Dynamic Sparse Training for Deep Reinforcement Learning

Ghada Sokar[1], **Elena Mocanu**[2], **Decebal Constantin Mocanu**[1,2], **Mykola Pechenizkiy**[1] and **Peter Stone**[3]
[1]Eindhoven University of Technology, The Netherlands
[2]University of Twente, The Netherlands
[3]The University of Texas at Austin, Sony AI, United States
{g.a.z.n.sokar, m.pechenizkiy}@tue.nl, {e.mocanu, d.c.mocanu}@utwente.nl, pstone@cs.utexas.edu

Abstract

Deep reinforcement learning (DRL) agents are trained through trial-and-error interactions with the environment. This leads to a long training time for dense neural networks to achieve good performance. Hence, prohibitive computation and memory resources are consumed. Recently, learning efficient DRL agents has received increasing attention. Yet, current methods focus on accelerating *inference* time. In this paper, we introduce for the *first time* a dynamic sparse training approach for deep reinforcement learning to accelerate the *training* process. The proposed approach trains a *sparse* neural network from scratch and dynamically adapts its topology to the changing data distribution during training. Experiments on continuous control tasks show that our *dynamic sparse* agents achieve higher performance than the equivalent dense methods, reduce the parameter count and floating-point operations (FLOPs) by 50%, and have a faster learning speed that enables reaching the performance of dense agents with $40-50\%$ reduction in the training steps.

1 Introduction

Deep reinforcement learning (DRL) has achieved remarkable success in many applications. The power of deep neural networks as function approximators allows RL agents to scale to environments with high-dimensional state and action spaces. This enables high-speed growth in the field and the rise of many methods that improve the performance and stability of DRL agents [Wang *et al.*, 2020]. While the achieved performance is impressive, a long training time is required to obtain this performance. For instance, it took more than 44 days to train a Starcraft II agent using 32 third-generation tensor processing units (TPUs) [Vinyals *et al.*, 2019]. The very long training time leads to high energy consumption and prohibitive memory and computation costs. In this paper, we ask the following question: *Can we provide efficient DRL agents with less computation cost and energy consumption while maintaining superior performance?*

Few recent works attempt to accelerate the *inference* time of DRL agents via pruning [Livne and Cohen, 2020] or training a compact network under the guidance of a larger network (knowledge distillation) [Zhang *et al.*, 2019]. Despite the computational improvement achieved at inference, extensive computations throughout the training of *dense* networks are still consumed. Our goal is to accelerate the training process as well as the inference time of DRL agents.

The long training time of a DRL agent is due to two main factors: **(1)** the extensive computational cost of training deep neural networks caused by the very high number of network parameters [Jouppi *et al.*, 2017] and **(2)** the learning nature of a DRL agent in which its policy is improving through many trial-and-error cycles while interacting with the environment and collecting a large amount of data. In this paper, we introduce dynamic sparse training (DST) [Mocanu *et al.*, 2021; Hoefler *et al.*, 2021] in the DRL paradigm for the first time to address these two factors[1]. Namely, we propose an efficient training approach that can be integrated with existing DRL methods. Our approach is based on training *sparse* neural networks from scratch with a fixed parameter count throughout training **(1)**. During training, the sparse topology is optimized via adaptation cycles to *quickly adapt* to the online changing distribution of the data **(2)**. Our training approach enables reducing memory and computation costs substantially. Moreover, the quick adaptation to the new samples from the improving policy during training leads to a faster learning speed.

In fact, the need for neural networks that can adapt, e.g., change their control policy dynamically as environmental conditions change, was broadly acknowledged by the RL community [Stanley, 2003]. Although prior works related to the automatic selection of function approximation based on neuroevolution exist [Heidrich-Meisner and Igel, 2009], perhaps the most connected in the spirit to our proposed method is a combination of NeuroEvolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen, 2002] and temporal difference (TD) learning (i.e., NEAT+Q [Whiteson and Stone, 2006]). Still, the challenge remains, and cutting-edge DRL algorithms do not account for the benefits of adaptive neural networks training yet.

Our contributions in this paper are as follows:

- The principles of dynamic sparse training are introduced

[1]Code is available at: https://github.com/GhadaSokar/Dynamic-Sparse-Training-for-Deep-Reinforcement-Learning. The extended version of this paper is available at https://arxiv.org/abs/2106.04217.

in the deep reinforcement learning field for the first time.

- Efficient improved versions of two state-of-the-art algorithms, TD3 [Fujimoto *et al.*, 2018] and SAC [Haarnoja *et al.*, 2018a], are obtained by integrating our proposed DST approach with the original algorithms.
- Experimental results show that our training approach reduces the memory and computation costs of training DRL agents by 50% while achieving superior performance. Moreover, it achieves a faster learning speed, reducing the required training steps.
- Analysis insights demonstrate the promise of dynamic sparse training in advancing the field and allowing for DRL agents to be trained and deployed on low-resource devices (e.g., mobile phones, tablets, and wireless sensor nodes) where the memory and computation power are strictly constrained.

2 Related Work

Sparsity in DRL. To the best of our knowledge, the current advance in deep reinforcement learning is achieved using *dense* neural networks. Few recent studies have introduced sparsity in DRL via pruning. PoPS [Livne and Cohen, 2020] first trains a dense teacher neural network to learn the policy. This dense teacher policy network guides the iterative pruning and retraining of a student policy network via knowledge distillation. In [Zhang *et al.*, 2019], the authors aim to accelerate the behavior policy network and reduce the time for sampling. They use a smaller network for the behavior policy and learn it simultaneously with a large dense target network via knowledge distillation. GST [Lee *et al.*, 2021] was proposed as an algorithm for weight compression in DRL training by simultaneously utilizing weight grouping and pruning. Some other works [Yu *et al.*, 2019; Vischer *et al.*, 2021] studied the existence of the lottery ticket hypothesis [Frankle and Carbin, 2018] in RL, which shows the presence of sparse subnetworks that can outperform dense networks when they are trained from scratch. Pruning dense networks increases the computational cost of the training process as it requires iterative cycles of pruning and retraining. This work introduces the first efficient training algorithm for DRL agents that trains sparse neural networks directly from scratch and adapts to the changing distribution.

Dynamic Sparse Training (DST). DST is the class of algorithms that train sparse neural networks *from scratch* and jointly optimize the weights and the sparse topology during training. This direction aims to reduce the computation and memory overhead of training dense neural networks by leveraging the redundancy in the parameters (i.e., being overparametrized) [Denil *et al.*, 2013]. Efforts in this line of research are devoted to supervised and unsupervised learning. The first work in this direction was proposed by [Mocanu *et al.*, 2018]. They proposed a Sparse Evolutionary Training algorithm (SET) that dynamically changes the sparse connectivity during training based on the values of the connections. SET achieves higher performance than dense models and static sparse networks trained from scratch. The success of the SET algorithm opens the path to many interesting DST methods that bring higher performance gain. These algorithms differ from each other in the way the sparse topology is adapted during training [Mostafa and Wang, 2019; Evci *et al.*, 2020; Jayakumar *et al.*, 2020; Liu *et al.*, 2021; Sokar *et al.*, 2021].

In this work, we adopt the topological adaptation from the SET method in our proposed approach. The motivation is multifold. First, SET is simple yet effective; it achieves the same or even higher accuracy than dense models with high sparsity levels across different architectures (e.g., multi-layer perceptrons, convolutional neural networks, restricted Boltzmann machines). Second, unlike other DST methods that use the values of non-existing (masked) weights in the adaptation process, SET uses only the values of existing sparse connections. This makes SET truly sparse and memory-efficient [Liu *et al.*, 2020]. Finally, it does not need high computational resources for the adaptation process. It uses readily available information during the standard training. These factors are favorable for our goal to train efficient DRL agents suitable for real-world applications. We leave evaluating other topological adaptation strategies for future work.

3 Proposed Method

In this section, we describe our proposed method, which introduces dynamic sparse training for DRL. Here, we focus on integrating our training approach with one of the state-of-the-art DRL methods; Twin Delayed Deep Deterministic policy gradient (TD3) [Fujimoto *et al.*, 2018]. We named our new approach Dynamic Sparse TD3 or "DS-TD3" for short. TD3 is an efficient DRL method that offers good performance in many tasks [Shi *et al.*, 2020; Woo *et al.*, 2020]. Yet, our approach can be merged into other DRL methods as well. The integration with soft actor-critic (SAC) [Haarnoja *et al.*, 2018a] is presented in the extended version of the paper.

TD3 is an actor-critic method that addresses the overestimation bias in previous actor-critic approaches. In actor-critic methods, a policy π is known as the *actor*, and a state-value function Q is known as the *critic*. Target networks are used to maintain fixed objectives for the actor and critic networks over multiple updates. In short, TD3 limits the overestimation bias using a pair of critics. It takes the smallest value of the two critic networks to estimate the Q value to provide a more stable approximation. To increase the stability, TD3 proposed a delayed update of the actor and target networks. In addition, the weights of the target networks are slowly updated by the current networks by some proportion τ. *In this work*, we aim to dynamically train the critics and actor networks along with their corresponding target networks from scratch with sparse neural networks to provide efficient DRL agents. In the rest of this section, we will explain our proposed DST approach for TD3. The full details are also provided in Algorithm 1.

Our proposed DS-TD3 consists of four main phases: sparse topology initialization, adaptation schedule, topological adaptation, and maintaining sparsity levels.

Sparse Topology Initialization (Algorithm 1 L1-L4). TD3 uses two critic networks ($Q_{\boldsymbol{\theta}_1}, Q_{\boldsymbol{\theta}_2}$) and one actor network ($\pi_{\boldsymbol{\phi}}$) parameterized by $\boldsymbol{\theta_1} = \{\boldsymbol{\theta}_1^l\}|_{l=1}^L$, $\boldsymbol{\theta_2} = \{\boldsymbol{\theta}_2^l\}|_{l=1}^L$, and $\boldsymbol{\phi} = \{\boldsymbol{\phi}^l\}|_{l=1}^L$ respectively; where L is the number of

layers in a network. We initialize each of the actor and critic networks with a sparse topology. Sparse connections are allocated in each layer between the hidden neurons at layer $l-1$ and layer l. We represent the locations of the sparse connections by a binary mask $\boldsymbol{M} = \{\boldsymbol{M}^l\}|_{l=1}^{L}$. Following [Mocanu *et al.*, 2018], we use Erdős–Rényi random graph [Erdos *et al.*, 1960] to initialize a sparse topology in each layer l. Namely, the probability of a connection j in layer l is given by:

$$p(\boldsymbol{M}^j) = \lambda^l \frac{n^l + n^{l-1}}{n^l \times n^{l-1}}, \tag{1}$$

where λ^l is a hyperparameter to control the sparsity level in layer l, and n^{l-1} and n^l are the neurons count in layers $l-1$ and l, respectively. $\boldsymbol{M}^j \in \{0,1\}$; a value of 1 means the existence of a weight in location j. We omit the index l from the mask and weight matrices for readability. A sparse topology is created in each layer for the actor and critic networks:

$$\begin{aligned} \boldsymbol{\phi} &= \boldsymbol{\phi} \odot \boldsymbol{M}_{\boldsymbol{\phi}}, \\ \boldsymbol{\theta_i} &= \boldsymbol{\theta_i} \odot \boldsymbol{M}_{\boldsymbol{\theta_i}}, \qquad \forall i \in \{1,2\}, \end{aligned} \tag{2}$$

where $\odot$ is an element-wise multiplication operator and $\boldsymbol{M}_{\boldsymbol{\phi}}$, $\boldsymbol{M}_{\boldsymbol{\theta_1}}$, and $\boldsymbol{M}_{\boldsymbol{\theta_2}}$ are binary masks to represent the sparse weights in the actor and two critic networks, respectively. The initial sparsity level is kept fixed during the training.

The target policy and target critic networks are parameterized by $\boldsymbol{\phi'}$, $\boldsymbol{\theta}'_1$, and $\boldsymbol{\theta}'_2$, respectively. Initially, the target networks have the same sparse topology and the same weights as the current networks: $\boldsymbol{\phi'} \leftarrow \boldsymbol{\phi}$, $\boldsymbol{\theta}'_1 \leftarrow \boldsymbol{\theta_1}$, $\boldsymbol{\theta}'_2 \leftarrow \boldsymbol{\theta_2}$.

After the topological and weight initialization, the agent collects enough data before training using a purely exploratory policy. During training, for each time step, TD3 updates the pair of critics towards the minimum target value of actions selected by the target policy $\pi_{\boldsymbol{\phi'}}$:

$$y = r + \gamma \min_{i=1,2} Q_{\boldsymbol{\theta}'_i}(s', \pi_{\boldsymbol{\phi'}}(s') + \epsilon), \tag{3}$$

where γ is the discounting factor, r is the current reward, s' is the next state, and $\epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$ is the proposed clipped noise by TD3, defined by $\tilde{\sigma}$, to increase the stability; where c is the clipped value. As discussed, TD3 proposed to delay the update of the policy network to first minimize the error in the value network before introducing a policy update. Therefore, the actor network is updated every d steps with respect to $Q_{\theta 1}$ as shown in Algorithm 1 **L17-L19**.

During the weight optimization of the actor and critic networks, the values of the existing sparse connections are only updated (i.e., the sparsity level is kept fixed). The sparse topologies of the networks are also optimized during training according to our proposed adaptation schedule.

Adaptation Schedule. The typical practice in DST methods applied in the supervised setting is to perform the dynamic adaptation of the sparse topology after each training epoch. However, this would not fit the RL setting directly due to its dynamic learning nature. In particular, an RL agent faces instability during training due to the lack of a true target objective. The agent learns through trial and error cycles, collecting the data online while interacting with the environment. Adapting the topology very frequently in this learning paradigm would limit the exploration of effective topologies for the data distribution and give a biased estimate of the current one. To address this point, we propose to delay the adaptation process and perform it every e time steps, where e is a hyperparameter. This would allow the newly added connections from the previous adaptation process to grow. Hence, it would also give better estimates of the connections with the least influence in the performance and an opportunity to explore other effective ones. Analysis of the effect of the adaptation schedule in the success of applying dynamic sparse training in the RL setting is provided in Section 4.2.

Algorithm 1 DS-TD3 (λ^l, η, e, N, τ, d)

1: Initialize critic networks $Q_{\boldsymbol{\theta_1}}$, $Q_{\boldsymbol{\theta_2}}$ and actor network $\pi_{\boldsymbol{\phi}}$ with sparse parameters $\boldsymbol{\theta_1}$, $\boldsymbol{\theta_2}$, $\boldsymbol{\phi}$ with a sparsity level defined by λ^l:
2: Create $\boldsymbol{M}_{\boldsymbol{\phi}}$, $\boldsymbol{M}_{\boldsymbol{\theta_1}}$, and $\boldsymbol{M}_{\boldsymbol{\theta_2}}$ with Erdős–Rényi graph
3: $\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} \odot \boldsymbol{M}_{\boldsymbol{\theta_1}}$, $\boldsymbol{\theta_2} \leftarrow \boldsymbol{\theta_2} \odot \boldsymbol{M}_{\boldsymbol{\theta_2}}$, $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} \odot \boldsymbol{M}_{\boldsymbol{\phi}}$
4: Initialize target networks $\boldsymbol{\theta}'_1 \leftarrow \boldsymbol{\theta_1}$, $\boldsymbol{\theta}'_2 \leftarrow \boldsymbol{\theta_2}$, $\boldsymbol{\phi'} \leftarrow \boldsymbol{\phi}$
5: Initialize replay buffer $\mathcal{B}$
6: **for** $t = 1$ **to** T **do**
7: Select action with exploration noise $a \sim \pi_{\boldsymbol{\phi}}(s) + \epsilon$,
8: $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
9: Store transition tuple (s, a, r, s') in $\mathcal{B}$
10: Sample mini-batch of N transitions from $\mathcal{B}$
11: $\tilde{a} \leftarrow \pi_{\boldsymbol{\phi'}}(s') + \epsilon, \quad \epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
12: $y \leftarrow r + \gamma \min_{i=1,2} Q_{\boldsymbol{\theta}'_i}(s', \tilde{a})$
13: $\boldsymbol{\theta_i} \leftarrow argmin_{\boldsymbol{\theta_i}} \frac{1}{N} \sum (y - Q_{\boldsymbol{\theta_i}}(s,a))^2$
14: **if** t mod e **then**
15: $\boldsymbol{\theta_i} \leftarrow$ TopologicalAdaptation($\boldsymbol{\theta_i}, \boldsymbol{M}_{\boldsymbol{\theta_i}}, \eta$) (Algo. 2)
16: **end if**
17: **if** t mod d **then**
18: Update $\boldsymbol{\phi}$ by the deterministic policy gradient:
19: $\nabla_{\boldsymbol{\phi}} J(\boldsymbol{\phi}) \leftarrow \frac{1}{N} \sum \nabla_a Q_{\boldsymbol{\theta_1}}(s,a)|_{a=\pi_{\boldsymbol{\phi}}(s)} \nabla_{\boldsymbol{\phi}} \pi_{\boldsymbol{\phi}}(s)$
20: **if** t mod e **then**
21: $\boldsymbol{\phi} \leftarrow$ TopologicalAdaptation($\boldsymbol{\phi}, \boldsymbol{M}_{\boldsymbol{\phi}}, \eta$) (Algo. 2)
22: **end if**
23: Update target networks:
24: $\boldsymbol{\theta}'_i \leftarrow \tau \boldsymbol{\theta_i} + (1-\tau)\boldsymbol{\theta}'_i$
25: $\boldsymbol{\phi'} \leftarrow \tau \boldsymbol{\phi} + (1-\tau)\boldsymbol{\phi'}$
26: $\boldsymbol{\theta}'_i \leftarrow$ MaintainSparsity($\boldsymbol{\theta}'_i, \|\boldsymbol{\theta_i}\|_0$) (Algo. 3)
27: $\boldsymbol{\phi'} \leftarrow$ MaintainSparsity($\boldsymbol{\phi'}, \|\boldsymbol{\phi}\|_0$) (Algo. 3)
28: **end if**
29: **end for**

Topological Adaptation (Algorithm 2). We adopt the adaptation strategy of the SET method [Mocanu *et al.*, 2018] in our approach. The sparse topologies are optimized according to our adaptation schedule. Every e steps, we update the sparse topology of the actor and critic networks. Here, we explain the adaptation process on the actor network as an example. The same strategy is applied for the critic networks.

The adaptation process is performed through a "drop-and-grow" cycle which consists of two steps. **The first step** is to *drop* a fraction η of the least important connections from each layer. This fraction is a subset (c_p) of the smallest positive weights and a subset (c_n) of the largest negative weights. Thus, the removed weights are the ones closest to zero. Let $\tilde{\phi}^p$ and $\tilde{\phi}^n$ be the c_p-th smallest positive and the c_n-th largest

Algorithm 2 Topological Adaptation ($\boldsymbol{X}$, $\boldsymbol{M}$, η)

1: $c \leftarrow \eta \, \|\boldsymbol{X}\|_0$
2: $c_p \leftarrow c/2$; $\quad c_n \leftarrow c/2$
3: $\tilde{\boldsymbol{X}}^p \leftarrow$ get_c_p-*th*_smallest_positive($\boldsymbol{X}$)
4: $\tilde{\boldsymbol{X}}^n \leftarrow$ get_c_n-*th*_largest_negative($\boldsymbol{X}$)
5: $\boldsymbol{M}^j \leftarrow \boldsymbol{M}^j - \mathbb{1}[(0 < \boldsymbol{X}^j < \tilde{\boldsymbol{X}}^p) \vee (0 > \boldsymbol{X}^j > \tilde{\boldsymbol{X}}^n)]$
6: Generate c random integers $\{x\}_1^c$
7: $\boldsymbol{M}^j \leftarrow \boldsymbol{M}^j + \mathbb{1}[(j == x) \wedge (\boldsymbol{X}^j == 0)]$
8: $\boldsymbol{X} \leftarrow \boldsymbol{X} \odot \boldsymbol{M}$

Algorithm 3 Maintain Sparsity ($\boldsymbol{X}$, k)

1: $\tilde{\boldsymbol{X}} \leftarrow$ Sort_Descending($|X|$)
2: $\boldsymbol{M}^j = \mathbb{1}[|\boldsymbol{X}^j| - \tilde{\boldsymbol{X}}^k \geq 0], \forall j \in \{1, ... \|\boldsymbol{X}\|_0\}$
3: $\boldsymbol{X} = \boldsymbol{X} \odot \boldsymbol{M}$

negative weights, respectively. The mask $\boldsymbol{M_\phi}$ is updated to represent the dropped connections as follows:

$$\boldsymbol{M_\phi^j} = \boldsymbol{M_\phi^j} - \mathbb{1}[(0 < \phi^j < \tilde{\phi}^p) \vee (0 > \phi^j > \tilde{\phi}^n)], \quad \forall j \in \{1, ..., \|\phi\|_0\}, \quad (4)$$

where $\boldsymbol{M_\phi^j}$ is the element j in $\boldsymbol{M_\phi}$, $\mathbb{1}$ is the indicator function, $\vee$ is the logical OR operator, and $\|.\|_0$ is the standard L_0 norm. **The second step** is to *grow* the same fraction η of removed weights in random locations from the non-existing weights in each layer. $\boldsymbol{M_\phi}$ is updated as follows:

$$\boldsymbol{M_\phi^j} = \boldsymbol{M_\phi^j} + \mathbb{1}[(j == x) \wedge (\phi^j == 0)], \quad \forall j \in \{1, .., \|\phi\|_0\}, \quad (5)$$

where x is a random integer generated from the discrete uniform distribution in the interval $[1, n^{(l-1)} \times (n^l)]$ and $\wedge$ is the logical AND operator. The weights of the newly added connections are zero-initialized ($\boldsymbol{\phi} = \boldsymbol{\phi} \odot \boldsymbol{M_\phi}$).

Maintain Sparsity Level in Target Networks (Algorithm 3). TD3 delays the update of the target networks to be performed every d steps. In addition, the target networks are slowly updated by some proportion τ instead of making the target networks exactly match the current ones (Algorithm 1 **L23-L25**). These two points lead to a slow deviation of the sparse topologies of the target networks from current networks. Consequently, the slow update of the target networks by τ would slowly increase the number of non-zero connections in the target networks over time. *To address this*, after each update of the target networks, we prune the extra connections that make the total number of connections exceed the initial defined one. We prune the extra weights based on their smallest magnitude. Assume we have to retain k connections. The target masks of the actor ($\boldsymbol{M'_{\phi'}}$) and critics ($\boldsymbol{M'_{\theta'_1}}$, $\boldsymbol{M'_{\theta'_2}}$) are calculated as follows:

$$\begin{aligned} \boldsymbol{M'^j_{\phi'}} &= \mathbb{1}[|\boldsymbol{\phi'}^j| - \tilde{\boldsymbol{\phi'}}^k \geq 0], \quad \forall j \in \{1, ..., \|\boldsymbol{\phi'}\|_0\}, \\ \boldsymbol{M'^j_{\theta'_i}} &= \mathbb{1}[|\boldsymbol{\theta'}_i^j| - \tilde{\boldsymbol{\theta'}}_i^k \geq 0], \quad \forall j \in \{1, ..., \|\boldsymbol{\theta'}_i\|_0\}, \quad \forall i \in \{1, 2\}, \end{aligned} \quad (6)$$

where $\tilde{\boldsymbol{\phi'}}^k$ and $\tilde{\boldsymbol{\theta'}}_i^k$ is the k-th largest magnitude in the actor and critics respectively, and $|(.)^j|$ is the magnitude of element j in the matrix. The target networks are updated as follows:

$$\begin{aligned} \boldsymbol{\phi'} &= \boldsymbol{\phi'} \odot \boldsymbol{M'_{\phi'}}, \\ \boldsymbol{\theta'_i} &= \boldsymbol{\theta'_i} \odot \boldsymbol{M'_{\theta'_i}} \quad \forall i \in \{1, 2\}. \end{aligned} \quad (7)$$

Environment	TD3	Static-TD3	DS-TD3 (ours)	SAC
HalfCheetah-v3	1.7686	1.7666	**1.9560**	1.7297
Walker2d-v3	0.5264	0.5167	**0.6956**	0.6128
Hopper-v3	0.4788	0.4984	0.5435	**0.5572**
Ant-v3	0.5524	0.5807	0.6623	**0.7969**
Humanoid-v3	0.3635	0.5182	**0.6089**	0.5639

Table 1: Learning curve area (LCA) ($\times$ 5000) of different methods.

4 Experiments and Results

In this section, we assess the efficiency of our proposed dynamic sparse training approach for the DRL paradigm and compare it to state-of-the-art algorithms. Experimental settings are provided in the extended version of the paper.

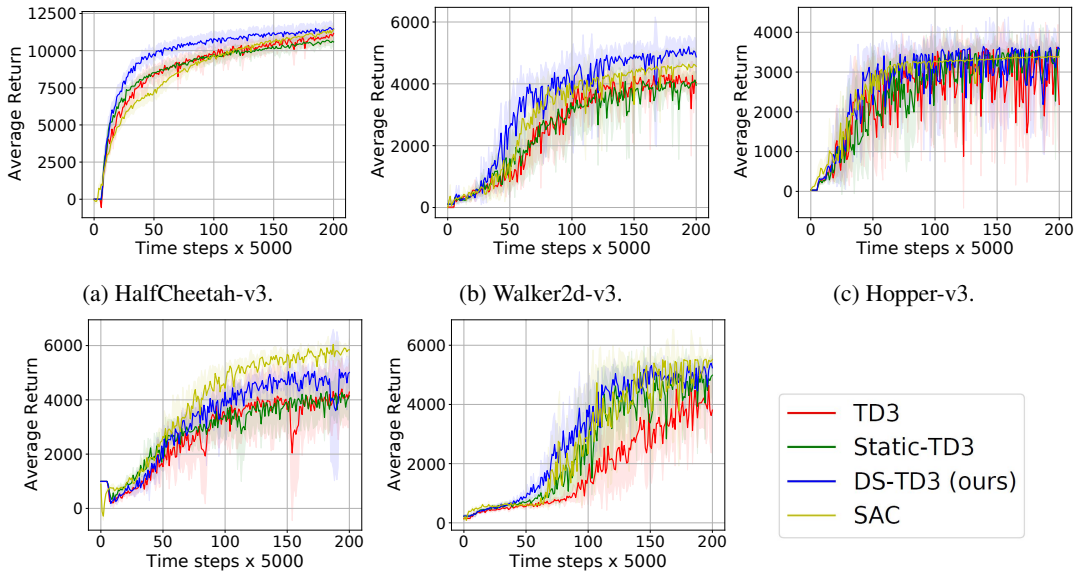
Baselines. We compare our proposed DS-TD3 against the following baselines: (1) *TD3* [Fujimoto *et al.*, 2018], the original TD3 where dense networks are used for actor and critic models, (2) *Static-TD3*, a variant of TD3 where the actor and critic models are initialized with sparse neural networks which are kept fixed during training (i.e., there is no topological optimization), and (3) *SAC* [Haarnoja *et al.*, 2018b], a popular off-policy algorithm in which the policy is trained to maximize a trade-off between expected return and entropy which results in policies that explore better.

Benchmarks. We performed our experiments on MuJoCo continuous control tasks, interfaced through OpenAI Gym. We evaluate our proposed approach on five challenging environments (HalfCheetah-v3, Hopper-v3, Walker2d-v3, Ant-v3, and Humanoid-v3).

Metrics. We use multiple metrics to assess the efficiency of the studied DRL methods: **(1) Return** which is the standard metric used in DRL to measure the *performance* of an agent, **(2) Learning curve area (LCA)** which estimates the *learning speed* of a model (i.e., how quickly a model learns) [Chaudhry *et al.*, 2018] by measuring the area under the training curve of a method, **(3) Network size (#params)** which measures the *memory cost* via the number of network parameters, and **(4) Floating-point operations (FLOPs)** which estimate the *computational cost* required for training. It is the typical metric in the literature to compare a DST method against its dense counterpart. Details are in the extended version of the paper.

4.1 Results

Learning Behavior and Speed. Figure 1 shows the learning curve of studied methods. DS-TD3 has a much faster learning speed than the baselines, especially at the beginning of the training. After 40-50% of the steps, DS-TD3 can achieve the final performance of TD3. Static-TD3 does not have this favorable property which reveals the importance of optimizing the sparse topology during training to adapt to the incoming data. The learning behavior of DS-TD3 is also faster than SAC in all environments except one. Table 1 shows the learning curve area (LCA) of each method. DS-TD3 has higher LCA than TD3 and static-TD3 in all environments. It is also higher than SAC in three environments out of five. This metric is important to differentiate between two agents with similar final performance but very different LCA.



(a) HalfCheetah-v3. (b) Walker2d-v3. (c) Hopper-v3.

(d) Ant-v3. (e) Humanoid-v3.

Figure 1: Learning curves of the studied algorithms on different continuous control tasks. The shaded region represents the standard deviation of the average evaluation over 5 runs.

Method	HalfCheetah-v3	Walker2d-v3	Hopper-v3	Ant-v3	Humanoid-v3
TD3	11153.48±473.29	4042.36±576.57	2184.78±1224.14	4287.69±1080.88	3809.15±1053.40
Static-TD3	10583.84±307.03	3951.01±443.78	3570.88±43.71	4148.61±801.34	4989.47±546.32
DS-TD3 (ours)	**11459.88±482.55**	**4870.57±525.33**	**3587.17±70.62**	5011.56±596.95	5238.16±121.71
SAC	11415.23±357.22	4566.18±448.25	3387.36±148.73	**5848.64±385.85**	**5518.61±97.03**

Table 2: Average return (R) over the last 10 evaluations of 1 million time steps.

Performance. Table 2 shows the average return (R) over the last 10 evaluations. DS-TD3 outperforms TD3 in all environments. Interestingly, it improves TD3 performance by 2.75%, 20.48%, 64.18%, 16.88%, and 37.51% on HalfCheetah-v3, Walker2d-v3, Hopper-v3, Ant-v3, and Humanoid-v3 respectively. Static-TD3 has a close performance to TD3 in most cases except for Humanoid-v3, where Static-TD3 outperforms TD3 by 30.98%. DS-TD3 has a better final performance than SAC in three environments.

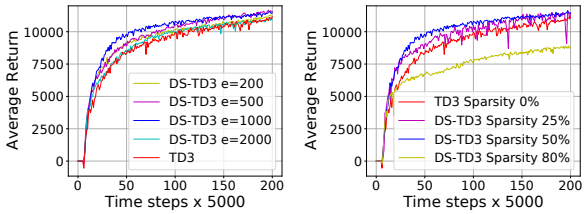
4.2 Analysis

Memory and Computation Costs

We analyze the costs needed for the training process by calculating the FLOPS and #params for the actor and critics. We performed this analysis on Half-Cheetah-v3. #params for dense TD3 is 214784, which requires $1\times(1.07e14)$ FLOPs to train. With our DS-TD3, we can find a much smaller topology that can effectively learn the policy and the function value, achieving higher performance than TD3 with a sparsity level of 51%. This consequently reduces the number of required FLOPs to $0.49\times(1.07e14)$.

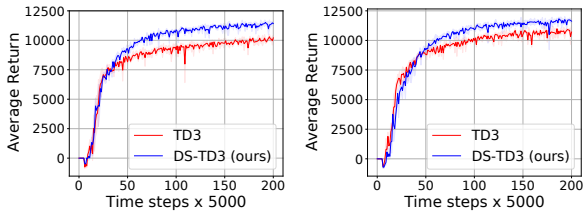
Adaptation Schedule

We analyze the effect of the adaptation schedule on the performance. In particular, we ask how frequently the sparse topology should be adapted? We performed this analysis on HalfCheetah-v3. Figure 2a shows the learning curves of DS-TD3 using different adaptation schedules controlled by the hyperparameter e (Section 3). Adapting the topology very frequently ($e \in \{200, 500\}$) would not allow the connections to grow and learn in the dynamic changing nature of RL. The current adaptation process could remove some promising newly added connections from the past adaptation process. This would be caused by a biased estimate of a connection's importance as it becomes a factor of the length of its lifetime. Hence, the very frequent adaptation would increase the chance of replacing some promising topologies. With less frequent adaptation cycles, $e = 1000$ (the setting from the paper), DS-TD3 can learn faster and eventually achieves higher performance than other baselines. Giving the connections a chance to learn helps in having better estimates of the importance of the connections. Hence, it enables finding more effective topologies by replacing the least effective connections with ones that better fit the data. However, increasing the gap between every two consecutive adaptation processes to 2000 steps decreases the exploration speed of different topologies. As illustrated in the figure, DS-TD3 with $e = 2000$ has a close learning behavior and final performance to TD3. Yet, it still offers a substantial reduction in memory and computation costs. This analysis reveals the importance of the adaptation schedule in the success of introducing DST to the DRL field.



(a) (b)

Figure 2: The learning curves of DS-TD3 on HalfCheetah-v3 using different adaptation schedules (a) and sparsity levels (b).



(a) (b)

Figure 3: Learning curves of agents that start training with samples drawn from policies trained for 5×10^5 (a) and 7×10^5 steps (b).

Sparsity Level

We analyze the performance of our proposed method using different sparsity levels. Figure 2b shows the learning curves of the dense TD3 and DS-TD3. By removing 25% of the connections and training the sparse topology dynamically using DS-TD3, we can achieve a faster learning speed and a performance increase of 2.11%. More interestingly, with a higher reduction in the size of the networks by 50%, we achieve a much faster learning speed. However, when the network has a very high sparsity level (i.e., 80%), it fails to learn effective representations for the reinforcement learning setting. Learning DRL agents using very high sparse networks is still an open-challenging task.

Learning Behavior and Speed

DRL agents learn through trial-and-error due to the lack of true labels. An agent starts training with samples generated from a purely exploratory policy, and new samples are drawn from the learning policy over time. Our results show that dynamic sparse agents have faster adaptability to the newly improved samples, thanks to the generalization ability of sparse neural networks [Hoefler *et al.*, 2021]. This leads to higher learning speed, especially at the beginning of the training. We hypothesize that dense neural networks, being over-parameterized, are more prone to memorize and overfit the inaccurate samples. A longer time is required to adapt to the newly added samples by the improved policy and forget the old ones.

To validate this hypothesis, we analyze the behavior of a dense TD3 agent when it starts training with samples generated from a learned policy instead of a purely exploratory one. We use two learned policies trained for 5×10^5 and 7×10^5 steps to draw the initial samples (see the extended version of the paper). We performed this experiment on HalfCheetah-v3. As illustrated in Figure 3, the learning speed of DS-TD3 and TD3 becomes close to each other at the beginning. Afterward, DS-TD3 performs better than TD3 since the new samples are generated from the current learning policies. With initial samples drawn from more improved policy (Figure 3b), dense TD3 learns faster. It achieves higher performance than the dense baseline that starts learning with samples drawn from the policy trained for 5×10^5 steps (Figure 3a). On the other hand, DS-TD3 is more robust to over-fitting, less affected by the initial samples, and quickly adapt to the improved ones over time.

5 Conclusion

Introducing dynamic sparse training principles to the deep reinforcement learning field provides an efficient training process for DRL agents. Our dynamic sparse agents achieve higher performance than the state-of-the-art methods while reducing the memory and computation costs by 50%. Optimizing the sparse topology during training to adapt to the incoming data increases the learning speed. Our findings show the potential of dynamic sparse training in advancing the DRL field. This would open the path to efficient DRL agents that could be trained and deployed on low-resource devices where memory and computation are strictly constrained.

References

[Chaudhry *et al.*, 2018] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations*, 2018.

[Denil *et al.*, 2013] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume 2*, pages 2148–2156, 2013.

[Erdos *et al.*, 1960] Paul Erdos, Alfréd Rényi, et al. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[Evci *et al.*, 2020] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

[Frankle and Carbin, 2018] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

[Fujimoto *et al.*, 2018] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[Haarnoja *et al.*, 2018a] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[Haarnoja *et al.*, 2018b] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[Heidrich-Meisner and Igel, 2009] Verena Heidrich-Meisner and Christian Igel. Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, 64(4):152–168, 2009. Special Issue: Reinforcement Learning.

[Hoefler *et al.*, 2021] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

[Jayakumar *et al.*, 2020] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. *Advances in Neural Information Processing Systems*, 33:20744–20754, 2020.

[Jouppi *et al.*, 2017] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.

[Lee *et al.*, 2021] Juhyoung Lee, Sangyeob Kim, Sangjin Kim, Wooyoung Jo, and Hoi-Jun Yoo. Gst: Group-sparse training for accelerating deep reinforcement learning. *arXiv preprint arXiv:2101.09650*, 2021.

[Liu *et al.*, 2020] Shiwei Liu, Decebal Constantin Mocanu, Amarsagar Reddy Ramapuram Matavalam, Yulong Pei, and Mykola Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33:2589–2604, 2020.

[Liu *et al.*, 2021] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6893–6904. PMLR, 18–24 Jul 2021.

[Livne and Cohen, 2020] Dor Livne and Kobi Cohen. Pops: Policy pruning and shrinking for deep reinforcement learning. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):789–801, 2020.

[Mocanu *et al.*, 2018] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

[Mocanu *et al.*, 2021] Decebal Constantin Mocanu, Elena Mocanu, Tiago Pinto, Selima Curci, Phuong H Nguyen, Madeleine Gibescu, Damien Ernst, and Zita A Vale. Sparse training theory for scalable and efficient agents. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 34–38, 2021.

[Mostafa and Wang, 2019] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pages 4646–4655. PMLR, 2019.

[Shi *et al.*, 2020] Qian Shi, Hak-Keung Lam, Chengbin Xuan, and Ming Chen. Adaptive neuro-fuzzy pid controller based on twin delayed deep deterministic policy gradient algorithm. *Neurocomputing*, 402:183–194, 2020.

[Sokar *et al.*, 2021] Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Spacenet: Make free space for continual learning. *Neurocomputing*, 439:1–11, 2021.

[Stanley and Miikkulainen, 2002] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[Stanley, 2003] Kenneth O. Stanley. Evolving adaptive neural networks with and without adaptive synapses. In *In Proceeedings of the 2003 Congress on Evolutionary Computation (CEC 2003*, pages 2557–2564. Press, 2003.

[Vinyals *et al.*, 2019] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[Vischer *et al.*, 2021] Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. On lottery tickets and minimal task representations in deep reinforcement learning. *arXiv preprint arXiv:2105.01648*, 2021.

[Wang *et al.*, 2020] Hao-nan Wang, Ning Liu, Yi-yun Zhang, Da-wei Feng, Feng Huang, Dong-sheng Li, and Yi-ming Zhang. Deep reinforcement learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, pages 1–19, 2020.

[Whiteson and Stone, 2006] Shimon Whiteson and Peter Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7:877–917, May 2006.

[Woo *et al.*, 2020] Jong Ha Woo, Lei Wu, Jong-Bae Park, and Jae Hyung Roh. Real-time optimal power flow using twin delayed deep deterministic policy gradient algorithm. *IEEE Access*, 8:213611–213618, 2020.

[Yu *et al.*, 2019] Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. *arXiv preprint arXiv:1906.02768*, 2019.

[Zhang *et al.*, 2019] Hongjie Zhang, Zhuocheng He, and Jing Li. Accelerating the deep reinforcement learning with neural network compression. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2019.