Pruning Randomly Initialized Neural Networks with Iterative Randomization

Daiki Chijiwa†§, Shin’ya Yamaguchi†, Yasutoshi Ida†
Kenji Umakoshi‡, Tomohiro Inoue‡

†NTT Computer and Data Science Laboratories, NTT Corporation
‡NTT Social Informatics Laboratories, NTT Corporation

Abstract

Pruning the weights of randomly initialized neural networks plays an important role in the context of lottery ticket hypothesis. Ramanujan et al. [26] empirically showed that only pruning the weights can achieve remarkable performance instead of optimizing the weight values. However, to achieve the same level of performance as the weight optimization, the pruning approach requires more parameters in the networks before pruning and thus more memory space. To overcome this parameter inefficiency, we introduce a novel framework to prune randomly initialized neural networks with iteratively randomizing weight values (IteRand). Theoretically, we prove an approximation theorem in our framework, which indicates that the randomizing operations are provably effective to reduce the required number of the parameters. We also empirically demonstrate the parameter efficiency in multiple experiments on CIFAR-10 and ImageNet. The code is available at https://github.com/dchiji-ntt/iterand.

1 Introduction

The lottery ticket hypothesis, which was originally proposed by Frankle and Carbin [6], has been an important topic in the research of deep neural networks (DNNs). The hypothesis claims that an over-parameterized DNN has a sparse subnetwork (called a winning ticket) that can achieve almost the same accuracy as the fully-trained entire network when trained independently. If the hypothesis holds for a given network, then we can reduce the computational cost by using the sparse subnetwork instead of the entire network while maintaining the accuracy [15, 22]. In addition to the practical benefit, the hypothesis also suggests that the over-parametrization of DNNs is no longer necessary and their subnetworks alone are sufficient to achieve full accuracy.

Ramanujan et al. [26] went one step further. They proposed and empirically demonstrated a conjecture related to the above hypothesis, called the strong lottery ticket hypothesis, which informally states that there exists a subnetwork in a randomly initialized neural network such that it already achieves almost the same accuracy as a fully trained network, without any optimization of the weights of the network. A remarkable consequence of this hypothesis is that neural networks could be trained by solving a discrete optimization problem. That is, we may train a randomly initialized neural network by finding an optimal subnetwork (which we call weight-pruning optimization), instead of optimizing the network weights continuously (which we call weight-optimization) with stochastic gradient descent (SGD).

However, the weight-pruning optimization requires a problematic amount of parameters in the random network before pruning. Pensia et al. [25] theoretically showed that the required network width for the weight-pruning optimization needs to be logarithmically wider than the weight-optimization at

*Corresponding author: daiki.chijiwa.mk@hco.ntt.co.jp

35th Conference on Neural Information Processing Systems (NeurIPS 2021).
least in the case of shallow networks. Therefore, the weight-pruning optimization requires more parameters, and thus more memory space, than the weight-optimization to achieve the same accuracy. In other words, under a given memory constraint, the weight-pruning optimization can have lower final accuracy than the weight-optimization in practice.

In this paper, we propose a novel optimization method for neural networks called weight-pruning with iterative randomization (IteRand), which extends the weight-pruning optimization to overcome the parameter inefficiency. The key idea is to virtually increase the network width by randomizing pruned weights at each iteration of the weight-pruning optimization, without any additional memory consumption. Indeed, we theoretically show that the required network width can be reduced by the randomizing operations. More precisely, our theoretical result indicates that, if the number of randomizing operations is large enough, we can reduce the required network width for weight-pruning to the same as that for a network fully trained by the weight-optimization up to constant factors, in contrast to the logarithmic factors of the previous results [25, 24]. We also empirically demonstrate that, under a given amount of network parameters, IteRand boosts the accuracy of the weight-pruning optimization in multiple vision experiments.

2 Background

In this section, we review the prior works on pruning randomly initialized neural networks.

Notation and setup. Let $d, N \in \mathbb{N}$. Let $f(x; \theta)$ be an $l$-layered ReLU neural network with an input $x \in \mathbb{R}^d$ and parameters $\theta = (\theta_i)_{1 \leq i \leq n} \in \mathbb{R}^n$, where each weight $\theta_i$ is randomly sampled from a distribution $D_{\text{param}}$ over $\mathbb{R}$. A subnetwork of $f(x; \theta)$ is written as $f(x; \theta \odot m)$ where $m \in \{0, 1\}^n$ is a binary mask and "$\odot$" represents an element-wise multiplication.

Ramanujan et al. [26] empirically observed that we can train the randomly initialized neural network $f(x; \theta)$ by solving the following discrete optimization problem, which we call weight-pruning optimization:

$$\min_{m \in \{0, 1\}^n} \mathbb{E}_{(x, y) \sim D_{\text{labeled}}} \left[ L(f(x; \theta \odot m), y) \right],$$

where $D_{\text{labeled}}$ is a distribution on a set of labeled data $(x, y)$ and $L$ is a loss function. To solve this optimization problem, Ramanujan et al. [26] proposed an optimization algorithm, called edge-popup (Algorithm 1).

**Algorithm 1: Weight-pruning optimization by edge-popup [26]**

1. Initialize $\theta \sim D_{\text{param}}^n$, $s \sim D_{\text{score}}^n$; \hspace{1cm} // $D_{\text{param}}$ and $D_{\text{score}}$ are distributions over $\mathbb{R}$
2. while $k = 0, \ldots, N - 1$
3. \hspace{1cm} Sample a labeled data $(x, y) \sim D_{\text{labeled}}$;
4. \hspace{1cm} $m, s \leftarrow \text{TrainMask}(\theta, s; (x, y))$; \hspace{1cm} // Optimize importance scores $s$ and update $m$
5. end
6. return $m, \theta$

The TrainMask (Algorithm 2) is the key process in Algorithm 1. It has a latent variable $s = (s_i)_{1 \leq i \leq n} \in \mathbb{R}^n$, where each element $s_i$ represents an importance score of the corresponding weight $\theta_i$, and optimizes $s$ instead of directly optimizing the discrete variable $m$. Given the score $s$, the corresponding $m$ is computed by the function CalculateMask($s$), which returns $m = (m_i)_{1 \leq i \leq n}$ defined as follows: $m_i = 1$ if $s_i$ is top $100(1 - p)\%$ in $\{s_i\}_{1 \leq i \leq n}$, otherwise $m_i = 0$, where $p \in (0, 1)$ is a hyperparameter representing a sparsity rate of the pruned network. In the line 3 of Algorithm 2, $\text{SGD}_{\eta, \lambda, \mu}(s, g)$ returns the updated value of $s$ by stochastic gradient descent with a learning rate $\eta$, weight decay $\lambda$, momentum coefficient $\mu$, and gradient vector $g$.

On the theoretical side, Malach et al. [19] first provided a mathematical justification of the above empirical observation. They formulated it as an approximation theorem with some assumptions on the network width as follows.

**Theorem 2.1 (informal statement of Theorem 2.1 in [19])** Let $f_{\text{target}}(x)$ be an $l$-layered network with bounded weight matrices, and $g(x)$ be a randomly initialized $2l$-layered neural network. If the width of $g(x)$ is larger than $f_{\text{target}}(x)$ by the factor of a polynomial term, then there probably exists a subnetwork of $g(x)$ that approximates $f_{\text{target}}(x)$.
Algorithm 2: Pseudo code of TrainMask

1 Input: $\theta, s \in \mathbb{R}^n, (x, y)$: a labeled data;
2 $m \leftarrow \text{CalculateMask}(s)$; // Calculate the mask $m$ with the current scores
3 $s \leftarrow \text{SGD}(\theta, \lambda, \mu)(s, \nabla_{\bar{s} = m} L(f(x; \theta \odot \bar{s}), y))$; // Update $s$ by the gradient at $\bar{s} = m$
4 $m \leftarrow \text{CalculateMask}(s)$; // Calculate new mask $m$ with the updated scores
5 return $m, s$;

By considering a well-trained network as $f_{\text{target}}$, Theorem 2.1 indicates that pruning a sufficiently wide $g(x)$ may reveal a subnetwork which achieves good test accuracy as $f_{\text{target}}$, in principle.

In the follow-up works [25, 24], the assumption on the network width was improved by reducing the factor of the required width to a logarithmic term. However, Pensia et al. [25] showed that the logarithmic order is unavoidable at least in the case of $l = 1$. While their results imply the optimality of the logarithmic bound, it also means that we cannot further relax the assumption on the network width as long as we work in the same setting. This indicates a limitation of the weight-pruning optimization, i.e. the weight-pruning optimization can train only less expressive models than ones trained with the conventional weight-optimization like SGD, under a given amount of memory or network parameters.

3 Method

In this section, we present a novel method called weight-pruning with iterative randomization (IteRand) for randomly initialized neural networks.

As discussed in Section 2, although the original weight-pruning optimization (Algorithm 1) can achieve good accuracy, it still has a limitation in the expressive power under a fixed amount of memory or network parameters. Our method is designed to overcome this limitation. The main idea is to randomize pruned weights at each iteration of the weight-pruning optimization. As we prove in Section 4, this reduces the required size of an entire network to be pruned.

We use the same notation and setup as Section 2. In addition, we assume that each weight $\theta_i$ of the network $f(x; \theta)$ can be re-sampled from $D_{\text{param}}$ at each iteration of the weight-pruning optimization.

3.1 Algorithm

Algorithm 3 describes our proposed method, IteRand, which extends Algorithm 1. The differences from Algorithm 1 are lines 5-7. IteRand has a hyperparameter $K_{\text{per}} \in \mathbb{N} \geq 1$ (line 5). At the $k$-th iteration, whenever $k$ can be divided by $K_{\text{per}}$, pruned weights are randomized by $\text{Randomize}(\theta, m)$ function (line 6). There are multiple possible designs for $\text{Randomize}(\theta, m)$, which will be discussed in the next subsection.

Algorithm 3: Weight-pruning optimization with iterative randomization (IteRand)

1 Initialize $\theta \sim D_{\text{param}}, s \sim D_{\text{score}}$;
2 while $k = 0, \cdots, N - 1$ do
3 Sample a labeled data $(x, y) \sim D_{\text{labeled}}$;
4 $m, s \leftarrow \text{TrainMask}(\theta, s; (x, y))$;
5 if $k + 1$ can be divided by $K_{\text{per}}$ then // New if-block added to Algorithm 1
6 $\theta \leftarrow \text{Randomize}(\theta, m)$; // Randomize a subset of pruned weights
7 end
8 end
9 return $m, \theta$;

Note that $K_{\text{per}}$ controls how frequently the algorithm randomizes the pruned weights. Indeed the total number of the randomizing operations is $\lceil N/K_{\text{per}} \rceil$. If $K_{\text{per}}$ is too small, the algorithm is likely to be unstable because it may randomize even the important weights before their scores are well-optimized, and also the overhead of the randomizing operations cannot be ignored. In contrast,
if \( K_{\text{per}} \) is too large, the algorithm becomes almost same as the original weight-pruning Algorithm 1, and thus the effect of the randomization disappears. We fix \( K_{\text{per}} = 300 \) on CIFAR-10 (about 1 epoch) and \( K_{\text{per}} = 1000 \) on ImageNet (about 1/10 epochs) in our experiments (Section 5).

3.2 Designs of Randomize(\( \theta, m \))

Here, we discuss how to define Randomize(\( \theta, m \)) function. There are several possible ways to randomize a subset of the parameters \( \theta \).

**Naive randomization.** For any distribution \( D_{\text{param}} \), a naive definition of the randomization function (which we call naive randomization) can be given as follows.

\[
\text{Randomize}(\theta, m)_i := \begin{cases} 
\theta_i, & \text{if } m_i = 1 \\
\tilde{\theta}_i, & \text{otherwise}
\end{cases}
\]

where we denote the \( i \)-th component of Randomize(\( \theta, m \)) as Randomize(\( \theta, m \))\(_i \), and each \( \tilde{\theta}_i \in \mathbb{R} \) is a random variable with the distribution \( D_{\text{param}} \). Also this can be written in another form as

\[
\text{Randomize}(\theta, m) := \theta \odot m + \tilde{\theta} \odot (1 - m),
\]

where \( \tilde{\theta} = (\tilde{\theta}_i)_{1 \leq i \leq n} \in \mathbb{R}^n \) is a random variable with the distribution \( D_{\text{param}}^{n} \).

**Partial randomization.** The naive randomization (Eq. (2)) is likely to be unstable because it entirely replaces all pruned weights with random values every \( K_{\text{per}} \) iteration. To increase the stability, we modify the definition of the naive randomization as it replaces a randomly chosen subset of the pruned weights as follows (which we call partial randomization):

\[
\text{Randomize}(\theta, m) := \theta \odot m + \left( \theta \odot (1 - b_r) + \tilde{\theta} \odot b_r \right) \odot (1 - m),
\]

where \( \tilde{\theta} \in \mathbb{R}^n \) is the same as in Eq. (2), \( r \in [0, 1] \) is a hyperparameter and \( b_r = (b_{r,i})_{1 \leq i \leq n} \in \{0, 1\}^n \) is a binary vector whose each element is sampled from the Bernoulli distribution \( \text{Bernoulli}(r) \), i.e. \( b_{r,i} = 1 \) with probability \( r \) and \( b_{r,i} = 0 \) with probability \( 1 - r \).

The partial randomization replaces randomly chosen \( 100r\% \) of all pruned weights with random values. Note that, when \( r = 1 \), the partial randomization is equivalent to the naive randomization (Eq. (2)). In contrast, when \( r = 0 \), it never randomizes any weights and thus is equivalent to Algorithm 1.

In Figure 1, we observe that \( r = 0.1 \) works well with various network architectures on CIFAR-10, where we use the Kaiming uniform distribution (whose definition will be given in Section 5) for \( D_{\text{param}} \) and \( K_{\text{per}} = 300 \).

![Figure 1: Analysis on \( r \). We compare partial randomizations with \( r \in \{0.0, 0.01, 0.1, 1.0\} \) applied to CNNs. The y-axis is the validation accuracy on CIFAR-10. \( r = 0.1 \) achieves better mean accuracy for every CNNs.](image)

4 Theoretical justification

In this section, we present a theoretical justification for our iterative randomization approach on the weight-pruning of randomly initialized neural networks.

4.1 Setup

We consider a target neural network \( f : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_l} \) of depth \( l \), which is described as follows.

\[
f(x) = F_l \sigma(F_{l-1} \sigma(\cdots F_1(x) \cdots)),
\]

where \( x \) is a \( d_0 \)-dimensional real vector, \( \sigma \) is the ReLU activation, and \( F_i \) is a \( d_i \times d_{i-1} \) matrix. Our objective is to approximate the target network \( f(x) \) by pruning a randomly initialized neural network \( g(x) \), which tends to be larger than the target network.
Similar to the previous works [19, 25], we assume that \( g(x) \) is twice as deep as the target network \( f(x) \). Thus, \( g(x) \) can be described as
\[
g(x) = G_{2l} \sigma(G_{2l-1} \sigma(\cdots G_1(x) \cdots)),
\]
where \( G_j \) is a \( \tilde{d}_j \times \tilde{d}_{j-1} \) matrix \((j = 1, \ldots, 2l)\) with \( \tilde{d}_{2i} = d_i \). Each element of the matrix \( G_j \) is assumed to be drawn from the uniform distribution \( U[-1, 1] \). Since there is a one-to-one correspondence between pruned networks of \( g(x) \) and sequences of binary matrices \( M = \{M_j\}_{j=1, \ldots, 2l} \) with \( M_j \in \{0, 1\}^{	ilde{d}_j \times \tilde{d}_{j-1}} \), every pruned network of \( g(x) \) can be described as
\[
g_M(x) = (G_{2l} \odot M_{2l})\sigma((G_{2l-1} \odot M_{2l-1})\sigma(\cdots (G_1 \odot M_1)(x) \cdots)).
\]
Under these setups, we recall that the previous works showed that, with high probability, there exists a subnetwork of \( g(x) \) that approximates \( f(x) \) when the width of \( g(x) \) is larger than \( f(x) \) by polynomial factors [19] or logarithmic factors [25, 24].

\[ \text{Theorem 4.1 (Main Theorem)} \]

Fix \( \epsilon, \delta > 0 \), and we assume that \( \|F_i\|_{\text{Frob}} \leq 1 \). Let \( R \in \mathbb{N} \), and assume that \( g(x) \) satisfies the re-sampling assumption for \( R \).

If \( \tilde{d}_{2i-1} \geq 2d_{i-1} \frac{4d_i^2d_i^2}{\epsilon^2 \log \frac{2d_{i-1}d_i}{\delta}} \) holds for all \( i = 1, \ldots, l \), then with probability at least \( 1 - \delta \), there exist binary matrices \( M = \{M_j\}_{1 \leq j \leq 2l} \) such that
\[
\|f(x) - g_M(x)\|_2 \leq \epsilon, \text{ for } \|x\|_\infty \leq 1. \tag{7}
\]

In particular, if \( R \) is larger than \( \frac{8d_{i-1}}{\epsilon} \sqrt{d_i \log \frac{2d_{i-1}d_i}{\delta}} \), then \( \tilde{d}_{2i-1} = 2d_{i-1} \) is enough.

Theorem 4.1 shows that the iterative randomization is provably helpful to approximate wider networks in the weight-pruning optimization of a random network. In fact, the required width for \( g(x) \) in Theorem 4.1 is reduced to even twice as wide as \( f(x) \) when the number of re-sampling is sufficiently large, in contrast to the prior results without re-sampling assumption where the required width is logarithmically wider than \( f(x) \)[25, 24]. This means that, under a fixed amount of parameters of \( g(x) \), we can achieve a higher accuracy by weight-pruning of \( g(x) \) with iterative randomization since a wider target network has a higher model capacity.

In the rest of this section we present the core ideas by proving the simplest case \((l = d_0 = d_1 = 1)\) of the theorem, while the full proof of Theorem 4.1 is given in Appendix A. Note that the full proof is obtained essentially by applying the argument for the simplest case inductively on the widths and depths.

\[ \text{4.3 Proof ideas for Theorem 4.1} \]

Let us consider the case of \( l = d_0 = d_1 = 1 \). Then the target network \( f(x) \) can be written as \( f(x) = wx : \mathbb{R} \to \mathbb{R} \), where \( w \in \mathbb{R} \), and \( g(x) \) can be written as \( g(x) = v^T \sigma(wx) \), where \( u, v \in \mathbb{R}^{\tilde{d}_1} \). Also, subnetworks of \( g(x) \) can be written as \( g_m(x) = (v \odot m)^T \sigma((u \odot m)x) \) for some \( m \in \{0, 1\}^{\tilde{d}_1} \).

There are two technical points in our proof. The first point is the following splitting of \( f(x) \):
\[
f(x) = w\sigma(x) - w\sigma(-x), \tag{8}
\]
for any \( x \in \mathbb{R} \). This splitting is very similar to the one used in the previous works [19, 25, 24]:
\[
f(x) = \sigma(wx) - \sigma(-wx). \tag{9}
\]
However, if we use the latter splitting Eq. (9), it turns out that we cannot obtain the lower bound of $d_{2l-1}$ in Theorem 4.1 when $d_0 > 0$. (Here we do not treat this case, but the proof for $d_0 > 0$ is given in Appendix A.) Thus we need to use our splitting Eq. (8) instead.

Using Eq. (8), we can give another proof of the following approximation result without iterative randomization, which was already shown in the previous work [19].

**Lemma 4.2** Fix $\epsilon, \delta \in (0, 1), w \in [-1, 1], d \in \mathbb{N}$. Let $u, v \sim U[-1,1]$ be uniformly random weights of a 2-layered neural network $g(x) := v^T \sigma(u \cdot x)$. If $d \geq 2\left( \frac{16}{\epsilon^2} \log(\frac{2}{\delta}) \right)$ holds, then with probability at least $1 - \delta$,

$$|w x - g_m(x)| \leq \epsilon, \text{ for all } x \in \mathbb{R}, |x| \leq 1,$$

where $g_m(x) := (v \odot m)^T \sigma(u \cdot x)$ for some $m \in \{0, 1\}^d$.

**Proof (sketch):** We assume that $d$ is an even number as $d = 2d'$ so that we can split an index set $\{0, \ldots, d - 1\}$ of $d$ hidden neurons of $g(x)$ into $I = \{0, \ldots, d' - 1\}$ and $J = \{d', \ldots, d - 1\}$. Then we have the corresponding subnetworks $g_I(x) := \sum_{k \in I} v_k \sigma(u_k x)$ and $g_J(x) := \sum_{k \in J} v_k \sigma(u_k x)$, which satisfy the equation $g(x) = g_I(x) + g_J(x)$.

By the splitting Eq. (8), it is enough to consider the probabilities for approximating $w \sigma(x)$ by a subnetwork of $g_I(x)$ and for approximating $-w \sigma(-x)$ by a subnetwork of $g_J(x)$. Now we have

$$\mathbb{P}\left( \exists i \in I \text{ s.t. } |u_i - w| \leq \epsilon, |v_i - w| \leq \epsilon \frac{d'}{2} \right) \leq \left( 1 - \frac{\epsilon^2}{16} \right)^{d'} \leq \frac{\delta}{2},$$

$$(11)$$

and

$$\mathbb{P}\left( \exists j \in J \text{ s.t. } |u_j + 1| \leq \epsilon, |v_j + w| \leq \epsilon \frac{d'}{2} \right) \leq \left( 1 - \frac{\epsilon^2}{16} \right)^{d'} \leq \frac{\delta}{2},$$

$$(12)$$

for $d' \geq \frac{16}{\epsilon^2} \log\left( \frac{2}{\delta} \right)$, by a standard argument of the uniform distribution and the inequality $e^x \geq 1 + x$ for $x \geq 0$. By the union bound, with probability at least $1 - \delta$, we have $i \in I$ and $j \in J$ such that

$$|w \sigma(x) - v_i \sigma(u_i x)| \leq \epsilon \frac{d'}{2},$$

$$|w \sigma(-x) - v_i \sigma(u_i x)| \leq \epsilon \frac{d'}{2}.$$

Combining these inequalities, we finish the proof. 

The second point of our proof is introducing projection maps to leverage the re-sampling assumption, as follows. As in the proof of Lemma 4.2, we assume that $d = 2d'$ for some $d' \in \mathbb{N}$ and let $I = \{0, \ldots, d' - 1\}$, $J = \{d', \ldots, d - 1\}$. Now we define a projection map

$$\pi : \tilde{I} \to I, \ k \mapsto \lfloor k/R \rfloor,$$

$$(13)$$

where $\tilde{I} := \{0, \ldots, d' R - 1\}$, and $\lfloor \cdot \rfloor$ denotes the floor function. Similarly for $J$, we can define $J := \{d' R, \ldots, d R - 1\}$ and the corresponding projection map. Using these projection maps, we can extend Lemma 4.2 to the one with the re-sampling assumption, which is the special case of Theorem 4.1:

**Proposition 4.3** (Theorem 4.1 with $l = d_0 = d_1 = 1$) Fix $\epsilon, \delta \in (0, 1), w \in [-1, 1], d \in \mathbb{N}$. Let $u, v \sim U[-1,1]^d$ be uniformly random weights of a 2-layered neural network $g(x) := v^T \sigma(u \cdot x)$. Let $R \in \mathbb{N}$ and we assume that each element of $u$ and $v$ can be re-sampled with replacement up to $R - 1$ times. If $d \geq 2\left( \frac{16}{\epsilon^2} \log\left( \frac{2}{\delta} \right) \right)$ holds, then with probability at least $1 - \delta$,

$$|w x - g_m(x)| \leq \epsilon, \text{ for all } x \in \mathbb{R}, |x| \leq 1,$$

$$(14)$$

where $g_m(x) := (v \odot m)^T \sigma(u \cdot x)$ for some $m \in \{0, 1\}^d$.

**Proof (sketch):** Similarly to the proof of Lemma 4.2, we utilize the splitting Eq. (8). We mainly argue on the approximation of $w \sigma(x)$ since the argument for approximating $-w \sigma(x)$ is parallel.
By the assumption that each element of \( \mathbf{u} \) and \( \mathbf{v} \) can be re-sampled up to \( R - 1 \) times, we can replace the probability in Eq. (11) in the proof of Lemma 4.2, using the projection map \( \pi : \tilde{I} \to I \), by
\[
P \left( \tilde{I} \right. \left. \left. \tilde{I} \mid \pi(1), \cdots, \pi(k), \tilde{I} \mid \pi(1) = \pi(i_2) \right) \leq \frac{\epsilon}{2}, \right) \leq \frac{\epsilon}{2}, \right)
\]
where \( \tilde{u_1}, \cdots, \tilde{u}_{R}, \tilde{v_1}, \cdots, \tilde{v}_{R} \sim U[-1, 1] \). Indeed, since we have
\[
\# \{ (i_1, i_2) \in \tilde{I} \times \tilde{I} : \pi(i_1) = \pi(i_2) \} = d'R^2, \right)
\]
we can evaluate the probability Eq. (15) as
\[
\text{Eq. (15)} \leq \left( 1 - \frac{\epsilon^2}{16} \right) d'R^2 \lesssim \frac{\delta}{2}, \right)
\]
for \( d' \geq \frac{16}{\epsilon^2 R^2} \log \left( \frac{d}{\epsilon} \right) \). Eq. (17) can play the same role as Eq. (11) in the proof of Lemma 4.2.
Parallel argument can be applied for the approximation of \(-w(x)\) by replacing \( I \) with \( J \). The rest of the proof is the same as Lemma 4.2.

5 Experiments

In this section, we perform several experiments to evaluate our proposed method, IteRand (Algorithm 3). Our main aim is to empirically verify the parameter efficiency of IteRand, compared with edge-popup [26] (Algorithm 1) on which IteRand is based. Specifically, we demonstrate that IteRand can achieve better accuracy than edge-popup under a given amount of network parameters. In all experiments, we used the partial randomization with \( r = 0.1 \) (Eq. (3)) for Randomize in Algorithm 3.

Setup. We used two vision datasets: CIFAR-10 [13] and ImageNet [28]. CIFAR-10 is a small-scale dataset of 32 × 32 images with 10 class labels. It has 50k images for training and 10k for testing. We randomly split the 50k training images into 45k for actual training and 5k for validation. ImageNet is a dataset of 224 × 224 images with 1000 class labels. It has the train set of 1.28 million images and the validation set of 50k images. We randomly split the training images into 99 : 1, and used the former for actual training and the latter for validating models. When testing models, we used the validation set of ImageNet (which we refer to as the test set). For network architectures, we used multiple convolutional neural networks (CNNs): Conv6 [6] as a shallow network and ResNets [11] as deep networks. Conv6 is a 6-layered VGG-like CNN, which is also used in the prior work [26]. ResNets are more practical CNNs with skip connections and batch normalization layers. Following the settings in Ramanujan et al. [26], we used non-affine batch normalization layers, which are layers that only normalize their inputs and do not apply any affine transform, when training ResNets with edge-popup and IteRand. All of our experiments were performed with 1 GPU (NVIDIA GTX 1080 Ti, 11GB) for CIFAR-10 and 2 GPUs (NVIDIA V100, 16GB) for ImageNet. The details of the network architectures and hyperparameters for training are given in Appendix B.

Parameter distributions. With the same notation as Section 2, both IteRand and edge-popup requires two distributions: \( D_{\text{param}} \) and \( D_{\text{score}} \). In our experiments, we consider Kaiming uniform (KU) and signed Kaiming constant (SC) distribution. The KU distribution is the uniform distribution over the interval \( [-\sqrt{\frac{6}{\text{fanin}}}, \sqrt{\frac{6}{\text{fanin}}} \] where \( \text{fanin} \) is the constant defined for each layer of ReLU neural networks [1][10]. The SC distribution is the uniform distribution over the two-valued set \( \{-\sqrt{\frac{2}{\text{fanin}}}, \sqrt{\frac{2}{\text{fanin}}} \} \), which is introduced by Ramanujan et al. [26]. We fix \( D_{\text{score}} \) to the KU distribution, and use the KU or SC distribution for \( D_{\text{param}} \).

5.1 Varying the network width

To demonstrate the parameter efficiency, we introduce a hyperparameter \( \rho \) of the width factor for Conv6, ResNet18 and ResNet34. The details of this modification are given in Appendix B. We train and test these networks on CIFAR-10 using IteRand, edge-popup and SGD, varying the width factor \( \rho \) in \{0.25, 0.5, 1.0, 2.0\} for each network (Figure 2). In the experiments for IteRand and edge-popup, we used the sparsity rate of \( \rho = 0.5 \) for Conv6 and \( \rho = 0.6 \) for ResNet18 and ResNet34. Our method outperforms the baseline method for various widths. The difference in accuracy is large both when the width is small (\( \rho \leq 0.5 \)), and when \( D_{\text{param}} \) is the KU distribution where edge-popup struggles.
Figure 2: We train and evaluate three CNNs (Conv6, ResNet18 and ResNet34) with various widths on CIFAR-10. The x-axis is the width factor \( \rho \) and the y-axis is the accuracy on the test set. We plot the mean \( \pm \) standard deviation over three runs for each experiment. KU/SC in the legend represents the distribution used for \( D_{\text{param}} \).

(a) Network Size vs. Test Accuracy. (b) Training Curves.

Figure 3: We compare IteRand with edge-popup on ImageNet. We used a fixed \( p = 0.7 \) for the sparsity rate (Section 2) following Ramanujan et al. [26]. Figure (a) shows parameter-accuracy tradeoff curves on the test set. In Figure (b), we plot the train/val accuracy of ResNet50 during optimization. IteRand outperforms edge-popup from early epochs.

5.2 ImageNet experiments

The parameter efficiency of our method is also confirmed on ImageNet (Figure 3a). ImageNet is more difficult than CIFAR-10, and thus more complexity is required for networks to achieve competitive performance. Since our method can increase the network complexity as shown in Section 4, the effect is significant in ImageNet especially when the complexity is limited such as ResNet34.

In addition to the parameter efficiency, we also observe the effect of iterative randomization on the behavior of optimization process, by plotting training curves (Figure 3b). Surprisingly, IteRand achieves significantly better performance than edge-popup at the early stage of the optimization, which indicates that the iterative randomization accelerates the optimization process especially when the number of iterations is limited.

6 Related work

Lottery ticket hypothesis Frankle and Carbin [6] originally proposed the lottery ticket hypothesis. Many properties and applications have been studied in subsequent works, such as transferability of winning subnetworks [22], sparsification before training [15, 31, 29] and further ablation studies on the hypothesis [34].

Zhou et al [34] surprisingly showed that only pruning randomly initialized networks without any optimization on their weights can be a training method surprisingly. Ramanujan et al. [26] went
further by proposing an effective pruning algorithm on random networks, called edge-popup, and achieved competitive accuracy compared with standard training algorithms by weight-optimization [27]. Malach et al. [19] mathematically formalized their pruning-only approach as an approximation problem for a target network and proved it with lower bound condition on the width of random networks to be pruned. Subsequent works [25, 24] successfully relaxed the lower bound to the logarithmic factor wider than the width of a target network. Our work can be seen as an extension of their works to allow re-sampling of the weights of the random networks for finite $R$ times, and we showed that the logarithmic factor can be reduced to a constant one when $R$ is large enough. (See Section 4.)

Neural network pruning and regrowth Studies of finding sparse structures of neural networks date back to the late 1980s [9, 14]. There are many approaches to sparsify networks, such as magnitude-based pruning [8], $L_0$ regularization [17] and variational dropout [21]. Although these methods only focus on pruning unnecessary weights from the networks, there are several studies on re-adding new weights during sparsification [12] to maintain the model complexity of the sparse networks. Han et al. [7] proposed a dense-sparse-dense (DSD) training algorithm consisting of three stages: dense training, pruning, and recovering the pruned weights as zeros followed by dense training. Mocanu et al. [20] proposed sparse evolutionary training (SET), which repeats pruning and regrowth with random values at the end of each training epoch. Pruning algorithms proposed in other works [2, 23, 4] are designed to recover pruned weights by zero-initialization instead of random values, so that the recovered weights do not affect the outputs of the networks. While these methods are similar to our iterative randomization method in terms of the re-adding processes, all of them use weight-optimization to train networks including re-added weights, in contrast to our pruning-only approach.

7 Limitations
There are several limitations in our theoretical results. (1) Theorem 4.1 indicates only the existence of subnetworks that approximate a given neural network, not whether our method works well empirically. (2) The theorem focused on the case when the parameter distribution $D_{\text{param}}$ is the uniform distribution over the interval $[-1, 1]$. Thus, generalizing our theorem to other distributions, such as the uniform distribution over binary values $\{-1, 1\}$ [3], is left for future work. (3) The required width given in the theorem may not be optimal. Indeed, prior work [25] showed that we can reduce the polynomial factors in the required width to logarithmic ones in the case when the number of re-sampling operations $R = 1$. Whether we can reduce the required width for $R > 1$ remains an open question.

Also our algorithm (IteRand) and its empirical results have several limitations. (1) Pruning randomly-initialized networks without any randomization can reduce the storage cost by saving only a single random seed and the binary mask representing the optimal subnetwork. However, if we save the network pruned with IteRand in the same way, it requires more storage cost: $R$ random seeds and $R$ binary masks, where $R$ is the number of re-samplings. (2) Although our method can be applied with any score-based pruning algorithms (e.g. Zhou et al [34] and Wang et al [32]), we evaluated our method only combined with edge-popup [26], which is the state-of-the-art algorithm for pruning random networks. Since our theoretical results do not depend on any pruning algorithms, we expect that our method can be effectively combined with better pruning algorithms that emerge in the future. (3) We performed experiments mainly on the tasks of image classification. An intriguing question is how effectively our method works on other tasks such as language understanding, audio recognition, and deep reinforcement learning with various network architectures.

8 Conclusion
In this paper, we proposed a novel framework of iterative randomization (IteRand) for pruning randomly initialized neural networks. IteRand can virtually increase the network widths without any additional memory consumption, by randomizing pruned weights of the networks iteratively during the pruning procedure. We verified its parameter efficiency both theoretically and empirically.

Our results indicate that the weight-pruning of random networks may become a practical approach to train the networks when we apply the randomizing operations enough times. This opens up the
possibility that the weight-pruning can be used instead of the standard weight-optimization within
the same memory budget.

Acknowledgement

The authors thank Kyosuke Nishida and Hengjin Tang for valuable discussions in the early phase
of our study. We also thank Osamu Saisho for helpful comments on the manuscript. We appreciate
CCI team in NTT for building and maintaining their computational cluster on which most of our
experiments were computed.

References

[1] torch.nn.init – PyTorch 1.8.1 documentation. https://pytorch.org/docs/stable/nn.
init.html. Accessed: 2021-10-21.
[2] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring:
Training very sparse deep networks. International Conference on Learning Representations,
2018.
[3] James Diffenderfer and Bhavya Kailkhura. Multi-prize lottery ticket hypothesis: Finding
accurate binary neural networks by pruning a randomly weighted network. In International
Conference on Learning Representations, 2021.
[4] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the
lottery: Making all tickets winners. In International Conference on Machine Learning, pages
2943–2952. PMLR, 2020.
[5] Facebook. facebookresearch/open_lth. https://github.com/facebookresearch/open_1th.
Accessed: 2021-10-21.
[6] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable
neural networks. In International Conference on Learning Representations, 2019.
[7] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter
Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural
networks. International Conference on Learning Representations, 2017.
[8] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections
for efficient neural networks. In Proceedings of the 28th International Conference on Neural
Information Processing Systems-Volume 1, pages 1135–1143, 2015.
[9] Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with
back-propagation. Advances in Neural Information Processing Systems, 1:177–185, 1988.
[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers:
Surpassing human-level performance on imagenet classification. In 2015 IEEE International
Conference on Computer Vision (ICCV), pages 1026–1034. IEEE Computer Society, 2015.
[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image
recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),
pages 770–778. IEEE Computer Society, 2016.
[12] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in
deep learning: Pruning and growth for efficient inference and training in neural networks. arXiv
preprint arXiv:2102.00554, 2021.
[13] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.
[14] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In Advances in Neural
Information Processing Systems, pages 598–605, 1990.
[15] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning
based on connection sensitivity. In International Conference on Learning Representations, 2018.
[16] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In
International Conference on Learning Representations, 2017.
[17] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through $l_0$ regularization. In International Conference on Learning Representations, 2018.

[18] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[19] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In International Conference on Machine Learning, pages 6682–6691. PMLR, 2020.

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

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

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

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

[24] Laurent Orseau, Marcus Hutter, and Omar Rivasplata. Logarithmic pruning is all you need. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33. Curran Associates, Inc., 2020.

[25] Ankit Pensia, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos. Optimal lottery tickets via subset sum: Logarithmic over-parameterization is sufficient. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 2599–2610. Curran Associates, Inc., 2020.

[26] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What’s hidden in a randomly weighted neural network? In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 11893–11902, 2020.

[27] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. Nature, 323(6088):533–536, 1986.

[28] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252, 2015.

[29] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, Advances in Neural Information Processing Systems, volume 33, pages 6377–6389. Curran Associates, Inc., 2020.

[30] Ben Trevett. bentrevett/pytorch-sentiment-analysis. https://github.com/bentrevett/pytorch-sentiment-analysis. Accessed: 2021-10-21.

[31] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In International Conference on Learning Representations, 2020.

[32] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu. Pruning from scratch. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 34, pages 12273–12280, 2020.

[33] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, Proceedings of the British Machine Vision Conference (BMVC), pages 87.1–87.12. BMVA Press, September 2016.
A Proof of main theorem (Theorem 4.1)

A.1 Settings and main theorem

Let \( d_0, \ldots, d_l \in \mathbb{N}_{\geq 1} \). We consider a target neural network \( f : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_l} \) of depth \( l \), which is described as follows.

\[
f(x) = F_l \sigma(F_{l-1} \sigma(\cdots F_1(x) \cdots)),
\]

where \( x \) is a \( d_0 \)-dimensional real vector, \( \sigma \) is the ReLU activation, and \( F_j \) is a \( d_j \times d_{j-1} \) matrix. Our objective is to approximate the target network \( f(x) \) by pruning a randomly initialized neural network \( g(x) \), which tends to be larger than the target network.

Similar to the previous works [19, 25], we assume that \( g(x) \) is twice as deep as the target network \( f(x) \). Thus, \( g(x) \) can be described as

\[
g(x) = G_{2l} \sigma(G_{2l-1} \sigma(\cdots G_1(x) \cdots)),
\]

where \( G_j \) is a \( \tilde{d}_j \times \tilde{d}_{j-1} \) matrix \( (\tilde{d}_j \in \mathbb{N}_{\geq 1} \) for \( j = 1, \ldots, 2l \) with \( \tilde{d}_{2l} = d_l \). Each element of the matrix \( G_j \) is assumed to be drawn from the uniform distribution \( \mathbb{U}[-1, 1] \). Since there is a one-to-one correspondence between pruned networks of \( g(x) \) and sequences of binary matrices \( M = \{ M_j \}_{j=1,\ldots,2l} \) with \( M_j \in \{ 0, 1 \}^{\tilde{d}_j \times \tilde{d}_{j-1}} \), every pruned network of \( g(x) \) can be described as

\[
g_M(x) = (G_{2l} \odot M_{2l}) \sigma((G_{2l-1} \odot M_{2l-1}) \sigma(\cdots (G_1 \odot M_1)(x) \cdots)).
\]

We introduce an idealized assumption on \( g(x) \) for a given number \( R \in \mathbb{N}_{\geq 1} \): each element of the weight matrix \( G_j \) can be re-sampled with replacement from the uniform distribution \( \mathbb{U}[-1, 1] \) up to \( R - 1 \) times, for all \( j = 1, \ldots, 2l \) (re-sampling assumption for \( R \)). Here, re-sampling with replacement means that we sample a new value for an element of \( G_j \) and replace the old value of the element by the new one.

Under this re-sampling assumption, we describe our main theorem as follows.

**Theorem A.1 (Main Theorem)** Fix \( \epsilon, \delta > 0 \), and we assume that \( \| F_i \|_{\text{Frob}} \leq 1 \). Let \( R \in \mathbb{N} \) and we assume that each element of \( G_i \) can be re-sampled with replacement from the uniform distribution \( \mathbb{U}[-1, 1] \) up to \( R - 1 \) times.

If \( \tilde{d}_{2i-1} \geq 2d_{i-1} \left( \frac{64l^2 \log(d_{i-1})}{\epsilon^2 R^2 \log(2d_i d_{i-1})} \right) \) holds for all \( i = 1, \ldots, l \), then with probability at least \( 1 - \delta \), there exist binary matrices \( M = \{ M_j \}_{1 \leq j \leq 2l} \) such that

\[
\| f(x) - g_M(x) \|_2 \leq \epsilon, \text{ for } \| x \|_\infty \leq 1.
\]

In particular, if \( R \) is larger than \( \frac{8ld_{i-1}^2}{\epsilon^2} \left( d_i \log(2d_{i-1} d_i) \right) \), then \( \tilde{d}_{2i-1} = 2d_{i-1} \) is enough.

A.2 Proof of Theorem A.1

Our proof is based on the following simple observation, similar to the arguments in Malach et al. [19].

**Lemma A.2** Fix some \( n \in \mathbb{N}, \alpha \in [-1, 1] \) and \( \epsilon, \delta \in (0, 1) \). Let \( X_1, \ldots, X_n \sim \mathbb{U}[-1, 1] \). If \( n \geq \frac{\delta}{\epsilon^2} \log(\frac{1}{\delta}) \) holds, then with probability at least \( 1 - \delta \), we have

\[
|\alpha - X_i| \leq \epsilon,
\]

for some \( i \in \{ 1, \ldots, n \} \).
We assume that
Thus it follows that
Fix Lemma A.4
Now we extend Lemma A.3 to the one with the re-sampling assumption.
Combining these inequalities and Eq. (24), we finish the proof.

First, we consider to approximate a single variable linear function \( f(x) = wx : \mathbb{R} \to \mathbb{R}, w \in \mathbb{R} \) by some subnetwork of 2-layered neural network \( g(x) \) with \( d \) hidden neurons, \textit{without} the re-sampling assumption. Note that this is the same setting as in Malach et al. [19], but we give another proof so that we can later extend it to the one with the resampling assumption.

\textbf{Lemma A.3} Fix \( \epsilon, \delta \in (0,1), w \in [-1,1], d \in \mathbb{N} \). Let \( u, v \sim U[-1,1]^d \) be uniformly random weights of a 2-layered neural network \( g(x) := v^T \sigma(u \cdot x) \). If \( d \geq 2 \left\lceil \frac{16}{\epsilon^2} \log \left( \frac{\delta}{\alpha} \right) \right\rceil \) holds, then with probability at least \( 1 - \delta \),

\[
|wx - g_m(x)| \leq \epsilon, \text{ for all } x \in \mathbb{R}, |x| \leq 1,
\]

where \( g_m(x) := (v \odot m)^T \sigma(u \cdot x) \) for some \( m \in \{0,1\}^d \).

\textbf{Proof:} The core idea is to decompose \( wx \) as

\[
wx = w(\sigma(x) - \sigma(-x)) = w\sigma(x) - w\sigma(-x).
\]

We assume that \( d \) is an even number as \( d = 2d' \) so that we can split an index set \( \{1, \cdots, d\} \) of hidden neurons of \( g(x) \) into \( I = \{1, \cdots, d'\} \) and \( J = \{d' + 1, \cdots, d\} \). Then we have the corresponding subnetworks \( g_I(x) \) and \( g_J(x) \) given by \( g_I(x) := \sum_{k \in I} v_k \sigma(u_k x) \) and \( g_J(x) := \sum_{k \in J} v_k \sigma(u_k x) \), which satisfy the equation \( g(x) = g_I(x) + g_J(x) \).

From Eq. (24), it is enough to consider the probabilities for approximating \( w\sigma(x) \) by a subnetwork of \( g_I(x) \) and for approximating \( -w\sigma(-x) \) by a subnetwork of \( g_J(x) \). Now we have

\[
\mathbb{P} \left( \exists i \in I \text{ s.t. } |u_i - 1| \leq \frac{\epsilon}{2}, |v_i - w| \leq \frac{\epsilon}{2} \right) \leq \left( 1 - \frac{\epsilon^2}{16} \right)^{d'} \leq \frac{\delta}{2},
\]

\[
\mathbb{P} \left( \exists j \in J \text{ s.t. } |u_j + 1| \leq \frac{\epsilon}{2}, |v_j + w| \leq \frac{\epsilon}{2} \right) \leq \left( 1 - \frac{\epsilon^2}{16} \right)^{d'} \leq \frac{\delta}{2},
\]

for \( d' \geq \frac{16}{\epsilon^2} \log \left( \frac{\delta}{\alpha} \right) \) as well as in the proof of Lemma A.2. By using the union bound, with probability at least \( 1 - \delta \), we have \( i \in I \) and \( j \in J \) such that

\[
|w\sigma(x) - v_i \sigma(u_i x)| \leq \frac{\epsilon}{2},
\]

\[
|-w\sigma(-x) - v_j \sigma(u_j x)| \leq \frac{\epsilon}{2}.
\]

Combining these inequalities and Eq. (24), we finish the proof.

Now we extend Lemma A.3 to the one with the re-sampling assumption.

\textbf{Lemma A.4} Fix \( \epsilon, \delta \in (0,1), w \in [-1,1], d \in \mathbb{N} \). Let \( u, v \sim U[-1,1]^d \) be uniformly random weights of a 2-layered neural network \( g(x) := v^T \sigma(u \cdot x) \). Let \( R \in \mathbb{N} \) and we assume that each elements of \( u \) and \( v \) can be re-sampled with replacement up to \( R - 1 \) times. If \( d \geq 2 \left\lceil \frac{16}{\epsilon^2} \log \left( \frac{\delta}{\alpha} \right) \right\rceil \) holds, then with probability at least \( 1 - \delta \),

\[
|wx - g_m(x)| \leq \epsilon, \text{ for all } x \in \mathbb{R}, |x| \leq 1,
\]

where \( g_m(x) := (v \odot m)^T \sigma(u \cdot x) \) for some \( m \in \{0,1\}^d \).
where \( \tilde{\vec{v}}_{1}, \cdots, \tilde{\vec{v}}_{d'}, \vec{v}_{1}, \cdots, \vec{v}_{d'} \sim U[-1, 1]. \) Since we have
\[
\#\{(i_1, i_2) \in \tilde{I} \times \tilde{I} : \pi(i_1) = \pi(i_2)\} = d'R^2, \tag{29}
\]
we can evaluate the probability Eq. (28) as
\[
\text{Eq. (28)} \leq \left(1 - \frac{\epsilon^2}{16}\right)^{d'R^2} \leq \frac{\delta}{2},
\]
for \( d' \geq \frac{10}{\epsilon^2 R^2} \log \left(\frac{\delta}{8}\right). \) The rest of the proof is same as Lemma A.3.

Then, we generalize the above lemma to the case which the target function \( f(x) \) is a single-variable linear map with higher output dimensions.

**Lemma A.5** Fix \( \epsilon, \delta \in (0, 1), d_1, d_2 \in \mathbb{N}, w \in [-1, 1]^{d_2}. \) Let \( u \sim U[-1, 1]^{d_1}, V \sim U[-1, 1]^{d_2 \times d_1} \) be uniformly random weights of a 2-layered neural network \( g(x) \) for each \( x \in \mathbb{R}, \) and assume that each elements of \( u \) and \( V \) can be re-sampled with replacement up to \( R - 1 \) times.

If \( d \geq 2\left[\frac{16d_2}{\epsilon^2 R^2} \log \left(\frac{2d_2}{\delta}\right)\right] \) holds, then with probability at least \( 1 - \delta, \)
\[
\|w \cdot x - g_M(x)\|_2 \leq \epsilon, \text{ for all } x \in \mathbb{R}, |x| \leq 1, \tag{30}
\]
where \( g_M(x) := (V \odot M)^T \sigma(u \cdot x) \) for some \( M \in \{0, 1\}^d. \)

**Proof:** We denote \( V = \{V_{ki}\}_{1 \leq k \leq d_2, 1 \leq i \leq d_1}. \) As in the proof of Lemma A.3, we assume \( d_1 = 2d'_1 \) and split the index set \( \{1, \cdots, d_1\} \) into \( \tilde{I} = \{1, \cdots, d'_1\}, J = \{d'_1 + 1, \cdots, d_1\}. \) Also we consider the corresponding subnetworks of \( g(x) \):
\[
g_I(x) := \left(\sum_{i \in I} V_{ki} \sigma(u_{ki}x)\right)_{1 \leq k \leq d_2}, \quad g_J(x) := \left(\sum_{i \in J} V_{ki} \sigma(u_{ki}x)\right)_{1 \leq k \leq d_2}.
\]
Similar as the proof of Lemma A.3 and Lemma A.4, it is enough to show that there probably exists a subnetwork of \( g_I(x) \) which approximates \( w \cdot \sigma(x) \) and also that there simultaneously exists a subnetwork of \( g_J(x) \) which approximates \( -w \cdot \sigma(-x) \).

For simplicity, we focus on \( g_I(x) \) in the following argument, but same conclusion holds for \( g_J(x) \) as well. Fix \( k \in \{1, \cdots, d_2\} \). Then we consider the following probability,
\[
P \left( \tilde{u}_{i_1}, \tilde{u}_{i_2} \in \tilde{I} \text{ s.t. } \pi(i_1) = \pi(i_2), \ |\tilde{u}_{i_1} - 1| \leq \frac{\epsilon}{2\sqrt{d_2}}, \ |\tilde{V}_{k,i_1} - w| \leq \frac{\epsilon}{2\sqrt{d_2}} \right), \tag{31}
\]
where \( \tilde{u}_{i_1}, \tilde{V}_{ki} \sim U[-1, 1] \) for \( i = 1, \cdots, d'R. \) By using Eq. (29), if \( d' \geq \frac{10d_2}{\epsilon^2 R^2} \log \left(\frac{2d_2}{\delta}\right) \), we have
\[
\text{Eq. (31)} \leq \left(1 - \frac{\epsilon^2}{16d_2}\right)^{d'R^2} \leq \frac{\delta}{2d_2},
\]
Therefore, by the union bound over \( k = 1, \cdots, d_2, \) we have \( i_1, i_2 \in \tilde{I} \) for each \( k \) such that \( \pi(i_1) = \pi(i_2), \ |\tilde{u}_{i_1} - 1| \leq \frac{\epsilon}{2\sqrt{d_2}}, \ |\tilde{V}_{k,i_1} - w_k| \leq \frac{\epsilon}{2\sqrt{d_2}}, \) with probability at least \( 1 - \delta, \) and thus
\[
|w_k \sigma(x) - V_{ki} \sigma(u_{ki}x)| \leq \frac{\epsilon}{2\sqrt{d_2}}, \text{ for } x \in \mathbb{R}, |x| \leq 1, \tag{32}
\]
if we substitute \( \tilde{u}_{i_1} \) for \( u_{i_1}, \) and \( \tilde{V}_{k,i_2} \) for \( V_{ki}. \) We note that the choice of \( \tilde{u}_{i_1} \) and \( \tilde{V}_{k,i_2} \) may not be unique, but Eq. (32) does not depend on these choice.
Applying Lemma A.5 to each independent summands in Eq. (36), with probability at least $w$, where each $M$.

Therefore, by taking $M$ appropriately, we have
\[ \|w x - g_M(x)\|_2 \leq \|w \sigma(x) - g_I(x)\|_2 + \|w \sigma(-x) - g_I(x)\|_2 \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon. \]

for all $x \in \mathbb{R}$ with $|x| \leq 1$.

Subsequently, we can generalize Lemma A.5 to multiple variables version:

**Lemma A.6** Fix $\epsilon, \delta \in (0, 1)$, $d_1, d_1, d_2 \in \mathbb{N}$, $W \in [-1, 1]^{d_2 \times d_0}$. Let $U \sim U[-1, 1]^{d_1 \times d_0}, V \sim U[-1, 1]^{d_2 \times d_1}$ be uniformly random weights of a 2-layered neural network $g(x) := V \sigma(Ux)$. Let $R \in \mathbb{N}$ and we assume that each elements of $U$ and $V$ can be re-sampled with replacement up to $R - 1$ times.

If $d_1 \geq 2d_0 \left( \frac{16d_2}{\epsilon^3} \log \left( \frac{2d_0d_2}{\delta} \right) \right)$ holds, then with probability at least $1 - \delta$,
\[ \|W x - g_{M,N}(x)\|_2 \leq \epsilon, \text{ for all } x \in \mathbb{R}^{d_0}, \|x\|_\infty \leq 1, \tag{33} \]

where $g_{M,N}(x) := (V \circ M)^T \sigma((U \circ N) \cdot x)$ for some $M \in \{0, 1\}^{d_2 \times d_1}, N \in \{0, 1\}^{d_1 \times d_0}$.

**Proof:** Let $d'_1 = d_1/d_0$, and we assume $d'_i \in \mathbb{N}$. We take $N$ as the following binary matrix:
\[ N = \begin{pmatrix} 1 & \cdots & 0 \\ 0 & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}, \text{ where } 1 = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^{d'_1 \times 1}. \]

By the decomposition $U \circ N = u_1 \oplus \cdots \oplus u_{d_0}$, where each $u_i$ is a $d'_i \times 1$-matrix, we have
\[ g_{M,N}(x) = (V \circ M)^T (\sigma(u_1x_1) \oplus \cdots \oplus \sigma(u_{d_0}x_{d_0})). \tag{34} \]

Here, we denote $M$ as follows:
\[ M = \begin{pmatrix} M_1 & \cdots & M_{d_0} \end{pmatrix}, \]

where each $M_i$ is a $d_2 \times d'_i$-matrix with binary coefficients. Then we have
\[ V \circ M = (V_1 \circ M_1) + \cdots + (V_{d_0} \circ M_{d_0}), \quad V_i \in \mathbb{R}^{d_2 \times d'_i}. \tag{35} \]

By combining Eq. (34) and Eq. (35), we have
\[ g_{M,N}(x) = \sum_{1 \leq i \leq d_0} (V_i \circ M_i) \sigma(u_i x_i). \tag{36} \]

Applying Lemma A.5 to each independent summands in Eq. (36), with probability at least $1 - \frac{\delta}{d_0}$, there exists $M_i$ for fixed $i \in \{1, \cdots, d_0\}$ such that
\[ \|w_i x_i - (V_i \circ M_i) \sigma(u_i x_i)\|_2 \leq \frac{\epsilon}{d_0}, \text{ for } x_i \in \mathbb{R}, |x_i| \leq 1, \tag{37} \]

where $w_i$ is the $i$-th column vector of $W$.

Using the union bound, we have $M_1, \cdots, M_{d_0}$ satisfying Eq. (37) simultaneously with probability at least $1 - \delta$. Therefore, by combining Eq. (36), Eq. (37) and the decomposition $Wx = \sum_j w_j x_j$, we obtain Eq. (33).

Finally, by using Lemma A.6, we prove Theorem A.1. The outline of the proof is same as prior works [19],[25].

**Proof of Theorem A.1:** By Lemma A.6, for each fixed $k \in \{1, \cdots, l\}$, we know that there exists binary matrices $M_{2k-1}, M_{2k}$ such that
\[ \|F_k x - (G_{2k} \circ M_{2k}) \sigma((G_{2k-1} \circ M_{2k-1}) x)\|_2 \leq \frac{\epsilon}{2l}, \tag{38} \]
for all $x \in \mathbb{R}^{d_0}$ with $\|x\|_\infty \leq 1$, with probability at least $1 - \frac{\delta}{7}$. Taking the union bound, we can get $M = (M_1, \cdots, M_{2l})$ satisfying Eq. (38) for all $k = 1, \cdots, l$ with probability at least $1 - \delta$.

For the above $M$ and any $x_0 \in \mathbb{R}^{d_0}$ with $\|x_0\|_\infty \leq 1$, we define sequences $\{x_k\}_{0 \leq k \leq l}$ and $\{\bar{x}_k\}_{0 \leq k \leq l}$ as

$$x_k := f_k(x_{k-1}),$$

$$\bar{x}_0 := x_0, \quad \bar{x}_k := g_{M,k}(\bar{x}_{k-1}),$$

where $f_k(x)$ and $g_{M,k}(x)$ are given by

$$f_k(x) := \begin{cases} \sigma(F_kx), & (1 \leq k \leq l-1) \\ F_lx, & (k = l) \end{cases}$$

$$g_{M,k}(x) := \begin{cases} \sigma((G_{2k} \circ M_{2k})\sigma((G_{2k-1} \circ M_{2k-1})x)), & (1 \leq k \leq l-1) \\ (G_{2l} \circ M_{2l})\sigma((G_{2l-1} \circ M_{2l-1})x), & (k = l) \end{cases}$$

By induction on $k \in \{0, \cdots, l\}$, we can show that

$$\|x_k - \bar{x}_k\|_2 \leq \frac{k\epsilon}{l}, \quad \|\bar{x}_k\|_\infty \leq 2. \tag{39}$$

For $k = 1$, this is trivial by definition. Consider the $k > 1$ case. First of all, we remark that the following inequality is obtained by Eq. (38) and the 1-Lipschitz property of ReLU function $\sigma$:

$$\|f_k(x) - g_{M,k}(x)\|_2 \leq \frac{\epsilon}{2l}, \quad \text{for } x \in \mathbb{R}^{d_0}, \|x\|_\infty \leq 1.$$

By scaling $x$, the above inequality can be rewritten as follows:

$$\|f_k(x) - g_{M,k}(x)\|_2 \leq \frac{\epsilon}{l}, \quad \text{for } x \in \mathbb{R}^{d_0}, \|x\|_\infty \leq 2.$$

Then, we have

$$\|x_k - \bar{x}_k\|_2 = \|f_k(x_{k-1}) - g_{M,k}(\bar{x}_{k-1})\|_2$$

$$\leq \|f_k(x_{k-1}) - f_k(\bar{x}_{k-1}) + f_k(\bar{x}_{k-1}) - g_{M,k}(\bar{x}_{k-1})\|_2$$

$$\leq \|f_k(x_{k-1}) - f_k(\bar{x}_{k-1})\|_2 + \|f_k(\bar{x}_{k-1}) - g_{M,k}(\bar{x}_{k-1})\|_2$$

$$\leq \|F_k\|_{\text{Frob}} \cdot \|x_{k-1} - \bar{x}_{k-1}\|_2 + \frac{\epsilon}{l}$$

$$\leq \frac{k\epsilon}{l} \quad \text{(by the induction hypothesis),}$$

and $\|\bar{x}_k\|_\infty \leq \|\bar{x}_k\|_2 \leq \|x_k\|_2 + \|x_k - \bar{x}_k\|_2 \leq 1 + \frac{k\epsilon}{l} \leq 2$.

In particular, for $k = l$, Eq. (39) is nothing but Eq. (21).

\section{Details for our experiments}

\subsection{Network architectures}

In our experiments on CIFAR-10 and ImageNet, we used the following network architectures: Conv6, ResNet18, ResNet34, ResNet50, and ResNet101. In Table 1 (for CIFAR-10) and Table 2 (for ImageNet), we describe their configurations with a width factor $\rho \in \mathbb{R}_{>0}$. When $\rho = 1.0$, the architectures are standard ones.

For each ResNet network, the bracket $\cdots$ represents the basic block for ResNet18 and ResNet34, and the bottleneck block for ResNet50 and ResNet101, following the original settings by He et al. [11]. We have a batch normalization layer right after each convolution operation as well. Note that, when we train and evaluate these networks with IteRand or edge-popup [26], we replace the batch normalization to the non-affine one, which fixes its all learnable multipliers to 1 and all learnable bias terms to 0 following the design by Ramanujan et al. [26].
### B.2 Hyperparameters

In our experiments, we trained several neural networks by three methods (SGD, edge-popup [26], and IteRand) on two datasets (CIFAR-10 and ImageNet). For each dataset, we adopted different hyperparameters as follows.

#### CIFAR-10 experiments.

- **SGD:** We used SGD with momentum for the optimization. It has the following hyperparameters: a total epoch number $E$, batch size $B$, learning rate $\eta$, weight decay $\lambda$, momentum coefficient $\mu$. For all network architectures, we used common values except for the learning rate and weight decay: $E = 100$, $B = 128$, and $\mu = 0.9$. For the learning rate and weight decay, we used $\eta = 0.01$, $\lambda = 1.0 \times 10^{-4}$ for Conv6, $\eta = 0.1$, $\lambda = 5.0 \times 10^{-4}$ for ResNet18 and ResNet34, following Ramanujan et al. [26]. Moreover, we decayed the learning rate by cosine annealing [16].

- **edge-popup:** With the same notation in Section 2, edge-popup has the same hyperparameters as SGD and an additional one, a sparsity rate $p$. We used the same values as SGD for each network except for the learning rate and the sparsity rate. For the learning rate, we used $\eta = 0.2$ for Conv6, and $\eta = 0.1$ for ResNet18 and ResNet34. We decayed the learning rate by cosine annealing, same as SGD. For the sparsity rate, we used $p = 0.5$ for Conv6, and $p = 0.6$ for ResNet18 and ResNet34.

- **IteRand:** With the notation in Section 3, IteRand has the same hyperparameters as edge-popup and the following additional ones: a randomization period $K_{\text{per}} \in \mathbb{N}_{\geq 1}$ and a sampling rate $r \in [0, 1]$ for partial randomization. We used the same values as edge-popup for the former hyperparameters, and $K_{\text{per}} = 300$, $r = 0.1$ for the latter ones.

#### ImageNet experiments.

- **SGD:** For all network architectures, we used the following hyperparameters (except for the learning rate): $E = 105$, $B = 128$, $\lambda = 1.0 \times 10^{-4}$, and $\mu = 0.9$. For the first 5 epochs, we gradually increased the learning rate as $\eta = 0.1 \times (i/5)$ for each $i$-th epoch ($i = 1, \cdots, 5$). For the next 95 epochs, we decayed the learning rate by cosine annealing starting from
\( \eta = 0.1 \). For the final 5 epochs, we set the learning rate \( \eta = 1.0 \times 10^{-5} \) to ensure the optimization converges.

- **edge-popup**: For all network architectures, we used the same hyperparameters as SGD and the sparsity rate \( p = 0.7 \).
- **IteRand**: For all network architectures, we used the same hyperparameters as edge-popup and \( K_{\text{per}} = 1000, r = 0.1 \).

### B.3 Experimental results in table forms

In Table 3, we give means ± one standard deviations which are plotted in Figure 2 in Section 5.

| Network | Method | \( D_{\text{param}} \) | \( \rho = 0.25 \) | \( \rho = 0.5 \) | \( \rho = 1.0 \) | \( \rho = 2.0 \) |
|---------|--------|----------------|----------------|----------------|----------------|----------------|
| Conv6   | IteRand SC | 75.16 ± 0.23 | 84.31 ± 0.13 | 88.80 ± 0.20 | 90.89 ± 0.17 |
|         | KU     | 73.90 ± 0.47 | 83.18 ± 0.38 | 88.20 ± 0.38 | 90.53 ± 0.28 |
|         | edge-popup SC | 70.35 ± 1.16 | 81.54 ± 0.11 | 87.60 ± 0.11 | 90.25 ± 0.06 |
|         | KU     | 67.02 ± 1.14 | 79.37 ± 0.23 | 86.14 ± 0.37 | 89.91 ± 0.23 |
|         | SGD    | KU     | 79.51 ± 0.88 | 84.46 ± 0.34 | 87.69 ± 0.36 | 89.63 ± 0.22 |
| ResNet18| IteRand SC | 86.09 ± 0.24 | 90.50 ± 0.36 | 92.61 ± 0.17 | 93.82 ± 0.15 |
|         | KU     | 84.71 ± 0.42 | 89.96 ± 0.06 | 92.47 ± 0.16 | 93.52 ± 0.16 |
|         | edge-popup SC | 84.23 ± 0.33 | 89.61 ± 0.06 | 92.29 ± 0.04 | 93.57 ± 0.13 |
|         | KU     | 81.13 ± 0.09 | 88.45 ± 0.53 | 91.79 ± 0.19 | 93.25 ± 0.06 |
|         | SGD    | KU     | 90.99 ± 0.29 | 93.10 ± 0.14 | 94.43 ± 0.05 | 95.02 ± 0.23 |
| ResNet34| IteRand SC | 88.26 ± 0.35 | 91.87 ± 0.17 | 93.54 ± 0.27 | 94.25 ± 0.15 |
|         | KU     | 87.36 ± 0.14 | 91.58 ± 0.16 | 93.27 ± 0.19 | 94.05 ± 0.10 |
|         | edge-popup SC | 87.37 ± 0.30 | 91.41 ± 0.26 | 93.27 ± 0.09 | 94.25 ± 0.13 |
|         | KU     | 85.31 ± 0.66 | 90.85 ± 0.11 | 93.00 ± 0.11 | 93.96 ± 0.10 |
|         | SGD    | KU     | 92.26 ± 0.12 | 94.20 ± 0.25 | 94.66 ± 0.18 | 95.33 ± 0.10 |

In Table 4, we give means ± one standard deviations which are plotted in Figure 3 in Section 5.

| Network | Method | \( D_{\text{param}} \) | Accuracy (Top-1) | Sparsity (%) | # of Params |
|---------|--------|----------------|-----------------|--------------|------------|
| ResNet18| SGD    | KU     | 69.89           | 0%           | 11.16M     |
| ResNet34| SGD    | KU     | 73.82           | 0%           | 21.26M     |
| ResNet34| IteRand SC | 65.86 | 70%              | 6.38M        |
|         | edge-popup SC | 62.14 | 70%              | 6.38M        |
| ResNet50| IteRand SC | 69.19 | 70%              | 7.04M        |
|         | edge-popup SC | 67.11 | 70%              | 7.04M        |
| ResNet101| IteRand SC | 72.99 | 70%              | 12.72M       |
|         | edge-popup SC | 71.85 | 70%              | 12.72M       |

### C Additional experiments

#### C.1 Ablation study on hyperparameters: \( K_{\text{per}} \) and \( r \)

IteRand has two hyperparameters: \( K_{\text{per}} \) (see line 5 in Algorithm 3) and \( r \) (see Eq. (3) in Section 3).
$K_{\text{per}}$ controls the frequency of randomizing operations during the optimization. Note that, in our experiments in Section 5, we fixed it to $K_{\text{per}} = 300$ on CIFAR-10, which is nearly 1 epoch (= 351 iterations) when the batch size is 128. As we discussed in Section 3, too small $K_{\text{per}}$ may degrade the performance because it may randomize even the important weights before their scores are well-optimized. In contrast, too large $K_{\text{per}}$ makes IteRand almost same as edge-popup, and thus the effect of the randomization disappears. Figure 4 shows this phenomenon with varying $K_{\text{per}}$ in $\{1, 30, 300, 3000, 30000\}$. In relation to our theoretical results (Theorem A.1), we note that the expected number (here we denote $R'$) of randomizing operations for each weight is in inverse proportion to $K_{\text{per}}$. The theoretical results imply that greater $R'$ leads to better approximation ability of IteRand. We observe that the results in Figure 4 are consistent with this implication, in the region where $K_{\text{per}}$ is not too small ($K_{\text{per}} \geq 300$).

![Figure 4](image)

Figure 4: We train and validate Conv6, ResNet18 and ResNet34 on CIFAR-10 with various $K_{\text{per}} \in \{1, 30, 300, 3000, 30000\}$. The x-axis is $K_{\text{per}}$ in a log scale, and the y-axis is validation accuracy.

Also, we investigate the relationship between $K_{\text{per}}$ and $r$. Figure 5 shows how test accuracy changes when both $K_{\text{per}}$ and $r$ vary. From this result, we find that the accuracies seem to depend on $r/K_{\text{per}}$. This may be because each pruned parameter in the neural network is randomized $Nr/K_{\text{per}}$ times in expectation during the optimization. On the other hand, when we use larger $r \in [0, 1]$, we have to explore $K_{\text{per}}$ in longer period (e.g. 3000 iterations when $r = 1.0$). Thus appropriately choosing $r$ leads to shrink the search space of $K_{\text{per}}$.

![Figure 5](image)

Figure 5: Test accuracies on CIFAR-10 with ResNet18. The x-axis is $K_{\text{per}} \in \{1, 30, 300, 3000\}$ and the y-axis is $r \in \{0.001, 0.01, 0.1, 1.0\}$. 

19
C.2 Computational overhead of iterative randomization

IteRand introduces additional computational cost to the base method, edge-popup, by iterative randomization. However, the additional computational cost is negligibly small in all our experiments. We measured the average overhead of a single randomizing operation, which is the only difference from edge-popup, as follows: 97.10 ms for ResNet18 (11.2M params), 200.55 ms for ResNet50 (23.5M params). Thus, the total additional cost should be about 10 seconds in the whole training (1.5 hours) for ResNet18 on CIFAR-10 and 200–300 seconds in the whole training (one week) for ResNet50 on ImageNet.

Also, we measured the total training times of our experiments for ResNet-18 on CIFAR-10 (Table 5). The additional computational cost of IteRand over edge-popup is tens of seconds, which is quite consistent with the above estimates.

| Networks | Methods | $p = 0.1$ | $p = 0.3$ | $p = 0.5$ | $p = 0.7$ | $p = 0.9$ | $p = 0.95$ | $p = 0.99$ |
|----------|---------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| Conv6    | IteRand (SC) | 87.17 ± 0.21 | 89.08 ± 0.14 | 89.19 ± 0.13 | 87.67 ± 0.07 | 76.09 ± 0.23 | 59.03 ± 1.55 | 12.04 ± 3.40 |
|          | edge-popup (SC) | 80.34 ± 0.27 | 86.55 ± 0.22 | 87.57 ± 0.03 | 86.40 ± 0.13 | 73.25 ± 0.79 | 55.23 ± 1.48 | 12.13 ± 3.46 |
| ResNet18 | IteRand (SC) | 91.79 ± 0.20 | 92.67 ± 0.11 | 92.66 ± 0.22 | 92.61 ± 0.15 | 91.82 ± 0.15 | 90.40 ± 0.42 | 76.31 ± 2.56 |
|          | edge-popup (SC) | 87.37 ± 0.18 | 91.43 ± 0.16 | 92.25 ± 0.18 | 92.32 ± 0.10 | 91.64 ± 0.18 | 90.28 ± 0.30 | 75.21 ± 2.71 |

C.3 Experiments with varying sparsity

Table 6 shows the comparison of IteRand and edge-popup when varying the sparsity parameter $p \in \{0.1, 0.3, 0.5, 0.7, 0.9, 0.95, 0.99\}$. We can see that IteRand is effective for almost all the sparsities $p$.

Moreover, we compared the pruning-only approach (IteRand and edge-popup) and the iterative magnitude pruning (IMP) approach [6] with various sparsity rates. We employed the OpenLTH framework [5], which contains the implementation of IMP, as a codebase for this experiment and implemented both edge-popup and IteRand in this framework. The results are shown in Table 7. Overall, the IMP outperforms the pruning-only methods. However, there is still room for improvement in the pruning-only approach such as introducing scheduled sparsities or an adaptive threshold, which is left to future work.

Table 7: Comparison of the pruning-only approach and magnitude-based one.

| Networks | Methods | $p = 0.5$ | $p = 0.7$ | $p = 0.9$ | $p = 0.95$ | $p = 0.99$ |
|----------|---------|-----------|-----------|-----------|-----------|-----------|
| VGG11    | IteRand (SC) | 88.46 ± 0.22 | 88.29 ± 0.42 | 87.05 ± 0.07 | 84.37 ± 0.59 | 64.93 ± 4.81 |
|          | edge-popup (SC) | 87.09 ± 0.31 | 87.34 ± 0.21 | 85.11 ± 0.55 | 81.06 ± 0.84 | 61.71 ± 0.65 |
|          | IMP with 3 retraining [6] | 91.47 ± 0.15 | 91.48 ± 0.16 | 90.89 ± 0.08 | 90.39 ± 0.27 | 88.076 ± 0.17 |
| ResNet20 | IteRand (SC) | 84.17 ± 0.78 | 82.31 ± 0.52 | 70.96 ± 0.55 | 55.60 ± 0.70 | 24.45 ± 0.67 |
|          | edge-popup (SC) | 76.57 ± 0.91 | 75.83 ± 2.75 | 49.25 ± 6.33 | 42.96 ± 3.91 | 20.11 ± 3.43 |
|          | IMP with 3 retraining [6] | 90.70 ± 0.37 | 89.79 ± 0.14 | 86.87 ± 0.26 | 84.28 ± 0.08 | 71.78 ± 1.66 |

C.4 Detailed empirical analysis on the parameter efficiency

We conducted experiments to see how much more network width edge-popup requires than IteRand to achieve the same accuracy (Table 8). Here we use ResNet18 with various width factors $p \in \mathbb{R}_{>0}$. We first computed the test accuracies of IteRand with $p = 0.5, 1.0$ as target values. Next we explored the width factors for which edge-popup achieves the same accuracy as the target values. Table 8 shows that edge-popup requires 1.3 times wider networks than IteRand in this specific setting.
Table 8: Test accuracies for ResNet18 with various width factors.

| ρ   | IteRand (KU)    | edge-popup (KU) |
|-----|----------------|-----------------|
| 0.5 | 89.96 ± 0.06   | 88.45 ± 0.53    |
| 0.65| -              | 89.99 ± 0.08    |
| 1.0 | 92.47 ± 0.16   | 91.79 ± 0.19    |
| 1.3 | -              | 92.54 ± 0.19    |

C.5 Experiments with large-scale networks

In addition to the experiments in Section 5, we conducted experiments to see the effectiveness of IteRand with large-scale networks: WideResNet-50-2 [33] and ResNet50 with the width factor ρ = 2.0. Table 9 shows that the iterative randomization is still effective for these networks to improve the performance of weight-pruning optimization.

Table 9: Experiments with large-scale networks.

| Networks   | IteRand (SC) | edge-popup (SC) | # of parameters |
|------------|--------------|-----------------|-----------------|
| WideResNet-50-2 | 73.57%       | 71.59%          | 68.8 M          |
| ResNet50 (ρ = 2.0) | 74.05%       | 72.96%          | 97.8 M          |

C.6 Experiments with a text classification task

Although our main theorem (Theorem 4.1) indicates that the effectiveness of IteRand does not depend on any specific tasks, we only presented the results on image classification datasets in the body of this paper. In Table 10, we present experimental results on a text classification dataset, IMDB [18], with recurrent neural networks (see Table 11 for the network architectures). For this experiment, we implemented both edge-popup and IteRand on the Jupyter notebook originally written by Trevett [30]. All models are trained for 15 epochs and the learning rate η we used is η = 1.0 for SGD and η = 2.5 for edge-popup and IteRand. Note that the learning rate η = 2.5 does not work well for SGD, thus we employed the different value from the one for edge-popup and IteRand. Also we set the hyperparameters for IteRand as p = 0.5, K_{per} = ⌊270/6⌋ (∼1/6 epochs) and r = 1.0.

Table 10: Test accuracies on the IMDB dataset over 5 runs.

| Networks | Methods   | IteRand (SC) | edge-popup (SC) | SGD       |
|----------|-----------|--------------|-----------------|-----------|
| LSTM     | 88.44 ± 0.28% | 88.16 ± 0.12% | 87.39 ± 0.37%   |
| BiLSTM   | 88.51 ± 0.24% | 88.34 ± 0.19% | 87.62 ± 0.22%   |

Table 11: The network architectures for IMDB.

| Layer       | Network                      | (Bi)LSTM                     |
|-------------|------------------------------|------------------------------|
| Embedding Layer | dim = 100                   |                              |
| LSTM Layer   | hidden_dim = 256, num_layers = 1, (bidirectional = True for BiLSTM) |
| Dropout Layer | p = 0.2                     |                              |
| Linear Layer | output_dim = 1              |                              |
| Output Layer | sigmoid                     |                              |