IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 1 November 2023, accepted 3 December 2023, date of publication 12 December 2023,
date of current version 15 December 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3341507

RESEARCH ARTICLE

Enhanced Deep Deterministic Policy Gradient Algorithm Using Grey Wolf Optimizer for Continuous Control Tasks

EBRAHIM HAMID HASAN SUMIEA[1,2], SAID JADID ABDULKADIR[1,2], (Senior Member, IEEE), MOHAMMED GAMAL RAGAB[1,2], SAFWAN MAHMOOD AL-SELWI[1,2], SULIAMN MOHAMED FATI[3], (Senior Member, IEEE), ALAWI ALQUSHAIBI[1,2], AND HITHAM ALHUSSIAN[1,2]

[1]Department of Computer and Information Sciences, Universiti Teknologi PETRONAS, Seri Iskandar 32610, Malaysia
[2]Center for Research in Data Science (CeRDaS), Universiti Teknologi PETRONAS, Seri Iskandar 32610, Malaysia
[3]Information Systems Department, Prince Sultan University, Riyadh 11586, Saudi Arabia

Corresponding author: Ebrahim Hamid Hasan Sumiea (ebrahim_22006040@utp.edu.my)

The authors would like to thank the Ministry of Higher Education(MOHE) - Malaysia for providing financial assistance under Fundamental Research Grant Scheme (FRGS/1/2022/ICT02/UTP/02/4), Universiti Teknologi PETRONAS under the Yayasan Universiti Teknologi PETRONAS (YUTP-FRG/015LC0-308) for providing the required facilities to conduct this research work, and Prince Sultan University - Riyadh Saudi Arabia for their support.

ABSTRACT Deep Reinforcement Learning (DRL) allows agents to make decisions in a specific environment based on a reward function, without prior knowledge. Adapting hyperparameters significantly impacts the learning process and time. Precise estimation of hyperparameters during DRL training poses a major challenge. To tackle this problem, this study utilizes Grey Wolf Optimization (GWO), a metaheuristic algorithm, to optimize the hyperparameters of the Deep Deterministic Policy Gradient (DDPG) algorithm for achieving optimal control strategy in two simulated Gymnasium environments provided by OpenAI. The ability to adapt hyperparameters accurately contributes to faster convergence and enhanced learning, ultimately leading to more efficient control strategies. The proposed DDPG-GWO algorithm is evaluated in the 2DRobot and MountainCarContinuous simulation environments, chosen for their ease of implementation. Our experimental results reveal that optimizing the hyperparameters of the DDPG using the GWO algorithm in the Gymnasium environments maximizes the total rewards during testing episodes while ensuring the stability of the learning policy. This is evident in comparing our proposed DDPG-GWO agent with optimized hyperparameters and the original DDPG. In the 2DRobot environment, the original DDPG had rewards ranging from −150 to −50, whereas, in the proposed DDPG-GWO, they ranged from −100 to 100 with a running average between 1 and 800 across 892 episodes. In the MountainCarContinuous environment, the original DDPG struggled with negative rewards, while the proposed DDPG-GWO achieved rewards between 20 and 80 over 218 episodes with a total of 490 timesteps.

INDEX TERMS Deep deterministic policy gradient, deep reinforcement learning, grey wolf optimization, hyperparameters optimization.

I. INTRODUCTION

The integration of RL with the Deep Q-Network (DQN) framework has been successfully achieved, utilizing Deep Neural Networks (DNN) as the foundation for Q-learning [1], [2]. DQN has shown remarkable performance across a variety of Atari games, thereby contributing to the advancement of diverse DRL systems [3], [4]. However, DQN's applicability was limited to tasks involving discrete and relatively small state and action spaces [5]. In contrast, many RL problems encompass continuous states and action spaces. Although DQN can handle continuous tasks by discretizing the

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

state and action spaces, this approach increases the entire unpredictability of the control mechanism [6]. To tackle and overcome this challenge, the Deterministic Policy Gradient (DPG) algorithm [7] was introduced, which combined DNN techniques and proved to be suitable for addressing continuous tasks. Consequently, the Deep Deterministic Policy Gradient (DDPG) was created [8]. Nevertheless, the DDPG algorithm is susceptible to insufficient exploration and intermittent instability during training [9].

Within the framework of a continuous control problem, the DDPG algorithm necessitates the pre-definition of a set of parameters, commonly referred to as hyperparameters, to enable autonomous exploration and learning in a complex environment. These hyperparameters encompass aspects such as the batch size, the size of the network, the exploration strategies, the learning rates, time steps, and others [10]. During the training time, these parameters are not dynamically optimized; instead, researchers rely on their knowledge to manually select the most suitable values. Hyperparameters configuration significantly impact the efficacy of learning processes, interactions with the environment, and the time required for learning [11]. Therefore, it is crucial to determine the most suitable hyperparameters carefully and accurately to enhance the performance of the model. Each environment possesses unique characteristics and complexities that require specific hyperparameters tailored to its requirements. Typically, a common approach to selecting these hyperparameters is through manual search, which demands expertise to identify robust sets of hyperparameters [12]. However, finding the optimal hyperparameters is a challenging task [13].

Recently, optimization has emerged as a captivating area of study across various domains, and nature-inspired metaheuristic optimization algorithms have gained considerable attention as promising techniques for achieving optimal solutions [14]. These algorithms have found applications in conjunction with AI methods due to several advantageous characteristics: (i) their ease of implementation and reliance on straightforward concepts, (ii) their ability to operate without requiring gradient information, (iii) their capability to overcome local optima, and (iv) their applicability to a wide range of problems spanning diverse research fields. Metaheuristic algorithms inspired by nature tackle optimization problems by mimicking biological or physical phenomena. These techniques can be broadly classified into numerous main categories, such as swarm-based methods, evolution-based methods, and physics-based methods [15].

Ant Colony Optimization (ACO) [16], and Particle Swarm Optimization (PSO) [17] are widely recognized as popular swarm-based optimization algorithms. PSO draws inspiration from the collective behavior of bird flocks, while ACO is motivated by the cooperative behavior observed in ant colonies. Another notable algorithm is the Grey Wolf Optimizer (GWO) [18], which involves initializing a population of grey wolves to represent potential solutions. GWO has demonstrated efficient problem-solving capabilities in literature [18].

A recent study by Faris et al. [19] has recently assessed the scientific applications of the GWO and reported promising results across various optimization problems. The high success rate of GWO in literature might appear from its remarkable characteristics relative to other swarm intelligence methods. This review underscores that GWO does not require prior knowledge of the search space and involves only a few parameters. Moreover, it is scalable, flexible, user-friendly, and straightforward, maintaining a fine balance between exploration and exploitation throughout the search process, resulting in excellent convergence.

In this research, we employed the GWO to enhance the DDPG algorithm performance in continuous control problems. To accomplish this, we validated our proposed DDPG-GWO method on Gymnasium environments, as they possess highly complex state spaces, continuous action spaces, and necessitate precise fine control. Moreover, OpenAI's simulation environments present a wide array of virtual spaces, enabling the training and evaluation of AI models within realistic and interactive scenarios spanning diverse domains [20]. To effectively adapt the DDPG algorithm to Gymnasium environments, this study focuses on the optimization of seven key hyperparameters: actor learning rate, critic learning rate, discount factor, exploration, batch size, Polyak averaging, and learning rate of target networks. These hyperparameters play a crucial role in governing the performance of the overall system [8]. To achieve this optimization task, the GWO is employed as the chosen metaheuristic optimization technique. GWO is selected based on its promising outcomes and potential for achieving desirable results in this context.

Therefore, we conducted a comprehensive evaluation of our agent in a training mode using OpenAI's simulation environments. The performance of our DDPG-GWO agent is compared against the original DDPG agent, which utilizes a list of the DDPG algorithm's hyperparameters. The evaluation results indicate that our agent exhibited superior performance in optimizing (maximizing) the cumulative rewards throughout the training episodes and during the test episodes, with a significant margin of improvement compared to its competitors. This article provides the following contributions:

- This study surveys the latest and most remarkable advancements in DDPG, highlighting the research endeavors dedicated to enhancing the optimization of its hyperparameters.
- Narrowing down the scope to optimizing a specific set of seven hyperparameters in the DDPG algorithms, these parameters are widely acknowledged as pivotal for augmenting the learning process's efficiency.
- Utilizing the GWO to identify the optimal settings of the chosen hyperparameters, enabling our agent to

effectively implement continuous control within the simulation environments provided by OpenAI.
- A performance comparison is conducted between the DDPG agent employing the optimized hyperparameters by the GWO and the original DDPG agent utilizing a set of hyperparameters.

The rest of this paper is structured as follows: in section II, we provide the fundamental background of this study. Section III presents the discussion on related work. The detailed explanation of the proposed approach is presented in section IV. In section V, the experimental results of the proposed method are discussed. Section VI provides the discussion and analysis. Finally, section VII concludes the paper.

II. FUNDAMENTAL BACKGROUND

In this section, a fundamental overview concerning DRL is elaborated in (Sub-section II-A), DDPG (Sub-section II-B), GWO (Sub-section II-C), and OpenAI simulation environments (Sub-section II-D).

A. DEEP REINFORCEMENT LEARNING

DRL is an advanced approach that combines deep learning (DL), which involves training deep neural networks, with RL, a technique used to teach agents to make decisions in an environment through trial and error [21]. Two well-known algorithms in this field are the DQN algorithm and the DDPG algorithm. The DDPG algorithm builds upon the DQN algorithm, enhancing its capabilities. One key aspect of the DDPG algorithm is that it does not require a predefined model and can learn from the experience gathered while acting in the environment off-policy. This flexibility allows the algorithm to adapt and improve its decision-making abilities over time.

B. DDPG ALGORITHM

The DDPG algorithm, as introduced by Lillicrap et al. [22], represents a distinctive approach in DRL that combines the principles of being both off-policy and model-free. This fusion effectively integrates the strengths of the DQN algorithm and the Actor-Critic (AC) algorithm, melding DL and RL principles.

While the DDPG algorithm shares a structural framework with the AC algorithm, it distinguishes itself through a more refined neural network configuration. Notably, while the DQN algorithm excels in discrete problem domains, the DDPG algorithm builds upon DQN's experiences to tackle the complexities of continuous control tasks, achieving end-to-end learning. Illustrated in Figure 1, the DDPG's structure embodies an actor-network that processes input states, selects actions, and generates action values. Concurrently, a critic network evaluates the chosen action's efficacy and computes the associated reward. The DDPG algorithm's procedural steps are elaborated as follows:

1) Initializing the neural network parameters. The actor, guided by the behavior policy, chooses an action. To promote exploration, the action produced by the policy network is augmented with noise N_t. This modified action, denoted as a_t, is then sent to the environment for execution.
$$a_t = \mu\left(s_t \mid \theta^{\mu}\right) + N_t \tag{1}$$
2) Once the environment performs the action a_t, the resulting outcome is observed, including the reward r_t received and the new state a_{t+1} of the system.
3) The actor archives the state transition s_t, a_t, r_t, s_{t+1} in the replay memory, which functions as the training dataset for the online network.
4) DDPG employs two separate neural network copies, namely the policy and the Q network. These copies consist of the online network and the target network. The policy network's update method can be described as follows:
$$\begin{cases} \text{online} : Q\left(s, a \mid \theta^{\mu}\right), & \text{gradient update } \theta^{\mu}, \\ \text{target} : Q\left(s, a \mid \theta^{\mu'}\right), & \text{soft update } \theta^{\mu'}. \end{cases} \tag{2}$$
The update procedure for the Q network is outlined as follows:
$$\begin{cases} \text{online} : Q\left(s, a \mid \theta^{Q}\right), & \text{gradient update } \theta^{Q}, \\ \text{target} : Q\left(s, a \mid \theta^{Q'}\right), & \text{soft update } \theta^{Q'}. \end{cases} \tag{3}$$
N transition data are selected randomly from the replay memory to serve as training data for the online policy network and online network. Each individual transition data within a mini-batch is denoted by s_t, a_t, r_t, s_{t+1}.
5) During the critical step, compute the gradient of the Q values for the online Q network.
The loss of the Q network is defined as
$$L = \frac{1}{N}\sum_{i}\left(y_i - Q\left(s_i, a_i \mid \theta^{Q}\right)\right)^2,$$
$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'\left(s_{i+1} \mid \theta^{\mu'}\right) \mid \theta^{Q'}\right) \tag{4}$$
The derivative of the loss function L with respect to the parameters θ^{Q} can be computed as $\nabla_{\theta} L$. This calculation involves utilizing the target policy network μ' and the target Q network Q'.
6) Update the online Q network, and update θ^{Q} using the Adam optimizer.
7) Within the actor component, the policy gradient is computed by calculating the gradient of the policy network.
$$\nabla_{\theta^{\mu}} J_{\beta}(\mu) \approx \frac{1}{N} \cdot \left(\nabla_{\alpha} Q\left(s, a \mid \theta^{Q}\right)\Big|_{s=s_i, a=w(s_k)} \cdot \nabla_{\theta^{\mu}} \mu\left(s \mid \theta^{\mu}\right)\Big|_{s=s_i} \right) \tag{5}$$
8) Updating the online policy networks: updating with the Adam optimizing.

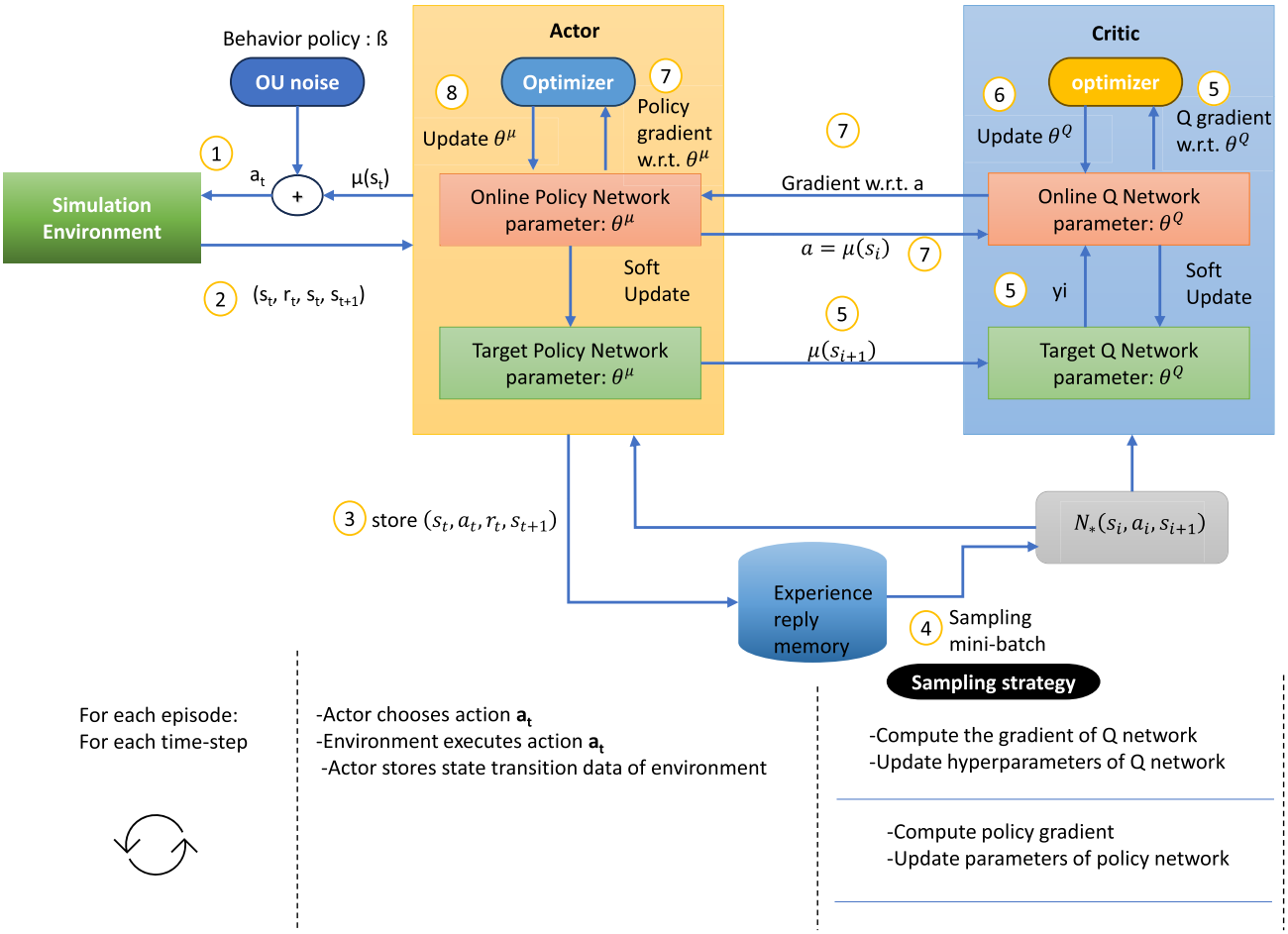

FIGURE 1. Structure of DDPG algorithm.

9) The hyperparameters of the target networks adopting the method of soft update:

$$\begin{cases} \theta^{Q'} \longleftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \\ \theta^{\mu'} \longleftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{cases} \tag{6}$$

Typically, the DDPG algorithm employs the Actor-Critic framework to iteratively train the policy and the Q network by facilitating interactions among the environment, actor, and critic components.

C. GREY WOLF OPTIMIZATION

The GWO algorithm proposed by Mirjalili et al. [18], draws inspiration from the social intelligence observed in grey wolves, which exhibit a preference for living in groups consisting of 5 to 12 individuals. To replicate the leadership hierarchy observed in GWO, the algorithm incorporates four levels: alpha α, beta β, delta δ, and omega ω.

In this hierarchy, the alpha wolf is known as the male and female leader of the pack. It holds the primary responsibility for making decisions such as hunting, selecting sleeping locations, and determining wake-up times. Beta, on the other hand, assists alpha in decision-making and focuses mainly on providing feedback and suggestions. Delta fulfills multiple roles within the group, serving as hunters, sentinels, caretakers, scouts, and elders, and controls the omega wolves by following the commands of the alpha and beta wolves. The omega wolves, in turn, are required to obey all other wolves within the group [23].

In the context of the GWO algorithm, the hunting process is orchestrated by the α, β, and δ wolves, with the ω wolves obediently adhering to their guidance. The encircling behavior characteristic of GWO can be computed using this formula:

$$\vec{X}\,(t+1) = \vec{X}_p\,(t) + \vec{A} \cdot \vec{D} \tag{7}$$

$\vec{A}$ and $\vec{D}$ denotes coefficient vectors, $\vec{X}_p$ denotes the vector of the prey's positions, and X denotes the positions of the wolves in a d-dimensional space, where D denotes the number of variables. The variable (t) denotes the number of iterations, and $\vec{D}$ is defined as follows:

$$\vec{D} = \left| \vec{C} \cdot \vec{X}_p(t) - \vec{X}\,(t) \right| \tag{8}$$

where the following notation is used to represent $\vec{A}$ and $\vec{C}$:

$$\vec{A} = 2\vec{a} \cdot \vec{r_1} - \vec{a} \tag{9}$$

$$\vec{C} = 2 \cdot \vec{r_2} \tag{10}$$

where r_1, r_2 are selected randomly in the normal range of zero to unity. Over the course of iterations, the components of $\vec{a}$ are linearly decreased from 2 to 0. Using Eq 8, a grey wolf can approach the prey by changing its position around the prey randomly.

The values of x_1, x_2, and x_3 are defined and computed as follows:

$$\begin{aligned}\overrightarrow{x_1} &= \overrightarrow{X_\alpha} - A_1 \cdot \left(\overrightarrow{D_\alpha}\right),\\ \overrightarrow{x_2} &= \overrightarrow{X_\beta} - A_2 \cdot \left(\overrightarrow{D_\beta}\right),\\ \overrightarrow{x_3} &= \overrightarrow{X_\delta} - A_3 \cdot \left(\overrightarrow{D_\delta}\right)\end{aligned} \tag{11}$$

At a given iteration t, the vectors $\overrightarrow{x_1}$, $\overrightarrow{x_2}$, and $\overrightarrow{x_3}$ represent the three top-performing wolves (solutions) within the swarm. These values are determined by following the calculations outlined in Equation (3) for A_1, A_2, and A_3. Additionally, the vectors $\overrightarrow{D_\alpha}$, $\overrightarrow{D_\beta}$, and $\overrightarrow{D_\delta}$ are computed using the method described in (11).

$$\begin{aligned}\overrightarrow{D_\alpha} &= \left|\overrightarrow{C_1} \cdot \overrightarrow{X}_\alpha - \overrightarrow{X}\right|,\\ \overrightarrow{D_\beta} &= \left|\overrightarrow{C_2} \cdot \overrightarrow{X}_\beta - \overrightarrow{X}\right|,\\ \overrightarrow{D_\delta} &= \left|\overrightarrow{C_3} \cdot \overrightarrow{X}_\delta - \overrightarrow{X}\right|\end{aligned} \tag{12}$$

The vectors $\overrightarrow{C_1}$, $\overrightarrow{C_2}$, and $\overrightarrow{C_3}$ are determined by performing calculations specified in (4).

One of the key elements in the GWO algorithm for balancing exploitation and exploration is the vector $\vec{a}$. In the baseline paper of this algorithm, it is recommended to gradually decrease the vector's value for each dimension in a linear manner, starting from 2 and ending at 0, as the number of iterations progresses. The equation used to update the vector is as follows:

$$\overrightarrow{a} = 2 - t \cdot \frac{2}{\underset{i}{max}\ ter} \tag{13}$$

In the given equation, the variable t represents the current iteration number, while "ter" represents the total number of iterations for the optimization process.

D. OPENAI GYMNASIUM

The OpenAI Gymnasium has become a popular toolkit in the machine learning community for RL research [24]. This work follows the established structure used by researchers and builds upon it by creating 2D Robot and Mountain Car Continuous simulation environments. OpenAI Gymnasium places a primary emphasis on the episodic setting of RL, aiming to maximize the expected total reward per episode and achieve satisfactory performance quickly. The toolkit also aims to integrate the GymnasiumAPI with physical robotic hardware, allowing for the validation of RL algorithms in real-world environments [24].

OpenAI Gymnasium [25] includes a collection of environments known as Partially Observable Markov Decision Processes (POMDPs), which will continue to expand over time. The initial beta release of Gymnasium featured various environments, including:

- **Classic control and toy text**: They encompass small-size tasks that are commonly encountered in the RL literature.
- **Algorithmic**: Tasks that revolve around computation, such as performing operations like adding multi-digit numbers or reversing sequences. Memory is frequently essential for these tasks, and their level of challenge can be finely tuned by modifying the length of the sequence.
- **Atari**: This category involves classic Atari games, where input is derived from RAM or screen images, and the Arcade Learning Environment is leveraged [26].
- **Board games**: Initially featuring the Go on 9×9 and 19×19 board games, with the Pachi engine acting as a competitor [27].
- **2D and 3D robots**: These entail simulated robot control tasks that utilize the MuJoCo physics engine, acclaimed for its speed and precision in simulating robots [28]. Some of these tasks were adapted from RLLab [29].

Since its initial release, the collection of environments has expanded to encompass additional options, including those grounded in the Box2D, an open-source physics engine, or the Doom game engine through VizDoom [30].

1) OPENAI GYMNASIUM ENVIRONMENTS

In our study, we leverage two OpenAI Gymnasium environments: the 2D Robot Arm Environment and the Mountain Car Continuous Environment.

- **Mountain Car Continuous**: The Mountain Car Continuous environment is part of the Classic Control environments. It features a deterministic Markov Decision Process with a car initially placed at the bottom of a sinusoidal valley. The car can only accelerate in either direction to reach a goal state on top of the right hill. This version offers continuous action spaces, with observations containing position and velocity information. Actions are directional forces in the range $[-1,1]$ scaled by 0.0015. Transition dynamics are governed by specific equations, with inelastic collisions, position, and velocity constraints. Rewards include negative penalties for large actions and a positive reward upon reaching the goal. Episodes end either by reaching the goal or after 999 steps [25].
- **2D Robot Arm Environment**: The presented OpenAI Gymnasium environment, named RobotArm-V0, simulates a two-link robot arm operating in a 2D space using PyGame. In RobotArm-V0, the robot comprises two 100-pixel length links, and the objective is to reach a randomly generated red target point in each episode. The observation space includes target positions in both x and y directions, as well as the current positions of the two arm joints in radians. The action space consists of discrete actions to hold the current joint angles

or increment/decrement them, with a default rate of 0.01 radians. The reward function penalizes the robot if the current tip-to-target distance is greater than or equal to the previous distance, and rewards it if the distance is within a tolerance of 10 pixels. The episode terminates when the reward reaches −10 or +10 [25].

III. RELATED WORK

This section presents the discussion on related work. First of all, Chen et al. [31] introduced a method for adapting hyperparameters in DRL based on the Bayesian approach. Their study stands out as the most comprehensive investigation of RL hyperparameters to date, specifically focusing on configuring the AlphaGo algorithm. They achieved the automatic refinement of game-playing hyperparameters for AlphaGo, an achievement that conventional methods are unable to attain. The application of Bayesian optimization not only improved the winning probabilities of AlphaGo but also generated valuable data that can be used to develop enhanced versions of self-play agents incorporating Monte Carlo Tree Search (MCTS). Nonetheless, this approach requires a substantial number of experiments and relies on advanced information. Additionally, it is primarily effective for adapting individual hyperparameters rather than a range of hyperparameters.

Liessner et al. [11] introduced a model-based approach for optimizing hyperparameters in the DDPG algorithm, which demonstrated effectiveness in real-world industrial applications. The authors addressed the challenge of limited training time by imposing strict constraints on the DDPG algorithm within the specific domain. By successfully optimizing the hyperparameters under these time limitations, they achieved improved performance. However, one limitation of the study is that the strict constraints on the DDPG algorithm are not applicable to other domains or applications.

Oktay et al. [32] suggested employing an artificial bee colony (ABC) algorithm to fine-tune the weights of an artificial neural network (ANN) that functions as the objective function in an optimization procedure. The ANN is trained using specific input and output datasets, and the objective function, which relies on the ANN, is enhanced using the ABC algorithm to achieve improved outcomes. This study demonstrates the application of metaheuristic optimization approaches alongside artificial intelligence methods.

Another independent study by Sehgal et al. [33] utilized a Genetic Algorithm (GA) to optimize hyperparameters in the HER (Hindsight Experience Replay)+DDPG algorithm. The GA-based approach effectively identified hyperparameters that required fewer training epochs while still achieving enhanced task performance. The research employed a range of robotics manipulation tasks encompassing actions such as push, slide, reach, fetch, place, pick, and open operations, serving to showcase the effectiveness of the proposed methodology.

Elfwing et al. [34] introduced a method that shares similarities with population-based training (PBT). They presented an alternative technique called OMPAC, which focuses on the evolutionary mechanism and represents the initial strategy for adapting multiple hyperparameters in DRL using a population-based approach. In a related study by Jaderberg et al. [35], the authors also employed a PBT to optimize a group of models along with their hyperparameter configurations. This was accomplished within a predetermined computational budget, with the objective of achieving optimal performance. The proposed approach demonstrated promising results in various domains such as Machine Translation, DRL, and GANs. However, it should be noted that PBT relies on basic stochastic perturbations to adapt hyperparameters, which may not effectively track changes in potentially optimal hyperparameter configurations over time.

Another study by Zhou et al. [36] introduced an online method for hyperparameter optimization for DRL. This method enhanced the existing PBT procedure, resulting in efficient online adaptation of hyperparameters. The authors incorporated a recombination operation inspired by GA into the population optimization process. This recombination operation accelerated the convergence of the population towards the optimal hyperparameter configuration. The authors empirically validated the effectiveness of this approach and demonstrated improved results compared to the classical PBT method, which aligns with their research findings.

A recent study by Parker-Holder et al. [37] introduced a novel and proven PBT-style approach called Population-Based Bandits (PB2). PB2 is a procedure that can identify exceptional hyperparameter configurations using a smaller number of agents compared to traditional PBT. Through multiple RL trials, the authors demonstrated that PB2 can achieve remarkable performance levels while adhering to a moderate computational budget. In another study by Moghanian et al. [38], a swarm-based metaheuristic algorithm is employed to minimize errors in intrusion detection. In the novel approach, the Grasshopper Optimization Algorithm (GOA) is harnessed to enhance the precision of artificial neural networks (ANNs) in order to decrease the rate of intrusion detection errors.

Additionally, the authors in [39] and [40] discussed the advantages and disadvantages of different deep architectures, as well as the different optimization methods that have been used. Alqushaibi et al. [41] propose a new weight optimization method based on the sine cosine algorithm (SCA).Balogun et al. [42] evaluate a number of different methods on a real-world dataset of software defects and show that they can significantly improve the performance of defect prediction models.

To sum up, optimization strategies like PBT, which aim to learn optimal schedules for hyperparameters rather than relying on fixed settings, have shown promising results. However, these strategies can be susceptible to sample ineffectiveness, which can impact their performance. Through the review of existing literature, it is evident that many papers have focused on utilizing grid search [43], Bayesian methods, or GA [33] to optimize various hyperparameters

TABLE 1. Original DDPG algorithm's hyperparameters in 2DRobot and MountainCarContinuous environments.

Hyperparameter	2DRobot	MountainCarContinuous
Episodes	1000	1000
Time steps	500	500
Explorations	0.1	0
Critic learning rate	0.0005	0.002
Actor learning rate	0.0005	0.001
Discount factor	0.99	0.99
Memory size	100000	100000
Polyak averaging	0.001	0.005
Critic layer sizes	(16, 16)	(8, 8)
Actor layer sizes	(16, 16)	(8, 8)
Batch size	128	64
Noise type	OU	OU
Noise std. dev.	0.5	0.3

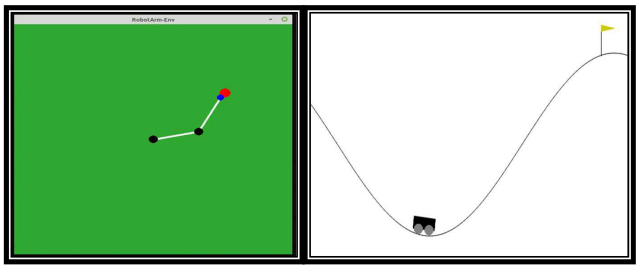

FIGURE 2. The experimental environments setup: (A) "2DRobot" comprises a 2DRobotic arm setup, posing a challenging control task to the learning agents, and (B) "MountainCarContinuous" involves a continuous-action variant of the classic "Mountain Car" problem, where an agent must learn to navigate a car to surmount a steep hill using continuous acceleration.

in DRL. While these approaches have demonstrated some success, they also have noticeable limitations. To overcome these limitations and achieve remarkable results, this work proposes the utilization of a metaheuristic optimization algorithm known as GWO. The GWO algorithm is employed to optimize the hyperparameters of the DDPG algorithm in two simulated Gymnasium environments provided by OpenAI, namely the 2DRobot and MountainCarContinuous simulation environments.

IV. THE PROPOSED FRAMEWORK

This section introduces the paper's primary contribution, which involves the implementation of the GWO algorithm to explore the hyperparameters space of the DDPG algorithm. The objective is to identify the set of hyperparameters that maximizes the total rewards obtained in the 2DRobot and MountainCarContinuous simulation environments and then compare the optimized results with the original DDPG algorithm results. The subsequent Table 1 presents the hyperparameters of the original DDPG algorithm used for comparison with the optimized DDPG-GWO algorithm within the simulation environments of 2DRobot and MountainCarContinuous as shown in Figure 2. These hyperparameters were selected by GWO to fine-tune the DDPG algorithm to maximize the total rewards achieved in the aforementioned simulation environments.

The rest of this section is divided into two sub-sections: the training of DDPG networks using GWO (IV-A), and the learning process (IV-B).

A. TRAINING OF DDPG NETWORKS USING GWO

As shown in Figure 3 and Algorithm 1, the training process of DDPG networks using GWO begins by initializing the DDPG networks (actor and critic networks) and the GWO algorithm. The DDPG networks learn the policy and value functions, while the GWO algorithm generates a group of grey wolves for hyperparameter search. Each grey wolf's chosen hyperparameter set is evaluated by executing the DDPG algorithm and measuring its performance. A reward function reflecting the task's objective is used to assess performance.

The GWO algorithm updates the best solutions found so far, representing superior hyperparameter configurations. Each grey wolf's hyperparameter settings are modified based on the position of the optimal solutions. This collective intelligence guides the search toward promising regions in the hyperparameter space. Once optimized, the DDPG networks are trained using the updated configuration.

The training phase involves iterative interactions with the environment, experience collection, and network parameter updates. Network performance is periodically evaluated to monitor progress. If satisfactory performance or convergence is achieved, the training process can be halted.

If performance is unsatisfactory, the GWO algorithm refines the hyperparameters, and the DDPG networks undergo further training. The iterative nature of training enables continuous exploration and exploitation of the hyperparameter space. This leads to the identification of hyperparameter configurations maximizing the DDPG algorithm's overall performance.

B. LEARNING PROCESS

In this section, we explicitly explain the learning process employed to optimize the hyperparameters of the DDPG algorithm using the GWO approach. The training process of each wolf in GWO to optimize DDPG hyperparameters can be represented using equations. By denoting the hyperparameters to be optimized as h, and the position of each wolf as x. In GWO, the three types of wolves: alpha, beta, and delta, are represented as x_α, x_β, and x_δ, respectively. The position update equation for each wolf in the GWO can be defined as follows:

$$x_{\text{new}} = x + A \cdot D \tag{14}$$

where x_{new} is the updated position, x is the current position, A is the updating amplitude, and D is the random vector.

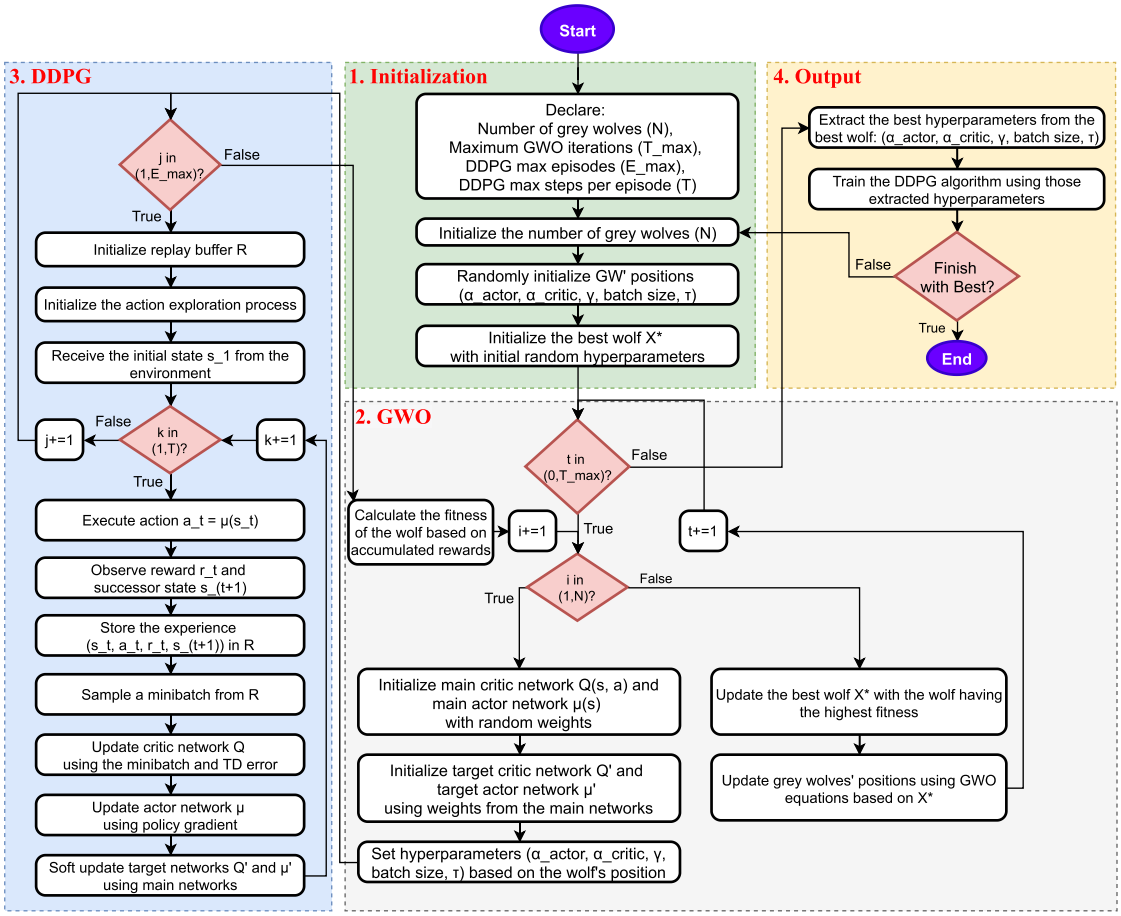

FIGURE 3. The flowchart of the proposed DDPG-GWO hyperparameter optimization in this study.

The updating amplitude A is calculated as:

$$A = 2a \cdot r - a \tag{15}$$

where a is a linearly decreasing parameter, and r is a random number between 0 and 1.

The random vector D is calculated using the positions of the alpha α, beta β, delta $\delta$and omega ω wolves:

$$D = |C \cdot x_\alpha - x| + |C \cdot x_\beta - x| + |C \cdot x_\delta - x| + |C \cdot x_\omega - x| \tag{16}$$

where C is a constant that controls the influence of each wolf's position on the update.

Finally, the updated hyperparameters h_{new} can be obtained by applying the GWO update equation to each element of h:

$$h_{\text{new}}[i] = h[i] + x_{\text{new}}[i] \tag{17}$$

where $h[i]$ and $x_{\text{new}}[i]$ denote the i-th element of h and x_{new}, respectively.

By iteratively applying the above equations for each wolf in the GWO, the hyperparameters h can be optimized to enhance the performance of the DDPG algorithm.

The GWO algorithm refines the positions of its wolf population, and these refined positions are subsequently used to update the hyperparameters of the DDPG algorithm. This iterative process continues to optimize the hyperparameters, ultimately enhancing the performance of the DDPG algorithm.

V. EXPERIMENTS

This section presents the experimental analysis conducted to compare the performance of the original DDPG algorithm with our proposed DDPG-GWO method, in two distinct simulation environments: the 2DRobot and the MountainCarContinuous. By evaluating the results obtained from both algorithms, we aim to assess the effectiveness and improvements brought by the DDPG-GWO method. A comparison is made between these optimized hyperparameters and the original DDPG hyperparameters used in the same environments as those presented by Lillicrap et al. [8].

The rest of this section is structured as follows: the experimental setting (V-A), the 2D robot environment results (V-B), and the MountainCarContinuous environment results (V-C).

A. EXPERIMENTAL SETTINGS

Table 2 shows the experimental settings to perform the experimental analysis. We employed a hardware setup

Algorithm 1 GWO-DDPG: Optimizing DDPG Hyperparameters

Require: Number of grey wolves (N), maximum GWO iterations ($T_{\max}$), DDPG max episodes ($E_{\max}$), DDPG max steps per episode (T)

Ensure: Optimized hyperparameters for DDPG: (α_{actor}, α_{critic}, γ, batch size, τ)

Initialize number of grey wolves (N)
Initialize grey wolves' positions (α_{actor}, α_{critic}, γ, batch size, τ) randomly
Initialize best wolf X^* with initial random hyperparameters
$t \leftarrow 0$
while $t < T_{\max}$ **do**
  for $i = 1$ **to** N **do**
    Initialize main critic network $Q(s, a)$ and main actor network $\mu(s)$ with random weights
    Initialize target critic network Q' and target actor network μ' with weights from main networks
    Set hyperparameters (α_{actor}, α_{critic}, γ, batch size, τ) based on wolf position
    for $j = 1$ **to** $E_{\max}$ **do**
      Initialize replay buffer R
      Initialize action exploration process
      Receive initial state s_1 from environment
      for $k = 1$ **to** T **do**
        Execute action $a_t = \mu(s_t)$
        Observe reward r_t and successor state s_{t+1}
        Store experience (s_t, a_t, r_t, s_{t+1}) in R
        Sample minibatch from R
        Update critic network Q using minibatch and TD error
        Update actor network μ using policy gradient
        Soft update target networks Q' and μ' with main networks
      end for
    end for
    Calculate the fitness of the wolf based on accumulated rewards
  end for
  Update best wolf X^* with the wolf having highest fitness
  Update grey wolves' positions using GWO equations based on X^*
  $t \leftarrow t + 1$
end while
return Hyperparameters from best wolf: (α_{actor}, α_{critic}, γ, batch size, τ)

comprising a Windows 11 Pro OS running on an Intel(R) Core(TM) i7-9700 CPU clocked at 3.00GHz with 16.0 GB of RAM (15.8 GB usable). The software environment consisted of the Python programming language, OpenAI Gymnasium toolkit, and TensorFlow library for DRL computations.

TABLE 2. Experimental settings.

A: Hardware Specifications	
Operating System	Windows 11 Pro 64-bit (10.0, Build 22621)
System Model	OptiPlex 3070
Processor	Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz (8 CPUs), 3.0GHz
Memory	16.0 GB
B: Software Specifications	
Python	3.10.5
TensorFlow	2.11.0
Anaconda	22.11.1
Keras	2.11.0
Gymnasium	0.15.4
Matplot	3.5.1
C: Original DDPG Configurations	
Learning rate	0.001
Discount factor (gamma)	0.99
Target network update rate	0.001
Replay buffer size	100,000
Batch size	64
D: Proposed DDPG-GWO Configurations	
GWO population size	20
Max. number of iterations	100
Exploration factor	2
Exploitation factor	0.5

For both the 2DRobot and MountainCarContinuous environments, we utilized the DDPG algorithm as the baseline. The DDPG algorithm was implemented with the following configurations: a learning rate of 0.001, a discount factor (gamma) of 0.99, a target network update rate of 0.001, a replay buffer size of 100,000, and a batch size of 64.

In contrast, our proposed approach, DDPG-GWO, introduced an optimization technique called GWO into the DDPG framework. This hybrid algorithm incorporated the following parameters: a GWO population size of 20, a maximum number of iterations set to 100, and exploration and exploitation factors set to 2 and 0.5, respectively.

B. RESULTS FOR THE 2DROBOT ENVIRONMENT

This sub-section presents the 2DRobot environment's experimental analysis conducted to compare the performance of the original DDPG algorithm (V-B1) with our proposed DDPG-GWO (V-B2).

1) ORIGINAL DDPG ALGORITHM'S RESULTS IN 2DROBOT ENVIRONMENT

The 2DRobot agent's performance was unstable during its training process as shown in Figure 4. In the beginning, it showed poor performance with negative rewards ranging from −150 to −50. However, there was some improvement

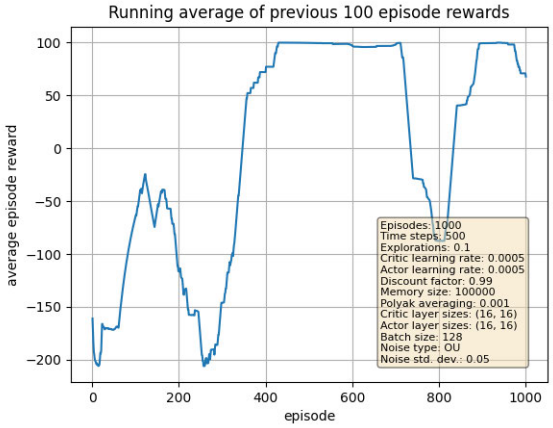

FIGURE 4. Original DDPG algorithm's results in 2DRobot environment.

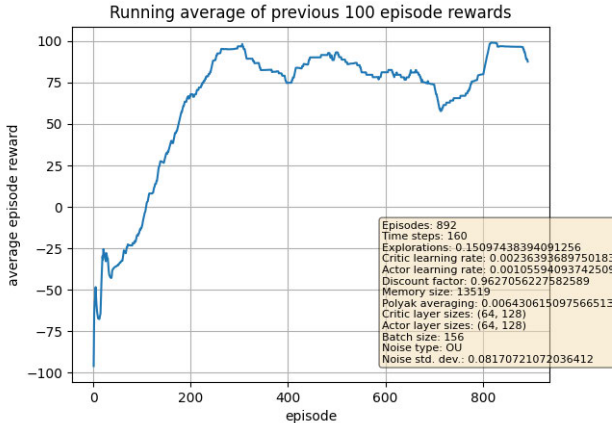

FIGURE 5. DDPG-GWO algorithm's results in 2DRobot environment.

with positive rewards between episodes 400 and 800. Unfortunately, the agent's behavior regressed to a sub-optimal state after that. The inconsistency in its performance could be attributed to various factors, such as hyperparameters like a high exploration rate of 0.1 and relatively small learning rates of 0.0005. Additionally, the agent's architecture, with 16,16 layers for both the actor and critic, not have been adequate for the task at hand.

2) DDPG-GWO ALGORITHM'S RESULTS IN 2DROBOT ENVIRONMENT

Figure 5 illustrates promising outcomes for the optimized DDPG-GWO algorithm compared with the original DDPG algorithm (Figrue 4). Throughout 892 episodes, the average episode reward ranged from −100 to 100, with a running average of the previous 100 episode rewards fluctuating between 1 and 800. The algorithm effectively explored the environment using a moderate exploration rate of 0.15. The critic and actor networks had learning rates of 0.002 and 0.001, respectively. A discount factor of 0.96 was applied to future rewards, enabling the agent to balance immediate and future rewards. Experience replay during training was facilitated by a memory size of 13519, contributing to improved performance. The target networks' stable update mechanism was achieved through Polyak averaging with a value of 0.006. Exploration utilized an Ornstein–Uhlenbeck (OU) process as a noise source, with the noise standard deviation set to 0.08.

C. RESULTS FOR THE MOUNTAINCARCONTINUOUS ENVIRONMENT

This sub-section presents the MountainCarContinuous environment's experimental analysis conducted to compare the performance of the original DDPG algorithm (V-C1) with our proposed DDPG-GWO (V-C2).

1) ORIGINAL DDPG ALGORITHM'S RESULTS IN MOUNTAINCARCONTINUOUS ENVIRONMENT

As depicted in Figure 6, the DDPG results for the MountainCarContinuous task are not satisfactory. One major problem

TABLE 3. Comparative analysis between the original DDPG and the proposed DDPG-GWO algorithm in 2DRobot and MountainCarContinuous environments.

Environment	Algorithm	Average Episode Reward	Running Average (over X episodes)
2DRobot	DDPG	-150 to 50	1 to 1000
	DDPG-GWO	-25 to 85	1 to 892
MountainCar Continuous	DDPG	-15 to 15	1 to 1000
	DDPG-GWO	20 to 80	1 to 218

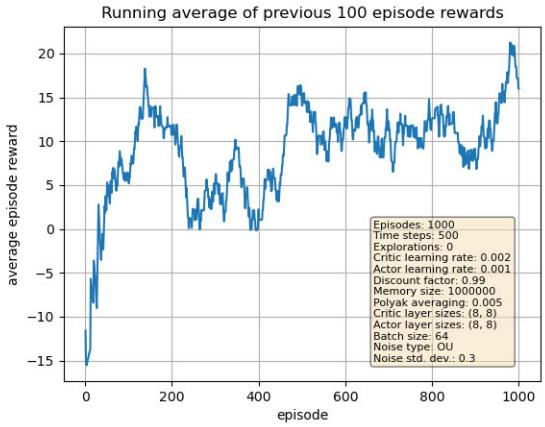

FIGURE 6. Original DDPG algorithm's results in MountainCarContinuous environment.

is the rather low average reward, which even goes negative at times, indicating frequent failures in reaching the goal. It is possible that the chosen actor and critic network structures are not fit enough for the task, and the high noise standard deviation may be causing erratic policy updates. Moreover, the algorithm seems to be struggling to effectively utilize past experiences, pointing to potential issues with the memory or the learning process itself.

TABLE 4. Hyperparameters comparison between the original DDPG and the proposed DDPG-GWO algorithms.

Hyperparameters	Original DDPG		Proposed DDPG-GWO	
	2DRobot	MountainCar Continuous	2DRobot	MountainCar Continuous
Episodes	1000	1000	892	218
Time steps	500	500	160	490
Explorations	0.1	0	0.1501	0.1810
Critic learning rate	0.0005	0.002	0.00236	0.00346
Actor learning rate	0.0005	0.001	0.00106	0.00032
Discount factor	0.99	0.99	0.9627	0.9890
Memory size	100000	100000	13519	52670
Polyak averaging	0.001	0.005	0.00643	0.00034
Critic layer sizes	(16, 16)	(8, 8)	(64, 128)	(63, 128)
Actor layer sizes	(16, 16)	(8, 8)	(64, 128)	(64, 128)
Batch size	128	64	156	171
Noise type	OU	OU	OU	OU
Noise std. dev.	0.5	0.3	0.0817	0.1840

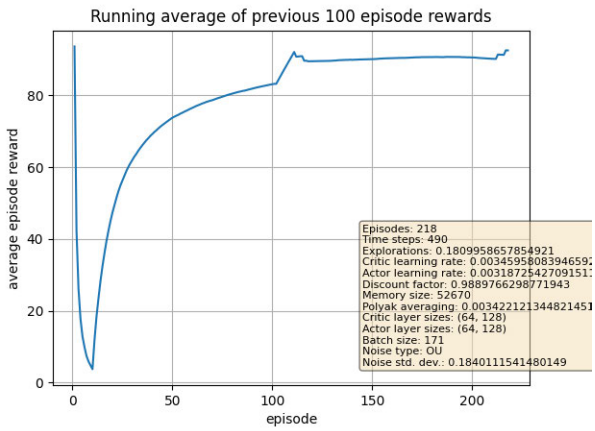

FIGURE 7. DDPG-GWO algorithm's results in MountainCarContinuous environment.

2) DDPG-GWO ALGORITHM'S RESULTS IN MOUNTAINCARCONTINUOUS ENVIRONMENT

The integration of GWO and DDPG in the MountainCarContinuous environment demonstrated remarkable improvements in hyperparameter optimization, as shown in Figure 7, leading to a significant boost in performance. The average episode reward witnessed a substantial increase, going from 80 to 20, indicating a marked enhancement in learning efficiency. The algorithm achieved a running average of the previous 100 episode rewards throughout 218 episodes, with a total of 490 time steps. Stability was ensured through the utilization of OU noise with a standard deviation of 0.184. These outcomes strongly suggest that the DDPG-GWO approach holds great potential to elevate the performance of the algorithm across a wide range of real-world applications.

VI. DISCUSSION AND ANALYSIS

The performance comparison analysis between the original DDPG and the proposed DDPG-GWO is illustrated in Figure 8. It indicates that the DDPG-GWO algorithm outperforms the original DDPG algorithm in both environments 2DRobot and MountainCarContinuous. In the 2DRobot environment, the DDPG-GWO algorithm demonstrated promising outcomes with a moderate exploration rate of 0.15 and optimized hyperparameters. It effectively explored the environment, achieving a running average of 100-episode rewards between 1 and 800. On the other hand, DDPG alone exhibited unstable performance with negative rewards, indicating difficulties in learning an effective policy. This proves that the integration of GWO in the DDPG algorithm aids in better hyperparameter optimization and learning efficiency, resulting in more stable and improved performance.

Furthermore, Table 3 provides a comparative analysis of the original DDPG algorithm and the proposed DDPG-GWO algorithm in 2DRobot and MountainCarContinuous environments. It shows that our proposed DDPG-GWO algorithm outperforms the original DDPG algorithm, achieving an average episode reward between -25 to 85 compared to -150 to 50 of DDPG. Additionally, the running average of DDPG-GWO over X episodes (892) is significantly higher than the original DDPG (1000). Similarly, in the MountainCarContinuous environment, the DDPG-GWO performs better with an average episode reward between 20 to 80, while the original DDPG ranges from -15 to 15. Moreover, the running average of DDPG-GWO (218) is superior to that of DDPG (1000) over the same number of episodes.

The proposed DDPG-GWO algorithm in this study holds significant theoretical and practical implications. The theoretical significance lies in its effective hyperparameter optimization, addressing a critical challenge in DRL. By employing GWO as a metaheuristic algorithm, DDPG-GWO demonstrates improved learning performance with faster convergence rates, enhancing control strategies in the simulated Gymnasium environments. This research enriches

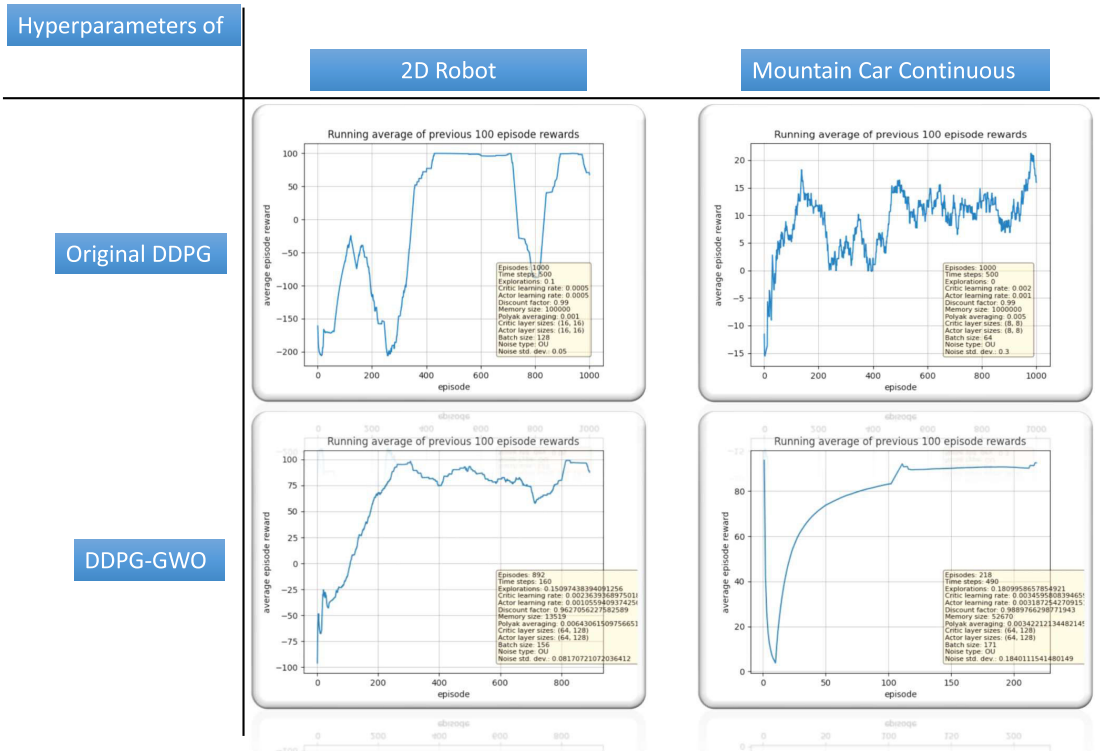

FIGURE 8. Comparison between the original DDPG and the proposed DDPG-GWO in 2DRobot and MountainCarContinuous.

the theoretical foundations of DRL algorithms and explores the application of nature-inspired optimization techniques, contributing to the advancement of DRL methodologies.

On the practical front, the DDPG-GWO algorithm offers real-world applicability. Its potential to optimize decision-making processes in industries like robotics, autonomous vehicles, and finance holds significant promise. The algorithm's ability to handle complex continuous action spaces showcases its relevance and adaptability to diverse scenarios. Additionally, by reducing training time and resource requirements, the DDPG-GWO presents cost-effective and efficient AI-driven systems, paving the way for more practical and impactful implementations in dynamic and uncertain environments. Overall, the successful integration of GWO into DDPG signifies the potential of metaheuristic algorithms to enhance learning performance and control strategy, with implications spanning across various industries and research areas.

Table 4 shows the hyperparameters comparison between the original DDPG and the proposed DDPG-GWO algorithms.

VII. CONCLUSION AND FUTURE WORKS

To conclude, the DDPG-GWO resulted in significant improvements in hyperparameter optimization and learning efficiency in both the 2DRobot and MountainCarContinuous environments. In the 2DRobot environment, the optimized hyperparameters led to rewards ranging from −100 to 100 across 892 episodes, achieving a balanced trade-off between immediate and future rewards with a 0.96 discount factor. Stability during training was ensured through experience replay with a memory size of 13519 and Polyak averaging 0.006. Similarly, in the MountainCarContinuous environment, DDPG-GWO notably enhanced the average episode reward from 80 to 20 over 218 episodes, indicating a marked improvement in learning efficiency. The stability was maintained with the OU noise process having a standard deviation of 0.184. Nevertheless, DDPG's performance in both environments fell short, likely due to the actor-critic network's insufficient complexity, high noise standard deviation, and ineffective utilization of past experiences. These findings highlight the value of DDPG-GWO in elevating performance and learning efficiency across diverse real-world applications, while also underscoring potential areas for future improvement.

The paper's future endeavors involve bolstering DDPG algorithms to achieve better sample efficiency, managing high-dimensional inputs more effectively, and tackling non-stationarity. Researchers have the opportunity to investigate hybrid methods, fusing DDPG with other algorithms, and pushing the boundaries of theoretical comprehension. As DDPG gains enhanced robustness and dependability, its real-world applications in robotics, finance, and healthcare are expected to witness increased adoption.

ABBREVIATIONS

Table 5 contains definitions for all abbreviations used in this study.

TABLE 5. Abbreviations.

Abbreviations	Definition
AC	Actor-Critic
ABC	artificial bee colony
DNN	Deep Neural Networks
DL	Deep Learning
DDPG	Deep Deterministic Policy Gradient
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
GA	Genetic Algorithm
GWO	Grey Wolf Optimization
OU	Ornstein–Uhlenbeck
PBT	Population Based Training
PB2	Population-Based Bandits
POMDPs	Partially Observable Markov Decision Processes

ACKNOWLEDGMENT

The authors would like to thank the Ministry of Higher Education(MOHE) - Malaysia for providing financial assistance under Fundamental Research Grant Scheme (FRGS/1/2022/ICT02/UTP/02/4), Universiti Teknologi PETRONAS under the Yayasan Universiti Teknologi PETRONAS (YUTP-FRG/015LC0-308) for providing the required facilities to conduct this research work, and Prince Sultan University - Riyadh Saudi Arabia for their support.

REFERENCES

[1] K. Arshad, R. F. Ali, A. Muneer, I. A. Aziz, S. Naseer, N. S. Khan, and S. M. Taib, "Deep reinforcement learning for anomaly detection: A systematic review," *IEEE Access*, vol. 10, pp. 124017–124035, 2022.
[2] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[3] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[4] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2016, pp. 1995–2003.
[5] J. Jäger, F. Helfenstein, and F. Scharf, *Bring Color to Deep Q-Networks: Limitations and Improvements of DQN Leading to Rainbow DQN*. Cham, Switzerland: Springer, 2021, pp. 135–149.
[6] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 4026–4034.
[7] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn.* PMLR, 2014, pp. 387–395.
[8] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[9] Z. Zheng, C. Yuan, Z. Lin, and Y. Cheng, "Self-adaptive double bootstrapped DDPG," in *Proc. Int. Joint Conf. Artif. Intell.*, 2018, pp. 3198–3204.
[10] A. Fuchs, Y. Heider, K. Wang, W. Sun, and M. Kaliske, "DNN2: A hyper-parameter reinforcement learning game for self-design of neural network based elasto-plastic constitutive descriptions," *Comput. Struct.*, vol. 249, Jun. 2021, Art. no. 106505.
[11] R. Liessner, J. Schmitt, A. Dietermann, and B. Bäker, "Hyperparameter optimization for deep reinforcement learning in vehicle energy management," in *Proc. 11th Int. Conf. Agents Artif. Intell.*, 2019, pp. 134–144.
[12] N. M. Ashraf, R. R. Mostafa, R. H. Sakr, and M. Z. Rashad, "Optimizing hyperparameters of deep reinforcement learning for autonomous driving based on whale optimization algorithm," *PLoS ONE*, vol. 16, no. 6, Jun. 2021, Art. no. e0252754.
[13] V. Sharma and A. K. Tripathi, "A systematic review of meta-heuristic algorithms in IoT based applications," *Array*, vol. 14, p. 100164, 2022.
[14] V. Sharma and A. K. Tripathi, "A systematic review of meta-heuristic algorithms in IoT based application," *Array*, vol. 14, Jul. 2022, Art. no. 100164.
[15] S. Zhao, T. Zhang, S. Ma, and M. Chen, "Dandelion optimizer: A nature-inspired Metaheuristic algorithm for engineering applications," *Eng. Appl. Artif. Intell.*, vol. 114, Sep. 2022, Art. no. 105075.
[16] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.
[17] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, 1995, ICNN.
[18] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.
[19] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, "Grey wolf optimizer: A review of recent variants and applications," *Neural Comput. Appl.*, vol. 30, no. 2, pp. 413–435, Jul. 2018.
[20] J. Arroyo, C. Manna, F. Spiessens, and L. Helsen, "An open-AI gym environment for the building optimization testing (boptest) framework," in *Proc. Building Simulation*, 2021, pp. 175–182.
[21] Y. Li, "Deep reinforcement learning: An overview," 2017, *arXiv:1701.07274*.
[22] 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," 2015, *arXiv:1509.02971*.
[23] Q. Al-Tashi, T. M. Shami, S. J. Abdulkadir, E. A. P. Akhir, A. Alwadain, H. Alhussain, A. Alqushaibi, H. M. Rais, A. Muneer, M. B. Saad, J. Wu, and S. Mirjalili, "Enhanced multi-objective grey wolf optimizer with Lévy flight and mutation operators for feature selection," *Comput. Syst. Sci. Eng.*, vol. 47, no. 2, pp. 1937–1966, 2023.
[24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*.
[25] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, K. G. Arjun, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. (Mar. 2023). *Gymnasium*. [Online]. Available: https://zenodo.org/record/8127025
[26] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, Jun. 2013.
[27] P. Baudis and J.-L. Gailly, "PACHI: State of the art open source go program," in *Advances in Computer Games*. Berlin, Germany: Springer, 2011, pp. 24–38.
[28] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 5026–5033.
[29] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," 2016, *arXiv:1604.06778*.
[30] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZDoom: A doom-based AI research platform for visual reinforcement learning," 2016, *arXiv:1605.02097*.
[31] Y. Chen, A. Huang, Z. Wang, I. Antonoglou, J. Schrittwieser, D. Silver, and N. de Freitas, "Bayesian optimization in AlphaGo," 2018, *arXiv:1812.06855*.
[32] T. Oktay, S. Arik, I. Turkmen, M. Uzun, and H. Celik, "Neural network based redesign of morphing UAV for simultaneous improvement of roll stability and maximum lift/drag ratio," *Aircr. Eng. Aerosp. Technol.*, vol. 90, no. 8, pp. 1203–1212, Nov. 2018.
[33] A. Sehgal, H. La, S. Louis, and H. Nguyen, "Deep reinforcement learning using genetic algorithm for parameter optimization," in *Proc. 3rd IEEE Int. Conf. Robotic Comput. (IRC)*, Feb. 2019, pp. 596–601.
[34] S. Elfwing, E. Uchibe, and K. Doya, "Online meta-learning by parallel algorithm competition," in *Proc. Genet. Evol. Comput. Conf.*, 2018, pp. 426–433.
[35] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," 2017, *arXiv:1711.09846*.

[36] Y. Zhou, W. Liu, and B. Li, "Efficient online hyperparameter adaptation for deep reinforcement learning," in *Applications of Evolutionary Computation: 22nd International Conference, EvoApplications 2019, Held as Part of EvoStar 2019, Leipzig, Germany, April 24–26, 2019, Proceedings 22*. Springer, 2019, pp. 141–155.
[37] J. Parker-Holder, V. Nguyen, and S. J. Roberts, "Provably efficient online hyperparameter optimization with population-based bandits," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 17200–17211.
[38] S. Moghanian, F. B. Saravi, G. Javidi, and E. O. Sheybani, "GOAMLP: Network intrusion detection with multilayer perceptron and grasshopper optimization algorithm," *IEEE Access*, vol. 8, pp. 215202–215213, 2020.
[39] N. Talpur, S. J. Abdulkadir, H. Alhussian, M. H. Hasan, N. Aziz, and A. Bamhdi, "A comprehensive review of deep neuro-fuzzy system architectures and their optimization methods," *Neural Comput. Appl.*, vol. 34, no. 3, pp. 1837–1875, Feb. 2022.
[40] S. M. Al-Selwi, M. F. Hassan, S. J. Abdulkadir, and A. Muneer, "LSTM inefficiency in long-term dependencies regression problems," *J. Adv. Res. Appl. Sci. Eng. Technol.*, vol. 30, no. 3, pp. 16–31, May 2023.
[41] A. Alqushaibi, S. J. Abdulkadir, H. M. Rais, Q. Al-Tashi, M. G. Ragab, and H. Alhussian, "Enhanced weight-optimized recurrent neural networks based on sine cosine algorithm for wave height prediction," *J. Mar. Sci. Eng.*, vol. 9, no. 5, p. 524, May 2021.
[42] A. O. Balogun, S. Basri, S. A. Jadid, S. Mahamad, M. A. Al-momani, A. O. Bajeh, and A. K. Alazzawi, "Search-based wrapper feature selection methods in software defect prediction: An empirical analysis," in *Intelligent Algorithms in Software Engineering: Proceedings of the 9th Computer Science On-line Conference 2020*, vol. 1. Cham, Switzerland: Springer, 2020, pp. 492–503, doi: 10.1007/978-3-030-51965-0_43.
[43] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proc. Workshop Mach. Learn. High-Perform. Comput. Environ.*, Nov. 2015, pp. 1–5.

EBRAHIM HAMID HASAN SUMIEA received the bachelor's degree in software engineering from Asia Pacific University (APU), Malaysia, in 2014, and the master's degree in MBA specialist engineering management from Universiti Malaysia Pahang, Malaysia, in 2021. He is currently pursuing the Ph.D. degree in information technology with Universiti Teknologi PETRONAS (UTP), Malaysia. He has a total experience of five years in the industry. His industry working experience is related to software engineering and engineering management. His research interests include artificial intelligence, machine learning, deep reinforcement learning, metaheuristic algorithms, and optimization.

SAID JADID ABDULKADIR (Senior Member, IEEE) received the B.Sc. degree in computer science from Moi University, the M.Sc. degree in computer science from Universiti Teknologi Malaysia (UTM), and the Ph.D. degree in information technology from Universiti Teknologi PETRONAS (UTP). He is currently an Associate Professor and a member of the Centre for Research in Data Science (CeRDaS), UTP. He is also involved in flagship consultancy projects for PETRONAS under pipeline integrity, materials corrosion, and inspection. His research interests include machine learning, deep learning architectures, optimizations, and applications in predictive analytics. He is serving as the Treasurer for the IEEE Computational Intelligence Society Malaysia Chapter and the Editor-in-Chief for *Platform*.

MOHAMMED GAMAL RAGAB received the Bachelor of Science degree in software engineering and the master's degree by research in machine learning from Universiti Teknologi PETRONAS, in 2019, where he is currently pursuing the Ph.D. degree in information technology. He demonstrated a keen interest in machine learning, computer vision, and data analysis, and has a particular interest in metastatic studies. His ongoing research builds on his previous work, with a focus on the development of new and innovative techniques for optimizing the performance of deep learning models.

SAFWAN MAHMOOD AL-SELWI received the bachelor's degree in software engineering from Taiz University, Yemen, in 2012, and the master's degree in computer applications from Bangalore University, India, in 2018. He is currently a Research Assistant with the Computer and Information Sciences Department, Universiti Teknologi PETRONAS (UTP), Malaysia. He has a total experience of eight years both in academic institutions and in the industry. His industry working experience is related to Android applications and website development. His research interests include artificial intelligence, machine learning, predictive and time-series analysis, metaheuristic algorithms, and optimization.

SULIAMN MOHAMED FATI (Senior Member, IEEE) received the B.Sc. degree from Ain Shams University, Egypt, in 2002, the M.Sc. degree from Cairo University, Egypt, in 2009, and the Ph.D. degree from Universiti Sains Malaysia (USM), Malaysia, in 2014. He is currently an Assistant Professor and the Chairperson of the Information Systems Department, Prince Sultan University (PSU), Saudi Arabia. His research interests include the Internet of Things, machine learning, social media mining, cloud computing, cloud computing security, and information security.

ALAWI ALQUSHAIBI received the B.Sc. degree in computer networks and security from Universiti Teknologi Malaysia, in 2012, and the master's degree by research from Universiti Teknologi PETRONAS, in 2021. He is an Academic Researcher. During his academic journey, he has acquired knowledge and skills in conducting independent research, producing academic writing, and teaching computer science courses. His research background in computer networks and security has led him to explore the application of machine learning techniques in data science analysis. His current research in image processing and GANs has the potential to make significant contributions to the field and impact various applications. His research interests include machine learning, data science, optimization, feature selection, classification, data analytics, and image processing, specifically in generative adversarial networks (GANs).

HITHAM ALHUSSIAN received the B.Sc. and M.Sc. degrees in computer science from the School of Mathematical Sciences, Khartoum University, Sudan, and the Ph.D. degree from Universiti Teknologi PETRONAS, Malaysia. He is currently an Associate Professor with the Department of Computer and Information Sciences and a Core Research Member of the Centre for Research in Data Science (CeRDaS), Universiti Teknologi PETRONAS. His main research interests include real-time parallel distributed systems, cloud computing, big data mining, machine learning, and secure computer-based management systems.