Knowledge-Based Systems 203 (2020) 106140

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Generating attentive goals for prioritized hindsight reinforcement learning

Peng Liu [a], Chenjia Bai [a], Yingnan Zhao [a], Chenyao Bai [b], Wei Zhao [a,*], Xianglong Tang [a]

[a] School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China
[b] Department of Public Education, Shanghai Customs College, Shanghai 201204, China

ARTICLE INFO

Article history:
Received 18 June 2019
Received in revised form 10 June 2020
Accepted 12 June 2020
Available online 17 June 2020

Keywords:
Attentive goals generation
Prioritized hindsight model
Hindsight experience replay
Reinforcement learning

ABSTRACT

Typical reinforcement learning (RL) performs a single task and does not scale to problems in which an agent must perform multiple tasks, such as moving a robot arm to different locations. The multi-goal framework extends typical RL using a goal-conditional value function and policy, whereby the agent pursues different goals in different episodes. By treating a virtual goal as the desired one, and frequently giving the agent rewards, hindsight experience replay has achieved promising results in the sparse-reward setting of multi-goal RL. However, these virtual goals are uniformly sampled after the replay state from experiences, regardless of their significance. We propose a novel prioritized hindsight model for multi-goal RL in which the agent is provided with more valuable goals, as measured by the expected temporal-difference (TD) error. An attentive goals generation (AGG) network, which consists of temporal convolutions, multi-head dot product attentions, and a last-attention network, is structured to generate the virtual goals to replay. The AGG network is trained by following the gradient of TD-error calculated by an actor–critic model, and generates goals to maximize the expected TD-error with replay transitions. The whole network is fully differentiable and can be learned in an end-to-end manner. The proposed method is evaluated on several robotic manipulating tasks and demonstrates improved sample efficiency and performance.

1. Introduction

Reinforcement learning (RL) [1] provides a framework for training an agent to perform a task by trial and error. RL has been successfully applied to a wide range of problems, including human-level performance on Atari games [2,3], the board game Go [4,5], and challenging robotic tasks [6,7]. In each of these tasks, the RL algorithm learns a policy to perform a single task and optimizes a single reward function. However, in many real-world scenarios, the agent needs to perform a diverse set of tasks, such as moving a robot arm to different target positions. In this case, each target position is regarded as a specific goal, and the agent pursues different goals in different episodes. The multi-goal RL [8] algorithm learns a policy to achieve all possible goals in the whole goal space by extending the value function, policy, and reward to become goal-conditional functions. The performance of multi-goal RL algorithm is evaluated by the average success rate of all possible goals.

In multi-goal RL, the agent only receives a reward if it successfully reaches the desired goal. It is almost impossible to reach the goal by chance, even in the simplest environment. Hindsight experience replay (HER) [9] has achieved promising results in the sparse-reward setting of multi-goal RL by treating a virtual goal as the desired one, and frequently giving the agent rewards. Specifically, whereas a typical RL agent learns nothing if it fails to achieve the original goal, the HER agent obtains information from the already-achieved goals by substituting them for the original goal. Because the achieved goals have been completed under the current policy, the agent receives rewards more frequently than when using the original goal, thus accelerating the learning process. HER can be combined with off-policy methods that use a replay buffer to reuse experiences.

However, the virtual goals in HER are uniformly sampled from experience after the replay state, regardless of their significance. The HER agent only considers those goals that easily result in rewards, but does not consider which goals might be more valuable for the agent to pursue. It would be more efficient to prioritize different goals and use the more important goals in the training process.

We adopt the principle of prioritized experience replay [10] in a multi-goal RL setting. The proxy of prioritized experience replay is the magnitude of a transition's temporal-difference (TD) error

* Corresponding author.
E-mail addresses: pengliu@hit.edu.cn (P. Liu), bai_chenjia@stu.hit.edu.cn (C. Bai), ynzhao_atari@hit.edu.cn (Y. Zhao), baichenyao@shcc.edu.cn (C. Bai), zhaowei@hit.edu.cn (W. Zhao).

https://doi.org/10.1016/j.knosys.2020.106140
0950-7051/

is suitable to indicate how valuable the transition is. Transitions with a higher magnitude of TD-error are more important for learning. In this study, we measure the importance of goals by their expected TD-error, which is calculated by the TD-error of replay transitions with the specific goal. However, naively sampling goals with high expected TD-errors from the replay buffer has two limitations: (i) the goals included in the replay buffer all lie in the trajectories generated by previous policies, thus restricting the search space of appropriate goals, which may be anywhere in the goal space; (ii) constructing a prioritized replay buffer is known to be computationally expensive. Thus, we propose an independent module to *generate* goals rather than *sample* them, which enables us to overcome both limitations. We find that goals with high expected TD-errors are not randomly distributed in the goal space, but are gathered in some specific areas that change as the policy updates. It should be possible to use a neural network to learn how to generate goals with high expected TD-errors.

In this paper, we propose a prioritized hindsight model for multi-goal RL in a sparse-reward setting. The attentive goals generation (AGG) network is structured to generate the replay goals. The AGG network consists of temporal convolution networks (TCNs), multi-head dot product attention (MHDPA) [11], and a last-attention network. TCNs have a large receptive field and position-sensitive properties, enabling the AGG network to capture long-term information and position-dependence from episodic experience. MHDPA, proposed in [11], allows the AGG network to use the attention mechanism to perform relational reasoning involving various time steps in an episode. The last attention network converts the feature sequence to a single element by extracting related features of the particular replay state, and then finally outputs the goal. The generated goal is used to recompute the reward and forms the input to an actor–critic model. The AGG network is trained to maximize the expected TD-error of the generated goals by following the gradient of the actor–critic model, and is thus updated as the policy changes. A fully differentiable gradient back-propagation process is derived so that the AGG network can be trained in an end-to-end manner. We evaluate the proposed method on several robotic manipulation tasks, and demonstrate the improved performance and sample efficiency of our approach.

The main contributions of this paper are as follows: (i) a novel prioritized hindsight model is proposed for multi-goal RL in a sparse-reward setting; (ii) a multi-goal RL agent learns from virtual goals generated by an AGG network, which uses temporal convolution and attention mechanism to reason goals; (iii) the AGG network is trained to maximize the expected TD-error of the generated goals and the whole network is fully differentiable; and (iv) several robotic manipulation experiments demonstrate the effectiveness of the proposed method.

The remainder of this paper is organized as follows. Section 2 reviews some related work. Section 3 introduces the background knowledge used in subsequent sections. In Section 4, we describe the proposed prioritized hindsight method and AGG network in detail. Section 5 presents the experimental results. A discussion of the results and the conclusions to this study are given in Sections 6 and 7, respectively.

2. Related work

The multi-goal RL framework generalizes the typical RL policy in terms of both states and goals. Multi-goal RL was first employed in the development of Horde [12], which uses several sub-policies learned by sub-agents to represent the goal-conditional knowledge. Universal value function approximators (UVFAs) [13] formally describe the multi-goal RL problem and provide goal-conditional value functions and policies. HER [9] encourages the agent to learn from previous information in sparse-reward environments, with achieved goals substituted for the original goal to provide the agent with frequent rewards. The knowledge gained in this way can be generalized from replay goals to the whole goal space, and so the agent finally learns to achieve arbitrary goals. Hindsight policy gradient [14] extends HER to on-policy RL domains by adding importance sampling. RIG [15] extends HER to image-based observation environments by constructing a latent space learned by a variational auto-encoder [16], and randomly samples goals from the latent space rather than the observation space. ARCHER [17] analyzes the bias caused by HER and uses aggressive rewards for hindsight transitions. GHER [18] proposes to use guided goals with high policy level to improve generalization. Dynamic HER [19] extends HER to handle dynamic goals by automatically assembling successful experiences from two relevant failures.

Curriculum RL [20] is another research direction that learns goal-conditional policies. However, the purpose of curriculum RL is to solve a final task, rather than a task set. The final task is usually the most complex or comprehensive one, and is typically difficult to solve directly. Curriculum RL learns to solve simple tasks first, and then gradually solves more complex tasks. Powerplay [21] is a classic curriculum method that continually adds new tasks and task solvers to the current policy, enabling the agent to learn new skills without forgetting previously learned skills. This idea has been extended to the creation of increasingly hard initial points and goal points in navigation tasks [22,23]. An asymmetric self-play curricula method has been developed to explore the environment before training for the main task [24]. The task solver explores the environment using intrinsic motivation [25] from the task setter.

Hierarchical RL [26,27] uses the concept of goals to decompose hard problems into several hierarchies. In the two-layer hierarchy architecture [28], the high-level agent proposes subgoals with temporal abstraction, and the low-level agent learns a policy to solve the subgoal in each time step. Recent achievements [29,30] in hierarchical RL also follow the option framework to learn hierarchical policies, where the high-level agent learns from external rewards and the low-level agent learns from intrinsic rewards. The hindsight idea has also been used in hierarchical RL to construct hierarchies with more than three levels [31]. In each level, the agent substitutes hindsight goals for the original goal to obtain rewards more frequently.

Temporal convolution was first used in Wavenet [32] for speech generation. Recent research [33] indicates that temporal convolution can achieve good results in audio synthesis, language modeling, and machine translation by using dilated convolution [34] to enlarge the receptive field for learning long-term dependencies. It has been shown [35] that temporal convolution has a longer memory than recurrent neural networks [36,37], and is thus suitable for our problem of reasoning goals from both current and past experiences. The temporal convolution network has been used in RL domains to solve imitation learning [38] and meta-learning [39] problems.

The attention mechanism [40,41] aims to extract useful problem-related information while removing redundant information. Transformer [42], which is based solely on attention [11], has achieved state-of-the-art results in machine translation. Recently, the attention architecture has been used in RL to solve partial observation problems [43], discover controllable elements of observations [44], derive relational inductive biases of states [45], and develop interpretable agents for deep Q-learning [46].

3. Background

In this section, we introduce the background to the proposed method. Specifically, we discuss the Markov Decision Process (MDP), multi-goal RL, HER, and prioritized experience replay.

3.1. Markov decision process

The RL problem is formed as an MDP. We assume that the environment is fully observable. In each time step, the agent obtains the current state $\boldsymbol{s}_t \in \mathcal{S}$, takes action $\boldsymbol{a}_t$, interacts with the environment, receives a reward r_t, and updates to the next state $\boldsymbol{s}_{t+1}$. The action is chosen based on the current policy $\pi(\boldsymbol{a}|\boldsymbol{s})$. The return at time step t is the cumulative discount reward in the future: $\sum_{i=t}^{T} \gamma^{i-t} r_i$, where γ is the discount factor. The value function $V^{\pi}(s)$ represents the expected return when starting in state s and following policy π thereafter. The action value function $Q^{\pi}(s, a)$ represents the expected return starting from state s, taking action a, and thereafter following policy π until the end of the episode.

The actor–critic model [47] is a kind of RL algorithm that is suitable for environments with a continuous action space, such as robotics. The actor–critic model consists of an actor to parameterize the policy and a critic to represent the value function. Deep deterministic policy gradient (DDPG) [48] is a type of actor–critic method that uses a deterministic policy. The actor outputs an action according to the current state, and then Gaussian noise is added to the action to encourage exploration. The agent interacts with the environment through exploratory actions, determines the next state, and receives a reward. In the training stage, the critic calculates a Q function and then updated to minimize the TD-error. The actor is updated by maximizing the Q function calculated by the critic. DDPG uses a target network and experience replay to stabilize the learning process, as in deep Q-learning [49–51]. The proposed method uses DDPG as the fundamental algorithm.

3.2. Multi-goal RL

We consider multi-goal RL in which an agent learns a policy to achieve multiple goals. Each episode starts with a specific goal $\boldsymbol{g}$, which is uniformly sampled from the goal space $\mathcal{G}$. The goal is fixed for the whole episode, and the agent uses this goal as part of the input to the actor–critic model to interact with the environment. The action $\boldsymbol{a}_t$ is chosen by following policy $\pi(\boldsymbol{s}_t, \boldsymbol{g})$, which is conditioned on both the state and the goal. The reward function $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \boldsymbol{g})$ is a binary reward that becomes 0 only when the state reaches the pre-specified goal; otherwise, it is -1. The reward function indicates that the agent must reach the goal as soon as possible. The definition of the reward is

$$r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \boldsymbol{g}) = \mathbb{1}\{d(f(\boldsymbol{s}_{t+1}), \boldsymbol{g}) < \epsilon\} - 1, \tag{1}$$

where ϵ is the tolerance distance, $d(\cdot, \cdot)$ is the Euclidean distance function, and $\mathbb{1}\{\cdot\}$ is the indicator function. $f(\cdot)$ projects the state from the state space $\mathcal{S}$ to the goal space $\mathcal{G}$. In robotics, the goal space is usually a subspace of the state space. For example, the state space in robot reaching tasks may include the joint position, joint angles, joint velocities, and so on. However, the goal space only consists of three-dimensional space coordinates, because these are sufficient to determine whether the robot arm has reached the desired position. Therefore, the goal space has fewer dimensions than the state space. The projection function only needs to extract specific elements of the state space.

A multi-goal RL agent tries to maximize the cumulative reward in the whole goal space, and should be evaluated by sampling goals from the goal space according to the uniform distribution, i.e., $\boldsymbol{g} \sim unif(\mathcal{G})$. The Q-function $Q(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g})$ implies the expected return of state, action, and goal, as

$$Q(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}) = \mathbb{E}_{s\sim\rho, a\sim\pi}[\sum_{i=t}^{T-1} r(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}_{i+1}, \boldsymbol{g})], \tag{2}$$

where T is the episode length and ρ is the distribution of state determined by the environment. The objective of multi-goal RL is to find a policy $\pi^*(\boldsymbol{s}, \boldsymbol{g})$ that optimizes the value function in both the goal space and the state space: $\pi^*(\boldsymbol{s}, \boldsymbol{g}) = \arg\max_{\pi} Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{g})$ [13].

The Q-learning algorithm in multi-goal RL is trained by minimizing the square of the TD-error of replay experience. The absolute value of the TD-error is computed as

$$|\delta| = |Q(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}) - y_t|, \tag{3}$$

where $y_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \boldsymbol{g}) + \gamma \max_{\boldsymbol{a}'} Q'(\boldsymbol{s}_{t+1}, \boldsymbol{a}', \boldsymbol{g})$ and Q' is computed by the target network. The Q network is trained to minimize $\mathbb{E}[|\delta|^2]$.

The reason for using binary rewards rather than shaped rewards [52] is that they are more suitable for practical tasks. Shaped rewards need to be carefully hand-crafted through some heuristic method, and often result in biased learning. Binary rewards more naturally reflect the original purpose of the task, as there are only two kinds of results: success or failure. However, the binary reward function may make it difficult for the agent to obtain rewards and can hamper the learning process. The HER algorithm is used to overcome this problem.

3.3. Hindsight experience replay (HER)

The HER algorithm [9] is used to solve the sparse-reward problem in multi-goal RL settings. Each episode starts with an original goal $\boldsymbol{og}$ and follows policy $\pi(\boldsymbol{s}_t, \boldsymbol{og})$. The episodic transitions stored in the replay buffer are represented as:

$$E = \{(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}, \boldsymbol{ag}_t, \boldsymbol{og})\}_{t=0}^{T-1}, \tag{4}$$

where $\boldsymbol{ag}_t = f(\boldsymbol{s}_t)$ indicates the *achieved* goal at time step t and $f(\cdot)$ is the projection function. $\boldsymbol{og}$ is the original goal, which is fixed in E. The reward r_t is computed with $\boldsymbol{og}$ using Eq. (1). If the transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}, \boldsymbol{ag}_t, \boldsymbol{og})$ is sampled for training, then the reward r_t will be almost -1 because the agent finds it difficult to reach $\boldsymbol{og}$, and the policy will update slowly.

The idea of HER is to present a virtual goal $\boldsymbol{g}_{Her}$ that is easier to achieve than $\boldsymbol{og}$. $\boldsymbol{g}_{Her}$ is sampled after the replay time step from the same episode as the replay transition. For example, if the replay transition is sampled from E at step t, then $\boldsymbol{g}_{Her}$ is randomly sampled from the achieved goals after step t,

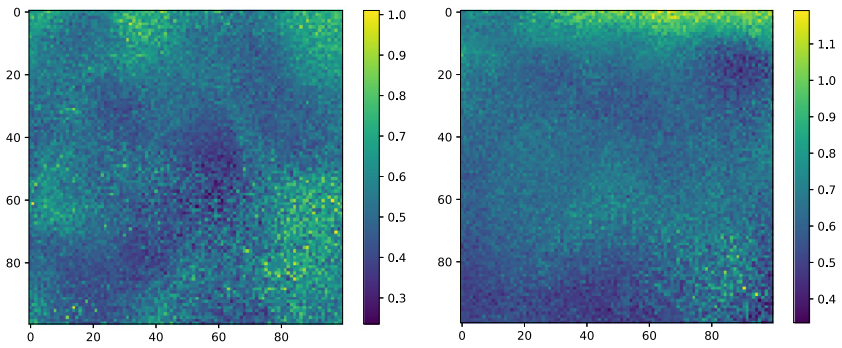
$$\boldsymbol{g}_{Her} = unif(\boldsymbol{ag}_{t+1}, \boldsymbol{ag}_{t+2}, \ldots, \boldsymbol{ag}_{T-1}). \tag{5}$$

The goal-conditional reward function is then recomputed as $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \boldsymbol{g}_{Her})$. Because $\boldsymbol{g}_{Her}$ is in the same episode as the replay state, the probability of $d(\boldsymbol{ag}_{t+1}, \boldsymbol{g}_{Her}) < \epsilon$ will generally be greater than that of $d(\boldsymbol{ag}_{t+1}, \boldsymbol{og}) < \epsilon$. The virtual goal used in HER allows the agent to receive rewards more frequently, thus accelerating the learning process.

However, the substitute goals are uniformly sampled from the achieved-goal sequence in the future, regardless of their significance. The HER algorithm only considers which virtual goals easily produce rewards, but does not consider which goals are more important for the agent to learn. It would be more efficient if we could prioritize goals and use more valuable goals for training.

3.4. Prioritized experience replay

Deep RL requires massive amounts of data for training. In off-policy RL methods, such as DQN and DDPG, the experiences are stored in the replay buffer to reuse. Although randomly sampling from the replay buffer is an easy default choice, the performance of RL algorithms can be further improved by introducing some sampling guidelines. Prioritized experience replay [10] is based



(a) Step 100 (b) Step 500

Fig. 1. Visualization of the distribution of expected TD-error in the goal space of the FetchSlide task. We sample 10000 goals equidistantly in the goal space. For each goal, the policy executes for 50 episodes, then the TD-errors in all time steps are averaged. The distribution changes when the policy is updated; we visualize the distribution at training steps 100 and 500.

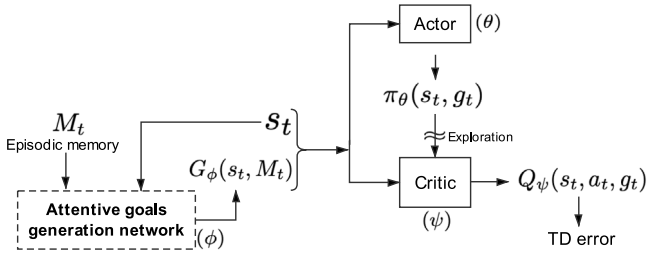


Fig. 2. The AGG network parameterized by $\boldsymbol{\phi}$ uses current state $\boldsymbol{s}_t$ and episodic memory M_t as input, then outputs $\boldsymbol{g}_t = G_{\boldsymbol{\phi}}(\boldsymbol{s}_t, M_t)$. The actor parameterized by $\boldsymbol{\theta}$ calculates an action according to $\pi_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{g}_t)$. The action has exploration noise added, and is then input to the critic. The critic parameterized by $\boldsymbol{\psi}$ outputs the action value function $Q_{\boldsymbol{\psi}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}_t)$, and then the TD-error is computed.

on the idea that the TD-error provides a good proxy for the utility of an experience. High TD-errors indicate that an experience is more surprising and unexpected to the agent; therefore, the probability of this experience being sampled again should be higher than that of an experience associated with a low TD-error.

We integrate the principle of prioritized experience replay into multi-goal RL by using goals that have high expected TD-errors for training. Prioritized experience replay is an approximate means of accomplishing this by sampling the transitions of a specific goal and averaging them to get the expected TD-error. However, the process can restrict the search space and is computationally expensive. Thus, we propose an independent AGG network to *generate* goals with high expected TD-errors, rather than *sample* goals from the replay buffer. The AGG network can be updated to follow the changes in TD-error distribution and policy.

4. Proposed method

This section describes the proposed method in detail. In Section 4.1, we discuss our motivation for generating goals based on TD-error. In Section 4.2, we introduce the overall process of the proposed method and derive the training rules of the model. Section 4.3 presents the architecture of the proposed AGG network. The algorithmic description of the proposed method is given in Section 4.4, and the relationship between our method and HER is discussed in Section 4.5.

4.1. Motivation

To analyze the distribution of expected TD-error in the goal space, we visualize the value of TD-error with sampled goals in the *FetchSlide* task. This task requires the robot's arm to hit and push a slider into the target position. The goal space is represented by a three-dimensional vector (x, y, z) that describes the position of the slider in space; z is fixed to imply that the slider is always lying on the table. The goal space of *FetchSlide* only varies in the x and y dimensions, making it convenient for visualization.

We denote the x, y, and z axes of the goal space as $\mathcal{G}_x$, $\mathcal{G}_y$, and $\mathcal{G}_z$, respectively. Because the z-axis is fixed, we only consider the variation in $\mathcal{G}_x$ and $\mathcal{G}_y$. We sample 100 values equidistantly in the range of $\mathcal{G}_x$; these values are denoted as $\{\boldsymbol{g}_{x_1}, \boldsymbol{g}_{x_2}, \ldots, \boldsymbol{g}_{x_i}, \ldots, \boldsymbol{g}_{x_{100}}\}$. Similarly, we sample 100 values equidistantly in the range of $\mathcal{G}_y$; these values are denoted as $\{\boldsymbol{g}_{y_1}, \boldsymbol{g}_{y_2}, \ldots, \boldsymbol{g}_{y_j}, \ldots, \boldsymbol{g}_{y_{100}}\}$. Then, we combine each $\boldsymbol{g}_{x_i}$ and $\boldsymbol{g}_{y_j}$ with a fixed $\boldsymbol{g}_z$ to obtain the sampled goals in the whole goal space, which are denoted as

$$\{(\boldsymbol{g}_{x_1}, \boldsymbol{g}_{y_1}, \boldsymbol{g}_z), \ldots, (\boldsymbol{g}_{x_i}, \boldsymbol{g}_{y_j}, \boldsymbol{g}_z), \ldots, (\boldsymbol{g}_{x_{100}}, \boldsymbol{g}_{y_{100}}, \boldsymbol{g}_z)\}.$$

Furthermore, $\boldsymbol{g}_{ij} = (\boldsymbol{g}_{x_i}, \boldsymbol{g}_{y_j}, \boldsymbol{g}_z)$. We obtain 10000 sampled goals because $i \in [1, 2, \ldots, 100]$ and $j \in [1, 2, \ldots, 100]$. For each sampled goal $\boldsymbol{g}_{ij}$, we compute the expected TD-error $\mathbb{E}_{\boldsymbol{\pi}}[\delta_{\boldsymbol{g}_{ij}}]$ of $\boldsymbol{g}_{ij}$ with a specific policy π. In particular, we use π at training epochs 100 and 500 of the HER algorithm. The distributions of the absolute values of $\mathbb{E}_{\boldsymbol{\pi}}[\delta_{\boldsymbol{g}_{ij}}]$ are shown in Figs. 1(a) and 1(b), respectively.

To compute $\mathbb{E}_{\boldsymbol{\pi}}[\delta_{\boldsymbol{g}_{ij}}]$, the policy $\boldsymbol{\pi}$ is executed with $\boldsymbol{g}_{ij}$ for 50 episodes, and the TD-error is computed in each time step. Subsequently, all TD-errors are averaged to describe the expected TD-error of the goal, as follows:

$$\mathbb{E}_{\pi}[\delta_{\boldsymbol{g}_{ij}}] = \frac{1}{50T}\left[\sum_{e=0}^{50}\sum_{t=0}^{T-1}\left[r_t^e + \gamma Q(\boldsymbol{s}_{t+1}^e, \boldsymbol{a}_{t+1}^e, \boldsymbol{g}_{ij}) - Q(\boldsymbol{s}_t^e, \boldsymbol{a}_t^e, \boldsymbol{g}_{ij})\right]\right]. \tag{6}$$

We compute the expected TD-error of 10000 sampled goals using Eq. (6); thus, we obtain 10000 expected TD-errors as $\{\mathbb{E}_{\pi}[\delta_{g_{ij}}]\}$, where $i \in [1, 2, \ldots, 100]$ and $j \in [1, 2, \ldots, 100]$. Each point in Fig. 1(a) or Fig. 1(b) represents an absolute value of $\mathbb{E}_{\pi}[\delta_{g_{ij}}]$ for a specific goal g_{ij} under the policy π at epoch 100 or 500. We arrange these 10000 values as a square image of size $100 * 100$ according to its position in the goal space, as shown below. The coordinate in the figure indicates the relative position of goals in the goal space; the correspondence between the aforementioned coordinates and the actual physical coordinates can be obtained using the definition of g_{ij}. We analyze the visualization shown in the aforementioned figure as follows.

1. The choice of goals can significantly affect the value of the TD-error. Because the reward function ensures that the agent only receives rewards when the goal is reached, the trajectory of the whole episode exhibits some tendency towards the goal. The goal significantly affects the states and actions chosen in the trajectory, and thus impact greatly on TD-error. For example, if the goal lies in an area that the agent can reach easily, then the average TD-error of the whole episode becomes low because the value-function estimation is accurate in this area. The goals with low expected TD-errors are less useful for improving the agent's skills.
2. The TD-error distribution changes when the policy is updated. As shown in Fig. 1(a), in the early stages of training, the goals with high expected TD-errors are scattered in blocks across the goal space. As the policy improves, more and more goals can be accurately estimated by the value function. The goals with high expected TD-errors that are yet to be learned become clustered at the edge of the goal space, as shown in Fig. 1(b).

Based on the above analysis, the distribution of expected TD-error is not random or irregular in the goal space. Randomly sampling goals from each episode, as in HER [9], fails to utilize the structure of the TD-error distribution and is less efficient for learning. It would be better if we could use the structure of the TD-error distribution and replay goals with high expected TD-errors during training. Naively sampling goals with high TD-errors only searches through those goals that are stored in the replay buffer, rather than over the whole goal space. Thus, we propose the use of a neural network to learn how to *generate* goals with high expected TD-errors in the whole goal space. This overcomes the restriction of the search space when sampling goals and is more computationally efficient. Moreover, the parameters of the network can update with changes in the TD-error distribution, by connecting the proposed network with the actor–critic model, ensuring that the generated goals change as the policy is updated.

4.2. Prioritized hindsight model

The architecture of the proposed method is shown in Fig. 2. The replay goal $\boldsymbol{g}_t$ used in our architecture is not sampled from experience, but generated by the AGG network (the dotted box in Fig. 2). The AGG network parameterized by $\boldsymbol{\phi}$ uses the current replay state $\boldsymbol{s}_t$ and episodic memory M_t as input, then outputs $\boldsymbol{g}_t$ as according to

$$\boldsymbol{g}_t = G_{\boldsymbol{\phi}}(\boldsymbol{s}_t, M_t), \tag{7}$$

where

$$M_t = [\boldsymbol{ag}_0, \boldsymbol{ag}_1, \ldots, \boldsymbol{ag}_{T-1}, \boldsymbol{og}]. \tag{8}$$

The first T elements of M_t are the achieved-goal sequence of replay episode E, from which $\boldsymbol{s}_t$ is taken. The last element of M_t is the original goal of E. The reasons for using the achieved-goal sequence as episodic input M_t, rather than the state sequence, are as follows: (i) the goal space is a subspace of the state space, which is usually low-dimensional compared to the state space, and thus has lower computational complexity; and (ii) the goal sequence has a better long-term structure than the state sequence because it only contains the essential elements of the state.

The goal $\boldsymbol{g}_t$ generated by the AGG network is used to recompute the reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \boldsymbol{g}_t)$ of the replay transition according to Eq. (1). The generated goal and state are then concatenated as $[\boldsymbol{s}_t, \boldsymbol{g}_t]$ and input into the actor–critic network, which is defined by DDPG. The actor, parameterized by $\boldsymbol{\theta}$, computes the deterministic action $\pi_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{g}_t)$ according to the state and goal. The action has Gaussian noise $\mathcal{N}$ added to encourage exploration, resulting in

$$\boldsymbol{a}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{g}_t) + \mathcal{N}. \tag{9}$$

The action, state, and goal are then concatenated to form the input to the critic network, which is parameterized by $\boldsymbol{\psi}$. The critic network calculates the action value function $Q_{\boldsymbol{\psi}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}_t)$ and the TD-error as $|Q_{\boldsymbol{\psi}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}_t) - y_t|$, where y_t is the target value given by

$$y_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \boldsymbol{g}_t) + \gamma Q_{\boldsymbol{\psi}'}(\boldsymbol{s}_{t+1}, \pi_{\boldsymbol{\theta}'}(\boldsymbol{s}_{t+1}, \boldsymbol{g}_t), \boldsymbol{g}_t), \tag{10}$$

and $Q_{\boldsymbol{\psi}'}$ is the target network. As usual, the proposed method also contains a target network for stable training. The target network is not involved in the backward propagation.

4.2.1. Update rule

The AGG network is trained to generate goals by maximizing the expected TD-error of goals with replay states and actions. During training, parameter $\boldsymbol{\phi}$ of the AGG network follows the gradient of TD-error from the actor–critic model and updates the parameters to maximize the TD-error. The AGG network is updated by applying the chain rule to the expected TD-error calculated by the actor–critic model with respect to the AGG network:

$$\Delta\boldsymbol{\phi} \propto \mathbb{E}_{\boldsymbol{s}\sim\rho,\boldsymbol{a}\sim\pi}[\nabla_{\boldsymbol{g}}(Q_{\boldsymbol{\psi}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}_t) - y_t)^2|_{\boldsymbol{g}_t=G_{\boldsymbol{\phi}}(\boldsymbol{s}_t,M_t)}\nabla_{\boldsymbol{\phi}}G_{\boldsymbol{\phi}}(\boldsymbol{s}_t, M_t)]. \tag{11}$$

According to Eq. (11), the gradient calculation has two stages: (i) we calculate the gradient of the square of the TD-error $(Q_{\boldsymbol{\psi}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}_t) - y_t)^2$ to the generated goal $\boldsymbol{g}_t$, where $\boldsymbol{g}_t$ is the output of the AGG network with the replay transition and parameters $\boldsymbol{\phi}$. We denote the first gradient item as $\Delta\boldsymbol{g}_t$; (ii) $\Delta\boldsymbol{g}_t$ is back-propagated to parameter $\boldsymbol{\phi}$ of the AGG network according to $\boldsymbol{g}_t = G_{\boldsymbol{\phi}}(\boldsymbol{s}_t, M_t)$. Because the AGG network is differentiable, this process can be easily implemented using automatic differentiation. The final gradient of the TD-error with respect to $\boldsymbol{\phi}$ is denoted as $\Delta\boldsymbol{\phi}$.

After computing the gradient $\Delta\boldsymbol{\phi}$, the parameter $\boldsymbol{\phi}$ is updated to maximize the TD-error as $\boldsymbol{\phi} := \boldsymbol{\phi} + \Delta\boldsymbol{\phi}$. As a result, the AGG network follows the gradient of the actor–critic model and the whole network is fully differentiable. The AGG network is trained with the updates to the actor–critic model, thus not only learns to find goals with high TD-errors, but also learns to adapt following policy changes.

The other modules in the proposed model still follow the update rule defined in DDPG. The critic, parameterized by $\boldsymbol{\psi}$, is trained to minimize the TD-error. The gradient is calculated as

$$\Delta\boldsymbol{\psi} \propto \mathbb{E}_{\boldsymbol{s}\sim\rho,\boldsymbol{a}\sim\pi}[\nabla_{\boldsymbol{\psi}}(Q_{\boldsymbol{\psi}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}_t) - y_t)^2], \tag{12}$$

where y_t is calculated by Eq. (10). The parameter of the critic is updated as $\boldsymbol{\psi} := \boldsymbol{\psi} - \Delta\boldsymbol{\psi}$. The actor is trained to maximize the Q value by applying the chain rule to the Q function with respect to the actor parameter:

$$\Delta\boldsymbol{\theta} \propto \mathbb{E}_{\boldsymbol{s}\sim\rho}[\nabla_{\boldsymbol{a}}Q_{\boldsymbol{\psi}}(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}_t)|_{\boldsymbol{a}_t=\pi_{\boldsymbol{\theta}}(\boldsymbol{s}_t,\boldsymbol{g}_t)}\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(\boldsymbol{s}_t, \boldsymbol{g}_t)]. \tag{13}$$

The parameter of the actor is then updated as $\boldsymbol{\theta} := \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.

4.2.2. Exploration of goals

The proposed AGG network follows a deterministic process to generate goals by using TCNs and MHDPA to extract features from episodic memory and the last-attention module. A concern with regard to deterministic goals is that they can overfit the value estimate. When the critic is updated, a learning Q-function that employs deterministic generative goals is highly susceptible

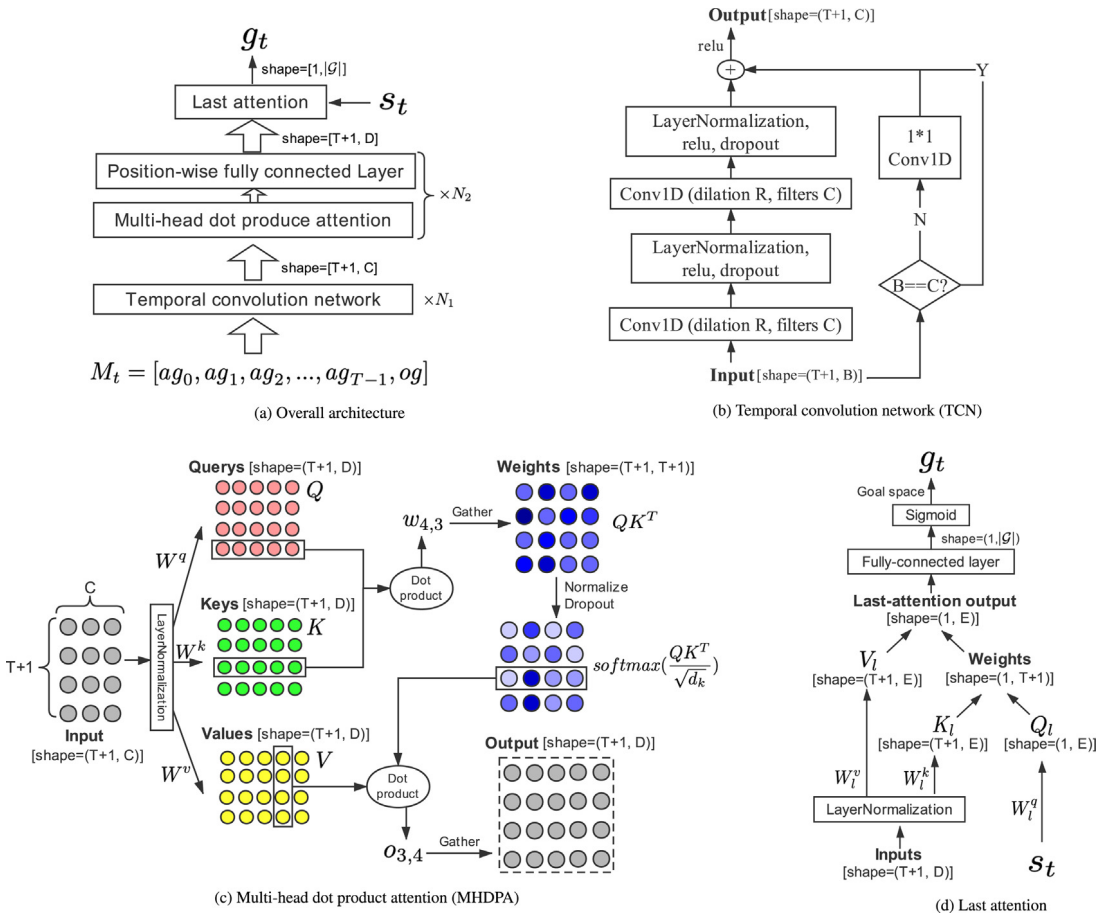


Fig. 3. Architecture of the proposed AGG network. (a) The overall architecture including N_1 temporal convolution network (TCN) blocks, N_2 multi-head dot product attention (MHDPA) and position-wise fully connected layer blocks. The replay state $\boldsymbol{s}_t$ is integrated by the last attention block to output the goal $\boldsymbol{g}_t$. (b) Computation graph of a single TCN block. (c) Computation graph of a single head self-attention block. The query, key, and value all come from the same input, and the output is a sequence of features. (d) Computation graph of the last-attention block. The key and value both come from the same previous features, but the query comes from replay state $\boldsymbol{s}_t$. The output of the last-attention block is a single goal rather than a sequence.

to inaccuracies induced by the function approximation error in deep learning, which can increase the variance of the Q-value. This induced variance can be reduced through regularization. We propose the addition of Gaussian noise to the generated goals as a means of regularization. Our approach enforces the notion that similar goals should have similar expected Q-values. Whereas the function approximation does this implicitly, the relationship between similar goals can be enforced explicitly by modifying the training procedure. The Gaussian noise is added to the generated goal, and the TD-error in the critic update becomes

$$\delta = r + \gamma Q_{\psi'}(\boldsymbol{s}_{t+1}, \pi_{\theta'}(\boldsymbol{s}_{t+1}, \tilde{\boldsymbol{g}}_t + \epsilon_1), \tilde{\boldsymbol{g}}_t + \epsilon_2) - Q(\boldsymbol{s}_t, \boldsymbol{a}_t, \tilde{\boldsymbol{g}}_t + \epsilon_3), \tag{14}$$

where $\epsilon_1, \epsilon_2, \epsilon_3 \sim \mathcal{N}(0, \sigma * \mathcal{G}_{range})$. σ is a scalar and keeps same in all dimensions of the goal space and remains fixed in the training process, $\mathcal{G}_{range}$ is the range of the goal space, which is computed by $\mathcal{G}_{max} - \mathcal{G}_{min}$, and $\tilde{\boldsymbol{g}}_t$ is the generated goal before the noise added. The noise added to the target policy is chosen independently of the policy and AGG network. The Q-value estimate is learned with respect to a noisy policy defined by the parameter σ.

This update rule will smoothen the value estimate by bootstrapping it from similar state–action–goal value estimates, which is beneficial. Furthermore, the addition of Gaussian noise provides diverse goals, which is vital for effective learning in the initial stage of training. Upon the initialization of training, the trajectory generated by the agent will concentrate near the starting point owing to the random policy. The episode memory that is input to the AGG network will also be similar, causing the generated goals to lack diversity. Using Gaussian noise enables the AGG network to increase the variety of generated goals, which leads to better generalization in the goal space and helps the agent explore the surrounding environment.

To avoid introducing stochastic nodes into the AGG network and destroying the differentiability, reparameterization [16] is applied. We first sample the Gaussian noise $\epsilon \sim N(0, I)$, and then compute the exploratory goal $\boldsymbol{g}_t$ as

$$\boldsymbol{g}_t = \tilde{\boldsymbol{g}}_t + \sigma * \epsilon * \mathcal{G}_{range}, \tag{15}$$

The exploration process is considered part of the AGG network.

4.3. AGG Network architecture

The overall architecture of the proposed AGG network is shown in Fig. 3(a). The network takes episodic memory $M_t = [\boldsymbol{ag}_0, \boldsymbol{ag}_1, \ldots, \boldsymbol{ag}_{T-1}, \boldsymbol{og}] \in \mathbb{R}^{T+1,|\mathcal{G}|}$ as its input, where $|\mathcal{G}|$ is the dimension of the goal space. The N_1 temporal convolution network (TCN, Fig. 3(b)) blocks follow, and these output a $(T+1, C)$-dimensional feature sequence. Next, N_2 multi-head dot product attention (MHDPA, Fig. 3(c)) blocks and position-wise fully connected layer blocks are used, and these output a $(T+1, D)$-dimensional feature sequence. Finally, the replay state $\boldsymbol{s}_t$ is integrated into the model by the last-attention block (Fig. 3(d)), which combines useful features in the sequence that

are related to $\boldsymbol{s}_t$ and converts the feature sequence to a single element $\boldsymbol{g}_t \in \mathbb{R}^{1,|\mathcal{G}|}$ as the final output of the AGG network.

The TCNs are used in the AGG network to extract features from episodic input M_t. The reasons for using TCNs are as follows: (i) the input sequence M_t usually has more than 100 elements, and the network needs to have a large receptive field to extract comprehensive information from the input. The receptive field of TCNs increases exponentially through dilation convolution and increasing the dilation rate exponentially; (ii) the TCNs are sensitive to position order, which is suitable for learning position-dependence from episodic experiences that have temporal orders; and (iii) the TCN model also exhibits a long useful memory, and is thus suitable for our problem of reasoning goals from both the current experience and the path using implicit memory. Moreover, TCNs are simpler and clearer for sequence modeling than recurrent networks, and are also computationally efficient as they support parallel computing. Details of the TCNs are discussed in Section 4.3.1.

MHDPA, proposed in [11], is a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between the input and output. An attention function can be described as mapping a query and a set of key–value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed using a compatibility function of the query with the corresponding key. Multi-head attention allows the model to jointly consider information from different representation subspaces at various positions. We adopt the MHDPA network from [11] to perform relational reasoning across different time steps in an episode. The reasons for using MHDPA are as follows: (i) the feature sequence has $T+1$ time steps, and feature vectors in different time steps may be related to one another. The TCNs perform well in extracting the relationship between adjacent features, using convolution to provide a temporal context. Thus, MHDPA is used to learn the relationship between features that are far apart. The MHDPA blocks connect all the time steps using a constant number of operations, allowing the model to pinpoint specific knowledge from a large contextual space [11]; and (ii) because the feature sequence is long, it is inefficient to consider all the features. We use the attention mechanism to select useful features and remove redundant ones. However, MHDPA also has limitations in dealing with RL problems. The MHDPA blocks treat different steps separately in terms of query, key, and value, regardless of their position order. In RL problems, the positions of states are important because the agent reaches them sequentially. Therefore, we use TCNs before MHDPAs to learn position order features and position-dependence, thus overcoming this shortcoming of MHDPA. Further details of MHDPA are described in Section 4.3.2.

The computation process for the last-attention block is similar to that for MHDPA. However, in MHDPA, the query, key, and value all come from the same feature representation (referred to as *self-attention*); in the last-attention block, the key and value also come from the same feature representation, but the *query* comes from replay state $\boldsymbol{s}_t$. The last-attention block is used to extract useful information related to the specific replay state $\boldsymbol{s}_t$. Because the query is a single position rather than a sequence, the last attention converts the feature sequence to a single element that only contains information related to $\boldsymbol{s}_t$. The output $\boldsymbol{g}_t \in \mathbb{R}^{1,|\mathcal{G}|}$ of the last attention is the final output of the AGG network. Details of this module are described in 4.3.3.

4.3.1. Temporal convolution network

We use N_1 TCN blocks to extract a feature representation from an episodic memory of length $T+1$. Each block employs dilated convolutions to enlarge the receptive field. Dilated convolution uses a filter f with kernel size $k = 2r+1$ to convolute a 1D sequence M. The output is $\sum_{i=-r}^{r} f(i) \cdot M_{s+d \cdot i}$, where d is the dilation factor. When $d = 1$, the dilated convolution is equal to regular convolution.

Each TCN block uses two convolution operations. After each convolution, layer normalization [53] is used to scale the features so that they lie in similar ranges across environments and units. A rectified linear unit is then added, and spatial dropout is used for regularization. A residual connection [54] is used in each block for fast back-propagation. In addition, because the input and output may have different dimensions, 1×1 convolution is used to ensure that the skip connection is valid.

With the increase of TCN blocks, the dilation rate increases exponentially (i.e., $d = 2^i$) to increase the receptive field of the network exponentially. The whole process exhibits below-up.

---

function TCNs(Input, n_blocks=N_1)
  x := Input
  for $i = 0, 1, 2, \ldots, N_1 - 1$ **do**
    x := Conv1D(nfilters=D, dilation rate=2^i)(x)
    x := Dropout(Relu(LayerNorm(x)))
    x := Conv1D(nfilters=D, dilation rate=2^i)(x)
    x := Dropout(Relu(LayerNorm(x)))
    x := Relu(x + Input(or $Conv_{1\times1}$ (Input)))
    Input := x
  end for
  return x
end function

4.3.2. Multi-head dot product attention

The MHDPA is applied after the TCN blocks to perform relational reasoning over different time steps (see Fig. 3(c)). The input to MHDPA also has $T+1$ elements. If the dimension of each element is C and the input is $\boldsymbol{M}$, then $\boldsymbol{M} \in \mathbb{R}^{(T+1)\times C}$.

First, the matrices $\boldsymbol{W}^q$, $\boldsymbol{W}^k$, $\boldsymbol{W}^v$ are used to construct the query ($\boldsymbol{Q}$), key ($\boldsymbol{K}$), and value ($\boldsymbol{V}$), respectively, through a linear projection function. The attention weight is computed through the dot-product and softmax function of $\boldsymbol{Q}$ and $\boldsymbol{K}$. Then the attention weight is applied to weight average the different time steps of $\boldsymbol{V}$. Formally,

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\frac{\boldsymbol{Q}\boldsymbol{K}^{\mathrm{T}}}{\sqrt{d_k}})\boldsymbol{V}, \tag{16}$$

where $\boldsymbol{Q} = \boldsymbol{M}\boldsymbol{W}^q$, $\boldsymbol{K} = \boldsymbol{M}\boldsymbol{W}^k$, and $\boldsymbol{V} = \boldsymbol{M}\boldsymbol{W}^v$. d_k is a scaling factor that is equal to the dimension of the key vector. The only trainable parameters in a single-head MHDPA are $\boldsymbol{W}^q$, $\boldsymbol{W}^k$, and $\boldsymbol{W}^v$. The dimensions of each variable are shown in Fig. 3(c).

We use MHDPA including multiple heads to perform complex reasoning. Each head has the independent parameters $\boldsymbol{W}_i^q$, $\boldsymbol{W}_i^k$, and $\boldsymbol{W}_i^v$ to perform dot-product attention. The result from each head is then concatenated. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions, i.e.,

$$\text{MHDPA}(\boldsymbol{M}) = \text{concat}([\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n])\boldsymbol{W}^o, \tag{17}$$

where

$$\boldsymbol{h}_i = \text{Attention}(\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i), \tag{18}$$

and $\boldsymbol{W}^o$ is used for the output projection. Layer normalization is performed before each MHDPA, and residual connection is used after each MHDPA.

A position-wise fully connected network (PFN) is employed after each MHDPA. This operation is applied to each position separately and identically. Layer normalization and residual connection are also included in PFN. Formally, if the output of MHDPA is denoted as $\tilde{\boldsymbol{M}}$, the process of PFN is

$$\text{PFN}(\tilde{\boldsymbol{M}}) = \tilde{\boldsymbol{M}} + \text{Dense}(\text{Relu}(\text{Dense}(\text{LayerNorm}(\tilde{\boldsymbol{M}})))), \tag{19}$$

where "Dense" is the fully connected network.

4.3.3. Last attention

The process of the last-attention module is similar to that of MHDPA. However, the query does not come from the previous feature sequence, but from the current replay state $\boldsymbol{s}_t$, as shown in Fig. 3(d). We denote the query, key, and value as $\boldsymbol{Q}_l, \boldsymbol{K}_l, \boldsymbol{V}_l$. Because $\boldsymbol{s}_t$ is a single element rather than a sequence, the attention weights computed by $\boldsymbol{Q}_l \boldsymbol{K}_l^{\mathrm{T}}$ form a $(T+1)$-dimensional vector rather than a $(T+1, T+1)$ matrix, as in MHDPA. The output of the last-attention block becomes a single element rather than a sequence. Formally, if we denote the input feature sequence as $\boldsymbol{X}_l$, then

$$\text{LastAttention}(\boldsymbol{X}_l, \boldsymbol{s}_t) = \text{softmax}(\frac{(\boldsymbol{s}_t \boldsymbol{W}_l^q)\boldsymbol{K}_l^{\mathrm{T}}}{\sqrt{d_k}})\boldsymbol{V}_l, \tag{20}$$

where $\boldsymbol{s}_t \boldsymbol{W}_l^q = \boldsymbol{Q}_l \in \mathbb{R}^{1\times E}$, $\boldsymbol{K}_l = \boldsymbol{X}_l \boldsymbol{W}_l^k \in \mathbb{R}^{(T+1)\times E}$, and $\boldsymbol{V}_l = \boldsymbol{X}_l \boldsymbol{W}_l^v \in \mathbb{R}^{(T+1)\times E}$. The matrices $\boldsymbol{W}_l^q, \boldsymbol{W}_l^k, \boldsymbol{W}_l^v$ are used to construct the query, key, and value, respectively, in the last-attention block. Layer normalization is used for preprocessing. In addition, an entropy regularization loss is added to the attention weight to encourage exploration and improve performance; details are discussed in the experimental section.

We apply a position-wise fully connected layer after the last-attention block to change the dimension of the output to that of the goal space, $|\mathcal{G}|$. A sigmoid function is then used to change the range of each dimension to $0 \sim 1$. Finally, we project the output into the goal space and obtain $\boldsymbol{g}_t$. Formally,

$$\boldsymbol{g}_t = \mathcal{G}_{min} + \mathcal{G}_{range} * \text{sigmoid}(\text{Dense}(\text{LastAttention}(\boldsymbol{X}_l, \boldsymbol{s}_t))), \tag{21}$$

where $\mathcal{G}_{min}$ is the low boundary of the goal space, $\mathcal{G}_{range}$ is the range of the goal space, and $\boldsymbol{g}_t \in \mathbb{R}^{|\mathcal{G}|}$ is the output.

The generated goal $\boldsymbol{g}_t$ is the output of the whole AGG network. This is used to recompute the reward function and as the input to the actor–critic model. The gradient $\Delta\boldsymbol{\phi}$ calculated in Eq. (11) is used to update the weights in the TCNs, MHDPAs, and last-attention block.

4.4. Algorithmic description

The proposed method is formally described in Alg. 1. In lines 3–8, the agent interacts with the environment to identify experiences. Each episode starts with an original goal, and all experiences are stored in the replay buffer. In lines 9–22, the experiences are replayed and used to train the model. In each training step, n episodes are sampled. The achieved goal sequence and current replay state are used as inputs to the AGG network. The AGG network then generates goals according to the parameters of the TCNs, MHDPAs, and last-attention module. Next, exploration noise is added, and the goal-conditional reward function is recomputed according to the generated goal. Finally, the gradients are computed and the whole network is updated. We have included a loop in lines 11–21 to ensure a precise description. In practice, a batch operation is used to calculate the gradients and update the network.

Algorithm 1 Prioritized hindsight model

1: Initialize the AGG network ($\boldsymbol{\phi}$), actor ($\boldsymbol{\theta}$), and critic ($\boldsymbol{\psi}$).
2: **for** episode $i = 1, 2, ..., M$ **do**
3: Sample the original goal $\boldsymbol{g}_o \in \mathcal{G}$ uniformly.
4: **for** $t = 0, 1, ..., T-1$ **do**
5: Get action from actor $\boldsymbol{a}_t = \pi(\boldsymbol{s}_t, \boldsymbol{g}_o) + \mathcal{N}$.
6: Execute action $\boldsymbol{a}_t$ and observe a new state $\boldsymbol{s}_{t+1}$.
7: **end for**
8: Store $E^{(i)} = \{(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \boldsymbol{g}_t, \boldsymbol{g}_o)\}_{t=0}^{T-1}$ in replay buffer $\mathcal{D}$.
9: **for** $j = 1, ..., m$ **do**
10: Sample n episodes $\mathcal{D}^j = \{E^{(1)}, ..., E^{(n)}\}$ from $\mathcal{D}$.
11: **for** $k = 1, ..., n$ **do**
12: Sample $(\boldsymbol{s}_t^{(k)}, \boldsymbol{a}_t^{(k)}, \boldsymbol{s}_{t+1}^{(k)}, \boldsymbol{g}_t^{(k)}, \boldsymbol{g}_o^{(k)})$ from $E^{(k)}$.
13: Construct $M_t = [\boldsymbol{g}_0^{(k)}, ..., \boldsymbol{g}_{T-1}^{(k)}, \boldsymbol{g}_o^{(k)}]$, $\boldsymbol{s}_t = \boldsymbol{s}_t^{(k)}$.
14: Compute $\boldsymbol{g}_t = G_\phi(\boldsymbol{s}_t, M_t)$ as the substitute goal.
15: Add exploration noise to $\boldsymbol{g}_t$ as Eq. (15).
16: Recompute goal-conditional reward as Eq. (1).
17: Compute the gradients $\Delta\boldsymbol{\phi}$, $\Delta\boldsymbol{\theta}$, and $\Delta\boldsymbol{\psi}$.
18: ▷ Eqs. (11), (12), and (13)
19: Use $\Delta\boldsymbol{\phi}$ to update TCN, MHDPA, last attention.
20: Use $\Delta\boldsymbol{\theta}$ and $\Delta\boldsymbol{\psi}$ to update actor–critic network.
21: **end for**
22: **end for**
23: **end for**

4.5. Relation to HER

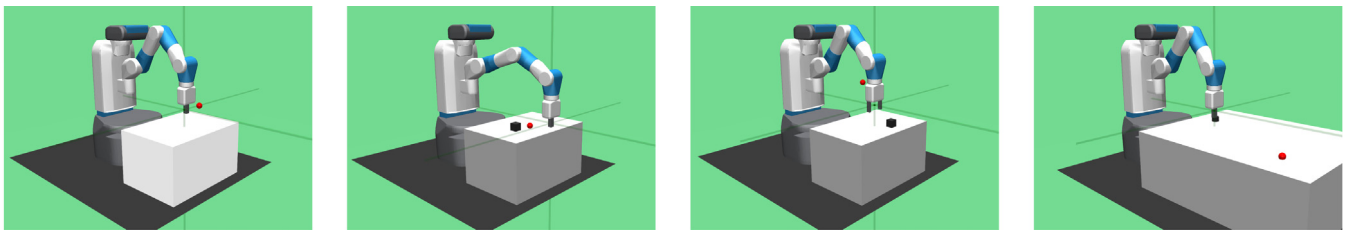
The HER algorithm [9] chooses goals after the replay state from the replay trajectory. The hypothesis of HER is that there exist some valuable elements in the interaction trajectory that are important for the replay state to pursue. The HER algorithm considers these elements to lie behind the replay state uniformly. However, this hypothesis is relatively subjective. In this study, we do not assume the positions of valuable goals, and instead use the AGG network to learn the proxy automatically. The parameter of the AGG network can be trained and changed along with the policy updates. The TCNs and MHDPAs obtain the feature representation of the replay trajectory, and the last-attention block extracts specific information related to the replay state.

If all parameters of the TCNs and MHDPAs are set to zero, these two modules approximately represent the identity mapping because of the use of the residual connection. Moreover, if the last-attention block uses random parameters, the attention weight will also be random. The final generated goal $\boldsymbol{g}_t$ can be regarded as the mixture of different elements in different positions of the initial input M_t, with equal probabilities. The whole model is similar to the random sampling of elements from the original achieved-goal sequence, which can be considered a variant of HER. Hence, the HER method is included in the representation space of our model.

5. Experiments

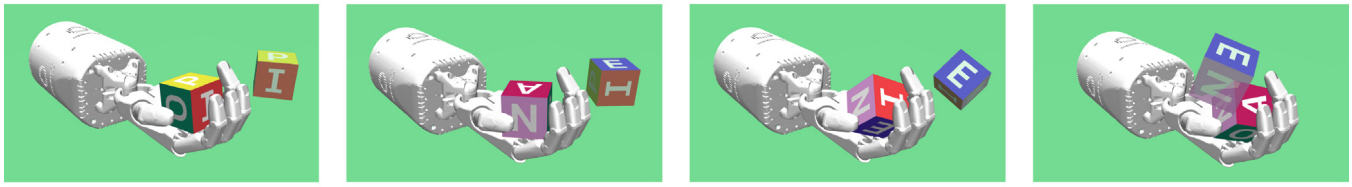
This section describes a series of experiments conducted to evaluate the performance of our method. We first describe the experimental environments and implementation details, and then the architecture, hyper-parameters, and attention visualization used in the experiments. Finally, the results are compared with those from several baselines. The supplementary material is available online.[1]

[1] https://sites.google.com/view/prioritized-her.



(a) FetchReach (b) FetchPush (c) FetchPickAndPlace (d) FetchSlide

Fig. 4. Robot fetch environment.



(a) HandManipulateBlockRotateZ (b) HandManipulateBlockRotateParallel (c) HandManipulateBlockRotateXYZ (d) HandManipulateBlockFull

Fig. 5. Robot hand block environment.

5.1. Environments

The proposed method is evaluated using the robot fetch and hand block environments from OpenAI Gym [55].

5.1.1. Fetch environment

The robot fetch environment uses a seven degree-of-freedom (7-DoF) robotic arm and a gripper to complete tasks. This environment contains four tasks, as shown in Fig. 4.

- *FetchReach*. Move the gripper to the target position.
- *FetchPush*. Push the box to the target position on the table. The gripper is not used in this task.
- *FetchPickAndPlace*. Move the gripper and use it to grasp the box, then move the box to the target position, which may be on the table or in the air.
- *FetchSlide*. Push and hit the slider to the target position. Because the target position may lie beyond the reachable limit of the arm, the gripper may need to hit the slider to make it slide.

Goal. The goal indicates the desired position of the task. In the above tasks, the goal space has three dimensions. The reward is 0 when the distance between the goal and the current state is less than 5 cm, and is -1 otherwise.

State. The state is described by the angles and velocities of the arm, and also the position of the box. The state space has 10 dimensions in FetchReach and 25 dimensions in the other three environments.

Action. The action controls the movement of the arm and the opening of the gripper. The action space has four dimensions.

5.1.2. Hand block environment

The robot block environment uses a 24-DoF anthropomorphic robotic hand to manipulate a block to a target position or rotation. This environment contains four tasks, as shown in Fig. 5.

- *HandManipulateBlockRotateZ*. Manipulate the box to meet the z-axis of the target rotation. No target position required.
- *HandManipulateBlockRotateParallel*. Manipulate the box to meet the x-axis and y-axis of the target rotation; no requirement for the z-axis. No target position required.
- *HandManipulateBlockRotateXYZ*. Manipulate the box to meet all axes of the target rotation. No target position required.
- *HandManipulateBlockFull*. Manipulate the box to meet all axes of the target rotation and also the target position.

Goal. The goal indicates the target position and target rotation of the task. The goal space has seven dimensions in all tasks.

State. The state is described by the position, velocity, Cartesian position, and Cartesian rotation of all joints. The state space has 61 dimensions in all tasks.

Action. The action uses absolute position control for all non-coupled joints of the hand. The action space has 20 dimensions in all tasks.

5.2. AGG Network details

Architecture details. The AGG network contains eight TCN blocks. The dilation rate of each block increases exponentially. The receptive field of the top layer is more than 100 to cover the input sequence. Each convolution layer has 32 filters. A dropout rate of 0.1 is applied after each convolution. After temporal convolutions, the network contains two MHDPA blocks. The linear projection matrices $\boldsymbol{W}^q$, $\boldsymbol{W}^k$, $\boldsymbol{W}^v$ are $(32, 32)$-dimensional. Each MHDPA has four attention heads. The query, key, and value are $(T + 1, 32)$-dimensional. A dropout rate of 0.1 is applied to the attention weights. After each MHDPA, a position-wise fully connected block with two layers is used, consisting of 64 and 32 units, respectively. A dropout rate of 0.1 is applied between the two layers. The linear projection matrices $\boldsymbol{W}_l^q$, $\boldsymbol{W}_l^k$, $\boldsymbol{W}_l^v$ are $(32, 32)$-dimensional in the last-attention block. No dropout is applied in this module.

The actor–critic network in our method is the same as in the HER algorithm. The actor and critic both use three fully connected layers, with 256 hidden units and ReLu. The actor network uses $L2$ regularization to punish large actions; the ratio is set to 1.0.

Network scale ablation. The parameters used in our method are compared with those of the HER algorithm in Table 1. Because the dimensions of the goal space are different in the robot fetch and hand block environments, the numbers of parameters are also different. According to Table 1, although the AGG network has several modules for temporal convolutions and attention, the total number of parameters is only 24.6% greater than that of HER in the robot fetch environment and 22.9% greater in the

Table 1
Parameter comparison.

Network	Fetch Env		Hand Env	
	Ours	HER [9]	Ours	HER [9]
Actor	140,289	140,289	154,625	154,625
Critic	140,036	140,036	154,388	154,388
AGG-net	68,899	–	70,695	–
Total	**349,224**	**280,325**	**379,708**	**309,013**

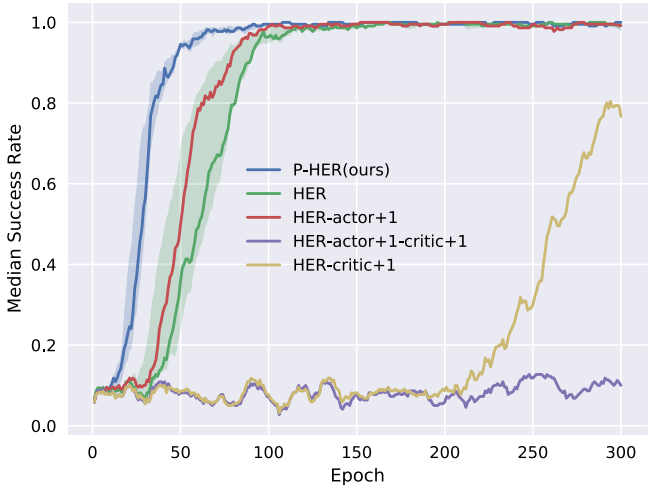


Fig. 6. Performance comparison of **Network scale ablation**. We add layers to the actor and critic networks in HER. The results show that simply increasing the number of parameters does not improve the performance, and actually causes adverse effects in 'HER-critic+1' and 'HER-actor+1-critic+1'. The performance improvement in our method is not due to the increase in network scale.

robot hand block environment. This is because temporal convolution and attention both share weights in different time steps, which reduces the number of parameters compared with a fully connected actor–critic network.

To verify that the improved performance achieved by our method is not the result of the increased number of parameters, we increased the number of parameters in HER [9] to show that the parameter scale is not the main reason to affect the performance. A comparison was conducted using the FetchPush task, as shown in Fig. 6. The proposed prioritized hindsight model is denoted as P-HER. The HER model is added by additional layers to increase the number of parameters:

- HER-actor+1: adding an additional layer in the actor network, causing the total number of parameters to approach that of our method;
- HER-critic+1: adding an additional layer in the critic network, also causing the total number of parameters to approach that of our method;
- HER-actor+1-critic+1: an additional layer is added to both the actor and critic networks, resulting in 18% more parameters than in our method.

The results in Fig. 6 show that increasing the number of parameters in HER does not improve the performance, and actually causes adverse effects in 'HER-critic+1' and 'HER-actor+1-critic+1'. Thus, the performance improvement of our method is not the result of an increase in the network scale, but comes from considering the expected TD-error of goals and using the AGG network to generate more valuable goals for training.

TCN ablation. TCNs are used in the AGG network mainly because they have a large receptive field and enable the position-dependence of episodic experiences to be learned. We performed several comparative experiments in FetchPush to verify these properties.

As shown in Fig. 7(a), we reduced the number of TCN blocks to verify the importance of a large receptive field to the network. The receptive field is halved when a TCN block is removed. For every two TCN blocks removed, we add an additional MHDPA block so that the total number of parameters is generally unchanged. The receptive field for TCNs with 6 blocks is 2^6 (i.e., 64). This value slightly exceeds the length of episode sequence M_t (51 in the Fetch environment). The result shows that the performance decreases substantially when the number of TCN blocks is less than 6. We also employ more TCN blocks, specifically, 8 TCN blocks, and achieve improved performance. However, when the number of TCN blocks is increased to 10, we achieve a performance similar to that achieved using 8 TCN blocks, which is because using more blocks also increases the parameters of the network. Thus, we use 8 TCN blocks in all experiments.

As shown in Fig. 7(b), we replaced the TCNs with position encoding (PE) to compare the ability of TCNs and PE to model the positional information in multi-goal RL. PE uses sine and cosine functions to encode the position information, and is often used in sequence modeling to inject the relative or absolute positions of tokens in a sequence. PE is computed as

$$PE_{pos,2i} = sin(pos/10000^{2i/d}), \tag{22}$$

$$PE_{pos,2i+1} = cos(pos/10000^{2i/d}), \tag{23}$$

where pos is the position, i is the dimension, and d is the scale factor. We conducted PE experiments with $d = 10^2$, 10^3, and 10^4. In each case, we removed all TCNs and added four MHDPA blocks so that the total number of parameters in the AGG network is generally unchanged.

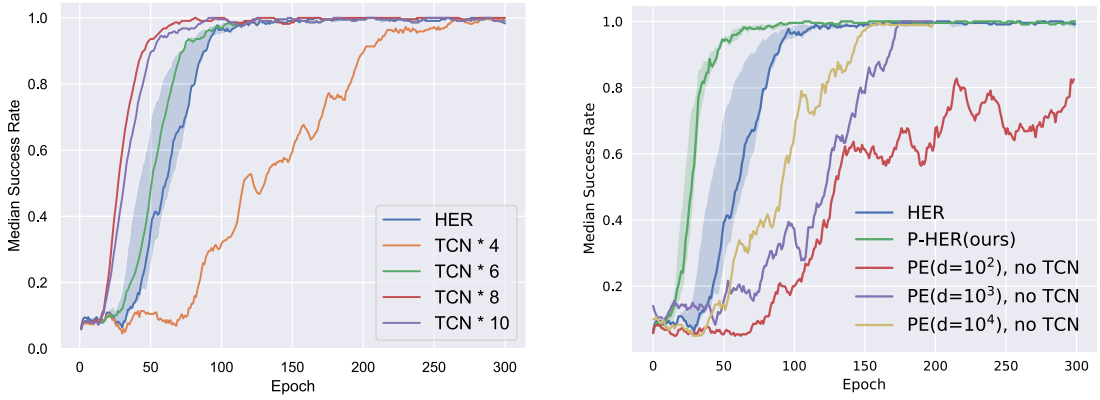
The performance with PE is relatively worse than that using TCNs. This is mainly because temporal convolution not only learns the encoding of position order, but also learns the position-dependence between each time step. Moreover, PE is unchanged throughout the training process, but TCNs are updated along with the policy. On the basis of these results, we do not use PE in our model.

MHDPA and Last attention. The AGG network contains two MHDPA blocks in all tasks. We increased the number of MHDPA blocks to 3–5, but no significant performance increases were observed. The network is not sensitive to the number of MHDPA blocks.

The last-attention block converts the feature sequence to a single element by using the replay state to construct the attention query. The last-attention block is indispensable to the AGG network and is only used once.

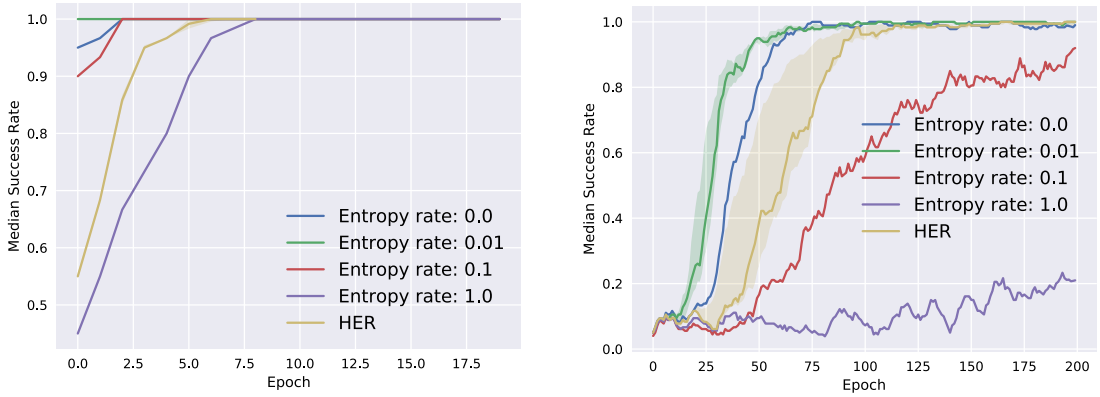
5.3. Prioritized hindsight model detail

Training details. The training process of our model is almost the same as for HER [9]. Each task is trained for 10^3 epochs, each epoch has 50 cycles, and each cycle has two parallel interactions. The actions are chosen by the policy network using an ϵ-greedy strategy with a probability of 0.3. The actions have Gaussian noise added with a factor of 0.2. After each cycle, the experiences of two episodes are stored in the replay buffer, which has a size of 10^6. The experiences are then replayed for training. The original goal of transition is substituted for the goal generated by the AGG network, and then the reward is recomputed. The gradient of the network is clipped by a norm of 5, and the batch size is set to 256. The episode length is 50 in the robot fetch tasks and 100 in the robot hand block tasks. The target value in TD-error is calculated by the target network, which is updated by Polyak averaging with a ratio of 0.95. The exploration factor σ of generated goals is set to 10^{-3}. The original goals and HER goals are used to enhance the



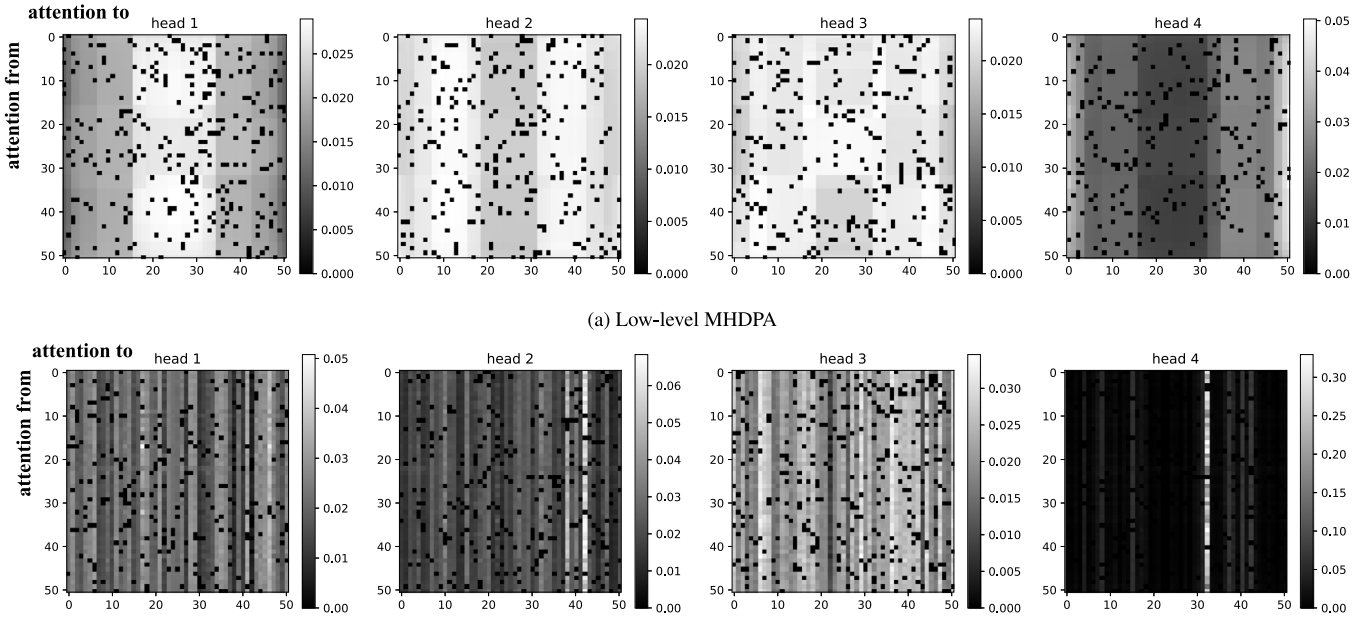
(a) Number of TCN blocks (b) Position encoding

Fig. 7. Performance comparison of **TCN ablation**. (a) Comparing the different number of TCN blocks to verify the importance of the large receptive field. (b) Comparing the ability of TCNs and position encoding (PE) to model positional dependence. The PE experiments were conducted with $d = 10^2$, 10^3, and 10^4.



(a) FetchReach (b) FetchPush

Fig. 8. Performance comparison of **Entropy regularization** in the last-attention block. Adding entropy regularization with an appropriate scale improves the performance, such as with $\alpha = 0.01$. If α is too large (e.g., 0.1 and 1.0), the algorithm struggles to converge, especially in the FetchPush task.



(a) Low-level MHDPA

(b) High-level MHDPA

Fig. 9. Attention visualization in FetchPush. The black dots are caused by dropout. (a) In low-level MHDPA, the features with high attention weights are gathered together in adjacent time steps, which implies that goals with high TD-errors are reasoned from some local features rather than the whole sequence. (b) In high-level MHDPA, the distribution of attention weights is more centralized and the network often focuses on several time steps.

performance. The ratio of original goals is set to 0.2, as in HER, for a fair comparison. The HER goals are used to improve the diversity of the replay goals, and we use the same proportion of AGG goals and HER goals following a coarse search. The learning rates of the actor, critic, and AGG networks are all set to 10^{-3}. We use one CPU core and one GPU to train each task.

Entropy regularization. The entropy regularization of the attention weight is used in the last-attention module to encourage exploration, and also to prevent premature convergence of the AGG network. The attention weight in the last-attention block is a vector $\boldsymbol{w} \in \mathbb{R}^{T+1}$ indicating the weight of the focus on each time step. The entropy of $\boldsymbol{w}$ is calculated by $-\alpha \sum_{i=0}^{T} \boldsymbol{w}_i log \boldsymbol{w}_i$, and this is added to the loss function. A comparison of different α settings in entropy regularization is presented in Fig. 8. The results show that using entropy regularization with an appropriate scale improves the performance. However, when α is too large, the training is hindered and the algorithm struggles to converge. We use $\alpha = 0.01$ in all tasks.

5.4. Attention visualization

Every intermediate result in the computation process of the AGG network can be represented as a real-valued tensor; this tensor is extracted from the episodic memory and indicates the features that are useful for training agents. However, it is widely known that deep neural networks are similar to a black box, and it is difficult to determine the specific meaning of features. Recently, several visualization methods for networks or policies used in RL have been proposed; these methods include t-SNE embeddings [56], saliency maps [57], and Fisher vectors [58]. However, these methods are developed for environments with image-based observations, which usually involve specific objectives that are easy to describe. In our experiment with the robot arm, the state is represented by the angles and velocities of the arm; consequently, the visualization is more difficult in this case than that of an image. An attention-based architecture for the actor–critic network is proposed in [46] to visualize the policy of the agent. We perform attention visualization in our experiment to visualize the MHDPA in the AGG network. The attention weight of the MHDPA in the FetchPush task for training step 500 is visualized in Fig. 9. The black dots in the figure are caused by dropout. Because the AGG network contains two MHDPA blocks, we denote them as "low-level MHDPA" and "high-level MHDPA", respectively. We analyze the attention visualization as follows.

In the low-level MHDPA, the features with high attention weights are gathered together in adjacent time steps, which implies that goals with high TD-errors are reasoned from some local features rather than the whole sequence. Different attention heads usually focus differently. Some heads are more centralized, while others may be distributed. There often exists one attention head (e.g., head 4 in Fig. 9(a)) that focuses on the last step, which verifies that always choosing the last element of an episode as the replay goal still achieves reasonable results in HER. The features in the last time step usually have more comprehensive information than those from other steps.

In high-level MHDPA, the distribution of attention weights is more centralized. The maximum attention weight is 0.66, compared with only 0.05 in low-level MHDPA. This is mainly because, after several TCNs and an MHDPA block, the useful information has gathered in specific time steps. The low- and high-level MHDPAs play different roles in the learning process. The low-level MHDPA tends to learn features from some area, whereas the high-level MHDPA tends to focus on several specific time steps. The attention weights change with the different replay transitions, and also become more centralized as the training process progresses.

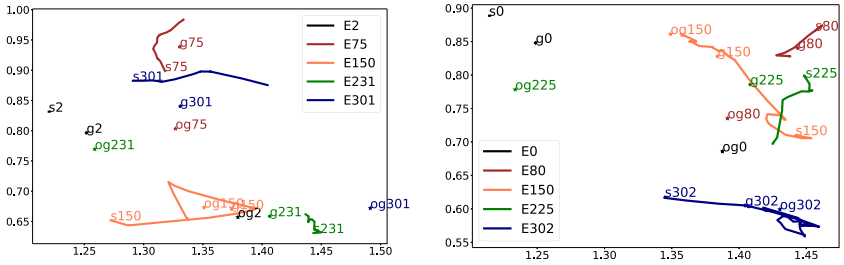
We perform additional experiments to visualize the goals generated by the AGG network, which are shown in Fig. 10. For each figure, we randomly sample 5 episodic trajectories, along with the original goal and the goal generated by the AGG network during training; subsequently, we employ PCA to reduce the dimension to 2, for visualization. In the figures, each color represents an episode. 's·' denotes the start of an episode, 'og·' denotes the original goal, and 'g·' denotes the goal generated by the AGG network. From Fig. 10, it can be observed that the generated goal provides a kind of guidance from the trajectory to the original goal. The generated goals are usually located at a certain distance from the trajectory. The TD-error near the generated goals is high because such goals are not close to the trajectory the agent is familiar with. Furthermore, the generated goals are also not very far from the trajectory. Q-values are inaccurate and close to their initial value in locations that are far from the trajectories; this is because there are no transitions for learning, and TD-errors in these locations also become low. Consequently, the generated goal provides a moderately difficult learning task for the agent to pursue. The goals make it easy for the agent to obtain rewards; furthermore, they generate transitions with high TD-errors.

5.5. Result comparison

We compared the results from the proposed method with those from the following baselines:

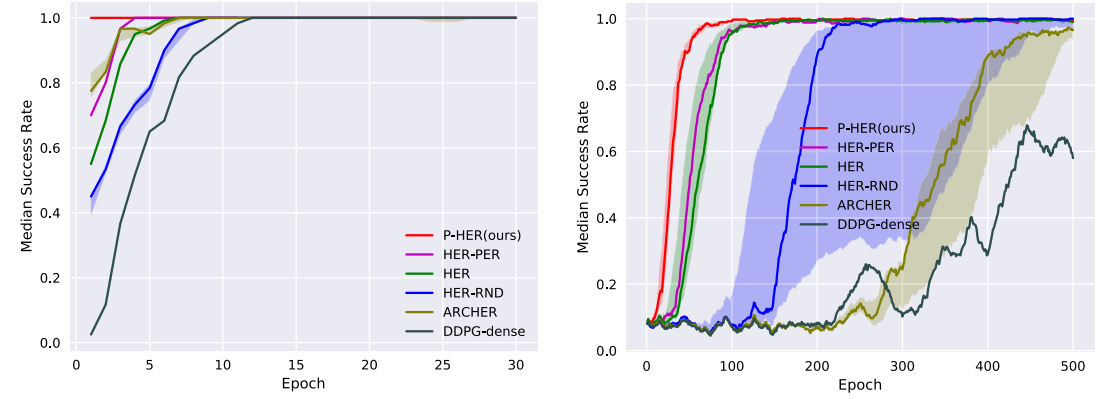
- HER [9]. We use HER implemented by OpenAI baselines [59]. The hyper-parameters are the same as in the original paper.
- HER with prioritized experience replay (HER-PER). We construct a prioritized replay buffer in the HER algorithm as a baseline. The TD-errors of transitions are stored and organized in the 'sum-tree'. Several episodes are sampled according to priority in each training step, and goals are chosen using the HER rule. The hyper-parameter α used to compute the sample probability is set to 0.6, and β, which is used to correct the bias, is set to 0.4 following a coarse search.
- HER with random goals (HER-RND) [15]. The result of RIG [15] (in section 9.4) indicates resampling goals with probability 0.5 from the HER strategy and probability 0.5 uniformly from the goal space can enhance performance, mainly because of the increase in sample diversity and bias elimination. We implement this idea as a baseline.
- HER with aggressive rewards (ARCHER) [17]. The HER algorithm uses the substitute goals $\boldsymbol{g}_{Her}$ to replace the original goal $\boldsymbol{og}$. This process overestimates the probability assigned by the policy to $\boldsymbol{g}_{Her}$ by assuming $\pi(\boldsymbol{s}, \boldsymbol{g}_{Her}) = \pi(\boldsymbol{s}, \boldsymbol{og})$. ARCHER analyzes the bias caused by hindsight goals and performs bias-correction by amplifying the reward for hindsight transitions. We add this bias-correction to HER as a baseline. The hindsight reward and ordinary reward factors are set to λ_r and λ_h, respectively.
- DDPG with shaped rewards (DDPG-dense). We use DDPG with heuristic shaped rewards as a baseline. The shaped reward is the negative distance between the current state and the original goal.

The proposed method and baselines all use a sparse-reward setting, except DDPG-dense, which uses a shaped reward. During training, we evaluate the performance after each epoch by performing 10 test rollouts and compute the average success rate. Each test rollout follows the policy from the actor network without exploration. Each baseline is evaluated several times with different seeds, except HER-PER, which is very computationally expensive. We compare the median test success rate as well as

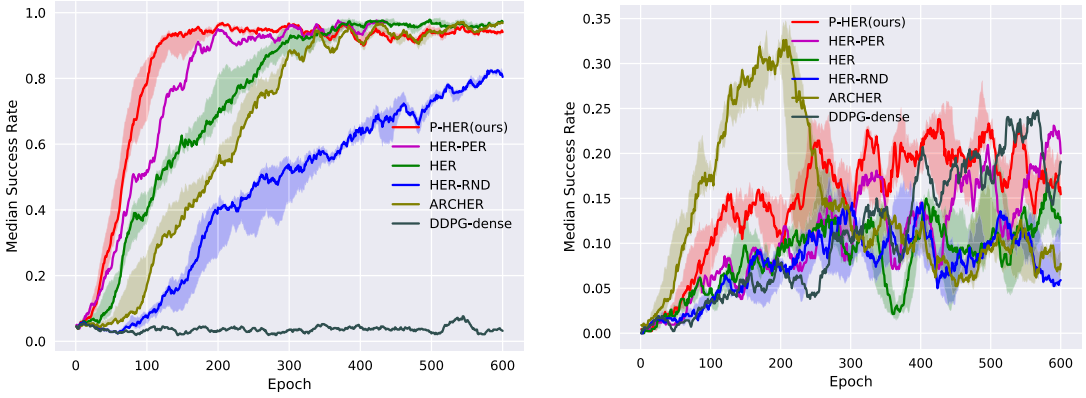


(a) (b)

Fig. 10. Visualization of goals and trajectory in FetchPush.

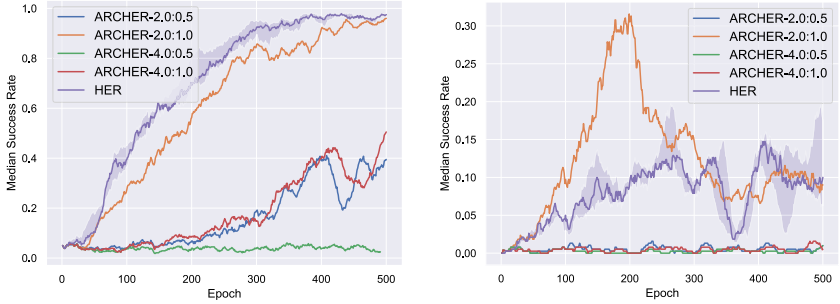


(a) FetchReach (b) FetchPush



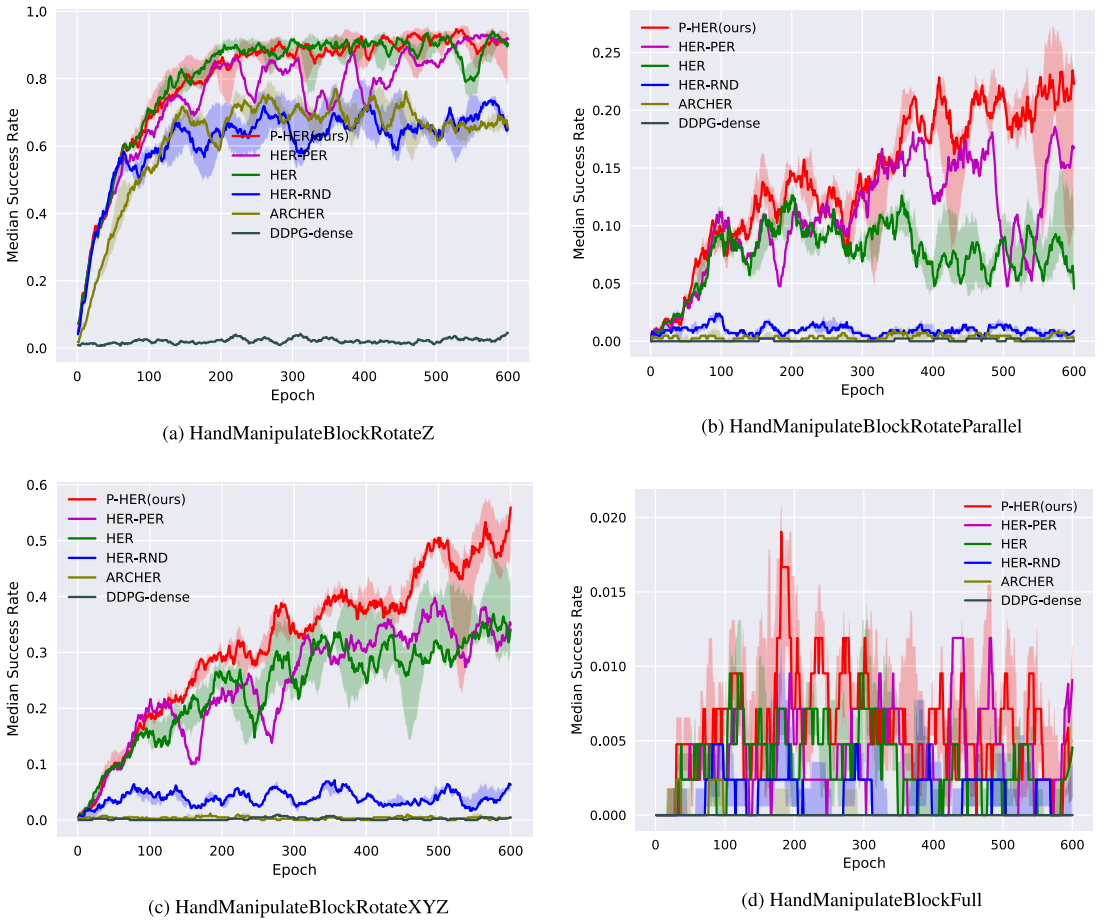
(c) FetchPickAndPlace (d) FetchSlide

Fig. 11. Median test success rate in robot fetch tasks.



(a) FetchPickAndPlace (b) FetchSlide

Fig. 12. Ablation study about ARCHER with different choice of λ_h and λ_r.



(a) HandManipulateBlockRotateZ

(b) HandManipulateBlockRotateParallel

(c) HandManipulateBlockRotateXYZ

(d) HandManipulateBlockFull

Fig. 13. Median test success rate in robot hand block tasks.

the interquartile range. HER-PER requires an average of five days to train each task, which is much longer than the other baselines. All models are trained using the same hardware, which includes an RTX-2080Ti GPU and AMD 2990WX CPU. We record the training time for all environments and all methods for 500 epochs. Moreover, the training time corresponding to each environment is normalized with respect to HER and summarized in Table 2 for comparison.

Table 2 shows that the training times of HER-RND and ARCHER are similar to that of HER. DDPG-dense does not perform goal substitution; thus, it requires less training time. HER-PER is computationally expensive because a prioritized replay buffer is used. In contrast, our method follows a principle similar to that followed by HER-PER but only needs 16.5% more training time in the Fetch environment and 15.3% more training time in the Hand environment on average, compared to those required by HER. Thus, our method does not incur an excessive time consumption.

Fig. 11 shows the median test success rate for all fetch tasks. Our method outperforms all the baselines in the FetchReach, FetchPush, and FetchPickAndPlace tasks by improving the sample efficiency. HER-PER also performs better than HER, which indicates the importance of replaying valuable goals with high TD-errors. However, the HER-PER method is computationally intensive. We find that DDPG-dense produces good results in the FetchSlide task, possibly because the goals lie in a table, and the shaped reward based on distance provides good guidance to the agent. HER-RND [15] typically has a large variance and does not perform well in these tasks. FetchSlide is the most difficult of all the Fetch environments. To decide on a suitable action, the agent must consider the friction of the table and slider as well as the weight. This task is more challenging than other tasks such as reach, push, and pick-place. All baseline models perform poorly in this task. ARCHER [17] performs better than other methods in FetchSlide, but does not perform well in the other three tasks. We also perform an additional ablation study on ARCHER for various λ_h and λ_r, as shown in Fig. 12. In particular, we set (λ_h, λ_r) to (2.0, 0.5), (2.0, 1.0), (4.0, 0.5), and (4.0, 1.0), and the performance is best when $(\lambda_h, \lambda_r) = (2.0, 1.0)$. However, the performance is highly sensitive to the setting of λ_h and λ_r. When λ_h and λ_r are changed, the performance of ARCHER decreases rapidly. Thus, we set $(\lambda_h, \lambda_r) = (2.0, 1.0)$ to achieve the best performance.

Fig. 13 shows the results from the robot hand block tasks. By replaying more valuable goals, the proposed method achieves a significant improvement in performance in the HandManipulateBlockRotateParallel and HandManipulateBlockRotateXYZ tasks. Our method also performs well in the HandManipulateBlockRotateZ task. In the hardest task, HandManipulateBlockFull, none of the methods achieves good results. HER-PER also performs well in these tasks, but the performance is relatively unstable. In all hand block tasks, HER and HER-PER are the strongest baselines. HandManipulateBlockFull is the most difficult of all Hand environments. This is because the agent needs to manipulate the box to achieve the target *rotation* in all axes and also satisfy the target *position* requirement. However, in other tasks, the agent only needs to achieve target rotation in various axes and no target position requirement needs to be met. Thus, all baselines could not achieve a reasonable performance in this task.

The performance of all methods can still be improved by using multiple workers [8]. However, multiple workers can only be employed in simulations. In real-world applications, we usually

Table 2
Wall-clock time normalized by HER (500 epochs).

Environment	Baselines					
	HER	HER-PER	HER-RND	ARCHER	DDPG-dense	Ours
FetchReach	100%	364.8%	75.0%	93.5%	75.0%	121.3%
FetchPush	100%	754.4%	97.3%	102.1%	90.6%	114.0%
FetchPickAndPlace	100%	773.0%	98.4%	106.3%	86.0%	113.6%
FetchSlide	100%	785.7%	97.1%	104.7%	88.0%	117.1%
HandManipulateBlockRotateZ	100%	702.8%	94.8%	99.7%	87.9%	114.7%
HandManipulateBlockRotateParallel	100%	717.2%	96.0%	102.3%	88.2%	113.8%
HandManipulateBlockRotateXYZ	100%	706.2%	95.2%	101.3%	89.6%	115.6%
HandManipulateBlockFull	100%	698.3%	98.0%	99.7%	91.2%	116.9%

have one robot arm to perform tasks. Thus, we believe that it is crucial that the algorithm should achieve reasonable performance using a single worker; therefore, we compared all methods using a single worker. Imitation learning also enables the agent to improve performance by using demonstrations [60]. In the future, we aim to combine the proposed method with distributed training and imitation learning.

The proposed prioritized hindsight model provides the agent with more valuable goals, thus improving the performance and sample efficiency in robot fetch tasks and hand block tasks. The AGG network learns alongside the actor–critic network in an end-to-end manner. Compared with HER, our method only increases the number of parameters slightly and needs about 16% more training time. As a result, our method is computationally efficient and achieves improved performance in multi-goal RL.

6. Discussion

The AGG network uses the achieved-goal sequence of replay episodes as its input. The achieved-goal sequence only contains information about the current episode, which may not be sufficient to reason goals. Thus, we have used temporal convolution and attention mechanism, which have a form of implicit memory, to partly overcome this problem. However, because the RL problem has a large number of experiences and long-time interactions, such methods may still struggle to gather all of the useful information for learning. Several measures can be employed to further solve this problem: (i) following DQN, which uses several adjacent states as input, the AGG network can stack the transitions of several recent episodes as input to obtain more historical information; and (ii) change the network of actor–critic to contain an explicit memory unit, and uses this memory unit as an additional input to the AGG network. The memory unit contains useful information for policy training, and thus provides previous knowledge to the AGG network. There are several kinds of neural networks that contain a dynamic memory unit, such as Differentiable Neural Computer [61] and Relational RNN [43]. We leave these extensions for future research.

The AGG network can be further used to generate goals with other specific properties, not limited to maximizing the TD error. The only requirement is that such a property can be expressed as a differentiable loss function L. The AGG network is updated to follow $\mathbb{E}[\nabla_{\boldsymbol{g}}(L(\cdot, \boldsymbol{g}_t)|_{\boldsymbol{g}_t=G_{\boldsymbol{\phi}}(\boldsymbol{s}_t,M_t)}\nabla_{\boldsymbol{\phi}}G_{\boldsymbol{\phi}}(\boldsymbol{s}_t, M_t)]$. For example, if we set the loss function as $L = [\min(Q-a,0)]^2 + [\max(Q-b,0)]^2$, then $L = 0$ if $Q \in (a, b)$, and $L > 0$ otherwise. In other words, we want the AGG network to generate goals that cause the expected Q value to lie between a and b, which can be regarded as a kind of difficulty control for the goals. For instance, if $(a, b) = (-20, -10)$, then the agent would expect to reach the goal in 10–20 steps in the future. We leave this method as a topic for future studies.

7. Conclusions

In this paper, we have described a prioritized hindsight model for multi-goal RL in a sparse-reward setting. The AGG network enables us to generate goals with high expected TD-errors, and can be trained along with an actor–critic network in an end-to-end manner. TCNs enable comprehensive information and position-dependence to be captured from long episodic sequences. MHDPAs use attention to perform relational reasoning from specific areas and time steps, the last-attention module captures useful information related to the replay transition. Experiments show that our method is computationally efficient and outperforms several baselines in robot fetch and hand tasks, verifying that the proposed AGG network enables valuable goals to be generated for multi-goal RL.

CRediT authorship contribution statement

Peng Liu: Writing - review & editing, Resources, Supervision. **Chenjia Bai:** Writing - original draft, Conceptualization, Methodology, Software. **Yingnan Zhao:** Validation, Methodology, Writing - review & editing. **Chenyao Bai:** Validation, Writing - review & editing. **Wei Zhao:** Supervision, Writing - review & editing. **Xianglong Tang:** Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study is supported by the National Natural Science Foundation of China (61671175), the Sichuan Science and Technology Program, China (2019YFS0069), and the Lab of Space Optoelectronic Measurement & Perception, China (LabSOMP-2018-01).

References

[1] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT press, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M.A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.

[3] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, D. Silver, Rainbow: combining improvements in deep reinforcement learning, in: Proceedings of AAAI Conference on Artificial Intelligence, AAAI, 2018.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L.R. Baker, M. Lai, A. Bolton, Y. Chen, T.P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, Nature 550 (2017) 354–359.

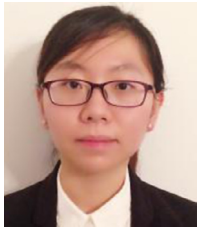
[5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, Science 362 (2018) 1140–1144.
[6] X.B. Peng, P. Abbeel, S. Levine, M. van de Panne, Deepmimic: Example-guided deep reinforcement learning of physics-based character skills, ACM Trans. Graph. 37 (2018) 1–14.
[7] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, D. Quillen, Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection, Int. J. Robot. Res. 37 (4–5) (2018) 421–436.
[8] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, W. Zaremba, Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018, arXiv:arXiv:1802.09464.
[9] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, W. Zaremba, Hindsight experience replay, in: Proceedings of the 30th Advances in Neural Information Processing Systems, NeurIPS, 2017, pp. 5048–5058.
[10] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, in: Proceedings of International Conference on Learning Representations, ICLR, 2016.
[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of Advances in Neural Information Processing Systems, NeurIPS, 2017, pp. 5998–6008.
[12] R.S. Sutton, J. Modayil, M. Delp, T. Degris, P.M. Pilarski, A. White, D. Precup, Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction, in: The 10th International Conference on Autonomous Agents and Multiagent Systems, vol. 2, 2011, pp. 761–768.
[13] T. Schaul, D. Horgan, K. Gregor, D. Silver, Universal value function approximators, in: Proceedings of the 32nd International Conference on Machine Learning, ICML, vol. 37, 2015, pp. 1312–1320.
[14] P. Rauber, F. Mutz, J. Schmidhuber, Hindsight policy gradients, in: Proceedings of International Conference on Learning Representations, ICLR, 2019.
[15] A.V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, S. Levine, Visual reinforcement learning with imagined goals, in: Proceedings of Advances in Neural Information Processing Systems, NeurIPS, 2018, pp. 9191–9200.
[16] D.P. Kingma, M. Welling, Auto-encoding variational bayes, 2013, arXiv preprint arXiv:1312.6114.
[17] S. Lanka, T. Wu, ARCHER: aggressive rewards to counter bias in hindsight experience replay, 2018, arXiv preprint arXiv:1809.02070.
[18] C. Bai, P. Liu, W. Zhao, X. Tang, Guided goal generation for hindsight multi-goal reinforcement learning, Neurocomputing 359 (2019) 353–367.
[19] M. Fang, C. Zhou, B. Shi, B. Gong, J. Xu, T. Zhang, DHER: Hindsight experience replay for dynamic goals, in: Proceedings of International Conference on Learning Representations, ICLR, 2019.
[20] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML, 2009, pp. 41–48.
[21] J. Schmidhuber, Powerplay: training an increasingly general problem solver by continually searching for the simplest still unsolvable problem, Front. Psychol. 4 (2013) 313.
[22] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, P. Abbeel, Reverse curriculum generation for reinforcement learning, in: Proceedings of the 1st Conference on Robot Learning, CoRL, 2017.
[23] C. Florensa, D. Held, X. Geng, P. Abbeel, Automatic goal generation for reinforcement learning agents, in: Proceedings of the 35th International Conference on Machine Learning, ICML, 2018, pp. 1514–1523.
[24] S. Sukhbaatar, I. Kostrikov, A. Szlam, R. Fergus, Intrinsic motivation and automatic curricula via asymmetric self-play, in: Proceedings of the 5th International Conference on Learning Representations, ICLR, 2018.
[25] Y. Burda, H. Edwards, A. Storkey, O. Klimov, Exploration by random network distillation, in: Proceedings of the 5th International Conference on Learning Representations, ICLR, 2019.
[26] R.S. Sutton, D. Precup, S. Singh, Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning, Artif. Intell. 112 (1999) 181–211.
[27] P.-L. Bacon, J. Harb, D. Precup, The option-critic architecture, in: Proceedings of AAAI Conference on Artificial Intelligence, AAAI, 2017, pp. 1726–1734.
[28] T.D. Kulkarni, K. Narasimhan, A. Saeedi, J. Tenenbaum, Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, in: Proceedings of Advances in Neural Information Processing Systems, NeurIPS, 2016, pp. 3675–3683.
[29] K. Marino, A. Gupta, R. Fergus, A. Szlam, Hierarchical RL using an ensemble of proprioceptive periodic policies, in: Proceedings of International Conference on Learning Representations, ICLR, 2019.
[30] O. Nachum, S. Gu, H. Lee, S. Levine, Near-optimal representation learning for hierarchical reinforcement learning, in: Proceedings of International Conference on Learning Representations, ICLR, 2019.
[31] A. Levy, G. Konidaris, R. Platt, K. Saenko, Learning multi-level hierarchies with hindsight, in: Proceedings of International Conference on Learning Representations, ICLR, 2019.
[32] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A.W. Senior, K. Kavukcuoglu, Wavenet: a generative model for raw audio, SSW 125 (2016).
[33] J. Gehring, M. Auli, D. Grangier, D. Yarats, Y.N. Dauphin, Convolutional sequence to sequence learning, in: Proceedings of International Conference on Machine Learning, ICML, 2017, pp. 1243–1252.
[34] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, in: Proceedings of International Conference on Learning Representations, ICLR, 2016.
[35] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018, arXiv preprint arXiv:1803.01271.
[36] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.
[37] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Proceedings of Empirical Methods in Natural Language Processing, EMNLP, 2014.
[38] Y. Duan, M. Andrychowicz, B. Stadie, O.J. Ho, J. Schneider, I. Sutskever, P. Abbeel, W. Zaremba, One-shot imitation learning, in: Proceedings of Advances in Neural Information Processing Systems, NeurIPS, 2017, pp. 1087–1098.
[39] N. Mishra, M. Rohaninejad, X. Chen, P. Abbeel, A simple neural attentive meta-learner, in: Proceedings of International Conference on Learning Representations, ICLR, 2018.
[40] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: Proceedings of International Conference on Learning Representations, ICLR, 2015.
[41] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, Y. Bengio, Show, attend and tell: Neural image caption generation with visual attention, in: Proceedings of the 32nd International Conference on Machine Learning, ICML, 2015, pp. 2048–2057.
[42] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL, 2019.
[43] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, T. Lillicrap, Relational recurrent neural networks, in: Proceedings of Advances in Neural Information Processing Systems, NeurIPS, 2018, pp. 7299–7310.
[44] J. Choi, Y. Guo, M. Moczulski, J. Oh, N. Wu, M. Norouzi, H. Lee, Contingency-aware exploration in reinforcement learning, in: Proceedings of International Conference on Learning Representations, ICLR, 2019.
[45] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, et al., Deep reinforcement learning with relational inductive biases, in: Proceedings of International Conference on Learning Representations, ICLR, 2019.
[46] A. Mott, D. Zoran, M. Chrzanowski, D. Wierstra, D.J. Rezende, Towards interpretable reinforcement learning using attention augmented agents, in: Advances in Neural Information Processing Systems, NeurIPS, 2019, pp. 12329–12338.
[47] R.S. Sutton, D.A. McAllester, S.P. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: Advances in Neural Information Processing Systems, NeurIPS, 2000, pp. 1057–1063.
[48] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, in: Proceedings of the 4th International Conference on Learning Representations, ICLR, 2016.
[49] Y. Yuan, Z.L. Yu, Z. Gu, Y. Yeboah, W. Wei, X. Deng, J. Li, Y. Li, A novel multi-step Q-learning method to improve data efficiency for deep reinforcement learning, Knowl.-Based Syst. 175 (2019) 107–117.
[50] P. Liu, Y. Zhao, W. Zhao, X. Tang, Z. Yang, An exploratory rollout policy for imagination-augmented agents, Appl. Intell. (2019) 1–16.
[51] Y. Zhao, P. Liu, C. Bai, W. Zhao, X. Tang, Obtaining accurate estimated action values in categorical distributional reinforcement learning, Knowl.-Based Syst. (2020) 105511.
[52] A.Y. Ng, D. Harada, S. Russell, Policy invariance under reward transformations: theory and application to reward shaping, in: Proceedings of the 32nd International Conference on Machine Learning, ICML, 1999, pp. 278–287.
[53] J. Ba, R. Kiros, G.E. Hinton, Layer normalization, 2016, arXiv:arXiv:1607.06450.
[54] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 770–778.
[55] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, 2016, arXiv:arXiv:1606.01540.

[56] T. Zahavy, N. Ben-Zrihem, S. Mannor, Graying the black box: understanding dqns, in: International Conference on Machine Learning, ICML, 2016, pp. 1899–1908.
[57] S. Greydanus, A. Koul, J. Dodge, A. Fern, Visualizing and understanding atari agents, in: International Conference on Machine Learning, 2018, pp. 1792–1801.
[58] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, K.-R. Müller, Unmasking clever hans predictors and assessing what machines really learn, Nat. Commun. 10 (1) (2019) 1–8.
[59] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, P. Zhokhov, Openai baselines, 2017, https://github.com/openai/baselines.
[60] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, P. Abbeel, Overcoming exploration in reinforcement learning with demonstrations, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 6292–6299.
[61] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S.G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al., Hybrid computing using a neural network with dynamic external memory, Nature 538 (7626) (2016) 471.

Peng Liu is an associate professor at the School of Computer Science and Technology, Harbin Institute of Technology. He received his Ph.D. in microelectronics and solid-state electronics from Harbin Institute of Technology in 2007. His research interests cover image processing, video analysis, pattern recognition, and the design of large-scale integrated circuits.

Chenjia Bai received B.S. and M.S. degrees from Harbin Institute of Technology, Harbin, China, in 2015 and 2017, respectively. He is currently working toward a Ph.D. at the Pattern Recognition and Intelligent System Research Center, Harbin Institute of Technology. His main research interests include reinforcement learning and neural networks.

Yingnan Zhao received B.S. and M.S. degrees from Harbin Institute of Technology, Harbin, China, in 2015 and 2017, respectively. He is currently working toward a Ph.D. in Harbin Institute of Technology. His main research interests include reinforcement learning and deep learning.



Chenyao Bai is a lecturer in Shanghai Customs College. She received her M.Sc. degree in Electronic Systems with Sensor Technology from University of Warwick, UK, in 2012, and her Ph.D. study in School of Engineering at University of Warwick in 2016. Her research interests include machine learning, molecular communication, quorum sensing, nanonetworks, mathematical modeling, error correction and detection codes and communication protocols.

Wei Zhao is an associate professor at the School of Computer Science and Technology, Harbin Institute of Technology. She won a First Prize of Heilongjiang Province Science and Technology Progress. Her research fields include pattern recognition, machine learning, and computer vision.

Xianglong Tang is a profess or at the School of Computer Science and Technology, Harbin Institute of Technology. He received his Ph.D. in computer application technology from Harbin Institute of Technology in 1995. His research interest covers pattern recognition, image processing, and machine learning.