A Novel Topology Adaptation Strategy for Dynamic Sparse Training in Deep Reinforcement Learning

Meng Xu, Xinhong Chen, and Jianping Wang, *Fellow, IEEE*

***Abstract*— Deep reinforcement learning (DRL) has been widely adopted in various applications, yet it faces practical limitations due to high storage and computational demands. Dynamic sparse training (DST) has recently emerged as a prominent approach to reduce these demands during training and inference phases, but existing DST methods achieve high sparsity levels by sacrificing policy performance as they rely on the absolute magnitude of connections for pruning and randomly generating connections. Addressing this, our study presents a generic method that can be seamlessly integrated into existing DST methods in DRL to enhance their policy performance while preserving their sparsity levels. Specifically, we develop a novel method for calculating the importance of connections within the model. Subsequently, we dynamically adjust the sparse network topology by dropping existing connections and introducing new connections based on their respective importance values. Through validation on eight widely used simulation tasks, our method improves two state-of-the-art (SOTA) DST approaches by up to 70% in episode return and average return across all episodes under various sparsity levels.**

***Index Terms*— Deep reinforcement learning (DRL), dynamic sparse training (DST), topology adaptation strategy.**

NOMENCLATURE

r	Current reward.
s	Current state.
a	Current action.
s'	Next state.
a'	Next action.
$\hat{a}$	Predicted action.
N	Size of the mini-batch.
α	Learning rate.
α'	Temperature parameter in SAC.
θ^{μ}	Actor.
θ	Critic.
γ	Discount.
$Q(s,\cdot)$	Q-value.
$\hat{Q}(s,\cdot)$	Target Q-value.
τ	Coefficient of soft update.
ϵ	Noise in exploration.
$\pi(\cdot;\theta^{\mu})$	Action policy.
$\mathcal{L}(\theta)$	Loss function.
$\mathcal{D}$	Replay buffer.
$\mathcal{D}_0$	Mini-batch.
$\boldsymbol{M}$	Binary mask.
λ^i	Coefficient setting the sparsity level in layer i.
c^i	Number of neurons in layer i.
$\mathcal{S}$	Importance of each connection or neuron.

Manuscript received 28 February 2024; revised 20 May 2024; accepted 16 August 2024. This work was supported in part by Hong Kong Research Grant Council under General Research Fund (GRF) under Grant 11218621 and in part by the Research Impact Fund (RIF) under Project R5060-19. *(Corresponding author: Jianping Wang.)*

The authors are with the Department of Computer Science, City University of Hong Kong, Hong Kong, SAR, China (e-mail: mxu247-c@my.cityu.edu.hk; xinhong.chen@cityu.edu.hk; jianwang@cityu.edu.hk).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNNLS.2024.3446996, provided by the authors.

Digital Object Identifier 10.1109/TNNLS.2024.3446996

I. INTRODUCTION

DEEP reinforcement learning (DRL) has demonstrated remarkable achievements in cutting-edge fields, such as robotics [1] and autonomous driving [2]. Nevertheless, when DRL models are deployed in real-world applications, such as large-scale StarCraft competitions, they often consume significant computational and storage resources, which limits their applicability in devices without advanced hardware. Sparse models, initially achieved through pruning techniques derived from deep supervised learning, are an effective approach to reducing storage requirements [3]. Although efforts have been made to apply these pruning techniques in DRL for model compression and training acceleration, they still require iterative training of dense networks, leading to prohibitively high costs.

Recently, dynamic sparse training (DST) has emerged as the primary approach for implementing sparse models, effectively reducing the number of model parameters and computational resources required during both training and inference. It involves initializing a sparse network structure and dynamically adjusting the sparse topology throughout the training process to achieve a refined sparse network. Several state-of-the-art (SOTA) studies [4], [5], [6] have explored DST in the context of DRL, offering a highly promising approach for sparse DRL. Significantly, the process of dropping and growing connections plays a crucial role in DST, allowing for dynamic adaptation of the network topology. This feature sets DST apart from static sparse training (SST). While DST effectively reduces the model size of DRL, it typically introduces a certain degree of performance trade-off at different levels of sparsity. This trade-off becomes particularly pronounced when the model's sparsity reaches or exceeds 90%. In other words, while achieving 90% sparsity can reduce 90% of the connections in the model, it often introduces a substantial performance decrease compared to the original dense model. This significantly impedes the practicality of DRL methods

that integrate DST. For instance, in DRL-based robot 2-D navigation, when the sparsity level reaches 95%, existing DST methods can lead to a policy performance degradation of up to 90% compared to the dense model. This substantial decline results in a poor robot motion planning strategy with frequent collisions, significantly reducing the efficiency of task completion.

This study aims to develop a generic method that can be integrated into existing DST methods in DRL to reduce the performance gap between sparse and dense models while preserving high sparsity levels. To achieve this goal, we need to address the following challenge. Existing DST methods in the DRL community first identify the connections to drop by evaluating the absolute value of each connection during the connection drop stage and then randomly add new connections during the connection growth stage. However, relying solely on absolute values as the exclusive criterion for removing connections is unjustifiable. This is because: 1) the importance of a connection with the same absolute value varies across different stages of training [7], [8]; and 2) considering the interconnectedness of connections within the network, evaluating the importance of a connection should also take into account its neighboring connections. Additionally, randomly growing connections at arbitrary locations is suboptimal due to the inherent randomness involved. This randomness can lead to the creation of connections that do not significantly contribute to improving network performance.

To tackle this challenge, we develop a novel topology adaptation strategy specifically for the critic. Moreover, we have ensured consistency between the actor's strategy and the baseline DST method, as corroborated by prior studies [9], [10], consistently demonstrating that prioritizing improvements in the critic alone within DRL can yield superior performance. During the connection drop stage, we begin by assessing the importance of each layer of neurons, recognizing their crucial role in processing information and influencing the network's output. This evaluation takes into account both the temporal difference (TD) error and the sum of absolute values of the neuron's forward or subsequent connections. Consequently, we determine the importance of each connection by considering the importance of the connected neuron and the absolute value of the connection. In our evaluation of connection importance, the TD error serves as an indicator of variations observed during different stages of learning. On the other hand, the sum of absolute values of a neuron's connections can, to some extent, represent the importance of the neighboring connections associated with a specific connection. Finally, we selectively drop connections that are considered less important. Similarly, during the connection growth stage, we evaluate the importance of each location where a connection is missing by considering the importance of its preceding and following neurons. We then selectively add connections to locations with high importance.

The contributions of this article are summarized as follows.

1) We develop a generic approach to enhance existing DST methods in the DRL community, which results in higher returns while maintaining the same level of sparsity. This approach effectively helps bridge the performance gap between dense models and sparse models.
2) We develop a novel topology adaptation strategy specifically for critic networks. This strategy incorporates a new method for calculating the importance of connections within the network and accordingly dropping and growing connections based on their importance values.
3) We validate the effectiveness of our method by enhancing two SOTA DST approaches across eight diverse simulation tasks. The experimental results demonstrate that, at the highest level of sparsity (e.g., 95% sparsity), existing DST methods exhibit a certain degree of performance degradation, with up to a 90% reduction in return compared to regular dense models. In contrast, our method effectively improves existing DST methods, with improvement reaching up to 70% in terms of return. This achievement enables our method to achieve performance that is close to or even surpasses that of dense models.

This study is organized as follows. In Section II, a review of related research is provided. Section III describes the background. Section IV presents the proposed method with detailed explanations. The experimental results are demonstrated in Section V. Finally, Section VI concludes the study.

II. Related Work

This section begins with a concise review of relevant literature on DRL. Subsequently, we will delve into the DST technique and its current development status within the DRL community. Furthermore, we will discuss the limitations of the existing research on DST in the context of DRL.

A. Deep Reinforcement Learning

The main objective of reinforcement learning (RL) is to determine an optimal policy that maximizes the cumulative discounted rewards over time [2], [11]. Traditionally, tabular RL has been commonly used to tackle various learning tasks. However, this approach becomes less practical in environments with large state spaces. In 2015, Minh et al. made a significant breakthrough in RL by introducing deep neural networks to handle high-dimensional observations [12]. This advancement paved the way for DRL, which has gained widespread popularity and found diverse applications across multiple domains. Among these domains, one of the most extensively explored areas is learning-based control. Representative studies in this field include Hamiltonian-driven adaptive dynamic programming [13], cooperative finitely excited learning [14], and model-free policy iteration [15].

Despite the application of DRL in various domains, it often encounters challenges related to storage and computational requirements, especially when dealing with complex learning tasks that involve training large-scale models. These limitations significantly restrict the wider adoption of DRL.

B. DST in Offline Deep Learning

In the conventional deep learning community, DST has already been applied to numerous offline deep learning tasks.

DST methods distinguish themselves from traditional pruning approaches by incorporating sparse connectivity learning throughout the entire training process. Instead of conducting weight pruning after training or during initialization, DST methods dynamically adjust the network's connectivity by adding and removing weights based on saliency criteria. Notable saliency criteria in this context include sparse evolutionary training (SET), RigL, and its variants. For instance, SET removes weights with the smallest magnitude and introduces random weight additions. On the other hand, RigL prunes weights with the smallest magnitude while regrowing weights with significant gradient magnitudes. These techniques collectively enable the exploration of a broader parameter space and contribute to overall performance improvements. Additionally, Liu et al. [16] extended the original RigL approach by introducing modifications to the sparse connectivity update schedule and drop rate, leading to enhanced outcomes compared to the initial RigL method.

Notable recent studies in this area include Powers et al. [17], Wołczyk et al. [18], Yildirim et al. [19], Ren and Honavar [20], and Wang et al. [21]. Recently, Yin et al. [22] made significant advancements by identifying sparse amenable channels in DST algorithms. Their method, Chase, achieves SOTA generalization performance by incorporating a soft memory constraint similar to Yuan et al. [23] and calculating parameter saliency based on global statistics. Additionally, various unstructured DST methods, including DeepR [24], SNFS [25], and MEST [23], have also been proposed. Currently, DST has also been widely applied in tasks such as federated learning [26], graph neural networks [27], image processing [28], [29], [30], and language models [31].

However, the research on sparse training in the deep learning community primarily focuses on developing sparse training methods for offline deep learning models. These methods are not directly applicable to DRL models due to the interactive and online learning nature of DRL. Consequently, the DRL community has developed sparse training methods specifically tailored for DRL, such as Sokar et al. [4], Tan et al. [6], and Grooten et al. [5].

C. DST in DRL

The most promising solution currently for achieving lightweight DRL is sparse neural networks because this approach can effectively reduce storage requirements by training a model with sparse connections. Previous studies have explored the integration of sparsity into DRL through pruning techniques. Livne and Cohen [3] employed a dense teacher network for iterative pruning and retraining of a student policy network via knowledge distillation. Lee et al. [32] proposed an algorithm combining connection grouping and pruning during DRL training. Vischer et al. [33] investigated the lottery ticket hypothesis in DRL, demonstrating the superiority of sparse subnetworks over dense networks when trained from scratch, along with the work by Zhao et al. [34]. However, the process of pruning dense networks typically involves pretraining a dense model and incurs additional computational costs due to the iterative cycles of pruning and retraining the dense model.

Recent research in the DRL community has sparked interest in DST for training sparse DRL models, which has emerged as the current best approach for achieving sparse DRL models, primarily due to the significant advantages that sparse training has demonstrated in the field of deep learning compared to pruning techniques. Compared to sparse training methods for offline deep learning, there are relatively fewer solutions tailored specifically for DRL. The most recent studies in this area include Tan et al. [6], who have demonstrated the ability to train a DRL model entirely on sparse networks using sparse topology evolution. Sokar et al. [4] have dynamically adapted the topology of a sparse network to the changing data distribution during training. Furthermore, Grooten et al. [5] have applied DST to filter out environmental noise for DRL.

While DST methods have proven to be effective in reducing the size of DRL models, it is important to acknowledge that high sparsity levels can potentially have a detrimental impact on policy performance. Particularly when the sparsity exceeds 90%, there is often a noticeable degradation in performance compared to the original dense models, resulting in deviations from the desired outcome.

III. Preliminaries

In this section, we present the key concepts and implementation steps of DST within the context of DRL. Furthermore, we illustrate the problem formulation to provide the engineering background of our approach and clarify the objective we aim to achieve. Nomenclature provides a comprehensive list of the key notations used throughout the article. Furthermore, within our method, we adopt twin delayed deep deterministic policy gradient (TD3) and soft actor–critic (SAC) as the underlying DRL algorithms. These algorithms have been widely utilized as the foundation for various advanced DRL technologies, including lifelong DRL [35], sparse DRL [4], [6], and evolutionary DRL [36]. The descriptions of TD3 and SAC can be found in the supplementary materials. Indeed, in addition to the SAC and TD3 algorithms, a wide range of off-policy DRL algorithms can potentially be utilized as the underlying algorithms for both our method and existing DST methods.

A. Basic Concepts of DST

In this section, we introduce three fundamental concepts that hold paramount importance in DST within the domains of deep learning and DRL: layer density, model density, and model sparsity.

Let us consider a neural network denoted as f_θ, where θ corresponds to the network's parameters. The symbol θ^l represents the parameters associated with the lth layer. This particular network undergoes training using a dataset $\mathcal{D}$ with the primary objective of minimizing the loss function $\mathcal{L}(\theta; \mathcal{D})$.

1) Layer Density: The density d^l of layer l is calculated by dividing the L_0 norm of θ^l (which represents the number of nonzero entries) by the latent dimension n^l (which represents the total number of connection positions). Mathematically, this can be expressed as $d^l = \|\theta^l\|_0 / n^l$.

2) Model Density: The model density, denoted as Des, is determined by summing the densities of all layers and normalizing the result by the total number of dimensions. Specifically, we calculate Des $= (\sum_{l=1}^{L} d^l n^l)/(\sum_{l=1}^{L} n^l)$, where L denotes the total number of layers in the network.

3) Model Sparsity: The sparsity of the model is defined as the complement of the density, i.e., $S = 1 - \text{Des}$.

B. General Steps of DST

We outline the general steps for implementing established DST approaches [4], [5], [6] within DRL.

1) Initialization of Sparse Topology: Begin by creating the initial sparse topology from scratch using methods like the Erdös-Rényi random graph. Set the network's sparsity and prune specific connections to zero. This initial sparse structure serves as the starting point for training.

2) Training of Current Networks: Follow the general training procedure of DRL, adapting it to the specific training approach of the chosen DRL algorithm, such as TD3.

3) Update of Target Networks: Employ target networks with delayed updating, as commonly used in off-policy DRL algorithms like TD3. For DRL algorithms that do not have a target network, this step can be omitted.

4) Dynamic Adaptation of Sparse Topology: The dynamic adaptation of the sparse topology is a critical step in DST, distinguishing it from SST methods. Unlike SST, which lacks dynamic topology adaptation, DST dynamically adjusts the sparse topology in the current actor and critics. This involves selectively dropping and growing connections within the network using a topology adaptation strategy.

Next, we present the SET method, which serves as the topology adaptation strategy employed by current DST methods [4], [5], [6] in DRL. SET involves a two-step "remove-and-add" cycle, as shown below.

First, we initiate the process by sorting all connections according to their absolute values in descending order. Connections with smaller absolute values are considered less important. Our goal is to remove 10% of the connections in the current model, specifically those identified as the least important. Let n_p denote the number of connections with the smallest positive values, and n_q denote the number of connections with the largest negative values. The sum of n_p and n_q is set to 10% of the total number of connections in the current model. Next, we identify the n_q-th largest negative connection, denoted as $\tilde{\theta}'_q$, and the n_p-th smallest positive connection, denoted as $\tilde{\theta}'_p$. For each connection θ^j, it will be removed if $(0 < \theta^j < \tilde{\theta}'_p) \vee (0 > \theta^j > \tilde{\theta}'_q)$, where $\vee$ represents the logical OR operator.

Second, we randomly introduce additional connections to positions where no connections currently exist in each layer. The total number of connections added in this process is $n_p + n_q$, and these newly added connections are initialized to zero. The procedure for adding connections to each layer is the same, which is outlined below. For layer i, we generate a random integer x from a discrete uniform distribution in the interval $[1, c^{(i-1)} \times c^i]$, where c^{i-1} represents the number of neurons in layer $i-1$, and c^i represents the number of neurons in layer i. For each connection θ^j, it will be added if $(\theta^j == 0) \wedge (j == x)$, where $\wedge$ represents the logical AND operator.

The aforementioned analysis highlights a limitation of the SET method, as it relies solely on the absolute value of connections to determine their retention or removal. However, the absolute value alone cannot fully capture the importance of a connection [7], [8]. Furthermore, the SET method introduces new connections at random positions, which is suboptimal. This limitation leads to a noticeable degradation in performance when the model has high sparsity, as observed in the existing DRL community's DST methods.

C. Problem Formulation

The objective of sparse training in DRL is fundamentally the same as DRL itself, which is to learn an optimal policy $\pi(s;\phi)$ that maximizes the cumulative long-term reward

$$J = \max_{\phi} \mathbb{E}_{\pi(\phi)}\left[\sum_{i=0}^{T} \gamma^{i-t} r_i \middle| s_0, a_0\right]. \tag{1}$$

Here, ϕ represents the policy in DRL. This objective is pursued within the framework of a Markov decision process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, r, \mathbb{P}, \gamma \rangle$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ represents the action space, r is the reward function, $\mathbb{P}$ is the transition matrix, and γ is the discount factor. During each time step t, the agent operates within the environment by observing the current state $s_t \in \mathcal{S}$ and selecting an action $a_t \in \mathcal{A}$ based on its policy $\pi : \mathcal{S} \to \mathcal{A}$. The action taken by the agent leads to a corresponding reward r_t.

For regular DRL, dense models (e.g., dense actor–critic architectures) are usually employed to learn a policy that achieves the aforementioned objective. In contrast, sparse training in DRL utilizes sparse models (e.g., sparse actor–critic architectures) to learn a policy that achieves the same objective. However, when the sparsity level is high (e.g., removing 95% of model parameters), existing sparse training methods can only learn a suboptimal policy. As a result, the cumulative long-term reward J is noticeably lower compared to what a dense model would achieve, which leads to performance degradation. Our goal is to learn a better policy and increase the cumulative long-term reward J while maintaining the same sparsity level as the existing sparse training methods.

IV. Methodology

In this section, we first introduce the framework of our method. Then, we present our approach for calculating connection importance, the topology adaptation strategy, and the complexity analysis. Finally, we outline the overall training workflow to demonstrate how our approach seamlessly integrates into existing DST methods, thereby enhancing their performance.

A. Framework for the Proposed Method

The framework of our method is illustrated in Fig. 1. Our approach involves several steps. First, we sample a mini-batch of data from the replay buffer. Then, we compute the current Q-value and target Q-value using the current

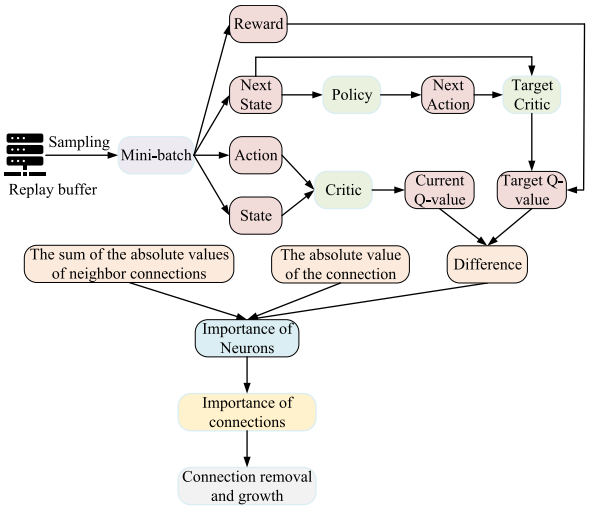


Fig. 1. Framework of our approach. In the figure, "policy" refers to the "actor."

critic and target critic networks, respectively. Subsequently, we calculate the discrepancy between the current Q-value and the target Q-value. This discrepancy, along with the absolute values of the connection itself and its neighboring connections, is aggregated to determine the importance of the neuron. Based on the importance of neurons, we further calculate the importance of connections. Finally, we perform connection deletion and growth based on the importance of connections, resulting in the topology adaptation of sparse networks.

B. Calculating the Importance of Connections

Let $(\boldsymbol{\theta}^{\boldsymbol{\mu}}, \boldsymbol{\theta})$ represent the network parameters for DRL. In this case, $\boldsymbol{\theta}^{\boldsymbol{\mu}}$ corresponds to the actor, and $\boldsymbol{\theta}$ corresponds to the critic. Let $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s}')$ represent a transition. The current Q-value is denoted as $Q(\boldsymbol{s}, \boldsymbol{a})$, while the target Q-value is represented as $\hat{Q}(\boldsymbol{s}, \boldsymbol{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma Q(\mathbf{s}', \mathbf{a}')$. Let $\mathcal{L}(Q(\boldsymbol{s}, \boldsymbol{a}), \hat{Q}(\boldsymbol{s}, \boldsymbol{a}))$ be the TD error for DRL. Through the process of learning, DRL gradually minimizes the TD error.

As mentioned earlier, our efforts focus specifically on the critic while maintaining consistency in the actor's topological adaptation strategy with the baseline DST. Below, we outline the specific steps involved in computing connection importance.

1) Calculating the Discrepancy Between the Current Q-Value and the Expected Value: We begin by examining the impact of change in the Q-value $Q(\boldsymbol{s}, \boldsymbol{a})$ on the TD error $\mathcal{L}(Q(\boldsymbol{s}, \boldsymbol{a}), \hat{Q}(\boldsymbol{s}, \boldsymbol{a}))$. As we know, in off-policy DRL, the collected samples during the training process are stored in a replay buffer, and a mini-batch of samples is taken from this replay buffer to calculate the loss and update the network. Let N be the number of samples in the mini-batch. The alteration in TD error that arises from a small perturbation $\boldsymbol{\delta} = \{\delta_1, \delta_2, \ldots, \delta_N\}$ in the output $\boldsymbol{Q} = \{Q(\boldsymbol{s_1}, \boldsymbol{a_1}), Q(\boldsymbol{s_2}, \boldsymbol{a_2}), \ldots, Q(\boldsymbol{s_N}, \boldsymbol{a_N})\}$ for the N transitions can be approximated using a first-order Taylor expansion, as follows:

$$\mathcal{L}(\boldsymbol{Q} + \boldsymbol{\delta}) - \mathcal{L}(\boldsymbol{Q}) \approx \sum_{i=1}^{N} \frac{\partial \mathcal{L}_i\big(Q(\mathbf{s}_i, \mathbf{a}_i), \hat{Q}(\mathbf{s}_i, \mathbf{a}_i)\big)}{\partial Q(\mathbf{s}_i, \mathbf{a}_i)} \delta_i. \quad (2)$$

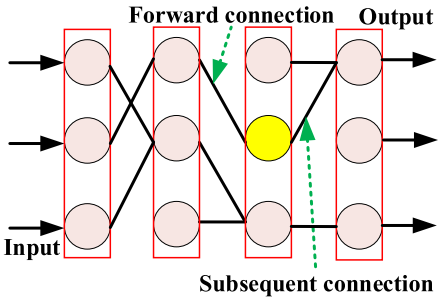


Fig. 2. Forward and subsequent connections of neurons. This figure employs a yellow neuron as an example to illustrate the forward connections and subsequent connections.

Because $\boldsymbol{\delta}$ represents a very small perturbation, we can assess the disparity between $\mathcal{L}(\boldsymbol{Q} + \boldsymbol{\delta})$ and $\mathcal{L}(\boldsymbol{Q})$ by calculating $\sum_{i=1}^{N} |(\partial \mathcal{L}_i(Q(\mathbf{s}_i, \mathbf{a}_i), \hat{Q}(\mathbf{s}_i, \mathbf{a}_i)))/(\partial Q(\mathbf{s}_i, \mathbf{a}_i))|$. Here, $|(\partial \mathcal{L}_i(Q(\mathbf{s}_i, \mathbf{a}_i), \hat{Q}(\mathbf{s}_i, \mathbf{a}_i)))/(\partial Q(\mathbf{s}_i, \mathbf{a}_i))|$ is the absolute value of the gradient of the TD error $\mathcal{L}_i(Q(\mathbf{s}_i, \mathbf{a}_i), \hat{Q}(\mathbf{s}_i, \mathbf{a}_i))$ with respect to the output $Q(\mathbf{s}_i, \mathbf{a}_i)$, given by

$$\left| \frac{\partial \mathcal{L}_i\big(Q(\mathbf{s}_i, \mathbf{a}_i), \hat{Q}(\mathbf{s}_i, \mathbf{a}_i)\big)}{\partial Q(\mathbf{s}_i, \mathbf{a}_i)} \right| \equiv \left| r(\mathbf{s}_i, \mathbf{a}_i) + \gamma Q\big(\mathbf{s}_i', \mathbf{a}_i'\big) - Q(\mathbf{s}_i, \mathbf{a}_i) \right|. \quad (3)$$

Therefore, according to (3), we evaluate the disparity between $\mathcal{L}(\boldsymbol{Q} + \boldsymbol{\delta})$ and $\mathcal{L}(\boldsymbol{Q})$ by utilizing the difference between the current Q-value $Q(\mathbf{s}_i, \mathbf{a}_i)$ and the target value $r(\mathbf{s}_i, \mathbf{a}_i) + \gamma Q(\mathbf{s}_i', \mathbf{a}_i')$. In the context of DRL, modifications in Q-values are a direct result of changes in the connections and neurons within the critic network. In other words, the connections and neurons in the critic network are directly associated with the variations in TD error. Therefore, it is essential to consider the magnitude of $|(\partial \mathcal{L}_i(Q(\mathbf{s}_i, \mathbf{a}_i), \hat{Q}(\mathbf{s}_i, \mathbf{a}_i)))/(\partial Q(\mathbf{s}_i, \mathbf{a}_i))|$ when assessing the importance of neurons and connections.

2) Calculating the Importance of Neurons in Each Layer: At this step, we calculate the importance of each neuron in each layer. When assessing the importance of neurons, we consider both $|(\partial \mathcal{L}_i(Q(\mathbf{s}_i, \mathbf{a}_i), \hat{Q}(\mathbf{s}_i, \mathbf{a}_i)))/(\partial Q(\mathbf{s}_i, \mathbf{a}_i))|$ and the magnitudes of their subsequent or forward connections. This consideration is necessary because the magnitude of these connections plays a direct role in transmitting information from these neurons or directly influencing the input to these neurons. The forward and subsequent connections of neurons in the network are depicted in Fig. 2.

We first compute the importance $\mathcal{S}_{O_i}$ of ith neuron O_i in the output layer as follows:

$$\mathcal{S}_{O_i}^{(t)} = \mathcal{S}_{O_i}^{(t-1)} + \frac{1}{2}\left(\sum_{i=1}^{N} \left| \frac{\partial \mathcal{L}_i\big(Q(\mathbf{s}_i, \mathbf{a}_i), \hat{Q}(\mathbf{s}_i, \mathbf{a}_i)\big)}{\partial Q(\mathbf{s}_i, \mathbf{a}_i)} \right| + \sum_{j=1}^{h} |\boldsymbol{\omega}_{ji}| \right). \quad (4)$$

Here, t is the current step. $|\boldsymbol{\omega}_{ji}|$ denotes the magnitude of the forward connection from neuron j to neuron i. h denotes the number of the forward connections for neuron O_i.

Next, we proceed to compute the importance of each neuron in the input layer. The importance $\mathcal{S}_{I_i}$ of the neuron I_i in the input layer is calculated in a similar manner as that of the

output layer, as demonstrated below

$$\mathcal{S}_{I_i}^{(t)} = \mathcal{S}_{I_i}^{(t-1)} + \frac{1}{2}\left(\sum_{i=1}^{N}\left|\frac{\partial \mathcal{L}_i\big(Q(\mathbf{s}_i,\mathbf{a}_i),\hat{Q}(\mathbf{s}_i,\mathbf{a}_i)\big)}{\partial Q(\mathbf{s}_i,\mathbf{a}_i)}\right| + \sum_{j=1}^{h}\left|\boldsymbol{\omega}_{ij}\right|\right) \quad (5)$$

where $|\boldsymbol{\omega}_{ij}|$ represents the magnitude of the subsequent connection from neuron I_i. h denotes the number of subsequent connections for neuron I_i.

Finally, we calculate the importance of neurons in each hidden layer. The importance of each neuron in the first hidden layer is consistent with that of the second layer. We start the calculation from the second hidden layer, applying the same method used for the neurons in the output layer [as shown in (4)], to sequentially compute the neuron importance of each hidden layer. The importance $\mathcal{S}_{H_i}$ of the ith neuron H_i in the hidden layer is determined as follows:

$$\mathcal{S}_{H_i}^{(t)} = \mathcal{S}_{H_i}^{(t-1)} + \frac{1}{2}\left(\sum_{i=1}^{N}\left|\frac{\partial \mathcal{L}_i\big(Q(\mathbf{s}_i,\mathbf{a}_i),\hat{Q}(\mathbf{s}_i,\mathbf{a}_i)\big)}{\partial Q(\mathbf{s}_i,\mathbf{a}_i)}\right| + \sum_{j=1}^{h}\left|\boldsymbol{\omega}_{ji}\right|\right). \quad (6)$$

Here, we utilize the forward connection $\boldsymbol{\omega}_{ji}$ instead of $\boldsymbol{\omega}_{ij}$, as each neuron in the hidden layer is tasked with processing the information received from its forward connections. Using the aforementioned approach, we can calculate the importance of each neuron in the network.

3) Calculating the Importance of Connections: After determining the importance of each neuron, we employ distinct approaches to assess the importance of connections during the connection dropout and growth stages.

Specifically, during the connection dropout stage, we estimate the importance $\mathcal{S}_{\omega}^{(t)}$ of a connection ω by considering both its magnitude $|\omega|$ and the importance of its connected neuron $\mathcal{S}_i^{(t)}$, as shown in

$$\mathcal{S}_{\omega}^{(t)} = |\omega|\mathcal{S}_i^{(t)}. \quad (7)$$

Here, for the connections of neurons in the input and output layers, $\mathcal{S}_i^{(t)}$ represents the importance of the neurons in the input and output layers, respectively. In the case of connections within the remaining hidden layers, $\mathcal{S}_i^{(t)}$ denotes the importance of the subsequent neurons. Connections with low importance will be removed.

During the connection growth stage, new connections are established in locations where there were previously no connections, based on the importance of the connections. The importance $\mathcal{S}_{\omega}^{(t)}$ of each connection ω is determined by

$$\mathcal{S}_{\omega}^{(t)} = \mathcal{S}_i^{(t)} + \mathcal{S}_{i-1}^{(t)} \quad (8)$$

where $\mathcal{S}_{i-1}^{(t)}$ and $\mathcal{S}_i^{(t)}$ are the two neurons that are connected prior to and subsequent to the connection ω. Connections with high importance are grown.

C. Topology Adaptation Strategy

In this section, we discuss the topology adaptation strategy using the importance of connection defined above. This strategy dynamically adjusts the sparse topology of the critic while keeping the actor's topology adjustment strategy consistent with the baseline. It involves a "drop and grow" procedure in two steps, utilizing the computed importance to determine connections for dropping and growing.

1) Drop: In the first step, we calculate the importance of each connection, and then a set of the least important connections (low $\mathcal{S}_{\boldsymbol{\theta}_i}$) is dropped from each layer. The dropped connections are assigned a value of 0 using a mask $\boldsymbol{M}_{\boldsymbol{\theta}}$, specifically $\boldsymbol{\theta} = \boldsymbol{\theta} \odot \boldsymbol{M}_{\boldsymbol{\theta}}$. The mask $\boldsymbol{M}_{\boldsymbol{\theta}}$ is computed to reflect the dropped connections as illustrated below

$$\boldsymbol{M}_{\boldsymbol{\theta}} = \boldsymbol{M}_{\boldsymbol{\theta}} - \mathbf{1}\big(-\mathcal{S}_{\boldsymbol{\theta}_i}, N_{\text{drop}}\big), \quad \text{for all } i \in \{1,\ldots,\|\boldsymbol{\theta}\|_0\}. \quad (9)$$

Here, N_{drop} represents the number of connections that will be dropped. $\boldsymbol{\theta}_i$ represents the ith connection. The function $\mathbf{1}$ serves as an indicator, where the first parameter specifies the condition for converting a value to 1, and the second parameter determines the count of values to be converted. $\|\cdot\|_0$ denotes the standard L_0 norm. $-\mathcal{S}_{\boldsymbol{\theta}_i}$ indicates the lower importance.

2) Grow: In the second step, we compute the importance of connections at each location without existing connections, then sort them in descending order, and select the top N_{grow} important positions for growing connections with $\boldsymbol{M}_{\boldsymbol{\theta}}$. The computation for $\boldsymbol{M}_{\boldsymbol{\theta}}$ is as follows:

$$\boldsymbol{M}_{\boldsymbol{\theta}} = \boldsymbol{M}_{\boldsymbol{\theta}} + \mathbf{1}\big(\mathcal{S}_{\boldsymbol{\theta}_i} \wedge (\boldsymbol{\theta}_i == 0), N_{\text{grow}}\big) \quad \text{for all } i \in \{1,\ldots,\|\boldsymbol{\theta}\|_0\}. \quad (10)$$

Here, $N_{\text{grow}} = N_{\text{drop}}$ and $\wedge$ represents the logical AND operator. $\mathcal{S}_{\boldsymbol{\theta}_i}$ indicates the larger importance. The values of the newly added connections are initialized to zero. At this stage, we have presented our topology adaptation strategy. The remaining steps and modules, including the initialization of sparse topology, align with the baseline DST method.

Remark 1: As demonstrated in (9) and (10), our topology adaptation strategy offers two advantages compared to existing DST methods.

1) During the connection dropout phase, we consider not only the absolute value of the connection itself but also the absolute values of neighboring connections and the TD error to assess the importance of connections. This approach is more reasonable and comprehensive than existing DST methods that solely rely on the absolute value of the connection itself to determine importance and hence can identify and remove unimportant connections more effectively.
2) During the connection growth phase, we evaluate the importance of each location where a connection does not exist to decide whether to grow a connection at that location. Unlike existing DST methods that generate connections randomly, our method can identify and grow connections at important locations, thus avoiding randomness.

D. Complexity Analysis

This section analyzes the complexity of the proposed method. Given that our method presents a novel approach for

evaluating connection importance without excessively altering the topological adaptation strategy, we will focus our analysis solely on the computational complexity of computing connection importance. The analysis is outlined as follows.

1) Time Complexity: In Step 1, computing the gradient of TD error requires calculations for each sample. With N samples, the gradient needs to be computed N times. Therefore, the time complexity of Step 1 is $O(N)$.

In Step 2, computing the importance of neurons in each layer. Let us assume there are m neurons in the output layer, n neurons in the input layer, and k neurons in the hidden layers. The calculation of importance for each neuron requires at most H connections. Since the computation of importance involves several simple addition operations, the time complexity of Step 2 is $O(m \cdot H + n \cdot H + k \cdot H)$.

In Step 3, computing the importance of connections. Let us assume that there are p connections in the output layer, q connections in the input layer, and r connections in the hidden layer. Since calculating the importance of each connection involves a simple multiplication operation, the time complexity of Step 3 is $O(p+q+r)$.

Overall, the total time complexity of the method is $O(N + m \cdot H + n \cdot H + k \cdot H + p + q + r)$.

2) Space Complexity: Since our method does not increase network parameters or the mini-batch size, the space complexity remains the same as the original DST methods.

3) Network Size and FLOPs: In the field of DST, network size and floating-point operations (FLOPs) are commonly utilized as metrics for evaluating complexity [4], [5], [6]. Specifically, network size represents the scale of the network by measuring the number of parameters it contains, while FLOPs quantify the computational burden of the network. Empirically, these metrics are dependent on the network structure, i.e., the number of neurons in each layer [4], [5], [6]. As our approach solely focuses on calculating connection importance and the topology adaptation strategy without altering the network structure. Moreover, while integrating our method into existing DST methods may introduce some additional computational overhead, it is important to note that the computational cost associated with topology adaptation is often overlooked when evaluating FLOPs in the existing literature [4], [5], [6]. Taking these two points into consideration, the network size and FLOPs of our method are the same as those of the original DST methods. Therefore, we solely compare the returns of our method and the baseline methods.

E. Overall Training Procedure

In this section, we present the training process of our method using SAC and TD3 as examples, which are commonly employed underlying algorithms in previous DST literature [4], [5], [6]. Specifically, our method follows a systematic procedure that includes several key steps: sparse topology initialization, training, target network updates, and dynamic topology adaptation. These steps align with the standard DST approach, as illustrated in Fig. 3. However, it is important to acknowledge that different DRL algorithms employ unique methods for model updates. Consequently, when applying our method and existing DST techniques to other underlying DRL algorithms, it becomes imperative to make appropriate modifications to align our method with the training process of those specific algorithms. The specific training steps of our method are as follows.

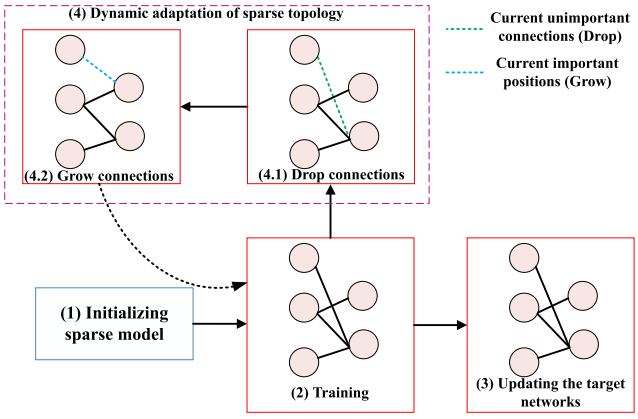


Fig. 3. Training procedure of our method.

1) Initialization: Let θ^{μ} be actor. Let $\theta^{Q_1}, \theta^{Q_2}$ be the two critics. Let L be the number of layers in each network. Let $\theta^{\mu'}, \theta^{Q_1'}, \theta^{Q_2'}$ be the target actor and target critics, respectively. The target networks have the same initial sparse topology as the current network.

To sparsity our networks, we initialize them with a sparse topology that applies a binary mask $\boldsymbol{M} = \{\boldsymbol{M}^i\}|_{i=1}^{L}$ to indicate the locations where sparse connections exist between layers. For $M^{ij} \in \{0,1\}$, $M^{ij} = 1$ indicates that connection j exists in layer $i-1$ to layer i. Let c^{i-1} and c^i be the number of neurons in layer $i-1$ and layer i, respectively. Sparse training introduces a coefficient, denoted by λ^i, to set the sparsity level in layer i. For example, when λ^i is 64 and $c^i = c^{i-1} = 256$, the sparsity level for layer i is 0.5. Consistent with existing DST approaches in DRL [4], [5], [6], we use the Erdös-Rényi random graph to initialize a sparse topology in each layer i, where the mask probability $p(M^i)$ in layer i is computed using

$$p(M^i) = \lambda^i \frac{c^i + c^{i-1}}{c^i \times c^{i-1}}. \tag{11}$$

Here, $1 - p(M^i)$ represents the sparsity level of the ith layer. Let $\boldsymbol{M}_{\theta^{\mu}}$ be the binary mask for the actor and $\boldsymbol{M}_{\theta^{Q_1}}, \boldsymbol{M}_{\theta^{Q_2}}$ be the binary mask for each critic. With the introduction of the binary mask and the mask probability, the sparse topology of each actor and critic is given by

$$\begin{aligned} \theta^{\mu} &= \theta^{\mu} \odot \boldsymbol{M}_{\theta^{\mu}} \\ \theta^{Q_l} &= \theta^{Q_l} \odot \boldsymbol{M}_{\theta^{Q_l}} \quad \forall l \in \{1,2\}. \end{aligned} \tag{12}$$

Here, $\odot$ is the element-wise multiplication operator.

2) Training: Our approach involves training two components: the actor and the critics. We will now outline the training process for each of these components. To update the actor and critic, we utilize a mini-batch $\mathcal{D}_0$ consisting of N samples $\{(s,a,r,s')\}$ from the replay buffer $\mathcal{D}$.

a) Update of critics: The critic θ^{Q_i} is updated as follows:

$$\theta^{Q_i} \leftarrow \underset{\theta^{Q_i}}{\operatorname{argmin}}\, N^{-1} \sum_{(s,a)\in\mathcal{D}_0} \left(\hat{Q}(s,a) - Q_{\theta^{Q_i}}(s,a)\right)^2 \tag{13}$$

where $\hat{Q}(s,a) \leftarrow r+\gamma \min_{i=1,2} Q_{\theta^{Q_i'}}(s',\tilde{a})$, $\tilde{a} \leftarrow \pi_{\theta^{\mu'}}(s')+\epsilon$; $\epsilon \sim \mathcal{N}(0,0.2)$ is a noise incorporated to the actions selected by the target actor, which are then clipped to the range of $(-0.5,0.5)$.

b) Update of the actor: In this context, we utilize TD3 and SAC as examples to illustrate the actor update process. Specifically, in the case of TD3, the actor update involves the utilization of $Q_{\theta^{Q_1}}(s,\pi_{\theta^\mu(s)})$ and $\pi_{\theta^\mu}(s)$. The update rule can be expressed as follows:

$$\nabla_{\theta^\mu} J(\theta^\mu) = N^{-1} \sum_{(s,a)\in\mathcal{D}_0} \nabla_a Q_{\theta^{Q_1}}\left(s,\pi_{\theta^\mu(s)}\right) \nabla_{\theta^\mu}\pi_{\theta^\mu}(s). \quad (14)$$

On the other hand, in the case of SAC, the actor update method differs. The actor is updated based on the following expression:

$$\nabla_{\theta^\mu} J(\theta^\mu) = \nabla_{\theta^\mu}\left(-\frac{1}{N}\sum_{(s,a)\in\mathcal{D}_0}\left(\frac{\sum_{i=1}^{2} Q_{\theta^{Q_i}}(s,a)}{2} + \alpha'\mathcal{H}\right)\right). \quad (15)$$

Here, $\mathcal{H} = \pi_{\theta^\mu}(a|s)\log\pi_{\theta^\mu}(a|s)$. The temperature parameter α' determines the degree of entropy regularization $\mathcal{H}$.

3) Updating the Target Networks: The update of target networks involves the actor (in the case of TD3, not SAC) and two critics, which are updated using a soft method with a delay. The update process is given by

$$\begin{aligned} \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \\ \theta^{Q_i'} &\leftarrow \tau\theta^{Q_i} + (1-\tau)\theta^{Q_i'}, \quad i=1,2 \end{aligned} \quad (16)$$

where $\tau = 0.005$ is the coefficient.

4) Dynamic Adaptation of Sparse Topology: The key distinction between SST and DST is that the former does not incorporate dynamic adaptation of the sparse topology for the current networks. The details of this step refer to the topology adaptation strategy discussed in Section IV-B. Therefore, we will refrain from reiterating it here.

Algorithm 1 serves as an illustrative example, providing a demonstration of the training process employed by our method within the context of TD3. However, the true strength of our approach lies in its exceptional adaptability, encompassing a broad spectrum of DST technologies and underlying DRL algorithms. This inherent flexibility allows for seamless integration into various existing DST frameworks and underlying DRL algorithms, resulting in a substantial expansion of its applicability beyond the limitations imposed by TD3.

Algorithm 1 Training Procedure of Our Method

1: Initialize actor θ^μ and critics θ^{Q_1}, θ^{Q_2} with random weights
2: Generate $\boldsymbol{M}_{\theta^\mu}$ and $\boldsymbol{M}_{\theta^{Q_1}}, \boldsymbol{M}_{\theta^{Q_2}}$ with the Erdös-Rényi random graph
3: Create initial sparse models: $\theta^\mu = \theta^\mu \odot \boldsymbol{M}_{\theta^\mu}$, $\theta^{Q_l} = \theta^{Q_l} \odot \boldsymbol{M}_{\theta^{Q_l}}, \quad \forall l \in \{1,2\}$
4: Initialize target networks $\theta^{Q_1'}$, $\theta^{Q_2'}$ with weights $\theta^{Q_1'} \leftarrow \theta^{Q_1}$, $\theta^{Q_2'} \leftarrow \theta^{Q_2}$
5: Initialize replay buffer $\mathcal{D}$
6: Initialize the maximum number of episodes M
7: $episode = 1$
8: **repeat**
9: Initialize a random process for action exploration
10: Receive initial observation state s_1
11: **for** $t = 1$ to T **do**
12: Select action $a_t = \text{clip}(\mu(s_t|\theta^\mu) + \epsilon)$
13: Execute action a_t and observe reward r_t and new state s_{t+1}
14: Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
15: Sample a random mini-batch $\mathcal{D}_0$ of N transitions (s_i, a_i, r_i, s_{i+1}) from $\mathcal{D}$
16: Compute the target Q value: $\hat{Q}(s_i,a_i) = r_i + \gamma \min_{j=1,2} Q_j'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q_j'})$
17: Update critics by minimizing the TD error: $\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(\hat{Q}(s_i,a_i) - Q_j(s_i,a_i|\theta^{Q_j}))^2$, for $j=1,2$
18: Update $\boldsymbol{M}_{\theta^{Q_1}}, \boldsymbol{M}_{\theta^{Q_2}}$ with our topology adaptation strategy
19: Update the sparse topology of critics: $\theta^{Q_l} = \theta^{Q_l} \odot \boldsymbol{M}_{\theta^{Q_l}}, \quad \forall l \in \{1,2\}$
20: Update actor using the sampled policy gradient: $\nabla_{\theta^\mu} \approx \sum_{i=1}^{N} \zeta \nabla_{\mu(s_i)} Q(s_i, \mu(s_i)|\theta^{Q_1}) \nabla_{\theta^\mu}\mu(s_i|\theta^\mu)$
21: Update $\boldsymbol{M}_{\theta^\mu}$ with baseline methods' topology adaptation strategies
22: Update the sparse topology of the actor: $\theta^\mu = \theta^\mu \odot \boldsymbol{M}_{\theta^\mu}$
23: Every d steps, update target networks: $\theta^{Q_j'} \leftarrow \tau\theta^{Q_j} + (1-\tau)\theta^{Q_j'}$, $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$, for $j=1,2$
24: **end for**
25: $episode{+}{+}$
26: **until** $M - episode$

V. Experiments

This section commences by offering a comprehensive overview of the experimental preparations, encompassing the baseline, benchmarks, evaluation criteria, implementation specifics, and other pertinent details. Subsequently, we present the results and analysis obtained from the experiments conducted across various tasks. Regarding software, we utilized Torch 1.2.0, gym 0.16.0, and mujoco-py 1.50.0.1. In terms of hardware, our setup consisted of an Intel Core i7-9700 processor, 64 GB RAM, and NVIDIA RTX 2080 GPU. Each method was executed five times with different random seeds, following prior work [4]. The plot illustrates the average performance (depicted by the bold curve) and the corresponding standard deviation (represented by the shaded region) across five runs. The implementation details of our method can be found in the supplementary materials.

A. Experiment Preparation

1) Baselines: We employed three SOTA sparse training methods as baselines: SST [4], DST [5], and rigged reinforcement learning lottery (RLx2) [6]. These methods were chosen as baselines due to their status as the most recent research in this field. DST and RLx2 are SOTA DST methods, whereas SST is a static method. Meanwhile, we conducted a comparison with the SAC and TD3 algorithms implemented

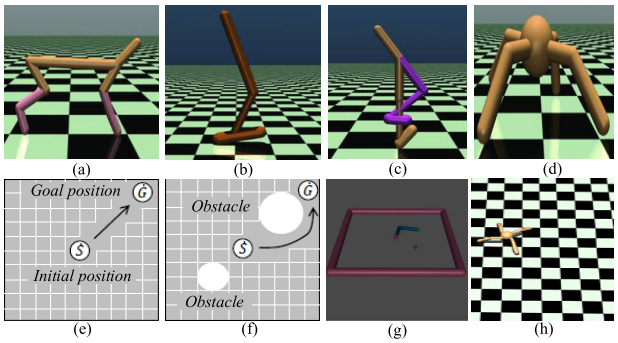


Fig. 4. Experimental scenes. (a) HalfCheetah. (b) Hopper. (c) Walker2d. (d) Ant. (e) 2DNav-v1. (f) 2DNav-v2. In this case, the white circle represents an obstacle. The initial position is denoted by S, while the desired position is represented by G. (g) ReacherNav. (h) AntNav.

using dense models. Detailed descriptions of these methods can be found in the supplementary materials.

2) Benchmarks: In order to assess the effectiveness of each method, we conducted an evaluation following the SOTA DST studies [4], [5], [6]. The evaluation was performed on four standard MuJoCo continuous control tasks: HalfCheetah-v2, Hopper-v2, Walker2d-v2, and Ant-v2. Additionally, we included four navigation tasks commonly used in the DRL literature as supplementary validation tasks: 2DNav-v1 [37], 2DNav-v2 [37], reacher navigation (ReacherNav) [37], and ant navigation (AntNav) [37]. Notably, ReacherNav and AntNav are derived from the standard MuJoCo tasks (Ant-v2 and Reacher-v2, respectively) and involve guiding a two-joint torque-controlled robot arm or an ant robot to a specific target point from an initial position. Brief descriptions of 2DNav-v1 and 2DNav-v2 are provided below:

2DNav-v1 and 2DNav-v2 are tasks where a point agent is required to navigate to a target position in a 2-D environment. In 2DNav-v1, the robot is trained to navigate in an obstacle-free environment, with different initial conditions achieved by changing the target position. On the other hand, 2DNav-v2 focuses on training the robot to navigate in an environment with obstacles, where the different initial conditions include variations in both the target position and the position of the obstacles. The eight simulation task scenes are shown in Fig. 4.

3) Metrics: Since our method does not modify the network size or FLOPs of the original DST methods, we primarily focus on analyzing the episode return and the average return across all episodes as the key evaluation metrics for each method. These two metrics are commonly utilized in previous literature [37] to assess the performance of DRL approaches. In the main text, we present the average return across all episodes for each method, while the detailed training process of each method in terms of the episode return is provided in the supplementary material.

Specifically, the episode return represents the average reward obtained over a certain number of episodes (e.g., ten episodes), which is visualized as a curve that changes with each episode. The average return $\hat{\mathcal{R}}$ across all episodes represents the average reward obtained throughout the training episodes for a specific task. Mathematically, it is given by $\hat{\mathcal{R}} = (1/m)\sum_{i=1}^{m} r_i$. Here, m corresponds to the number of episodes tested for each seed, and r_i denotes the cumulative reward obtained in the ith episode. $\hat{\mathcal{R}}$ is a specific numerical value that quantifies the average performance. Both of these metrics are considered better when they are larger.

TABLE I
AVERAGE RETURN FOR THE VERIFICATION UNDER TD3

Method	Ant (TD3)	HalfCheetah (TD3)	Hopper (TD3)	Walker2d (TD3)
Dense	4090.86 ± 1018.85	10246.20 ± 1142.33	3326.85 ± 460.98	3909.78 ± 354.05
SST	2246.04 ± 678.76	4783.12 ± 1754.40	1913.34 ± 1013.24	2427.80 ± 1005.71
DST	2964.19 ± 2171.52	7087.58 ± 1033.81	2757.16 ± 781.49	3329.47 ± 795.70
Our DST	**4564.81** ± 661.83	**7170.01** ± 552.11	**3213.33** ± 486.54	**3957.70** ± 320.17
RLx2	3424.29 ± 1043.96	8310.90 ± 1709.62	2367.41 ± 901.62	4330.01 ± 433.49
Our RLx2	**4298.68** ± 1203.23	**9066.60** ± 1468.21	**3152.97** ± 686.58	**4756.62** ± 552.00

TABLE II
AVERAGE RETURN FOR THE VERIFICATION UNDER SAC

Method	Ant (SAC)	HalfCheetah (SAC)	Hopper (SAC)	Walker2d (SAC)
Dense	4213.93 ± 727.10	11643.70 ± 455.040	2472.15 ± 712.19	4786.79 ± 557.30
SST	4176.63 ± 919.49	6505.21 ± 541.43	2192.89 ± 1153.63	4263.96 ± 501.177
DST	**4234.84** ± 1041.60	8850.27 ± 331.80	2647.96 ± 801.61	4084.34 ± 609.099
Our DST	3736.50 ± 783.87	**9667.22** ± 206.96	**3309.30** ± 452.88	**4876.09** ± 555.88
RLx2	3880.39 ± 933.48	9257.79 ± 1125.23	**2999.54** ± 726.76	4267.02 ± 460.75
Our RLx2	**4868.28** ± 1467.30	**10193.73** ± 181.53	2874.69 ± 768.21	**4847.51** ± 608.33

B. Evaluation on the Standard MuJoCo Tasks

This section aims to validate the effectiveness of our method in enhancing existing DST methods on four standard MuJoCo tasks. Given that the highest sparsity levels verified by the SOTA DST literature [6] on the SAC network and TD3 network are 0.9 and 0.95, respectively, we have also chosen these two maximum sparsity levels in our evaluation.

1) Experimental Results: We calculated the mean and standard deviation of the returns for the final 125 000 steps of the training phase for each method, and additional training details for each method pertaining to episode returns are provided in the supplemental material. The mean and standard deviation of the returns achieved by the methods are presented in Tables I and II. These results offer valuable insights into the performance of the policy learned at the conclusion of training, thereby demonstrating the effectiveness of the final policy obtained.

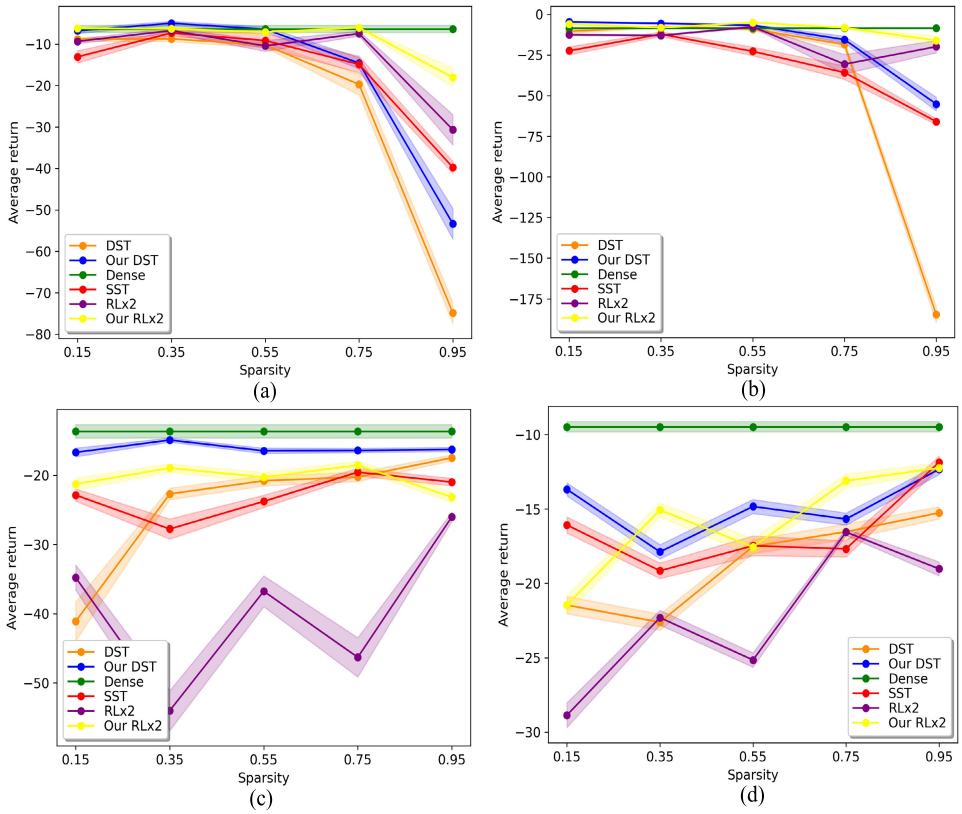


Fig. 5. Average return across five different runs on TD3. The average return across all episodes is indicated on the y-axis, while the x-axis represents the sparsity levels ranging from 0.15 to 0.95. (a) 2DNav-v1, (b) 2DNav-v2, (c) ReacherNav, and (d) AntNav.

Upon careful examination of the outcomes presented in Tables I and II, it is evident that our method (Our DST and Our RLx2) consistently achieved higher returns compared to the original DST methods (DST and RLx2) in the majority of the test cases. Remarkably, our method even outperformed the dense model in certain tasks, such as Ant, in terms of returns. These experimental findings serve as compelling evidence of the effectiveness of our method in enhancing the performance of the SOTA DST techniques.

2) Analysis of Results: The observed improvement in the above experiments can be attributed to the introduction of a novel topology adaptation strategy in our method. The topology adaptation strategy in our approach incorporates the evaluation of the TD error, the significance of neighboring connections, and the absolute value of the connection itself in determining the importance of a connection for removal. Furthermore, our approach assesses the significance of each position by considering the importance of both preceding and subsequent neurons, as opposed to randomly increasing connections. As a result, our method outperforms the previous approach, which solely relies on the absolute value of the connections to determine whether to add or remove them.

C. Evaluation on the Navigation Tasks

In this section, we present an evaluation of the performance of our method across different sparsity levels on four distinct navigation tasks: ReacherNav, AntNav, 2DNav-v1, and 2DNav-v2. Specifically, for the TD3 algorithm, we conducted tests with our method and baselines at sparsity levels ranging from 0.15 to 0.95. For the SAC algorithm, tests were conducted with our method and baselines at sparsity levels ranging from 0.1 to 0.9. The methods evaluated in this section are consistent with those discussed in Section V-B.

1) Experimental Results: Figs. 5 and 6 depict the average return across all episodes (referred to as the average return) of each method on the four navigation tasks at various sparsity levels. The experimental results reveal distinct performances for DST, RLx2, and SST across varying sparsity levels. However, when the sparsity reaches 0.9 (for SAC) or 0.95 (for TD3), the returns are generally significantly weaker than those of the original dense model. Nevertheless, our method (Our DST, Our RLx2) achieves notably higher returns compared to the original methods (DST, RLx2), effectively reducing the performance gap with dense models. In fact, in certain cases, our method even surpasses the original dense model when the sparsity is set to 0.9, as demonstrated in Fig. 6(a), (b), and (d).

Similarly, we showcase the average return of all methods at the highest sparsity levels (TD3: 0.95, SAC: 0.9), along with the corresponding training details, in terms of episode return, which are presented in the supplementary material. The corresponding results are shown in Tables III and IV. The experimental results from Tables III and IV highlight the effectiveness of our method (Our DST, Our RLx2) in significantly improving the performance of the original DST approach (DST, RLx2) across these four challenging tasks.

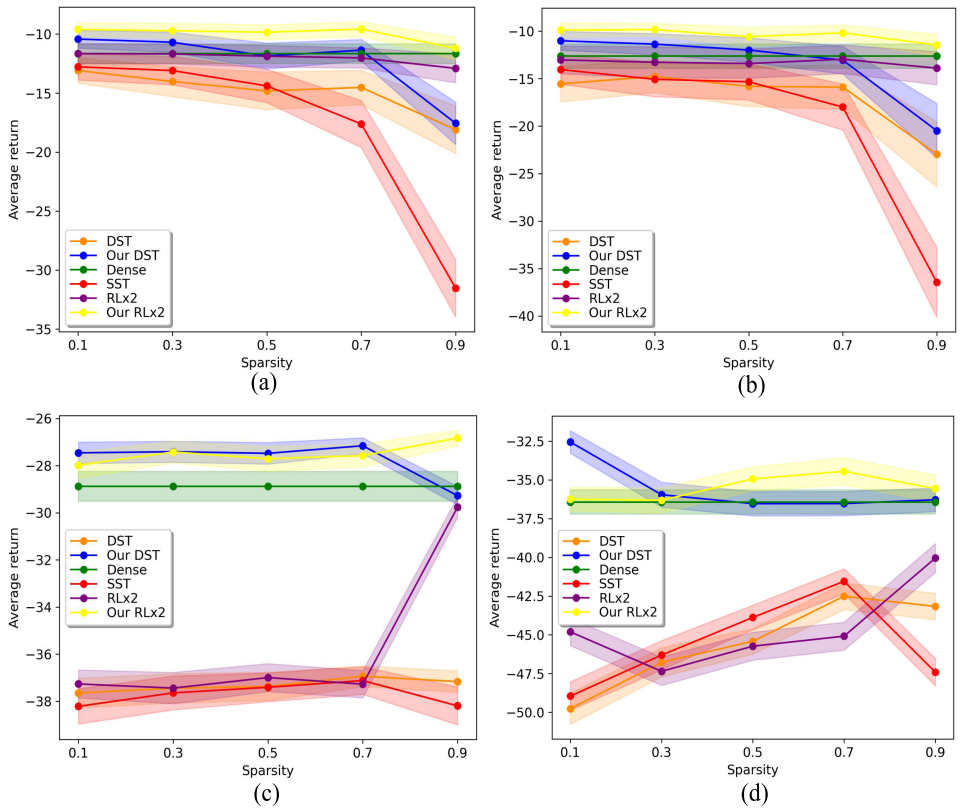


Fig. 6. Average return across five different runs on SAC. The average return across all episodes is indicated on the y-axis, while the x-axis represents the sparsity levels ranging from 0.1 to 0.9. (a) 2DNav-v1, (b) 2DNav-v2, (c) ReacherNav, and (d) AntNav.

TABLE III

AVERAGE RETURN EVALUATION OF FOUR NAVIGATION TASKS USING TD3

Method	2DNav-v1 (TD3)	2DNav-v2 (TD3)	ReacherNav (TD3)	AntNav (TD3)
Dense	-6.393 ± 0.867	-8.488 ± 0.572	-13.642 ± 0.968	-9.486 ± 0.349
SST	-39.748 ± 1.463	-65.972 ± 1.797	-20.968 ± 0.543	-11.865 ± 0.418
DST	-74.865 ± 2.513	-184.753 ± 4.878	-17.461 ± 0.442	-15.262 ± 0.411
Our DST	**-53.339** $\pm$ 3.678	**-55.172** $\pm$ 4.17	**-16.253** $\pm$ 0.31	**-12.33** $\pm$ 0.358
RLx2	-30.659 ± 3.66	-19.88 ± 3.784	-26.007 ± 0.675	-19.024 ± 0.464
Our RLx2	**-18.024** $\pm$ 2.221	**-16.074** $\pm$ 2.715	**-23.131** $\pm$ 0.768	**-12.238** $\pm$ 0.328

TABLE IV

AVERAGE RETURN EVALUATION OF FOUR NAVIGATION TASKS USING SAC

Method	2DNav-v1 (SAC)	2DNav-v2 (SAC)	ReacherNav (SAC)	AntNav (SAC)
Dense	-11.655 ± 0.83	-12.621 ± 1.289	-28.878 ± 0.627	-36.426 ± 0.782
SST	-31.528 ± 2.442	-36.451 ± 3.68	-38.194 ± 0.805	-47.405 ± 0.893
DST	-18.099 ± 2.029	-22.961 ± 3.435	-37.165 ± 0.451	-43.165 ± 0.851
Our DST	**-17.559** $\pm$ 1.793	**-20.484** $\pm$ 2.872	**-29.274** $\pm$ 0.391	**-36.28** $\pm$ 0.766
RLx2	-12.905 ± 1.213	-13.891 ± 1.745	-29.764 ± 0.435	-40.045 ± 0.933
Our RLx2	**-11.207** $\pm$ 0.969	**-11.451** $\pm$ 1.145	**-26.832** $\pm$ 0.333	**-35.558** $\pm$ 0.887

In particular, on the 2DNav-v2 task shown in Table III, the DST method exhibits a significant performance degradation of 90% compared to the original dense model, representing the most substantial performance decline among all the results we have presented. Conversely, on the same 2DNav-v2 task shown in Table III, our DST method outperforms the previous DST approach by an impressive 70% in terms of return, demonstrating the highest improvement among the results we have presented. Furthermore, our method demonstrates superior performance compared to the SST method and, in certain cases, even outperforms the original dense model, as demonstrated in Table IV.

2) Analysis of Results: Through rigorous validation, our method has consistently showcased substantial performance enhancements for two SOTA DST methods across four distinct navigation tasks, even under varying levels of sparsity. Notably, it outperforms existing DST methods in terms of returns, thereby demonstrating its effectiveness in enhancing policy performance.

TABLE V
AVERAGE RETURN EVALUATION OF THREE NAVIGATION TASKS USING SAC

Method	2DNav-v1 (SAC)	2DNav-v2 (SAC)	ReacherNav (SAC)
Our RLx2 (10%)	**-11.207** $\pm$ 0.969	**-11.451** $\pm$ 1.145	**-26.832** $\pm$ 0.333
Our RLx2 (20%)	-11.512 ± 0.860	-12.809 ± 1.256	-32.923 ± 0.584
Our RLx2 (30%)	-11.789 ± 0.961	-11.462 ± 0.934	-33.319 ± 0.629
Our RLx2 (40%)	-11.571 ± 0.781	-12.770 ± 1.273	-35.005 ± 0.524

Likewise, the remarkable impact of our method can be primarily attributed to the introduction of a novel topology adaptation strategy, as discussed in Section V-B. Furthermore, our approach demonstrates its versatility as a general method that seamlessly integrates with various existing DST methods to enhance their performance across different sparsity levels, further emphasizing its broad applicability.

D. Analysis of Hyperparameter Sensitivity

This section focuses on the hyperparameters of our method. Specifically, we analyze the number of connection drops and growths in the topology adaptation strategy stage, denoted as $N_{\text{grow}} = N_{\text{drop}}$. This hyperparameter is unique to DST and distinguishes it from regular DRL. On the other hand, other hyperparameters, such as the coefficient of soft update and the temperature parameter in SAC, are standard hyperparameters in regular DRL. Therefore, we do not provide a detailed analysis of these regular DRL hyperparameters in this section.

Our experimental configuration is identical to Section V-C, and we compare our results across three tasks: ReacherNav, 2DNav-v1, and 2DNav-v2. We utilize the RLx2 sparse training method, as it currently demonstrates the best performance. We set the sparsity level to 95% and employ the SAC algorithm as the underlying DRL algorithm. To observe the impact of the N_{grow} parameter on the performance of our approach (referred to as Our RLx2), we set N_{grow} to be 10%, 20%, 30%, and 40% of the connections in the current model.

1) Experimental Results: Similar to Section V-C, we present the average return in Table V to compare the performance of different methods. From Table V, it is evident that when N_{grow} is greater than 10%, the performance is weaker compared to when N_{grow} is set to 10%, across the three tasks in terms of return.

2) Analysis of Results: The experimental results indicate that as the value of N_{grow} increases from 10% to 40%, the performance of Our RLx2 differs; however, it consistently remains lower than when N_{grow} is set to 10%. This observation suggests that increasing the number of connections that are deleted or grown during the topology adaptation stage is not beneficial for performance improvement and also results in increased computational overhead. Conversely, when N_{grow} is set to 10%, the performance is optimal, and the computational requirements are relatively low. Therefore, we have determined that setting the N_{grow} hyperparameter to 10% of the current connections is the optimal choice for our method, which we have utilized in this work.

3) Guidance on Hyperparameter Design: Based on the experimental results presented in Section V-D, we have derived the following guidelines for hyperparameter design. First, the regular hyperparameters in DRL, such as discount factors, temperature parameters, and soft update coefficients should be set based on established design experience in regular DRL methods. Second, regarding the hyperparameter N_{grow} in our method and existing DST methods, we have determined that the optimal value for N_{grow} is 10% of the total number of connections in the current model.

E. Limitations of Our Approach

While our method effectively bridges the performance gap between dense and sparse models, it is important to acknowledge its limitations.

First, it is worth noting that in many scenarios, sparse training methods can still result in some performance sacrifices compared to the original dense models. Thus, achieving performance on par with or surpassing dense models while maintaining a sparsity level of 90% or higher remains an ongoing challenge. **Second**, we plan to extend our method to more advanced types of DRL, such as offline DRL and Federated DRL, in the future. These approaches have showcased exceptional performance in various practical applications. Exploring the compatibility and potential synergies between these methods and our approach would be valuable. **At last**, our method has not been validated in real-world scenarios involving robotic devices and complex robotic manipulation tasks. The inclusion of physical robotic platforms and intricate manipulation tasks introduces additional complexities and challenges, such as sensor noise, environmental dynamics, complex robot behaviors, and hardware limitations. Validating our method in such scenarios would provide a more comprehensive understanding of its practicality and robustness.

VI. CONCLUSION

This study addresses the observed performance degradation in existing DST methods, particularly when faced with high levels of sparsity. To overcome this challenge, we propose a generic method that can be seamlessly integrated into the existing DST methods within the framework of DRL, aiming to enhance their performance. Our approach involves the development of a novel method to evaluate the significance of connections within the model. Subsequently, we dynamically adjust the network topology by selectively removing and adding connections based on their respective importance values. Experimental results demonstrate significant improvements in two SOTA DST approaches across multiple tasks. In the future, we have plans to extend our method to more complex DRL algorithms and apply it to real-world tasks, such as autonomous driving.

REFERENCES

[1] M. Xu, X. Chen, and J. Wang, "Policy correction and state-conditioned action evaluation for few-shot lifelong deep reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 30, 2024, doi: 10.1109/TNNLS.2024.3385570.

[2] M. Xu and J. Wang, "Learning strategy for continuous robot visual control: A multi-objective perspective," *Knowl.-Based Syst.*, vol. 252, Sep. 2022, Art. no. 109448.

[3] D. Livne and K. Cohen, "PoPS: Policy pruning and shrinking for deep reinforcement learning," *IEEE J. Sel. Topics Signal Process.*, vol. 14, no. 4, pp. 789–801, May 2020.

[4] G. Sokar, E. Mocanu, D. C. Mocanu, M. Pechenizkiy, and P. Stone, "Dynamic sparse training for deep reinforcement learning," in *Proc. 31st Int. Joint Conf. Artif. Intell.*, Jul. 2022, pp. 3437–3443.

[5] B. Grooten et al., "Automatic noise filtering with dynamic sparse training in deep reinforcement learning," 2023, *arXiv:2302.06548*.

[6] Y. Tan, P. Hu, L. Pan, J. Huang, and L. Huang, "RLx2: Training a sparse deep reinforcement learning model from scratch," in *Proc. Int. Conf. Learn. Represent.*, 2023, pp. 1–13.

[7] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.

[8] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," 2015, *arXiv:1506.06579*.

[9] W. Zhou, Y. Li, Y. Yang, H. Wang, and T. Hospedales, "Online meta-critic learning for off-policy actor-critic methods," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 17662–17673.

[10] I. Kostrikov, R. Fergus, J. Tompson, and O. Nachum, "Offline reinforcement learning with Fisher divergence critic regularization," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 5774–5783.

[11] M. Xu and J. Wang, "Deep reinforcement learning for parameter tuning of robot visual servoing," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 2, pp. 1–27, Apr. 2023.

[12] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[13] Y. Yang, Y. Pan, C. Z. Xu, and D. C. Wunsch, "Hamiltonian-driven adaptive dynamic programming with efficient experience replay," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 3, pp. 3278–3290, Mar. 2024.

[14] Y. Yang, H. Modares, K. G. Vamvoudakis, and F. L. Lewis, "Cooperative finitely excited learning for dynamical games," *IEEE Trans. Cybern.*, vol. 54, no. 2, pp. 797–810, Feb. 2024.

[15] Y. Yang, B. Kiumarsi, H. Modares, and C. Xu, "Model-free λ-policy iteration for discrete-time linear quadratic regulation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 2, pp. 635–649, Feb. 2023.

[16] S. Liu, L. Yin, D. C. Mocanu, and M. Pechenizkiy, "Do we actually need dense over-parameterization? In-time over-parameterization in sparse training," in *Proc. 38th Int. Conf. Mach. Learn.*, vol. 139, 2021, pp. 6989–7000.

[17] S. Powers, E. Xing, E. Kolve, R. Mottaghi, and A. Gupta, "CORA: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents," in *Proc. Conf. Lifelong Learn. Agents*, 2022, pp. 705–743.

[18] M. Wołczyk, M. Zajac, R. Pascanu, L. Kucinski, and P. Miłoś, "Continual world: A robotic benchmark for continual reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 34, 2021, pp. 28496–28510.

[19] M. O. Yildirim, E. C. Gok, G. Sokar, D. C. Mocanu, and J. Vanschoren, "Continual learning with dynamic sparse training: Exploring algorithms for effective model updates," in *Proc. Conf. Parsimony Learn.*, 2024, pp. 94–107.

[20] W. Ren and V. G. Honavar, "EsaCL: Efficient continual learning of sparse models," 2024, *arXiv:2401.05667*.

[21] Z. Wang et al., "SparCL: Sparse continual learning on the edge," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 20366–20380.

[22] L. Yin et al., "Dynamic sparsity is channel-level sparsity learner," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024, pp. 1–14.

[23] G. Yuan et al., "MEST: Accurate and fast memory-economic sparse training framework on the edge," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 20838–20850.

[24] G. Bellec, D. Kappel, W. Maass, and R. Legenstein, "Deep rewiring: Training very sparse deep networks," 2017, *arXiv:1711.05136*.

[25] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," 2019, *arXiv:1907.04840*.

[26] S. Bibikar, H. Vikalo, Z. Wang, and X. Chen, "Federated dynamic sparse training: Computing less, communicating less, yet learning better," in *Proc. 36th AAAI Conf. Artif. Intell. (AAAI)*, 2022, vol. 36, no. 6, pp. 6080–6088.

[27] C. Liu et al., "Comprehensive graph gradual pruning for sparse training in graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 27, 2023, doi: 10.1109/TNNLS.2023.3282049.

[28] J. Liu, Z. Xu, R. Shi, R. C. C. Cheung, and H. K. H. So, "Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers," 2020, *arXiv:2005.06870*.

[29] S. Liu, J. Ye, S. Ren, and X. Wang, "Dynast: Dynamic sparse transformer for exemplar-guided image generation," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2022, pp. 72–90.

[30] G. Sokar, Z. Atashgahi, M. Pechenizkiy, and D. C. Mocanu, "Where to pay attention in sparse training for feature selection?" in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 1627–1642.

[31] Y. Zhang et al., "Dynamic sparse no training: Training-free fine-tuning for sparse LLMs," 2023, *arXiv:2310.08915*.

[32] J. Lee, S. Kim, S. Kim, W. Jo, and H.-J. Yoo, "GST: Group-sparse training for accelerating deep reinforcement learning," 2021, *arXiv:2101.09650*.

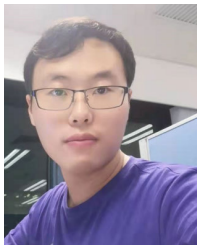
[33] M. A. Vischer, R. T. Lange, and H. Sprekeler, "On lottery tickets and minimal task representations in deep reinforcement learning," 2021, *arXiv:2105.01648*.

[34] H. Zhao, J. Wu, Z. Li, W. Chen, and Z. Zheng, "Double sparse deep reinforcement learning via multilayer sparse coding and nonconvex regularized pruning," *IEEE Trans. Cybern.*, vol. 53, no. 2, pp. 765–778, Feb. 2023.

[35] T. Zhang et al., "Replay-enhanced continual reinforcement learning," 2023, *arXiv:2311.11557*.

[36] O. Sigaud, "Combining evolution and deep reinforcement learning for policy search: A survey," *ACM Trans. Evol. Learn. Optim.*, vol. 3, no. 3, pp. 1–20, Sep. 2023.

[37] Z. Wang, C. Chen, and D. Dong, "A Dirichlet process mixture of robust task models for scalable lifelong reinforcement learning," *IEEE Trans. Cybern.*, vol. 53, no. 12, pp. 7509–7520, Dec. 2023.



Meng Xu received the B.Sc. and M.S. degrees in software engineering and computer science from Northwestern Polytechnical University, Xi'an, China, in 2017 and 2020, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong, Hong Kong.

His research interests include reinforcement learning and machine learning.

Xinhong Chen received the B.Eng. degree in software engineering from Sun Yat-sen University, Guangzhou, Guangdong, China, in 2018, and the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2022.

He is currently working as a Post-Doctoral With the Department of Computer Science, City University of Hong Kong. His research interests include natural language processing, sentiment analysis, autonomous driving, and causality mining.

Jianping Wang (Fellow, IEEE) received the B.S. and M.S. degrees in computer science from Nankai University, Tianjin, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from The University of Texas at Dallas, Richardson, TX, USA, in 2003.

She is currently working as a Full Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. Her research interests include security, autonomous driving, cloud computing, edge computing, and machine learning.