Complex & Intelligent Systems (2024) 10:4141–4155
https://doi.org/10.1007/s40747-024-01389-0

ORIGINAL ARTICLE

Time-aware MADDPG with LSTM for multi-agent obstacle avoidance: a comparative study

Enyu Zhao[1,2] · Ning Zhou[1,3] · Chanjuan Liu[1] · Houfu Su[1] · Yang Liu[1] · Jinmiao Cong[1]

Received: 7 April 2023 / Accepted: 10 February 2024 / Published online: 2 March 2024

Abstract
Intelligent agents and multi-agent systems are increasingly used in complex scenarios, such as controlling groups of drones and non-player characters in video games. In these applications, multi-agent navigation and obstacle avoidance are foundational functions. However, problems become more challenging with the increased complexity of the environment and the dynamic decision-making interactions among agents. The Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm is a classical multi-agent reinforcement learning algorithm successfully used to improve agents' performance. However, it ignores the temporal message hidden in agents' interaction with the environment and needs to be more efficient in scenarios with many agents due to its training technique. To address the limitations of MADDPG, we propose to explore modified algorithms of MADDPG for multi-agent navigation and obstacle avoidance. By combining MADDPG with Long Short-Term Memory (LSTM), we obtain the MADDPG-LSTMactor algorithm, which leverages continuous observations over time as input for the policy network, enabling the LSTM layer to capture hidden temporal patterns. Moreover, by simplifying the input of the critic network, we obtain the MADDPG-L algorithm for efficiency improvement in scenarios with many agents. Experimental results demonstrate that these algorithms outperform existing networks in the OpenAI multi-agent particle environment. We also conducted a comparative study of the LSTM-based approach with Transformer and self-attention models in the task of multi-agent navigation and obstacle avoidance. The results reveal that Transformer and self-attention do not consistently outperform LSTM. The LSTM-based model exhibits a favorable tradeoff across varying sequence lengths. Overall, this work addresses the limitations of MADDPG in multi-agent navigation and obstacle avoidance tasks, providing insights for developing intelligent agents and multi-agent systems.

Keywords Multi-agent RL · MADDPG · Navigation and obstacle avoidance · Transformer · Self-attention

Introduction

Intelligent agents have become increasingly prevalent in various industries, including manufacturing and military applications. However, as their operational environments become complex, single-agent systems are no longer adequate. The emergence of multi-agent systems has captured the attention of researchers, as they offer improved performance in challenging environments by enabling agents to communicate and collaborate effectively. Multi-agent systems have been applied in various fields, including traffic regulation, computer games, and robotics [1].

One of the most critical features of multi-agent systems is the ability to navigate and avoid obstacles. Efficient and safe navigation is crucial for the increasing adoption of mobile multi-agent systems in various fields, including autonomous driving cars and warehouse robots. Therefore, developing effective multi-agent navigation and obstacle avoidance techniques is significant in artificial intelligence. By incorporating machine learning and other advanced technologies, researchers can improve the performance of multi-agent systems and enable them to tackle increasingly complex challenges in various applications.

Enyu Zhao and Ning Zhou contributed equally.

✉ Chanjuan Liu
chanjuanliu@dlut.edu.cn

1 Department of Computer Science and Technology, Dalian University of Technology, Dalian, China

2 Department of Computer Science, University of Southern California Viterbi, Los Angeles, USA

3 School of Engineering Mathematics and Technology, University of Bristol, Bristol, UK

The problems

Obstacle avoidance is a critical aspect of autonomous navigation, and as such, research in the field of autonomous navigation has focused on exploring this topic. However, traditional route planning algorithms like A* [2] exhibit limitations, including inefficient adapting to dynamic environments and increased time complexity with adding agents. Fortunately, with advancements in deep learning, deep reinforcement learning has been introduced into multi-agent systems. Multi-agent reinforcement learning has emerged as a promising solution to obstacle avoidance and navigation challenges, offering improved performance and scalability.

In the early stages of multi-agent reinforcement learning, the initial approach was to have each agent run independently a separate reinforcement learning algorithm. For instance, Individual Q-Learning (IQL) [3] executes individual Q-learning [4] algorithms for each agent. Another example is the Individual Deep Deterministic Policy Gradient (IDDPG), which runs separate DDPG [5] algorithms for each agent. However, since each agent cannot access other agents' actions or observations, it cannot differentiate whether the environment change is due to the action of other agents or environment stochasticity. From the agent's perspective, this instability results in convergence failure [5].

Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [6] introduced the Centralized Training and Decentralized Execution (CTDE) approach, allowing agents to access other agents' actions and observations during the training phase, even the global observation of the entire environment. The CTDE approach can enable agents to understand the environment and ensure the stability of the training process, leading to higher performance. However, centralized training still has limitations, as it requires each agent to consider others' actions and observations during training, resulting in exponentially increasing joint action space as the number of agents grows. To address this issue, the Value Decomposition Network (VDN) [7] adds all agents' local critics' output to obtain the global action value, and the Monotonic Q-Value Decomposition Network (QMIX) [8] passes all agents' local action values to the mixer network to estimate the value of the joint action of all agents. As Q-learning, the primary reinforcement learning algorithm of QMIX, is only applicable for discrete action space, Factorized Multi-agent Actor-Critic (FACMAC) [9] was proposed as a supplement for continuous action space.

In the tasks of agent navigation and obstacle avoidance, the observation and action of each agent exhibit a continuous time-series pattern. Techniques such as Long Short-Term Memory (LSTM) [10] and Transformer [11] can be employed to improve performance and stability in the reinforcement learning process. However, their potential application in multi-agent navigation by reinforcement learning still needs to be explored and further investigated.

Our contribution

This article proposes to explore modified algorithms of MADDPG for solving the problems of obstacle avoidance and navigation in multi-agent systems. The algorithm will be trained and executed in a virtual environment, and the experiment will investigate whether incorporating the time-series pattern can enhance the algorithm's performance. Additionally, the study aims to explore how to scale the multi-agent algorithm to accommodate a more significant number of agents. Two algorithms, viz., MADDPG-LSTMactor and MADDPG-L, are obtained. The algorithms and the recent baseline algorithms, IDDPG, MADDPG, and FACMAC, were tested in several virtual environments. The results suggest that the MADDPG-LSTMactor algorithm performs better than the underlying MADDPG and other baseline algorithms. The MADDPG-L algorithm demonstrates the best performance as the number of agents increases.

The Transformer model and self-attention mechanism have recently become popular baselines when dealing with sequential data. These are crucial baselines that cannot be overlooked. Consequently, we conduct a comparative study of the LSTM-based approach with Transformer and self-attention models in the task of multi-agent navigation and obstacle avoidance. We substitute the LSTM layer in previous algorithms with the self-attention layer to see if the self-attention mechanism can outperform the relatively old LSTM model in four environments when the input sequence length is short. The results suggest no significant evidence that self-attention can outperform the LSTM. In another experiment, we compared the model with the LSTM layer, the self-attention layer, and the Transformer model in the obstacle prey-predator environment as the input sequence length increased. The results show that the Transformer model performs weakly when the sequence is short, but its performance improves as the size increases. However, its performance decreases as variance in the data increases due to long-length data. Although the Transformer model has a slower performance decrease than the LSTM model, it still cannot outperform the short-length LSTM model. On the other hand, the self-attention mechanism performs best with short-length data. Still, its performance deteriorates quickly as data length increases, and it is incapable of dealing with long-range data.

Related work

A work [12] closely related to our work on exploiting time-series features in the multi-agent setting took a different structure to our design due to the other focus of the task.

With its task being keeping a formation through moving in the environment, our work mainly focuses on the cooperation of agents in multi-agent path planning and obstacle avoidance. Regarding the performance differences between LSTM, self-attention, and Transformer, [13, 14] support the conclusion that LSTM still performs better than the other two in some instances. The first study compared a simple bidirectional LSTM and pre-trained BERT models on a task-specific small dataset. The second study applied both models to financial time-series prediction problems. Our work provides another comparative study of these models in multi-agent reinforcement learning scenarios.

Background

DDPG

In Deterministic Policy Gradient (DPG), the deterministic policy μ, parameterized by θ, will output a specific action u given a state s, $u = \mu_\theta(s)$, as opposed to a stochastic policy which gives the probability of different actions. This simplifies the calculation of policy gradient as integration is only required for the state rather than the state and action.

The policy's goal is to maximize a performance measure function available $J(\mu_\theta) = \mathbb{E}_{s\sim\rho}[r(s, \mu_\theta(s))]$ with ρ be the distribution of state s. And r is the reward function calculated by the action-state pair. The DPG algorithm learns the policy by improving according to its action-state estimation function Q through policy gradient ascent. The policy gradient is calculated as shown in Eq. (1).

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \mu_\theta(u|s)\nabla_u Q^\mu(s, u)|_{u=\mu_\theta(s)}\right] \tag{1}$$

Deep Deterministic Policy Gradient (DDPG) [15] is a deep learning-based algorithm that uses neural networks to represent both the policy network μ and evaluation function Q^μ. In the multi-agent setting, IDDPG extends DDPG by independently executing a DDPG algorithm on each agent.

MADDPG

The Multi-agent Deep Deterministic Policy Gradient (MADDPG) algorithm is a classic multi-agent reinforcement learning algorithm that extends the DDPG algorithm to the multi-agent domain by adopting the Centralized Training Distribution Execution (CTDE) training method. It uses offline learning to train agents that perform actions within a continuous action space. In the MADDPG algorithm, each agent is introduced with the basic structure of actor–critic, learning the deterministic policy network μ_a and the action evaluation function Q^{μ_a} that belongs exclusively to the agent a, respectively. Since each agent's policy and action evaluation function belongs only to the agent itself, it applies to totally-competitive, totally-cooperative, or mixed-setting environments.

In this article, agent$a$'s policy network μ_a is parameterized by θ_a, that is $\mu_a(\tau_a; \theta_a)$, τ_a is the observation of agent$a$. The joint policy of all agents is noted as μ, $\mu = \{\mu_a(\tau_a; \theta_a)\}_{a=1}^n$. Each agent's critic network is noted as Q^{μ_a}, with ϕ_a as its parameters. The critic network evaluates agent$a$'s action by the state s of the environment and the joint action u of all agents, which is $Q^{\mu_a}(s, u_1, u_2 \ldots u_n; \phi_a)$. The critic network of agents Q^{μ_a} is trained in a time differential fashion, aiming to minimize loss $\mathcal{L}(\phi_a)$, as in Eq. (2).

$$\mathcal{L}(\phi_a) = \mathbb{E}_\mathcal{D}\left[\left(y^a - Q^{\mu_a}(s, u_1, u_2 \ldots u_n; \phi_a)\right)^2\right] \tag{2}$$

where $y^a = r_a + \gamma Q^{\mu_a}\left(s', u'_1, u'_2 \ldots u'_n|_{u'_a=\mu_a(\tau_a;\theta_a^-)};\ \phi_a^-\right)$.

r_a denotes the reward obtained by agent a's interaction with the environment, ϕ_a^- is the parameter of the target critic network, $\left(u'_1, u'_2, \ldots u'_n\right)$ represent the target action of all agents chosen by their target policy network with parameter θ_a^-. The experience tuple stored in replay buffer $\mathcal{D}$ is $(s, s', u_1, u_2, \ldots u_n, r_1, r_2, \ldots r_n)$.

The policy gradient $J(\mu_a)$ of agent a is calculated in Eq. (3).

$$\nabla_{\theta_a} J(\mu_a) = \mathbb{E}_\mathcal{D}\left[\nabla_{\theta_a}\mu_a(\tau_a)\nabla_{u_a} Q^{\mu_a}(s, u_1, u_2, \ldots u_n)|_{u_a=\mu_a(\tau_a)}\right] \tag{3}$$

Importantly, it should be noted that during training, while the action taken by the currently trained agent a is computed using its policy network, the actions taken by other agents are retrieved from the replay buffer. MADDPG serves as the primitive algorithm for our method and a benchmark to compare against in the experiments.

VDN and QMIX

Value Decomposition Network (VDN) and Monotonic Q-Value Decomposition Network (QMIX) are multi-agent reinforcement learning algorithms utilizing centralized training distribution execution.

These algorithms aim to train a centralized critic that evaluates the global action-state value, denoted as Q^π_{tot}. Unlike the MADDPG algorithm, the environmental action evaluation function trained by VDN and QMIX has only one output that evaluates the global action-state value, and it is calculated in a decomposed fashion. Precisely, VDN sums each agent's environmental action evaluation functions, as shown in Eq. (4).

$$Q^{\pi}_{\text{tot}}(\tau, u; \phi) = \sum_{a=1}^{n} Q^{\pi_a}_a(\tau_a, u_a; \phi_a) \tag{4}$$

Here τ and u represent all agents' joint observation and action. $Q^{\pi_a}_a$ is the local critic network of agent a with π_a serving as its stochastic policy network. It is essential to acknowledge that $Q^{\pi_a}_a$ makes evaluation solely based on the agent's local observation record τ_a and its actions u_a, in contrast to MADDPG, where the evaluation functions take the global environment state and actions of all other agents into account.

The monotonic Q-value decomposition network (QMIX) uses a monotonic continuous mixing function to calculate the global action-state evaluation function Q^{π}_{tot}, as shown in Eq. (5):

$$\begin{aligned} &Q^{\pi}_{\text{tot}}(\tau, u, s; \phi, \psi) \\ &\quad = f_{\psi}\left(s, Q^{\pi_1}_1(\tau_1, u_1; \phi_1), \ldots Q^{\pi_n}_n(\tau_n, u_n; \phi_n)\right) \end{aligned} \tag{5}$$

Here f_{ψ} is the mixing function parameterized by ψ, s is the global state, $Q^{\pi_a}_a(\tau_a, u_a; \phi_a)$ is the local critic network of agent a, which is parameterized by ϕ_a with the agent's observation τ_a and action u_a as its input.

To ensure that the actions chosen by each agent when Q^{π}_{tot} is maximized are consistent with those selected when the agent's action-state evaluation function $Q^{\pi_a}_a$ is maximized, the mixing function must be monotonic. In practical applications, the mixing function in QMIX is often implemented as a neural network, which incorporates non-negative weights to ensure its monotonicity. The monotonic QMIX algorithm will still use the time differential algorithm for training with the target network and find the optimal solution by reducing the loss function $L(\phi, \psi)$, as shown in Eq. (6):

$$\mathcal{L}(\phi, \psi) = \mathbb{E}_{\mathcal{D}}\left[\left(y^{\text{tot}} - Q^{\pi}_{\text{tot}}(\tau, u, s; \phi, \psi)\right)^2\right] \tag{6}$$

where $y^{\text{tot}} = r + \gamma \max_{u'} Q^{\pi}_{\text{tot}}\left(\tau', u', s'; \phi^{-}, \psi^{-}\right)$

ϕ^{-} is the set of network parameters for the target critic network of each agent, and ψ^{-} is the target mixing network parameter.

FACMAC

The Factored Multi-Agent Centralized Policy Gradient (FACMAC) algorithm is designed to address multi-agent reinforcement learning problems with continuous action spaces. By contrast, VDN and QMIX are primarily utilized for tasks with discrete action spaces.

MADDPG adopts the centralized training of the action evaluation network Q^{μ}_{tot} using the CTDE method. This technique learns a centralized critic for each agent, which evaluates the overall environment and all agent actions. However, as the number of agents and the action space increase, optimizing the action evaluation network $Q^{\mu_a}_a$ becomes more challenging. This difficulty arises from the exponential increase in the input vector's dimension $(s, u_1, u_2 \ldots u_n)$ of the environmental action evaluation network $Q^{\mu_a}_a$ of each agent. Therefore, the Factorized Multi-Agent Centralized Policy Gradient Algorithm (FACMAC) proposes the use of value factorization to construct and train a centralized environmental action evaluation function Q^{μ}_{tot} that is easier to cope with the increase in the number of agents and the increase in the action space.

As the fundamental algorithm framework, FACMAC utilizes the Actor-Critic structure. In this structure, each agent a relies on its policy network μ_a to calculate a unique action u_a based on its local action observation history τ_a when selecting an action. Unlike the agents in VDN and QMIX, the agent only relies on the action value provided by the action evaluation function Q^{μ}_{tot} to choose an action. Therefore, there is no need to impose monotonicity restrictions on the mixing function when constructing the factored and centralized critic Q^{μ}_{tot}. This feature enables FACMAC to decompose the centralized environmental action evaluation function Q^{μ}_{tot} more freely without compromising its performance, making it more adept at handling complex environments.

The centralized action-value evaluation function of FACMAC is shown in Eq. (7):

$$\begin{aligned} &Q^{\mu}_{\text{tot}}(\tau, u, s; \phi, \psi) \\ &\quad = g_{\psi}\left(s, Q^{\mu_1}_1(\tau_1, u_1; \phi_1), \ldots Q^{\mu_n}_n(\tau_n, u_n; \phi_n)\right) \end{aligned} \tag{7}$$

ϕ and ϕ_a represent the parameters of the centralized action evaluation function Q^{μ}_{tot} and the local environmental action evaluation function $Q^{\mu_a}_a$ of each agent a, respectively. g_{ψ} is a nonlinear mixing function, which is not restricted by monotonicity, and the parameter is ψ. The learning method is the same as the QMIX, the target network is used to learn the centralized action evaluation function. The loss function is expressed in Eq. (8):

$$\mathcal{L}(\phi, \psi) = \mathbb{E}_{\mathcal{D}}\left[\left(y^{\text{tot}} - Q^{\mu}_{\text{tot}}(\tau, u, s; \phi, \psi)\right)^2\right] \tag{8}$$

where $y^{\text{tot}} = r + \gamma Q^{\mu}_{\text{tot}}\left(\tau', \mu\left(\tau'; \theta^{-}\right), s'; \phi^{-}, \psi^{-}\right)$

In the calculation of the policy gradient, FACMAC proposes a calculation method that computes the centralized policy gradient as in Eq. (9):

$$\nabla_{\theta} J(\mu) = \mathbb{E}_{\mathcal{D}}\left[\nabla_{\theta}\mu \nabla_{\mu} Q^{\mu}_{tot}(\tau, \mu_1(\tau_1), \mu_2(\tau_2) \ldots \mu_n(\tau_n), s)\right] \tag{9}$$

In contrast to the gradient calculation method of MADDPG presented in Eq. (3), FACMAC calculates all agents' actions u_a using the policy network μ_a of each agent a based on its local action observation history τ_a. In contrast, MADDPG only calculates the action u_a of the currently trained agent a, while sampling the actions of other agents from the replay buffer.

Transformer and self-attention mechanism

The Transformer model signals a departure from traditional sequential processing models, such as recurrent neural networks (RNNs), by relying solely on self-attention mechanisms. It features its attention mechanism, enabling it to simultaneously consider all input sequence positions, fostering more effective learning for contextual relationships.

The Transformer architecture consists of an encoder–decoder structure, but in our case, it can also be applied in an encoder-only configuration for feature extraction.

The core components of the Transformer are the multi-head self-attention mechanisms and feedforward neural networks. Given a sequence of input $X = \{x_1, x_2, \ldots, x_n\}$, and the output $Y = \{y_1, y_2, \ldots, y_n\}$, the computation of a single head is in Eq. (10):

$$Y_i = \sum_{j=1}^{n} \mathrm{softmax}\left(\frac{Q_j \cdot K_j^T}{\sqrt{d_k}}\right) V_j \tag{10}$$

where $Q_j = \mathrm{XW_q}$, $K_j = \mathrm{XW_k}$ and $V_j = \mathrm{XW_V}$ are linear transformations applied on the input sequence, and W_q, W_k, W_v are the weight matrices for the query, key, and value. Softmax is applied row-wise. The output and input of the self-attention layer are connected by residual blocks and a normalization layer.

The output of heads $Y_1, Y_2 \ldots Y_n$ are then concatenated and went through another linear projection $Y = W[Y_1, Y_2 \ldots Y_n]$. The output of the self-attention module is then passed into a feedforward neural network with two layers and a ReLU activation function. The output of the neural network and the input are also connected in a residual fashion with layer norm. These complete the computation inside a single encode or decode block, and by repeating this process for N times, we get the final output of the model.

The design of algorithms

MADDPG-LSTMactor

In multi-agent reinforcement learning tasks, agents interact with the environment using their policies and gain rewards from the influence of their actions on the environment. In this learning process, the actions and observations of each agent and the states of the environment all exhibit time-series characteristics. Leveraging such a time-series feature has shown a promising performance improvement in reinforcement learning since the agent can now consider historical data when making decisions akin to humans. With our task aims at providing a collision-free path for multiple agents in a partially observable environment, the agents rely on the observation sequence and action sequence to understand their current position and whether the change of the relative positions of obstacles (including other agents) to them is only caused by their action solely. Since the agents move continuously in our setting, the time-series feature of historical observation and action is more prominent. With this insight, we propose incorporating mechanisms like LSTM and Transformer into the existing multi-agent reinforcement learning algorithms to exploit the time-series feature hidden behind the agents' observation and action history. Our experiment chose the MADDPG, which has shown great success in multi-agent learning tasks, as our base policy to modify. A work closely related to ours is proposed by Yan et al. [12], which utilized the Gate Recurrent Unit (GRU) [16] layer in both the actor network and critic network in Multi-agent Policy Proximal Policy Optimization (MAPPO) to perform a multi-agent formation control task.

As the MADDPG algorithm adopts the centralized training and decentralized execution (CTDE) training method, each agent's critic network estimates the action u_a made by the actor network μ_a based on the environment's state s to guide the actor in the training phase. In contrast, the actor network generates actions using its observation history as input in both the training and execution phases. Therefore, compared to the centralized critic network, the actor network μ_a deals with time-series data more frequently and directly. Thus, we introduce layers that are specialized for series data into the actor network, allowing an agent a to determine its action u_a at timestep t based on the sequence of its local observations $\boldsymbol{\tau_a} \equiv \left\{\tau_a^{t-i}\right\}_{i=0}^{l}$ rather than solely relying on the observation τ_a^t at timestep t. We set the sequence length as a hyper-parameter $\mathrm{seq}-\mathrm{length} = l$. The action-taking process of agent a can be defined as Eq. (11).

$$u_a = \mu_a\left(\{\tau_a^{t-i}\}_{i=0}^{l};\ \theta_a\right) \tag{11}$$

To handle time-series data, the pipeline of the MADDPG algorithm requires modification. We brought forward a sequence buffer $\mathcal{B}$ to support the manipulation and storage of the interaction sequences. Specifically, a sequence buffer $\mathcal{B}$ is created and maintained during both the training and execution phases, which is a queue with the length of $\mathrm{seq}-\mathrm{length}$. This buffer stores experience tuples, including the environment state s, all agents' observation τ, all agents'

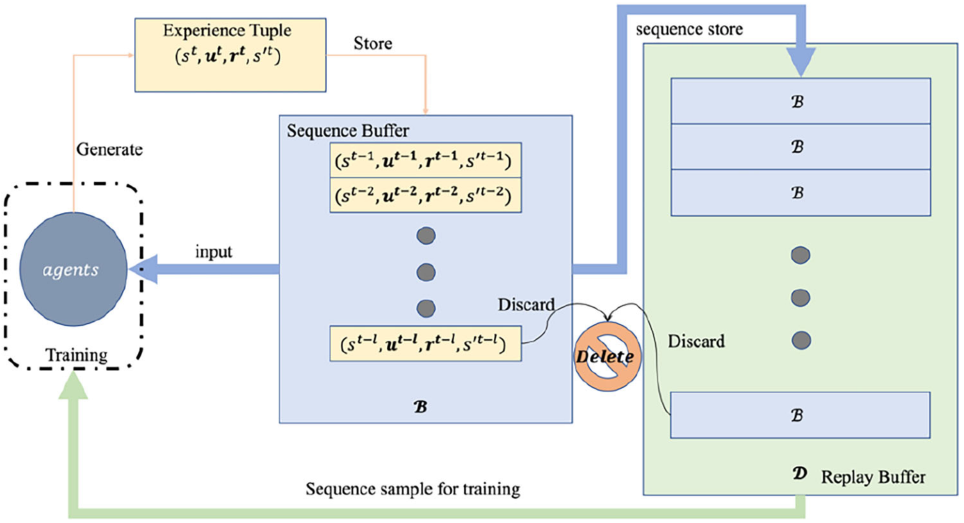

Experience Tuple
(s^t, u^t, r^t, s'^t)
Store
sequence store
Generate
Sequence Buffer
$(s^{t-1}, u^{t-1}, r^{t-1}, s'^{t-1})$
$(s^{t-2}, u^{t-2}, r^{t-2}, s'^{t-2})$
agents
input
Discard
$(s^{t-l}, u^{t-l}, r^{t-l}, s'^{t-l})$
Delete
Discard
Training
$\mathcal{B}$
$\mathcal{D}$ Replay Buffer
Sequence sample for training

Fig. 1 The illustration of MADDPG-LSTMactor. Here, "agents" represent the agents in the multi-agent system, which takes a whole sequence in the sequence buffer as input and interacts with the environment, generating a single-step experience tuple, first stored in the sequence buffer. The sequence buffer, if it is full, will delete the oldest experience tuple and be stored inside the replay buffer. During training, sequences will be sampled

action u, and reward r. At each timestep, the actor network takes action u_a based on the observation sequence stored in sequence buffer $\mathcal{B}$ and gains reward r_a. If the sequence buffer is full, the oldest experience tuple will be removed, and the most recent one will be stored at the end of the queue. In the meantime, the whole sequence buffer will be held as an element in the replay buffer $\mathcal{D}$. The sequence buffer will be emptied when the environment is reset. During training, instead of sampling experience tuples, the previously stored sequence buffer, which is a continuous sequence of experience tuples, will be sampled. A visual illustration of this pipeline is shown in Fig. 1.

In our experiment, Multi-Agent Deep Deterministic Policy Gradient with LSTM actor (MADDPG-LSTMactor) demonstrates a significant performance improvement over the ordinary MADDPG algorithm. Notably, leveraging Transformer instead of LSTM should boost the final performance, which we discuss later in the experiment. The MADDPG-LSTMactor algorithm maintains the same critic network structure and optimization procedure as the MADDPG algorithm. Therefore, the methods used to optimize the critic and actor networks are identical in both algorithms.

MADDPG-L

As a common multi-agent reinforcement learning (MARL) algorithm employing the CTDE method, the MADDPG algorithm requires agents to incorporate each other's observations and actions in the critic network during the training phase to stabilize the training process and enhance the algorithm's performance. However, the aforementioned demand for additional information can become problematic when the number of agents in the multi-agent system is large, resulting in an explosion of the action space. The Factorized Actor-Critic for Multi-Agent Reinforcement Learning (FACMAC) algorithm offers a potential solution by learning a single centralized but factorized critic, which decomposes the joint action-value function Q_{tot} into per-agent utility Q_a that is combined via a nonlinear function. Each factored action-value utility, i.e., each agent's local critic network estimates the value of its action, which can be written as $Q_a(\tau_a, u_a; \phi_a)$. τ_a is the local observation of agent a, u_a is its action, and ϕ_a stands for the parameters of the critic network. The mixing function, g_ψ, then aggregates the outputs of all agents' critic networks and the environment state s to produce the final joint action-value estimation for all agents. As a result, the agent's local critic network only needs to consider its observations and actions, regardless of the number of agents in the system. The mixing function ensures that all agents can obtain information about other agents and the environment during the training phase.

However, despite the advantages of algorithms that factor in a centralized critic, they are not immune to limitations. In a baseline test of the Multi-agent Particle Environment (MPE) [17], the QMIX tended to become stuck in the initial state. In the experiments described in this article, the FACMAC algorithm exhibited a similar issue.

The cold start problem and slow convergence in algorithms that factor the centralized critic may be due to the large number of networks that need to be optimized during training. The FACMAC algorithm, which creates the centralized critic through a mixing function that combines all local critics, requires optimization of all local critics during each agent's training. In contrast, MADDPG only requires optimization of an agent's critic network during training. This difference may make it harder for FACMAC to progress and exacerbate the cold start problem. The calculation of the centralized critic network in the experiment supports this suggestion. During an agent's training, the centralized critic requires other agents' critic results, and it is possible

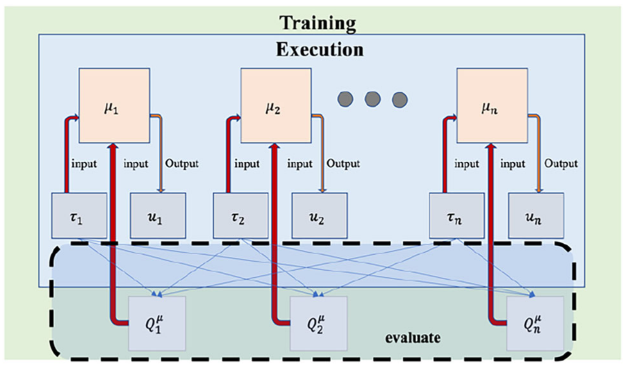


Fig. 2 Illustration of MADDPG-L. We modified MADDPG's structure by inputting each agent's critic network, the joint observation, and its action rather than the joint action of all agents

to use either the other agents' critic networks or the trained agent's critic network to simulate the action value of their actions. Using only the trained agent's critic network to simulate other agents' critic networks reduces the number of networks that need to be optimized and may make training easier. This method improved the experiment's performance, but the progress still needed improvement.

In this article, we present a new algorithm called MADDPG-L capable of scaling to a larger number of agents. The actor network of the proposed algorithm is similar to that of MADDPG, where the policy of agent a is denoted as $\mu_a(\tau_a;\ \theta_a)$, and the action of agent a is $u_a = \mu_a(\tau_a, \theta_a)$. θ_a is the parameter of μ_a. The key difference lies in the critic network of each agent, denoted by $Q_a^\mu(s, u_a;\ \phi_a)$, which considers only the action of the current agent instead of all agents' actions. Here ϕ_a is the parameter of Q_a^μ. Meanwhile, the environmental state s provides the necessary global information to the agent. This change mitigates the problem of action space explosion by requiring each agent to compute only its own action, regardless of the number of agents in the multi-agent system. The architecture is illustrated in Fig. 2. The critic network is trained using temporal difference learning, and its loss function is given by Eq. (12).

$$\mathcal{L}(\phi_a) = \mathbb{E}_\mathcal{D}\Big[\big(y^a - Q^{\mu_a}(s, u_a;\ \phi_a)\big)^2\Big] \tag{12}$$

where $y^a = r_a + \gamma Q^{\mu_a}\Big(s', u'_a|_{u'_a=\mu_a(\tau_a;\theta_a^-)};\ \phi_a^-\Big)$. r_a is the reward of agent$a$'s interaction with the environment, ϕ_a^- indicates the parameter of target network, u'_a is the action implemented by the target actor network with its parameter notified as θ_a^-. The experience tuples stored in the replay buffer $\mathcal{D}$ has the structure as $(s, s', u_1, u_2, \ldots u_n, r_1, r_2, \ldots r_n)$.

The policy gradient of agent a's actor network $J(\mu_a)$ is shown in Eq. (13):

Table 1 Experimental environment parameters

Term	Version
CPU	Intel i9 9900 K
Memory	64 GB
GPU	NVIDIA GTX1080 (VRAM 8 GB)
Operation system	Ubuntu 18.04 LTS
GPU driver version	495.46
Python version	3.8.13
Pytorch version	1.11.0
CUDA version	11.5

$$\nabla_{\theta_a} J(\mu_a) = \mathbb{E}_\mathcal{D}\big[\nabla_{\theta_a}\mu_a(\tau_a)\nabla_{u_a} Q^{\mu_a}(s, u_a)|_{u_a=\mu_a(\tau_a)}\big] \tag{13}$$

Experiment

Simulation environment

The experiment environment is based on the MPE environment. Other environmental parameters are listed in Table 1.

The experiment environments and result

In this study, several experiment environments were used to evaluate the performance of different multi-agent reinforcement learning algorithms. These environments include obstacle predator–prey, spread, tunnel, and simple tunnel. The obstacle predator–prey environment was chosen as the initial test environment to evaluate the algorithms' abilities in obstacle avoidance and dynamic target-pursuing tasks. The spread environment was designed to test the algorithms' abilities to avoid obstacles and reach designated destinations. This environment also had multiple versions with varying numbers of agents to assess the scalability of the algorithms. The tunnel environment was created to test the algorithms' navigation abilities through environments where obstacles form a distinct contour. Furthermore, the simple tunnel environment was subsequently derived from the original tunnel environment to provide a more realistic testing ground for these algorithms. Typically, all the multi-agent reinforcement learning algorithms proposed and mentioned in this article, IDDPG, MADDPG, MADDPG-LSTMactor, MADDPG-L, and FACMAC, were all tested in each virtual environment. However, the FACMAC algorithm was excluded from testing in the simple tunnel environment due to its requirement for equal rewards for all agents. The design of an appropriate penalty function for the FACMAC algorithm in this environment is challenging, given the environment's need for each agent to prioritize navigation toward its designated destination.

In addition to comparing with different baselines, we are also interested in whether the latest Transformer model and self-attention mechanism can perform better than the LSTM-based model. To test this assumption, we conducted experiments by replacing the LSTM layer in the MADDPG-LSTMactor model with the self-attention layer. The new model is called the MADDPG-ATTENTIONactor model, and we used it to see if the self-attention mechanism could outperform our proposed algorithm in the obstacle predator–prey and spread environments.

According to our understanding, the LSTM model suffers from gradient vanishing and explosion as the series grows longer. On the other hand, the Transformer model is better at handling long-range data and only remembers the most critical bits of the series. However, for our task of multiagent reinforcement learning, the time-series data within the replay buffer is relevant within short distances and is full of noise within long distances. This raises the question of whether the Transformer-based model MADDPG-TRANSactor can leverage the information better than the MADDPG-LSTMactor as input series length increases.

To address this question, we introduced a multi-head Transformer-based MADDPG algorithm. We conducted experiments in the obstacle predator–prey environment with input series lengths 5, 20, and 50 by MADDPG-TRANSactor, MADDPG-ATTENTIONactor, and MADDPG-LSTMactor. Our goal is to determine the best backbone network with a tradeoff between the lack of information and the high variance of long data series for the specific MADDPG multiagent reinforcement learning problem.

Hyper-parameters used in the experiments were listed in Table 2, with some remaining constant across all environments and algorithms, while others varied and were shown in the appendix.

Table 2 Hyperparameters of the experiment

Hyperparameters	Value	Meaning
Max-episode-len	100	The length of each episode
Time-steps	2,000,000	Total interaction steps
Num-adversaries	1 (default)	The learning rate of policy network
Lr-critic	0.01 (default)	The learning rate of the critic network
Epsilon	0.1	Parameter for ϵ-greedy action choosing a policy
Noise-rate	0.1	The rate of noise added to action during the training phase
Gamma	0.95	Time elapse coefficient
Batch-size	256 (default)	Batch-size for experience tuples sampled for training
Seq-length	5 (default)	The length of continuous steps for LSTM training
Num-heads	4 (default)	The number of heads for multi-heads Transformer
Dim-feedforward	256	The hidden layers of the feedforward network in Transformer
Dropout rate	0.1 (default)	The possibility for dropout in Transformer

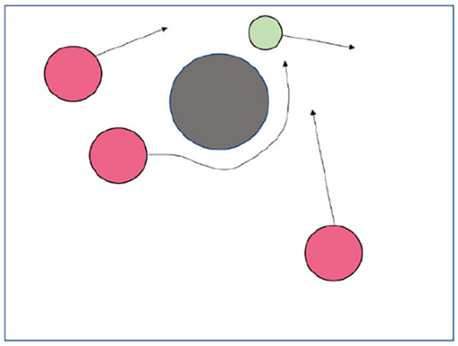


Fig. 3 Illustration of obstacle predator–prey environment

Obstacle predator–prey

The obstacle predator–prey environment was used to evaluate the performance of a team of agents trained to pursue an adversarial agent at a higher speed while avoiding crashing into obstacles and each other. The environment is illustrated in Fig. 3.

The experiment results are presented in Fig. 4, indicating that algorithms using the CTDE training paradigm generally outperform IDDPG, which adopts the decentralized training method. MADDPG-LSTMactor, with the aid of LSTM, exhibited better performance than MADDPG in this environment. Additionally, MADDPG-L, which only provides agents' critics with the agent's action and the environment state, demonstrated superior performance than MADDPG in the final stage. It is worth mentioning that FACMAC is an exception, as its performance issue is elaborated in 3.2.

Additional experiments were conducted to investigate FACMAC's poor performance in the obstacle predator–prey environment. The first experiment involved testing two methods of obtaining the final centralized critic. The calculation

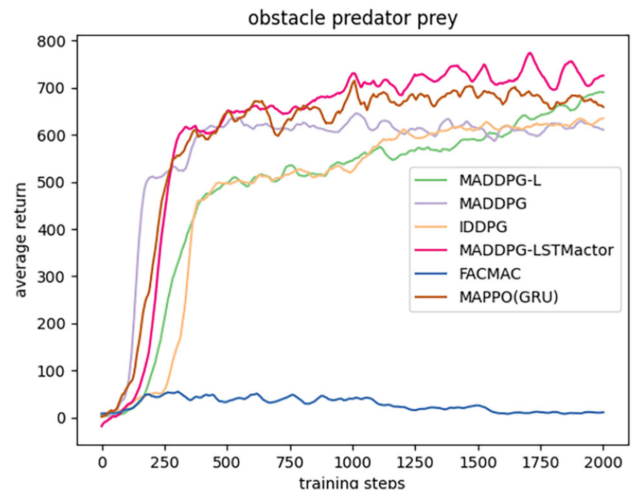


Fig. 4 Algorithms' performance in obstacle predator–prey environment

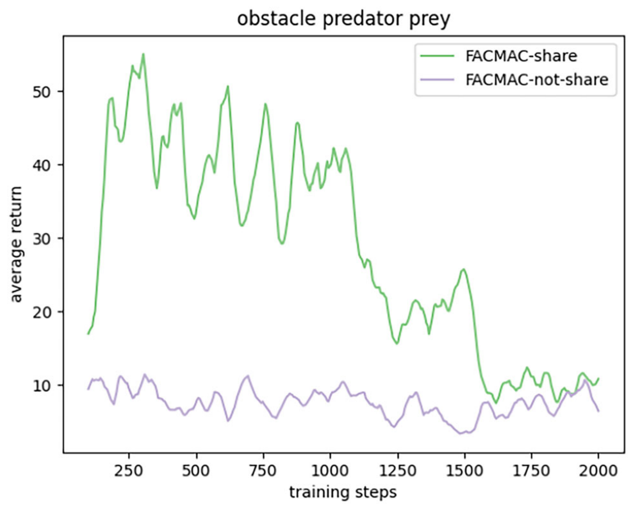


Fig. 5 Performance of FACMAC with different parameter sharing policy

of the centralized critic requires the results of other agents' critic networks, which can be estimated directly using their actions or simulated using the critic network of the currently trained agent, given that all agents share the same structure for their critic networks in the experiment. The results of the two calculation methods for the centralized critic in FACMAC are presented in Fig. 5.

The results in Fig. 5 indicate that simulating other agents' critic networks using an agent's critic network results in better performance than directly using other agents' critic networks. However, the performance still needs to catch up to other algorithms. The reason for this suboptimal performance could be attributed to the mixing network, which is considered a part of the critic network in FACMAC's design.

Updating the critic network's parameters also updates the mixing net-work's parameters, leading to uncertainty in

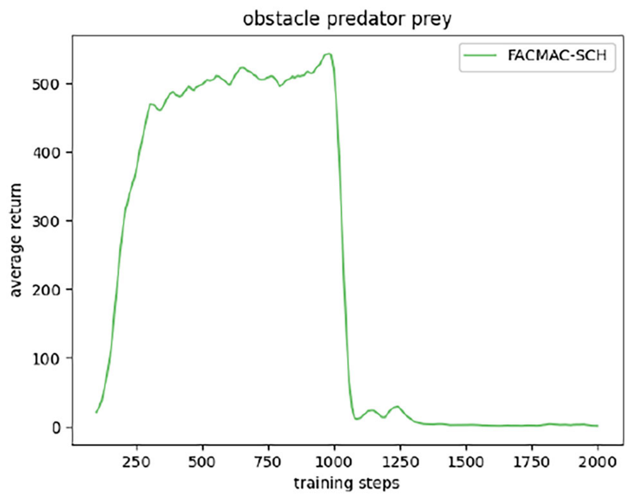


Fig. 6 FACMAC's performance by optimizing the critic networks and policy networks at the first stage, then training the mixer network

the gradient's direction. An additional experiment was conducted to validate this hypothesis.

In our second additional experiment, we conducted a study where we froze the parameters of the mixing network and solely optimized the local critics of agents in the initial stage of training, eliminating the centralized critic. In the later stage, we resumed updating the parameters of the mixing network and reverted to the regular training process of FACMAC. We used different methods to calculate the policy gradient in each stage. In the first stage, we used Eq. (14), and in the later stage, we used Eq. (15).

$$\nabla_\theta J(\mu_a) = \mathbb{E}_{\mathcal{D}}\left[\nabla_\theta \mu_a \nabla_{\mu_a} Q_a^\mu(\tau_a, \mu_a(\tau_a); \phi_a)\right] \tag{14}$$

$$\nabla_\theta J(\mu) = \mathbb{E}_{\mathcal{D}}\left[\nabla_\theta \mu \nabla_\mu Q_{tot}^\mu(\tau, \mu_1(\tau_1), \mu_2(\tau_2) \ldots \mu_n(\tau_n), s)\right] \tag{15}$$

The aim of this experiment was to reduce the complexity of centralized training by pre-training the agents' local critic networks while ensuring sufficient positive data in the replay buffer. The results of this experiment are presented in Fig. 6.

We set one million timesteps to be the end of the first stage. The algorithm's performance simply plunged to the level of the random policy after reaching the watershed. We altered each agent's local critic network from $Q_a^\mu(\tau_a, u_a; \theta_a)$ to $Q_a^\mu(s, u_a; \theta_a)$ and get rid of the mixing network. By doing this, we created MADDPG-L. The input of the algorithm's critic network doesn't have the problem of the explosion of action space as it only requires the agent's own action and still provides global information.

Spread

In the spread environment, target locations will be randomly generated after each environment reset, and a team of agents

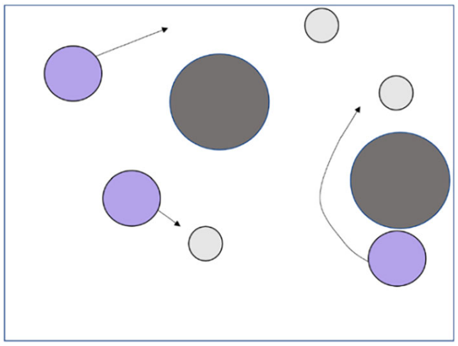


Fig. 7 The illustration of the spread environment

with the same number of target locations will be trained to occupy all target locations. Obstacles with random locations will also be generated in the space. Agents learn to occupy all target locations while avoiding collision with the obstacle or each other. The illustration of this environment is in Fig. 7.

Due to the different training goals, the reward function in this environment is changed from the one in the obstacle predator–prey. The reward function is set to be the negative distance sum of all target locations to their nearest agent. In other words, the reward will increase with each agent getting closer to its target. We also give agents negative rewards for having collisions with other agents or obstacles.

To examine algorithms' ability to scale to a larger number of agents, the spread environment will have different versions with different numbers of agents: Spread-3a with three agents, Spread-6a with six agents, and Spread-9a with nine agents. The performances of algorithms in these three environments are shown in Fig. 8.

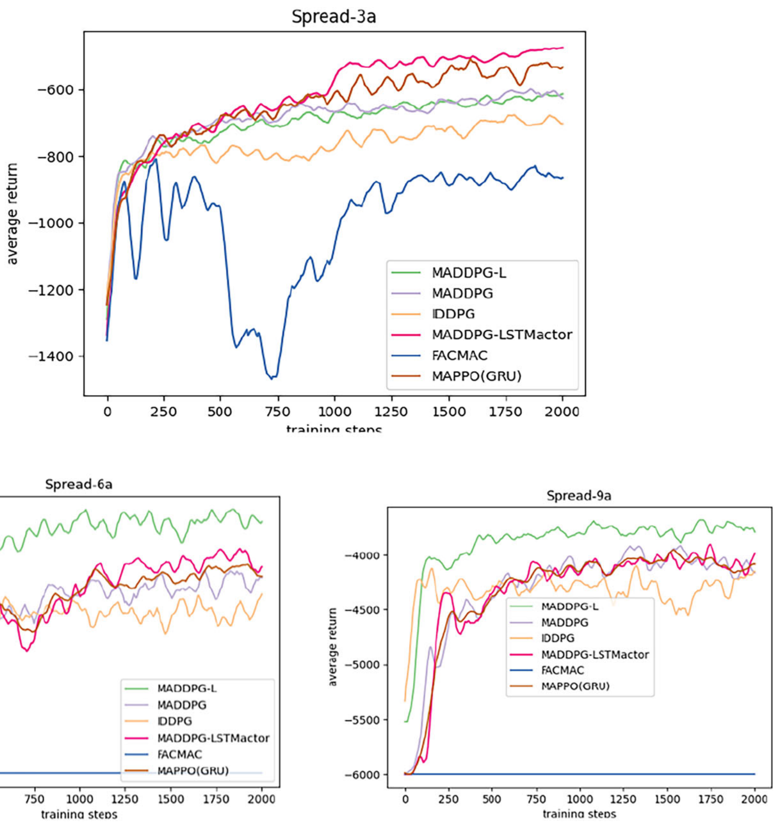
Based on the experimental results obtained from three different environments, it can be observed that MADDPG-LSTMactor outperforms MADDPG when the number of agents is relatively low. However, as the number of agents increases, MADDPG-LSTMactor does not exhibit any advantage over MADDPG. This could be attributed to the increased complexity introduced by LSTM.

Although MADDPG-L does not exhibit any significant improvement over MADDPG for a small number of agents, it demonstrates a significantly better performance as the number of agents increases. Therefore, it can be concluded that MADDPG-L scales better to more agents than MADDPG in the tested environment.

Tunnel and simple tunnel

The tunnel environment is characterized by two substantial obstacles forming a narrow passageway, necessitating the agents to devise a path to navigate through. At one end of the tunnel are N target locations with fixed positions. Upon

Fig. 8 Performance in spread environments with different amounts of agents



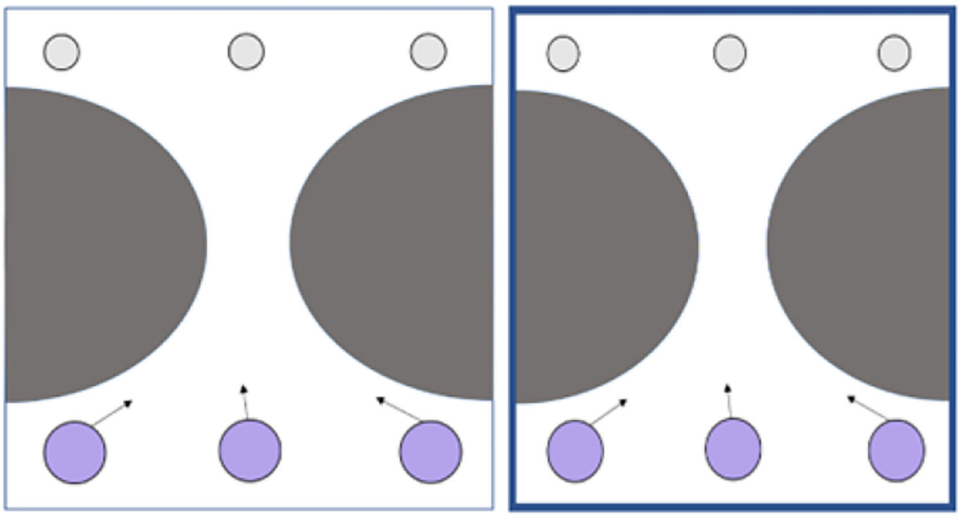


Fig. 9 The illustration of the tunnel environment (left) and simple tunnel environment (right)

resetting the environment, N agents appear at predetermined locations on the tunnel's opposite end. The agents' primary objective is to traverse the tunnel and occupy each target location while avoiding contact with each other or the obstacles. The environment's visual representation is presented in Fig. 9 (left).

The reward function of the tunnel environment, similar to the spread environment, is defined as the negative sum of each target location to its closest agent. Furthermore, agents are penalized for colliding with other agents or obstacles. To simplify the environment and be more realistic, we introduce some modifications to the tunnel environment. Specifically, we add walls around the obstacles to restrict the agents' mobility range. Additionally, the same number of agents and target locations will appear at both ends of the tunnel. However, each agent is now assigned a unique target location, and they must avoid collisions while moving toward their respective targets. The revised environment is called the simple tunnel environment, with its visual representation depicted in Fig. 9.

Due to the different goals of agents in the simple tunnel environment, the reward function is modified accordingly. Specifically, agents' rewards are no longer collaborative and shared, and instead, each agent's reward is determined by its distance to its designated target location, with penalties for collisions. As each agent's reward function is distinct, FACMAC is not applicable in this environment.

The experimental results in Figs. 10 and 11 demonstrate that the MADDPG-LSTMactor maintains its superiority over MADDPG when the number of agents is small. Moreover, in the simple tunnel environment, which is closer to reality, the MADDPG-LSTMactor exhibits better performance and faster learning speed. While MADDPG-L performs similarly to MADDPG, it also has a faster convergence rate.

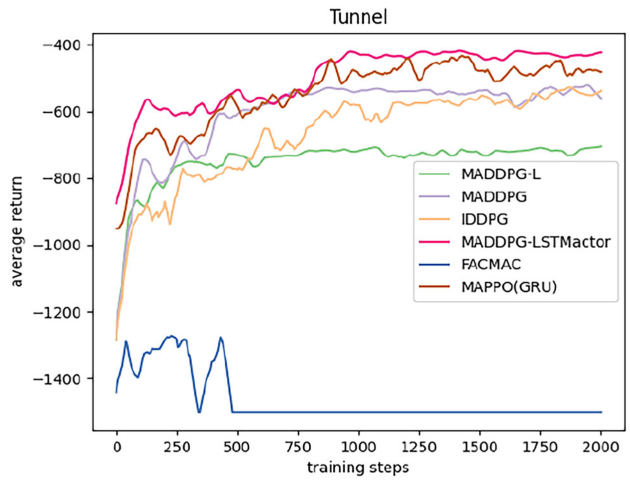


Fig. 10 Algorithms' performance in a tunnel environment

After experiments were conducted in the simple tunnel environment, the number of agents was increased from three to six, and the new environment was designated as simple tunnel 6a. The results of the algorithmic tests in this environment, as shown in Fig. 11, indicated that MADDPG-L outperformed MADDPG with an increase in the number of agents. At the same time, MADDPG-LSTMactor lost its performance advantage over MADDPG.

Comparing with transformer and self-attention

In recent years, we witnessed a growing presence of the Transformer model in both natural language processing and computer vision domains. The Transformer model is known for its capability of handling long-range dependence and parallel computation. Its success is mainly due to the self-attention mechanism, which encodes the representation by relating different parts in the same sequence. Along with the

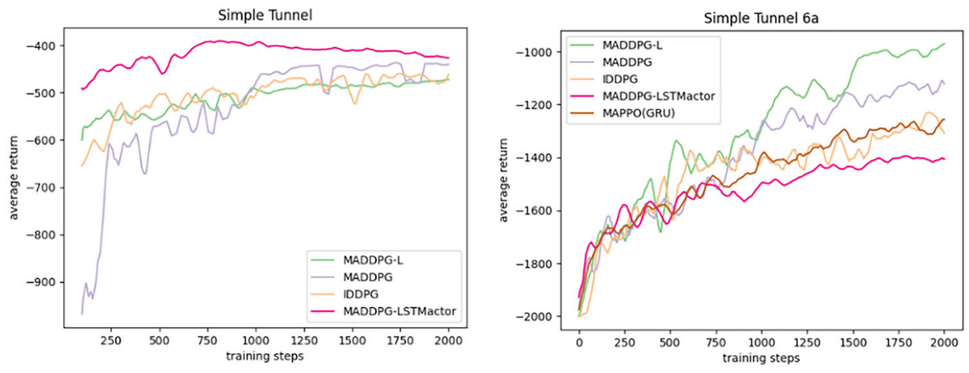


Fig. 11 Algorithms' performance in simple tunnel environments with different numbers of agents

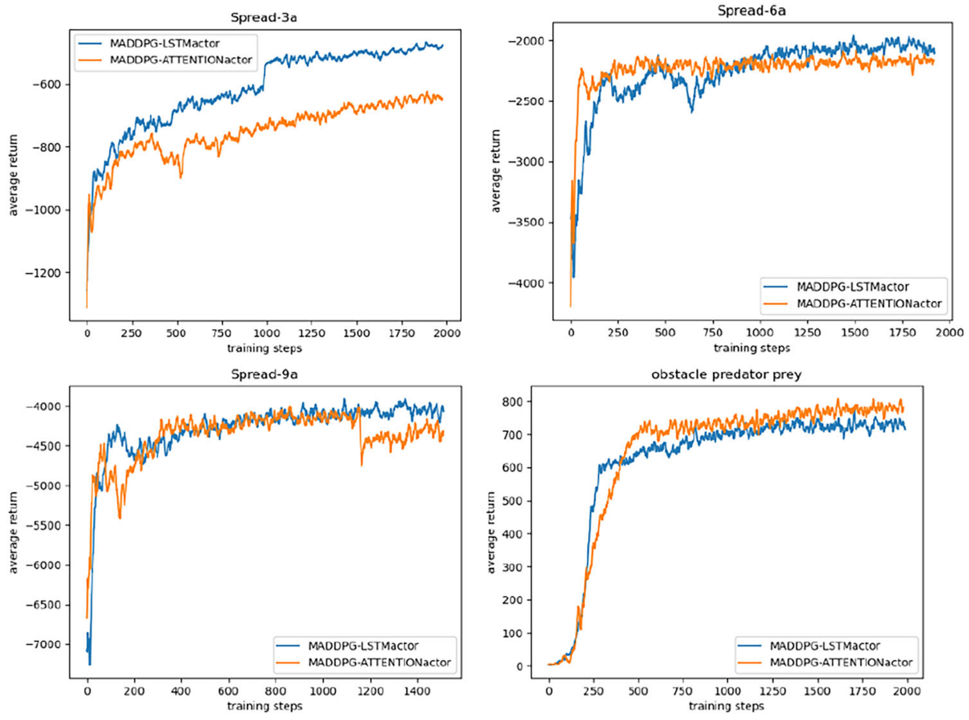


Fig. 12 Algorithms' performances in spread and obstacle predator environment

LSTM model, the self-attention mechanism and the Transformer model also fit into our primary instinct of leveraging knowledge inside the time-series information. Regarding computational cost, the Transformer model ranks highest, followed by the LSTM and the attention mechanism.

In this section, we look at the performance of the attention mechanism and the LSTM in the obstacle predator–prey and spread environment, with an input sequence length of 5. The results, depicted in Fig. 12, show that the MADDPG-ATENTIONactor outperforms the MADDPG-LSTMactor in the obstacle predator–prey environment. However, it exhibits weaker performance and training stability in the spread environment. This suggests that both models are capable of extracting time-series information when the length of the input is relatively short.

The input sequence length is an interesting factor to consider when using the LSTM and Transformer models. The LSTM model has difficulty handling long-range data due to gradient vanishing and explosion, while the Transformer model performs better in dealing with long-range dependencies. In the previous experiments, an input sequence length of 5, which is relatively short, was used. To investigate the ability to handle long-range data, we conducted experiments using MADDPG-TRANSactor, MADDPG-ATTENTIONactor, and MADDPG-LSTMactor with input series lengths 5, 20, and 50. The results are shown in Fig. 13.

Figure 13 indicates that the transformer model performs less with shorter sequences but demonstrates increased effectiveness as the sequence length extends. However, performance diminishes as the length of the input data increases due to higher variance. Although the Transformer model experiences a slower decline in performance compared to the LSTM model, it fails to surpass the performance of the

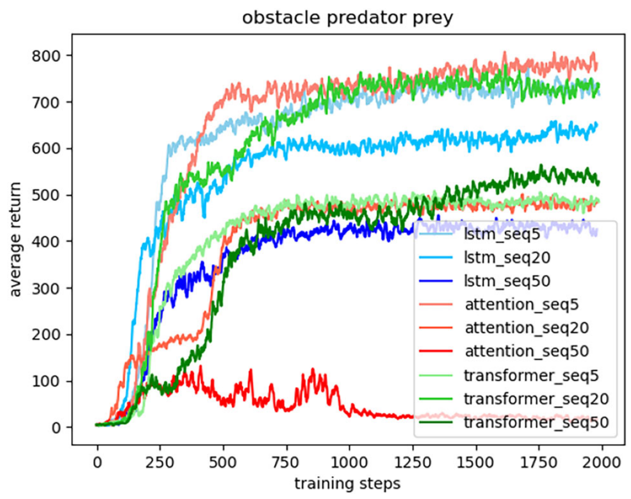


Fig. 13 Algorithms' performance in obstacle predator–prey environment

LSTM model with short sequence inputs. Notably, the self-attention mechanism is good at handling short-length data but experiences a rapid decline in performance as the data length increases, indicating it is less adept at handling long-range data.

Therefore, we conclude that the LSTM model strikes a better balance between the lack of information and high variance than the other two models. Unlike the Transformer-based model, which is ineffective when the input length is short and is restricted by the high variance of the long input data, the LSTM model avoids substantial performance deterioration faced by the self-attention-based model as the input length increases.

However, this does not undermine the efficacy of the Transformer and self-attention models. Notably, the Transformer-based model experiences a slower decline in performance as the input gets longer, and the self-attention mechanism manages to outperform the LSTM-based model with short input length and lower computational cost. In conclusion, if the data features high variance in the long-range and relevance in the short range, we should try the LSTM network before jumping to the latest Transformer backbone.

Conclusion

This work explores the modified algorithms of MADDPG, namely MADDPG-LSTMactor and MADDPG-L, to improve its performance in multi-agent navigation and obstacle avoidance. This study also conducts a comprarative study of the LSTM-based model with Transformer-based model and self-attention-based model.

The experimental results indicate that MADDPG-LSTMactor and MADDPG-L outperform the basedline algorithms IDDPG, MADDPG, and FACMAC with respect to their ability to avoid obstacles and handle an increase in the number of agents in various virtual environments. It was observed that transformer and self-attention do not consistently outperform LSTM, and the LSTM-based model exhibits a favorable tradeoff across vaing sequence lengths.

The algorithms proposed in this work possess specific capabilities in virtual environments and can serve as basic algorithms for future multi-agent obstacle avoidance navigation tasks. It should be noted that realistic obstacle avoidance algorithms are not limited to using reinforcement learning as the sole solution to the problem. There are also promising solutions for multi-agent navigation based on other frameworks, such as meta-learning [18]. Combining geolocation, computer vision, etc., may achieve better performance. Multi-agent decision-making in more challenging scenarios, e.g., with limited computation resouces [19], limited communication power [20], is also well worth exploring.

Funding This article is funded by National Natural Science Foundation of China, 62172072, Chanjuan Liu.

Declarations

Conflict of interest The authors would like to declare that the work was the authors' original research. We wish to confirm that the work described has not been published previously, that it is not under consideration for publication elsewhere, that its publication is approved by all authors and tacitly or explicitly by the responsible authorities where the work was carried out, and that, if accepted, it will not be published elsewhere in the same form, in English or in any other language, including electronically without the written consent of the copyright holder.

Appendix

See Tables 3, 4, 5, 6 and 7 here.

Table 3 Hyperparameters in obstacle predator–prey environment

Hyperparameter	Value
Num-adversaries	1
Lr-actor	0.001
Lr-critic	0.01
Batch-size	256
seq-length	3

Table 4 Hyperparameters in Spread-3a environment

Hyperparameter	Value
Num-adversaries	0
Lr-actor	0.001
Lr-critic	0.01
Batch-size	128
seq-length	5

Table 5 Hyperparameters in Spread-6a and Spread-9a environment

Hyperparameter	Value
Num-adversaries	0
Lr-actor	0.001
Lr-critic	0.01
Batch-size	32
seq-length	3

Table 6 Hyperparameters in tunnel and simple tunnel environment

Hyperparameter	Value
Num-adversaries	0
Lr-actor	0.001
Lr-critic	0.01
Batch-size	128
seq-length	5

Table 7 Hyperparameters in simple Tunnel-6a environment

Hyperparameter	Value
Num-adversaries	0
Lr-actor	0.001
Lr-critic	0.001
Batch-size	32
seq-length	3

References

1. Zhong J, Wang T, Cheng L (2022) Collision-free path planning for welding manipulator via hybrid algorithm of deep reinforcement learning and inverse kinematics. Complex Intell Syst 8:1899–1912
2. Dechter R, Pearl J (1985) Generalized best-first search strategies and the optimality of A. J ACM 32(3):505–536
3. Tan M (1993) Multi-agent reinforcement learning: independent vs. cooperative agents. Machine Learning Proceedings, pp 330–337
4. Watkins CJCH, Dayan P (1992) Q-learning. Mach Learn 8(3):279–292
5. de Witt CS, Gupta T, Makoviichuk D, et al. (2020) Is independent learning all you need in the starcraft multi-agent challenge? arXiv preprint arXiv:2011.09533. Accessed 15 Dec 2023.
6. Lowe R, Wu Y, Tamar A, et al. (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. International Conference on Neural Information Processing Systems, Long Beach, pp 6382–6393
7. Sunehag P, Lever G, Gruslys A, et al. (2017) Value-decomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296. Accessed 15 Dec 2023.
8. Rashid T, Samvelyan M, Schroeder C, et al. (2018) Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. International Conference on Machine Learning, Stockholm, pp 4295–4304
9. Peng B, Rashid T, de Witt CAS, et al. (2021) FACMAC: factored multi-agent centralised policy gradients. Neural Information Processing Systems, Online, pp 12208–12221
10. Naveed K B, Qiao Z, Dolan J M (2021) Trajectory planning for autonomous vehicles using hierarchical reinforcement learning. 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), Indianapolis, pp 601–606
11. Parisotto E, Song F, Rae J, et al. (2020) Stabilizing transformers for reinforcement learning. International Conference on Machine Learning, Online, pp 7487–7498
12. Yan Y, Li X, Qiu X, et al. (2022) Relative distributed formation and obstacle avoidance with multi-agent reinforcement learning//2022 International Conference on Robotics and Automation (ICRA). IEEE, pp 1661–1667
13. Ezen-Can A (2020) A comparison of LSTM and BERT for small corpus. arXiv preprint arXiv:2009.05451. Accessed 2 Jan 2024.
14. Bilokon P, Qiu Y (2023) Transformers versus LSTMs for electronic trading. arXiv preprint arXiv:2309.11400. Accessed 2 Jan 2024.
15. Sutton RS, McAllester D, Singh S et al (1999) Policy gradient methods for reinforcement learning with function approximation. Adv Neural Inf Process Syst 12:1–7
16. Dey R, Salem FM (2017) Gate-variants of gated recurrent unit (GRU) neural networks//2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS). IEEE, pp 1597–1600
17. Opendilab,Dl-engine-docs[OL]. https://di-engine-docs.readthedocs.io/zh_CN/latest/env_tutorial/multiagent_particle_zh.html. Accessed 12 Dec 2023.
18. Salar A, Mo C, Mehran M et al (2021) Toward observation based least restrictive collision avoidance using deep meta reinforcement learning. IEEE Robot Autom Lett 6(4):7445–7452

19. Liu C, Zhu E, Zhang Q, Wei X (2021) Exploring the effects of computational costs in extensive games via modeling and simulation. Int J Intell Syst 36:4065–4087
20. Yuan Z, Bo D, Xuan L et al (2021) Decentralized multi-robot collision avoidance in complex scenarios with selective communication. IEEE Robot Autom Lett 6(4):8379–8386