Sparsifying Networks via Subdifferential Inclusion

Sagar Verma [1] **Jean-Christophe Pesquet** [1]

Abstract

Sparsifying deep neural networks is of paramount interest in many areas, especially when those networks have to be implemented on low-memory devices. In this article, we propose a new formulation of the problem of generating sparse weights for a pre-trained neural network. By leveraging the properties of standard nonlinear activation functions, we show that the problem is equivalent to an approximate subdifferential inclusion problem. The accuracy of the approximation controls the sparsity. We show that the proposed approach is valid for a broad class of activation functions (ReLU, sigmoid, softmax). We propose an iterative optimization algorithm to induce sparsity whose convergence is guaranteed. Because of the algorithm flexibility, the sparsity can be ensured from partial training data in a minibatch manner. To demonstrate the effectiveness of our method, we perform experiments on various networks in different applicative contexts: image classification, speech recognition, natural language processing, and time-series forecasting. Project page: https://sagarverma.github.io/compression

1. Introduction

Deep neural networks have evolved to the state-of-the-art techniques in a wide array of applications: computer vision (Simonyan & Zisserman, 2015; He et al., 2016; Huang et al., 2017), automatic speech recognition (Hannun et al., 2014; Dong et al., 2018; Li et al., 2019; Watanabe et al., 2018; Hayashi et al., 2019; Inaguma et al., 2020), natural language processing (Turc et al., 2019; Radford et al., 2019; Dai et al., 2019b; Brown et al., 2020), and time series forecasting (Oreshkin et al., 2020). While their performance in various applications has matched and often exceeded human capabilities, neural networks may remain difficult to apply in real-world scenarios. Deep neural networks leverage the power of Graphical Processing Units (GPUs), which are power-hungry. Using GPUs to make billions of predictions per day, thus comes with a substantial energy cost. In addition, despite their quite fast response time, deep neural networks are not yet suitable for most real-time applications where memory-limited low-cost architectures need to be used. For all those reasons, compression and efficiency have become a topic of high interest in the deep learning community.

Sparsity in DNNs has been an active research topic generating numerous approaches. DNNs achieving the state-of-the-art in a given problem usually have a large number of layers with non-uniform parameter distribution across layers. Most sparsification methods are based on a global approach, which may result in a sub-optimal compression for a reduced accuracy. This may occur because layers with a smaller number of parameters may remain dense, although they may contribute more in terms of computational complexity (e.g., for convolutional layers). Some methods, also known as magnitude pruning, use a hard or soft-thresholding to remove less significant parameters. Soft thresholding techniques achieve a good sparsity-accuracy trade-off at the cost of additional parameters and increased computation time during training (Kusupati et al., 2020). Searching for a hardware efficient network is another area that has been proven quite useful, but it requires a huge amount of computational resources. Convex optimization techniques such as those used in (Aghasi et al., 2017) often rely upon fixed point iterations that make use of the proximity operator (Moreau, 1962). The related concepts are fundamental for tackling nonlinear problems and have recently come into play in the analysis of neural networks (Combettes & Pesquet, 2020a) and nonlinear systems (Combettes & Woodstock, 2021).

This paper shows that the properties of nonlinear activation functions can be utilized to identify highly sparse subnetworks. We show that the sparsification of a network can be formulated as an approximate subdifferential inclusion problem. We provide an iterative algorithm called subdifferential inclusion for sparsity (SIS) that uses partial training data to identify a sparse subnetwork while maintaining good accuracy. SIS makes even few-parameter

[1] Université Paris-Saclay, CentraleSupélec, Inria, Centre de Vision Numérique. Correspondence to: Sagar Verma <sagar.verma@centralesupelec.fr>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

layers sparse, resulting in models with significantly lower inference FLOPs than the baselines. For example, SIS for 90% sparse MobileNetV3 on ImageNet-1K achieves 66.07% top-1 accuracy with 33% fewer inference FLOPs than its dense counterpart and thus provides better results than the state-of-the-art method RigL. For non-convolutional networks like Transformer-XL trained on WikiText-103, SIS is able to achieve 70% sparsity while maintaining 21.1 perplexity score. We experiment on the following activation functions: Capped ReLU (Jasper), QuadReLU (MobileNet-V3), and ReLU/SoftMax (all networks). We evaluate our approach across four applicative domains and show that our compressed networks can achieve competitive accuracy for potential use on commodity hardware and edge devices.

2. Related Work

2.1. Inducing sparsity post training

Methods inducing sparsity after a dense network is trained involve several pruning and fine-tuning cycles till desired sparsity and accuracy are reached (Mozer & Smolensky, 1989; LeCun et al., 1990; Hassibi et al., 1993; Han et al., 2015; Molchanov et al., 2017; Guo et al., 2016; Park et al., 2020). (Renda et al., 2020) proposed weight rewinding technique instead of vanilla fine-tuning post-pruning. Net-Trim algorithm (Aghasi et al., 2017) removes connections at each layer of a trained network by convex programming. The proposed method works for networks using rectified linear units (ReLUs). Lowering rank of parameter tensors (Jaderberg et al., 2014; vahid et al., 2020; Lu et al., 2016), removing channels, filters and inducing group sparsity (Wen et al., 2016; Li et al., 2017; Luo et al., 2017; Gordon et al., 2018; Yu et al., 2019; Liebenwein et al., 2020) are some methods that take network structure into account. All these methods rely on pruning and fine-tuning cycle(s) often from full training data.

2.2. Inducing sparsity during training

Another popular approach has been to induce sparsity during training. This is achieved by modifying the loss function to consider sparsity as part of the optimization (Chauvin, 1989; Carreira-Perpinan & Idelbayev, 2018; Ullrich et al., 2017; Neklyudov et al., 2017). Bayesian priors (Louizos et al., 2017), L_0, L_1 regularization (Louizos et al., 2018), and variational dropout (Molchanov et al., 2017) get accuracy comparable to (Zhu & Gupta, 2018) but at a cost of 2× memory and 4× computations during training. (Liu et al., 2019; Savarese et al., 2020; Kusupati et al., 2020; Lee, 2019; Xiao et al., 2019; Azarian et al., 2020) have proposed learnable sparsity methods through training of the sparse masks and weights simultaneously with minimal heuristics. Although these methods are cheaper than pruning after training, they need at least the same computational effort as training a dense network to find a sparse sub-network. This makes them expensive when compressing big networks where the number of parameters ranges from hundreds of millions to billions (Dai et al., 2019b; Li et al., 2019; Brown et al., 2020). Methods like (Zhu & Gupta, 2018; Bellec et al., 2018; Mocanu et al., 2018; Dai et al., 2019a; Lin et al., 2020b) can be sub-classified as methods where dynamic pruning is performed during training by observing the network flow. (Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2020; Evci et al., 2020) computes weight magnitude and reallocates weights at every step of model training.

2.3. Training sparsely initialized networks

(Frankle & Carbin, 2019) showed that it is possible to find sparse sub-networks that, when trained from scratch, were able to match or even outperform their dense counterparts. (Lee et al., 2019) presented SNIP, a method to estimate, at initialization, the importance that each weight could have later during training. In (Lee et al., 2020) the authors perform a theoretical study of pruning at initialization from a signal propagation perspective, focusing on the initialization scheme. Recently, (Wang et al., 2020) proposed GraSP, a different method based on the gradient norm after pruning, and showed a significant improvement for moderate levels of sparsity. (Ye et al., 2020) starts with a small subnetwork and progressively grow it to a subnetwork that is as accurate as its dense counterpart. (Tanaka et al., 2020) proposes SynFlow that avoids flow collapse of a pruned network during training. (Jorge et al., 2020) proposed FORCE, an iterative pruning method that progressively removes a small number of weights. This method is able to achieve extreme sparsity at little accuracy expense. These methods are not usable for big pre-trained networks and are expensive as multiple training rounds are required for different sparse models depending on deployment scenarios (computing devices).

2.4. Efficient Neural Architecture Search

Hardware-aware NAS methods (Zoph et al., 2018; Real et al., 2019; Cai et al., 2018; Wu et al., 2019; Tan et al., 2019; Cai et al., 2019; Howard et al., 2019) directly incorporate the hardware feedback into efficient neural architecture search. (Cai et al., 2020) proposes to learn a single network composed of a large number of subnetworks from which a hardware aware subnetwork can be extracted in linear time. (Lin et al., 2020a) proposes a similar approach wherein they identify subnetworks that can be run efficiently on microcontrollers (MCUs).

The proposed algorithm applies to possibly large pre-trained networks. In contrast with methods presented in Section 2.1, ours can use a small amount of training data during pruning and fewer epochs during fine-tuning. As we will see in the next section, a key feature of our approach is that it is based on a fine analysis of the mathematical properties of activation functions, so allowing the use of powerful convex optimization tools. Through its block-iterative structure, our algorithm makes it possible to perform minibatch processing, while offering sound convergence guarantees. In Section 4, extensive numerical experiments show the good performance of this strategy.

3. Proposed Method

3.1. Variational principles

A basic neural network layer can be described by the relation:

$$y = R(Wx + b) \tag{1}$$

where $x \in \mathbb{R}^M$ is the input, $y \in \mathbb{R}^N$ the output, $W \in \mathbb{R}^{N\times M}$ is the weight matrix, $b \in \mathbb{R}^N$ the bias vector, and R is a nonlinear activation operator from $\mathbb{R}^N$ to $\mathbb{R}^N$. A key observation is that most of the activation operators currently used in neural networks are proximity operators of convex functions (Combettes & Pesquet, 2020a;b). We will therefore assume that there exists a proper lower-semicontinuous convex function f from $\mathbb{R}^N$ to $\mathbb{R} \cup \{+\infty\}$ such that $R = \operatorname{prox}_f$. We recall that f is a proper lower-semicontinuous convex function if the area overs its graph, its epigraph $\big\{(y,\xi) \in \mathbb{R}^N \times \mathbb{R} \;\big|\; f(y) \leqslant \xi\big\}$, is a nonempty closed convex set. For such a function the proximity operator of f at $z \in \mathbb{R}^N$ (Moreau, 1962) is the unique point defined as

$$\operatorname{prox}_f(z) = \underset{p\in\mathbb{R}^N}{\operatorname{argmin}} \; \frac{1}{2}\|z-p\|^2 + f(p). \tag{2}$$

It follows from standard subdifferential calculus that Eq. (1) can be re-expressed as the following inclusion relation:

$$Wx + b - y \in \partial f(y), \tag{3}$$

where $\partial f(y)$ is the Moreau subdifferential of f at y defined as

$$\partial f(y) = \big\{t \in \mathbb{R}^N \;\big|\; (\forall z \in \mathbb{R}^N) f(z) \geqslant f(y) + \langle t \mid z - y\rangle\big\}. \tag{4}$$

The subdifferential constitutes a useful extension of the notion of differential, which is applicable to nonsmooth functions. The set $\partial f(y)$ is closed and convex and, if y satisfies Eq. (1), it is nonempty. The distance to this set of a point $z \in \mathbb{R}^N$ is given by

$$d_{\partial f(y)}(z) = \inf_{t\in\partial f(y)} \|z - t\|. \tag{5}$$

We thus see that the subdifferential inclusion in Eq. (3) is also equivalent to

$$d_{\partial f(y)}(Wx + b - y) = 0. \tag{6}$$

Therefore, a suitable accuracy measure for approximated values of the layer parameters (W, b) is $d_{\partial f(y)}(Wx+b-y)$.

3.2. Optimization problem

Compressing a network consists of a sparsification of its parameters while keeping a satisfactory accuracy. Let us assume that, for a given layer, a training sequence of input/output pairs is available which results from a forward pass performed on the original network for some input dataset of length K. The training sequence is split in J minibatches of size T so that $K = JT$. The j-th minibatch with $j \in \{1,\ldots,J\}$ is denoted by $(x_{j,t}, y_{j,t})_{1\leqslant t\leqslant T}$. In order to compress the network, we propose to solve the following constrained optimization problem.

Problem 1 We want to

$$\underset{(W,b)\in C}{\operatorname{minimize}} \; g(W,b) \tag{7}$$

with

$$C = \Big\{(W,b) \in \mathbb{R}^{N\times M} \times \mathbb{R}^N \mid (\forall j \in \{1,\ldots,J\}) \\ \sum_{t=1}^{T} d^2_{\partial f(y_{j,t})}(Wx_{j,t} + b - y_{j,t}) \leqslant T\eta\Big\}, \tag{8}$$

where g is a sparsity measure defined on $\mathbb{R}^{N\times M} \times \mathbb{R}^N$ and $\eta \in [0, +\infty[$ is some accuracy tolerance.

Since, for every $j \in \{1,\ldots,J\}$, the function $(W,b) \mapsto \sum_{t=1}^{T} d^2_{\partial f(y_{j,t})}(Wx_{j,t} + b - y_{j,t})$ is continuous and convex, C is a closed and convex subset of $\mathbb{R}^{N\times M} \times \mathbb{R}^N$. In addition, this set is nonempty when there exist $\overline{W} \in \mathbb{R}^{N\times M}$ and $\overline{b} \in \mathbb{R}^N$ such that, for every $j \in \{1,\ldots,J\}$ and $t \in \{1,\ldots,T\}$,

$$d^2_{\partial f(y_{j,t})}(\overline{W}x_{j,t} + \overline{b} - y_{j,t}) = 0. \tag{9}$$

As we have seen in Section 3.1, this condition is satisfied when $(\overline{W}, \overline{b})$ are the parameters of the uncompressed layer. Often, the sparsity of the weight matrix is the determining factor whereas the bias vector represents a small number of parameters, so that we can make the following assumption.

Assumption 2 For every $W \in \mathbb{R}^{N\times M}$ and $b \in \mathbb{R}^N$, $g(W,b) = h(W)$ where h is a function from $\mathbb{R}^{N\times M}$ to $\mathbb{R} \cup \{+\infty\}$, which is lower-semicontinuous, convex, and coercive (i.e. $\lim_{\|W\|_{\mathrm{F}}\to+\infty} h(W) = +\infty$). In addition, there exists $(\overline{W}, \overline{b}) \in C$ such that $h(\overline{W}) < +\infty$ and there exists $(j^*, t^*) \in \{1,\ldots,J\} \times \{1,\ldots,T\}$ such that y_{j^*,t^*} lies in the interior of the range of R.

Under this assumption, the existence of a solution to Problem 1 is guaranteed (see Appendix A). A standard choice for such a function is the ℓ_1-norm of the matrix elements, $h = \|\cdot\|_1$, but other convex sparsity measures could also be easily incorporated within this framework, e.g. group sparsity measures. Another point worth being noticed is that constraints other than (8) could be imposed. For example, one could make the following alternative choice for the constraint set

$$C = \Big\{(W,b) \in \mathbb{R}^{N\times M} \times \mathbb{R}^N \mid \sup_{j\in\{1,\ldots,J\}, t\in\{1,\ldots,T\}} d_{\partial f(y_{j,t})}(Wx_{j,t} + b - y_{j,t}) \leqslant \sqrt{\eta}\Big\}. \tag{10}$$

Although the resulting optimization problem could be tackled by the same kind of algorithm as the one we will propose, Constraint (8) leads to a simpler implementation.

3.3. Optimization algorithm

A standard proximal method for solving Problem 1 is the Douglas-Rachford algorithm (Lions & Mercier, 1979; Combettes & Pesquet, 2007). This algorithm alternates between a proximal step aiming at sparsifying the weight matrix and a projection step allowing a given accuracy to be reached. This algorithm reads as shown below.

Algorithm 1 Douglas-Rachford algorithm for network compression

```
Initialize: $\widehat{W}_0 \in \mathbb{R}^{N\times M}$ and $b_0 \in \mathbb{R}^N$
for $n = 0, 1, \ldots$ do
    $W_n = \mathrm{prox}_{\gamma h}(\widehat{W}_n)$
    $(\widetilde{W}_n, \widetilde{b}_n) = \mathrm{proj}_C(2W_n - \widehat{W}_n, b_n)$
    $\widehat{W}_{n+1} = \widehat{W}_n + \lambda_n(\widetilde{W}_n - W_n)$
    $b_{n+1} = b_n + \lambda_n(\widetilde{b}_n - b_n)$.
```

The Douglas-Rachford algorithm uses positive parameters γ and $(\lambda_n)_{n\in\mathbb{N}}$. Throughout this article, proj_S denotes the projection onto a nonempty closed convex set S. The convergence of Algorithm 1 is guaranteed by the following result (see illustrations in Subsection 4.3).

Proposition 3 *(Combettes & Pesquet, 2007) Assume that Problem 1 has a solution and that there exists $(\overline{W}, \overline{b}) \in C$ such $\overline{W}$ is a point in the interior of the domain of h. Assume that $\gamma \in\ ]0, +\infty[$ and $(\lambda_n)_{n\in\mathbb{N}}$ in $]0, 2[$ is such that $\sum_{n\in\mathbb{N}} \lambda_n(2-\lambda_n) = +\infty$. Then the sequence $(W_n, b_n)_{n\in\mathbb{N}}$ generated by Algorithm 1 converges to a solution to Problem 1.*

The proximity operator of function γh has a closed-form for standard choices of sparsity measures[1]. For example, when $h = \|\cdot\|_1$, this operator reduces to a soft-thresholding (with threshold value γ) of the input matrix elements. In turn, since the convex set C has an intricate form, an explicit expression of proj_C does not exist. Finding an efficient method for computing this projection for large datasets thus constitutes the main challenge in the use of the above Douglas-Rachford strategy, which we will discuss in the next section.

3.4. Computation of the projection onto the constraint set

For every mini-batch index $j \in \{1, \ldots, J\}$, let us define the following convex function:

$$(\forall (W,b) \in \mathbb{R}^{N\times M} \times \mathbb{R}^N)$$
$$c_j(W,b) = \sum_{t=1}^{T} d^2_{\partial f(y_{j,t})}(Wx_{j,t} + b - y_{j,t}) - T\eta. \tag{11}$$

Note that, for every $j \in \{1, \ldots, J\}$, function c_j is differentiable and its gradient at $(W,b) \in \mathbb{R}^{N\times M} \times \mathbb{R}^N$ is given by

$$\nabla c_j(W,b) = (\nabla_{\mathsf{W}} c_j(W,b), \nabla_{\mathsf{b}} c_j(W,b)), \tag{12}$$

where

$$\nabla_{\mathsf{W}} c_j(W,b) = 2\sum_{t=1}^{T} e_{j,t} x_{j,t}^\top, \quad \nabla_{\mathsf{b}} c_j(W,b) = 2\sum_{t=1}^{T} e_{j,t} \tag{13}$$

with, for every $t \in \{1, \ldots, T\}$,

$$e_{j,t} = Wx_{j,t} + b - y_{j,t} - \mathrm{proj}_{\partial f(y_{j,t})}(Wx_{j,t} + b - y_{j,t}). \tag{14}$$

A pair of weight/bias parameters belongs to C if and only if it lies in the intersection of the 0-lower level sets of the functions $(c_j)_{1\leqslant j\leqslant J}$. To compute the projection of some $(W,b) \in \mathbb{R}^{N\times M} \times \mathbb{R}^N$ onto this intersection, we use Algorithm 2 ($\|\cdot\|_{\mathrm{F}}$ denotes here the Frobenius norm).

This iterative algorithm has the advantage of proceeding in a minibatch manner. It allows us to choose the mini-batch index j_n at iteration n in a quasi-cyclic manner. The simplest rule is to activate each minibatch once within J successive iterations of the algorithm so that they correspond to an epoch. The proposed algorithm belongs to the family of block-iterative outer approximation schemes for solving constrained quadratic problems, which was introduced in (Combettes, 2003). The convergence of the sequence $(W_n, b_n)_{n\in\mathbb{N}}$ generated by Algorithm 2 to $\mathrm{proj}_C(W,b)$ is thus guaranteed. One of the main features of the algorithm is that it does not require to perform any projection onto the 0-lower level sets of the functions c_j, which would be intractable due to their expressions. Instead, these projections are implicitly replaced by subgradient projections, which are much easier to compute in our context.

[1] http://proximity-operator.net

Algorithm 2 Minibatch algorithm for computing $\mathrm{proj}_C(W, b)$

```
Initialize : $W_0 = W$ and $b_0 = b$
for $n = 0, 1, \ldots$ do
    Select a batch of index $j_n \in \{1, \ldots, J\}$
    if $c_{j_n}(W_n, b_n) > 0$ then
        Compute $\nabla_{\mathsf{W}} c_{j_n}(W_n, b_n)$ and $\nabla_{\mathsf{b}} c_{j_n}(W_n, b_n)$ by using Eqs. (13) and (14)
        $\delta W_n = \frac{c_{j_n}(W_n,b_n)\,\nabla_{\mathsf{W}} c_{j_n}(W_n,b_n)}{\|\nabla_{\mathsf{W}} c_{j_n,n}(W_n,b_n)\|_{\mathrm{F}}^2 + \|\nabla_{\mathsf{b}} c_{j_n}(W_n,b_n)\|^2}$
        $\delta b_n = \frac{c_{j_n}(W_n,b_n)\,\nabla_{\mathsf{b}} c_{j_n}(W_n,b_n)}{\|\nabla_{\mathsf{W}} c_{j_n,n}(W_n,b_n)\|_{\mathrm{F}}^2 + \|\nabla_{\mathsf{b}} c_{j_n}(W_n,b_n)\|^2}$
        $\pi_n = \mathrm{tr}((W_0 - W_n)^\top \delta W_n) + (b_0 - b_n)^\top \delta b_n$
        $\mu_n = \|W_0 - W_n\|_{\mathrm{F}}^2 + \|b_0 - b_n\|^2$
        $\nu_n = \|\delta W_n\|_{\mathrm{F}}^2 + \|\delta b_n\|^2$
        $\zeta_n = \mu_n \nu_n - \pi_n^2$
        if $\zeta_n = 0$ and $\pi_n \geqslant 0$ then
            $W_{n+1} = W_n - \delta W_n$
            $b_{n+1} = b_n - \delta b_n$
        else if $\zeta_n > 0$ and $\pi_n \nu_n \geqslant \zeta_n$ then
            $W_{n+1} = W_0 - (1 + \frac{\pi_n}{\nu_n})\delta W_n$
            $b_{n+1} = b_0 - (1 + \frac{\pi_n}{\nu_n})\delta b_n$
        else
            $W_{n+1} = W_n + \frac{\nu_n}{\zeta_n}(\pi_n(W_0 - W_n) - \mu_n \delta W_n)$
            $b_{n+1} = b_n + \frac{\nu_n}{\zeta_n}(\pi_n(b_0 - b_n) - \mu_n \delta b_n)$
    else
        $W_{n+1} = W_n$
        $b_{n+1} = b_n$
```

3.5. Dealing with various nonlinearities

For any choice of activation operator R, we have to calculate the projection onto $\partial f(y)$ for every vector y satisfying Eq. (1). This projection is indeed required in the computation of the gradients of functions $(c_j)_{1 \leqslant j \leqslant J}$, as shown by Eq. (14). Two properties may facilitate this calculation. First, if f is differentiable at y, then $\partial f(y)$ reduces to a singleton containing the gradient $\nabla f(y)$ of f at y, so that, for every $z \in \mathbb{R}^N$, $\mathrm{proj}_{\partial f(y)}(z) = \nabla f(y)$. Second, R is often separable, i.e. consists of the application of a scalar activation function $\rho\colon \mathbb{R} \to \mathbb{R}$ to each component of its input argument. According to our assumptions, there thus exists a proper lower-semicontinuous convex function φ from $\mathbb{R}$ to $\mathbb{R} \cup \{+\infty\}$ such that $\rho = \mathrm{prox}_\varphi$ and, for every $z = (\zeta^{(k)})_{1 \leqslant k \leqslant N} \in \mathbb{R}^N$, $f(z) = \sum_{k=1}^N \varphi(\zeta^{(k)})$. This implies that, for every $z = (\zeta^{(k)})_{1 \leqslant k \leqslant N} \in \mathbb{R}^N$, $\mathrm{proj}_{\partial f(y)}(z) = (\mathrm{proj}_{\partial \varphi(\upsilon^{(k)})}(\zeta^{(k)}))_{1 \leqslant k \leqslant N}$, where the components of y are denoted by $(\upsilon^{(k)})_{1 \leqslant k \leqslant N}$. Based on these properties, a list of standard activation functions ρ is given in Table 1, for which we provide the associated expressions of the projection onto $\partial \varphi$. The calculations are detailed in Appendix B.

An example of non-separable activation operator frequently employed in neural network architectures is the softmax operation defined as

$$(\forall z = (\zeta^{(k)})_{1 \leqslant k \leqslant N} \in \mathbb{R}^N) \quad R(z) = \left(\frac{\exp(\zeta^{(k)})}{\sum_{k'=1}^N \exp(\zeta^{(k')})} \right)_{1 \leqslant k \leqslant N}. \tag{15}$$

It is shown in Appendix C that, for every $y = (\upsilon^{(k)})_{1 \leqslant k \leqslant N}$ in the range of R,

$$(\forall z \in \mathbb{R}^N) \quad \mathrm{proj}_{\partial f(y)}(z) = Q(y) + \frac{\mathbf{1}^\top (z - Q(y))}{N} \mathbf{1}, \tag{16}$$

where $\mathbf{1} = [1, \ldots, 1]^\top \in \mathbb{R}^N$ and

$$Q(y) = (\ln \upsilon^{(k)} + 1 - \upsilon^{(k)})_{1 \leqslant k \leqslant N}. \tag{17}$$

3.6. SIS on multi-layered networks

Algorithm 3 Parallel SIS for multi-layered network

```
Input: input sequence $X \in \mathbb{R}^{M \times K}$, compression parameter $\eta > 0$, weight matrices $W^{(1)}, \ldots, W^{(L)}$, and bias vectors $b^{(1)}, \ldots, b^{(L)}$
$Y^{(0)} \leftarrow X$
for $l = 1, \ldots, L$ do
    $Y^{(l)} = R_l(W^{(l)\top} Y^{(l-1)} + b^{(l)})$
    $\widehat{W}^{(l)}, \widehat{b}^{(l)} \leftarrow \mathrm{SIS}(\eta, W^{(l)}, b^{(l)}, Y^{(l)}, Y^{(l-1)})$
Output: $\widehat{W}^{(1)}, \ldots, \widehat{W}^{(L)}$ and $\widehat{b}^{(1)}, \ldots, \widehat{b}^{(L)}$
```

Algorithm 3 describes how we make use of SIS for a multi-layered neural network. We use a pre-trained network and part of the training sequence to extract layer-wise input-output features. Then we apply SIS on each individual layer l by passing η, layer parameters $(W^{(l)}, b^{(l)})$ and extracted input-output features $(Y^{(l-1)}, Y^{(l)})$ to Algorithm 1. The benefit of applying SIS to each layer independently is that we can run SIS on all the layers of a network in parallel. This reduces the time required to process the whole network and compute resources are optimally utilized.

4. Experiments

In this section, we conduct various experiments to validate the effectiveness of SIS in terms of test accuracy vs. sparsity and inference time FLOPs vs. sparsity by comparing against RigL (Evci et al., 2020). We also include SNIP (Lee et al., 2019), GraSP (Wang et al., 2020), SynFlow (Tanaka et al., 2020), STR (Kusupati et al., 2020), and FORCE (Jorge et al., 2020). These methods start training from a sparse network

Name	$\rho(\zeta)$	$\mathrm{proj}_{\partial\varphi(\upsilon)}(\zeta)$
Sigmoid	$(1+e^{-\zeta})^{-1}-\frac{1}{2}$	$\ln(\upsilon+1/2)-\ln(\upsilon-1/2)-\upsilon$
Arctangent	$(2/\pi)\arctan(\zeta)$	$\tan(\pi\upsilon/2)-\upsilon$
ReLU	$\max\{\zeta,0\}$	$\begin{cases}0 & \text{if } \upsilon>0 \text{ or } \zeta\geqslant 0\\ \zeta & \text{otherwise}\end{cases}$
Leaky ReLU	$\begin{cases}\zeta & \text{if } \zeta>0\\ \alpha\zeta & \text{otherwise}\end{cases}$	$\begin{cases}0 & \text{if } \upsilon>0\\ (1/\alpha-1)\upsilon & \text{otherwise}\end{cases}$
Capped ReLU	$\mathrm{ReLU}_\alpha(\zeta)=\min\{\max\{\zeta,0\},\alpha\}$	$\begin{cases}\zeta & \text{if } (\upsilon=0 \text{ and } \zeta<0)\\ & \text{or } (\upsilon=\alpha \text{ and } \zeta>0)\\ 0 & \text{otherwise}\end{cases}$
ELU	$\begin{cases}\zeta & \text{if } \zeta\geqslant 0\\ \alpha\big(\exp(\zeta)-1\big) & \text{otherwise}\end{cases}$	$\begin{cases}0 & \text{if } \upsilon>0\\ \ln\left(\frac{\upsilon+\alpha}{\alpha}\right)-\upsilon & \text{otherwise}\end{cases}$
QuadReLU	$\dfrac{(\zeta+\alpha)\mathrm{ReLU}_{2\alpha}(\zeta+\alpha)}{4\alpha}$	$\begin{cases}\upsilon & \text{if } \upsilon=0 \text{ and } \zeta\leqslant-\alpha\\ -\upsilon+2\sqrt{\alpha\upsilon}-\alpha & \text{if } \upsilon\in]0,\alpha]\\ & \text{or } (\upsilon=0 \text{ and } \zeta>-\alpha)\\ \upsilon-\alpha & \text{otherwise}\end{cases}$

Table 1. Expression of $\mathrm{proj}_{\partial\varphi(\upsilon)}(\zeta)$ for $\zeta\in\mathbb{R}$ and υ in the range of ρ, for standard activation functions ρ. α is a positive constant.

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG19 (Baseline)	94.23	-	-	74.16	-	-
SNIP (Lee et al., 2019)	93.63	93.43	92.05	**72.84**	71.83	58.46
GraSP (Wang et al., 2020)	93.30	93.04	92.19	71.95	71.23	68.90
SynFlow (Tanaka et al., 2020)	93.35	93.45	92.24	71.77	71.72	70.94
STR (Kusupati et al., 2020)	93.73	93.27	92.21	71.93	71.14	69.89
FORCE (Jorge et al., 2020)	93.87	93.30	92.25	71.9	71.73	70.96
LRR (Renda et al., 2020)	**94.03**	**93.53**	91.73	72.12	71.36	70.39
RigL (Evci et al., 2020)	93.47	93.35	93.14	71.82	71.53	70.71
SIS (Ours)	93.99	93.31	**93.16**	72.06	**71.85**	**71.17**
ResNet50 (Baseline)	94.62	-	-	77.39	-	-
SNIP (Lee et al., 2019)	92.65	90.86	87.21	73.14	69.25	58.43
GraSP (Wang et al., 2020)	92.47	91.32	88.77	73.28	70.29	62.12
SynFlow (Tanaka et al., 2020)	92.49	91.22	88.82	73.37	70.37	62.17
STR (Kusupati et al., 2020)	92.59	91.35	88.75	73.45	70.45	62.34
FORCE (Jorge et al., 2020)	92.56	91.46	88.88	73.54	70.37	62.39
LRR (Renda et al., 2020)	92.62	91.27	89.11	**74.13**	70.38	62.47
RigL (Evci et al., 2020)	92.55	91.42	89.03	73.77	70.49	62.33
SIS (Ours)	**92.81**	**91.69**	**90.11**	73.81	**70.62**	**62.75**

Table 2. Test accuracy of sparse VGG19 and ResNet50 on CIFAR-10 and CIFAR-100 datasets.

and have some limitations when compared to methods that prune a pre-trained network (Blalock et al., 2020; Gale et al., 2019). For a fair comparison we also include LRR (Renda et al., 2020) which uses a pre-trained network and multiple rounds of pruning and retraining by leveraging learning rate rewinding. The experimental setup is described in Appendix D.

4.1. Modern ConvNets on CIFAR and ImageNet

We compare SIS with competitive baselines on CIFAR-10/100 for three different sparsity regimes 90%, 95%, 98%, and the results are listed in Table 2. It can be observed that LRR, RigL and SIS are able to maintain high accuracy with increasing sparsity. LRR performs better than both RigL and SIS for VGG19 on CIFAR-10 at 90% and 95%

Sparsity	60%			80%			90%			96.5%		
	Train/Prune FLOPs (×e16)	Top-1 Acc(%)	Infer FLOPs	Train/Prune FLOPs (×e16)	Top-1 Acc(%)	Infer FLOPs	Train/Prune FLOPs (×e16)	Top-1 Acc(%)	Infer FLOPs	Train/Prune FLOPs (×e16)	Top-1 Acc(%)	Infer FLOPs
SNIP	0.978	74.06	1.88G	0.696	72.34	941M	0.537	66.97	409M	0.502	59.16	292M
GraSP	0.903	75.95	1.63G	0.650	74.21	786M	0.555	70.71	470M	0.501	69.55	290M
SynFlow	0.898	76.54	1.61G	0.665	74.14	776M	0.553	71.01	465M	0.500	70.10	288M
FORCE	**0.833**	75.47	1.39G	0.619	73.42	685M	0.550	72.59	455M	0.497	72.04	276M
SparseVD	1.827	76.75	1.71G	1.737	74.68	811M	1.702	69.73	461M	1.685	67.13	286M
BC-GHS.	1.825	76.45	1.69G	1.737	74.15	813M	1.701	71.33	454M	1.684	68.54	282M
$L_{0_{hc}}, \lambda = e - 5$	1.825	76.98	1.69G	1.736	76.67	802M	1.702	71.61	459M	1.684	68.61	276M
STR	0.891	**77.75**	1.59G	0.625	76.11	704M	0.516	75.72	341M	0.449	71.87	117M
NetTrim	1.148	74.52	1.71G	0.866	72.88	842M	0.465	67.62	461M	0.283	62.01	281M
SIS (Ours)	0.923	77.05	**1.34G**	**0.435**	**76.96**	**647M**	**0.351**	**76.31**	**298M**	**0.102**	**73.11**	**101M**

Table 3. Pruning phase compute cost, test Top-1 accuracy and single image inference FLOPs of sparse ResNet50 on ImageNet where baseline accuracy and inference FLOPs are 77.37% and 4.14G, respectively. All methods were applied on same pre-trained "dense" ResNet50. 20% samples per class were used during pruning phase of all the methods and were run for 40 epochs.

sparsity. When compared to SNIP, our method achieves impressive performance for VGG19 on CIFAR-100 (58.46 → 71.17). In the case of ResNet50, SIS outperforms all the other methods for CIFAR-10/100 except for CIFAR-100 at 90%.

Sparsity	75%			90%		
	LRR	RigL	SIS (Ours)	LRR	RigL	SIS (Ours)
V1 (70.90)	68.79	69.97	**70.11**	66.59	67.10	**67.15**
FLOPs (569M)	498M	461M	**367M**	401M	331M	**284M**
V2 (71.88)	68.83	69.60	**69.83**	64.17	**65.23**	65.11
FLOPs (300M)	267M	211M	**182M**	192M	174M	**162M**
V3 (72.80)	68.97	70.21	**70.47**	64.32	65.13	**66.07**
FLOPs (226M)	187M	198M	**172M**	185M	167M	**151M**

Table 4. Test accuracy and inference FLOPs of sparse MobileNet versions using RigL and SIS on ImageNet, baseline accuracy and inference FLOPs shown in brackets.

Due to its small size and controlled nature, CIFAR-10/100 may not appear sufficient to draw solid conclusions. We thus conduct further experiments on ImageNet using ResNet50 and MobileNets. For ResNet50 on ImageNet experiment, we adapt SNIP (Lee, 2019), GraSP (Wang et al., 2020), SynFlow (Tanaka et al., 2020), STR (Kusupati et al., 2020), FORCE (Jorge et al., 2020), SpraseVD (Molchanov et al., 2017), Bayesian Compression (Louizos et al., 2017), and L0 regularization (Louizos et al., 2018) methods to use pre-trained weights. We also include results from NetTrim (Aghasi et al., 2017) which is another convex optimization based pruning method. Table 3 shows that, in the case of ResNet50, STR performs marginally better than SIS at 60% sparsity. At 80%, 90%, and 96.5% sparsity SIS outperforms all other methods. For all sparsity regimes, SIS achieves least inference FLOPs. Training FLOPs is best for SIS in 80%, 90%, and 96.5% regimes, FORCE achieves best training FLOPs in 60% regime. MobileNets are compact architectures designed specifically for resource-constrained devices. Table 4 shows results for RigL and SIS on MobileNets. We observe that SIS outperforms all MobileNet versions at 75% sparsity level. For a 90% sparsity level, SIS outperforms RigL for MobileNet V1 and V3 whereas, for MobileNetV2, RigL performs slightly better than SIS at 90% sparsity level. In all the cases, we can see that the resulting SIS sparse network uses fewer FLOPs than RigL. A possible explanation for this fact is that SIS leverages activation function properties during the sparsification process.

4.2. Sequential Tasks

Jasper on LibriSpeech. Jasper is a speech recognition model that uses 1D convolutions. The trained network is a 333 million parameter model and has a word error rate (WER) of 12.2 on the test set. We apply SIS on this network and compare it with RigL and SNIP in terms of sparsity. Table 5 reports WER and inference FLOPs for all three methods. SIS marginally performs better than LRR on this task in terms of WER and FLOPs for 70% sparsity. The main advantage of our approach lies in the fact that we can use a single pre-trained Jasper network and achieve different sparsity level for different types of deployment scenarios with less computational resources than RigL.

Transformer-XL on WikiText-103. Transformer-XL is a language model with 246 million parameters. The trained network on WikiText-103 has a perplexity score (PPL) of 18.6. In Table 5, we see that SIS performs better than SNIP and RigL in terms of PPL and has 68% fewer inference FLOPs. This is due to the fact that large language models can be efficiently trained and then compressed easily, but training a sparse sub-network from scratch is hard (Li et al., 2020), as is the case with SNIP and RigL. SNIP uses one-shot pruning to obtain a random sparse sub-network, whereas RigL is able to change its structure during training, which allows it to perform better than SNIP.

Network	JASPER		Transformer-XL		N-BEATS	
	WER	FLOPs	PPL	FLOPs	SMAPE	FLOPs
Dense	12.2	4.53G	18.6	927.73G	8.3	41.26M
SNIP (Lee et al., 2019)	14.3	2.74G	24.6	398.92G	10.1	21.45M
LRR (Renda et al., 2020)	13.7	2.61G	23.1	339.21G	**9.3**	14.47M
RigL (Evci et al., 2020)	13.9	2.69G	22.4	326.56G	10.2	15.13M
SIS (Ours)	**13.1**	**2.34G**	**21.1**	**290.38G**	9.7	**14.21M**

Table 5. Test accuracy and inference FLOPs of JASPER, Transformer-XL, and N-BEATS at 70% sparsity.

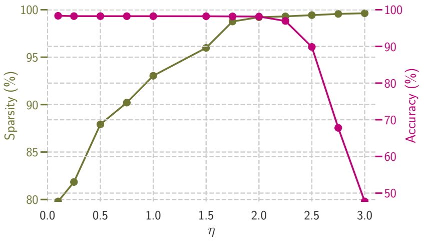

Figure 1. Effect of η on LeNet-FCN

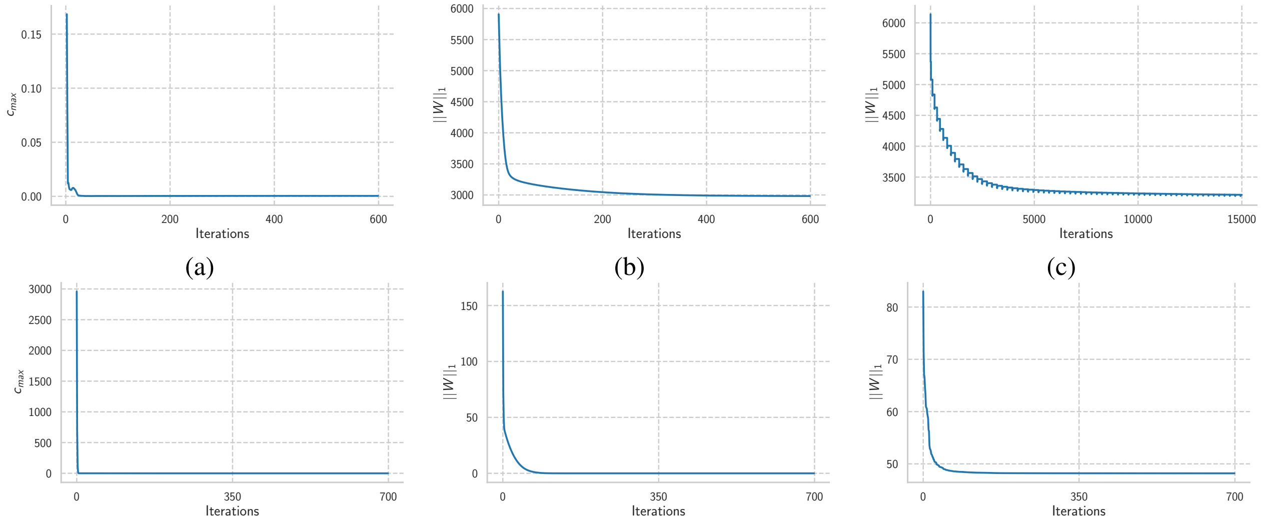

Figure 2. Convergence of SLIC: Top row coresponds to the first layer (ReLU activated) and bottom row corresponds to the last one (softmaxed) in LeNet-FCN. (a) and (d) show the evolution of the maximum value $c_{\max}$ of the constraint functions $(c_j)_{1 \leqslant j \leqslant J}$, (b) and (e) show the evolution of $\|W\|_1$ along Algorithm 1 iterations. (c) and (f) show $\|W\|_1$ evolution in Algorithm 2.

N-BEATS on M4. N-BEATS is a very deep residual fully-connected network to perform forecasting in univariate time-series problems. It is a 14 million parameter network. The Symmetric Mean Absolute Percentage Error (SMAPE) of the dense network on the M4 dataset is 8.3%. We apply SIS on this network and compare its performance with respect to RigL and SIS. As shown Table 5, SIS performs better than both methods and results in 65% fewer inference FLOPs.

4.3. Empirical Convergence Analysis

The η parameter in our algorithm controls the accuracy tolerance. The higher, the more tolerant we are on the loss of precision and the sparser the network is. Thus, this parameter also controls the network sparsity. The choice of this parameter should be the result of an accuracy-sparsity trade-off. This is illustrated in Figure 1.

We illustrate the convergence of our method on LeNet-FCN trained on MNIST. LeNet-FCN is a fully-connected network having four layers with 784-300-1000-300-10 nodes (two 300 nodes and one 1000 node hidden layers). Figure 2 shows the convergence of SIS when applied to dense LeNet-FCN. We observe that the convergence is smooth and SIS finds a global solution for the first (ReLU activated) and last (softmax) layer cases. This fact is in agreement with our theoretical claims. SIS attains a sparsity of 99.21% at an error of 1.86%. The trained dense network has an error of 1.65%. This result is obtained at $\eta = 2$.

5. Conclusion

In this article, we have proposed a novel method for sparsifying neural networks. The compression problem for each layer has been recast as the minimization of a sparsity measure under accuracy constraints. This constrained optimization problem has been solved by means of advanced convex optimization tools. The resulting SIS algorithm is

i) reliable in terms of iteration convergence guarantees,

ii) applicable to a wide range of activation operators,

iii) able to deal with large datasets split into mini-batches.

Our numerical tests demonstrate that the approach is not only appealing from a theoretical viewpoint but also practically efficient.

Acknowledgements

J.-C. Pesquet would like to thank P. L. Combettes for fruitful discussions concerning the mathematical formulation of the problem. Part of this work was supported by Institut Universitaire de France and the ANR Research and Teaching Chair in Artificial Intelligence BRIDGEABLE.

References

Aghasi, A., Abdi, A., Nguyen, N., and Romberg, J. Net-trim: Convex pruning of deep neural networks with performance guarantee. In *NeuIPS*, 2017.

Azarian, K., Bhalgat, Y., Lee, J., and Blankevoort, T. Learned threshold pruning. *arXiv preprint arXiv:2003.00075*, 2020.

Bauschke, H. H. and Combettes, P. L. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer, 2019.

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. Deep rewiring: Training very sparse deep networks. In *ICLR*, 2018.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. What is the state of neural network pruning? In Dhillon, I., Papailiopoulos, D., and Sze, V. (eds.), *Proceedings of Machine Learning and Systems*, volume 2, pp. 129–146, 2020.

Brown, T. B., Mann, B. P., Ryder, N., Subbiah, M., Kaplan, J., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Cai, H., Yang, J., Zhang, W., Han, S., and Yu, Y. Path-level network transformation for efficient architecture search. *arXiv preprint arXiv:1806.02639*, 2018.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020.

Carreira-Perpinan, M. A. and Idelbayev, Y. "Learning-Compression” algorithms for neural net pruning. In *CVPR*, 2018.

Chauvin, Y. A back-propagation algorithm with optimal use of hidden units. In *NeurIPS*. 1989.

Combettes, P. L. A block-iterative surrogate constraint splitting method for quadratic signal recovery. *IEEE TSP*, 2003.

Combettes, P. L. and Pesquet, J.-C. A Douglas–Rachford splitting approach to nonsmooth convex variational signal recovery. *IEEE JSTSP*, 2007.

Combettes, P. L. and Pesquet, J.-C. Deep neural network structures solving variational inequalities. *SVVA*, 2020a.

Combettes, P. L. and Pesquet, J.-C. Lipschitz certificates for layered network structures driven by averaged activation operators. *SIMODS*, 2020b.

Combettes, P. L. and Woodstock, Z. C. A fixed point framework for recovering signals from nonlinear transformations. In *EUSIPCO*, 2021.

Dai, X., Yin, H., and Jha, N. K. NeST: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE TC*, 2019a.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., et al. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*, 2019b.

Dettmers, T. and Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2020.

Dong, L., Xu, S., and Xu, B. Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. In *ICASSP*, 2018.

Evci, U., Elsen, E., Castro, P., and Gale, T. Rigging the lottery: Making all tickets winners. In *ICML*, 2020.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., et al. Morphnet: Fast simple resource-constrained structure learning of deep networks. In *CVPR*, 2018.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *NeurIPS*, 2016.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural networks. In *NeurIPS*, 2015.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., et al. Deepspeech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *ICNN*, 1993.

Hayashi, T., Yamamoto, R., Inoue, K., Yoshimura, T., Watanabe, S., et al. ESPnet-TTS: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit. *arXiv preprint arXiv:1910.10909*, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

Howard, A., Sandler, M., Chu, G., Chen, L., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3. In *ICCV*, 2019.

Huang, G., Liu, Z., and Weinberger, K. Q. Densely connected convolutional networks. In *CVPR*, 2017.

Inaguma, H., Kiyono, S., Duh, K., Karita, S., Soplin, N. E. Y., et al. ESPnet-ST: All-in-one speech translation toolkit. *arXiv preprint arXiv:2004.10234*, 2020.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In *BMVC*, 2014.

Jorge, P., Sanyal, A., Behl, H., Torr, P., Rogez, G., and Dokania, P. Progressive skeletonization: Trimming more fat from a network at initialization. *arXiv preprint arXiv:2006.09081*, 2020.

Kusupati, A., Ramanujan, V., Somani, R., Wortsman, M., Jain, P., Kakade, S., et al. Soft threshold weight reparameterization for learnable sparsity. In *ICML*, 2020.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *NeurIPS*, 1990.

Lee, N., Ajanthan, T., and Torr, P. SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

Lee, N., Ajanthan, T., Gould, S., and Torr, P. H. S. A signal propagation perspective for pruning neural networks at initialization. In *ICLR*, 2020.

Lee, Y. Differentiable sparsification for deep neural networks. *arXiv preprint arXiv:1910.03201*, 2019.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *ICLR*, 2017.

Li, J., Lavrukhin, V., Ginsburg, B., Leary, R., Kuchaiev, O., Cohen, J. M., Nguyen, H., and Gadde, R. T. Jasper: An end-to-end convolutional neural acoustic model. In *Interspeech*, 2019.

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., et al. Train large, then compress: Rethinking model size for efficient training and inference of transformers. *arXiv preprint arXiv:2002.11794*, 2020.

Liebenwein, L., Baykal, C., Lang, H., Feldman, D., and Rus, D. Provable filter pruning for efficient neural networks. In *ICLR*, 2020.

Lin, J., Chen, W.-M., Lin, Y., Cohn, J., Gan, C., and Han, S. Mcunet: Tiny deep learning on iot devices. In *NeurIPS*, 2020a.

Lin, T., Stich, S. U., Barba, L., Dmitriev, D., and Jaggi, M. Dynamic model pruning with feedback. In *ICLR*, 2020b.

Lions, P.-L. and Mercier, B. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 1979.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *ICLR*, 2019.

Louizos, C., Ullrich, K., and Welling, M. Bayesian compression for deep learning. In *NeurIPS*, 2017.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through l_0 regularization. In *ICLR*, 2018.

Lu, Z., Sindhwani, V., and Sainath, T. N. Learning compact recurrent neural networks. In *ICASSP*, 2016.

Luo, J., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *ICLR*, 2017.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 2018.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. In *ICML*, 2017.

Moreau, J.-J. Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 1962.

Mostafa, H. and Wang, X. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *ICML*, 2019.

Mozer, M. C. and Smolensky, P. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *NeurIPS*, 1989.

Neklyudov, K., Molchanov, D., Ashukha, A., and Vetrov, D. Structured bayesian pruning via log-normal multiplicative noise. In *NeurIPS*, 2017.

Oreshkin, B. N., Carpov, D., Chapados, N., and Bengio, Y. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *ICLR*, 2020.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. Librispeech: An ASR corpus based on public domain audio books. In *ICASSP*, 2015.

Park, S., Lee, J., Mo, S., and Shin, J. Lookahead: A far-sighted alternative of magnitude-based pruning. In *ICLR*, 2020.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. 2019.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

Renda, A., Frankle, J., and Carbin, M. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*, 2020.

Savarese, P., Silva, H., and Maire, M. Winning the lottery with continuous sparsification. *arXiv preprint arXiv:arXiv:1912.04427*, 2020.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

Tanaka, H., Kunin, D., Yamins, D., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

Turc, I., Chang, M.-W., Lee, K., and Toutanova, K. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*, 2019.

Ullrich, K., Meeds, E., and Welling, M. Soft weight-sharing for neural network compression. In *ICLR*, 2017.

vahid, K. A., Prabhu, A., Farhadi, A., and Rastegari, M. Butterfly transform: An efficient fft based neural architecture design. In *CVPR*, 2020.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020.

Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., et al. Espnet: End-to-end speech processing toolkit. In *Interspeech*, 2018.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *NeurIPS*. 2016.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., et al. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*, 2019.

Xiao, X., Wang, Z., and Rajasekaran, S. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *NeurIPS*. 2019.

Ye, M., Gong, C., Nie, L., Zhou, D., Klivans, A., and Liu, Q. Good subnetworks provably exist: Pruning via greedy forward selection. In *ICML*, 2020.

Yu, J., Yang, L., Xu, N., Yang, J., and Huang, T. Slimmable neural networks. In *ICLR*, 2019.

Zhu, M. and Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *ICLR*, 2018.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.