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

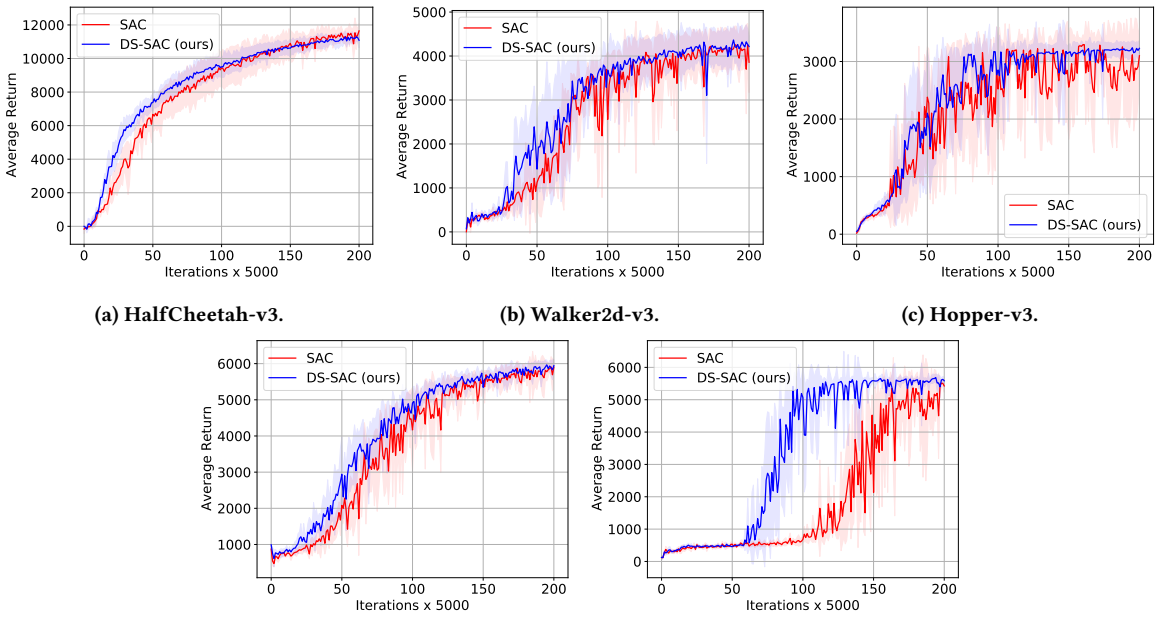
Require: λ^l, η, e
Create $\boldsymbol{M_\phi}$, $\boldsymbol{M_{\theta_1}}$, and $\boldsymbol{M_{\theta_2}}$ with Erdős–Rényi random graph with sparsity level λ^l
$\theta_1 \leftarrow \theta_1 \odot \boldsymbol{M_{\theta_1}}$, $\theta_2 \leftarrow \theta_2 \odot \boldsymbol{M_{\theta_2}}$, $\phi \leftarrow \phi \odot \boldsymbol{M_\phi}$
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.