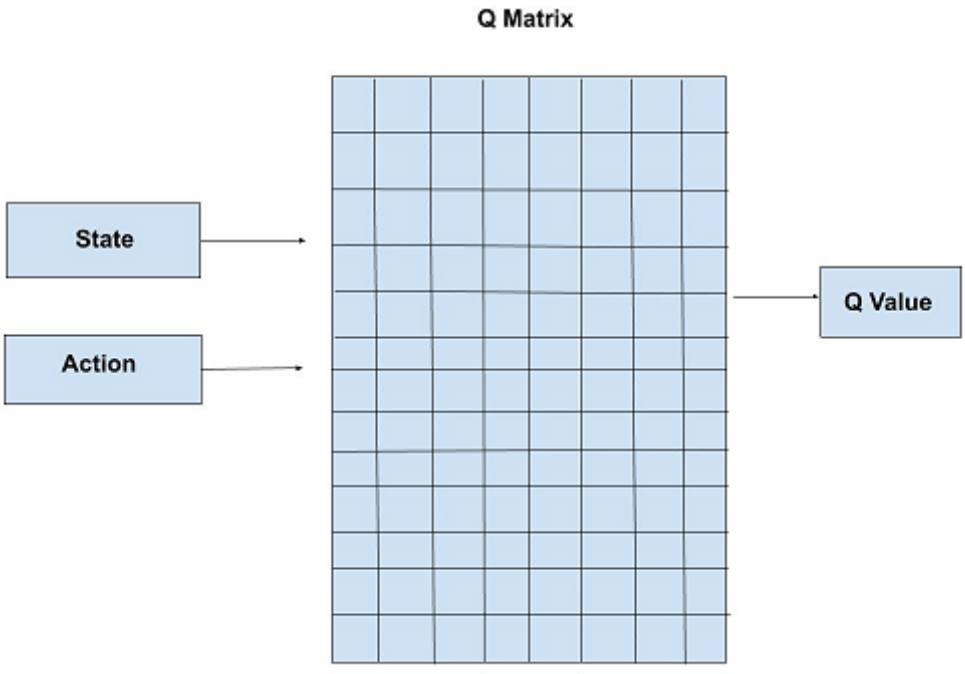

Figure 4: Graphic Representation of Q-Learning Algorithm

The Q-learning update rule is based on the Bellman equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

Where:

- $Q(s, a)$ is the state-action value function for state s and action a.

- α is the learning rate, determining the step size of the updates.

- r is the reward received after taking action a in state s.

- γ is the discount factor, representing the importance of future rewards.

- s' is the next state after taking action a in state s.

Another idea that the DDPG algorithm utilized is Deep Q-Network (DQN).

DQN is an extension of Q-learning that uses deep neural networks to approximate the Q-function. Instead of explicitly storing Q-values for each state-action pair, DQN learns to approximate the Q-function using a neural network. The network takes the state as input and outputs Q-values for all possible actions. DQN is trained using experience replay and target networks to improve stability and convergence.

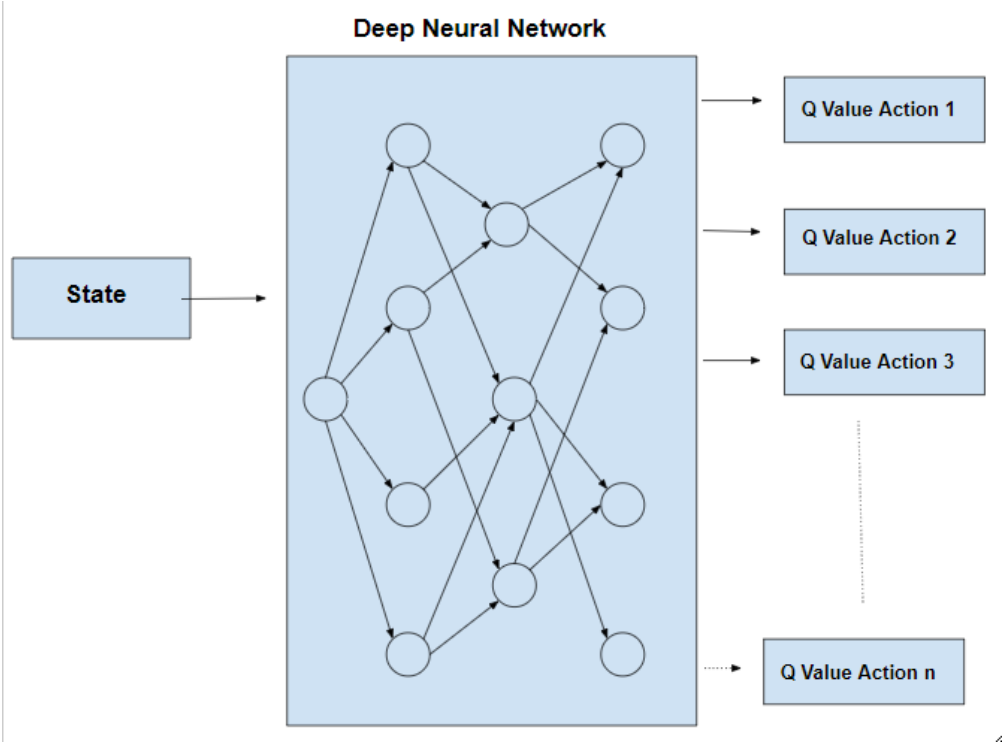

Figure 5: Graphic Representation of Deep Q-Network Algorithm

The DQN is a value-based reinforcement learning approach that approximates the maximum expected return for each state-action pair by learning an action-value function, $Q(s, a)$. It employs a neural network to model this action-value function and iteratively updates it using gradient descent. The aim is to reduce the temporal difference (TD) error, which is the discrepancy between the predicted and actual rewards following an action taken in a

given state. It can be represented as:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

Where:

- $L(\theta)$ is the loss function representing the mean squared temporal difference error.

- θ are the parameters of the Q-network.

- (s, a, r, s') are the state, action, reward, and next state tuples sampled from the replay buffer D.

- γ is the discount factor.

- θ^- are the parameters of the target Q-network, which are periodically updated.

5 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is a model-free, off-policy, actor-critic reinforcement learning algorithm using a deep neural network to approximate functions, enabling it to learn policies in continuous action spaces with high dimensions.

DDPG has four neural network parameters:

θ^Q: Q network

θ^μ: deterministic policy network

$\theta^{Q'}$: target Q network

$\theta^{\mu'}$: target policy network.

The DDPG algorithm adapts ideas from both DPG and DQN, leveraging the deterministic policy and actor-critic approach to operate over continuous action spaces from DPG, while also incorporating the ideas of replay buffer samples and target Q network from DQN.

The assumption that samples are independently and identically distributed can be violated when the samples are generated from sequential exploration in an environment. To address this issue, the DDPG paper introduced a replay buffer. The replay buffer acts as a finite-sized

cache where tuples (s_t, a_t, r_t, s_{t+1}) are stored. When the replay buffer reaches its capacity, the oldest samples are discarded to make room for new ones. During the training process, the algorithm samples a minibatch from the replay buffer at each time step to update both the actor and critic networks. This approach helps stabilize the training process by breaking the temporal correlation between consecutive samples.

The Q-Learning, in any given states, the optimal policy $\mu(s) = \text{argmax}Q(s, a)$ can be determined by maximizing the optimal action-value function Q(s, a). Consider the parameter θ^Q, we can optimize this parameter by minimizing the loss function:

$$L(\theta) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim \mathcal{E}} \left[(Q(s_t, a_t|\theta) - y_t)^2 \right]$$
$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$$

The DQN introduced two modifications to Q-Learning: the use of a replay buffer and separate target networks for calculating target values.

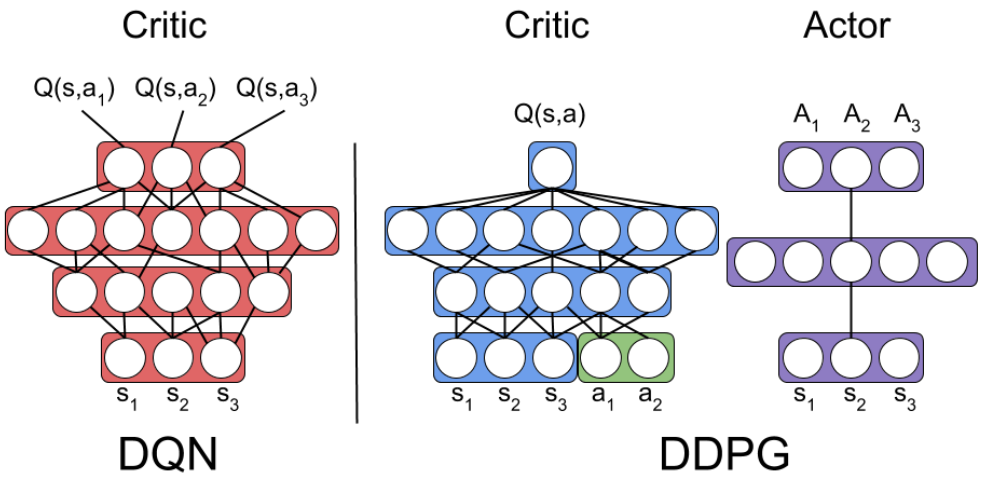

Figure 6: Difference between DGN and DDPG

However, directly implementing Q-learning with neural networks has shown to be unstable in numerous conditions. This instability arises because the network being updated, is also used in calculating the target value, leading to the potential divergence of the Q update. Instead,

the paper utilized actor-critic and "soft" target updates, rather than directly copying the weights periodically as in DQN. They create a copy of the actor and critic networks used to calculate the target values.

The DPG, which DDPG has combined from, is defined as:

$$\begin{aligned}\nabla_{\theta_\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)\big|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_a Q(s, a|\theta^Q)\big|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu}\mu(s|\theta^\mu)\big|_{s=s_t}\right],\end{aligned}$$

this approximates the gradient of the expected return with respect to the parameters of the actor network.

In Reinforcement Learning, while exploration in discrete action spaces often involves randomly choosing an action, in continuous action spaces, it is typically achieved by directly adding noise to the actions. The DDPG paper introduces the use of the Ornstein-Uhlenbeck process for this purpose, adding structured noise to the actions in continuous spaces.

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + N$$

The Ornstein-Uhlenbeck Process is a stochastic process that generates temporally correlated noise over time.

In the pseudo-code, the algorithm begins by initializing the critic network $Q(s, a|\theta^Q)$ and the actor network $\mu(s|\theta^\mu)$ with random weights. Additionally, target networks Q' and μ' are initialized with the same weights as the main networks. A replay buffer R is then initialized to store transitions observed during training. Following this, a random process N for action exploration is set up, and the initial observation state s_1 is received from the environment. The algorithm then enters a loop over episodes, where, at each time step t, an action a_t is selected based on the current policy μ and exploration noise N_t. This action is executed in the environment, resulting in a reward r_t and a new state s_{t+1}. The transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer R. Additionally, a random minibatch of transitions is sampled

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

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

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure 7: Pseudo-Code of the Deep Deterministic Policy Gradient Algorithm

from R for training the networks. The critic network is updated by minimizing the loss function, which computes the mean squared error between the predicted Q-values and the target values y_t. The actor policy is updated using the sampled policy gradient, approximated using the critic's gradients with respect to actions and the actor's gradients with respect to its parameters. Finally, the target networks Q' and μ' are updated by copying the weights from the main networks Q and μ periodically to improve training stability.

6 Application

DDPG is particularly effective for environments with continuous action spaces, which has led to its widespread adoption in numerous practical applications. Here are some key uses of DDPG:

1. **Robotics Control**: DDPG is employed in various robotics tasks, including manipulation by robotic arms, controlling locomotion in legged robots, and enabling robotic hands to grasp

objects. Its proficiency in managing continuous action spaces is ideal for robotic systems that operate in high-dimensional spaces.

2. **Autonomous Vehicles**: DDPG is utilized in the training of autonomous vehicles such as self-driving cars and drones. It facilitates the development of sophisticated control policies that ensure safe and efficient navigation in changing environments.

3. **Finance**: In the financial sector, DDPG is used for tasks like algorithmic trading and managing investment portfolios, learning strategies to maximize returns and reduce risk amidst fluctuating market conditions.

4. **Healthcare**: In healthcare, DDPG aids in personalized treatment planning and analyzing medical images, optimizing individualized treatment plans based on a patient's unique health profile and conditions.

5. **Game Playing**: DDPG is also used to train agents in video and board games, learning to manage game characters and actions in complex, continuous action space environments.

6. **Resource Management**: The algorithm is applied to optimize the allocation and management of resources across various fields such as energy systems, supply chain management, and telecommunications.

7. **Virtual Simulations**: DDPG trains agents within virtual simulations for tasks ranging from virtual prototyping and autonomous virtual characters to creating simulated training environments.

8. **Industrial Control**: In industrial settings, DDPG enhances automation and control systems, optimizing processes in manufacturing, managing robotic arm operations, and conducting predictive maintenance on machines.

The suitability of DDPG for continuous action spaces combined with its integration with deep neural networks makes it a powerful tool for a broad spectrum of practical applications across various industries.

References

GeeksforGeeks (2023). Deep Q-Learning. Available online at: https://www.geeksforgeeks.org/deep-q-learning/, last accessed on 2024-05-10.

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Marekar, A. (2022). How DDPG (Deep Deterministic Policy Gradient) Algorithms works in reinforcement learning ?, Medium. Available online at: https://medium.com/@amaresh.dm/how-ddpg-deep-deterministic-policy-gradient-algorithms-works-in-reinforcement-learning last accessed on 2024-05-10.

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Yoon, C. (2019). Deep Deterministic Policy Gradients Explained, Towards Data Science. Available online at: https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b, last accessed on 2024-05-10.