The Elastic Lottery Ticket Hypothesis

Anonymous Author(s)
Affiliation
Address
email

Abstract

Lottery Ticket Hypothesis (LTH) raises keen attention to identifying sparse trainable subnetworks, or winning tickets, of training, which can be trained in isolation to achieve similar or even better performance compared to the full models. Despite many efforts being made, the most effective method to identify such winning tickets is still *Iterative Magnitude-based Pruning (IMP)*, which is computationally expensive and has to be run thoroughly for every different network. A natural question that comes in is: *can we "transform" the winning ticket found in one network to another with a different architecture, yielding a winning ticket for the latter at the beginning, without re-doing the expensive IMP?* Answering this question is not only practically relevant for efficient "once-for-all" winning ticket finding, but also theoretically appealing for uncovering inherently scalable sparse patterns in networks. We conduct extensive experiments on CIFAR-10 and ImageNet, and propose a variety of strategies to tweak the winning tickets found from different networks of the same model family (e.g., ResNets). Based on these results, we articulate the *Elastic Lottery Ticket Hypothesis* (**E-LTH**): by mindfully replicating (or dropping) and re-ordering layers for one network, its corresponding winning ticket could be stretched (or squeezed) into a subnetwork for another deeper (or shallower) network from the same family, whose performance is nearly the same competitive as the latter's winning ticket directly found by IMP. We have also thoroughly compared E-LTH with pruning-at-initialization and dynamic sparse training methods, and discuss the generalizability of E-LTH to different model families, layer types, or across datasets. All codes will be released.

1 Introduction

Lottery Ticket Hypothesis (LTH) [11] suggests the existence of sparse subnetworks in over-parameterized neural networks at their random initialization, early training stage, or pre-trained initialization [30, 39, 5, 3, 2]. Such subnetworks, usually called *winning tickets*, contain much fewer non-zero parameters compared with the original dense networks, but can achieve similar or even better performance when trained in isolation. This discovery undermines the necessity of over-parameterized initialization for successful training and good generalization ability of neural networks [40, 26]. This new scheme implies the possibility to train a highly compact subnetwork instead of a prohibitively one without compromising performance, potentially drastically reducing the computational cost for network training.

However, the current success of LTH essentially depends on *Iterative Magnitude-based Pruning (IMP)*, which requires repeated cycles of training networks from scratch, pruning and resetting the remaining parameters. IMP makes it extremely expensive and sometimes unstable to find winning tickets at scale, with large models and large datasets [13]. To alleviate this drawback, many efforts have been devoted to finding more efficient alternatives to IMP that can identify sparse trainable subnetworks at random initialization, with little-to-no training [23, 35, 33, 14]. Unfortunately, they all see some performance gap when compared to the winning tickets found by IMP, with often different structural patterns [14]. Hence, IMP remains to be the *de facto* scheme for lottery ticket finding.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

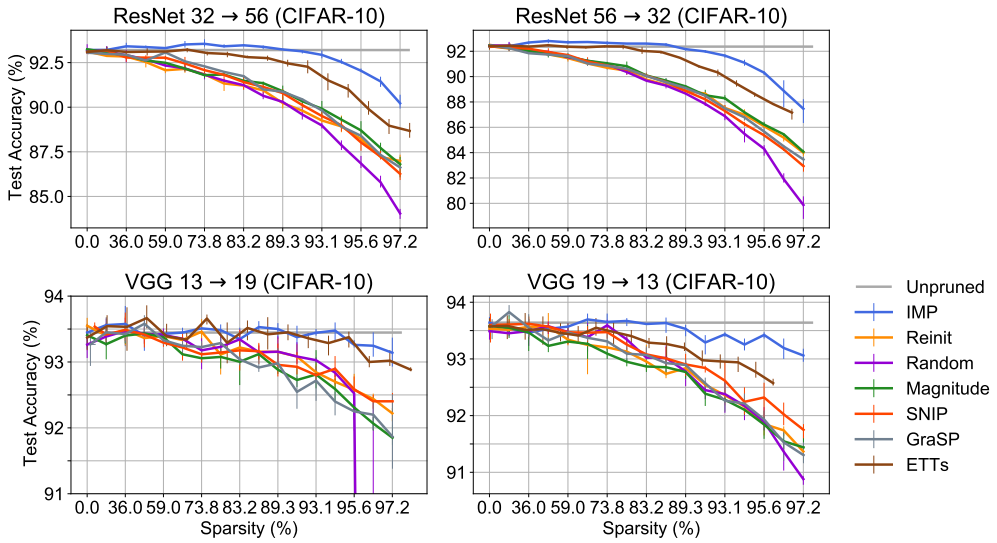

Figure 1: Validating Elastic Lottery Ticket Hypothesis (E-LTH) between two pairs of networks (ResNet-32 and ResNet-56, VGG-13 and VGG-19) trained on the CIFAR-10 dataset. We transform the winning tickets in the source models at different levels of sparsity using the proposed *Elastic Ticket Transformations (ETTs)*, and compare with other pruning methods (please refer to Section 4 for those methods' details), including the state-of-the-art pruning-at-initialization methods (e.g., SNIP and GraSP). Substantial accuracy gaps can be observed between ETTs and other pruning methods, corroborating that ETTs provide high-quality tickets on the target models that are comparable with the winning tickets found by the expensive IMP. All results are based on three independent runs.

Prior work [28] found that a winning ticket of one dense network can generalize across datasets and optimizers, beyond the original training setting where it was identified. Their work provided a new perspective of reducing IMP cost – to only find one generic, dataset-independent winning ticket for each backbone model, then transferring and re-training it on various datasets and downstream tasks. Compared to this relevant prior work which studies the transferablity of a winning ticket in the same network architecture, in this paper, we ask **an even bolder question**:

Can we transfer the winning ticket found for one network to other different network architectures?

This question not only has strong practical relevance but also arises theoretical curiosity. On the practicality side, if its answer is affirmative, then we will perform only expensive IMP for one network and then automatically derive winning tickets for others. It would point to a tantalizing possibility of *once-for-all* lottery ticket finding, and the extraordinary cost of IMP on a "source architecture" is amortized by transferring to a range of "target architectures". A promising application is to first find winning tickets by IMP on a small source architecture and then transfer it to a much bigger target architecture, leading to drastic savings compared to directly performing IMP on the latter. Another use case is to compress a larger winning ticket directly to smaller ones, in order to fit in the resource budgets on different platforms. On the methodology side, this new form of transferability would undoubtedly shed new lights on the possible mechanisms underlying the LTH phenomena by providing another perspective for understanding lottery tickets through their transferablity, and identify shared and transferable patterns that make sparse networks trainable [15, 33, 12]. Moreover, many deep networks have regular building blocks and repetitive structures, lending to various model-based interpretations such as dynamical systems or unrolled estimation [16, 37, 17, 1]. Our explored empirical methods seem to remind of those explanations too.

1.1 Our Contributions

We take the first step to explore how to transfer winning tickets across different architectures. The goal itself would be too daunting if no constraint is imposed on the architecture differences. Just like general transfer learning, it is natural to hypothesize that two architectures must share some similarity so that their winning tickets may transfer: *we stick to this assumption in this preliminary study*.

We focus our initial agenda on network architectures from the same design family (e.g., a series of models via repeating or expanding certain building blocks) but of different depths. For a winning ticket found on one network, we propose various strategies to "stretch" it into a winner ticket for a

deeper network of the same family, or "squeeze" it for a shallower network. We then compare their performance with tickets directly found on those deeper or shallower networks. We conduct extensive experiments on CIFAR-10 with models from the ResNet and VGG families, and further extend to ImageNet. Our results seem to suggest an affirmative answer to our question in this specific setting.

We formalize our observations by articulating the *Elastic Lottery Ticket Hypothesis* (**E-LTH**): by mindfully replicating (or dropping) and re-ordering layers for one network, its corresponding winning ticket could be stretched (or squeezed) into a subnetwork for another deeper (or shallower) network from the same family, whose performance is nearly the same as the latter's winning ticket directly found by IMP. Those stretched or squeezed winning tickets largely outperform the sparse subnetworks found by pruning-at-initialization approaches [23], and show competitive efficiency to state-of-the-art dynamic sparse training [9]. We also provide intuitive explanations for the preliminary success.

Lastly, we stress that our method has a pilot-study nature, and is not assumption-free. The assumption that different architectures come from one design "family" might look restrictive. However, we point out many state-of-the-art deep models are designed in "families", such as ResNets [19], MobileNets [20], EfficientNets [32], and Transformers [34]. Hence we see practicality in the current results, and we are also ready to extend them to more general notions - which we discuss in Section 5.

2 Related Work

2.1 Lottery Ticket Hypothesis Basics

The pioneering work [11] pointed out the existence of winning tickets at random initialization, and showed that these winning tickets can be found by IMP. Denote $f(x;\theta)$ as a deep network parameterized by θ and x as its input. A sub-network of f can be characterized by a binary mask m, which has exactly the same dimension as θ. When applying the mask m to the network, we obtain the sub-network $f(x;\theta \odot m)$, where $\odot$ is the Hadamard product operator.

For a network initialized with θ_0, the IMP algorithm with rewinding [13, 12] works as follows: (1) initialize m as an all-one mask; (2) train $f(x;\theta_0 \odot m)$ for r steps to get θ_r; (3) continue to fully train $f(x;\theta_r \odot m)$ to obtain a well-trained θ; (3) remove a small portion $p \in (0,1)$ of the remaining weights with the smallest magnitudes from $\theta \odot m$ and update m; (5) repeat (3)-(4) until a certain sparsity ratio is achieved. Note that when $r = 0$, the above algorithm reduces to IMP without rewinding [11]. Rewinding is found to be essential for successfully and stably identifying winning tickets in large networks [13, 12]. [39] identified Early-Bird Tickets that contain structured sparsity, which emerge at the early stage of the training process. [28, 3] studied the transferability of winning tickets between datasets; the former focuses on showing one winning ticket to generalize across datasets and optimizers; and the latter investigates LTH in large pre-trained NLP models, and demonstrates the winning ticket transferability across downstream tasks.

2.2 Pruning in the Early Training, and Dynamic Sparse Training

A parallel line of works study the pruning of networks at either the initialization or the early stage of training, so that the resulting subnetworks can achieve close performance to the dense model when fully trained. *SNIP* [23] proposed to prune weights that are the least salient for the loss in the one-shot manner. *GraSP* [35] further exploited the second-order information of the loss at initialization and prune the weights that affect gradient flows least. [33] unified these methods under a newly proposed invariant measure called *synaptic saliency*, and showed that pruning iteratively instead of in one-shot is essential for avoiding layer collapse. The authors then propose an iterative pruning method called *SynFlow* based on synaptic saliency that requires no access to data.

However, it is observed in [14] that vanilla magnitude-based pruning [18] is as competitive as the above carefully designed pruning methods in most cases, yet all are inferior to IMP by clear margins. The pruning-at-initialization methods such as SNIP and GraSP were suggested to identify no more than a layer-wise pruning ratio configuration. That was because the SNIP/GraSP masks were found to be insensitive to mask shuffling within each layer, while LTH masks were impacted a lot by the same. Those suggest that existing early-pruning methods are not yet ready to replace IMP.

Instead of searching for static sparse subnetworks, another line of influential works, called *dynamic sparse training (DST)*, focuses on training sparse subnetworks from scratch while dynamically changing the connectivity patterns. DST was first proposed in [27]. Following works improve DST by parameter redistribution [29, 24] and gradient-based methods [7, 9]. A recent work [25] suggested that successful DST needed to explore the training of possible connections sufficiently.

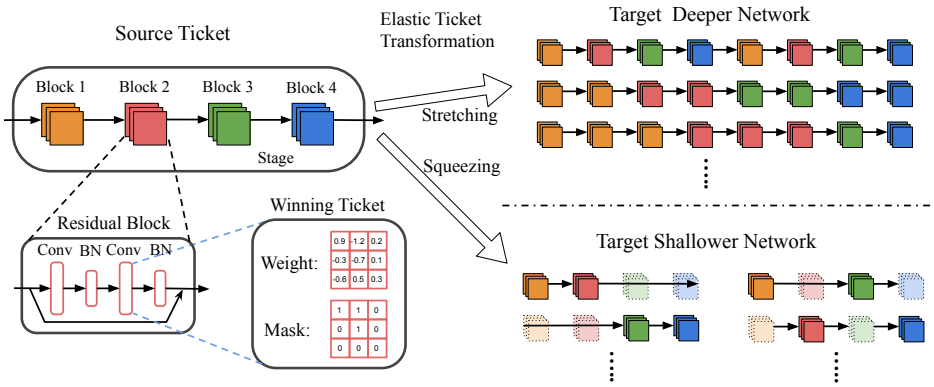

Figure 2: An illustration of Elastic Lottery Ticket Hypothesis, using a ResNet-type network as an example, which includes five residual blocks in each stage. Note that we hide "Block 0" who downsamples the input features as we will always preserve this block, without replicating or dropping. On the left, we use IMP to find winning tickets of the source network. On the right, we can either stretch the source ticket to a deeper network, or squeeze it to a shallower network, using the proposed Elastic Ticket Transformation (ETT). In both cases, we have a number of strategies for mindfully replicating, dropping and re-ordering the blocks.

2.3 Network Growing

Broadly related to this paper also includes the research on network growing. Net2Net [4] provided growing operations that preserve the functional equivalence for widening and deepening the network. Network Morphism [36] further extended Net2Net to more flexible operations that change the network architecture but maintains its functional representation. Network Morphism is used in [8] to generate several neighbouring networks to select the best one based on some training and evaluation. However, this requires comparing multiple candidate networks simultaneously. A more recent work, FireFly [38], proposed a joint optimization framework that grows the network during training.

Despite our similar goal of transferring across architectures, the above methods do not immediately fit into our case of winning tickets. We face the extra key hurdle that we need to transfer (stretch or squeeze) not only general weights, but also the sparse mask structure while preserving approximately the same sparsity. For example, Net2Net and FireFly add (near) identity layers as the initialization for the extra layers, which are already highly sparse and incompatible with IMP without modification. Network Morphism decomposes one layer into two by alternating optimization, which will break the sparse structure of the source winning tickets.

3 Elastic Lottery Ticket Hypothesis

The overall framework of E-LTH is illustrated in Figure 2. At the core of E-LTH are a number of *Elastic Ticket Transformations (ETTs)* proposed by us to convert the sparse architecture of a winning ticket to its deeper or shallower variants. We present rules of thumb for better performance, as well as discussions and intuitive explanation for why they do/do not work. For the simplicity of illustration, we will use the ResNet family as examples in this section.

Terminology: Suppose we have identified a winning ticket $f^s(\theta_r^s, m^s)$ on a **source network** using IMP, characterized by the rewinding weights θ_r^s and the binary mask m_s, where r is the rewinding steps and s stands for "source". Our goal is to transform the ticket f^s into a **target network** $f^t\left(g(\theta_r^s, m^s), h(\theta_r^s, m^s)\right)$ with a different depth directly, avoiding running the expensive IMP on the target network again. Here, the superscript t stands for "target". $g(\cdot,\cdot)$ and $h(\cdot,\cdot)$ are the transformation mapping for rewinding weights and the mask, respectively, taking the source weights and source mask as inputs.

3.1 Stretching into Deeper Tickets

We first present ETTs for stretching a winning ticket, which select certain layers to replicate, both in the (rewinding) weights and the sparse mask. Below, we discuss the major design choices.

Minimal unit for replication. Intuitively, "layer" is the most natural choice for the minimal unit in neural networks. Normalization layers (e.g., Batch Normalization layers [21]) are widely used in modern networks and play essential roles in the successful training of the networks. Therefore, in

this paper, ETTs considers a linear or convolutional layer and the normalization layer associated with it as a whole "layer". We find this to work well for VGG networks [31].

However, in ResNet [19] or its variants, which consist of multiple residual blocks, we consider the residual block as the minimal unit, because they are the minimal repeating structure. Moreover, the residual blocks, which are composed of two or three convolutional layers with normalization layers and a shortcut path, can be interpreted from different perspectives. For example, one residual block is interpreted as one time step for a forward Euler discretization in dynamical systems [37], or as one iteration that refines a latent representation estimation [16]. More discussion are in Section 3.3.

Invariant components. To ensure generalizability, we prefer minimal modification to the architectures involved. Hence, we will keep several components of the networks unaffected during stretching:

- **The number of stages**, which are delimited by the occurrences of down-sampling blocks. Adding new stages change the dimensions of intermediate features and create new layers with totally different dimensions, which are hard to be transferred from source tickets.
- **The down-sampling blocks**. In ResNet networks, the first block of each stage is a down-sampling block. In ETTs, we directly transplant the down-sampling blocks to the target network without modification and keep the one-to-one relationship with the stages.
- **The input and output layers**. Thanks to the first invariance, the input and output layers have consistent dimensions in the source and target networks and thus can be directly reused.

Which units to replicate? Let us take ResNet-20 (source) and ResNet-32 (target) as an example. Each stage of ResNet-20 contains one down-sampling block, B_0^s, and two normal blocks, B_1^s and B_2^s, while ResNet-32 has four normal blocks for each stage. To stretch a winning ticket of ResNet-20 to ResNet-32, we need to add two residual blocks in each stage. Then, *should we replicate both B_1^s and B_2^s once, or only replicate B_1^s (or B_2^s) twice? If the latter, which one?*

For the first question, in ETTs we choose to replicate more unique source blocks and for fewer times, i.e., we prefer to replicate both B_1^s and B_2^s once in the above example. As will be explained in Section 3.3, this could be understood as a uniform interpolation of the discretization steps in the dynamical system, a natural way to go finer-resolution along time. Another practical motivation for the choice is that such strategy can better preserve the sparsity ratio of each stage and thus the overall network sparsity. If B_1^s and B_2^s have different sparsity ratios (we observe that usually later blocks are sparser), replicating only one of them for several times will either increase or decrease the overall sparsity; in comparison, replicating both B_1^s and B_2^s proportionally maintains the overall sparsity.

If we compare replicating either B_1^s or B_2^s, the former will lead to better performance due to resultant lower sparsity (see above explained), and the latter causing lower performance due to effective higher sparsity. Both are understandable and could be viewed as trade-off options.

How to order the replicated units? Following the ResNet-20 to ResNet-32 example above, we replicate B_1^s and B_2^s once for each of them, resulting in two possible ordering of the replicated blocks: (1) $B_0^s \to B_1^s \to B_2^s \to B_1^s \to B_2^s$, which we call *appending* as the replicated blocks are appended as a whole after the last replicated block; (2) $B_0^s \to B_1^s \to B_1^s \to B_2^s \to B_2^s$, which we call *interpolation* as each replicated block is inserted right after its source block. We conduct experiments using both ordering strategy and observe comparable performance.

3.2 Squeezing into Shallower Tickets

Squeezing the winning tickets in a source network into a shallower network is the reverse process of stretching, and now we need to decide which units to drop. Therefore, we have symmetric design choices as ticket stretching, besides following the same minimal unit and invariances.

To squeeze a winning ticket from ResNet-32 into ResNet-20, we need to drop two blocks in each stage. The first question is: should we drop consecutive blocks, e.g., B_1^s, B_2^s or B_3^s, B_4^s, which corresponds to the inverse process of the *appending* ordering, or drop non-consecutive blocks, e.g., B_1^s, B_3^s or B_2^s, B_4^s, which corresponds to the inverse process of the *interpolation* ordering above? The second question is: in either case, should we drop the earlier or the later blocks?

According to the ablation study in Section 4.1, ETTs are not sensitive to whether we drop blocks consecutively or at intervals; however, it is critical that we do not drop too many early blocks.

3.3 Rationale and Preliminary Hypotheses

We draw two perspectives that may intuitively explain the effectiveness of ETTs during stretching and squeezing. Note that both explanations are fairly restricted to the ResNet family, while E-LTH seems

to generalize well beyond ResNets. Hence they are only our preliminary hypotheses, and further theoretical understandings of ETTs will be future work.

Dynamical systems perspective: ResNets have been interpreted as a discretization of dynamical systems [37, 17, 1]. Each residual block in the network can be seen as one step of a forward Euler discretization, with an implicit time step size of an initial value ordinary differential equation (ODE). Under this interpretation, adding a residual block right after each source block, that copies the weights and batch normalization parameters from the source block, can be seen as a uniform interpolation of this forward Euler discretization, by doubling the number of time steps while halving the implicit step size. Under the same unified view, if we replicate only B_1^s (or B_2^s), that could be seen as performing non-uniform interpolation, that super-solves only one time interval but not others. Without pre-assuming which time step is more critical, the uniform interpolation is the plausible choice, hence providing another possible understanding of our design choice in Section 3.1.

Unrolled estimation perspective: [16] interprets ResNets as unrolled iterative estimation process: each stage has a latent representation for which the first block in this stage generates a rough estimation and the remaining blocks keep refining it. This perspective provides a strong motivation to keep the first block of each stage untouched, as it contributes to the important initial estimate for the latent representation. Replicating or dropping the remaining residual blocks will incrementally affect the latent representation estimation. The interpolation method for stretching then corresponds to running every refining step for multiple times and the appending method corresponds to re-running (part of) the refining process again for better estimation. It is also implied by [16] that since each residual block is incremental, removing blocks only has a mild effect on the final representation, providing intuition for the effectiveness of squeezing tickets by dropping blocks.

4 Numerical Evaluations

We conduct extensive experiments on CIFAR-10 [22] and then extend to ImageNet [6], transferring the winning tickets across multiple models from ResNet family and VGG family. The implementation and experiment details are presented in Appendix. We run all experiments three times independently with different seeds. Besides IMP and the unpruned dense models, we will also compare ETTs with state-of-the-art pruning-at-initialization methods, including **SNIP** [23], **GraSP** [35] and One-shot **Magnitude**-based pruning which was suggested by [14] as a strong baseline. We also include two common pruning baselines in LTH works: (1) *Reinitialization* (**Reinit**), which preserves the identified sparse mask but reinitialize the rewinding weights; and (2) **Random** Pruning, which keeps the rewinding weights yet permuting the masks in a layerwise fashion (e.g., only preserving the layerwise sparsity ratios).

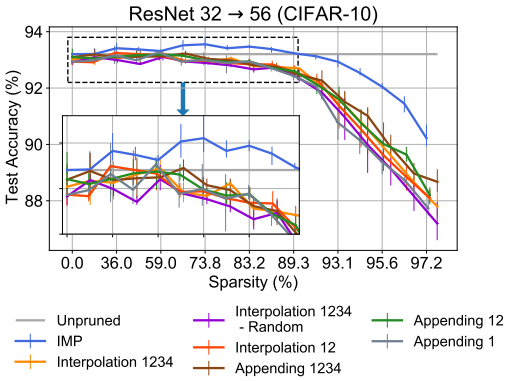

Figure 3: Ablation study on replicating unique units on CIFAR-10, using ETTs to transform tickets in ResNet-32 to ResNet-56.

Throughout this paper, we use "sparsity" or "sparsity ratio" to denote the portion of zero elements in networks due to pruning. Thus, the higher the sparsity ratio, the more parameters pruned. Note that ETTs may cause (slightly) misaligned sparsity ratios with IMP and other pruning methods because replicating and dropping blocks may result in the sparsity changes in a less controllable way. For a fair comparison, all pruning methods will match the sparsity ratios of the winning tickets generated by IMP, to the best possible extent. We apply pruning methods to the rewinding weights instead of initialization because the former was found to substantially improve LTH performance [14].

4.1 A Thorough Ablation Study on ResNet-32 and Resnet-56

In this subsection, we conduct an ablation study about the rules of thumb for constructing better ETTs as discussed in Section 3. We investigate the selection of replicated or squeezed units when we apply ETTs to stretch or squeeze a winning ticket, and the order of replicated units for the stretching case. All results reported are the average of three independent runs of experiments.

Should we replicate more unique units? ResNet-32 has five residual blocks in each stage, notated as $[B_0, \ldots B_4]$. As we discussed in Section 3, we leave the downsampling block B_0 alone and only

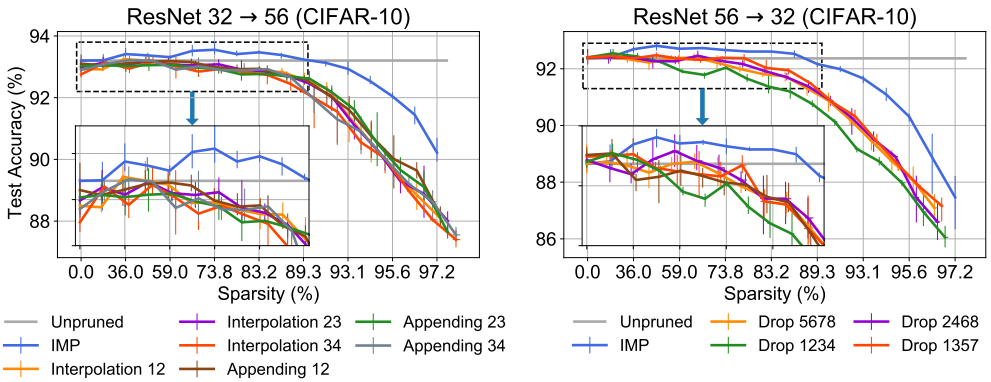

Figure 4: Results of the ablation study on replicating/dropping earlier or later units the networks. Left: stretching the winning tickets of ResNet-32 into ResNet-56 by appending/interpolating different residual blocks. Right: squeezing the tickets of ResNet-56 into ResNet-32 by dropping four blocks at different positions. Results are the average of three independent runs with error bars.

play with the rest four. We consider three options, i.e., replicating (1) all four blocks; (2) the first two blocks; (3) and the first block (B_1). We run for each option both the appending and the interpolation methods (note that appending interpolating the first block yields the same resulting model).

The results are shown in Figure 3, where the numbers in the legend are the indices of replicated blocks. We can see that replicating four blocks is slightly better than replicating two for both appending and interpolation and much better than only replicating the first by clear gap. The gaps enlarge when the sparsity ratios grow higher. At lower sparsity ratios, the differences between those options are smaller. We also include an option, "Interpolation 1234 - Random", in which we randomly permute the masks of the replicated blocks (but preserves the masks of the original blocks). That shows better than its random-pruning variant at all sparsity ratios and the advantage is more evident at high sparsity ratios.

The earlier or the later units? The next question in ETTs that follows is: should we replicate earlier units in stretching for better performance, or the later ones? And similarly, should we drop earlier units when squeezing the tickets? Is one option always better than the other? For the stretching part, we try to replicate (B_1, B_2), (B_2, B_3), (B_3, B_4) using appending and interpolation methods. Results in the top subfigure of Figure 4 show the advantage of replicating earlier blocks than the later ones with high sparsity, while the differences are less obvious in the low sparsity range. For squeezing, we also try different options of dropping blocks as shown in the bottom subfigure of Figure 4 and find that ETTs is not sensitive to dropping consecutive or non-consecutive blocks, which is consistent to our observation on the stretching experiments, as long as we do not drop earlier blocks too much – we can see a significant decrease in performance when we drop the first four blocks at the same time.

4.2 More Experiments on CIFAR-10

Experiments on more ResNets. In addition to the transformation between ResNet-32 and ResNet-56, we apply ETTs that uses *appending* operations to more ResNet networks to transfer the winning tickets across different structures. More complete results are presented in Figure 5. We can see that the tickets generated by ETTs from different source networks clearly outperform other pruning methods, with the only exception of transferring then smallest ResNet-14 to large target networks.

Another (perhaps not so surprising) observation we can draw from Figure 5 is that ETTs usually works better when the source and target networks have a smaller difference in depth. For example, when ResNet-44 is the target network, ETTs tickets transformed from ResNet-32/56 outperform those transformed from ResNet-14/20 by notable gaps. Similar comparisons can also be drawn on other target networks. On the other hand, the tickets transformed from ResNet-14 have the best performance than other source networks on ResNet-20, and are nearly as competitive as IMP. However, on ResNet-32, the performance of ETTs from ResNet-14 lies between ETTs from other source models and pruning methods. When the target model goes to ResNet-44 or beyond, ETTs from ResNet-14 performs no better than pruning-at-initialization methods. This implies that our replicating or dropping potentially still introduce noise, which may be acceptable within a moderate range, but might become too amplified when replicating or dropping too many times.

Experiments on VGG networks. We also apply ETTs to VGG networks, and report another comprehensive suite of experiments of transforming across VGG-13, VGG-19 and VGG-19 networks.

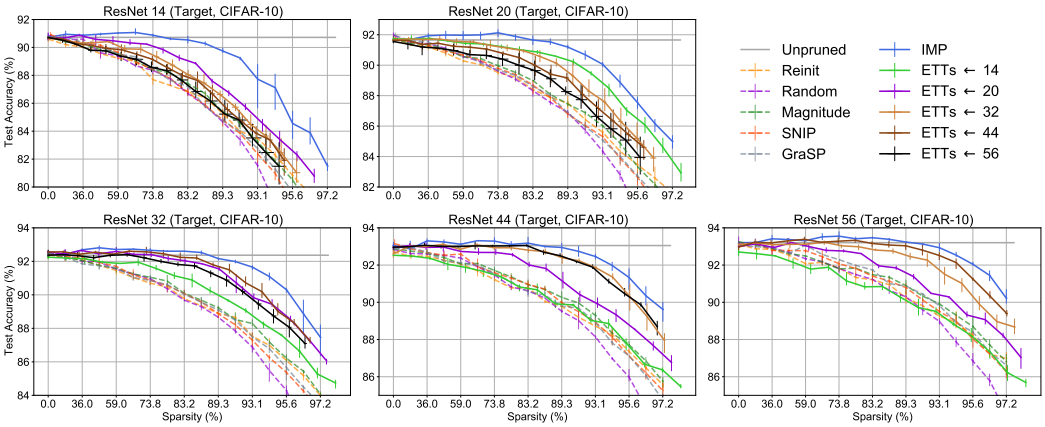

Figure 5: Results of the experiments on CIFAR-10 using 5 different ResNet networks: ResNet-14, ResNet-20, ResNet-32, ResNet-44, ResNet-56, left to right, top to bottom. ETTs $\leftarrow n$ in the legend stands for the ticket transformed from ResNet-n using ETTs, which is also absent in the subfigure where ResNet-n is the target network. Curves for pruning methods are dashed.

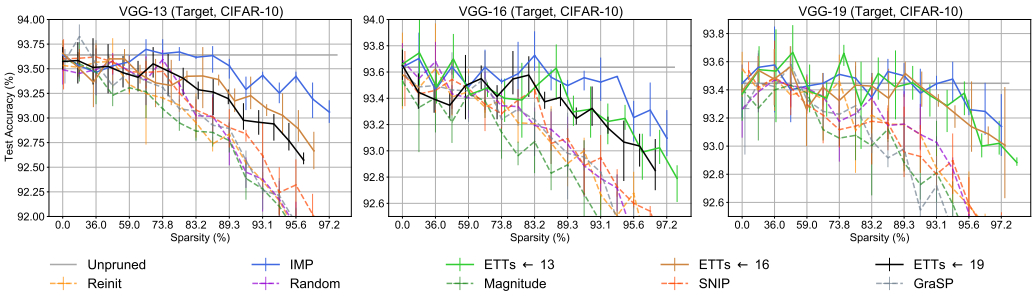

Figure 6: Results of the experiments on CIFAR-10 with VGG networks - VGG-13, -16 and -19. ETTs $\leftarrow n$ in the legend stands for the ticket transformed from VGG-n using ETTs, which is also absent in the subfigure where VGG-n is the target network. Curves for pruning methods are dashed.

Here we directly replicate/drop "layers", instead of residual blocks. The results shown in Figure 6 convey similar messages as the ResNet experiments, where we see very competitive performance with IMP, even at high sparsity ratios, and significant gaps when compared against other pruning counterparts, especially SNIP and GraSP.

4.3 Experiments on ImageNet

Next, we extend the experiments to ImageNet. We adopt three models: ResNet-18, ResNet-26, and ResNet-34. We select sparsity ratio 73.79% as the test bed, at which level IMP with rewinding can successfully identify the winning tickets [13]. Our results are presented in Table 1, where we can see ETTs works effectively when transforming tickets to ResNet-26 and ResNet-34, yielding accuracies comparable to the full model and the IMP-found winning ticket, which are much higher than other pruning counterparts. Transforming to ResNet-18 seems slightly more challenging for ETTs, but its superiority over other pruning methods is still solid.

Table 1: E-LTH for ResNets on ImageNet.

Network	Res-18	Res-26	Res-34
Full Model	69.96%	72.56%	73.77%
IMP	70.22%	72.94%	74.20%
Reinit	62.88%	66.97%	68.36%
Random	63.10%	67.07%	68.68%
Magnitude	64.96%	68.51%	69.74%
SNIP	62.23%	67.51%	69.35%
GraSP	62.85%	67.24%	69.23%
ETTs ← 18	-	71.29%	71.86%
ETTs ← 26	**68.17%**	-	**73.37%**
ETTs ← 34	68.08%	**72.47%**	-

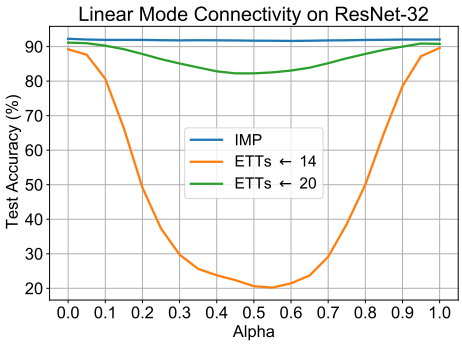

Figure 7: Linear mode connectivity property on ResNet-32. We compare winning tickets directly found on ResNet-32 using IMP and E-LTH tickets transferred from on ResNet-l4 and -20.

4.4 Linear Mode Connectivity of E-LTH

[12] observed that winning tickets are stably optimized into a linearly connected minimum under different samples of SGD noises (e.g., data order and augmentation), which means that linear interpolations of two independently trained winning tickets have comparable performance. In contrast, linear interpolations for other sparse subnetworks have severely degraded performance. Here, we observe such "linear mode connectivity" property on tickets transferred from ResNet-14 and ResNet-20 to ResNet-32, shown in Figure 7. We can see that IMP is stable at the linear interpolations of two trained IMP tickets, as observed in [12]. ETTs from ResNet-20 is less stable than IMP in terms of linear mode connectivity, with a maximal 8.72% accuracy drop, but much more stable than ETTs from ResNet-14, with a maximal 69.72% drop. This is consistent with our observation on model accuracies – ETTs has better linear mode connectivity when the source and target networks have a smaller difference in depth, thus having better performance.

Implication: Possibility to Transfer General Pruned Solutions. The above experiments indicate lottery tickets transfer better via ETTs when they stay in the same basin, which was found to be the same basin as the pruning solution [10]. That implies ETTs might be able to transfer pruned solutions in general [30] – this is out of the current work's focus, but would definitely be our future work.

4.5 Comparison to Dynamic Sparse Training (RigL)

Dynamic sparse training (DST) is an uprising direction to train sparse networks from scratch by dynamically changing the connectivity patterns, while keeping the overall sparse ratios (thus computation FLOPs) low. However, **a crucial difference** between E-LTH and DST is that E-LTH is a "one-for-all" method. For example if we run IMP on ResNet-34 once, we then obtain tickets for ResNet-20, ResNet-44 and more "for free" simultaneously, by applying ETTs. In contrast, DST methods have to run on each architecture independently from scratch. Therefore, any overhead of E-LTH is *amortized* by transferring to many different-depth architectures: that is conceptually similar to LTH's value in pre-training [2, 3]. E-LTH hence has strong potential in scenarios where we have different hardware constraints on various types of devices - we can flexibly adapt one ticket to satisfy different constraints without applying IMP or DST methods repeatedly.

Table 2: Compare E-LTH and RigL-5x on ImageNet using ResNet-18/26/34 (use ResNet-26 as the source model).

Method	E-LTH	RigL-5x
ResNet-18	68.17%	69.59%
ResNet-26	72.94%	72.66%
ResNet-34	73.37%	73.88%

To quantitatively understand our relative costs, we compare E-LTH with one recent state-of-the-art DST algorithm, RigL [9], using ResNet-18/26/34 on ImageNet (ResNet-26 as source model). We use 73.79% sparsity to be consistent with Section 4.3. For result clarity, we have normalized all FLOPs numbers, by the FLOPs of one-pass standard dense training on ResNet-34[1]. Table 2 indicates comparable actuaries between E-LTH and RigL on three models. RigL uses 0.69x normalized FLOPs to train ResNet-18, 1.0x FLOPs for ResNet-26, and 1.30x FLOPs for ResNet-34. Meanwhile, E-LTH first uses 2.77x FLOPs to find the mask once on ResNet-26; then after applying the (transferred) mask, it onlu takes 0.14x/0.20x/0.26x additional FLOPs to train ResNet-18/26/34, respectively. Taking together, E-LTH and RigL use similar total FLOPs on training the three models; but note that E-LTH will become more cost-effective as the found mask is transferred to more networks.

5 Conclusions and Discussions of Broad Impact

This paper presents the first study of the winning ticket transferability between different network architectures (within the same design family), and concludes with the Elastic Lottery Ticket Hypothesis (E-LTH). E-LTH has significantly outperformed SOTA pruning-at-initialization methods, e.g., SNIP and GraSP. Our finding reveals inherent connections between lottery ticket subnetworks derived from similar models, and suggests brand-new opportunities in practice, such as efficient ticket finding for large networks or adaptation under resource constraints. For further discussions about the limitation, deeper questions and future directions, please see the Appendix. We do not see that this work will directly impose any negative social risk. Besides the intellectual merits, the largest potential societal impact that we can see in this work is helping understand lottery tickets and sparse network training. It also provides a solution to flexibly deploy networks with different complexity,

[1] Different from what was calculated in the **appendix** of [9], here we use a **different sparsity ratio** (73.79%) than the original RigL paper (90%) for fair comparison, and uses the default 5× training steps for the best of RigL performance, which results in (1-73.79%)x5 = 1.30x FLOPs. Similar calculation applies to ResNet-18 and -26. In comparison, E-LTH uses normal training steps (1×).

References

[1] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level residual networks from dynamical systems view. In *International Conference on Learning Representations*, 2018.

[2] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Michael Carbin, and Zhangyang Wang. The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. *arXiv preprint arXiv:2012.06908*, 2020.

[3] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks, 2020.

[4] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[5] Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*, 2020.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

[8] Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *arXiv preprint arXiv:1711.04528*, 2017.

[9] 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.

[10] Utku Evci, Yani A Ioannou, Cem Keskin, and Yann Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. *arXiv preprint arXiv:2010.03533*, 2020.

[11] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[12] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020.

[13] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

[14] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

[15] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[16] Klaus Greff, Rupesh K Srivastava, and Jürgen Schmidhuber. Highway and residual networks learn unrolled iterative estimation. *arXiv preprint arXiv:1612.07771*, 2016.

[17] Eldad Haber, Lars Ruthotto, Elliot Holtham, and Seong-Hwan Jun. Learning across scales—multiscale methods for convolution neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[18] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[22] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[23] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY. In *International Conference on Learning Representations*, 2019.

[24] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. *arXiv preprint arXiv:2101.09048*, 2021.

[25] Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. *arXiv preprint arXiv:2102.02887*, 2021.

[26] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

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

[28] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *NeurIPS*, pages 4932–4942, 2019.

[29] 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.

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

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[32] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019.

[33] Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

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

[36] Tao Wei, Changhu Wang, Yong Rui, and Chang Wen Chen. Network morphism. In *International Conference on Machine Learning*, pages 564–572. PMLR, 2016.

[37] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[38] Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

[39] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G. Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Toward more efficient training of deep networks. In *ICLR*, 2020.

[40] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *NeurIPS*, pages 3597–3607, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Please refer to the Appendix.
   (c) Did you discuss any potential negative societal impacts of your work? [No] We do not see that this work will impose direct social risks on the society.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We will publish them upon acceptance.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report error bars for all figures in this paper.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] The original papers of CIFAR-10 and ImageNet include the license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include the official lottery ticket GitHub repo as a url in the supplementary.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We only use public datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]