RLx2: Training a Sparse Deep Reinforcement Learning Model from Scratch

Anonymous authors
Paper under double-blind review

Abstract

Training deep reinforcement learning (DRL) models usually requires high computation costs. Therefore, compressing DRL models possesses immense potential for training acceleration and model deployment. However, existing methods that generate small models mainly adopt the knowledge distillation based approach by iteratively training a dense network, such that the training process still demands massive computing resources. Indeed, sparse training from scratch in DRL has not been well explored and is particularly challenging due to non-stationarity in bootstrap training. In this work, we propose a novel sparse DRL training framework, "the **R**igged **R**einforcement **L**earning **L**ottery" (RLx2), which is capable of training a DRL agent *using an ultra-sparse network throughout* for off-policy reinforcement learning. The systematic RLx2 framework contains three key components: gradient-based topology evolution, multi-step Temporal Difference (TD) targets, and dynamic-capacity replay buffer. RLx2 enables efficient topology exploration and robust Q-value estimation simultaneously. We demonstrate state-of-the-art sparse training performance in several continuous control tasks using RLx2, showing $7.5\times$-$20\times$ model compression with less than 3% performance degradation, and up to $20\times$ and $50\times$ FLOPs reduction for training and inference, respectively.

1 Introduction

Deep reinforcement learning (DRL) has found successful applications in many important areas, e.g., games Silver et al. (2017), roboticsGu et al. (2017) and nuclear fusion Degrave et al. (2022). However, training a DRL model demands heavy computational resources. For instance, AlphaGo-Zero for Go games Silver et al. (2017), which defeats all Go-AIs and human experts, requires more than 40 days of training time on four tensor processing units (TPUs). The heavy resource requirement hinders the application of DRL on resource-limited devices and results in expensive consumption.

Sparse networks, initially proposed in deep supervised learning, have demonstrated great potential for model compression and training acceleration of deep reinforcement learning. In deep supervised learning, the state-of-the-art sparse training frameworks, e.g., SET Mocanu et al. (2018) and RigL Evci et al. (2020), can train a 90%-sparse network (i.e., the resulting network size is 10% of the original network) from scratch without performance degradation. Existing works Rusu et al. (2016); Schmitt et al. (2018); Zhang et al. (2019) also attempt to compress DRL models using knowledge distillation. The lottery ticket hypothesis (LTH) Frankle & Carbin (2019), which claims the existence of a good fixed sparse network (a.k.a. the "winning ticket") in deep supervised learning, is also observed in Yu et al. (2020) to be useful for DRL compression. It is further shown in Vischer et al. (2022) that a "winning ticket" achieves high sparsity levels of 90% by behavioral cloning (BC). However, we note that the results mentioned above mainly focus on generating an ultimately small network and are based on knowledge distillation or supervised learning type methods. Thus, they still require iteratively training a dense network, e.g., due to the need to pre-train dense teachers. As a result, the training cost remains prohibitively high, and the methods cannot be directly implemented on resource-limited devices, leading to low flexibility in adapting the compressed models to new environments.

Training a sparse DRL model from scratch, if done perfectly, has the potential to significantly reduce computation expenditure and enable efficient implementation on resource-limited devices, and

achieves excellent flexibility in model adaptation. However, training a sparse network (e.g., 90% sparsity) from scratch in DRL is challenging due to the non-stationarity in bootstrap training. Specifically, in DRL, the learning target is not fixed but evolves in a bootstrap way (i.e., TD target) Tesauro et al. (1995), and the distribution of the training data can also be non-stationary Desai et al. (2019). Moreover, using a sparse network structure means searching in a smaller hypothesis (topology) space, which further reduces the learning target's confidence. As a result, improper sparsification can cause irreversible damage to the learning path Igl et al. (2021), resulting in poor performance. Indeed, it is shown in Sokar et al. (2021) that a direct adoption of a dynamic sparse training framework in DRL, i.e., SET in TD3, can only compress the model by a half (50% sparsity) without much performance degradation. This result indicates the significant performance gap between training from scratch and using knowledge distillation and BC Vischer et al. (2022), and raises the following interesting open question:

Can an efficient DRL agent be trained from scratch with an ultra-sparse network throughout?

In this paper, we give an affirmative answer to the problem and propose a novel sparse training framework, "the **R**igged **R**einforcement **L**earning **L**ottery" (RLx2), for off-policy RL. Sparse training throughout is significant since it can significantly reduce the computation cost and can be directly implemented on resource-limited devices. Our proposed sparse training framework RLx2 is the first algorithm to achieve sparse training throughout using a sparsity of more than 90% with minimal performance loss.

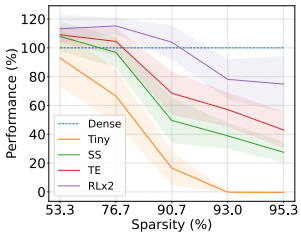


Figure 1: Comparison of four methods with TD3 on Ant-v3 under different sparsity. The performance is normalized to that of the original dense model.

RLx2 is motivated by our observation that topology evolution and accurate value estimation are two critical steps in training DRL from scratch. As an illustrating experiment, Fig. 1 presents a comparison on Ant-v3, one of MuJoCo Todorov et al. (2012) environments, for four methods generating sparse networks for DRL under different sparsity [1], including a small dense network (Tiny Dense, orange), a fixed sparse network (Static Sparse, green), a dynamic-structure training scheme without accurate value estimation (Topology Evolution, red), and RLx2 (purple). The result indicates that both topology evolution and accurate value estimation are indispensable. In particular, we apply a gradient-guided topology search scheme Evci et al. (2020) to enable dynamic network evolution. We further equip RLx2 with multi-step temporal difference (TD) targets for more accurate value estimation, with a novel dynamic-capacity replay buffer for correcting the off-policy issue. These three key steps simultaneously achieve efficient topology exploration and better value estimates in sparse models. Through extensive experiments in various environments using RLx2 with two popular off-policy RL algorithms, TD3 and SAC, we demonstrate that RLx2 is able to achieve superb model compression with minimal performance degradation by *using an ultra-sparse network throughout training*. We summarize the main contributions of the paper as follows.

- We conduct an in-depth investigation on the obstacles to training a sparse neural network from scratch in DRL, and discover two key ingredients necessary for success, namely topology evolution and accurate value estimation.
- We propose RLx2, a novel sparse training framework for DRL. RLx2 consists of a gradient-guided topology search scheme, which dynamically grows and deletes network connections with magnitude and value-based gradient criteria and a multi-step Temporal Difference (TD) target and dynamic-capacity replay buffer to handle data inconsistency and improve the value estimation. RLx2 simultaneously achieves efficient topology exploration in the reduced search space, which enables model compression, and accurate value estimation, which guarantees performance.
- We demonstrate the state-of-the-art sparse training performance of RLx2 with two popular off-policy RL algorithms, TD3 Fujimoto et al. (2018) and SAC Haarnoja et al. (2018) , in various environments. The extensive experimental results demonstrate up to $20\times$ model compression, even better than the current best sparsity obtained based on BC in Vischer

[1] Here sparsity equals to $(1-r)\times 100\%$, where r is the fraction of model size reduced (detailed in Appendix). For instance, a sparse model whose size is 10% of the original model has a 90% sparsity.

et al. (2022). RLx2 also achieves 20× acceleration in training and 50× in inference in floating-point operations (FLOPs).

2 Related Works

We discuss the related works on training sparse models in deep supervised learning and reinforcement learning below. We also provide a comprehensive performance comparison in Table 1.

Sparse Models in Deep Supervised Learning Han et al. (2015; 2016); Srinivas et al. (2017); Zhu & Gupta (2018) focus on finding a sparse network by pruning pre-trained dense networks. Iterative Magnitude Pruning (IMP) in Han et al. (2016) achieves a sparsity of more than 90%. Neuron characteristic Hu et al. (2016), dynamic network surgery Guo et al. (2016), derivatives Dong et al. (2017); Molchanov et al. (2019b), regularization Louizos et al. (2018); Tartaglione et al. (2018), dropout Molchanov et al. (2017) have also been applied in network pruning. Another line of work focus on the Lottery Ticket Hypothesis (LTH), first proposed in Frankle & Carbin (2019), which shows that training from a sparse network from scratch is possible if one finds a sparse "winning ticket" initialization in deep supervised learning. The LTH is also validated in other deep learning models Chen et al. (2020); Brix et al. (2020); Chen et al. (2021). Many works Bellec et al. (2018); Mocanu et al. (2018); Mostafa & Wang (2019); Dettmers & Zettlemoyer (2019); Evci et al. (2020) also try to train a sparse neural network from scratch without having to pre-trained dense models. These works adjust structures of sparse networks during training, including Deep Rewiring (DeepR) Bellec et al. (2018), Sparse Evolutionary Training (SET) Mocanu et al. (2018), Dynamic Sparse Reparameterization (DSR) Mostafa & Wang (2019), Sparse Networks from Scratch (SNFS) Dettmers & Zettlemoyer (2019), and Rigged Lottery (RigL) Evci et al. (2020). On the contrary, Single-Shot Network Pruning (SNIP) Lee et al. (2019) and Gradient Signal Preservation (GraSP) Wang et al. (2020) focus on finding static sparse networks before training begins.

Table 1: Comparison of different sparse training techniques in DRL. Here ST and TA stand for "sparse throughout training" and "training acceleration," respectively. The shown sparsity is the maximum sparsity level without performance degradation under the algorithms.

Name	Paradigm	Scenario	ST	TA	Sparsity
PoPS Livne & Cohen (2020)	IMP	Online	No	No	$\sim 99\%$
LTH-RL Yu et al. (2020)	IMP	Online	Yes	No	$\sim 99\%$
LTH-IL Vischer et al. (2022)	IMP	Online	Yes	No	$\sim 95\%$
SSP Arnob et al. (2021)	Single-shot pruning	Offline	Yes	Yes	$\sim 95\%$
GST Lee et al. (2021)	Gradual pruning	Online	No	No	$\sim 70\%$
DST Sokar et al. (2021)	Topology Evolution	Online	Yes	Yes	$\sim 50\%$
RLx2 (Ours)	Topology Evolution	Online	Yes	Yes	$\sim \mathbf{95}\%$

Sparse Models in DRL Evci et al. (2020); Sokar et al. (2021) show that finding a sparse model in DRL is difficult due to training instability. Existing works Rusu et al. (2016); Schmitt et al. (2018); Zhang et al. (2019) leverage knowledge distillation with static data to avoid unstable training and obtains small dense agents. Policy Pruning and Shrinking (PoPs) in Livne & Cohen (2020) obtains a sparse DRL agent with iterative policy pruning (similar to IMP). LTH in DRL is investigated in Yu et al. (2020). Vischer et al. (2022) shows that a sparse winning ticket can be found by behavior cloning (BC). Another line of works Lee et al. (2021); Sokar et al. (2021); Arnob et al. (2021) attempts to train a sparse neural network from scratch without pre-training a dense teacher. Group Sparse Training (GST) in Lee et al. (2021) utilizes block-circuit compression and reward-aware pruning. Sokar et al. (2021) proposes using SET in topology evolution in DRL and achieves 50% sparsity. Arnob et al. (2021) proposes single-shot pruning (SSP) for offline RL. Graesser et al. (2022) finds that pruning often obtains the best results and dynamic sparse training improves over static sparse training significantly. We further improve the performance of the dynamic sparse training by making a robust value estimation.

3 DEEP REINFORCEMENT LEARNING PRELIMINARIES

In reinforcement learning, an agent interacts with an unknown environment to learn an optimal policy. The learning process is formulated as a Markov decision process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, r, \mathbb{P}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, r is the reward function, $\mathbb{P}$ denotes the transition matrix, and γ stands for the discount factor. Specifically, at time slot t, given the current state $s_t \in \mathcal{S}$, the agent selects an action $a_t \in \mathcal{A}$ by policy $\pi : \mathcal{S} \to \mathcal{A}$, which then incurs a reward $r_t(s, a)$.

Denote the Q function associated with the policy π for state-action pair (s, a) as

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{i=t}^{T} \gamma^{i-t} r\left(s_i, a_i\right) | s_t = s, a_t = a \right]. \tag{1}$$

In actor-critic methods Silver et al. (2014), the policy $\pi(s; \phi)$ is parameterized by a policy (actor) network with weight parameter ϕ, and the Q function $Q(s, a; \theta)$ is parameterized by a value (critic) network with parameter θ. The goal of the agent is to find an optimal policy $\pi^*(s; \phi^*)$ which maximizes long-term cumulative reward, i.e., $J^* = \max_\phi \mathbb{E}_{\pi(\phi)}[\sum_{i=0}^{T} \gamma^{i-t} r(s_i, a_i) | s_0, a_0]$.

There are various DRL methods for learning an efficient policy. In this paper, we focus on off-policy TD learning methods, including a broad range of state-of-the-art algorithms, e.g., TD3 Fujimoto et al. (2018) and SAC Haarnoja et al. (2018). Specifically, the critic network is updated by gradient descent to fit the TD targets $\mathcal{T}_1(s, a)$ generated by a target network $Q(s, a; \theta')$, i.e.,

$$\mathcal{T}(s, a) = r(s, a) + \gamma Q\left(s, a; \theta'\right) \tag{2}$$

for each state-action pair (s, a), where the action $a = \pi(s; \phi)$. The loss function of the value network is defined as the expected squared loss between current value network and TD targets:

$$\mathcal{L}(\theta) = \mathbb{E}_{\pi(\phi)} \left[Q(s_i, a_i; \theta) - \mathcal{T} \right]^2. \tag{3}$$

The policy $\pi(s; \phi)$ is updated by the deterministic policy gradient algorithm in Silver et al. (2014) as

$$\nabla_\phi J(\phi) = \mathbb{E}_{\pi(\phi)} \left[\nabla_a Q_\pi(s, a; \theta)|_{a=\pi(s)} \nabla_\phi \pi_\phi(s) \right].$$

4 RLX2: RIGGING THE LOTTERY IN DRL

We introduce our RLx2 algorithm in this section. First, we compare sparse DRL training methods using TD3 with different topology update schemes on InvertedPendulum-v2, a simple control task from MuJoCo Todorov et al. (2012), in Figure 2. From the results, we make the following important observations. First, *dynamic topology evolution is crucial*. This can be seen from the fact that a random static sparse network (SS, orange) does not achieve a satisfying performance and performs much worse than that with topology evolution (TE, green). Second, *accurate value estimation is essential*. Unlike supervised learning, DRL training heavily relies on value estimation. Thus, improving the value estimation can intuitively boost the learning performance. This is confirmed by the comparison between TE and TE+Q^* (purple), which is an ideal scheme using topology evolution (same as in TE) yet replacing TD targets with expert Q-values, i.e., Q-values from a fully trained expert critic with a dense network. Our RLx2 algorithm, which contains both topology evolution Evci et al. (2020) (using the same scheme in TE) and accurate value estimation (using multi-step TD targets Hessel et al. (2018)), is able to achieve a performance close to TE+Q^* without the need for pre-trained expert Q-values.

These results clearly demonstrate that both topology evolution and accurate value estimation are critical for training a sparse DRL agent from scratch.

An overview of RLx2 on an actor-critic architecture is shown in Figure 3. Below, we will explain the key components of RLx2 in detail, including topology evolution, multi-step TD target, and dynamic-capacity buffer.

4.1 TOPOLOGY EVOLUTION

The topology evolution in RLx2 is made by adopting the RigL Evci et al. (2020) method. Specifically, we compute the gradient values of the loss function with respect to each link weight. Then, we

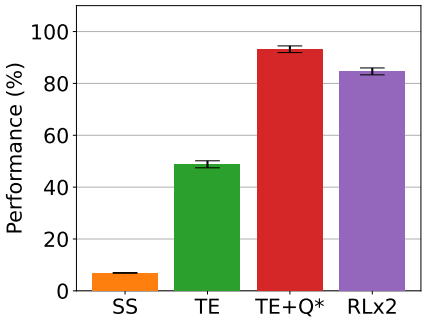


Figure 2: Performance of sparse DRL agents obtained under different methods.

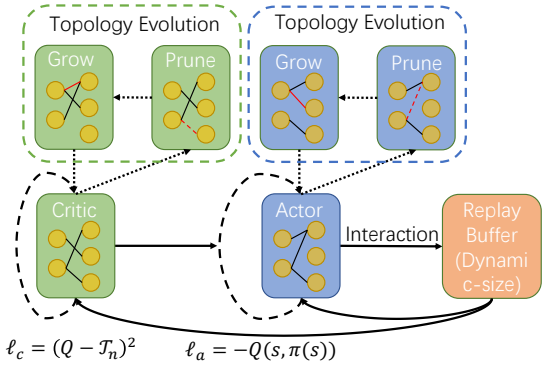


Figure 3: Overview of RLx2.

dynamically grow connections (connecting neurons) with large gradients and delete existing links with the least weights. In this way, we obtain a sparse mask that evolves by self-adjustment. Please refer to Evci et al. (2020) for more details.

The pseudo-code of our scheme is given in Algorithem 1, where $\odot$ is an element-wise multiplication operatorand M_ϕ is a binary mask to represent the sparse architecture of the network ϕ. The update fraction anneals during the training process as $\zeta_t = \frac{\zeta_0}{2}(1 + \cos(\frac{\pi t}{T_{\text{end}}}))$, where ζ_0 is the initial update fraction and T_{end} is the total number of iterations.

Adopting the RigL scheme in deep reinforcement learning can have more profound implications. Due to the potential data inconsistency problem in value estimation and the smaller hypothesis search space under sparse networks, training with a single fixed topology does not fully reap the benefit of a sparse topology and can result in significantly degraded performance. Please see further discussion on this in Section 5.3.

Algorithm 1 Topology Evolution Evci et al. (2020)

N_l: Number of parameters in layer l
θ_l: Parameters in layer l
M_{θ_l}: Sparse mask of layer l
s_l: Sparsity of layer l
L: Loss function
ζ_t: Update fraction in training step t
for each layer l **do**
  $k = \zeta_t(1 - s_l)N_l$
  $\mathbb{I}_{\text{drop}} = \text{ArgTopK}(-|\theta_l \odot M_{\theta_l}|, k)$
  $\mathbb{I}_{\text{grow}} = \text{ArgTopK}_{i \notin \theta_l \odot M_{\theta_l} n \mathbb{I}_{\text{drop}}}(|\nabla_{\theta_l} L, k|)$
Update M_{θ_l} according to $\mathbb{I}_{\text{drop}}$ and $\mathbb{I}_{\text{grow}}$
  $\theta_l \leftarrow \theta_l \odot M_{\theta_l}$
end for

4.2 Accurate Value Learning

As discussed above, the learning of the value function is crucial in sparse training. It not only serves to guarantee the efficiency of the DRL agent but also guides the adjustment of the sparse network during training. To achieve accurate value estimation, RLx2 utilizes two components, including a multi-step TD target to bootstrap value estimation, and a dynamic-capacity replay buffer to eliminate the potential data-inconsistency problem due to policy change during training.

4.2.1 Multi-step TD Target

In TD learning, a TD target is generated to update the value network by minimizing the squared loss concerning the TD target. Single-step methods generate the TD target by one-step reward and discounted target network output, i.e.,

$$\mathcal{T}_1 = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}); \theta) \tag{4}$$

However, a sparse network $\widehat{\theta} = \theta \odot M_\theta$, obtained from θ, inevitably contains only a smaller hypothesis space compared to a dense network, which means the output of the sparse value network $\widehat{\theta}$ can be unreliable and lead to inaccurate value estimation. Denote the network fitting error as $\epsilon(s, a) = Q(s, a; \theta) - Q_\pi(s, a)$. We see that this error can be non-negligible when the sparsity of the value network is relatively high in Eq. (4).

To overcome this issue, we introduce multi-step targets in Eq. (5), where the target combines an N-step sample and the output of the sparse value network with reduced weight.

$$\mathcal{T}_n = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n Q(s_{t+n}, \pi(s_{t+n}); \theta) \tag{5}$$

By doing so, we reduce the expected error between the TD target and the real target. Specifically, Eq.(6) shows the expected TD error between multi-step TD target $\mathcal{T}_n$ and the real Q function Q_π associated with the target policy π conditioned on transitions from behavior policy b (see detailed derivation in Appendix).

$$\mathbb{E}_b[\mathcal{T}_n(s, a)] - Q_\pi(s, a) = \underbrace{(\mathbb{E}_b[\mathcal{T}_n(s, a)] - \mathbb{E}_\pi[\mathcal{T}_n(s, a)])}_{\text{Policy inconsistency error}} + \gamma^n \underbrace{\mathbb{E}_\pi[\epsilon(s_n, \pi(s_n))]}_{\text{Network fitting error}} \tag{6}$$

Eq.(6) shows that the expected error consists of two parts, i.e., the policy inconsistency error and network fitting error. In particular, introducing a multi-step target reduces the network fitting error by a reduced weight γ^n. However, it has been observed, e.g., in Fedus et al. (2020), that a immediate adoption of multi-step TD targets may enlarge the policy inconsistency error (the first term). To suppress the policy inconsistency and further improve value learning, we introduce a novel adaptive scheme to control the replay buffer capacity to eliminate data inconsistency (detailed in Section 4.2.2 below).

4.2.2 Dynamic-capacity Buffer

Off-policy algorithms use a replay buffer to store collected data and train networks with sampled batches from the buffer, and their performance generally improves with increased replay capacity Fedus et al. (2020). However, off-policy algorithms with unlimited-size replay buffers can suffer from policy inconsistency due to inconsistent multi-step targets and mismatched training data, which we discuss below.

(i) *Inconsistent multi-step targets:* In off-policy algorithms with multi-step TD targets, the value function is updated to minimize the squared loss in Eq. (3) by sampling transitions in replay buffer, i.e., reward sequence $r_t, r_{t+1}, \cdots, r_{t+n}$ collected during training. However, the fact that the policy can evolve during training means the data in the replay buffer, which is used to form Monte-Carlo approximation of the current policy π, may be collected with a different behavior policy b. As a result, the policy inconsistency error in Eq. (6 may be enlarged, leading to inaccuracy estimation.

(ii) *Mismatched training data:* In practice, the agent minimizes the value loss $\widehat{\mathcal{L}}(\theta)$ with respect to the sampled value of mini-batch $\mathcal{B}_t$, which is given as

$$\widehat{\mathcal{L}}(\theta) = \frac{1}{|\mathcal{B}_t|} \sum_{(s_i, a_i) \sim \mathcal{B}_t} (Q(s_i, a_i; \theta) - \mathcal{T})^2 \tag{7}$$

Compared to Eq. (3), the difference between the distribution of transitions in the mini-batch $\mathcal{B}_t$ and that of current policy also leads to a mismatch.

The off-policy issue may be exacerbated by sparse training due to using smaller models. To tackle these issues, we propose a dynamic-capacity buffer to reduce the policy gap based on the policy distance of the collected data. The formal scheme is given in Algorithm 2. Specifically, we introduce the following policy distance measure to evaluate the inconsistency of data in the buffer, i.e.,

$$\mathcal{D}(\phi) = \frac{1}{|\mathcal{B}_0|} \sum_{(s_i, a_i) \in \mathcal{B}_0} \parallel \pi(s_i; \phi) - a_i \parallel_2 \tag{8}$$

where $\mathcal{B}_0$ denotes the oldest mini-batch of the replay buffer and π is the current policy. If this measure is below a certain pre-specified threshold, we enlarge the buffer. Otherwise, we replace the oldest data with the newly collected samples.

5 EXPERIMENTS

In this section, we conduct an extensive evaluation to investigate the performance improvement of RLx2, and investigate the importance of each component in RLx2. We further analyze the differences between the dynamic network topology and a winning ticket. Specifically, we conduct extensive experiments on four MuJoCo environments: HalfCheetah-v3 (Hal.), Hopper-v3 (Hop.), Walker2d-v3 (Wal.), and Ant-v3 (Ant.), for our proposed sparse training framework RLx2 with two off-policy algorithms, TD3 and SAC. Instantiations of RLx2 on TD3 and SAC are provided in Appendix. Each result is averaged over eight random seeds.

5.1 COMPARATIVE EVALUATION

We first present a comparative evaluation of RLx2 with TD3 and SAC on four MuJoCo environments in Table 2. In our experiments, we compare RLx2 with the following baselines: (i) Tiny, which uses tiny dense networks with the same number of parameters as the sparse model in training. (ii) SS: using static sparse networks with random initialization. (iii) SET Bellec et al. (2018), which uses dynamic sparse training by dropping connections with magnitude criteria and growing connections randomly. Please notice that the previous work Sokar et al. (2021) also adopts the SET algorithm for topology evolution in reinforcement learning. Our implementations reach better performance due to different hyperparameters. (iv) RigL Evci et al. (2020), which uses dynamic sparse training by dropping and growing connections with magnitude and gradient criteria, respectively, the same as RLx2's topology evolution procedure.

In our experiments, we allow the actor and critic networks to take different sparsities. We define an ultimate compression ratio, i.e., the largest sparsity level ensuring that the performance degradation under RLx2 is within $\pm\%3$ of that under the original dense models. This can also be understood as the minimum size of the sparse model with the full performance of the original dense model. We present performance comparison results in Table 2 based on the ultimate compression ratio. The performance of each algorithm is evaluated with the average reward per episode over the last 30 evaluations of the training (evaluation is conducted every 5000 steps). Hyperparameters are fixed in all four environments for TD3 and SAC, respectively, which are presented in Appendix.

Table 2: Comparisons of RLx2 with sparse training baselines. Here "Sp." refers to the sparsity level (percentage of model size reduced), "Total Size" refers to the total parameters of both critic and actor networks (detailed calculation of training and inference FLOPs are given in Appendix). The right five columns show the final performance of different methods. The "Total size," "FLOPs" , ~~and "Performance" are all normalized w.r.t. the original large dense model (detailed in Appendix).~~

Alg.	Env.	Actor Sp.	Critic Sp.	Total Size	FLOPs (Train)	FLOPs (Test)	Tiny (%)	SS (%)	SET (%)	RigL (%)	RLx2 (%)
TD3	Hal.	90%	85%	0.133x	0.138x	0.100x	86.3	77.1	92.6	90.8	**99.8**
	Hop.	98%	95%	0.040x	0.043x	0.020x	64.5	67.7	66.5	90.6	**97.0**
	Wal.	97%	95%	0.043x	0.045x	0.030x	60.8	42.9	39.3	35.7	**98.1**
	Ant.	96%	88%	0.093x	0.100x	0.040x	16.5	49.6	62.5	68.5	**103.9**
	Avg.	95%	91%	0.077x	0.081x	0.048x	57.0	59.3	65.2	71.4	**99.7**
SAC	Hal.	90%	80%	0.180x	0.197x	0.100x	95.0	75.4	94.8	89.8	**102.2**
	Hop.	98%	95%	0.044x	0.048x	0.020x	89.1	81.6	103.9	**110.0**	**109.7**
	Wal.	90%	90%	0.100x	0.113x	0.100x	73.8	83.4	95.8	81.9	**103.2**
	Ant	90%	75%	0.220x	0.239x	0.100x	49.6	49.3	79.8	90.9	**105.6**
	Avg.	92%	85%	0.136x	0.149x	0.080x	76.9	72.4	93.6	93.2	**105.2**
Avg.		94%	88%	0.107x	0.115x	0.064x	67.0	65.9	79.4	82.3	**102.4**

From Table 2, we see that RLx2 achieves superior compression ratios (the reciprocal of the total size) compared to others. Specifically, RLx2 with TD3 achieves $7.5\times$-$25\times$ model compression, with the best compression ratios of $25\times$ on Hopper-v3. The actor can be compressed for each environment by more than 96% and the critic is compressed by 85%-95%. The results for SAC are similar. RLx2 with SAC achieves a $5\times$-$20\times$ model compression. In addition to the superior compression perfor-

mance, RLx2 also performs best among all baselines in all four environments (except for a close performance with RigL with SAC in Hopper) by a large margin, which validates the state-of-the-art performance of our RLx2 framework in model compression, training, and inference acceleration.

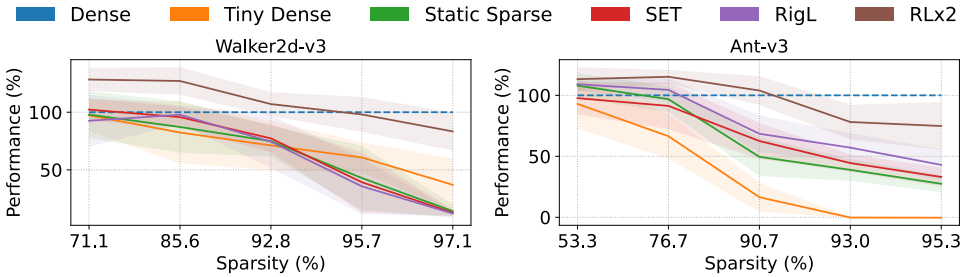


Figure 4: Comparison with different model sparsity.

Tiny dense (Tiny) and random static sparse networks (SS) performance are worst on average. SET and RigL are better yet fail to maintain the performance in Walker2d-v3 and Ant-v3, which means robust value learning is necessary under sparse training. We want to highlight here that different from knowledge-distillation/BC based methods, e.g., Livne & Cohen (2020); Vischer et al. (2022); Lee et al. (2021), RLx2 uses a sparse network *throughout* training. Thus, it has an additional advantage of immensely accelerating training and saving computation, i.e., $12\times$ training acceleration and $20\times$ inference acceleration for RLx2-TD3, and $7\times$ training acceleration and $12\times$ inference acceleration for RLx2-SAC. To further validate the performance of RLx2, we compare the performance of RLx2-TD3 with other baselines under different sparsity levels in Hopper-v3 and Ant-v3 in Figure 4. We find that RLx2 has a significant performance gain over other baselines.

5.2 Ablation Study

We conduct a comprehensive ablation study on three critical components of RLx2 on TD3, i.e., topology evolution, multi-step TD target, and dynamic-capacity buffer, to examine the effect of each component in RLx2 and their robustness in hyperparameters.

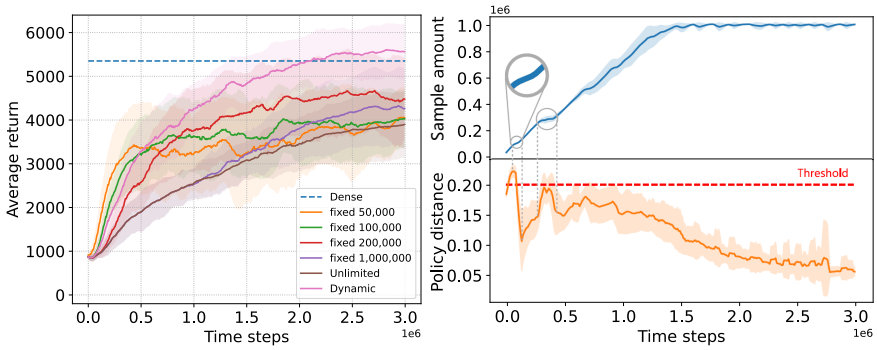
Topology evolution RLx2 drops and grows connections with magnitude and gradient criteria, respectively, which has been adopted in RigL Evci et al. (2020) for deep supervised learning. To validate the necessity of our topology evolution criteria, we compare RLx2 with three baselines, which replace the topology evolution scheme in RLx2 with Tiny, SS and SET, with other components in RLx2 remain unchanged, i.e., they are also equipped with multi-step targets and dynamic-capacity buffer.

The left part of Table 3 shows that RigL as a topology adjustment scheme (Note that in this case, the scheme is RLx2) performs best on average among four baselines. We also observe that Tiny performs worst, which is consistent with the conclusion in existing works Zhu & Gupta (2018) that a sparse network may contain a smaller hypothesis space and necessitates a topology evolution scheme.

Table 3: Ablation study on topology evolution and multi-step target, where the performance (%) is normalized with respect to the performance of dense models.

Env.	Topoloy Evolution				Multi-step Target				
	Tiny	SS	SET	**RLx2**	1-step	2-step	**3-step**	4-step	5-step
Hal.	93.3	86.1	100.1	99.8	96.5	101.7	99.8	98.8	97.0
Hop.	74.4	84.2	88.8	97.0	77.9	91.7	97.0	84.0	87.5
Wal.	84.1	83.8	89.4	98.1	73.9	93.7	98.1	99.1	99.3
Ant.	28.7	80.2	83.5	103.9	103.9	105.1	103.9	96.7	94.5
Avg.	70.1	83.6	90.4	**99.7**	88.1	98.1	**99.7**	94.6	94.6

N-step TD target We also compare different step lengths in multi-step TD targets for RLx2 in the right part of Table 3. We find that multi-step TD targets with step length 3 obtain the maximum performance. In particular, multi-step TD targets improve the performance dramatically in Hopper-v3 and Walker2d-v3, while the improvement in HalfCheetah-v3 and Ant-v3 is minor.



(a) Training curves (b) Buffer size and policy inconsistency

Figure 5: **Left**: Performance with different buffer schemes. Smaller buffer capacity benefits the performance in the early stage but may reduce the final performance. Our dynamic buffer outperforms the fixed capacity buffer. **Right**: Policy is updated quickly early on and leads to severe policy inconsistency. By dropping the oldest samples, our dynamic buffer scheme keeps the policy distance under the threshold. As the policy converges, the sample amount in the buffer also increases.

Dynamic-capacity Buffer We compare different replay buffer sizing schemes, including our dynamic scheme and using different fixed buffer capacities or an unlimited buffer. Figure 5(a) shows that our dynamic-capacity buffer performs best among all settings of the buffer. Using a smaller buffer results in higher sample efficiency in the early stage of training but fails in reaching high performance in the long term, whereas a large or even unlimited one may perform poorly in all stages. We also visualize in Figure 5(b) the sample amount and the proposed policy distance metric in Eq. (8) during training. The policy distance oscillates in the early stage as the policy evolves, but it is controlled and does not violate the threshold condition to effectively address the off-policyness issue. As the policy converges, the policy distance tends to decrease and converge.

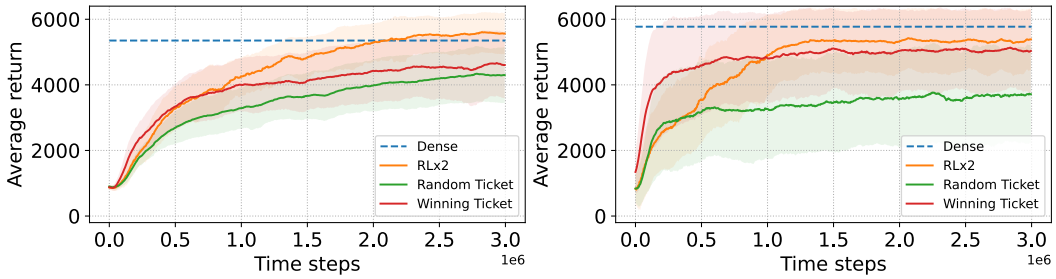
5.3 Dynamic Topology vs. Winning Ticket

Previous subsections show that RLx2 can train an ultra-sparse model from scratch in DRL. This section compares the performance of using a dynamic topology (i.e., RLx2) versus only using a fixed winning ticket initialization to investigate the Lottery Ticket Hypothesis (LTH) Frankle & Carbin (2019). Specifically, LTH demonstrates that a sparse network can be trained from scratch to reach the same performance as a dense one, if initialized with an appropriate mask Yu et al. (2020), i.e., the "winning ticket." In particular, we compare the performance of random sparse initialization (fixed topology obtained randomly), winning ticket initialization (fixed topology obtained under RLx2 after training, with architecture visualization in Appendix), and random initialization with topology evolution (dynamic topology using RLx2) in both DRL learning and behavior cloning (BC).[2]

From Figure 6(a), we see that RLx2 with random initialization (orange curve) achieves the best performance, which is comparable with that under the original dense model. Although the winning ticket initialization (red curve) performs better than a random initialization (green curve), both without topology evolution, they lead to significant performance loss compared to RLx2, showing the necessity of topology evolution in DRL training from scratch.

On the other hand, Figure 6(b) shows that under BC, the winning ticket initialization and random initialization with topology evolution perform almost the same as the dense model, while the static random initialization performs worst. This indicates that an appropriate fixed initialization can indeed be sufficient to reach satisfactory performance in BC, which is intuitive since BC adopts a supervised learning approach and eliminates non-stationarity due to bootstrapping training, a result also observed in Vischer et al. (2022). However, although the winning ticket initialization under BC achieves similar performance as RLx2, we emphasize that RLx2 uses a sparse topology throughout training, while BC-based methods rely on pre-training dense expert networks. As a result, RLx2 is able to accelerate training (in FLOPs) by a factor of $7.5\times$-$25\times$ compared to BC.

[2]In BC, the actor network is trained under the guidance of a well-trained expert, instead of the critic network.



(a) Rigging the Lottery Ticket in RL training (b) Rigging the Lottery Ticket in behavior cloning

Figure 6: Comparison of different methods for training an DRL agent with a sparse network.

6 Conclusion

This paper proposes a sparse training framework, RLx2, for off-policy reinforcement learning (RL). RLx2 utilizes gradient-based evolution to enable efficient topology exploration and establishes accurate value learning using multi-step TD targets and a dynamic-capacity replay buffer. RLx2 enables training an efficient DRL agent with minimal performance loss using an ultra-sparse network throughout and removes the need for pre-training dense networks. Our extensive experiments on RLx2 with TD3 and SAC demonstrate state-of-the-art sparse training performance, showing a $7.5\times$-$20\times$ model compression with less than 3% performance degradation and up to $20\times$ and $50\times$ FLOPs reduction in training and inference, respectively.

References

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International conference on machine learning*, pp. 22–31. PMLR, 2017.

Samin Yeasar Arnob, Riyasat Ohib, Sergey Plis, and Doina Precup. Single-shot pruning for offline reinforcement learning. *arXiv preprint arXiv:2112.15579*, 2021.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. 2018.

Christopher Brix, Parnia Bahar, and Hermann Ney. Successfully applying the stabilized lottery ticket hypothesis to the transformer architecture. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3909–3915, 2020.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*, pp. 1695–1706. PMLR, 2021.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

Shrey Desai, Hongyuan Zhan, and Ahmed Aly. Evaluating lottery tickets under distributional shifts. *arXiv preprint arXiv:1910.12708*, 2019.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *Advances in Neural Information Processing Systems*, 30, 2017.

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*, pp. 2943–2952. PMLR, 2020.

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pp. 3061–3071. PMLR, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International conference on learning representations*, 2019.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 7766–7792. PMLR, 2022.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396. IEEE, 2017.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *Advances in neural information processing systems*, 29, 2016.

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*, pp. 1861–1870. PMLR, 2018.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International conference on learning representations*, 2016.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International conference on learning representations*, 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.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International conference on learning representations*, 2019.

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.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International conference on learning representations*, 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.

Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pp. 2498–2507. PMLR, 2017.

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

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

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

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *International conference on learning representations*, 2016.

Simon Schmitt, Jonathan J Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M Czarnecki, Joel Z Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, et al. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pp. 387–395. PMLR, 2014.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. Dynamic sparse training for deep reinforcement learning. *arXiv preprint arXiv:2106.04217*, 2021.

Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. Training sparse neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 138–145, 2017.

Enzo Tartaglione, Skjalg Lepsøy, Attilio Fiandrotti, and Gianluca Francini. Learning sparse neural networks via sensitivity-driven regularization. *Advances in neural information processing systems*, 31, 2018.

Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. On lottery tickets and minimal task representations in deep reinforcement learning. In *International conference on learning representations*, 2022.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *International conference on learning representations*, 2020.

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)*, pp. 1–8. IEEE, 2019.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. In *Workshop at international conference on learning representations*, 2018.