PICKING WINNING TICKETS BEFORE TRAINING BY PRESERVING GRADIENT FLOW

Anonymous authors
Paper under double-blind review

ABSTRACT

Overparameterization has been shown to benefit both the optimization and generalization of neural networks, but large networks are resource hungry at both training and test time. Network pruning can reduce test-time resource requirements, but is typically applied to trained networks and therefore cannot avoid the expensive training process. We aim to prune networks at initialization, thereby saving resources at training time as well. Specifically, we argue that efficient training requires preserving the gradient flow through the network. This leads to a simple but effective pruning criterion we term Gradient Signal Preservation (GraSP). We empirically investigate the effectiveness of the proposed method with extensive experiments on CIFAR-10, CIFAR-100, Tiny-ImageNet and ImageNet, using VGGNet and ResNet architectures. Our method can prune 80% of the weights of a VGG-16 network on ImageNet at initialization, with only a 1.6% drop in top-1 accuracy. Moreover, our method achieves significantly better performance than the baseline at extreme sparsity levels.

1 INTRODUCTION

Deep neural networks exhibit good optimization and generalization performance in the overparameterized regime (Zhang et al., 2016; Neyshabur et al., 2019; Arora et al., 2019; Zhang et al., 2019b), but both training and inference for large networks are computationally expensive. Network pruning (LeCun et al., 1990; Hassibi et al., 1993; Han et al., 2015b; Dong et al., 2017; Zeng & Urtasun, 2019; Wang et al., 2019) has been shown to reduce the *test-time* resource requirements with minimal performance degradation. However, as the pruning is typically done to a trained network, these methods don't save resources at *training time*. Moreover, it has been argued that it is hard to train sparse architectures from scratch while maintaining comparable performance to their dense counterparts (Han et al., 2015a; Li et al., 2016). Therefore, we ask: can we prune a network prior to training, so that we can improve computational efficiency at training time?

Recently, Frankle & Carbin (2019) shed light on this problem by proposing the Lottery Ticket Hypothesis (LTH), namely that there exist sparse, trainable sub-networks (called "winning tickets") within the larger network. They identify the winning tickets by taking a *pre-trained* network and removing connections with weights smaller than a pre-specificed threshold. They then reset the remaining weights to their initial values, and retrain the sub-network from scratch. Hence, they showed that the pre-trained weights are not necessary, only the pruned architecture and the corresponding initial weight values. Nevertheless, like traditional pruning methods, the LTH approach still requires training the full-sized network for identifying the sparse sub-networks.

Can we identify sparse, trainable sub-networks at initialization? This would allow us to exploit sparse computation with specified hardware for saving computation cost. As shown by Dey et al. (2019), with pre-specified sparsity one can achieve 5x efficiency for both training and testing. At the first glance, a randomly initialized network seems to provide little information that we can use to judge the importance of individual connections, since the choice would seem to depend on complicated training dynamics. However, recent work suggests this goal is attainable. Lee et al. (2018) proposed the first algorithm for pruning at initialization time: Single-shot Network Pruning (SNIP), which uses a *connection sensitivity* criterion to prune weights with both small magnitude and small gradients. Their empirical results are promising in the sense that they can find sparse, trainable sub-networks at the initialization. However, connection sensitivity is sub-optimal as a criterion because the gradient of

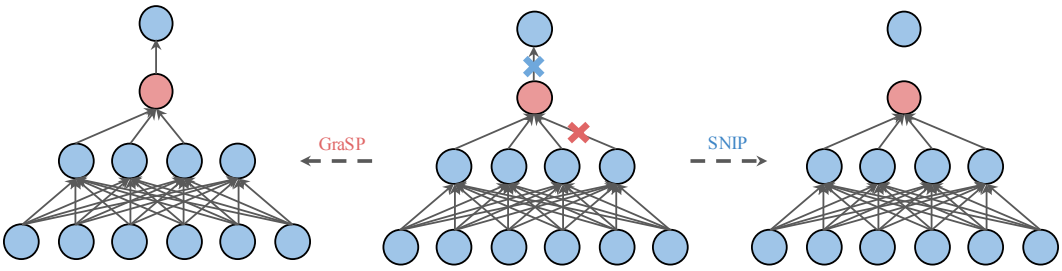

Figure 1: A high-level abstraction of the pruning choice (one weight) of GraSP (left) and SNIP (right) with a linear network, when assuming the output of red neuron is 0. SNIP will choose to prune the weight that takes 0 as input, which will block the entire information flow or gradient flow after pruning. GraSP will choose not to prune that weight, since the pruning of it will result in a significant decrease of the gradient flow.

each weight might change dramatically after pruning due to complicated interactions between weights. Since SNIP only considers the gradient for one weight in isolation, it could remove connections that are important to the flow of information through the network. This effect is illustrated in Figure 1. Practically, we find that this blocking of information flow manifests as a reduction in the norm of the gradient (*e.g.* In Figure 1, the network pruned by SNIP has a zero gradient).

Therefore, we aim to prune connections in a way that accounts for their role in the network's gradient flow. Specifically, we take the gradient norm *after pruning* as our criterion, and prune those weights whose removal will result in least decrease in the gradient norm after pruning. Because we rely on preserving the gradient flow to prune the network, we name our method Gradient Signal Preservation (GraSP). Our approach is easy to implement and conceptually simple. Moreover, the recently introduced Neural Tangent Kernel (NTK) (Jacot et al., 2018) provides tools for studying the learning dynamics in the output space. Building on the analysis of Arora et al. (2019), we show that our pruning criterion tends to keep those weights which will be beneficial for optimization. We evaluate GraSP on CIFAR-10, CIFAR-100 (Krizhevsky, 2009), Tiny-ImageNet and ImageNet (Deng et al., 2009) with modern neural networks, such as VGGNet (Simonyan & Zisserman, 2014) and ResNet (He et al., 2016). GraSP significantly outperforms SNIP in the extreme sparsity regime.

2 Related Work and Background

In this section, we review the literature on neural network pruning including pruning after training, during training and before training. Lastly, we also review the Neural Tangent Kernel (NTK) (Jacot et al., 2018), which builds up the foundation for justifying our method in Section 4.

2.1 Network Pruning

After training. Most pruning algorithms (LeCun et al., 1990; Hassibi et al., 1993; Dong et al., 2017; Han et al., 2015b; Li et al., 2016; Molchanov et al., 2016) operate on a pre-trained network. The main idea is to identify those weights which are most redundant, and whose removal will therefore least degrade the performance. Magnitude based pruning algorithms (Han et al., 2015b;a) remove those weights which are smaller than a threshold, which may incorrectly measure the importance of each weight. In contrast, Hessian-based pruning algorithms (LeCun et al., 1990; Hassibi et al., 1993) compute the importance of each weight by measuring how its removal will affect the loss. More recently, Wang et al. (2019) proposed a novel network reparameterization based on the Kronecker-factored Eigenbasis for further boosting the performance of Hessian-based methods. However, all the aforementioned methods require a pre-trained network, and therefore aren't applicable at initialization.

During training. There are also some works which attempt to incorporate pruning into the training procedure itself. Srinivas & Babu (2016) proposed generalized dropout, allowing for tuning the individual dropout rates during training, which can result in a sparse network after training. Louizos et al. (2018) proposed a method for dealing with discontinuity in training L_0 norm regularized networks in order to obtain sparse networks. Both methods require roughly the same computational cost as training the full network. Other works, such as Bellec et al. (2018); Mocanu et al. (2018); Mostafa & Wang (2019); Dettmers & Zettlemoyer (2019), follow the prune-redistribute-regrowth

cycle for pruning. Among them, Dettmers & Zettlemoyer (2019) proposed sparse momentum algorithm, which dynamically determines the sparse mask based on the mean momentum magnitude during training. However, their method requires to maintain the momentum of *all* the weights during training, and thus does not save the memory. These works not only change the standard training procedure, but also hard to enjoy the practical accelerations from hardware (Dey et al., 2019).

Before training. Pruning at initialization is more challenging because we need to account for the effect on the training dynamics when removing each weight. There have been several attempts to conduct pruning before training. Frankle & Carbin (2019); Frankle et al. (2019) proposed and validated the Lottery Ticket Hypothesis (LTH) that the network structure found by traditional pruning algorithms and the corresponding initialization are enough for training the sub-network from scratch. Lee et al. (2018) proposed the SNIP algorithm, which was the first attempt to directly identify trainable and sparse sub-networks at initialization time. Their method was based on *connection sensitivity*, which aims to preserve the loss after pruning, and achieved impressive results. Concurrently to our work, Lee et al. (2019b) studied the pruning problem from a signal propagation perspective, and proposed to use an orthogonal initialization for ensuring faithful signal propagations. Though sharing the same spirit as our GraSP algorithm, they focus on initialization scheme which is orthogonal to specific pruning method.

2.2 Neural Tangent Kernel and Convergence Analysis

Jacot et al. (2018) analyzed the dynamics of neural net training by directly analyzing the evolution of the network's predictions in output space. Let $\mathcal{L}$ denote the cost function, $\mathcal{X}$ the set of all training samples, $\mathcal{Z} = f(\mathcal{X}; \boldsymbol{\theta}) \in \mathbb{R}^{nk \times 1}$ the outputs of the neural network, and k and n the output space dimension and the number of training examples. For a step of gradient descent, the change to the network's predictions can be approximated with a first-order Taylor approximation:

$$f(\mathcal{X}; \boldsymbol{\theta}_{t+1}) = f(\mathcal{X}; \boldsymbol{\theta}_t) - \eta \boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X}) \nabla_{\mathcal{Z}} \mathcal{L}, \tag{1}$$

where the matrix $\boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X})$ is the *Neural Tangent Kernel (NTK)* at time step t:

$$\boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X}) = \nabla_{\boldsymbol{\theta}} f(\mathcal{X}; \boldsymbol{\theta}_t) \nabla_{\boldsymbol{\theta}} f(\mathcal{X}; \boldsymbol{\theta}_t)^\top \in \mathbb{R}^{nk \times nk} \tag{2}$$

where $\nabla_{\boldsymbol{\theta}} f(\mathcal{X}; \boldsymbol{\theta})$ denotes the network Jacobian over the whole training set. Jacot et al. (2018) showed that for infinitely wide networks, with proper initialization, the NTK exactly captures the output space dynamics throughout training, which means NTK reveals the final state of the network at initialization without any training. In particular, $\boldsymbol{\Theta}_t(\mathcal{X}, \mathcal{X})$ is constant. Arora et al. (2019) used the NTK to analyze optimization and generalization phenomena, showing that under the assumptions of constant NTK and squared error loss, the training dynamics can be analyzed in closed form:

$$\|\mathcal{Y} - f(\mathcal{X}; \boldsymbol{\theta}_t)\|_2 = \sqrt{\sum_{i=1}^{n} (1 - \eta \lambda_i)^{2t} (\mathbf{u}_i^\top \mathcal{Y})^2} \pm \epsilon \tag{3}$$

where $\mathcal{Y} \in \mathbb{R}^{nk \times 1}$ is all the targets, $\boldsymbol{\Theta} = \mathbf{U} \boldsymbol{\Lambda} \mathbf{U}^\top = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^\top$ is the eigendecomposition, and ϵ is a bounded error term. Although the constant NTK assumption holds only in the infinite width limit, Lee et al. (2019a) found close empirical agreement between the NTK dynamics and the true dynamics for wide but practical networks, such as wide ResNet architectures (Zagoruyko & Komodakis, 2016). Therefore, if running the training procedure itself is too expensive (as in our setting), one ought to instead be able to achieve a rough approximation to the training dynamics using the NTK.

3 Revisiting Single-Shot Network Pruning (SNIP)

Single-shot network pruning was introduced by Lee et al. (2018), who used the term to refer both to the general problem setting and to their specific algorithm. To avoid ambiguity, we refer to the general problem of pruning before training as *foresight pruning*. For completeness, we first revisit the formulation of foresight pruning, and then point out issues of SNIP for motivating our method.

Problem Formulation. Suppose we have a neural network f parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$, and our objective is to minimize the empirical risk $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\mathbf{x}, y) \sim \mathcal{D}} [\ell(f(\mathbf{x}; \boldsymbol{\theta}), y)]$ given a training set

Algorithm 1 Algorithm for GraSP: Foresight Pruning through Gradient Signal Preservation.

Require: pruning ratio p, training data $\mathcal{D}$, network f with initial parameters $\boldsymbol{\theta}_0$
1: $\mathcal{D}_{\text{mini}} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{b} \sim \mathcal{D}$ ▷ Sample a mini-batch from the training set $\mathcal{D}$
2: Compute the Hessian and gradient product as $\mathbf{Hg}$
3: $\mathbf{S}(-\boldsymbol{\theta}_0) = -\boldsymbol{\theta}_0 \odot \mathbf{Hg}$ ▷ Compute the importance of each weight
4: Compute p_{th} percentile of $\mathbf{S}(-\boldsymbol{\theta}_0)$ as τ
5: $\mathbf{m} = \mathbf{S}(-\boldsymbol{\theta}_0) < \tau$ ▷ Remove the weight if larger than the threshold
6: Train the network $f_{\mathbf{m}\odot\boldsymbol{\theta}}$ on $\mathcal{D}$ until convergence.

$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$. Then, the foresight pruning problem can be formulated as:

$$\min_{\mathbf{m}\in\{0,1\}^d} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\ell\left(f\left(\mathbf{x}; \mathcal{A}(\mathbf{m}, \boldsymbol{\theta}_0)\right), y\right)\right] \quad \text{s.t.} \quad \|\mathbf{m}\|_0/d = 1 - p \tag{4}$$

where $\lceil p \cdot d \rceil$ is the number of weights to be removed, and $\mathcal{A}$ is a known training algorithm (e.g. SGD), which takes the mask $\mathbf{m}$ (here we marginalize out the initial weights $\boldsymbol{\theta}_0$ for simplicity), and returns the trained weights. Since globally minimizing Eqn. 4 is intractable, we are instead interested in heuristics that result in good practical performance.

Revisiting SNIP. SNIP (Lee et al., 2018) was the first algorithm proposed for foresight pruning, and it leverages the notion of *connection sensitivity* to remove unimportant connections (weights). They define this in terms of how removing a single weight θ_q in isolation will affect the loss:

$$S(\theta_q) = \lim_{\epsilon\to 0} \left| \frac{\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_0 + \epsilon\boldsymbol{\delta}_q)}{\epsilon} \right| = \left| \theta_q \frac{\partial \mathcal{L}}{\partial \theta_q} \right| \tag{5}$$

where θ_q is the q_{th} element of $\boldsymbol{\theta}_0$, and $\boldsymbol{\delta}_q$ is a one-hot vector with only q_{th} element equals to θ_q. Essentially, SNIP aims at preserving the loss of the pruned network almost the same as that of the original network.

Preserving the loss value motivated several classic methods for pruning a *trained network*, such as optimal brain damage (LeCun et al., 1990) and optimal brain surgery (Hassibi et al., 1993). While the motivation for loss preservation of a trained network is clear, it is less clear why this is a good criterion for foresight pruning. After all, at initialization, the loss is no better than chance. We argue that at the *beginning* of training, it is more important to preserve the training dynamics than the loss itself. SNIP does not do this automatically, because even if removing a particular connection doesn't affect the loss, it could still block the flow of information through the network. For instance, we noticed in our experiments that SNIP with a high pruning ratio (e.g. 99%) tends to eliminate nearly all the weights in a particular layer, creating a bottleneck in the network. Therefore, we would prefer a pruning criterion which accounts for how the presence or absence of one connection influences the training of the rest of the network.

4 Gradient Signal Preservation

We now introduce and motivate our foresight pruning criterion, Gradient Signal Preservation (GraSP). To understand the problem we are trying to address, observe that the network after pruning will have fewer parameters and sparse connectivity, hindering the flow of gradients through the network and potentially slowing the optimization. This is reflected in Figure 3, which shows the reduction in gradient norm for random pruning with various pruning ratios. Moreover, the performance of the pruned networks is correspondingly worse (see Table 1).

Mathematically, a larger gradient norm indicates that, to the first order, each gradient update achieves a greater loss reduction, as characterized by the following directional derivative:

$$\Delta\mathcal{L}(\boldsymbol{\theta}) = \lim_{\epsilon\to 0} \frac{\mathcal{L}\left(\boldsymbol{\theta} + \epsilon\nabla\mathcal{L}(\boldsymbol{\theta})\right) - \mathcal{L}(\boldsymbol{\theta})}{\epsilon} = \nabla\mathcal{L}(\boldsymbol{\theta})^{\top}\nabla\mathcal{L}(\boldsymbol{\theta}) \tag{6}$$

Since we only care about the performance of the pruned network, our goal is to preserve or even increase (if possible) the gradient flow *after pruning* (*i.e.*, the gradient flow of the pruned network).

Table 1: Comparisons with Random Pruning with VGG19 and ResNet32 on CIFAR-10/100.

Dataset	CIFAR-10				CIFAR-100			
Pruning ratio	95%	98%	99%	99.5%	95%	98%	99%	99.5%
Random Pruning (VGG19)	89.47(0.5)	86.71(0.7)	82.21(0.5)	72.89(1.6)	66.36(0.3)	61.33(0.1)	55.18(0.6)	36.88(6.8)
GraSP (VGG19)	**93.04(0.2)**	**92.19(0.1)**	**91.33(0.1)**	**88.61(0.7)**	**71.23(0.1)**	**68.90(0.5)**	**66.15(0.2)**	**60.21(0.1)**
Random Pruning (ResNet32)	89.75(0.1)	85.90(0.4)	71.78(9.9)	50.08(7.0)	64.72(0.2)	50.92(0.9)	34.62(2.8)	18.51(0.43)
GraSP (ResNet32)	**91.39(0.3)**	**88.81(0.1)**	**85.43(0.5)**	**80.50(0.3)**	**66.50(0.1)**	**58.43(0.4)**	**48.73(0.3)**	**35.55(2.4)**

In order to achieve this, as adopted in LeCun et al. (1990), we cast the pruning operation as adding a perturbation $\boldsymbol{\delta}$ on the initial weights for characterizing how removing one weight will affect the gradient flow *after pruning*:

$$\begin{aligned} S\left(\boldsymbol{\delta}\right) &= \Delta\mathcal{L}(\boldsymbol{\theta}_0 + \boldsymbol{\delta}) - \underbrace{\Delta\mathcal{L}(\boldsymbol{\theta}_0)}_{\text{Const}} = 2\boldsymbol{\delta}^\top \nabla^2\mathcal{L}(\boldsymbol{\theta}_0)\nabla\mathcal{L}(\boldsymbol{\theta}_0) + \mathcal{O}(\|\boldsymbol{\delta}\|_2^2) \\ &= 2\boldsymbol{\delta}^\top \mathbf{H}\mathbf{g} + \mathcal{O}(\|\boldsymbol{\delta}\|_2^2) \end{aligned} \tag{7}$$

where $S(\boldsymbol{\delta})$ is a function of $\boldsymbol{\delta}$, and it denotes the change in (6) to a $\boldsymbol{\delta}$-perturbation on the initial weight $\boldsymbol{\theta}_0$. The Hessian matrix $\mathbf{H}$ captures the correlations between each weight, and thus modulates the effect of pruning on the remained weights. When $\mathbf{H}$ is identity, the above criteria recovers SNIP up to the absolute value (recall the criteria of SNIP is $|\boldsymbol{\delta}^\top \mathbf{g}|$). However, it has been observed that different weights are highly coupled (Hassibi et al., 1993), indicating that the Hessian $\mathbf{H}$ is far from identity.

In practice, we adopt the equation defined above (*i.e.*, eqn. (7)) as the measure of the importance of each weight. Specifically, if $S(\boldsymbol{\delta})$ is negative, then removing the corresponding weights will reduce the gradient flow, otherwise it will not. Hence, we will prefer removing those weights whose removal will not decrease the gradient flow first. For each weight, the importance can be computed in the following way:

$$S(-\theta_q) = -\theta_q \cdot [\mathbf{H}\mathbf{g}]_q \tag{8}$$

For a given pruning ratio p, we can obtain the resulting pruning mask by computing the pruning criteria in a single pass, sort them accordingly and then remove the top $p\%$ of the weights (see Algorithm 1). Roughly speaking, GraSP takes change in gradient flow after pruning into account, whereas SNIP only tries to preserve the loss after pruning, which may not perserve the gradient flow. Intuitively speaking, GraSP is accounting for how removing one weight will affect the gradient of each other weight. Moreover, our criteria is easy to implement, and can be computed by backward twice without explicitly computing the Hessian (Pearlmutter, 1994; Schraudolph, 2002).

4.1 Understanding GraSP through Linearized Training Dynamics

The above discussion concerns only the training dynamics at initialization time. To understand the longer-term dynamics, we leverage the recently proposed Neural Tangent Kernel (NTK), which has been shown to be able to capture the training dynamics throughout training for practical networks (Lee et al., 2019a). Specifically, as we introduced in section 2.2, eqn. (3) characterizes how the training error changes throughout the training process, which only depends on time step t, training targets $\mathcal{Y}$ and the NTK $\boldsymbol{\Theta}$. Since the NTK stays almost constant for wide but practical networks (Lee et al., 2019a), *e.g.*, wide ResNet (Zagoruyko & Komodakis, 2016), we can regard that eqn. 3 only depends on t once the network is initialized. This shows that NTK captures the training dynamics throughout the training process. Therefore, if we decompose eqn. (6) in the following form:

$$\nabla\mathcal{L}(\boldsymbol{\theta})^\top \nabla\mathcal{L}(\boldsymbol{\theta}) = \nabla_{\mathcal{Z}}\mathcal{L}^\top \boldsymbol{\Theta}(\mathcal{X},\mathcal{X})\nabla_{\mathcal{Z}}\mathcal{L} = (\mathbf{U}^\top\nabla_{\mathcal{Z}}\mathcal{L})^\top \boldsymbol{\Lambda}(\mathbf{U}^\top\nabla_{\mathcal{Z}}\mathcal{L}) = \sum_{i=1}^{n} \lambda_i(\mathbf{u}_i^\top \mathcal{Y})^2 \tag{9}$$

By relating it to eqn. (3), we can see that our objective is implicitly encouraging the eigenspace of $\boldsymbol{\Theta}$ to align well with the target $\mathcal{Y}$ by pruning, which controls how each gradient step changes the training error. Formally speaking, when we are able to maximize eqn. (9), it will encourage the eigenspace of the NTK distributing large eigenvalues in the direction of $\mathcal{Y}$, which in turn accelerates the decrease of the loss (Arora et al., 2019) and benefits the optimization in $\mathcal{A}$.

Table 2: Test accuracy of pruned VGG19 and ResNet32 on CIFAR-10 and CIFAR-100 datasets. The bold number is the higher one between the accuracy of GraSP and that of SNIP.

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG19 (Baseline)	94.23	-	-	74.16	-	-
OBD (LeCun et al., 1990)	93.74	93.58	93.49	73.83	71.98	67.79
MLPrune (Zeng & Urtasun, 2019)	93.83	93.69	93.49	73.79	73.07	71.69
LT (original initialization)	93.51	92.92	92.34	72.78	71.44	68.95
LT (reset to epoch 5)	93.82	93.61	93.09	74.06	72.87	70.55
SNIP (Lee et al., 2018)	**93.63±0.06**	**93.43±0.20**	92.05±0.28	**72.84±0.22**	**71.83±0.23**	58.46±1.10
GraSP	93.30±0.14	93.04±0.18	**92.19±0.12**	71.95±0.18	71.23±0.12	**68.90±0.47**
ResNet32 (Baseline)	94.80	-	-	74.64	-	-
OBD (LeCun et al., 1990)	94.17	93.29	90.31	71.96	68.73	60.65
MLPrune (Zeng & Urtasun, 2019)	94.21	93.02	89.65	72.34	67.58	59.02
LT (original initialization)	92.31	91.06	88.78	68.99	65.02	57.37
LT (reset to epoch 5)	93.97	92.46	89.18	71.43	67.28	58.95
SNIP (Lee et al., 2018)	**92.59±0.10**	91.01±0.21	87.51±0.31	68.89±0.45	65.22±0.69	54.81±1.43
GraSP	92.38±0.21	**91.39±0.25**	**88.81±0.14**	**69.24±0.24**	**66.50±0.11**	**58.43±0.43**

5 EXPERIMENTS

In this section, we conduct various experiments to validate the effectiveness of our proposed pruning algorithm in terms of the test accuracy vs. pruning ratios. We also include two traditional pruning algorithms (LeCun et al., 1990; Zeng & Urtasun, 2019), which operate on the pre-trained networks, for serving as an upper bound for foresight pruning. Besides, we study the convergence performance of the sub-networks obtained by different pruning methods for investigating the relationship between gradient norm and final performance. Lastly, we study the role of initialization and batch sizes in terms of the performance of GraSP for ablation study.

5.1 PRUNING RESULTS ON MODERN CONVNETS

To evaluate the effectiveness of GraSP on real world tasks, we test GraSP on four image classification datasets, CIFAR-10/100, Tiny-ImageNet and ImageNet, with two modern network architectures, VGGNet and ResNet[1]. For the experiments on CIFAR-10/100 and Tiny-ImageNet, we use a mini-batch with ten times of the number of classes for both GraSP and SNIP[2] according to Lee et al. (2018). The pruned network is trained with Kaiming initialization (He et al., 2015) using SGD for 160 epochs for CIFAR-10/100, and 300 epochs for Tiny-ImageNet, with an initial learning rate of 0.1 and batch size 128. The learning rate is decayed by a factor of 0.1 at $1/2$ and $3/4$ of the total number of epochs. Moreover, we run each experiment for 3 trials for obtaining more stable results. For ImageNet, we adopt the Pytorch (Paszke et al., 2017) official implementation, but we used more epochs for training according to Liu et al. (2019). Specifically, we train the pruned networks with SGD for 150 epochs, and decay the learning rate by a factor of 0.1 every 50 epochs.

We first compre GraSP against random pruning, which generates the mask randomly for a given pruning ratio. The test accuracy is reported in Table 1. We can observe GraSP clearly outperforms random pruning, and the performance gap can be up to more than 30% in terms of test accuracy. We further compare GraSP with more competitive baselines on CIFAR-10/100 and Tiny-ImageNet for pruning ratios $\{85\%, 90\%, 95\%, 98\%\}$, and the results can be referred in Table 2 and 4. When the pruning ratio is low, i.e. $85\%, 90\%$, both SNIP and GraSP can achieve very close results to the baselines, though still do not outperform pruning algorithms that operate on trained networks, as expected. However, GraSP achieves significant better results for higher pruning ratios with more complicated networks (e.g. ResNet) and datasets (e.g. CIFAR-100 and Tiny-ImageNet), showing the advantages of directly relating the pruning criteria with the gradient flow after pruning. Moreover, in most cases, we notice that either SNIP or GraSP can match or even slightly outperform LT with original initialization, which indicates that both methods can identify meaningful structures. We further experiment with the late resetting as suggested in Frankle et al. (2019) by resetting the weights

[1]For experiments on CIFAR-10/100 and Tiny-ImageNet, we adopt ResNet32 and double the number of filters in each convolutional layer for making it able to overfit CIFAR-100.

[2]For SNIP, we adopt the implementation public at https://github.com/mi-lad/snip

Table 4: Test accuracy of pruned VGG19 and ResNet32 on Tiny-ImageNet dataset. The bold number is the higher one between the accuracy of GraSP and that of SNIP.

Network	VGG19			ResNet32		
Pruning ratio	90%	95%	98%	85%	90%	95%
VGG19/ResNet32 (Baseline)	61.38	-	-	62.89	-	-
OBD (LeCun et al., 1990)	61.21	60.49	54.98	58.55	56.80	51.00
MLPrune (Zeng & Urtasun, 2019)	60.23	59.23	55.55	58.86	57.62	51.70
LT (original initialization)	60.32	59.48	55.12	56.52	54.27	49.47
LT (reset to epoch 5)	61.19	60.57	56.18	60.31	57.77	51.21
SNIP (Lee et al., 2018)	**61.02±0.41**	59.27±0.39	48.95±1.73	56.33±0.24	55.43±0.14	49.57±0.44
GraSP	60.76±0.23	**59.50±0.33**	**57.28±0.34**	**57.25±0.11**	**55.53±0.11**	**51.34±0.29**

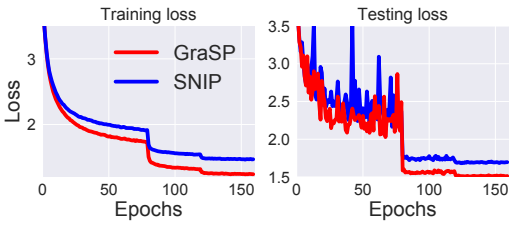

Figure 2: The training and testing loss on CIFAR-100 of SNIP and GraSP with ResNet32 and a pruning ratio of 98%.

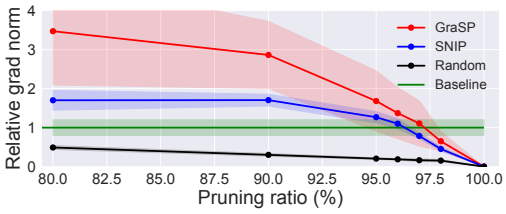

Figure 3: The gradient norm of ResNet32 after pruning on CIFAR-100 of various pruning ratios. Shaded area is the 95% confidence interval calculated with 10 trials.

of the winning tickets to their values at 5 epoch. By doing so, we observe a boost in the performance of the LT across all the settings, which is consistent with (Frankle et al., 2019).

Table 3: Test accuracy of ResNet-50 and VGG16 on ImageNet with pruning ratios 60%, 80% and 90%.

Pruning ratios	60%		80%		90%	
Accuracy	top-1	top-5	top-1	top5	top-1	top5
ResNet-50 (Baseline)	75.70	92.81	-	-	-	-
SNIP (Lee et al., 2018)	73.95	**91.97**	69.67	89.24	61.97	82.90
GraSP	**74.02**	91.86	**72.06**	**90.82**	**68.14**	**88.67**
VGG16 (Baseline)	73.37	91.47	-	-	-	-
SNIP (Lee et al., 2018)	**72.95**	**91.39**	69.96	89.71	65.27	86.14
GraSP	72.91	91.18	**71.65**	**90.58**	**69.94**	**89.48**

However, the above three datasets are overly simple and small-scale for reaching robust conclusions, and thus we conduct further experiments on ImageNet using ResNet-50 and VGG16 with pruning ratios $\{60\%, 80\%, 90\%\}$ to further validate the effectiveness of GraSP. The results are shown in Table 3, we can see that when the pruning ratio is 60%, both SNIP and GraSP can achieve very close performance to the original one, and these two methods perform almost the same. However, as we increase the pruning ratio, we observe that GraSP surpasses SNIP more and more. When the pruning ratio is 90%, GraSP can beat SNIP by 6.2% for ResNet-50 and 4.7% for VGG16 in top-1 accuracy, showing the advantages of GraSP at larger pruning ratios. We seek to investigate the reasons behind in Section 5.2 for obtaining a better understanding on GraSP.

5.2 Analysis on Convergence Performance and Gradient Norm

Based on our analysis in Section 4.1 and the observations in the previous subsection, we conduct further experiments on CIFAR-100 with ResNet32 to investigate where the performance gains come from in the high sparsity regions. We present the training statistics in terms of the training and test loss in Figure 2. We observe that the main bottleneck of pruned neural networks is underfitting, and thus support our optimization considerations when deriving the pruning criteria. As a result, the network pruned with GraSP can achieve much lower loss for both training and testing and the decrease in training loss is also much faster than SNIP.

Besides, we also plot the gradient norm of pruned networks at initialization for a ResNet32 on CIFAR-100. The gradient norm is computed as the average of the gradients over the entire dataset, and we normalize the gradient of the original network to be 1. We run each experiment for 10 trials in order to obtain more stable results. We observe that both SNIP and GraSP result in a lower gradient norm at high sparsity (e.g. 98%), but GraSP can better preserve the gradient norm after pruning, and also yield better results than SNIP at the high sparsity regions. Moreover, in these high sparsity regions, the pruned network usually underfits the training data, and thus the optimization will be a

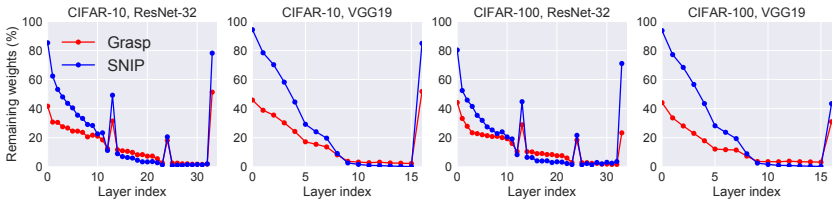

Figure 4: The portion of remaining weights at each layer after pruning with a pruning ratio of 95%.

problem. The randomly pruned network has a much lower gradient norm and performs the worst. In contrast, the network pruned by GraSP can start with a higher gradient norm and thus hopefully more training progress can be made as evidenced by our results in Figure 2.

5.3 Visualizing the Network after Pruning

In order to probe the difference of the network pruned by SNIP and GraSP, we present the portion of the remaining weights at each layer of the sparse network obtained by SNIP and GraSP in Figure. 4. We observe that these two pruning methods result in different pruning strategies. Specifically, GraSP aims for preserving the gradient flow after pruning, and thus will not prune too aggressively for each layer. Moreover, it has been known that those convolutional layers at the top usually learn highly sparse features and thus more weights can be pruned (Zhang et al., 2019a). As a result, both methods prune most of the weights at the top layers, but GraSP will preserve more weights than SNIP due to the consideration of preserving the gradient flow. In contrast, SNIP is more likely to prune nearly all the weights in top layers, and thus those layers will be the bottleneck of blocking the information flow from the output layer to the input layer.

5.4 Effect of Batch Size and Initialization

We also study how the batch size and initialization will affect GraSP on CIFAR-10 and CIFAR-100 with ResNet32 for showing the robustness to different hyperparameters. Specifically, we test GraSP with three different initialization methods: Kaiming normal (He et al., 2015), Normal $\mathcal{N}(0, 0.1)$, and Xavier normal (Glorot & Bengio, 2010), as well as different mini-batch sizes. We present the mean and standard variance of the test accuracy obtained with different initialization methods and by varying the batch sizes in Table 5. For CIFAR-10 and CIFAR-100, we use batch sizes $\{100, 400, \cdots, 25600, 50000\}$ and $\{1000, 4000, 16000, 32000, 50000\}$ respectively. We observe that GraSP can achieve reasonable performance with different initialization methods, and also the effect of batch size is minimal for networks pruned with Kaiming initialization, which is one of the most commonly adopted initialization techniques in training neural networks.

Table 5: Mean and standard variance of the test accuracy on CIFAR-10 and CIFAR-100 with ResNet32.

Dataset	CIFAR-10		CIFAR-100	
	60%	90%	60%	90%
Kaiming	93.42 ± 0.39	92.12 ± 0.39	71.60 ± 0.65	68.93 ± 0.36
Normal	93.31 ± 0.36	92.13 ± 0.36	71.48 ± 0.60	67.98 ± 0.83
Xavier	93.32 ± 0.25	92.22 ± 0.50	71.10 ± 1.27	68.11 ± 0.93

6 Discussion and Conclusion

We propose Gradient Signal Preservation (GraSP), a pruning criterion motivated by preserving the gradient flow through the network after pruning. It can also be interpreted as aligning the large eigenvalues of the Neural Tangent Kernel with the targets. Empirically, GraSP is able to prune the weights of a network *at initialization*, while still performing competitively to traditional pruning algorithms, which requires first training the network. More broadly, foresight pruning can be a way for enabling training super large models that no existing GPUs can fit in, and also reduce the training cost by adopting sparse matrices operations. Readers may notice that there is still a performance gap between GraSP and traditional pruning algorithms, and also the LT with late resetting performs better than LT with original initialization. However, this does not negate the possibility that foresight pruning can match the performance of traditional pruning algorithms while still enjoying cheaper training cost. As evidence, Evci et al. (2019) show that there exists a linear and monotonically decreasing path from the sparse initialization to the solution found by pruning the fully-trained dense network, but current optimizers fail to achieve this. Therefore, designing better gradient-based optimizers that can exploit such paths will also be a good direction to explore.

REFERENCES

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 322–332, 2019.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJ_wN01C-.

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*, pp. 248–255, 2009.

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

Sourya Dey, Kuan-Wen Huang, Peter A Beerel, and Keith M Chugg. Pre-defined sparse neural networks with hardware acceleration. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

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*, pp. 4857–4867, 2017.

Utku Evci, Fabian Pedregosa, Aidan Gomez, and Erich Elsen. The difficulty of training sparse neural networks. *arXiv preprint arXiv:1906.10732*, 2019.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019.

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*, pp. 249–256, 2010.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2015a.

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*, pp. 1135–1143, 2015b.

Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pp. 293–299. IEEE, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

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*, pp. 770–778, 2016.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

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

Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *arXiv preprint arXiv:1902.06720*, 2019a.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *International Conference on Learning Representations*, 2018.

Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip HS Torr. A signal propagation perspective for pruning neural networks at initialization. *arXiv preprint arXiv:1906.06307*, 2019b.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations*, 2016.

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

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *International Conference on Learning Representations*, 2018.

Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR 2017*, 2016.

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

Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BygfghAcYX.

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

Barak A Pearlmutter. Fast exact multiplication by the Hessian. *Neural computation*, 6(1):147–160, 1994.

Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2014.

Suraj Srinivas and R Venkatesh Babu. Generalized dropout. *arXiv preprint arXiv:1611.06791*, 2016.

Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. EigenDamage: Structured pruning in the Kronecker-factored eigenbasis. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pp. 6566–6575. PMLR, 2019. URL http://proceedings.mlr.press/v97/wang19g.html.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

Wenyuan Zeng and Raquel Urtasun. MLPrune: Multi-layer pruning for automated neural network compression, 2019. URL https://openreview.net/forum?id=r1g5b2RcKm.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *International Conference on Learning Representations*, 2016.

Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019a.

Guodong Zhang, James Martens, and Roger Grosse. Fast convergence of natural gradient descent for overparameterized neural networks. *arXiv preprint arXiv:1905.10961*, 2019b.