Dynamic Sparse Training for Deep Reinforcement Learning

Ghada Sokar
Eindhoven University of Technology
The Netherlands
g.a.z.n.sokar@tue.nl

Elena Mocanu
University of Twente
The Netherlands
e.mocanu@utwente.nl

Decebal Constantin Mocanu
University of Twente, Eindhoven
University of Technology
The Netherlands
d.c.mocanu@utwente.nl

Mykola Pechenizkiy
Eindhoven University of Technology
The Netherlands
m.pechenizkiy@tue.nl

Peter Stone
University of Texas at Austin, Sony AI
United States
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, providing 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].

KEYWORDS

Deep Reinforcement Learning, Dynamic Sparse Training, Sparse Neural Networks

1 INTRODUCTION

Deep reinforcement learning (DRL) has achieved remarkable success in many applications [29]. The power of deep neural networks as function approximators [6, 10, 15, 19, 30, 45, 67] 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 [1, 14, 16, 36, 47, 48, 50, 55, 56, 59]. 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) [57]. The very long training time of DRL agents 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?*

[1]Code available at: https://github.com/GhadaSokar/Dynamic-Sparse-Training-forDeep-Reinforcement-Learning.

Proc. of the Adaptive and Learning Agents Workshop (ALA 2022), Cruz, Hayes, da Silva, Santos (eds.), May 9-10, 2022, Online, https://ala2022.github.io/. 2022.

Few recent works attempt to accelerate the *inference* time of DRL agents via pruning [35] or training a compact network under the guidance of a larger network (knowledge distillation) [64]. 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 [27] and **(2)** the learning nature of a DRL agent in which its policy is improving through many cycles of trial and error while interacting with the environment and collecting a large amount of data. In this paper, we introduce dynamic sparse training (DST) [21, 37] in the DRL paradigm for the first time to address these two factors. 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 proposed training approach enables reducing memory and computation costs substantially. In addition, the quick adaptation of our dynamic sparse agents 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 [51]. Although prior works related to the automatic selection of function approximation based on neuroevolution exist [20], perhaps the most connected in the spirit to our proposed method is a combination between NeuroEvolution of Augmenting Topologies (NEAT) [52] and temporal difference (TD) learning (i.e., NEAT+Q [60]). 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 in the deep reinforcement learning field for the first time.
- Efficient improved versions of two state-of-the-art algorithms, TD3 [14] and SAC [16], are obtained by integrating our proposed dynamic sparse training 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. See Section 5 for discussion.

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 [35] 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 [64], 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 [28] was proposed as an algorithm for weight compression in DRL training by simultaneously utilizing weight grouping and pruning. Some other works [58, 63] studied the existence of the lottery ticket hypothesis [13] 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 [7, 18, 24, 39, 40, 42, 46]. 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 over-parametrized) [8]. Efforts in this line of research are devoted to supervised and unsupervised learning. The first work in this direction was proposed by [38]. They proposed a Sparse Evolutionary Training algorithm (SET) that dynamically changes the sparse connectivity during training based on the values of the connections. The method achieves higher performance than dense models and static sparse neural 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 [3, 9, 12, 25, 33, 41, 44]. DST demonstrated its success in other fields as well, such as feature selection [2], ensembling [31], federated learning [66], text classification and language modeling tasks [34], and adversarial training [43].

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 [32]. 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 the DRL paradigm. 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) [14]. We named our new approach Dynamic Sparse TD3 or "DS-TD3" for short. TD3 is a popular and efficient DRL method that offers good performance in many tasks [23, 26, 49, 61, 62]. Yet, our approach can be merged into other DRL algorithms as well. Appendix A shows the integration with soft actor-critic (SAC) [16].

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 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 ($Q_{\theta_1}, Q_{\theta_2}$) 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 $\boldsymbol{M} = \{\boldsymbol{M}^l\}|_{l=1}^L$. We use Erdős–Rényi random graph [11] to initialize a sparse topology in each layer l since it shows superiority over uniform distribution [12]. Namely, the probability of a connection j in layer l is given by:

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

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

1: Initialize critic networks Q_{θ_1}, Q_{θ_2} and actor network π_{ϕ} with sparse parameters $\boldsymbol{\theta_1}$, $\boldsymbol{\theta_2}$, $\boldsymbol{\phi}$ with a sparsity level defined by λ^l:
2: Create $\boldsymbol{M_\phi}$, $\boldsymbol{M_{\theta_1}}$, and $\boldsymbol{M_{\theta_2}}$ with Erdős–Rényi graph
3: $\boldsymbol{\theta_1} \leftarrow \boldsymbol{\theta_1} \odot \boldsymbol{M_{\theta_1}}$, $\boldsymbol{\theta_2} \leftarrow \boldsymbol{\theta_2} \odot \boldsymbol{M_{\theta_2}}$, $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} \odot \boldsymbol{M_\phi}$
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($\boldsymbol{\theta_i}, \boldsymbol{M_{\theta_i}}, \eta$) (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($\boldsymbol{\phi}, \boldsymbol{M_\phi}, \eta$) (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**

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:

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

where $\odot$ is an element-wise multiplication operator and $\boldsymbol{M_\phi}$, $\boldsymbol{M_{\theta_1}}$, and $\boldsymbol{M_{\theta_2}}$ 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 $\pi_{\boldsymbol{\phi'}}$:

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

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: $\boldsymbol{M}^j \leftarrow \boldsymbol{M}^j - \mathbb{1}[(\boldsymbol{X}^j < \tilde{\boldsymbol{X}}^p) \vee (0 > \boldsymbol{X}^j > \tilde{\boldsymbol{X}}^n)]$
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($|\boldsymbol{X}|$)
2: $\boldsymbol{M}^j = \mathbb{1}[|\boldsymbol{X}^j| - \tilde{\boldsymbol{X}}^k \geq 0], \forall j \in \{1, \ldots \|\boldsymbol{X}\|_0\}$
3: $\boldsymbol{X} = \boldsymbol{X} \odot \boldsymbol{M}$

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 4.6.

Topological Adaptation (Algorithm 2). We adopt the adaptation strategy of the SET method [38] 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 $\tilde{\boldsymbol{\phi}}^p$ and $\tilde{\boldsymbol{\phi}}^n$ be the c_p-th smallest positive and the c_n-th largest negative weights, respectively. The mask $\boldsymbol{M_\phi}$ is updated to represent the dropped connections as follows:

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

where $\boldsymbol{M}_{\boldsymbol{\phi}}^j$ 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:

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

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 ($\boldsymbol{M}'_{\boldsymbol{\phi}'}$) and critics ($\boldsymbol{M}'_{\boldsymbol{\theta}'_1}$, $\boldsymbol{M}'_{\boldsymbol{\theta}'_2}$) are calculated as follows:

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

where $\tilde{\boldsymbol{\phi}}'^k$ and $\tilde{\boldsymbol{\theta}}'^k_i$ 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:

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

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.

4.1 Baselines

We compare our proposed DS-TD3 against the following baselines: (1) *TD3* [14], 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* [17], 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.

4.2 Benchmarks

We performed our experiments on MuJoCo continuous control tasks [54], interfaced through OpenAI Gym [4]. We evaluate our proposed approach on five challenging environments (HalfCheetah-v3, Hopper-v3, Walker2d-v3, Ant-v3, and Humanoid-v3).

4.3 Metrics

We use multiple metrics to assess the efficiency of the studied DRL methods:

- **Return (R).** The average return is the standard metric used in the RL research to measure the *performance* of an agent. The return is the sum of rewards (r) obtained in one episode of T steps. R is calculated as follows:

$$R = \sum_{t=1}^{T} r_t. \quad (8)$$

- **Learning curve area (LCA).** This metric estimates the learning speed of a model. LCA measures the area under the training curve of a method (i.e., how quickly a model learns). Intuitively, the higher learning curve, the faster the learner is. We adapt this metric from [5] to fit the reinforcement learning paradigm. LCA is calculated as follows:

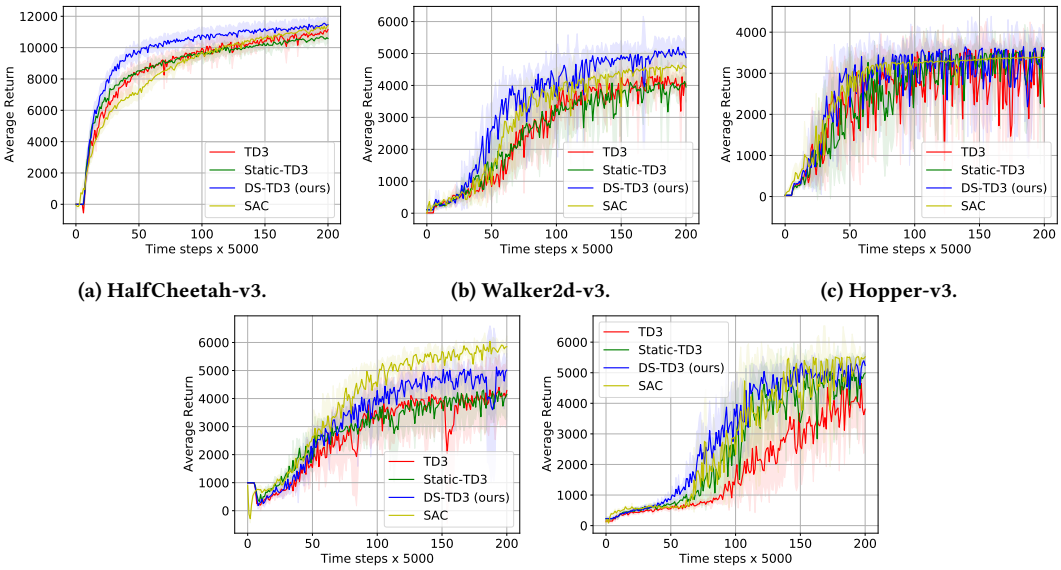
$$LCA = \frac{1}{\Delta} \int_0^{\Delta} R(t)dt = \frac{1}{\Delta} \sum_{t=0}^{\Delta} R(t), \quad (9)$$

  where Δ is the number of training steps and R is the average return.
- **Network size (#params).** This metric estimates the *memory cost* consumed by an agent. The network size is estimated by the summation of the number of connections allocated in its layers as follows:

$$\#params = \sum_{l=1}^{L} \left\| \boldsymbol{W}^l \right\|_0, \quad (10)$$

  where $\boldsymbol{W}^l$ is the actual weights used in layer l, $\|.\|_0$ is the standard L_0 norm, and L is the number of layers in the model. For sparse neural networks, $\left\| \boldsymbol{W}^l \right\|_0$ is controlled by its defined sparsity level for the model.
- **Floating-point operations (FLOPs).** This metric estimates the *computational cost* of a method by calculating how many FLOPs are required for training. We follow the method described in [12] to calculate the FLOPs. The FLOPs are calculated with the total number of multiplications and additions layer by layer in the network.

FLOPs is the typical used metric in the literature to compare a DST method against its dense counterpart. The motivation is twofold. First, it gives an unbiased estimate of the actual required number



(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.

of operations since the running time would differ from one implementation to another. Second, more importantly, existing dynamic sparse training methods in the literature are currently prototyped using masks over dense weights to simulate sparsity [21]. This is because most deep learning specialized hardware is optimized for dense matrix operations. Therefore, the running time using these prototypes would not reflect the actual gain in memory and speed using a truly sparse network. Hence, the FLOPs and network parameters are the current commonly used metrics to estimate the computation and memory costs respectively for sparse neural networks [21].

4.4 Experimental Settings

For a direct comparison with TD3, we follow the same setting as in [14]. We use multi-layer perceptrons for the actor and critics networks with two hidden layers of 256 neurons and a ReLU activation function. A Tanh activation is applied to the output layer of the actor network. Sparse connections are allocated in the first two layers for all networks while the output layer is dense. We use λ^2 of 64 for all environments. In contrast, λ^1 varies across environments because it depends on each environment's state and action dimensions. We use λ^1 of 7 for HalfCheetah-v3, Hopper-v3, and Walker2d-v3. For Ant-v3 and Humanoid-v3, we use λ^1 of 40 and 61, respectively. The same sparsity levels are used for Static-TD3.

We adapt the sparse connections every e time steps, with $e = 1000$. A fraction of the sparse connections η is adapted with $\eta = 0.05$. The networks are trained using Adam optimizer with a learning rate of 0.001 and a weight decay of 0.0002. The networks are trained with mini-batches (N) of 100, sampled uniformly from a replay buffer containing the entire history of the agent.

Following the TD3 algorithm [14], we added noise of $\epsilon \sim \mathcal{N}(0, 0.2)$ to the actions chosen by the target actor network and clipped to $(-0.5, 0.5)$. The actor and target networks are updated every 2 steps ($d = 2$). The τ used for updating the target networks equals 0.005. A purely exploratory policy is used for the first 25000 time steps, then an off-policy exploration strategy is used with Gaussian noise of $\mathcal{N}(0, 0.1)$ added to each action.

The hyperparameters for the dynamic sparse training $(\lambda^1, \lambda^2, \eta, e)$ are selected using random search. Each environment is run for 1 million time steps with evaluations every 5000 time steps, where each evaluation reports the average return over 10 episodes with no exploration noise. LCA is calculated using the average return computed every 5000 time steps. Our results are reported over 5 seeds.

All models are implemented with PyTorch and trained on Nvidia GPUs. We use the official code of TD3 [14], which has an MIT license, to reproduce the results of TD3 with the above settings.

For SAC, we use the Pytorch implementation from [53]. We follow the settings from the original paper [16] with the same architecture used for TD3. The networks are trained using Adam optimizer with a learning rate of 0.0003 and mini-batches of 256. We use τ of 0.005 and a target update interval of 1. The models are trained for 1M steps.

4.5 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

Table 1: Learning curve area (LCA) (↑) of different DRL methods.

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

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

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.

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.

4.6 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×(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×.

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 2 shows the learning curves of DS-TD3 using different adaptation schedules controlled by the hyperparameter e (Section 3). Adapting the topology very frequently (i.e., $e \in \{200, 500\}$) 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 previous adaptation process. This would be caused by a

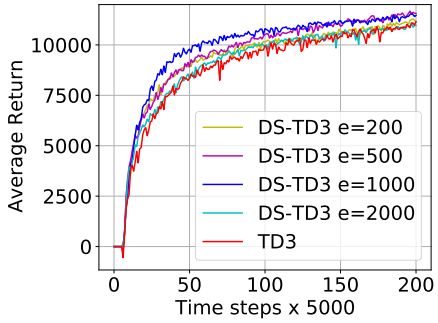


Figure 2: The learning curves of DS-TD3 on HalfCheetah-v3 using different adaptation schedules.

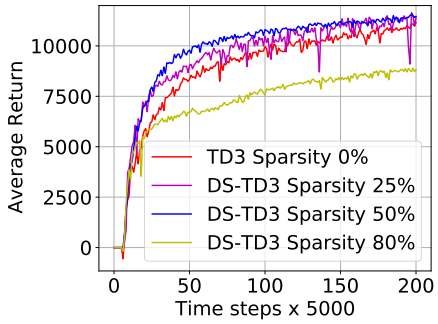


Figure 3: The learning curves of DS-TD3 on HalfCheetah-v3 using different sparsity levels.

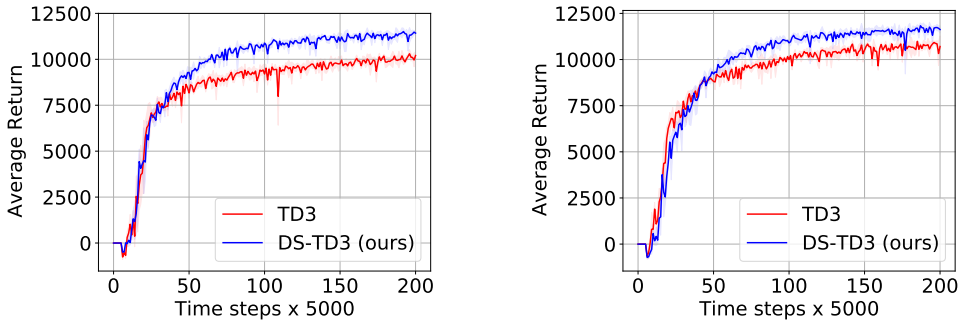


Figure 4: Learning curves of agents that start training with samples drawn from policies trained for 5×10^5 (a) and 7×10^5 steps (b) on HalfCheetah-v3.

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 the dense 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.

Sparsity Level. We analyze the performance of our proposed method using different sparsity levels. Figure 3 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 [21]. 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 performed this analysis on HalfCheetah-v3. We test two learned policies with different performance to study how the quality of the initial samples affects the learning behavior. To this end, we train two dense policies using TD3 for 5×10^5 and 7×10^5 steps on Half-Cheetah-v3. Similarly, we train two sparse policies for the same time steps using DS-TD3. Instead of using a purely exploratory policy, we draw samples from the learned policies to fill the initial buffers for dense and dynamic sparse agents that learn from scratch.

As illustrated in Figure 4, 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 4b), dense TD3 learns faster. It achieves higher performance than the baseline that starts learning with samples drawn from the policy trained for 5×10^5 steps (Figure 4a). This reveals that the performance of dense DRL agents is more affected by the initial samples. The better the samples are, the higher performance is. On the other hand, DS-TD3 is more robust to over-fitting, less affected by the quality of the initial samples, and quickly adapt to the improved ones over time.

5 DISCUSSION ON HARDWARE AND SOFTWARE SUPPORT

As a joint community effort, research on sparsity is going into three parallel directions: First, hardware that supports sparsity. NVIDIA released NVIDIA A100, which supports a 50% fixed sparsity level [65]. Second, software libraries that support truly sparse implementations. Efforts have been started to be devoted to supervised learning [32]. Third, algorithmic methods, our focus, that aim to provide approaches that achieve the same performance of dense models using sparse networks [21]. With the parallel efforts in the three directions, we would be able to actually provide faster, memory-efficient, and energy-efficient deep neural networks. This is further discussed in [22, 37].

6 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

[1] Oron Anschel, Nir Baram, and Nahum Shimkin. 2017. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International Conference on Machine Learning*. PMLR, 176–185.

[2] Zahra Atashgahi, Ghada Sokar, Tim van der Lee, Elena Mocanu, Decebal Constantin Mocanu, Raymond Veldhuis, and Mykola Pechenizkiy. 2020. Quick and Robust Feature Selection: the Strength of Energy-efficient Sparse Training for Autoencoders. *arXiv preprint arXiv:2012.00560* (2020).

[3] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. 2018. Deep Rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[5] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2018. Efficient Lifelong Learning with A-GEM. In *International Conference on Learning Representations*.

[6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2017. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2017), 834–848.

[7] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. 2021. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16306–16316.

[8] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. 2013. Predicting parameters in deep learning. *arXiv preprint arXiv:1306.0543* (2013).

[9] Tim Dettmers and Luke Zettlemoyer. 2019. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840* (2019).

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

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

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

[13] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

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

[15] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

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

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

[18] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. *Advances in Neural Information Processing Systems* 28 (2015).

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. 1026–1034.

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

[21] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research* 22, 241 (2021), 1–124. http://jmlr.org/papers/v22/21-0366.html

[22] Sara Hooker. 2021. The hardware lottery. *Commun. ACM* 64, 12 (2021), 58–65.

[23] Yangyang Hou, Huajie Hong, Zhaomei Sun, Dasheng Xu, and Zhe Zeng. 2021. The Control Method of Twin Delayed Deep Deterministic Policy Gradient with Rebirth Mechanism to Multi-DOF Manipulator. *Electronics* 10, 7 (2021), 870.

[24] Steven A Janowsky. 1989. Pruning versus clipping in neural networks. *Physical Review A* 39, 12 (1989), 6600.

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

[26] Tanuja Joshi, Shikhar Makker, Hariprasad Kodamana, and Harikumar Kandath. 2021. Application of twin delayed deep deterministic policy gradient learning for the control of transesterification process. *arXiv preprint arXiv:2102.13012* (2021).

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

[28] Juhyoung Lee, Sangyeob Kim, Sangjin Kim, Wooyoung Jo, and Hoi-Jun Yoo. 2021. GST: Group-Sparse Training for Accelerating Deep Reinforcement Learning. *arXiv preprint arXiv:2101.09650* (2021).

[29] Yuxi Li. 2017. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[30] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2117–2125.

[31] Shiwei Liu, Tianlong Chen, Zahra Atashgahi, Xiaohan Chen, Ghada Sokar, Elena Mocanu, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. 2021. Deep Ensembling with No Overhead for either Training or Testing: The All-Round Blessings of Dynamic Sparsity. *arXiv preprint arXiv:2106.14568* (2021).

[32] Shiwei Liu, Decebal Constantin Mocanu, Amarsagar Reddy Ramapuram Matavalam, Yulong Pei, and Mykola Pechenizkiy. 2020. Sparse evolutionary Deep Learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications* 33 (2020), 2589–2604.

[33] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. 2021. Selfish Sparse RNN Training. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 6893–6904.

[34] Shiwei Liu, Iftitahu Ni'mah, Vlado Menkovski, Decebal Constantin Mocanu, and Mykola Pechenizkiy. 2021. Efficient and effective training of sparse recurrent neural networks. *Neural Computing and Applications* (2021), 1–12.

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

[36] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[37] Decebal Constantin Mocanu, Elena Mocanu, Tiago Pinto, Selima Curci, Phuong H Nguyen, Madeleine Gibescu, Damien Ernst, and Zita A Vale. 2021. Sparse Training Theory for Scalable and Efficient Agents. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. 34–38.

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

[39] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11264–11272.

[40] P Molchanov, S Tyree, T Karras, T Aila, and J Kautz. 2019. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*.

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

[42] Michael C Mozer and Paul Smolensky. 1989. Using relevance to reduce network size automatically. *Connection Science* 1, 1 (1989), 3–16.

[43] Ozan Özdenizci and Robert Legenstein. 2021. Training adversarially robust sparse networks via Bayesian connectivity sampling. In *International Conference on Machine Learning*. PMLR, 8314–8324.

[44] Md Aamir Raihan and Tor Aamodt. 2020. Sparse Weight Activation Training. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 15625–15638. https://proceedings.neurips.cc/paper/2020/file/

b44182379bf9fae976e6ae5996e13cd8-Paper.pdf

[45] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. 2017. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225* (2017).

[46] Alex Renda, Jonathan Frankle, and Michael Carbin. 2020. Comparing rewinding and fine-tuning in neural network pruning. *arXiv preprint arXiv:2003.02389* (2020).

[47] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. PMLR, 1889–1897.

[48] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (2017).

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

[50] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

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

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

[53] Pranjal Tandon. 2018. PyTorch implementation of soft actor critic. https://github.com/pranz24/pytorch-soft-actor-critic

[54] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 5026–5033.

[55] Manan Tomar, Amy Zhang, Roberto Calandra, Matthew E Taylor, and Joelle Pineau. 2021. Model-Invariant State Abstractions for Model-Based Reinforcement Learning. *arXiv preprint arXiv:2102.09850* (2021).

[56] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

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

[58] Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. 2021. On Lottery Tickets and Minimal Task Representations in Deep Reinforcement Learning. *arXiv preprint arXiv:2105.01648* (2021).

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

[60] Shimon Whiteson and Peter Stone. 2006. Evolutionary Function Approximation for Reinforcement Learning. *Journal of Machine Learning Research* 7 (May 2006), 877–917.

[61] Jong Ha Woo, Lei Wu, Jong-Bae Park, and Jae Hyung Roh. 2020. Real-Time Optimal Power Flow Using Twin Delayed Deep Deterministic Policy Gradient Algorithm. *IEEE Access* 8 (2020), 213611–213618.

[62] Yujian Ye, Dawei Qiu, Huiyu Wang, Yi Tang, and Goran Strbac. 2021. Real-Time Autonomous Residential Demand Response Management Based on Twin Delayed Deep Deterministic Policy Gradient Learning. *Energies* 14, 3 (2021), 531.

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

[64] Hongjie Zhang, Zhuocheng He, and Jing Li. 2019. Accelerating the Deep Reinforcement Learning with Neural Network Compression. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[65] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. 2020. Learning N: M Fine-grained Structured Sparse Neural Networks From Scratch. In *International Conference on Learning Representations*.

[66] Hangyu Zhu and Yaochu Jin. 2019. Multi-objective evolutionary federated learning. *IEEE Transactions on Neural Networks and Learning Systems* 31, 4 (2019), 1310–1322.

[67] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.

A DS-SAC

In this appendix, we demonstrate that our proposed dynamic sparse training approach can be integrated with other state-of-the-art DRL methods. We use the soft actor-critic (SAC) method [17] and name our improved version of it as Dynamic Sparse SAC or DS-SAC.

SAC is an off-policy algorithm that optimizes a stochastic policy. A key feature of this method is entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy (a measure of randomness). Thus, the agent addresses the exploration-exploitation trade-off, which results in policies that explore better. Algorithm 4 shows our proposed DS-SAC. We integrated the four components of our approach (sparse topology initialization, adaptation schedule, topological adaptation, and maintain sparsity levels) into the original algorithm.

Tasks. We compared our proposed DS-SAC with SAC. We performed our experiments on five MuJoCo control tasks. Namely, we tested the following environments: HalfCheetah-v3, Hopper-v3, Walker2d-v3, Ant-v3, and Humanoid-v3.

Experimental settings. We follow the setting from the SAC method [16]. All networks are multilayer perceptrons with two hidden layers of 256 neurons and a ReLU activation function. The networks are training with Adam optimizer and a learning rate of 0.0003. We use mini-batches of 256. Each environment is run for 1 million steps. As DRL algorithms and their variants behave differently in various settings/environments, to cover a wider range of possible scenarios, we study here the case of hard target update where $\tau = 1$ [16]. Following the original paper, target update interval= 1000. We use a temperature α of 0.2. Table 3 shows the value used for λ^1 and λ^2 to determine the sparsity levels for DS-SAC. We use e of 1000 and η of 0.1. The hyperparameters are selected using a random search. Our results are reported over 5 seeds.

Metrics. We used the same metrics discussed in Section 4 to assess the performance of our proposed method.

Results. Figure 5 shows the learning behavior of DS-SAC and SAC. Consistent with our previous observations, DS-SAC learns faster, especially at the beginning of the training. The LCA of DS-SAC is higher than SAC for all environments, as shown in Table 4. DS-SAC outperforms the final performance of SAC for all environments except one where it achieves a very close performance to it, as illustrated in Table 5. Please note that the results of SAC are slightly different from the ones obtained in Section 4.5 as we study here the hard target update case of SAC [17].

These experiments reveal that we can achieve gain in a DRL agent's learning speed and performance while reducing its required memory and computation costs for training.

Table 3: The value used for λ^1 and λ^2 in each environment for the DS-SAC algorithm.

Environment	λ^1	λ^2
HalfCheetah-v3	12	80
Walker2d-v3	12	80
Hopper-v3	7	20
Ant-v3	30	64
Humanoid-v3	61	64

Algorithm 4 DS-SAC

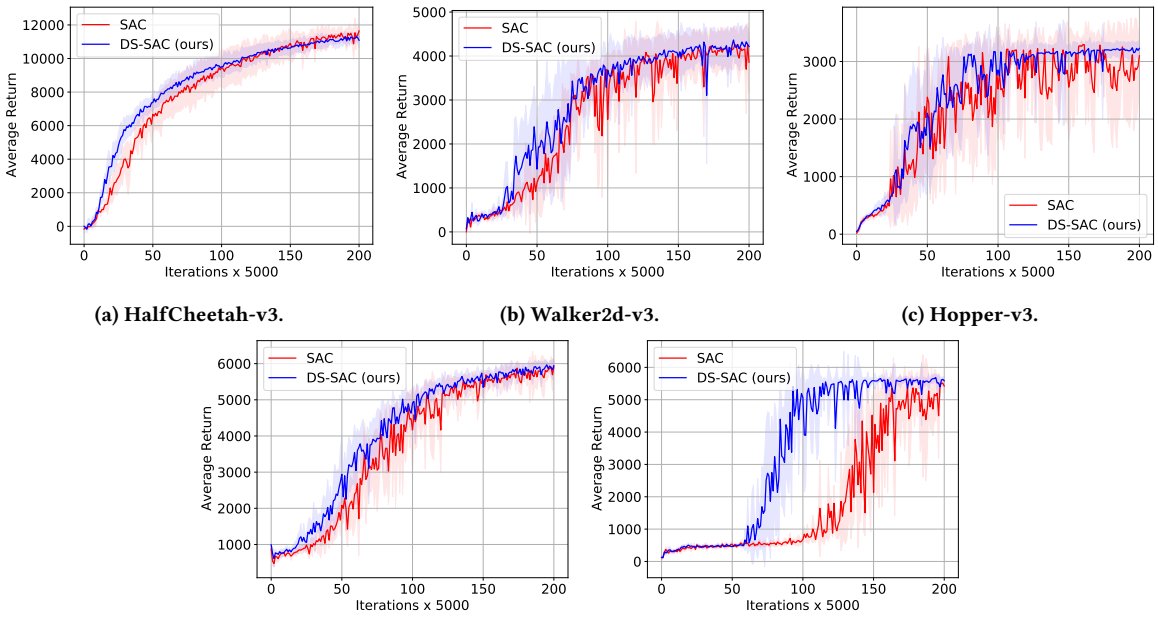
Require: λ^l, η, e
Create $\boldsymbol{M_\phi}$, $\boldsymbol{M_{\theta_1}}$, and $\boldsymbol{M_{\theta_2}}$ with Erdős–Rényi random graph with sparsity level λ^l
$\theta_1 \leftarrow \theta_1 \odot \boldsymbol{M_{\theta_1}}$, $\theta_2 \leftarrow \theta_2 \odot \boldsymbol{M_{\theta_2}}$, $\phi \leftarrow \phi \odot \boldsymbol{M_\phi}$
Initialize target networks $\bar{\theta}_1 \leftarrow \theta_1$, $\bar{\theta}_2 \leftarrow \theta_2$
$\mathcal{D} \leftarrow \emptyset$ // Initialize an empty replay pool
for each iteration **do**
  for each environment step **do**
    $a_t \sim \pi_\phi(a_t|s_t)$ // Sample action from the policy
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ // Sample transition from the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$ // Store the transition in the replay pool
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ // Update Q-functions
    if t mod e **then**
      $\theta_i \leftarrow$ TopologicalAdaptation$(\theta_i, \boldsymbol{M_{\theta_i}}, \eta)$ (Algo. 2)
    end if
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ // Update policy weights
    if t mod e **then**
      $\phi \leftarrow$ TopologicalAdaptation$(\phi, \boldsymbol{M_\phi}, \eta)$ (Algo. 2)
    end if
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ // Update target network
    $\bar{\theta}_i \leftarrow$ MaintainSparsity$(\bar{\theta}_i, \|\theta_i\|_0)$ (Algo. 3)
  end for
end for

Table 4: Learning curve area (LCA) of SAC and DS-SAC.

Environment	SAC	DS-SAC (ours)
HalfCheetah-v3	1.6229	**1.7081**
Walker2d-v3	0.5368	**0.5906**
Hopper-v3	0.4441	**0.4875**
Ant-v3	0.7504	**0.8229**
Humanoid-v3	0.3776	**0.6777**

Table 5: Average return over the last 10 evaluations of 1 million time steps using SAC and DS-SAC.

Environment	SAC	DS-SAC (ours)
HalfCheetah-v3	**11645.12 ± 425.585**	11084.39 ± 445.15
Walker2d-v3	3858.20 ± 689.913	**4216.77 ± 236.23**
Hopper-v3	3100.39 ± 374.45	**3229.39 ± 135.82**
Ant-v3	5899.30 ± 197.15	**5943.54 ± 169.95**
Humanoid-v3	5425.56 ± 196.33	**5584.64 ± 109.40**



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

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

Figure 5: Learning curves of SAC and DS-SAC on different continuous control tasks. The shaded region represents the standard deviation of the average evaluation over 5 runs.