Pruning neural networks without any data by iteratively conserving synaptic flow

Hidenori Tanaka*
Physics & Informatics Laboratories
NTT Reserach, Inc.
Department of Applied Physics
Stanford University

Daniel Kunin*
Institute for Computational and
Mathematical Engineering
Stanford University

Daniel L. K. Yamins
Department of Psychology
Department of Computer Science
Stanford University

Surya Ganguli
Department of Applied Physics
Stanford University

Abstract

Pruning the parameters of deep neural networks has generated intense interest due to potential savings in time, memory and energy both during training and at test time. Recent works have identified, through an expensive sequence of training and pruning cycles, the existence of winning lottery tickets or sparse trainable subnetworks at initialization. This raises a foundational question: can we identify highly sparse trainable subnetworks at initialization, without ever training, or indeed *without ever looking at the data*? We provide an affirmative answer to this question through theory driven algorithm design. We first mathematically formulate and experimentally verify a conservation law that explains why existing gradient-based pruning algorithms at initialization suffer from layer-collapse, the premature pruning of an entire layer rendering a network untrainable. This theory also elucidates how layer-collapse can be entirely avoided, motivating a novel pruning algorithm *Iterative Synaptic Flow Pruning (SynFlow)*. This algorithm can be interpreted as preserving the total flow of synaptic strengths through the network at initialization subject to a sparsity constraint. Notably, this algorithm makes no reference to the training data and consistently outperforms existing state-of-the-art pruning algorithms at initialization over a range of models (VGG and ResNet), datasets (CIFAR-10/100 and Tiny ImageNet), and sparsity constraints (up to 99.9 percent). Thus our data-agnostic pruning algorithm challenges the existing paradigm that data must be used to quantify which synapses are important.

1 Introduction

Network pruning, or the compression of neural networks by removing parameters, has been an important subject both for reasons of practical deployment [1, 2, 3, 4, 5, 6, 7] and for theoretical understanding of artificial [8] and biological [9] neural networks. Conventionally, pruning algorithms have focused on compressing pre-trained models [1, 2, 3, 5, 6]. However, recent works [10, 11] have identified through iterative training and pruning cycles (*iterative magnitude pruning*) that there exist sparse subnetworks (*winning tickets*) in randomly-initialized neural networks that, when trained in isolation, can match the test accuracy of the original network. Moreover, its been shown that some of these winning ticket subnetworks can generalize across datasets and optimizers [12]. While these results suggest training can be made more efficient by identifying winning ticket subnetworks at initialization, they do not provide efficient algorithms to find them. Typically, it requires significantly more

*Equal contribution. Correspondence to hidenori.tanaka@ntt-research.com and kunin@stanford.edu.

Preprint. Under review.

computational costs to identify winning tickets through iterative training and pruning cycles than simply training the original network from scratch [10, 11]. Thus, the fundamental unanswered question is: can we identify highly sparse trainable subnetworks at initialization, without ever training, or indeed *without ever looking at the data*? Towards this goal, we start by investigating the limitations of existing pruning algorithms at initialization [13, 14], determine simple strategies for avoiding these limitations, and provide a novel data-agnostic algorithm that improves upon state-of-the-art results. Our main contributions are:

1. We study *layer-collapse*, the premature pruning of an entire layer making a network untrainable, and formulate the axiom *Maximal Critical Compression* that posits a pruning algorithm should avoid layer-collapse whenever possible (Sec. 3).
2. We demonstrate theoretically and empirically that *synaptic saliency*, a general class of gradient-based scores for pruning, is conserved at every hidden unit and layer of a neural network (Sec. 4).
3. We show that these *conservation laws* imply parameters in large layers receive lower scores than parameters in small layers, which elucidates why single-shot pruning disproportionately prunes the largest layer leading to layer-collapse (Sec. 4).
4. We hypothesize that *iterative magnitude pruning* [10] avoids layer-collapse because gradient descent effectively encourages the magnitude scores to observe a conservation law, which combined with iteration results in the relative scores for the largest layers increasing during pruning (Sec. 5).
5. We prove that a pruning algorithm avoids layer-collapse entirely and satisfies Maximal Critical Compression if it uses iterative, positive synaptic saliency scores (Sec. 6).
6. We introduce a new data-agnostic algorithm *Iterative Synaptic Flow Pruning (SynFlow)* that satisfies Maximal Critical Compression (Sec. 6) and demonstrate empirically[2] that this algorithm achieves state-of-the-art pruning performance on 12 distinct combinations of models and datasets (Sec. 7).

2 Related work

While there are a variety of approaches to compressing neural networks, such as novel design of micro-architectures [15, 16, 17], dimensionality reduction of network parameters [18, 19], and training of dynamic sparse networks [20, 21], in this work we will focus on neural network pruning.

Pruning after training. Conventional pruning algorithms assign scores to parameters in neural networks *after* training and remove the parameters with the lowest scores [5, 22, 23]. Popular scoring metrics include weight magnitudes [4, 6], its generalization to multi-layers [24], first- [1, 25, 26, 27] and second-order [2, 3, 27] Taylor coefficients of the training loss with respect to the parameters, and more sophisticated variants [28, 29, 30]. While these pruning algorithms can indeed compress neural networks at test time, there is no reduction in the cost of training.

Pruning before Training. Recent works demonstrated that randomly initialized neural networks can be pruned *before* training with little or no loss in the final test accuracy [10, 13, 31]. In particular, the Iterative Magnitude Pruning (IMP) algorithm [10, 11] repeats multiple cycles of training, pruning, and weight rewinding to identify extremely sparse neural networks at initialization that can be trained to match the test accuracy of the original network. While IMP is powerful, it requires multiple cycles of expensive training and pruning with very specific sets of hyperparameters. Avoiding these difficulties, a different approach uses the gradients of the training loss at initialization to prune the network in a single-shot [13, 14]. While these single-shot pruning algorithms at initialization are much more efficient, and work as well as IMP at moderate levels of sparsity, they suffer from layer-collapse, or the premature pruning of an entire layer rendering a network untrainable [32, 33]. Understanding and circumventing this layer-collapse issue is the fundamental motivation for our study.

3 Layer-collapse: the key obstacle to pruning at initialization

Broadly speaking, a pruning algorithm at initialization is defined by two steps. The first step scores the parameters of a network according to some metric and the second step masks the parameters (removes or keeps the parameter) according to their scores. The pruning algorithms we consider will always mask the parameters by simply removing the parameters with the smallest scores. This ranking process can be applied globally across the network, or layer-wise. Empirically, its been shown that global-masking performs far better than layer-masking, in part because it introduces fewer hyperparameters and allows for flexible pruning rates across the network [23]. However, recent works [32, 14, 33] have identified a key failure mode, *layer-collapse*, for existing pruning algorithms using global-masking. Layer-collapse occurs when an algorithm prunes all

[2] All code is available at `github.com/ganguli-lab/Synaptic-Flow`.

parameters in a single weight layer even when prunable parameters remain elsewhere in the network. This renders the network untrainable, evident by sudden drops in the achievable accuracy for the network as shown in Fig. 1. To gain insight into the phenomenon of layer-collapse we will define some useful terms inspired by a recent paper studying the failure mode [33].

Given a network, *compression ratio* (ρ) is the number of parameters in the original network divided by the number of parameters remaining after pruning. For example, when the compression ratio $\rho = 10^3$, then only one out of a thousand of the parameters remain after pruning. *Max compression* ($\rho_{\max}$) is the maximal possible compression ratio for a network that doesn't lead to layer-collapse. For example, for a network with L layers and N parameters, $\rho_{\max} = N/L$, which is the compression ratio associated with pruning all but one parameter per layer. *Critical compression* (ρ_{cr}) is the maximal compression ratio a given algorithm can achieve without inducing layer-collapse. In particular, the critical compression of an algorithm is always upper bounded by the max compression of the network: $\rho_{\text{cr}} \leq \rho_{\max}$. This inequality motivates the following axiom we postulate any successful pruning algorithm should satisfy.

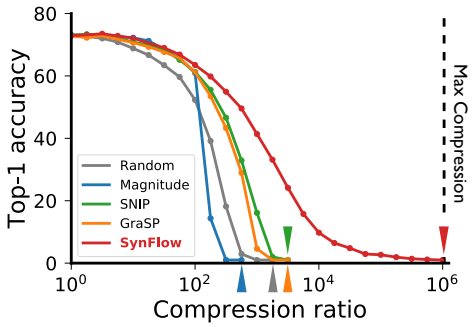


Figure 1: **Layer-collapse leads to a sudden drop in accuracy.** Top-1 test accuracy as a function of the compression ratio for a VGG-16 model pruned at initialization and trained on CIFAR-100. Colored arrows represent the critical compression of the corresponding pruning algorithm. Only our algorithm, SynFlow, reaches the theoretical limit of max compression (black dashed line) without collapsing the network. See Sec. 7 for more details on the experiments.

Axiom. ***Maximal Critical Compression.*** *The critical compression of a pruning algorithm applied to a network should always equal the max compression of that network.*

In other words, this axiom implies a pruning algorithm should never prune a set of parameters that results in layer-collapse if there exists another set of the same cardinality that will keep the network trainable. To the best of our knowledge, no existing pruning algorithm with global-masking satisfies this simple axiom. Of course any pruning algorithm could be modified to satisfy the axiom by introducing specialized layer-wise pruning rates. However, to retain the benefits of global-masking [23], we will formulate an algorithm, Iterative Synaptic Flow Pruning (SynFlow), which satisfies this property by construction. SynFlow is a natural extension of magnitude pruning, that preserves the total flow of synaptic strengths from input to output rather than the individual synaptic strengths themselves. We will demonstrate that not only does the SynFlow algorithm achieve Maximal Critical Compression, but it consistently outperforms existing state-of-the-art pruning algorithms (as shown in Fig. 1 and in Sec. 7), all while not using the data.

Throughout this work, we benchmark our algorithm, SynFlow, against two simple baselines, random scoring and scoring based on weight magnitudes, as well as two state-of-the-art single-shot pruning algorithms, Single-shot Network Pruning based on Connection Sensitivity (SNIP) [13] and Gradient Signal Preservation (GraSP) [14]. SNIP [13] is a pioneering algorithm to prune neural networks at initialization by scoring weights based on the gradients of the training loss. GraSP [14] is a more recent algorithm that aims to preserve gradient flow at initialization by scoring weights based on the Hessian-gradient product. Both SNIP and GraSP have been thoroughly benchmarked by [14] against other state-of-the-art pruning algorithms that involve training [2, 34, 10, 11, 35, 21, 20], demonstrating competitive performance.

4 Conservation laws of synaptic saliency

In this section, we will further verify that layer-collapse is a key obstacle to effective pruning at initialization and explore what is causing this failure mode. As shown in Fig. 2, with increasing compression ratios, existing random, magnitude, and gradient-based pruning algorithms will prematurely prune an entire layer making the network untrainable. Understanding why certain score metrics lead to layer-collapse is essential to improve the design of pruning algorithms.

Random pruning prunes every layer in a network by the same amount, evident by the horizontal lines in Fig. 2. With random pruning the *smallest layer*, the layer with the least parameters, is the first to be fully pruned. Conversely, magnitude pruning prunes layers at different rates, evident by the staircase pattern in Fig. 2. Magnitude pruning effectively prunes parameters based on the variance of their initialization, which for common network initializations, such as Xavier [36] or Kaiming [37], are inversely proportional to the width of a layer [33]. With magnitude pruning the *widest layers*, the layers with largest input or output dimensions, are the

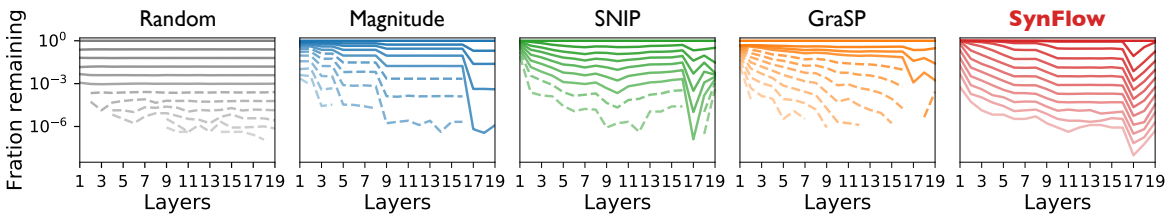


Figure 2: **Where does layer-collapse occur?** Fraction of parameters remaining at each layer of a VGG-19 model pruned at initialization with ImageNet over a range of compression ratios (4^n for $n = 0, 1, \ldots, 11$). A higher transparency represents a higher compression ratio. A dashed line indicates that there is at least one layer with no parameters, implying layer-collapse has occurred.

first to be fully pruned. Gradient-based pruning algorithms SNIP [13] and GraSP [14] also prune layers at different rates, but it is less clear what the root cause for this preference is. In particular, both SNIP and GraSP aggressively prune the *largest layer*, the layer with the most trainable parameters, evident by the sharp peaks in Fig. 2. Based on this observation, we hypothesize that gradient-based scores averaged within a layer are inversely proportional to the layer size. We examine this hypothesis by constructing a theoretical framework grounded in flow networks. We first define a general class of gradient-based scores, prove a conservation law for these scores, and then use this law to prove that our hypothesis of inverse proportionality between layer size and average layer score holds exactly.

A general class of gradient-based scores. *Synaptic saliency* is any score metric that can be expressed as the Hadamard product

$$\mathcal{S}(\theta) = \frac{\partial \mathcal{R}}{\partial \theta} \odot \theta, \tag{1}$$

where $\mathcal{R}$ is a scalar loss function of the output of a feed-forward network parameterized by θ. When $\mathcal{R}$ is the training loss $\mathcal{L}$, the resulting synaptic saliency metric is equivalent (modulo sign) to $-\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta$, the score metric used in Skeletonization [1], one of the first network pruning algorithms. The resulting metric is also closely related to $\left|\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta\right|$ the score used in SNIP [13], $-\left(H \frac{\partial \mathcal{L}}{\partial \theta}\right) \odot \theta$ the score used in GraSP, and $\left(\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta\right)^2$ the score used in the pruning after training algorithm Taylor-FO [27]. This general class of score metrics, while not encompassing, exposes key properties of gradient-based scores used for pruning.

The conservation of synaptic saliency. All synaptic saliency metrics respect two surprising conservation laws that hold at any initialization and step in training.

Theorem 1. ***Neuron-wise Conservation of Synaptic Saliency.*** *For a feedforward neural network with homogenous activation functions, $\phi(x) = \phi'(x)x$, (e.g. ReLU, Leaky ReLU, linear), the sum of the synaptic saliency for the incoming parameters to a hidden neuron ($\mathcal{S}^{in}$) is equal to the sum of the synaptic saliency for the outgoing parameters from the hidden neuron ($\mathcal{S}^{out}$).*

Proof. Consider the j^{th} hidden neuron of a network with outgoing parameters θ^{out}_{ij} and incoming parameters θ^{in}_{jk}, such that $\frac{\partial \mathcal{R}}{\partial \phi(z_j)} = \sum_i \frac{\partial \mathcal{R}}{\partial z_i} \theta^{\text{out}}_{ij}$ and $z_j = \sum_k \theta^{\text{in}}_{jk} \phi(z_k)$. The sum of the synaptic saliency for the outgoing parameters is

$$\mathcal{S}^{\text{out}} = \sum_i \frac{\partial \mathcal{R}}{\partial \theta^{\text{out}}_{ij}} \theta^{\text{out}}_{ij} = \sum_i \frac{\partial \mathcal{R}}{\partial z_i} \phi(z_j) \theta^{\text{out}}_{ij} = \left(\sum_i \frac{\partial \mathcal{R}}{\partial z_i} \theta^{\text{out}}_{ij} \right) \phi(z_j) = \frac{\partial \mathcal{R}}{\partial \phi(z_j)} \phi(z_j). \tag{2}$$

The sum of the synaptic saliency for the incoming parameters is

$$\mathcal{S}^{\text{in}} = \sum_k \frac{\partial \mathcal{R}}{\partial \theta^{\text{in}}_{jk}} \theta^{\text{in}}_{jk} = \sum_k \frac{\partial \mathcal{R}}{\partial z_j} \phi(z_k) \theta^{\text{in}}_{jk} = \frac{\partial \mathcal{R}}{\partial z_j} \left(\sum_k \theta^{\text{in}}_{jk} \phi(z_k) \right) = \frac{\partial \mathcal{R}}{\partial z_j} z_j. \tag{3}$$

When ϕ is homogeneous, then $\frac{\partial \mathcal{R}}{\partial \phi(z_j)} \phi(z_j) = \frac{\partial \mathcal{R}}{\partial z_j} z_j$. □

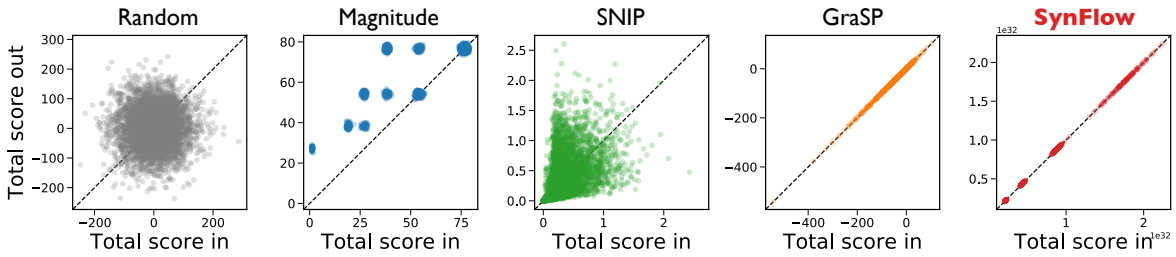


Figure 3: **Neuron-wise conservation of score.** Each dot represents a hidden unit from the feature-extractor of a VGG-19 model pruned at initialization with ImageNet. The location of each dot corresponds to the total score for the unit's incoming and outgoing parameters, $(\mathcal{S}^{\text{in}}, \mathcal{S}^{\text{out}})$. The black dotted line represents exact neuron-wise conservation of score.

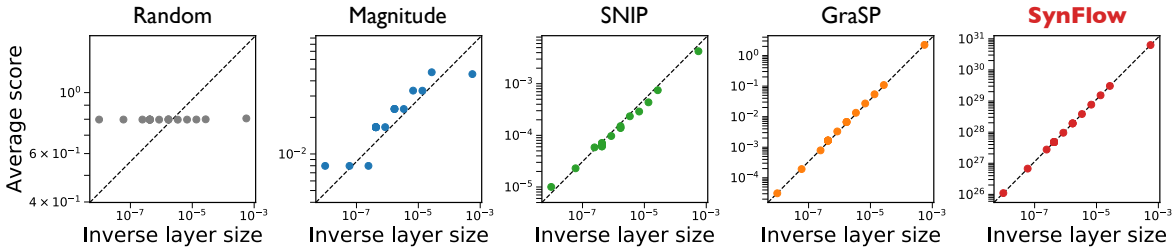


Figure 4: **Inverse relationship between layer size and average layer score.** Each dot represents a layer from a VGG-19 model pruned at initialization with ImageNet. The location of each dot corresponds to the layer's average score[4]and inverse number of elements. The black dotted line represents a perfect linear relationship.

The neuron-wise conservation of synaptic saliency implies network conservation as well.

Theorem 2. ***Network-wise Conservation of Synaptic Saliency.*** *The sum of the synaptic saliency across any set of parameters that exactly*[3] *separates the input neurons x from the output neurons y of a feedforward neural network with homogenous activation functions equals* $\langle \frac{\partial \mathcal{R}}{\partial x}, x \rangle = \langle \frac{\partial \mathcal{R}}{\partial y}, y \rangle$.

We prove this theorem in Appendix 10 by applying the neuron-wise conservation law recursively. Similar conservation properties have been noted in the neural network interpretability literature and have motivated the construction of interpretability methods such as Conductance [38] and Layer-wise Relevance Propagation [39], which have recently been modified for network pruning [9, 40]. While the interpretability literature has focused on attribution to the input pixels and hidden neuron activations, we have formulated conservation laws that are more general and applicable to any parameter and neuron in a network. Remarkably, these conservation laws of synaptic saliency apply to modern neural network architectures and a wide variety of neural network layers (e.g. dense, convolutional, batchnorm, pooling, residual) as visually demonstrated in Fig. 3.

Conservation and single-shot pruning leads to layer-collapse. The conservation laws of synaptic saliency provide us with the theoretical tools to validate our earlier hypothesis of inverse proportionality between layer size and average layer score as a root cause for layer-collapse of gradient-based pruning methods. Consider the set of parameters in a layer of a simple, fully connected neural network. This set would exactly separate the input neurons from the output neurons. Thus, by the network-wise conservation of synaptic saliency (theorem 2), the total score for this set is constant for all layers, implying the average is inversely proportional to the layer size. We can empirically evaluate this relationship at scale for existing pruning methods by computing the total score for each layer of a model, as shown in Fig. 4. While this inverse relationship is exact for synaptic saliency, other closely related gradient-based scores, such as the scores used in SNIP and GraSP, also respect this relationship. This validates the empirical observation that for a given compression ratio, gradient-based pruning methods will disproportionately prune the largest layers. Thus, if the compression ratio is large enough and the pruning score is only evaluated once, then a gradient-based pruning method will completely prune the largest layer leading to layer-collapse.

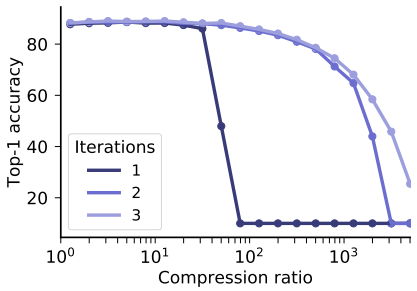
[3]Every element of the set is needed to separate the input neurons from the output neurons.

[4]For GraSP we negated the average layer score so that we could plot on a log-log plot.

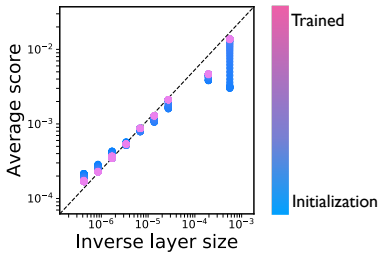
5 Magnitude pruning avoids layer-collapse with conservation and iteration

Having demonstrated and investigated the cause of layer-collapse in single-shot pruning methods at initialization, we now explore an iterative pruning method that appears to avoid the issue entirely. Iterative Magnitude Pruning (IMP) is a recently proposed pruning algorithm that has proven to be successful in finding extremely sparse trainable neural networks at initialization (winning lottery tickets) [10, 11, 12, 41, 42, 43, 44]. The algorithm follows three simple steps. First train a network, second prune parameters with the smallest magnitude, third reset the unpruned parameters to their initialization and repeat until the desired compression ratio. While simple and powerful, IMP is impractical as it involves training the network several times, essentially defeating the purpose of constructing a sparse initialization. That being said it does not suffer from the same catastrophic layer-collapse that other pruning at initialization methods are susceptible to. Thus, understanding better how IMP avoids layer-collapse might shed light on how to improve pruning at initialization.

As has been noted previously [10, 11], iteration is essential for stabilizing IMP. In fact, without sufficient pruning iterations, IMP *will* suffer from layer-collapse, evident in the sudden accuracy drops for the darker curves in Fig. 5a. However, the number of pruning iterations alone cannot explain IMP's success at avoiding layer-collapse. Notice that if IMP didn't train the network during each prune cycle, then, no matter the number of pruning iterations, it would be equivalent to single-shot magnitude pruning. Thus, something very critical must happen to the magnitude of the parameters during training, that when coupled with sufficient pruning iterations allows IMP to avoid layer-collapse. We *hypothesize* that gradient descent training effectively encourages the scores to observe an approximate layer-wise conservation law, which when coupled with sufficient pruning iterations allows IMP to avoid layer-collapse.



(a) Iteration is needed to avoid layer-collapse



(b) IMP obtains conservation by training

Figure 5: **How IMP avoids layer collapse.** (a) Multiple iterations of training-pruning cycles is needed to prevent IMP from suffering layer-collapse. (b) The average square magnitude scores per layer, originally at initialization (blue), converge through training towards a linear relationship with the inverse layer size after training (pink), suggesting layer-wise conservation. All data is from a VGG-19 model trained on CIFAR-10.

Gradient descent encourages conservation. To better understand the dynamics of the IMP algorithm during training, we will consider a differentiable score $\mathcal{S}(\theta_i) = \frac{1}{2}\theta_i^2$ algorithmically equivalent to the magnitude score. Consider these scores throughout training with gradient descent on a loss function $\mathcal{L}$ using an infinitesimal step size (i.e. gradient flow). In this setting, the temporal derivative of the parameters is equivalent to $\frac{d\theta}{dt} = -\frac{\partial \mathcal{L}}{\partial \theta}$, and thus the temporal derivative of the score is

$$\frac{d}{dt}\frac{1}{2}\theta_i^2 = \frac{d\theta_i}{dt} \odot \theta_i = -\frac{\partial \mathcal{L}}{\partial \theta_i} \odot \theta_i. \tag{4}$$

Surprisingly, this is a form of synaptic saliency and thus the neuron-wise and layer-wise conservation laws from Sec. 4 apply. In particular, this implies that for any two layers l and k of a simple, fully connected network, then $\frac{d}{dt}||W^{[l]}||_F^2 = \frac{d}{dt}||W^{[k]}||_F^2$. This invariance has been noticed before by [45] as a form of implicit regularization and used to explain the empirical phenomenon that trained multi-layer models can have similar layer-wise magnitudes. In the context of pruning, this phenomenon implies that gradient descent training, with a small enough learning rate, encourages the squared magnitude scores to converge to an approximate layer-wise conservation, as shown in Fig. 5b.

Conservation and iterative pruning avoids layer-collapse. As explained in section 4, conservation alone leads to layer-collapse by assigning parameters in the largest layers with lower scores relative to parameters in smaller layers. However, if conservation is coupled with iterative pruning, then when the largest layer is pruned, becoming smaller, then in subsequent iterations the remaining parameters of this layer will be assigned higher relative scores. With sufficient iterations, conservation coupled with iteration leads to a self-balancing pruning strategy allowing IMP to avoid layer-collapse. This insight on the importance of conservation and iteration applies more broadly to other algorithms with exact or approximate conservation properties (e.g. Skeletonization, SNIP, and GraSP as demonstrated in Sec. 3). Indeed, very recent work empirically confirms that iteration improves the performance of SNIP [46].

6 A data-agnostic algorithm satisfying Maximal Critical Compression

In the previous section we identified two key ingredients of IMP's ability to avoid layer-collapse: (i) approximate layer-wise *conservation* of the pruning scores, and (ii) the *iterative* re-evaluation of these scores. While these properties allow the IMP algorithm to identify high performing and highly sparse, trainable neural networks, it requires an impractical amount of computation to obtain them. Thus, we aim to construct a more efficient pruning algorithm while still inheriting the key aspects of IMP's success. So what are the essential ingredients for a pruning algorithm to avoid layer-collapse and provably attain Maximal Critical Compression? We prove the following theorem in Appendix 10.

Theorem 3. ***Iterative, positive, conservative scoring achieves Maximal Critical Compression.*** *If a pruning algorithm, with global-masking, assigns positive scores that respect layer-wise conservation and if the algorithm re-evaluates the scores every time a parameter is pruned, then the algorithm satisfies the Maximal Critical Compression axiom.*

The Iterative Synaptic Flow Pruning (SynFlow) algorithm

Theorem 3 directly motivates the design of our novel pruning algorithm, SynFlow, that provably reaches Maximal Critical Compression. First, the necessity for iterative score evaluation discourages algorithms that involve backpropagation on batches of data, and instead motivates the development of an efficient data-independent scoring procedure. Second, positivity and conservation motives the construction of a loss function that yields positive synaptic saliency scores. We combine these insights to introduce a new loss function (where $\mathbb{1}$ is the all ones vector and $|\theta^{[l]}|$ is the element-wise absolute value of parameters in the l^{th} layer),

$$\mathcal{R}_{\text{SF}} = \mathbb{1}^T \left(\prod_{l=1}^{L} |\theta^{[l]}| \right) \mathbb{1} \tag{5}$$

that yields the positive, synaptic saliency scores ($\frac{\partial \mathcal{R}_{\text{SF}}}{\partial \theta} \odot \theta$) we term Synaptic Flow. For a simple, fully connected network (i.e. $f(x) = W^{[N]} \dots W^{[1]} x$), we can factor the Synaptic Flow score for a parameter $w_{ij}^{[l]}$ as

$$\mathcal{S}_{\text{SF}}(w_{ij}^{[l]}) = \left[\mathbb{1}^\intercal \prod_{k=l+1}^{N} \left| W^{[k]} \right| \right]_i \left| w_{ij}^{[l]} \right| \left[\prod_{k=1}^{l-1} \left| W^{[k]} \right| \mathbb{1} \right]_j . \tag{6}$$

This perspective demonstrates that Synaptic Flow score is a generalization of magnitude score ($|w_{ij}^{[l]}|$), where the scores consider the product of synaptic strengths flowing through each parameter, taking the inter-layer interactions of parameters into account. We use the Synaptic Flow score in the Iterative Synaptic Flow Pruning (SynFlow) algorithm summarized in the pseudocode below.

Algorithm 1: Iterative Synaptic Flow Pruning (SynFlow).

Input: network $f(x; \theta_0)$, compression ratio ρ, iteration steps n

1: $\mu = \mathbb{1}$; ▷`Initialize binary mask`

for k *in* $[1, \dots, n]$ **do**

  2: $\theta_\mu \leftarrow \mu \odot \theta_0$; ▷`Mask parameters`

  3: $\mathcal{R} \leftarrow \mathbb{1}^T \left(\prod_{l=1}^{L} |\theta_\mu^{[l]}| \right) \mathbb{1}$; ▷`Evaluate SynFlow objective`

  4: $\mathcal{S} \leftarrow \frac{\partial \mathcal{R}}{\partial \theta_\mu} \odot \theta_\mu$; ▷`Compute SynFlow score`

  5: $\tau \leftarrow (1 - \rho^{-k/n})$ percentile of $\mathcal{S}$; ▷`Find threshold`

  6: $\mu \leftarrow (\tau < \mathcal{S})$; ▷`Update mask`

end

7: $f(x; \mu \odot \theta_0)$; ▷`Return masked network`

Given a network $f(x; \theta_0)$ and specified compression ratio ρ, the SynFlow algorithm requires only one additional hyperparameter, the number of pruning iterations n. We demonstrate in Appendix 11, that an exponential pruning schedule ($\rho^{-k/n}$) with $n = 100$ pruning iterations essentially prevents layer-collapse whenever avoidable (Fig. 1), while remaining computationally feasible, even for large networks.

7 Experiments

We empirically benchmark the performance of our algorithm, SynFlow (red), against the baselines random pruning and magnitude pruning, as well as the state-of-the-art algorithms SNIP [13] and GraSP [14]. In Fig. 6, we test the five algorithms on 12 distinct combinations of modern architectures (VGG-11, VGG-16, ResNet-18, WideResNet-18) and datasets (CIFAR-10, CIFAR-100, Tiny ImageNet) over an exponential sweep of compression ratios (10^{α} for $\alpha = [0, 0.25, \ldots, 2.75, 3]$). See Appendix 12 for more details and hyperparameters of the experiments. Consistently, SynFlow outperforms the other algorithms in the high compression regime ($10^{1.5} < \rho$) and demonstrates significantly more stability, as indicated by its tight intervals. Furthermore, SynFlow is the only algorithm that reliably shows better performance to the random pruning baseline: SNIP and GraSP perform significantly worse than random pruning with ResNet-18 and WideResNet-18 trained on Tiny ImageNet. SynFlow is also quite competitive in the low compression regime ($\rho < 10^{1.5}$). Although magnitude pruning can partially outperform SynFlow in this regime with models trained on Tiny ImageNet, it suffers from catastrophic layer-collapse as indicated by the sharp drops in accuracy.

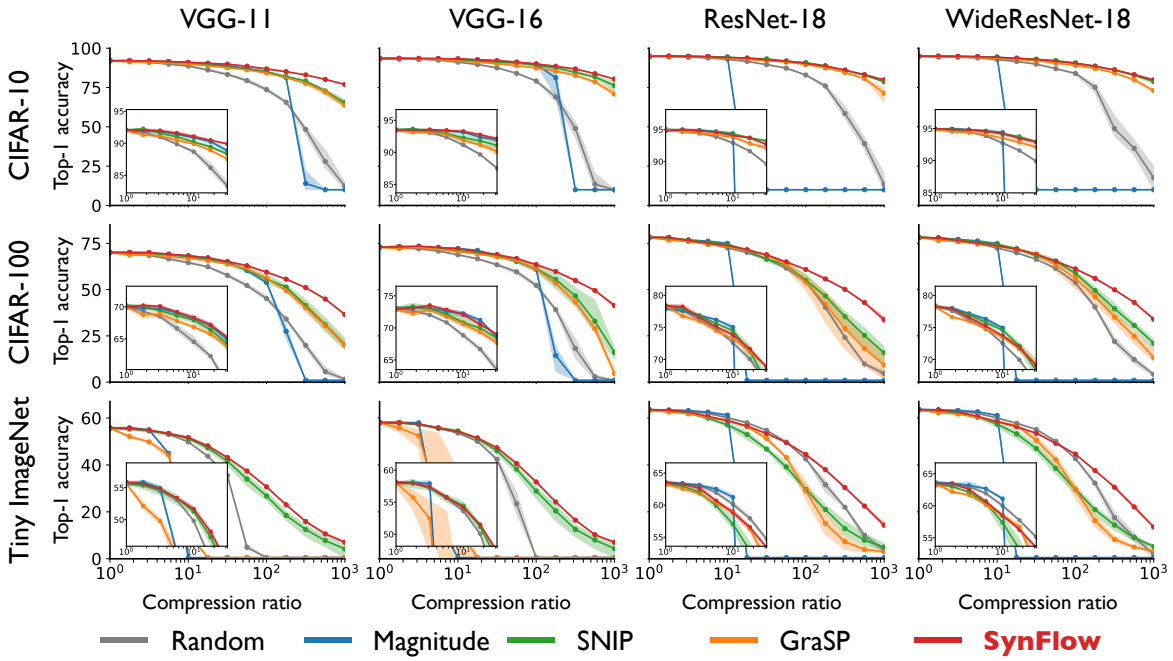


Figure 6: **SynFlow consistently outperforms other pruning methods.** Top-1 test accuracy as a function of different compression ratios over 12 distinct combinations of models and datasets. We performed three runs with the same hyperparameter conditions and different random seeds. The solid line represents the mean, the shaded region represents area between minimum and maximum performance of the three runs.

8 Conclusion

In this paper, we developed a unifying theoretical framework that explains why existing single-shot pruning algorithms at initialization suffer from layer-collapse. We applied our framework to elucidate how iterative magnitude pruning [10] overcomes layer-collapse to identify winning lottery tickets at initialization. Building on the theory, we designed a new data-agnostic pruning algorithm, SynFlow, that provably avoids layer-collapse and reaches Maximal Critical Compression. Finally, we empirically confirmed that our SynFlow algorithm consistently performs better than existing algorithms across 12 distinct combinations of models and datasets, despite the fact that our algorithm is data-agnostic and requires no pre-training. Promising future directions for this work are to (i) explore a larger space of potential pruning algorithms that satisfy Maximal Critical Compression, (ii) harness SynFlow as an efficient way to compute appropriate per-layer compression ratios to combine with existing scoring metrics, and (iii) incorporate pruning as a part of neural network initialization schemes. Overall, our data-agnostic pruning algorithm challenges the existing paradigm that data must be used to quantify which synapses of a neural network are important.

9 Acknowledgements

We thank Jonathan M. Bloom, Weihua Hu, Javier Sagastuy-Brena, Chengxu Zhuang, and members of the Stanford Neuroscience and Artificial Intelligence Laboratory for helpful discussions. We thank the Stanford Data Science Scholars program (DK), the Burroughs Wellcome, Simons and James S. McDonnell foundations, and an NSF career award (SG) for support.

References

[1] Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pages 107–115, 1989.

[2] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[3] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[4] Steven A Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600, 1989.

[5] Russell Reed. Pruning algorithms-a survey. *IEEE transactions on Neural Networks*, 4(5):740–747, 1993.

[6] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[7] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[8] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 254–263. PMLR, 2018.

[9] Hidenori Tanaka, Aran Nayebi, Niru Maheswaranathan, Lane McIntosh, Stephen Baccus, and Surya Ganguli. From deep learning to mechanistic understanding in neuroscience: the structure of retinal prediction. In *Advances in Neural Information Processing Systems*, pages 8535–8545, 2019.

[10] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[11] Jonathan Frankle, G Karolina Dziugaite, DM Roy, and M Carbin. Stabilizing the lottery ticket hypothesis. *arXiv, page*.

[12] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, pages 4933–4943, 2019.

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

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

[15] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[16] 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.

[17] Ameya Prabhu, Girish Varma, and Anoop Namboodiri. Deep expander networks: Efficient deep networks from graph theory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 20–35, 2018.

[18] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014. doi: http://dx.doi.org/10.5244/C.28.88.

[19] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450, 2015.

[20] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

[21] 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.

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

[23] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

[24] Sejun Park*, Jaeho Lee*, Sangwoo Mo, and Jinwoo Shin. Lookahead: A far-sighted alternative of magnitude-based pruning. In *International Conference on Learning Representations*, 2020.

[25] Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242, 1990.

[26] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

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

[28] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems*, pages 1379–1387, 2016.

[29] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[30] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

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

[32] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2020.

[33] 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 *International Conference on Learning Representations*, 2020.

[34] Wenyuan Zeng and Raquel Urtasun. Mlprune: Multi-layer pruning for automated neural network compression. 2018.

[35] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4646–4655. PMLR, 2019.

[36] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[38] Kedar Dhamdhere, Mukund Sundararajan, and Qiqi Yan. How important is a neuron. In *International Conference on Learning Representations*, 2019.

[39] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7), 2015.

[40] Seul-Ki Yeom, Philipp Seegerer, Sebastian Lapuschkin, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Pruning by explaining: A novel criterion for deep neural network pruning. *arXiv preprint arXiv:1912.08881*, 2019.

[41] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pages 3592–3602, 2019.

[42] 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 *International Conference on Learning Representations*, 2020.

[43] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. *arXiv preprint arXiv:1912.05671*, 2019.

[44] Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S. Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *International Conference on Learning Representations*, 2020.

[45] Simon S Du, Wei Hu, and Jason D Lee. Algorithmic regularization in learning deep homogeneous models: Layers are automatically balanced. In *Advances in Neural Information Processing Systems*, pages 384–395, 2018.

[46] Stijn Verdenius, Maarten Stol, and Patrick Forré. Pruning via iterative ranking of sensitivity statistics. *arXiv preprint arXiv:2006.00896*, 2020.

Appendix

10 Proofs

We provide a proof for Theorem 2 which we rewrite below.

Theorem 2. ***Network-wise Conservation of Synaptic Saliency.*** *The sum of the synaptic saliency across any set of parameters that exactly separates the input neurons x from the output neurons y of a feedforward neural network with homogenous activation functions equals $\langle \frac{\partial \mathcal{R}}{\partial x}, x \rangle = \langle \frac{\partial \mathcal{R}}{\partial y}, y \rangle$.*

Proof. We begin by defining the set of neurons (V) and the set of prunable parameters (E) for a neural network.

Consider a subset of the neurons $S \subset V$, such that all output neurons $y_c \in S$ and all input neurons $x_i \in V \backslash S$. Consider the set of parameters cut by this partition

$$C(S) = \{\theta_{uv} \in E : u \in S, v \in V \backslash S\}. \tag{7}$$

By theorem 1, we know that that sum of the synaptic saliency over $C(S)$ is equal to the sum of the synaptic saliency over the set of parameters adjacent to $C(S)$ and between neurons in S, $\{\theta_{tu} \in E : t \in S, u \in \partial S\}$. Continuing this argument, then eventually we get that this sum must be equal to the sum of the synaptic saliency over the set of parameters incident to the output neurons y, which is

$$\sum_{c,d} \frac{\partial \mathcal{R}}{\partial \theta_{cd}} \theta_{cd} = \sum_{c,d} \frac{\partial \mathcal{R}}{\partial y_c} \phi(z_d) \theta_{cd} = \sum_{c} \frac{\partial \mathcal{R}}{\partial y_c} \left(\sum_{d} \theta_{cd} \phi(z_d) \right) = \sum_{c} \frac{\partial \mathcal{R}}{\partial y_c} y_c = \langle \frac{\partial \mathcal{R}}{\partial y}, y \rangle. \tag{8}$$

We can repeat this argument iterating through the set $V \backslash S$ till we reach the input neurons x to show that this sum is also equal to $\langle \frac{\partial \mathcal{R}}{\partial x}, x \rangle$. □

We provide a proof for Theorem 3 which we rewrite below.

Theorem 3. ***Iterative, positive, conservative scoring achieves Maximal Critical Compression.*** *If a pruning algorithm, with global-masking, assigns positive scores that respect layer-wise conservation and if the algorithm re-evaluates the scores every time a parameter is pruned, then the algorithm satisfies the Maximal Critical Compression axiom.*

Proof. We prove this theorem by contradiction. Assume that a pruning algorithm with global-masking and iterative, positive, conservative scoring does not satisfy the Maximal Critical Compression axiom. This implies that at some iteration, the algorithm will prune the last parameter in a layer (layer l), despite there existing more than one parameters ($N^{[k]} > 1$) in another layer (layer k). Because the algorithm uses global-masking, then the score for the last parameter in layer l, $\mathcal{S}^{[l]}$, is less than or equal to the scores for each parameter, $\mathcal{S}_i^{[k]}$, in layer k:

$$\mathcal{S}^{[l]} \leq \mathcal{S}_i^{[k]}. \tag{9}$$

Because the scores respect a layer-wise conservation, then $\mathcal{S}^{[l]} = \sum_{i=1}^{N^{[k]}} \mathcal{S}_i^{[k]}$. This implies, by the positivity of the scores and because $N^{[k]} > 1$, that for all i,

$$\mathcal{S}^{[l]} > \mathcal{S}_i^{[k]}. \tag{10}$$

This is a contradiction to the previous inequality. □

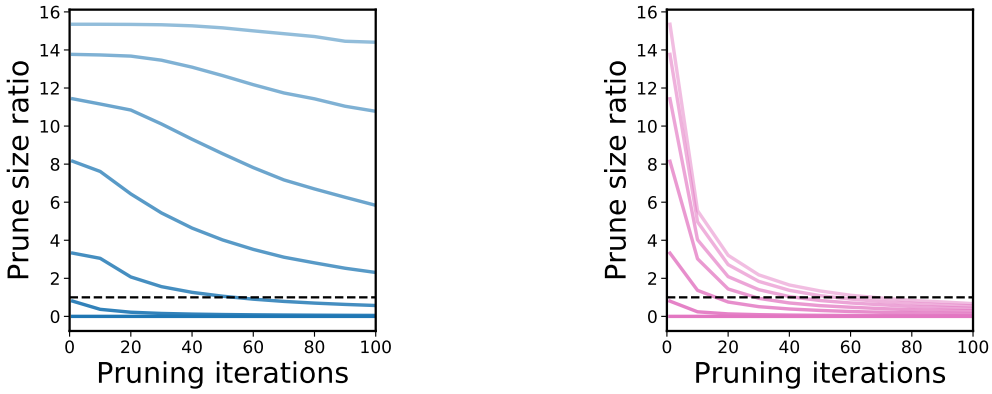
11 Hyperparameters choices for the SynFlow algorithm

Theorem 3 required that an algorithm re-evaluates the scores every time a parameter is pruned. However, theorem 2 provides a theoretical insight to drastically reduce the number of iterations needed to practically attain Maximal Critical Compression. We now introduce a modification to theorem 3 that motivates practical hyperparameter choices used in the SynFlow algorithm.

Theorem 4. ***Achieving Maximal Critical Compression practically.*** *If a pruning algorithm, with global-masking, assigns positive scores that respect layer-wise conservation and if the prune size, the total score for the parameters pruned at any iteration, is strictly less than the cut size, the total score for an entire layer, whenever possible, then the algorithm satisfies the Maximal Critical Compression axiom.*

Proof. We prove this theorem by contradiction. Assume there is an iterative pruning algorithm that uses positive, layer-wise conserved scores and maintains that the prune size at any iteration is less than the cut size whenever possible, but doesn't satisfy the Maximal Critical Compression axiom. At some iteration the algorithm will prune a set of parameters containing a subset separating the input neurons from the output neurons, despite there existing a set of the same cardinality that does not lead to layer-collapse. By theorem 2, the total score for the separating subset is $\langle \frac{\partial \mathcal{R}}{\partial y}, y \rangle$, which implies by the positivity of the scores, that the total prune size is at least $\langle \frac{\partial \mathcal{R}}{\partial y}, y \rangle$. This contradicts the assumption that the algorithm maintains that the prune size at any iteration is always strictly less than the cut size whenever possible. □

Motivated by Theorem 4, we can now choose a practical, yet effective, number of pruning iteration (n) and schedule for the compression ratios (ρ_k) applied at each iteration (k) for the SynFlow algorithm. Two natural candidates for a compression schedule would be either linear ($\rho_k = \frac{k}{n}\rho$) or exponential ($\rho_k = \rho^{\frac{k}{n}}$). Empirically we find that the SynFlow algorithm with 100 pruning iterations and an exponential compression schedule satisfies the conditions of theorem 4 over a reasonable range of compression ratios (10^n for $0 \leq n \leq 3$), as shown in Fig. 7b. This is not true if we use a linear schedule for the compression ratios, as shown in Fig. 7a. Interestingly, Iterative Magnitude Pruning also uses an exponential compression schedule, but does not provide a thorough explanation for this hyperparameter choice [10].



(a) Linear compression schedule

(b) Exponential compression schedule

Figure 7: **Choosing the number of pruning iterations and compression schedule for SynFlow.** Maximum ratio of prune size with cut size for increasing number of pruning iterations for SynFlow with a linear (left) or exponential (right) compression schedule. Higher transparency represents higher compression ratios. The black dotted line represents the maximal prune size ratio that can be obtained while still satisfying the conditions of theorem 4. All data is from a VGG-19 model at initialization using ImageNet.

Potential numerical instability. The SynFlow algorithm involves computing the SynFlow objective, $\mathcal{R}_{\text{SF}} = \mathbb{1}^T \left(\prod_{l=1}^{L} |\theta^{[l]}| \right) \mathbb{1}$, whose singular values may vanish or explode exponentially with depth L. This may lead to potential numerical instability for very deep networks, although we did not observe this for the models presented in this paper. One way to address this potential challenge would be to appropriately scale network parameters at each layer to maintain stability. Because the SynFlow algorithm is scale invariant at each layer $\theta^{[l]}$, this modification will not effect the performance of the algorithm.

12 Experimental details

An open source version of our code and the data used to generate all the figures in this paper are available at `github.com/ganguli-lab/Synaptic-Flow`.

12.1 Pruning algorithms

All pruning algorithms we considered in our experiments use the following two steps: (i) scoring parameters, and (ii) masking parameters globally across the network with the lowest scores. Here we describe details of how we computed scores used in each of the pruning algorithms.

Random: We sampled independently from a standard Gaussian.

Magnitude: We computed the absolute value of the parameters.

SNIP: We computed the score $\left|\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta\right|$ using a random subset of the training dataset with a size ten times the number of classes, namely 100 for CIFAR-10, 1000 for CIFAR-100, 2000 for Tiny ImageNet, and 10000 for ImageNet. The score was computed on a batch of size 256 for CIFAR-10/100, 64 for Tiny ImageNet, and 16 for ImageNet, then summed across batches to obtain the score used for pruning.

GraSP: We computed the score $-\left(H \frac{\partial \mathcal{L}}{\partial \theta}\right) \odot \theta$ using a random subset of the training dataset with a size ten times the number of classes, namely 100 for CIFAR-10, 1000 for CIFAR-100, 2000 for Tiny ImageNet, and 10000 for ImageNet. The score was computed on a batch of size 256 for CIFAR-10/100, 64 for Tiny ImageNet, and 16 for ImageNet, then summed across batches to obtain the score used for pruning.

SynFlow: We applied the pseudocode 1 with 100 pruning iterations motivated by the theoretical and empirical results discussed in Sec 11.

12.2 Model architectures

We adapted standard implementations of VGG-11 and VGG-16 from OpenLTH, and ResNet-18 and WideResNet-18 from PyTorch models. We considered all weights from convolutional and linear layers of these models as prunable parameters, but did not prune biases nor the parameters involved in batchnorm layers. For convolutional and linear layers, the weights were initialized with a Kaiming normal strategy and biases to be zero.

12.3 Training hyperparameters

Here we provide hyperparameters that we used to train the models presented in Fig. 1 and Fig. 6. These hyperparameters were chosen for the performance of the original model and were not optimized for the performance of the pruned networks.

	VGG-11		VGG-16		ResNet-18		WideResNet-18	
	CIFAR-10/100	Tiny ImageNet	CIFAR-10/100	Tiny ImageNet	CIFAR-10/100	Tiny ImageNet	CIFAR-10/100	Tiny ImageNet
Optimizer	momentum	momentum	momentum	momentum	momentum	momentum	momentum	momentum
Training Epochs	160	100	160	100	160	100	160	100
Batch Size	128	128	128	128	128	128	128	128
Learning Rate	0.1	0.01	0.1	0.01	0.01	0.01	0.01	0.01
Learning Rate Drops	60, 120	30, 60, 80	60, 120	30, 60, 80	60, 120	30, 60, 80	60, 120	30, 60, 80
Drop Factor	0.1	0.1	0.1	0.1	0.2	0.1	0.2	0.1
Weight Decay	10^{-4}	10^{-4}	10^{-4}	10^{-4}	5×10^{-4}	10^{-4}	5×10^{-4}	10^{-4}