Recent Advances on Neural Network Pruning at Initialization

Huan Wang¹, Can Qin¹, Yue Bai¹, Yulun Zhang² and Yun Fu¹

¹Northeastern University
²ETH Zürich

{wang.huan, qin.ca, bai.yue}@northeastern.edu, yulun100@gmail.com, yunfu@ece.neu.edu

Abstract

Neural network pruning typically removes connections or neurons from a pretrained converged model; while a new pruning paradigm, pruning at initialization (PaI), attempts to prune a randomly initialized network. This paper offers the first survey concentrated on this emerging pruning fashion. We first introduce a generic formulation of neural network pruning, followed by the major classic pruning topics. Then, as the main body of this paper, a thorough and structured literature review of PaI methods is presented, consisting of two major tracks (sparse training and sparse selection). Finally, we summarize the surge of PaI compared to PaT and discuss the open problems. Apart from the dedicated literature review, this paper also offers a https://github.com/mingsun-tse/smile-pruningcode base for easy sanity-checking and benchmarking of different PaI methods.

1 Introduction

Network architecture design is a central topic when developing neural networks for various artificial intelligence tasks, especially in deep learning [LeCun et al., 2015; Schmidhuber, 2015]. The wisdom from learning theory suggests that the best generalization comes from a good trade-off between sample size and model complexity [Kearns et al., 1994; Vapnik, 2013], which implies to use neural networks of proper sizes. However, it is non-trivial to know in practice what size fits properly. Over-parameterized networks are thus preferred due to their abundant expressivity when tackling complex real-world problems. In deep learning era, this rule-of-thumb is even more pronounced not only because we are handling more complex problems, but also over-parameterized networks are observed easier to optimize (with proper regularization) [Simonyan and Zisserman, 2015; He et al., 2016] and possibly lead to better generalization [Soltanolkotabi et al., 2018; Allen-Zhu et al., 2019; Zou et al., 2020] than their compact counterparts.

However, over-parameterization brings cost in either testing or training phase, such as excessive model footprint, slow inference speed, extra model transportation and energy consumption. As a remedy, neural network pruning is proposessed to remove unnecessary connections or neurons in a neural network without seriously compromising the performance. A typical pruning pipeline comprises 3 steps [Reed, 1993]: (1) pretraining a (redundant) dense model ⇒ (2) pruning the dense model to a sparse one ⇒ (3) finetuning the sparse model to regain performance. Namely, pruning is considered as a post-processing solution to fix the side-effect brought by pretraining the dense model. This post-processing paradigm of pruning has been practiced for more than 30 years and well covered by many surveys, e.g., a relatively outdated survey [Reed, 1993], recent surveys of pruning alone [Gale et al., 2019; Blalock et al., 2020; Hoefler et al., 2021] or as a sub-topic under the umbrella of model compression and acceleration [Sze et al., 2017; Cheng et al., 2018a; 2018b; Deng et al., 2020].

However, a newly surging pruning paradigm, pruning at initialization (PaI), is absent from these surveys (we are aware that Pal is discussed in one very recent survey [Hoefer et al., 2021], yet merely with one subsection; our paper aims to offer a more thorough and concentrated coverage). Unlike pruning after training (PaT) (see Tab. 1), PaI prunes a randomly initialized dense network instead of a pretrained one. Pal aims to train (or purely select) a sparse network out of a randomly initialized dense network to achieve (close to) full accuracy (the accuracy reached by the dense network). This had been believed unpromising as plenty of prior works have observed training the sparse network from scratch underperforms PaT, until recently, lottery ticket hy-

<table>
<thead>
<tr>
<th>Pruning paradigm</th>
<th>Weight src</th>
<th>Mask src</th>
<th>Train sparse net?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pruning after training</td>
<td>Convg. net</td>
<td>Convg. net</td>
<td>✓ (mostly)</td>
</tr>
<tr>
<td>Sparse training (LTH)</td>
<td>Init. net</td>
<td>Convg. net</td>
<td>✓</td>
</tr>
<tr>
<td>Sparse training (SNIP)</td>
<td>Init. net</td>
<td>Init. net</td>
<td>✓</td>
</tr>
<tr>
<td>Sparse selection (Hidden)</td>
<td>Init. net</td>
<td>Init. net</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 1: Comparison between pruning at initialization (PaI) and pruning after training (PaT). As seen, weight source is the axis that differentiates Pal methods from their traditional counterparts: Pal methods inherit weights from a randomly initialized (Init.) network instead of a converged (Convg.) network. Pal methods can be further classified into two major categories: sparse training (picked sparse network will be trained) and sparse selection (picked sparse network will not be trained). In the parentheses is the representative method of each genre: LTH [Frankle and Carbin, 2019], SNIP [Lee et al., 2019], Hidden [Ramanujan et al., 2020].
2 Background of Neural Network Pruning

2.1 A Generic Formulation of Pruning

The stochastic gradient descent (SGD) learner of a neural network parameterized by \( w \) produces a model sequence

\[
\{ w^{(0)}, w^{(1)}, \ldots, w^{(k)}, \ldots, w^{(K)} \}
\]

where \( K \) denotes the total number of training iterations. In PaT, we only need the last model checkpoint \( (k = K) \) as the base model for pruning. However, the new pruning paradigm asks for models at different iterations, \( e.g., \) LTH needs the model at iteration 0 \( [\text{Frankle and Carbin, 2019}] \) (or an early iteration \[\text{Frankle et al., 2020a}\], we consider this also as 0 for simplicity). To accommodate these new cases, we allow a pruning algorithm to have access to the whole model sequence (which we can easily cache during the model training). Then, pruning can be defined as a function, which takes the model sequence as input (along with the training dataset \( \mathcal{D} \)), and outputs a pruned model \( w^* \).

We need two pieces of information to make a neural network exact, its topology and the associated parameter values. (1) In the case of a sparse network, the sparse topology is typically defined by a mask tensor (denoted by \( m \), with the same shape as \( w \)), which can be obtained via a function, \( m = f_1(w^{(k_1)}; \mathcal{D}) \), that is, \( f_1 \) can utilize the data available to help decide the masks (\( e.g., \) in regularization-based pruning methods, such as \[\text{Wang et al., 2021}\]). (2) Meanwhile, the weights in the pruned model can be adjusted from their original values to mitigate the incurred damage \[\text{Hassibi and Stork, 1993; Wang et al., 2019a; 2019c; 2021}\]. This step can be modeled as another function \( f_2(w^{(k_2)}; \mathcal{D}) \). Together, pruning can be formulated as

\[
\begin{align*}
w' &= f_1(w^{(k_1)}; \mathcal{D}) \odot f_2(w^{(k_2)}; \mathcal{D}), \\
w^* &= f_3(w'; \mathcal{D}),
\end{align*}
\]

where \( \odot \) represents the Hadamard (element-wise) product; \( f_3 \) models the finetuning process (which is omitted in the sparse selection PaT methods). Note that, the input models for \( f_1(*) \) and \( f_2(*) \) can be different iterations. By this definition, the pruning paradigms in Tab. 1 can be specified as follows,

- Pruning after training (PaT): \( k_1 = K, k_2 = K \);
- Sparse training: \( k_1 = K \) (LTH) or 0 (SNIP), \( k_2 = 0 \), and \( f_2 = \text{I} \) (identity function), \( f_3 \neq \text{I} \);
- Sparse selection: \( k_1 = k_2 = 0 \), and \( f_2 = f_3 = \text{I} \).

2.2 Classic Topics in Pruning

There are mainly four critical questions we need to ask when pruning a specific model: \textit{what} to prune, \textit{how many} (connections or neurons) to prune, \textit{which} to prune exactly, and \textit{how} to schedule the pruning process, corresponding to four classic topics in pruning as follows. We only briefly explain them here as they are not our main focus. For a more comprehensive coverage, we refer the readers to \[\text{Hoefler et al., 2021}\].

(1) \textbf{Sparsity structure}. Weights can be pruned in patterns. The shape of the pattern, named sparsity structure, decides the basic pruning element of a pruning algorithm. The smallest structure, of course, is a single weight element, \( i.e., \) no structure at all. This kind of pruning is called unstructured pruning. The pruning pattern larger than a single weight element can be called structured pruning in general. Although there are different levels of granularity when defining structured pruning (see \[\text{Mao et al., 2017}\]), structured pruning typically means filter or channel pruning in the literature.

Practically, unstructured pruning is mainly favored for model size compression, structured pruning more favored for acceleration due to the hardware-friendly sparsity structure. Acceleration is more imperative than compression for modern deep networks, so most PaT works focus on structured pruning currently. We will see that, differently, most PaI methods focus on unstructured pruning instead.

(2) \textbf{Pruning ratio}. Pruning ratios indicate how many weights to remove. In general, there are two ways to determine pruning ratios. (i) The first is to pre-define them. Namely, we know exactly how many parameters will be pruned before the algorithm actually runs. This scheme can be further specified into two sub-schemes. One is to set a global pruning ratio \( (i.e., \) how many weights will be pruned for the whole network); the other is to set layer-wise pruning ratios. (ii) The second is to decide the pruning ratio by other means. This way mostly appears in the regularization-based pruning methods, which remove weights by driving them towards zero via penalty terms. A larger regularization factor typically leads to more sparsity, \( i.e., \) a larger pruning ratio. However, how to set a proper factor to achieve the desired sparsity usually demands heavy tuning. Several methods have been proposed to improve this \[\text{Wang et al., 2019c; 2021}\] – the pruning ratios are usually pre-specified. Recent years also have seen some works that automatically search the optimal layer-wise pruning ratio \[\text{He et al., 2018}\]. No consensus has been reached on which is better.

(3) \textbf{Pruning criterion}. Pruning criterion decides which weights to remove given a sparsity budget. It is considered one of the most critical problems in network pruning, thus has received the most research attention so far. The most simple criterion is weight magnitude (or equivalently, \( L_1 \)-norm for a tensor) \[\text{Han et al., 2015; Li et al., 2017}\]. Because of its simplicity, it is the most prevailing criterion in PaT now. We will see it also being widely used in PaI.

Albeit much exploration in this topic, there are no criteria that prove to be significantly better than the others (actually, the simple magnitude pruning has been argued as (one of) the SOTA \[\text{Gale et al., 2019}\]). Since this topic is already well discussed in prior surveys in PaT, we will not cover it in
depth here. One point worth mention is that, the core idea underpinning most pruning criteria now is to select the weights whose absence induces the least loss change. This idea and its variant have been followed for a long time in PaT [LeCun et al., 1990; Hassibi and Stork, 1993; Wang et al., 2019a; Molchanov et al., 2017; 2019], which continues to PaI.

(4) Pruning schedule. After the three aspects determined above, we finally need to specify the schedule of pruning. There are three typical choices [Wang et al., 2019b]. (i) One-shot: network sparsity (defined by the ratio of zeroed weights in a network) goes from 0 to a target number in a single step, then finetune. (ii) Progressive: network sparsity goes from 0 to a target number gradually, typically along with network training; then finetune. (iii) Iterative: network sparsity goes from 0 to an intermediate target number, then finetune; then repeat the process until the target sparsity is achieved. Note, both (ii) and (iii) are characterized by pruning interleaved with network training. Therefore, there is no fundamental boundary between the two. Some works thus use the two terms interchangeably. One consensus is that, progressive and iterative pruning outperform the one-shot counterpart when pruning the same number of weights because they allow more time for the network to adapt. In PaI, LTH employs iterative pruning, which increases the training cost significantly. PaI works featured by pre-selected masks (e.g., Wang et al., 2020) are thereby motivated to resolve this problem.

3 Pruning at Initialization (PaI)

3.1 Overview: History Sketch

Debut: LTH and SNIP. The major motivation of PaI vs. PaT is to achieve cheaper and simpler pruning – pruning a network at initialization prior to training “eliminates the need for both pretraining and the complex pruning schedule” (as quoted from SNIP [Lee et al., 2019]). Specially, two works lead this surge. LTH [Frankle and Carbin, 2019] and SNIP. Importantly, both make a similar claim: Non-trivially sparse networks can be trained to full accuracy in isolation, which was believed barely feasible before. Differently, LTH selects masks from a pretrained model while SNIP selects masks from the initialized model, i.e., post-selected vs. pre-selected masks, as shown in Fig. 1.

Follow-ups of LTH and SNIP. (1) There are two major lines of LTH follow-ups. Acknowledging the efficacy of LTH, one line is to expand its universe, e.g., scaling LTH to larger datasets and models (e.g., ResNet50 on ImageNet), validating it on non-vision domains (e.g., natural language processing), and proposing theoretical foundations. The other line takes a decent grain of salt about the efficacy of LTH, keeps sanity-checking. (2) At the same time, for the direction led by SNIP, more methods come out focusing on better pruning criteria.

Dynamic masks. The masks in both LTH and SNIP are static, namely, they are fixed during training. Some researchers (such as [Evci et al., 2020]) conjecture that dynamic and adaptive masks during training may be better, thus introduce another group of methods featured by dynamic masks. Together, the static-and dynamic-mask methods complete the realm of sparse training.

Sparse selection. When some researchers attempt to understand LTH through empirical studies, they discover an interesting phenomenon. [Zhou et al., 2019] surprisingly find that the random network selected by winning ticket in LTH actually has non-trivial accuracy even without any training. This discovery brings us strong LTH: Sparse subnet picked from a dense network can achieve full accuracy even without further training, and opens a new direction named sparse selection.

The current PaI universe is primarily made up of the above two categories, sparse training and sparse selection. A method tree overview of PaI is presented in Fig. 1. Next, we elaborate the major approaches in each genre in length.

3.2 Sparse Training

Static masks: post-selected. Sparse training methods of this group are pioneered by LTH [Frankle and Carbin, 2019]. The pruning pipeline in LTH has three steps: First, a randomly initialized network is trained to convergence; Second, employ magnitude pruning to obtain the masks, which defines the topology of the subnet; Third, apply the masks to the initial network to obtain a subnet and train the subnet to convergence. This process can be repeated iteratively (i.e., iterative magnitude pruning, or IMP). The authors surprisingly find the subnet can achieve comparable (or even better occasionally) accuracy to the dense network. The lottery ticket hypothesis is thus proposed: “dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that—when trained in isolation—reach test accuracy comparable to the.

Figure 1: Overview of pruning at initialization (PaI) approaches, classified into two general groups: sparse training and sparse selection. For readability, references are omitted in this figure but the paper abbreviations (right beside the abbreviation is the year when the paper appeared). Please see Sec. 3 for detailed introductions of them. Due to limited length, this paper only outlines the primary methods, see full collection at https://github.com/mingsun-tse/awesome-pruning-at-initialization.
original network in a similar number of iterations.”

Follow-up works of LTH mainly fall into two groups. One group identifies the legitimacy of LTH and attempts to seek a broader application or understanding of it. The other group proposes several sanity-check ablations of LTH, taking a decent grain of salt with its validity. These two groups are thus summarized as extensions and sanity-checks in Fig. 1.

(1) Extensions. The original LTH is only validated on small datasets (MNIST and CIFAR-10). [Frankle et al., 2020a] (LTH+) later generalize LTH to ResNet50 on ImageNet by applying the masks not to the initial weights but to the weights after a few epochs of training. [Morcos et al., 2019] (OneTicket) discover that winning tickets can generalize across a variety of datasets and optimizers within the natural images domain. Besides, winning tickets generated with larger datasets consistently transfer better than those generated with smaller datasets, suggesting that winning tickets contain inductive biases generic to neural networks. [Yu et al., 2020] (RL&NLP) find the lottery ticket phenomenon also presents on RL and NLP tasks and use it to train compressed Transformers to high performance. [Chen et al., 2020a] validates lottery ticket phenomenon on BERT subnetworks at 40% to 90% sparsity. [Chen et al., 2020b] (Lifelong) introduce two pruning methods, bottom-up and top-down, to find winning tickets in the lifelong training scenario. [Chen et al., 2021a] (Pretraining) locate matching subnetworks at 59.04% to 96.48% sparsity that transfer universally to multiple downstream tasks with no performance degradation for supervised and self-supervised pre-training. [Chen et al., 2021b] (GNN) present a unified graph neural network (GNN) sparsification framework that simultaneously prunes the graph adjacency matrix and the model weights and generalizes LTH to GNN for the first time. The iterative train-prune-retrain cycles on full training set in LTH can be very expensive. To resolve this, [Zhang et al., 2021b] introduce Pruning Aware Critical (PrAC) set, which is a subset of training data. PrAC set takes only 35.32% to 78.19% of the full set (CIFAR10, CIFAR100, Tiny ImageNet) with similar effect, saving up to 60% to 90% training iterations. Moreover, they find the PrAC set can generalize across different network architectures. Similarly, E-LTH [Chen et al., 2021c] also attempt to find winning tickets generic to different networks without the expensive IMP process. They achieve so by transforming winning tickets found in one network to another deeper or shallower one from the same family. [You et al., 2020] find that winning tickets can be identified at the very early training stage (thus they term the tickets early-bird, or EB, tickets) via low-cost training schemes (e.g., early stopping and low-precision training) at large learning rates, achieving up to 4.7× energy saving while maintaining the performance. [Diffenderfer and Kailkhura, 2021] (Multi-Prize) find winning tickets on binary neural networks for the first time.

Beside the technical extensions of LTH above, researchers also try to understand LTH more theoretically. [Frankle et al., 2020b] (EarlyPhase) analyze the early phase of deep network training. [Evci et al., 2020b] (GradFlow) present a gradient flow perspective to explain why LTH happens. [Zhang et al., 2021a] (Manifold) verify the validity of LTH by leveraging dynamical systems theory and inertial manifold theory. More works on the LTH theories study a stronger version of LTH, i.e., sparse selection. We thus defer them to Sec. 3.3.

In short, extensions of LTH mainly focus on LTH+X (i.e., generalizing LTH to other tasks or learning settings, etc.), identifying cheaper tickets, understanding LTH better.

(2) Sanity-checks. The validation of LTH seriously hinges on experimental setting, which actually has been controversial since the very beginning of LTH. In the same venue where LTH is published, another pruning paper [Liu et al., 2019] actually draws a pretty different conclusion from LTH. They argue that a subnetwork with random initialization (vs. the winning tickets picked by IMP in LTH) can be trained from scratch to match pruning a pretrained model. Besides, [Gale et al., 2019] also report that they cannot reproduce LTH. The reasons, noted by [Liu et al., 2019], may reside in the experimental settings – [Liu et al., 2019] focus on filter pruning and use momentum SGD with a large initial learning rate (LR) (0.1), while [Frankle and Carbin, 2019] tackle unstructured pruning and “mostly uses Adam [Kingma and Ba, 2015] with much lower learning rates”. This debate continues on. [Su et al., 2020] (RandomTickets) notice that randomly changing the preserved weights in each layer (the resulted initializations are thus named random tickets), while keeping the layer-wise pruning ratio, does not affect the final performance, thus introduce the random tickets. [Frankle et al., 2021] (MissMark) later report similar observation. This property “suggests broader challenges with the underlying pruning heuristics, the desire to prune at initialization, or both” [Frankle et al., 2021]. Meanwhile, [Liu et al., 2021] (Correlation) find that there is a strong correlation between initialized weights and the final weights in LTH, when the LR is not sufficiently large, thereby they argue that “the existence of winning property is correlated with an insufficient DNN pretraining, and is unlikely to occur for a well-trained DNN”. This resonates with the conjecture by [Liu et al., 2019] above that learning rate is a critical factor making their results seemingly contradicted with LTH. [Ma et al., 2021] (ReallyWin?) follow up this direction and present more concrete evidence to clarify whether the winning ticket exists across the major DNN architectures and/or applications.

Static masks. Pre-selected. SNIP [Lee et al., 2019] pioneers the direction featured by pre-selected masks. SNIP proposes a pruning criterion named connectivity sensitivity to select weights based on a straightforward idea of loss preservation, i.e., removing the weights whose absence leads to the least loss change. After initialization, each weight can be assigned a score with the above pruning criterion, then remove the last p fraction (p is the desired pruning ratio) parameters for subsequent training. Later, [Wang et al., 2020] argue that it is the training dynamics rather than the loss value itself that matters more at the beginning of training. Therefore they propose gradient signal preservation (GraSP) in contrast to the previous loss preservation, based on Hessian. Concurrently to GraSP [Wang et al., 2020], [Lee et al., 2020] seek to explain the feasibility of SNIP through the lens of signal propagation. They empirically find pruning damages the dynamical isometry [Saxe et al., 2014] of neural networks and propose an data-independent initialization, approximated isometry (AI),
which is an extension of exact isometry to sparse networks. Later, SynFlow [Tanaka et al., 2020] proposes a new data-independent criterion, which a variant of magnitude pruning, yet taking into account the interaction of different layers of a network. Meanwhile, SupSup [Wortsman et al., 2020] proposes a training method to select different masks (supermasks) for thousands of tasks in continual learning, from a single fixed random base network. Another very recent static-mask work is DLTH [Bai et al., 2022]. In DLTH, initial weights are given, the problem is to find the matching masks, while in DLTH, the masks are given, the problem is to find the matching initial weights. Specifically, they employ a growing $L_2$ regularization technique to transform the original random weights to the desired condition.

To summarize, similar to PaT, the primary research attention in this line of PaT works also lies in pruning criterion. Actually, many PaT works propose pretty similar criterion formulas to those in PaT (see Tab. 2).

**Dynamic masks.** Another sparse training track allows the masks to be changed during training. DeepR [Bellec et al., 2018] employs dynamic sparse parameterization with stochastic parameter updates for training. SET [Mocanu et al., 2018] adopts magnitude pruning and random growth for dynamic network topology optimization. DSR [Mostafa and Wang, 2019] adaptively allocates layer-wise sparsity ratio without need of manual setting and shows training-time structural exploration is necessary for best generalization. SNFS [Dettmers and Zettlemoyer, 2019] proposes to employ the momentum of gradients to bring back pruned weights. RigL [Evi et al., 2020a] also uses magnitudes for pruning, yet they employ the absolute gradients for weight growing.

The essential idea of dynamic masks is to enable zeroed weights to rejoin the training again. Of note, this idea is not newly brought up in PaT. It has been explored too in pruning after training. In terms of the criteria, notably, because the masks need to be re-evaluated frequently during training in dynamic-mask methods, the pruning and growing criteria cannot be costly. Therefore, all these criteria are based on magnitudes or gradients (see Tab. 3), which are readily available during SGD training.

<table>
<thead>
<tr>
<th>Method</th>
<th>Pruning criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skeletonization (1989)</td>
<td>$-\nabla_w L \odot w$</td>
</tr>
<tr>
<td>OBD (1990)</td>
<td>$\text{diag}(H)w \odot w$</td>
</tr>
<tr>
<td>Taylor-FO (2019)</td>
<td>$(\nabla_w L \odot w)^2$</td>
</tr>
<tr>
<td>SNIP (2019)</td>
<td>$\nabla_w L \odot w$</td>
</tr>
<tr>
<td>GraSP (2020)</td>
<td>$-H \nabla_w L \odot w$</td>
</tr>
<tr>
<td>SynFlow (2020)</td>
<td>$\frac{\partial}{\partial w} w, R = 1 \odot \left( \prod_{l=1}^{L}</td>
</tr>
</tbody>
</table>

Table 2: Summary of pruning criteria in static-mask sparse training methods. Above the dash line are the PaT methods. $L$ denotes the loss function; $H$ represents Hessian; $1$ is the all ones vector; $l$ denotes the $l$-th layer of all $L$ layers. Skeletonization [Mozer and Smolensky, 1989]. Taylor-FO [Molchanov et al., 2019].

which are optimized the weights, **optimize the network topology.** Namely, when a network is randomly initialized, all the values of each connection are fixed. The goal is to find a subnet from the dense network without further training the subnet. This line of works is firstly pioneered by [Zhou et al., 2019] (Deconstruct) where they try to understand the mysteries of LTH. They find the subnet picked by LTH can achieve non-trivial accuracies already (without training). This implies that, although the original full network is randomly initialized, the chosen subnet is not really random. The subnet picking process itself serves as a kind of training. Therefore, [Zhou et al., 2019] propose the notion of supermasks (or masking as training) along with a proposed algorithm to optimize the masks in order to find better supermasks.

In the method in [Zhou et al., 2019] is only evaluated on small-scale datasets (MNIST and CIFAR). Later, [Ramanujan et al., 2020] (Hidden) introduce a trainable score for each weight and update the score to minimize the loss function. The trainable scores are eventually used to decide the selected subnetwork topology. The method achieves strong performance for the first time. E.g., they manage to pick a subnet out of a random Wide ResNet50. The subnet is smaller than ResNet34 while delivers better top-1 accuracy than the trained ResNet34 on ImageNet. This discovery is summarized as a stronger version of LTH, strong LTH: “within a sufficiently over-parameterized neural network with random weights (e.g. at initialization), there exists a subnet-work that achieves competitive accuracy” [Ramanujan et al., 2020].

The above are the breakthroughs in terms of technical algorithms. In terms of theoretical progress, [Malach et al., 2020] (Proving) propose the theoretical basis of the strong LTH, suggesting that “pruning a randomly initialized network is as strong as optimizing the value of the weights”. Less favorably, they make assumptions about the norms of the inputs and of the weights. Later, [Orseau et al., 2020; Pensia et al., 2020] (Logarithmic) remove the aforementioned limiting assumptions while providing significantly tighter bounds: “the over-parameterized network only needs a logarithmic factor (in all variables but depth) number of neurons per weight of the target subnetwork” [Diffenderfer and Kharkura, 2021] have shown that scaled binary networks can be pruned to approximate any target function, where the optimization is polynomial in the width, depth, and approximation error, akin to [Malach et al., 2020]. [Sreenivasan et al., 2022] further demonstrate that a log-arithmically over-parameterized binary network is enough to approximate any target function, similar to the advance of [Orseau et al., 2020; Pensia et al., 2020] over [Malach et al., 2020].

<table>
<thead>
<tr>
<th>Method</th>
<th>Pruning criterion</th>
<th>Growing criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS (2016)</td>
<td>$w$</td>
<td>$w$ (pruned weights also updated)</td>
</tr>
<tr>
<td>DeepR (2018)</td>
<td>stochastic</td>
<td>random</td>
</tr>
<tr>
<td>SET (2018)</td>
<td>$w$</td>
<td>random</td>
</tr>
<tr>
<td>DSR (2019)</td>
<td>$w$</td>
<td>random</td>
</tr>
<tr>
<td>SNFS (2020)</td>
<td>$w$</td>
<td>momentum of $\nabla_w L$</td>
</tr>
<tr>
<td>RigL (2020)</td>
<td>$w$</td>
<td>$</td>
</tr>
</tbody>
</table>

Table 3: Summary of pruning and growing criteria in dynamic-mask sparse training methods. Above the dash line are the PaT methods. DNS [Guo et al., 2016]; see other references in Sec. 3.3.
4 Summary and Open Problems

4.1 Summary of Pruning at Initialization

Easily seen, the biggest paradigm shift from PaT to PaI is to use a randomly initialized model to inherit weights for the sparse network, just as the PaI term (pruning at initialization) suggested. The primary goal is to achieve efficient training, in contrast to efficient inference aimed by PaT. This section answers what are the similarities and major differences between PaI vs. the traditional fashion in a big-picture level.

Sparse training: Much overlap between PaI and PaT. Not surprisingly, there is considerable overlap between PaI and PaT. (1) For LTH-like works, the adopted pruning scheme is mainly (iterative) magnitude pruning, which has been developed for more than thirty years and widely considered as the most simple pruning method. (2) For pre-selected-mask methods, the proposed pruning criteria for PaI are reminiscent of those proposed in PaT (Tab. 2). Therefore, we may continue seeing the wisdom in PaT being transferred to PaI.

In terms of the four primary aspects in pruning (Sec. 2.2), PaI does not bring much change to the pruning ratio, criteria, and schedule. However, for sparsity structure, it does have a significant impact. First, structured pruning (or filter pruning) itself is not much interesting in the context of PaI because given the layer-wise sparsity ratios filter pruning at initialization reduces to training from scratch a dense network (with a smaller width). Then the major technical problem is how to initialize a slimmer network properly, which has been well-explored. This is probably why most PaI works focus on unstructured pruning rather than structured pruning. Second, the well-known LTH is proposed for unstructured pruning. As far as we know, to date, no work has validated the hypothesis with filter pruning. The reason behind is still elusive.

Sparse selection: Significant advance of PaI vs. PaT. The common belief behind PaT is that the remaining parameters possess knowledge learned by the original redundant model. Inheriting the knowledge is better than starting over, which has been also empirically justified by many works and thus become a common practice for more than thirty years. In stark contrast, PaI inherits the parameter values right from the beginning. It is tempting to ask: At such an early phase of neural network training, do the weights really possess enough knowledge (to provide base models for subsequent pruning)? Or more fundamentally, is neural network training essentially to learn the knowledge from nothing or to reveal the knowledge the model already has? PaT implies the former, while PaI (especially sparse selection [Ramanujan et al., 2020; Malach et al., 2020]) suggests the latter. This, we believe, is (arguably) the most profound advance PaI has brought about. The understanding of this question will not only bestow us practical benefits, but also, more importantly, a deeper theoretical understanding of deep neural networks.

4.2 Open Problems

Same problems as pruning after training. Given the overlap between PaI and PaT, nearly all the open problems in PaT will also apply to PaI, too, centering on the four classic topics in pruning: how to find better sparsity structure being hardware friendly and meanwhile optimization friendly, how to find better pruning criteria, how to allocate the sparsity more properly cross different layers, and how to schedule the sparsification process more wisely, in the hopes of higher performance given the same sparsity budget. We do not reiterate these topics here considering they have been well treated in existing pruning surveys (e.g., [Hoefler et al., 2021]).

Under-performance of PaI. The idea of PaI is intriguing, however, in terms of practical performance, PaI methods still underperform PaT methods by an obvious margin. E.g., according to the experiments in [Wang et al., 2020], with VGG19 and ResNet32 networks on CIFAR-10/100, both SNIP and GraSP are consistently outperformed across different sparsity levels by two traditional pruning methods (OBD [LeCun et al., 1990] and MLPPrune [Zeng and Urtasun, 2019]), which are not even close to the state-of-the-art. [Frankle et al., 2021] also report similar observation. In this sense, there is still a long road ahead before we can really “save resources at training time” [Wang et al., 2020].

Under-development of sparse libraries. Despite the promising potential of sparse training, the practical acceleration benefit to date has not been satisfactory. E.g., SNFS [Dettmers and Zettlemoyer, 2019] claims “up to 5.61x faster training”, yet in practice, due to under-development of sparse matrix multiplication libraries, this benefit cannot be materialized at present. To our best knowledge, few works have reported wall-time speedup in sparse training. Development of sparse training libraries thus can be a worthy future cause. In this regard, notably, dynamic-mask methods pose even more severe issues than static-mask methods as the constantly changing masks will make it harder for the hardware to gain acceleration. If the training is not really getting faster, this may fundamentally undermine the motivation of sparse training, which actually is the main force of PaI now.

To sum, the major challenges facing PaI is to deliver the practical training speedup with no (serious) performance compromised, as it promises. This hinges on a more profound comprehension of the performance gap between sparse training and PaT, as well as the advance of sparse matrix libraries.

Code base. As discussed, PaI works (especially LTH-like) severely hinge on empirical studies. However, it is non-trivial in deep learning to tune hyper-parameters. In this paper, we thereby also offer a code base1 with systematic support of popular networks, datasets, and logging tools, in the hopes of aiding researchers and practitioner focus more on the insights and core methods instead of the tedious engineering details.

5 Conclusion

This paper presents the first survey concentrated on neural network pruning at initialization (PaI). The road map of PaI vs. its pruning-after-training counterpart is studied with a thorough and structured literature review. We close by outlining the open problems and offer a code base towards easy sanity-checking and benchmarking of different PaI methods.

Acknowledgements

We thank Jonathan Frankle, Alex Renda, and Michael Carbin from MIT for their very helpful suggestions to our work.

1https://github.com/mingsun-tse/smile-pruning
References


[Han et al., 2015] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural network. In NeurIPS, 2015.


[Liu et al., 2021] Ning Liu, Geng Yuan, Zhengping Che, Xuan Shen, Xiaolong Ma, Qing Jin, Jian Ren, Jian Tang, Sijia Liu, and Yanzhi Wang. Lottery ticket implies accuracy degradation, is it a desirable phenomenon? In ICMRL, 2021.

[Malach et al., 2020] Eran Malach, Gilad Yehudai, Shai Shalev-Shwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In ICML, 2020.


