A UNIFIED FRAMEWORK OF SOFT THRESHOLD PRUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Soft threshold pruning is among the cutting-edge pruning methods with state-of-the-art performance[1]. However, previous methods either aimlessly perform searching on the threshold scheduler or simply train the threshold, lacking theoretical explanation from a unified perspective. In this work, we reformulate soft threshold pruning as an implicit optimization problem solved using the *Iterative Shrinkage-Thresholding* Algorithm (ISTA), a classic method from the fields of sparse recovery and compressed sensing. Under this theoretical framework, all threshold tuning strategies proposed in previous studies of soft threshold pruning are explained as a specific arrangement style of regularization term. We further derive an optimal threshold scheduler through an in-depth study of threshold scheduling based on our framework. This scheduler keeps L_1-regularization in the equivalent optimization problem stable and corresponds to a consistent objective function and can be, in principle, applied to sparsify any mathematical model that includes parameters trained via SGD. We conduct extensive experiments and verify its state-of-the-art performance on both Artificial Neural Networks (ResNet-50 and MobileNet-V1) and Spiking Neural Networks (SEW ResNet-18) on ImageNet datasets. It further evolves into a family of novel pruning methods, including sparsify-during-training, early pruning, and pruning at initialization via analysis based on our framework.

1 INTRODUCTION

Pruning has been a thriving area of network compression. Since the day deep neural networks stretch their tentacles to every corner of machine learning applications, the demand for shrinking the size of network parameters has never stopped growing. Fewer parameters usually imply less computing burden on resource-constrained hardware such as embedding devices or neuromorphic chips. Some pioneering studies have revealed considerable redundancies in both Artificial Neural Networks (ANNs) (Han et al., 2015; 2016; Wen et al., 2016; Liu et al., 2017) and Spiking Neural Networks (SNNs) (Qi et al., 2018; Chen et al., 2021; Yin et al., 2021; Deng et al., 2021; Kundu et al., 2021; Kim et al., 2022b).

In essence, pruning is an optimization problem under L_0 norm constraint on network parameters, which has been proved in Natarajan (1995) to be an NP-Hard problem. Relaxing L_0 norm to $L_p(0 < p < 1)$ norm will not make it more tractable for it is still strongly NP-Hard (Ge et al., 2011). Nowadays, research on pruning and sparse optimization is mainly focused on the L_1-regularized problem, the tightest convex relaxation of L_0 norm, which dates back to a series of groundbreaking studies on compressed sensing (Donoho, 2006; Candès et al., 2006). It technically allows us to solve L_1-regularized problem as an alternative or, sometimes even an equivalent option (Candès, 2008) to confront L_0 norm constraint. A variety of modern methods such as magnitude-based pruning are still firmly rooted in solving the L_1 regularized optimization problem. Be that as it may, in numerous recent papers, vanilla magnitude-based pruning is considered only as a baseline method. Meanwhile, L_1 regularization is mostly employed for shrinking the magnitude of weight before

[1]For example, STR (Kusupati et al., 2020) is the first to achieve >50% Top-1 accuracy of ImageNet on ResNet-50 under >99% sparsity. STDS (Chen et al., 2022) is the first pruning algorithm achieving acceptable performance degradation ($\sim 3\%$ under 88.8% sparsity) for spiking neural networks with 18+ layers.

the hard thresholding step, which has started to be replaced by other sorts of novel regularization (Zhuang et al., 2020).

In the past few years, a new range of pruning methods based on soft threshold reparameterization of weights has been developing gradually. Two representative ones are **S**oft **T**hreshold weight **R**eparameterization (**STR**) (Kusupati et al., 2020) and **S**tate **T**ransition of **D**endritic **S**pines (**STDS**) (Chen et al., 2022). These works adopt soft threshold mapping to map a set of hidden weights to the actual network weights. They both achieve the best performance of that time. STDS further demonstrates the analogy between soft threshold mapping and a structure in biological neural systems, *i.e.*, dendritic filopodia and mature dendritic spines. However, few researchers notice that soft threshold mapping is inherently equivalent to the shrinkage operator in the orthonormal design case of LASSO (Tibshirani, 1996), an L_1-constrained optimization problem. It further derives the *Iterative Shrinkage-Thresholding* Algorithm (ISTA) (Daubechies et al., 2004; Elad, 2006), which used to be popularized in sparse recovery and compressed sensing. ISTA has many variants (Bioucas-Dias & Figueiredo, 2007; Beck & Teboulle, 2009; Bayram & Selesnick, 2010) and has long been certified as an effective "pruning" algorithm in all sorts of fields. Despite STDS undertaking an abecedarian analysis on the scheduling of threshold and providing a preliminary result that the threshold scheduler acts similar to the iterative step of ISTA, many issues have not been fully addressed in previous soft threshold pruning literature, such as 1) the exact correspondence between ISTA and the manner of growing threshold, 2) the rationality of training threshold, 3) the underlying optimization problem, and 4) the way to improve existing methods without exhaustively trying different tricks for scheduling threshold.

In this work, we proposed a theoretical framework serving as a bridge between the underlying L_1-regularized optimization problem and threshold scheduling by viewing soft threshold pruning as an implicit executed ISTA. Specifically, we prove that the magnitude of the underlying L_1 regularized term is precisely equal to the quotient of forward finite difference of threshold scheduler function and the learning rate. To ease the burden, we use the derivative to replace the computation of finite differences and confirm it is a perfect substitute from the perspectives of sparsity and accuracy. In this way, any kind of threshold tuning strategy is interpreted as a special scheme for changing L_1 penalty. Additionally, we induce a strategy of tuning L_1 penalty called continuation strategy (Xiao & Zhang, 2012), which was once all the rage in the field of sparse optimization, and derive broad categories of algorithms covering several tracks in the present taxonomy of pruning. In summary, our contributions are as follows:

- **Theoretical cornerstone of threshold tuning strategy.** To the best of our knowledge, this is the first work interpreting increasing threshold as an ever-changing regularized term. Under theoretical analysis, we, for the first time, present a unified framework for the local equivalence of ISTA and soft threshold pruning using SGD as an optimizer. This formulation enables us to break precursory works into parts and make a comprehensive study on threshold scheduling alone using the classical method in sparse optimization.
- **Learning rate adapted threshold scheduler.** Through our proposed framework, we reveal the strong relation between the learning rate scheduler and the threshold scheduler. Then we raise the viewpoint that a stable regularized term implies a coherent pace of the foregoing two schedulers. We, therefore, proposed the *Learning rate Adapted Threshold Scheduler* (LATS), which achieves a state-of-the-art performance-sparsity tradeoff on both deep ANNs and SNNs.
- **Sibling schedulers cover multiple tracks of pruning.** We propose an early pruning algorithm via translating the homotopy continuation algorithm into a pruning algorithm with our framework. It achieves pretty close performance to LATS when used as a conventional early pruning method. Moreover, the algorithm in the pruning-at-initialization setting erases some subsequent layers in ResNet and maintains the identity mapping, shrinking deep ResNet to a shallow one.

2 Related works

There has been a deluge of pruning algorithms emerged since the term "deep compression" was invented. These various studies emphasize different points like granularity (structured or unstructured) and stage of pruning (at initialization, during training, post training). The difference in granularity is

similar to that between LASSO and group LASSO. Empirically, unstructured pruning tends to reach higher sparsity under the same accuracy degradation. For the pruning phase, sparsify during training commonly lead to higher accuracy than early phase one, *e.g.*, pruning at initialization. Moreover, pruning during training is cheaper than post-training pruning when the overhead of dense training is considered. Some most relevant works are introduced as follows.

Sparsify during training. The terminology *sparsify-during-training* (also called pruning-while-learning, pruning-during-training) is mentioned in Hoefler et al. (2021), which refers to pruning and training networks simultaneously including those iteratively pruned networks. Recent works in this area includes STR (Kusupati et al., 2020), Top-KAST (Jayakumar et al., 2020), CS (Continuous Sparsification) (Savarese et al., 2020), RigL (Evci et al., 2020), WoodFisher (Singh & Alistarh, 2020), PSGD (Kim et al., 2020), GraNet (Liu et al., 2021a), Powerprop (Schwarz et al., 2021), ProbMask (Zhou et al., 2021), GPO (Wang et al., 2022), OptG (Zhang et al., 2022) and STDS (Chen et al., 2022). Many of these works are based on the reparameterization of weights using either binary mask or element-wise nonlinear mapping. The former choose to confront L_0 constraint directly while the latter are committed to adjusting the landscape of loss function around zero. Our method is based on soft threshold reparameterization, which is piecewise linear and has an intrinsic connection to the ISTA with L_1 regularization.

Early pruning. We refer to a variant of sparsify-during-training as early pruning here, which only exerts pruning to network in the early stage of training. It includes pruning at initialization, *e.g.*, GraSP (Wang et al., 2020), SynFlow (Tanaka et al., 2020), SBP-SR (Hayou et al., 2021), ProsPr (Alizadeh et al., 2022), and the conventional early pruning methods which stop pruning at a certain time (You et al., 2020; Liu et al., 2021b; Rachwan et al., 2022). Most of these works are inspired by the discovery of Lottery Ticket Hypothesis (LTH) (Frankle & Carbin, 2019) or SNIP (Lee et al., 2019), if not both. LTH suggests we can find a sparse subnetwork with comparable performance to the dense network after iteratively retrained, while SNIP manages to train from an initial sparse network. A wide array of criteria like connectivity sensitivity are taken for finding such sparse networks in the early stage with promising accuracy after training.

3 PRELIMINARIES

Notation. We use $|\boldsymbol{x}|$ to denote the element-wise absolute value of $\boldsymbol{x}$, and $\|\boldsymbol{x}\|_p$ denotes the p-norm of $\boldsymbol{x}$. $\boldsymbol{x} \odot \boldsymbol{y}$ denotes the element-wise product of $\boldsymbol{x}$ and $\boldsymbol{y}$. If not otherwise specified, the superscript within parenthesis $\boldsymbol{x}^{(i)}$ denotes $\boldsymbol{x}$ at i-th iteration.

3.1 SOFT THRESHOLD PRUNING

Basically, soft threshold pruning relies on soft threshold reparameterization $\mathcal{S}_d : \mathbb{R}^n \to \mathbb{R}^n$ from a set of hidden weights $\boldsymbol{\theta}$ to actual weights $\boldsymbol{w}$, which is defined by the element-wise mapping $\mathcal{S}_d(\boldsymbol{x})_i := \operatorname{sign}(x_i) \cdot \max\{|x_i| - d, 0\}$ with scalar threshold d. Growing threshold d lead to shrinkage of weight magnitudes. In GPO, a convex combination of soft threshold mapping and identity mapping is introduced for reparameterization of weights. In the following discussion, STR and STDS are analyzed majorly in virtue of GPO's marginal performance improvement with respect to STR.

Gradient computing. In dense training, the loss function $\mathcal{L}(\boldsymbol{w})$ relies on the network weight $\boldsymbol{w}$. Some recent studies adopt soft threshold reparameterization $\boldsymbol{w}(\boldsymbol{\theta}, d) := \mathcal{S}_d(\boldsymbol{\theta})$, where network weight is parameterized by hidden weight $\boldsymbol{\theta}$ and threshold d. Gradient is propagated via path $\mathcal{L} \to \boldsymbol{w} \to \boldsymbol{\theta}$. The learning rule is thus used for updating $\boldsymbol{\theta}$ instead of $\boldsymbol{w}$. Note that $\mathcal{S}_d$ is non-differentiable at $\pm d$ and has zero gradient in interval $(-d, d)$. STR takes advantage of the subgradient at $\pm d$ and leaves zero gradients as it is. STDS views $\mathcal{S}_d$ as a identity mapping during backward and provides a convergence analysis by approximating $\mathcal{S}_d$ using a smooth surrogate $\hat{\mathcal{S}}_d(x) = \frac{1}{\alpha}[\zeta(\alpha(x-d)) - \zeta(-\alpha(x+d))]$, where $\zeta(x) := \log(1 + e^x)$ denotes the softplus function.

Threshold tuning. To enhance sparsity during training. Recent studies focused on manipulating d throughout learning. STR further parameterizes the threshold by another parameter $\boldsymbol{s}$ and assigns an

independent threshold for each layer. The mapping from $\boldsymbol{s}$ to threshold is individually designed for CNN and RNN. It also set $\boldsymbol{s}$ trainable and imposes L_2 regularization on $\boldsymbol{s}$ to optimize the threshold during learning. STDS manually preset a threshold scheduler function. It determines threshold $d^{(t)} = g(t/T) \cdot D$ at the t-th training iteration over all T iterations, wherein g is increasing and satisfies $g(0) = 0, g(1) = 1$. The final threshold D is also an adjustable hyperparameter for different sparsity levels.

The above techniques are summarized in Table 1. A general form of soft threshold pruning is formulated as Algorithm 1.

Algorithm 1 The general form of soft threshold pruning algorithm coupled with vanilla SGD (STR and STDS for instance).

Input: initialized network parameters $\boldsymbol{w}^{(0)}$, learning rate scheduler $\varphi(\cdot,\cdot)$, threshold scheduler $\phi(\cdot,\cdot)$, initial threshold $d^{(0)}$, initial learnable parameter s_{init}, loss function $\mathcal{L}(\boldsymbol{w})$, the number of training iterations T, L_2 penalty λ.

Output: trained sparse parameters $\boldsymbol{w}^{(T)}$

$\boldsymbol{s}^{(0)} \leftarrow s_{\text{init}}, \boldsymbol{d}^{(0)} \leftarrow \sigma(\boldsymbol{s}^{(0)}), \boldsymbol{\theta}^{(0)} \leftarrow \boldsymbol{w}^{(0)}$ ▷ Initialization

for $t = 0, 1, \ldots, T-1$ **do**

  $\eta^{(t)} \leftarrow \varphi(t, T)$ ▷ Step of learning rate scheduler

  $\Delta\boldsymbol{\theta}^{(t)} \leftarrow \nabla_{\boldsymbol{w}}(\mathcal{L}(\boldsymbol{w}^{(t)})) \odot \mathbf{1}_{|\boldsymbol{\theta}|>\boldsymbol{d}^{(t)}}$ ▷ Gradient with respect to hidden weight

  $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta^{(t)}(\Delta\boldsymbol{\theta}^{(t)} + \lambda\boldsymbol{\theta}^{(t)})$

  $\boldsymbol{s}^{(t+1)} \leftarrow \boldsymbol{s}^{(t)} - \eta^{(t)}(\nabla_{\boldsymbol{s}}(\mathcal{L}(\boldsymbol{w}; \boldsymbol{s}^{(t)})) + \lambda\boldsymbol{s}^{(t)})$ ▷ Threshold training of STR

  $\boldsymbol{d}^{(t+1)} \leftarrow \sigma(\boldsymbol{s}^{(t+1)})\phi(t+1, T)$ ▷ Update threshold

  $\boldsymbol{w}^{(t+1)} \leftarrow \mathcal{S}_{\boldsymbol{d}^{(t+1)}}(\boldsymbol{\theta}^{(t+1)})$ ▷ Update the actual weight

end for

return $\boldsymbol{w}^{(T)}$

Table 1: Techniques used in previous studies.

Method	Reparameterization mapping $\boldsymbol{w}(\boldsymbol{\theta}, d)$	Gradient of mapping	Threshold	Note
STR (Kusupati et al., 2020)	$\boldsymbol{w} = \mathcal{S}_d(\boldsymbol{\theta})$	Subgradient	$d := \begin{cases} \frac{1}{1+e^{-s}}, & \text{for CNN} \\ e^s, & \text{for RNN} \end{cases}$	s is layer-wise (global for STR-GS), trainable, and L_2 regularized
GPO (Wang et al., 2022)	$\boldsymbol{w} = (1-k)\mathcal{S}_d(\boldsymbol{\theta}) + k\boldsymbol{\theta}$	Gradient	$d := \frac{1}{1+e^{-s}}$	s, k are layer-wise, trainable, and L_2 regularized
STDS (Chen et al., 2022)	$\boldsymbol{w} = \mathcal{S}_d(\boldsymbol{\theta})$	$\mathbf{1}$ (Viewed as identity)	$d^{(t)} = g(t/T) \cdot D$	g is increasing, $g(0) = 0, g(1) = 1$

3.2 Iterative Shirinkage-Thresholding Algorithm

Consider a regularized optimization problem

$$\min_{\boldsymbol{x} \in \mathbb{R}^n} \{F(\boldsymbol{x}) := f(\boldsymbol{x}) + r(\boldsymbol{x})\} \tag{1}$$

with assumptions listed below

1. Objective function $f : \mathbb{R}^n \to \mathbb{R}$ is continuous differentiable, and L-smooth, *i.e.*, $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L_f \|\boldsymbol{x} - \boldsymbol{y}\|_2, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, where $L_f > 0$ is the Lipschitz constant of ∇f.
2. Regularization function $r : \mathbb{R}^n \to \mathbb{R}$ is continuous convex and can be nonsmooth.
3. F is bounded from below.

Leave the objective alone, applying vanilla SGD to f can be viewed as iteratively calculating proximal regularization (Martinet, 1970) of the linearized f at $\boldsymbol{x}$. In other words, for a given point $\boldsymbol{x}$, $F(\boldsymbol{y})$ can be approximated by expanding f to the quadratic term

$$\hat{F}_\eta(\boldsymbol{y}; \boldsymbol{x}) := f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x}) \rangle + \frac{1}{2\eta}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2 + r(\boldsymbol{y}), \eta > 0, \tag{2}$$

which has a unique minimizer

$$\arg\min_{\boldsymbol{y}} \hat{F}_\eta(\boldsymbol{y};\boldsymbol{x}) = \arg\min_{\boldsymbol{y}} \left\{ \frac{1}{2\eta}\|\boldsymbol{y} - (\boldsymbol{x} - \eta\nabla f(\boldsymbol{x}))\|_2^2 + r(\boldsymbol{y}) \right\}. \tag{3}$$

Note that $\eta = \frac{1}{L_f}$ gives a upperbound of $f(\boldsymbol{y})$, which is obtained by a property of L_f-smoothness: $f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \langle \boldsymbol{y} - \boldsymbol{x}, \nabla f(\boldsymbol{x})\rangle + \frac{L_f}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_2^2$. It implies with proper choice of η, we can optimize the upperbound of $F(\boldsymbol{y})$ using minimizer $\arg\min_{\boldsymbol{y}} \hat{F}_\eta(\boldsymbol{y};\boldsymbol{x})$.

The general form of ISTA iteratively solves Eq. 3 as

$$\boldsymbol{x}^{(t+1)} = \arg\min_{\boldsymbol{x}} \hat{F}_\eta(\boldsymbol{x};\boldsymbol{x}^{(t)}), \tag{4}$$

which is also known as proximal gradient methods (Combettes & Wajs, 2005). The detailed convergence analysis is presented in vast optimization literature like FISTA (Beck & Teboulle, 2009) and GIST (Gong et al., 2013). For sparse learning, we are interested in L_1 regularization term $r(\boldsymbol{x}) = \mu\|\boldsymbol{x}\|_1, \mu > 0$. Since L_1 norm is separable, Eq. 4 has a closed-form solution as

$$\boldsymbol{x}^{(t+1)} = \mathcal{S}_{\mu\eta}(\boldsymbol{x}^{(t)} - \eta\nabla f(\boldsymbol{x}^{(t)})). \tag{5}$$

Eq. 5 gives the ISTA update rule under L_1 regularization.

4 A FRAMEWORK FOR SOFT THRESHOLD PRUNING

In this part, we will formulate the growing threshold under soft threshold reparameterization as an implicit ISTA. For simplicity, we assume the threshold is global across all parameters, which is consistent with the setting of STDS and STR-GS, *i.e.*, the global threshold version of STR. Hence, we use scalar d rather than vector $\boldsymbol{d}$ in Algorithm 1 to denote the global threshold.

To begin with, we investigate the update rule of nonzero components in actual weight

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta^{(t)}\nabla_{\boldsymbol{w}}(\mathcal{L}(\boldsymbol{w}^{(t)})) \odot \nabla_{\boldsymbol{\theta}}(\boldsymbol{w}(\boldsymbol{\theta}^{(t)}, d^{(t)})), \tag{6}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \mathcal{S}_{d^{(t+1)}}(\boldsymbol{\theta}^{(t+1)}). \tag{7}$$

Assuming the sign of weight remains unchanged after an update, which happens when the gradient has the opposite sign of weight or the gradient magnitude is sufficiently small, we have the following Lemma:

Lemma 1 (Local update rule)**.** *The update rule in Eq. 6 and Eq. 7 implies the following update of a nonzero component $w(\theta, d)$ in actual weight $\boldsymbol{w}$*

$$w^{(t+1)} = \mathcal{S}_{d^{(t+1)} - d^{(t)}}(w^{(t)} - \eta^{(t)}\nabla_w(\mathcal{L}(w^{(t)}))). \tag{8}$$

The formal version and proof of Lemma 1 is given in Appendix A. Note that the form in Eq. 8 is equivalent to ISTA update rule with threshold equal to forward finite difference $d^{(t+1)} - d^{(t)}$. Recall Eq. 5, the corresponding optimization problem can be deduced as Theorem 1:

Theorem 1. *Let $\mathcal{L}(\boldsymbol{w})$ be the loss function depending on the network weight $\boldsymbol{w}$, which is further reparamterized by hidden weight $\boldsymbol{\theta}$ and threshold d as $\boldsymbol{w}(\boldsymbol{\theta}, d) = \mathcal{S}_d(\boldsymbol{\theta})$. When applying vanilla SGD to reparameterized nonzero weight, the update rule is locally equivalent to solving the following problem using ISTA with penalty term*

$$\min_{\boldsymbol{w}} \left\{ F(\boldsymbol{w}) := \mathcal{L}(\boldsymbol{w}) + \frac{d^{(t+1)} - d^{(t)}}{\eta^{(t)}}\|\boldsymbol{w}\|_1 \right\}, \tag{9}$$

where $\eta^{(t)}, d^{(t)}$ denotes the learning rate and threshold at the t-th iteration, respectively. The magnitude of L_1-penalty is the quotient of forward finite difference of threshold scheduler divided by learning rate.

Theorem 1 serves as the glue holding the learning rate scheduler $\eta^{(t)}$, threshold $d^{(t)}$ and penalty $\mu^{(t)}$ together. It also to some extent explains an increasing threshold is necessary for a positive penalty term. Under this framework, we explain in Appendix B that the original STDS uses an improper threshold scheduler. Furthermore, an analysis of threshold learning is given in Appendix C.

5 FINDING OPTIMAL THRESHOLD SCHEDULER

In this part, we are devoted to evaluating some threshold schedulers based on our theoretical framework and literature on sparse optimization. With the framework, any past strategy of tuning L_1 penalty can be converted to a feasible pruning algorithm today.

5.1 LEARNING RATE ADAPTED THRESHOLD SCHEDULER

If the optimization problem in Theorem 1 is fixed, or in other words, the L_1 penalty is invariant during learning, we can deduce a unique threshold scheduler **LATS**.

Corollary 1 (**L**earning rate **A**dapted **T**hreshold **S**cheduler)**.** *For fixed L_1 regularized problem*

$$\min_{\boldsymbol{w}} \left\{ F(\boldsymbol{w}) := \mathcal{L}(\boldsymbol{w}) + \mu \|\boldsymbol{w}\|_1 \right\}, \mu > 0, \tag{10}$$

the threshold scheduler is governed by the learning rate scheduler, that is

$$d^{(t)} = d^{(0)} + \mu \sum_{i=0}^{t-1} \eta^{(i)}. \tag{11}$$

For the most frequently used testbed, ResNet-50 (He et al., 2016) on ImageNet dataset (Deng et al., 2009), researchers usually use the cosine annealing learning rate scheduler with $\eta_{\min} = 0$. Assuming the initial threshold is zero, simple algebra shows the definition of corresponding LATS with a slight abuse of notation

$$d^{(n,b)} = \mu \eta_{\max} \left[\frac{B}{4} \left(2n + 1 + \frac{\sin\left(\frac{2n-1}{2N}\pi\right)}{\sin\frac{\pi}{2N}} \right) + \frac{b}{2}\left(1 + \cos\frac{n\pi}{N}\right) \right], \tag{12}$$

where the threshold $d^{(n,b)}$ depends on epoch id n and batch id b. Here $n = 0, 1, \ldots, N-1$ denotes the current id of training epoch, and $b = 1, 2, \ldots, B$ denotes the batch id in n-th epoch.

In effect, painstakingly coding LATS according to a learning rate scheduler is not inevitable. To ease the computing burden and erase the accumulative error of learning rate, we turn to a simplified threshold scheduler by replacing the summation of learning rate with integration and increasing the update frequency to per-iteration. Assuming the learning rate scheduler can be expressed by $\eta^{(n,b)} = h(n/N)$, the simplified threshold scheduler is defined by

$$d^{(t)} = d^{(N-1,B)} \cdot \frac{\int_0^{\frac{t}{T}} h(x)\mathrm{d}x}{\int_0^1 h(x)\mathrm{d}x}. \tag{13}$$

Now we have the **Simplified LATS** (**S-LATS** for short)

$$d^{(t)} = \frac{\mu\eta_{\max}T}{2} \cdot \frac{\int_0^{\frac{t}{T}} \frac{1}{2}(1+\cos\pi x)\mathrm{d}x}{\int_0^1 \frac{1}{2}(1+\cos\pi x)\mathrm{d}x} = \frac{\mu\eta_{\max}T}{2}\left[\frac{1}{\pi}\sin(\frac{t\pi}{T}) + \frac{t}{T}\right]. \tag{14}$$

It can be interpreted with scheduler form in STDS as $d^{(t)} = g(t/T) \cdot D$, where $g_{\text{S-LATS}}(x) = \sin(\pi x)/\pi + x$ and $D = \mu\eta_{\max}T/2$. The final threshold $D \propto \mu$ controls the overall strength of penalty. Tuning D is thus akin to changing the magnitude of penalty. In the following discussion, we employ the final threshold D instead of $d^{(N-1,B)}$ to lighten the notation. The threshold schedulers in the rest of this work will be expressed in a unified form of $d^{(t)} = g(t/T) \cdot D$.

S-LATS has almost the same accuracy and sparsity performance as the original LATS under identical final thresholds $D = 0.1, 0.5, 1.0, 5.0$, which can be gleaned from Tab. 2. Thus, we turn to simplified threshold scheduler in the following discussion.

5.2 CONTINUATION STRATEGY

Also known as "warm starting", *continuation strategy* is designated for accelerating convergence. Similar to annealing of learning rate, continuation refers to gradually reducing L_1 penalty during learning. It is also explained in Hale et al. (2008) as an analogue to the homotopy algorithms in statistics. Continuation method used to serve as a common trick in abundant classic literature concerning sparse optimization including GPSR (Figueiredo et al., 2007), fixed point continuation method (Hale et al., 2008), SpaRSA (Wright et al., 2009) and NESTA (Becker et al., 2011).

Table 2: Comparison of LATS and S-LATS when applied to ResNet-50 on ImageNet dataset.

Final threshold D	**STDS + LATS**		**STDS + S-LATS**	
	Sparsity (%)	Top-1 Acc. (%)	Sparsity (%)	Top-1 Acc. (%)
0.1	79.97	76.53	79.95	76.75
0.5	95.54	73.12	95.53	73.03
1.0	97.43	69.56	97.43	69.64
5.0	99.27	53.80	99.28	53.64

5.2.1 PGH SCHEDULER

In the series works of *proximal gradient homotopy* (PGH) (Xiao & Zhang, 2012; 2013; Lin & Xiao, 2014), the researchers provide proof of geometric convergence rate when inducing exponentially decaying regularization $\mu^{(t)} = \beta^{\frac{t}{T}}$, where $0 < \beta < 1$. Considering our formulation of soft threshold pruning in Theorem 1, PGH can be translated into the **PGH scheduler** (simplified using Eq.13), which can be written as

$$d^{(t)} = g_{\text{PGH}}(t/T) \cdot D := \frac{\int_0^{\frac{t}{T}} \frac{1}{2}(1+\cos \pi x)\beta^x \mathrm{d}x}{\int_0^1 \frac{1}{2}(1+\cos \pi x)\beta^x \mathrm{d}x} \cdot D, \tag{15}$$

where the analytic form of g_{PGH} is shown below

$$g_{\text{PGH}}(x) = \frac{\pi^2\left(\beta^x - 1\right) + \log^2(\beta)\left(\beta^x - 2\right) + \log(\beta)\beta^x(\log(\beta)\cos(\pi x) + \pi \sin(\pi x))}{\pi^2(\beta - 1) - 2\log^2(\beta)}. \tag{16}$$

5.2.2 LINK TO EARLY PRUNING

For the PGH scheduler, we interpolate β between 0 and 1 and get a series of PGH schedulers. In Fig. 1, one can easily find that PGH schedulers tend to increase threshold slower and slower along with training process, which is caused by decreasing regularization term. For $0 < \beta < 1$, if we set a threshold for such regularization and ignore pruning when its magnitude is below a preset threshold, we get a family of early pruning algorithms. They stop pruning at different stages. Unlike the stereotyped early pruning algorithm using static masks after the pruning stage, PGH scheduler uses a dynamic mask, enabling "dead" weights to be revived when the pruning stage ends. It indicates that the sparsity may drop in the sheer training stage. Recall that the regularized term is proportional to the forward finite difference and thus can be approximated by derivative, for conventional early pruning, we regard $g'_{\text{PGH}}(t/T) < 0.1$ as the termination criterion of pruning.

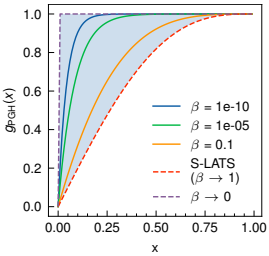

Figure 1: PGH schedulers under different choice of β. The union set of these scheduler functions are illustrated in lighter blue area.

Note that there are two limit cases when β approaches 0 or 1. It is obvious that $\beta \to 1$ leads to S-LATS for no decay is applied to regularization. When $\beta \to 0$, the regularization is zero except for the initial one. In this case, PGH scheduler degenerates to the magnitude-based pruning after initialization, which is followed by a normal training stage. This is also referred to as *sparse-to-sparse training* or *pruning at initialization* method. The dynamic mask property and soft threshold mapping still preserve after the pruning stage. Magnitude-based pruning is venturesome when the threshold is manually tweaked since an inappropriately large threshold will prune all weights in a layer. In feed-forward neural networks, a zeroed layer not merely blocks information from the previous layer but feeds a zeroed tensor to the next layer, leading to zeroed gradient to both upstream and downstream layers. These layers hence stop learning and get gradually pruned by weight decay, which is what we call the "zeroed layer" phenomenon. Despite such prior knowledge about neural networks, we determined to remain our algorithm unchanged to guarantee universality.

6 EXPERIMENTS

In this section, we test our proposed threshold scheduler S-LATS on both deep ANNs and SNNs. The favorable performances against the previous studies are confirmed. We also tune hyperparameter β in PGH scheduler to maintain different phases of pruning algorithm.

6.1 S-LATS

We test STDS integrated with S-LATS on various sparsity scenarios via tuning final threshold D. It achieves state-of-the-art performances on both ANNs (ResNet-50, MobileNet-V1) and SNNs (SEW ResNet-18 (Fang et al., 2021)). The results on ResNet-like networks and MobileNet-V1 are illustrated in Fig. 2 and Appendix D, respectively. In particular, it outperforms the default threshold scheduler of STDS, namely the sine scheduler, on SEW ResNet. Notably, our method surges ahead of all the other baselines under $<98\%$ sparsity for ANN experiments. It should be noted that the origin STDS excludes the last FC layer from pruning in SNNs, while the results reported here are shown by rerunning it with the last layer pruned. Our ANN experiments utilize standard training setting with the batch size of 256 and 100 training epochs to keep the comparison fair. We also admit it fails to achieve comparable performance with a few baselines like OptG and in extreme sparsity ($\geq 99\%$), which suggests the theory might be imperfect under such condition.

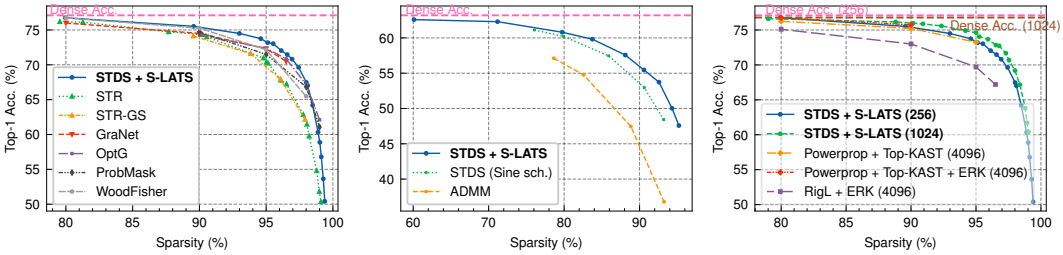

Figure 2: Performance of several SOTA pruning strategies of **ResNet-50 (Left & Right)** and **SEW ResNet-18 (Middle)** on ImageNet. The left figure includes trials using standard training setting (256 batch size). The right one use enlarged batch size, marked in parenthesis.

Table 3: Comparison of ResNet-50 Top-1 accuracy on ImageNet under typical sparsity settings in recent studies using 100 training epochs. Note that some methods do not allow precise control of sparsity and we thus choose the trials with the closest sparsity, which is specified behind the accuracy.

Method	Batch size	Sparsity						
		80%	90%	95%	96.5%	97.5%	98%	99%
STR	256	76.19 (79.55)	74.31 (90.23)	70.40 (95.03)	67.22 (96.53)	-	61.46 (98.05)	51.82 (98.98)
STR-GS	256	-	74.13 (89.54)	-	-	-	62.17 (97.91)	-
GraNet	256	76	74.5	72.3	70.5	-	-	-
ProbMask	256	-	74.68	71.5	-	-	66.83	61.07
OptG	256	-	74.28	72.38	70.85	-	67.2	**62.1**
WoodFisher	256	76.73	75.26	72.16	-	-	65.47	-
STDS + S-LATS	256	**76.75** (79.95)	**75.52** (89.57)	**73.18** (95.12)	**71.47** (96.50)	**69.64** (97.43)	**67.47** (98.01)	58.88 (99.02)
STDS + S-LATS	1024	**76.61** (79.00)	**75.97** (90.15)	**74.61** (95.01)	**72.84** (96.53)	**70.68** (97.54)	**69.20** (98.00)	**61.72** (98.93)
RigL (ERK)	4096	75.1	73.0	69.7	67.2	-	-	-
Top-KAST (Powerprop)	4096	76.24	75.23	73.25	-	-	-	-
Top-KAST (Powerprop+ERK)	4096	76.76	75.74	-	-	-	-	-

Pruning using large batch size. Some recent studies such as Powerprop (Schwarz et al., 2021) adopt batch size of 4096 for pruning of ResNet-50, which is much larger than standard training setting (256 batch size). They both achieve fascinating performances under high sparsity. Hence, we verify the influence of large batch size. Due to limited computing resources, we only increase it to 1024 and also enlarge the learning rate correspondingly. The right of Fig. 2 shows the enlarged batch size indeed leads to higher performance for STDS + S-LATS, which outperforms all of SOTA studies. Astonishingly, we observe that even though the performance of dense network slightly degrades from 77.1 to 76.78, the accuracy of the sparse ones is improved overall. Therefore, we attribute such performance improvement to the higher affinity to pruning when using large batch

size. On the basis of the above finding, we believe applying an even larger batch size, like 4096 used in Powerprop, to our method will lead to top-notch accuracy under all levels of sparsity.

6.2 PGH SCHEDULER

Conventional early pruning Our experiments of early pruning includes $\beta = 0.1, 10^{-5}, 10^{-10}$. The corresponding ending criteria $t/T = 0.743, 0.382, 0.231$ are given by numerical solution. For 100 training epochs, it indicates pruning roughly stops at the 74th, the 38th and the 23th epoch. The network using PGH scheduler with smaller β converges faster to a sparse one, which is illustrated in the left of Fig. 3. Surprisingly, we find in the middle of Fig. 3 that for different β, the datapoint

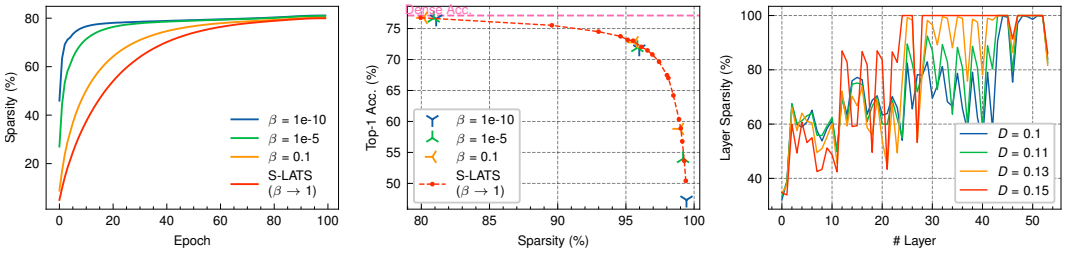

Figure 3: Overall sparsity during learning when final threshold $D = 0.1$ (**Left**). Performance under different sparsity levels (**Middle**). Layerwise sparsity of PGH scheduler under pruning at initilization setting $\beta \to 0$ (**Right**).

of accuracy against sparsity almost lies on the curve of S-LATS. It suggests these schedulers have practically the same performances as S-LATS under same sparsity levels, and the fast convergence to sparse network brought by PGH scheduler does not come along with degraded performance. With the help of PGH scheduler, we are able to find sparse network earlier with negligible performance degradation.

Pruning at initialization We also try $\beta \to 0$, which refers to increasing the threshold to its maximum at the first iteration. Since our method is agnostic about the object of pruning, the "zeroed layer" phenomenon mentioned earlier is unavoidable and observed in the right of Fig. 3, wherein three consecutive layers within a residual block tend to be pruned simultaneously. However, owing to the skip connection in ResNet, the feature can pass through shortcuts to the final FC layer, and thus most layers are still normally trained. The aforementioned results are collected in Tab. 4. Detailed layerwise sparsity is shown in Tab. 13 of Appendix G.

Table 4: Top-1 accuracy, overall sparsity and the number of zeroed layers in pruning at initialization experiments.

Final threshold D	0.1	0.11	0.13	0.15
Overall sparsity (%)	87.16	90.00	93.11	95.64
Top-1 Acc. (%)	74.69	72.89	68.23	62.22
# Zeroed layers	0	9	9	27

7 CONCLUSION & DISCUSSION

In this work, we present a framework interpreting increasing threshold as a constantly changing penalty term and reveal the underlying connection between soft threshold pruning and ISTA. We also derive a couple of threshold schedulers, which achieve comparable performance to current SOTA works and cover multiple tracks of pruning. It is worth noting that our method is agnostic about the object of pruning. This design on the one hand endows our method with versatility, but on the other treats weight wheresoever equally, becoming totally ignorant of nowadays pruning researches like sparsity budget allocation, *e.g.*, *Erdős-Rényi* (Mocanu et al., 2018) and *Erdős-Rényi-Kernel (ERK)* (Evci et al., 2020), or commonsense in this area like "Do not prune all parameters within any single layer". We believe our method can for sure further benefit from knowledge in the prosperous field of network pruning.

REFERENCES

Milad Alizadeh, Shyam A. Tailor, Luisa M Zintgraf, Joost van Amersfoort, Sebastian Farquhar, Nicholas Donald Lane, and Yarin Gal. Prospect pruning: Finding trainable weights at initialization using meta-gradients. In *International Conference on Learning Representations*, 2022.

Ilker Bayram and Ivan W. Selesnick. A subband adaptive iterative shrinkage/thresholding algorithm. *IEEE Transactions on Signal Processing*, 58(3):1131–1143, 2010.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

Stephen Becker, Jérôme Bobin, and Emmanuel J. Candès. NESTA: A fast and accurate first-order method for sparse recovery. *SIAM Journal on Imaging Sciences*, 4(1):1–39, 2011.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

JosÉ M. Bioucas-Dias and MÁrio A. T. Figueiredo. A new TwIST: Two-step iterative shrinkage/thresholding algorithms for image restoration. *IEEE Transactions on Image Processing*, 16 (12):2992–3004, 2007.

Emmanuel J. Candès. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique*, 346(9):589–592, 2008.

Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006.

Yanqi Chen, Zhaofei Yu, Wei Fang, Tiejun Huang, and Yonghong Tian. Pruning of deep spiking neural networks through gradient rewiring. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 1713–1721, 2021.

Yanqi Chen, Zhaofei Yu, Wei Fang, Zhengyu Ma, Tiejun Huang, and Yonghong Tian. State transition of dendritic spines improves learning of sparse spiking neural networks. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 3701–3715, 2022.

Patrick L. Combettes and Valérie R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

Ingrid Daubechies, Michel Defrise, and Christine De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.

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.

Lei Deng, Yujie Wu, Yifan Hu, Ling Liang, Guoqi Li, Xing Hu, Yufei Ding, Peng Li, and Yuan Xie. Comprehensive snn compression using admm optimization and activity regularization. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2021.

David L Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

Michael Elad. Why simple shrinkage is still relevant for redundant representations? *IEEE Transactions on Information Theory*, 52(12):5559–5569, 2006.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 2943–2952, 2020.

Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks. In *Advances in Neural Information Processing Systems*, pp. 21056–21069, 2021.

MÁrio A. T. Figueiredo, Robert D. Nowak, and Stephen J. Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE Journal of Selected Topics in Signal Processing*, 1(4):586–597, 2007.

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

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

Dongdong Ge, Xiaoye Jiang, and Yinyu Ye. A note on the complexity of L_p minimization. *Mathematical programming*, 129(2):285–299, 2011.

Pinghua Gong, Changshui Zhang, Zhaosong Lu, Jianhua Huang, and Jieping Ye. A general iterative shrinkage and thresholding algorithm for non-convex regularized optimization problems. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 37–45, 2013.

Elaine T. Hale, Wotao Yin, and Yin Zhang. Fixed-point continuation for ℓ_1-minimization: Methodology and convergence. *SIAM Journal on Optimization*, 19(3):1107–1130, 2008.

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*, 2015.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*, 2016.

Soufiane Hayou, Jean-Francois Ton, Arnaud Doucet, and Yee Whye Teh. Robust pruning at initialization. In *International Conference on Learning Representations*, 2021.

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 (CVPR)*, 2016.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

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

Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. In *Advances in Neural Information Processing Systems*, pp. 20744–20754, 2020.

Jangho Kim, KiYoon Yoo, and Nojun Kwak. Position-based scaled gradient for model quantization and pruning. In *Advances in Neural Information Processing Systems*, pp. 20415–20426, 2020.

Youngeun Kim, Yuhang Li, Hyoungseob Park, Yeshwanth Venkatesha, and Priyadarshini Panda. Neural architecture search for spiking neural networks. *arXiv preprint arXiv:2201.10355*, 2022a.

Youngeun Kim, Yuhang Li, Hyoungseob Park, Yeshwanth Venkatesha, Ruokai Yin, and Priyadarshini Panda. Lottery ticket hypothesis for spiking neural networks. *arXiv preprint arXiv:2207.01382*, 2022b.

Souvik Kundu, Gourav Datta, Massoud Pedram, and Peter A. Beerel. Spike-thrift: Towards energy-efficient deep spiking neural networks by limiting spiking activity via attention-guided compression. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 3953–3962, 2021.

Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 5544–5555, 2020.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*, 2019.

Qihang Lin and Lin Xiao. An adaptive accelerated proximal gradient method and its homotopy continuation for sparse optimization. In *Proceedings of the 31st International Conference on Machine Learning*, pp. 73–81, 2014.

Shiwei Liu, Tianlong Chen, Xiaohan Chen, Zahra Atashgahi, Lu Yin, Huanyu Kou, Li Shen, Mykola Pechenizkiy, Zhangyang Wang, and Decebal Constantin Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. In *Advances in Neural Information Processing Systems*, pp. 9908–9922, 2021a.

Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? in-time over-parameterization in sparse training. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 6989–7000, 2021b.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, 1997.

Bernard Martinet. Régularisation d'inéquations variationnelles par approximations successives. *Revue Francaise d'informatique et de Recherche operationelle*, 4:154–158, 1970.

Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

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

Byunggook Na, Jisoo Mok, Seongsik Park, Dongjin Lee, Hyeokjun Choe, and Sungroh Yoon. AutoSNN: Towards energy-efficient spiking neural networks. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 16253–16269, 2022.

Balas Kausik Natarajan. Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2):227–234, 1995.

Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks. *IEEE Signal Processing Magazine*, 36(6):51–63, 2019.

Yu Qi, Jiangrong Shen, Yueming Wang, Huajin Tang, Hang Yu, Zhaohui Wu, and Gang Pan. Jointly learning network connections and link weights in spiking neural networks. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 1597–1603, 2018.

John Rachwan, Daniel Zügner, Bertrand Charpentier, Simon Geisler, Morgane Ayle, and Stephan Günnemann. Winning the lottery ahead of time: Efficient early network pruning. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 18293–18309, 2022.

Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. In *Advances in Neural Information Processing Systems*, pp. 11380–11390, 2020.

Jonathan Schwarz, Siddhant Jayakumar, Razvan Pascanu, Peter E Latham, and Yee Teh. Power-propagation: A sparsity inducing weight reparameterisation. In *Advances in Neural Information Processing Systems*, pp. 28889–28903, 2021.

Sidak Pal Singh and Dan Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. In *Advances in Neural Information Processing Systems*, pp. 18098–18109, 2020.

Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems*, pp. 6377–6389, 2020.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

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

Xiaodong Wang, Xianxian Zeng, Yun Zhang, Dong Li, and Weijun Yang. Learning soft threshold for sparse reparameterization using gradual projection operators. *Neurocomputing*, 488:381–390, 2022.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2082–2090, 2016.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Stephen J. Wright, Robert D. Nowak, and MÁrio A. T. Figueiredo. Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493, 2009.

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in Neuroscience*, 12:331, 2018.

Lin Xiao and Tong Zhang. A proximal-gradient homotopy method for the l1-regularized least-squares problem. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 839–846, 2012.

Lin Xiao and Tong Zhang. A proximal-gradient homotopy method for the sparse least-squares problem. *SIAM Journal on Optimization*, 23(2):1062–1091, 2013.

Zeke Xie, Issei Sato, and Masashi Sugiyama. A diffusion theory for deep learning dynamics: Stochastic gradient descent exponentially favors flat minima. In *International Conference on Learning Representations*, 2021.

Hang Yin, John Boaz Lee, Xiangnan Kong, Thomas Hartvigsen, and Sihong Xie. Energy-efficient models for high-dimensional spike train classification using sparse spiking neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2017–2025, 2021.

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

Yuxin Zhang, Mingbao Lin, Mengzhao Chen, Zihan Xu, Fei Chao, Yunhan Shen, Ke Li, Yongjian Wu, and Rongrong Ji. Optimizing gradient-driven criteria in network sparsity: Gradient is all you need. *arXiv preprint arXiv:2201.12826*, 2022.

Xiao Zhou, Weizhong Zhang, Hang Xu, and Tong Zhang. Effective sparsification of neural networks with global sparsity constraint. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3599–3608, 2021.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. In *Advances in Neural Information Processing Systems*, pp. 9865–9877, 2020.

A PROOF OF THEOREMS AND LEMMAS

For clarity, we restate the theorem or lemma in the main text again here.

Lemma 1 (Local update rule)**.** *The update rule below*

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta^{(t)} \nabla_{\boldsymbol{w}}(\mathcal{L}(\boldsymbol{w}^{(t)})) \odot \nabla_{\boldsymbol{\theta}}(\boldsymbol{w}(\boldsymbol{\theta}^{(t)}, d^{(t)})) \tag{17}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \mathcal{S}_{d^{(t+1)}}(\boldsymbol{\theta}^{(t+1)}) \tag{18}$$

imply the following update of any nonzero component $w(\theta, d)$ in actual weight $\boldsymbol{w}$

$$w^{(t+1)} = \mathcal{S}_{d^{(t+1)}-d^{(t)}}(w^{(t)} - \eta^{(t)} \nabla_w(\mathcal{L}(w^{(t)}))). \tag{19}$$

when $|\theta^{(t+1)}| > d^{(t)}$ and the sign condition $\operatorname{sign}(\theta^{(t+1)}) = \operatorname{sign}(\theta^{(t)})$ are met.

Proof. For any nonzero weight $w^{(t)} \neq 0$, $\theta^{(t)} = w^{(t)} + d^{(t)} \operatorname{sign}(w^{(t)})$, using Eq. 17 we have

$$\theta^{(t+1)} = w^{(t)} + d^{(t)} \operatorname{sign}(w^{(t)}) - \eta^{(t)} \nabla_w(\mathcal{L}(w^{(t)})) \tag{20}$$

Let $\bar{w}^{(t+1)} := w^{(t)} - \eta^{(t)} \nabla_w(\mathcal{L}(w^{(t)}))$ be the target point of vanilla SGD without regularization. Recall Eq. 20, we have

$$\begin{aligned}
\bar{w}^{(t+1)} &= \theta^{(t+1)} - d^{(t)} \operatorname{sign}(w^{(t)}) \\
&= \operatorname{sign}(\theta^{(t+1)})|\theta^{(t+1)}| - \operatorname{sign}(\theta^{(t)}) d^{(t)} \\
&= \operatorname{sign}(\theta^{(t+1)})|\theta^{(t+1)}| - \operatorname{sign}(\theta^{(t+1)}) d^{(t)} \\
&= \operatorname{sign}(\theta^{(t+1)})(|\theta^{(t+1)}| - d^{(t)}),
\end{aligned} \tag{21}$$

which has the same sign as $\theta^{(t+1)}$. Now we have $\operatorname{sign}(w^{(t+1)}) = \operatorname{sign}(\theta^{(t+1)}) = \operatorname{sign}(\theta^{(t)}) = \operatorname{sign}(w^{(t)}) = \operatorname{sign}(\bar{w}^{(t+1)})$.

To evaluate the updated weight, by Eq. 18, Eq. 20, Eq. 21 and the definition of soft threshold mapping, we derive

$$\begin{aligned}
w^{(t+1)} &= \operatorname{sign}(\theta^{(t+1)}) \max\{|w^{(t)} + d^{(t)} \operatorname{sign}(w^{(t)}) - \eta^{(t)} \nabla_w(\mathcal{L}(w^{(t)}))| - d^{(t+1)}, 0\} \\
&= \operatorname{sign}(\theta^{(t+1)}) \max\{|\bar{w}^{(t+1)} + d^{(t)} \operatorname{sign}(\bar{w}^{(t+1)})| - d^{(t+1)}, 0\} \\
&= \operatorname{sign}(\theta^{(t+1)}) \max\{|\operatorname{sign}(\bar{w}^{(t+1)})(|\bar{w}^{(t+1)}| + d^{(t)})| - d^{(t+1)}, 0\} \\
&= \operatorname{sign}(\theta^{(t+1)}) \max\{|\bar{w}^{(t+1)}| + d^{(t)} - d^{(t+1)}, 0\} \\
&= \operatorname{sign}(\bar{w}^{(t+1)}) \max\{|\bar{w}^{(t+1)}| - (d^{(t+1)} - d^{(t)}), 0\} \\
&= \mathcal{S}_{d^{(t+1)}-d^{(t)}}(w^{(t)} - \eta^{(t)} \nabla_w(\mathcal{L}(w^{(t)}))).
\end{aligned} \tag{22}$$

□

B ORIGINAL THRESHOLD SCHEDULER IN STDS

In STDS, the authors propose the Sine scheduler $d^{(t)} = \frac{1}{2}(1+\sin(\pi(\frac{t}{T}-\frac{1}{2})))D = \frac{1}{2}(1-\cos(\frac{t\pi}{T}))D$. With the form of simplified threshold scheduler in Eq. 13, by Theorem 1, the corresponding penalty $\mu^{(t)}$ has the form

$$\begin{aligned}
\mu^{(t)} &= \frac{d^{(t+1)} - d^{(t)}}{\eta^{(t)}} \\
&= D \cdot \frac{\cos(\frac{t\pi}{T}) - \cos(\frac{(t+1)\pi}{T})}{\eta_{\max}(1 + \cos\frac{t\pi}{T})} \\
&= \frac{2D \sin\frac{\pi}{T}}{\eta_{\max}} \cdot \frac{\sin(\frac{t\pi}{T} + \frac{\pi}{2T})}{1 + \cos\frac{t\pi}{T}} \\
&\approx C \cdot \frac{\sin(\frac{t\pi}{T})}{1 + \cos\frac{t\pi}{T}} \\
&= C \cdot \tan(\frac{t\pi}{2T})
\end{aligned} \tag{23}$$

It is a function of training progress t/T with constant C. Investigate the function $\tan(x/2)$ on interval $(0, \pi)$, we have $\mu(x)$ is increasing from 0 to $+\infty$, which implies μ can be sufficiently large during training. The loss is thus insignificant compared to regularization term in the last stage of training and lead to performance degradation with respect to S-LATS.

C Discussion about training threshold

In the main text we propose a framework explaining style of growth in threshold as a everchanging optimization problem. However, we only cover manually designed threshold scheduler. We make a discussion here and show training threshold is not as easy as STR do. Moreover, we suggest not to simply set threshold trainable if one really need to investigate threshold tuning.

C.1 L_2 penalty dominates early training of STR.

In the official codebase of STR, we notice the trainable sparse threshold is also together with weight decay (L_2 regularization $\lambda\|\boldsymbol{s}\|_2^2$). λ is of magnitude around $10^{-5} \sim 10^{-4}$. The initial value s_{init} is usually set to negative number with large magnitude around $-10^4 \sim -10^3$ for CNN trials. It is easy to see the magnitude of L_2 regularized term is around $0.01 \sim 1$ in the early stage of training.

Take the CNN for instance. For given loss function $\mathcal{L}$ and weight $\boldsymbol{w}_l$ of the l-th layer, the gradient passed to threshold s_l can be estimated as

$$\begin{aligned}
\nabla_{s_l}\mathcal{L}(\boldsymbol{w}_l(s_l, \boldsymbol{\theta}_l)) &= \left\langle \nabla_{\boldsymbol{w}_l}\mathcal{L}(\boldsymbol{w}_l), \left(\frac{\partial \boldsymbol{w}_l}{\partial s_l}\right)^\top \right\rangle \\
&= \sum_{(w,\theta)\in(\boldsymbol{w}_l,\boldsymbol{\theta}_l)} \nabla_w \mathcal{L}(\boldsymbol{w}_l) \cdot \nabla_{s_l}(\mathcal{S}_{\sigma(s_l)}(\theta)) \\
&= -\sigma(s_l)(1-\sigma(s_l)) \sum_{\substack{(w,\theta)\in(\boldsymbol{w}_l,\boldsymbol{\theta}_l)\\ w\neq 0}} \nabla_w \mathcal{L}(\boldsymbol{w}_l) \cdot \operatorname{sign}(\theta)
\end{aligned} \tag{24}$$

wherein each term of sum has magnitude less than $\frac{1}{4}\nabla_w\mathcal{L}(\boldsymbol{w}_l)$ since sigmoid function $\sigma(x)$ is bounded in $(0, 1)$. Given that the stochastic gradient noise across parameters in $\boldsymbol{w}_l$ admits to Lévy distribution (Xie et al., 2021), some negative and positive term will balance each other, which makes us wondering whether the regularization term dominates training of threshold. Therefore, we compare the gradient to L_2 penalty by tracing the magnitude ratio of $\nabla_{s_l}\mathcal{L}$ to $\lambda|s_l|$ during training of ResNet-50 on ImageNet, which is shown in Fig. 4.

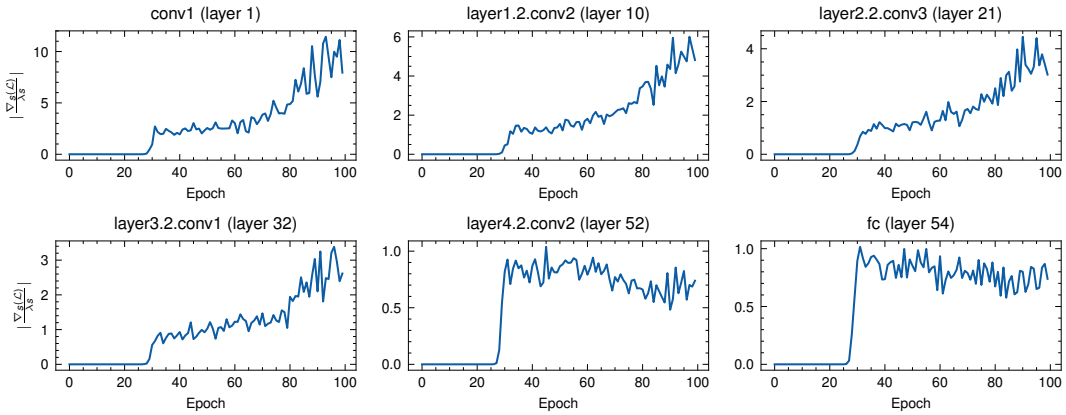

Figure 4: Magnitude ratios of gradient to L_2-regularization $\left|\frac{\nabla_{s_l}\mathcal{L}}{\lambda s_l}\right|$ in different layers when training using STR at 90.23% overall sparsity. Data within an epoch are averaged using geometric mean.

It is evident that L_2 penalty play the leading role in the early stage rather than gradient. We even observe for several final layers, penalty always dominates the training of threshold. The update of

s can thereby be rewritten to $s^{(t+1)} \approx s^{(t)}(1-\eta^{(t)}\lambda)$, where $\eta^{(t)}$ is the learning rate at the t-th iteration. Regardless of gradient, at the beginning of training, STR can be viewed as a special case of threshold scheduler as follows

$$d^{(t)} = \sigma\left(s_{\text{init}} \cdot \prod_{i=0}^{t-1}(1-\eta^{(i)}\lambda)\right), t = 1, 2, \ldots \tag{25}$$

By Theorem 1, the corresponding penalty can be derived by

$$\begin{aligned}
\mu^{(t)} &= \frac{d^{(t+1)} - d^{(t)}}{\eta^{(t)}} \\
&= \frac{\sigma(s^{(t+1)}) - \sigma(s^{(t)})}{\eta^{(t)}} \\
&= \frac{\sigma'(\bar{s})(s^{(t+1)} - s^{(t)})}{\eta^{(t)}} \\
&= -\frac{\sigma'(\bar{s})s^{(t)}\eta^{(t)}\lambda}{\eta^{(t)}} \\
&= -\sigma'(\bar{s})s^{(t)}\lambda
\end{aligned} \tag{26}$$

where $\bar{s}$ lies between $s^{(t+1)}$ and $s^{(t)}$. The existence of $\bar{s}$ is shown by Lagrange's Mean Value Theorem. It is rather difficult to analytically give the explicit expression of $\mu^{(t)}$ since it relies on the behavior of s, which has dynamic decay rate $1-\eta^{(t)}\lambda$.

Loosely speaking, we adopt the approximation $\bar{s} \approx s^{(t)}$. We still cannot show the explicit form of $\mu^{(t)}$, but now we can investigate the trend of penalty by analyzing function $\mu(x) = -\sigma'(x)x$, which is increasing in $(-\infty, \gamma)$. Here $\gamma \approx -1.5434$ is the unique negative root of $e^{-x}(x+1) - x + 1 = 0$. In the early stage, s is a negative number with large magnitude. Based on the above, μ is increasing in the early stage.

C.2 Focus on threshold scheduler instead of final threshold.

Conventional wisdom suggests that when a parameter is set trainable, it will be optimized automatically. This idea should only works for those directly determine the performance, *e.g.*, weights in dense network. For pruning, authors of STDS find the differences in positions of performance versus sparsity curve should be ascribed to the evolving patterns of threshold. To verify this, we replace threshold training mechanism in STR and default scheduler in STDS by several schedulers with same final threshold $D = 10^{-3}$ ($D = 0.5$ for STDS) shown below

- Sine: $d^{(t)} = \frac{1}{2}(1 + \sin(\pi(\frac{t}{T} - \frac{1}{2})))D$
- Linear: $d^{(t)} = \frac{t}{T}D$
- Log2: $d^{(t)} = \log_2(\frac{t}{T} + 1)D$

We conduct the training on ANN ResNet-50 for both methods, the results are shown in Tab. 5. It

Table 5: Comparison of threshold schedulers when applied to ResNet-50 on ImageNet.

Scheduler	**STR** ($D = 10^{-3}$)		**STDS** ($D = 0.5$)	
	Sparsity (%)	Top-1 Acc. (%)	Sparsity (%)	Top-1 Acc. (%)
Sine	94.14	71.51	95.72	72.42
Linear	95.41	69.85	98.46	59.79
Log2	95.95	68.55	98.05	64.74

is obvious that even the final thresholds are set equal, the accuracy and overall sparsity vary with scheduler.

The above results suggest the correct manner to manipulate threshold is pursuing a well-performed scheduler. Even if one studies the learning of threshold, it should be concentrated on optimization of threshold scheduler, which may requires tools in discrete-time optimal control. Due to the complex coupling between constantly changing threshold and final performance, discussion based on discrete-time optimal control is beyond the scope of this work.

D PRUNING EXPERIMENTS ON MOBILENET-V1 USING S-LATS

Besides the ResNet-like structures mentioned in the main text, we also conduct experiments on MobileNet-V1 (Howard et al., 2017) to show the power of our proposed methods S-LATS on the lightweight network. To make a fair comparison, we choose those SOTA studies using the standard training setting, *i.e.*, batch size of 256 and 100 training epochs. They include STR (Kusupati et al., 2020), gradual pruning in WoodFisher (Singh & Alistarh, 2020), and a modern implementation (Gale et al., 2019) of Gradual Magnitude Pruning (GMP) (Zhu & Gupta, 2017). Note that we do not compare S-LATS to OptG, since OptG use $1.8\times$ (180) training epochs, or the comparison would be unfair.

The results are shown in Tab. 6 and Fig. 5. Apparently, our proposed method towers over previous sparsify-during-training work. We elaborate the sparsity budgets and the corresponding final thresholds in Tab. 12 of Appendix G. The hyperparameters for training MobileNet-V1 are stated in Tab. 8 of Appendix F.

Table 6: Performance comparison of MobileNet-V1 on ImageNet using standard training setting (256 batch size, 100 epochs). The results of GMP are gleaned from the manuscripts of STR and OptG.

Method	Top-1 Acc. (%)	Sparsity (%)
Dense	71.95	0
GMP	67.70	74.11
STR	68.35	75.28
STR	66.52	79.07
WoodFisher	70.09	75.28
STDS + S-LATS	**68.44**	**81.84**
GMP	61.80	89.03
STR	64.83	85.80
STR	62.10	89.01
STR	61.51	89.62
WoodFisher	63.87	89.00
STDS + S-LATS	**66.64**	**85.87**
STDS + S-LATS	**65.41**	**88.22**
STDS + S-LATS	**65.13**	**89.08**

E SPARSITY VS FIRING RATE IN SNNS

E.1 AN OVERVIEW OF SNNS

Spiking neural networks (SNNs) are honored as the third generation of neural network models (Maass, 1997), which are derived from the modeling of biological neural networks. SNNs are composed of spiking neurons, which release spikes in binary form, and connections between neurons. The model of spiking neuron is a dynamical system described by one or more ordinary differential equations (ODE) and a firing threshold. The dynamical system is also called "subthreshold dynamics" in the context of computational neuroscience. When the variable representing *membrane potential* exceeds the firing threshold, a spike is generated and passed to all postsynaptic spiking neurons. Today, the most commonly used neuron model is the *Leaky Integrate-and-Fire* (LIF) model.

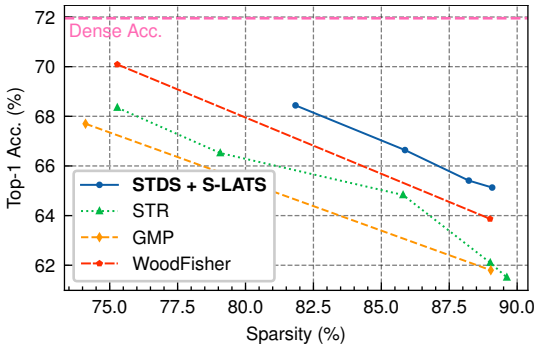

Figure 5: Performance comparison of MobileNet-V1 on ImageNet.

Specifically, LIF has the subthreshold dynamic as follows

$$\tau_m \frac{\mathrm{d}u(t)}{\mathrm{d}t} = -(u(t) - u_{\text{rest}}) + \sum Iw, \tag{27}$$

where $u(t)$ is the membrane potential at time t, u_{rest} is the resting potential, τ_m is the membrane constant, I and w denote input spikes and input weights respectively. The firing behavior of LIF neurons is depicted as a instantaneous jump of membrane potential shown below

$$\lim_{\Delta t \to 0^+} u(t^f + \Delta t) = u_{\text{rest}}, \text{ if } u(t^f) \geq u_{\text{th}}, \tag{28}$$

where u_{th}, t^f are the firing threshold and firing time respectively.

The ODE in Eq.27 can be discretized via Euler method and transformed into RNN-like iterative computing manner as follows

$$\begin{aligned} u[t^-] &= u[t-1] + \frac{1}{\tau_m}\left(-(u[t-1] - u_{\text{rest}}) + \sum\nolimits_i w_i I_i[t]\right), \\ s[t] &= H(u[t^-] - u_{\text{th}}), \\ u[t] &= s[t]u_{\text{rest}} + (1 - s[t])u[t^-]. \end{aligned} \tag{29}$$

where $u[t^-], u[t]$ are the membrane potential before and after firing at timestep t respectively, $H(\cdot)$ is the Heaviside step function modeling jump behavior when a spike is triggered. However, training techniques in RNN, such as backpropagation through time (BPTT) (Werbos, 1990) cannot be directly applied to SNNs for the spiking behavior described by Heaviside is non-differentiable. Thanks to the *surrogate gradient* method proposed in Wu et al. (2018); Neftci et al. (2019), researchers can now incorporate BPTT into training of SNNs by switching to a "differentiable mode" of Heaviside step function when computing gradient. It refers to replacing Heaviside step with a differentiable surrogate function. Surrogate gradient resembles the straight through estimator (Bengio et al., 2013) closely in both computing style and ideology.

E.2 Reducing SNNs cost on neuromorphic hardware

SNNs are considered energy efficient when deployed on a series of dedicated hardware, which is also known as *neuromorphic hardware* or *event-driven hardware*. On these chips, the computation is triggered only when there are incoming spikes and weights are nonzero (Merolla et al., 2014). For this reason, there are three mainstream methods for alleviating the energy cost of a given SNN on neuromorphic chips including 1) unstructured pruning of weights, 2) reducing the number of spikes, and 3) searching for efficient SNN structures. The NAS studies on SNNs (Na et al., 2022; Kim et al., 2022a) are not based on existing SNN structure, so we omit the discussion of NAS methods.

Many recent studies have made ample signs of progresses on pruning and reducing spike counts. They confirm there is weak correlation between the number of spikes and weight sparsity (Deng et al., 2021; Chen et al., 2022; Kim et al., 2022b). We also evaluate the spike counts of pruned SNNs. Compared to the number of spikes, a more frequently used metric is *average firing rate*, which is

obtained by averaging the number of spikes across timesteps and spiking neurons. We collect the average firing rate of each trial during inference using pruned SEW ResNet-18. We further provide a plot of the average firing rate against the sparsity, which is shown in Fig. 6. The weak relationship

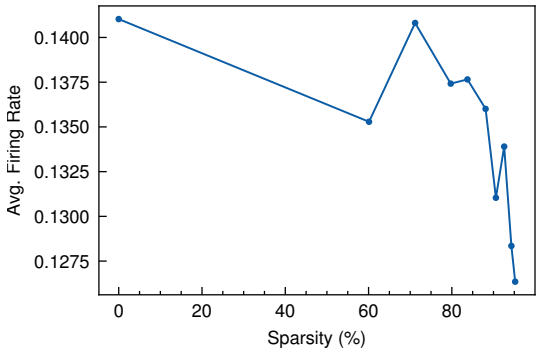

Figure 6: The trend of average firing rate against sparsity.

between the number of spikes and weight sparsity is manifested in the slightly decreased average firing rate. Despite a downward trend in the average firing rate, the relative magnitude of the decline is trifling. It is consistent with previous observations and suggests pruning is an inefficient means of reducing the number of spikes in SNNs.

In conclusion, pruning is an efficient way to induce weight sparsity and lower cost. However, we should not expect the suppression of firing rates as a bonus.

F Training hyperparameters

We make the detailed setting in our experiments clear in Tab. 7, Tab. 8 and Tab. 9.

Table 7: ANN ResNet-50 hyperparameters.

Description	Notation	Value
# Epoch	-	100
Optimizer	-	Momentum SGD ($\text{momentum} = 0.875$)
Overall batch size	-	256 1024
Max. learning rate	$\eta_{\max}$	0.256 0.512
Learning rate scheduler	-	Cosine annealing
Warmup epochs	-	5
Label smoothing	-	0.1
Weight decay	λ	3.05e-5
Prune BN layers?	-	No
Prune first and last layers?	-	Yes

G Sparsity budgets

Table 8: ANN MobileNet-V1 hyperparameters.

Description	Notation	Value
# Epoch	-	100
Optimizer	-	Momentum SGD ($\text{momentum} = 0.875$)
Overall batch size	-	256
Max. learning rate	$\eta_{\max}$	0.256
Learning rate scheduler	-	Cosine annealing
Warmup epochs	-	5
Label smoothing	-	0.1
Weight decay	λ	3.05e-5
Prune BN layers?	-	No
Prune first and last layers?	-	Yes

Table 9: SNN SEW ResNet-18 hyperparameters.

Description	Notation	Value
# Epoch	-	320
Optimizer	-	Momentum SGD ($\text{momentum} = 0.9$)
Overall batch size	-	256
Max. learning rate	$\eta_{\max}$	0.1
Learning rate scheduler	-	Cosine annealing
Weight decay	λ	0
Prune BN layers?	-	No
Prune first and last layers?	-	Yes
Simulation timesteps	-	4
SEW function	-	ADD

Table 10: Sparsity budgets of ResNet-50 using STDS + S-LATS on ImageNet (256 batch size).

Final threshold D	0.1	0.2	0.45	0.7	1.0	1.4	3.5
Layer(s)	Sparsity (%)						
Overall	79.95	89.57	95.12	96.50	97.43	98.01	99.02
conv1	34.18	49.83	64.35	70.89	74.31	78.36	85.58
layer1.0.conv1	43.04	58.79	72.22	76.03	82.25	84.69	89.72
layer1.0.conv2	73.59	82.82	90.82	93.18	94.96	95.97	97.96
layer1.0.conv3	67.26	77.89	88.71	92.95	95.28	95.83	98.12
layer1.0.downsample.0	58.61	73.39	83.91	87.01	90.31	91.83	95.14
layer1.1.conv1	69.42	78.74	87.51	91.72	93.51	94.32	96.99
layer1.1.conv2	73.06	84.09	90.32	94.73	95.62	95.72	98.02
layer1.1.conv3	69.48	79.34	88.12	92.60	94.74	95.12	98.66
layer1.2.conv1	64.66	75.52	87.00	88.64	92.55	94.18	97.97
layer1.2.conv2	63.14	76.05	88.20	90.06	93.02	94.81	98.01
layer1.2.conv3	68.90	80.36	89.11	92.69	94.42	96.64	99.04
layer2.0.conv1	59.82	73.85	83.13	89.19	91.19	93.37	97.02
layer2.0.conv2	72.90	85.04	92.57	94.80	95.74	97.05	98.27
layer2.0.conv3	71.13	82.92	90.37	93.15	94.38	96.03	98.00
layer2.0.downsample.0	81.42	90.05	94.78	96.28	97.40	97.88	99.03
layer2.1.conv1	83.44	90.47	95.96	96.73	97.95	98.25	99.21
layer2.1.conv2	82.60	90.41	96.22	96.90	98.31	98.68	99.46
layer2.1.conv3	73.53	83.15	91.63	93.80	96.08	96.79	99.04
layer2.2.conv1	74.48	85.75	92.37	94.26	95.85	96.09	98.46
layer2.2.conv2	75.42	87.51	93.18	94.89	96.32	96.25	98.36
layer2.2.conv3	70.68	82.69	90.35	93.93	95.32	96.33	98.39
layer2.3.conv1	71.05	83.26	90.17	92.75	94.34	95.50	97.94
layer2.3.conv2	74.50	83.83	91.28	94.27	94.89	97.02	98.25
layer2.3.conv3	72.81	85.69	92.36	94.37	95.33	96.74	98.73
layer3.0.conv1	61.26	74.70	84.81	88.01	90.43	91.82	95.56
layer3.0.conv2	82.41	91.42	96.15	97.27	97.94	98.33	99.04
layer3.0.conv3	71.21	82.18	89.38	92.04	93.73	94.97	97.50
layer3.0.downsample.0	86.21	93.20	96.96	97.98	98.53	98.83	99.44
layer3.1.conv1	86.76	92.91	96.48	97.80	98.37	98.67	99.41
layer3.1.conv2	86.67	93.68	97.09	98.09	98.52	98.92	99.51
layer3.1.conv3	75.18	86.50	93.44	95.48	96.75	97.53	98.98
layer3.2.conv1	83.83	90.82	96.06	96.80	97.58	98.36	99.32
layer3.2.conv2	84.24	92.07	96.17	97.26	97.93	98.62	99.42
layer3.2.conv3	75.66	85.49	93.13	94.99	96.24	97.61	99.14
layer3.3.conv1	80.26	90.40	94.41	95.87	97.20	98.17	99.03
layer3.3.conv2	83.78	91.46	96.14	97.70	98.34	98.75	99.49
layer3.3.conv3	76.82	87.45	93.32	96.24	97.10	98.08	99.26
layer3.4.conv1	77.62	87.58	93.41	96.03	96.44	97.39	98.84
layer3.4.conv2	83.13	91.87	96.17	97.48	98.14	98.61	99.53
layer3.4.conv3	76.09	87.14	93.79	95.94	97.00	97.93	99.23
layer3.5.conv1	73.48	85.83	92.45	94.68	95.73	96.69	98.88
layer3.5.conv2	81.39	90.78	95.85	97.39	98.09	98.60	99.53
layer3.5.conv3	73.98	85.35	92.53	95.16	95.98	96.99	99.14
layer4.0.conv1	64.48	77.49	87.00	90.33	92.09	93.65	96.25
layer4.0.conv2	84.02	94.14	98.07	98.45	98.82	99.05	99.38
layer4.0.conv3	72.53	83.17	90.39	93.09	94.63	95.87	97.78
layer4.0.downsample.0	85.36	92.49	96.39	97.42	97.92	98.33	99.13
layer4.1.conv1	81.34	90.10	95.55	97.17	98.00	98.60	99.35
layer4.1.conv2	83.00	91.75	96.68	97.85	98.52	98.95	99.55
layer4.1.conv3	75.37	86.50	93.42	95.43	96.63	97.40	98.59
layer4.2.conv1	74.54	86.22	93.83	96.01	97.23	97.82	98.93
layer4.2.conv2	79.66	90.89	97.38	98.07	98.56	98.77	99.29
layer4.2.conv3	66.98	79.13	88.74	91.60	93.82	95.04	97.79
fc	86.44	94.47	97.87	98.65	99.04	99.25	99.54

Table 11: Sparsity budgets of ResNet-50 using STDS + S-LATS on ImageNet (1024 batch size).

Final threshold D	0.1	0.23	0.475	0.73	1.13	1.5	3.5
Layer(s)	Sparsity (%)						
Overall	79.00	90.15	95.01	96.53	97.54	98.00	98.93
conv1	40.67	55.02	66.07	70.97	76.87	77.51	84.56
layer1.0.conv1	52.32	65.16	74.71	79.30	85.82	83.20	91.04
layer1.0.conv2	75.36	85.46	91.46	93.60	95.41	95.79	98.06
layer1.0.conv3	70.77	80.81	88.84	92.82	94.79	95.66	97.92
layer1.0.downsample.0	64.12	74.69	83.73	87.38	90.48	92.49	94.98
layer1.1.conv1	74.98	82.61	87.48	91.63	93.60	95.78	97.49
layer1.1.conv2	78.53	86.02	90.46	93.74	95.43	96.98	98.84
layer1.1.conv3	77.44	81.32	88.08	93.14	94.73	96.28	98.66
layer1.2.conv1	70.75	77.59	86.50	90.91	93.51	94.43	97.99
layer1.2.conv2	70.35	78.69	86.69	91.75	93.40	95.38	97.97
layer1.2.conv3	77.30	82.51	89.57	92.70	93.87	96.25	98.46
layer2.0.conv1	69.29	77.38	85.20	88.83	92.42	94.80	97.29
layer2.0.conv2	76.86	87.06	93.28	94.81	96.71	96.96	98.50
layer2.0.conv3	75.99	84.57	90.73	92.50	94.87	95.76	97.77
layer2.0.downsample.0	84.75	91.27	95.10	96.67	97.49	97.89	98.93
layer2.1.conv1	89.09	91.85	95.55	97.00	98.01	98.23	99.44
layer2.1.conv2	87.41	92.13	95.99	97.35	98.26	98.67	99.65
layer2.1.conv3	81.25	86.02	92.51	94.20	96.12	97.09	99.16
layer2.2.conv1	77.10	86.77	91.45	93.55	96.12	96.55	98.79
layer2.2.conv2	79.02	88.76	92.77	94.61	96.40	97.14	99.17
layer2.2.conv3	71.18	83.43	90.05	92.80	95.80	95.97	98.82
layer2.3.conv1	71.29	84.35	90.34	93.72	95.54	95.78	97.87
layer2.3.conv2	73.69	85.49	91.28	94.32	96.04	96.57	97.97
layer2.3.conv3	73.48	86.39	92.20	94.78	95.97	96.83	98.25
layer3.0.conv1	62.93	77.81	85.41	88.80	91.63	92.85	96.26
layer3.0.conv2	82.70	92.62	96.15	97.45	98.01	98.43	99.07
layer3.0.conv3	72.49	84.57	89.68	92.98	94.79	95.56	97.68
layer3.0.downsample.0	87.88	94.74	97.35	98.35	98.76	99.06	99.44
layer3.1.conv1	86.70	93.65	96.63	97.72	98.49	98.69	99.39
layer3.1.conv2	85.98	93.99	97.06	98.13	98.87	98.90	99.48
layer3.1.conv3	74.52	87.49	93.02	95.16	97.08	97.38	98.85
layer3.2.conv1	84.36	93.20	95.61	97.09	98.08	98.13	99.16
layer3.2.conv2	83.49	93.01	96.15	97.35	98.18	98.21	99.20
layer3.2.conv3	75.10	87.87	92.72	95.06	96.77	96.89	98.71
layer3.3.conv1	80.11	90.31	94.34	96.18	97.36	97.63	99.07
layer3.3.conv2	82.74	92.45	96.15	97.39	98.00	98.54	99.37
layer3.3.conv3	75.19	87.38	93.14	95.83	96.50	97.57	98.93
layer3.4.conv1	76.54	88.09	93.25	95.68	96.46	97.12	98.51
layer3.4.conv2	82.23	92.02	96.05	97.20	97.90	98.59	99.24
layer3.4.conv3	75.51	87.73	92.59	95.53	96.72	97.46	98.64
layer3.5.conv1	73.52	86.23	91.75	94.56	95.75	96.62	98.50
layer3.5.conv2	80.96	91.09	95.45	97.03	97.88	98.39	99.30
layer3.5.conv3	73.55	85.38	91.40	94.61	95.74	96.61	98.47
layer4.0.conv1	65.17	79.76	87.24	90.32	92.49	93.47	96.29
layer4.0.conv2	84.07	94.58	98.29	98.68	98.97	99.09	99.41
layer4.0.conv3	72.08	85.07	90.95	93.23	95.02	95.93	97.81
layer4.0.downsample.0	85.95	93.61	96.57	97.64	98.22	98.54	99.23
layer4.1.conv1	77.97	89.96	94.94	96.52	97.68	98.12	99.01
layer4.1.conv2	80.73	91.40	96.24	97.62	98.47	98.83	99.38
layer4.1.conv3	71.47	86.25	92.96	95.09	96.71	97.38	98.23
layer4.2.conv1	69.25	86.10	93.58	95.19	96.90	97.62	98.48
layer4.2.conv2	75.76	89.41	96.47	97.70	98.47	98.83	99.29
layer4.2.conv3	62.85	79.58	88.80	91.41	93.97	95.32	97.61
fc	88.02	96.39	98.54	99.16	99.41	99.52	99.69

Table 12: Sparsity budgets of MobileNet-V1 using STDS + S-LATS on ImageNet.

Final threshold D	0.4	0.6	0.8	0.9
Layer(s)	Sparsity (%)			
Overall	81.84	85.87	88.22	89.08
model.0.0	51.39	62.27	57.06	55.90
model.1.0	46.53	62.85	54.86	56.94
model.1.3	60.45	76.76	67.63	68.51
model.2.0	11.28	21.70	18.75	18.23
model.2.3	52.82	61.65	61.56	68.92
model.3.0	23.96	25.43	26.04	31.86
model.3.3	53.38	59.64	64.90	67.77
model.4.0	2.69	5.64	8.94	5.38
model.4.3	62.50	68.58	72.97	75.10
model.5.0	20.70	26.26	29.34	31.38
model.5.3	68.64	74.57	78.17	79.69
model.6.0	15.32	20.49	20.40	25.00
model.6.3	79.08	84.26	87.14	87.85
model.7.0	24.80	29.28	34.44	37.83
model.7.3	81.65	86.53	89.26	90.23
model.8.0	35.72	40.89	48.55	48.59
model.8.3	81.10	85.35	88.22	89.20
model.9.0	40.06	42.73	46.59	50.85
model.9.3	79.38	84.02	87.30	87.97
model.10.0	33.88	40.41	42.99	45.33
model.10.3	75.01	80.66	83.84	85.23
model.11.0	26.52	28.56	31.77	31.42
model.11.3	71.13	77.06	80.75	82.36
model.12.0	10.44	13.24	14.67	16.06
model.12.3	80.64	84.83	87.28	88.15
model.13.0	43.65	46.79	48.35	49.09
model.13.3	82.10	85.47	87.57	88.36
fc	92.40	95.25	96.54	96.93

Table 13: Sparsity budgets of ResNet-50 using PGH scheduler in pruning at initialization setting on ImageNet.

Final threshold D	0.1	0.11	0.13	0.15
Layer(s)	Sparsity (%)			
Overall	87.16	90.00	93.11	95.64
conv1	32.05	33.48	34.66	34.75
layer1.0.conv1	38.92	38.70	37.48	33.89
layer1.0.conv2	67.15	67.55	66.07	59.91
layer1.0.conv3	59.89	60.16	57.60	49.34
layer1.0.downsample.0	58.89	61.24	64.05	60.72
layer1.1.conv1	60.78	63.04	61.00	53.19
layer1.1.conv2	65.13	64.05	60.48	54.98
layer1.1.conv3	57.58	55.88	49.53	42.51
layer1.2.conv1	53.80	55.80	51.01	43.32
layer1.2.conv2	58.37	59.34	55.33	51.22
layer1.2.conv3	61.24	62.34	60.16	48.72
layer2.0.conv1	49.99	49.88	45.61	42.38
layer2.0.conv2	70.38	69.61	72.09	86.96
layer2.0.conv3	63.37	61.13	59.27	82.79
layer2.0.downsample.0	75.97	74.72	70.28	59.11
layer2.1.conv1	77.21	75.13	66.76	59.39
layer2.1.conv2	76.35	74.61	73.73	86.70
layer2.1.conv3	63.60	60.96	58.64	81.81
layer2.2.conv1	68.74	66.52	56.07	49.60
layer2.2.conv2	70.42	69.81	69.10	87.02
layer2.2.conv3	63.30	60.06	54.06	83.67
layer2.3.conv1	63.87	59.87	44.49	43.37
layer2.3.conv2	70.13	68.83	67.57	86.60
layer2.3.conv3	66.09	62.03	49.41	82.74
layer3.0.conv1	54.05	54.86	76.17	100.00
layer3.0.conv2	82.47	89.52	99.25	100.00
layer3.0.conv3	65.61	82.20	98.40	100.00
layer3.0.downsample.0	78.21	71.87	54.37	53.19
layer3.1.conv1	78.03	78.40	79.45	100.00
layer3.1.conv2	82.93	92.45	98.20	100.00
layer3.1.conv3	69.54	87.02	96.55	100.00
layer3.2.conv1	74.19	72.72	89.19	100.00
layer3.2.conv2	81.22	89.29	99.56	100.00
layer3.2.conv3	68.24	82.87	99.03	100.00
layer3.3.conv1	66.58	63.63	88.10	100.00
layer3.3.conv2	78.70	88.28	99.54	100.00
layer3.3.conv3	65.76	80.80	99.00	100.00
layer3.4.conv1	57.57	59.99	75.83	100.00
layer3.4.conv2	79.10	88.66	98.66	100.00
layer3.4.conv3	62.33	80.67	97.62	100.00
layer3.5.conv1	53.63	58.43	78.13	100.00
layer3.5.conv2	79.08	88.88	99.04	100.00
layer3.5.conv3	58.30	80.30	98.32	100.00
layer4.0.conv1	90.52	100.00	100.00	100.00
layer4.0.conv2	99.80	100.00	100.00	100.00
layer4.0.conv3	99.37	100.00	100.00	100.00
layer4.0.downsample.0	75.46	79.81	86.40	91.34
layer4.1.conv1	97.11	100.00	100.00	100.00
layer4.1.conv2	99.90	100.00	100.00	100.00
layer4.1.conv3	99.56	100.00	100.00	100.00
layer4.2.conv1	98.61	100.00	100.00	100.00
layer4.2.conv2	99.96	100.00	100.00	100.00
layer4.2.conv3	99.74	100.00	100.00	100.00
fc	83.95	82.61	81.69	86.05

Table 14: Sparsity budgets of SEW ResNet-18 using STDS + S-LATS on ImageNet.

Final threshold D	0.5	1.0	2.0	3.0	5.0	7.0	10	15	20
Layer(s)	Sparsity (%)								
Overall	60.11	71.18	79.74	83.75	88.11	89.96	92.57	94.30	95.21
conv1	38.70	49.83	62.40	66.90	73.93	79.44	80.71	85.43	86.90
layer1.0.conv1.0	48.98	60.42	73.58	76.98	82.13	86.19	87.39	90.02	91.90
layer1.0.conv2.0	37.88	50.73	61.17	66.69	71.79	76.28	79.39	82.53	84.40
layer1.1.conv1.0	40.37	54.44	66.75	72.12	77.59	80.65	84.02	86.53	87.84
layer1.1.conv2.0	39.94	51.82	62.50	67.47	73.18	77.47	80.40	84.16	85.53
layer2.0.conv1.0	39.78	52.67	64.46	68.41	74.30	77.91	81.07	83.65	85.84
layer2.0.conv2.0	46.68	59.70	70.55	74.26	79.64	83.32	85.99	88.92	90.62
layer2.0.downsample.0.0	16.81	25.96	36.11	41.39	47.29	54.48	59.00	63.48	67.22
layer2.1.conv1.0	48.87	62.95	73.58	77.19	81.24	84.64	87.51	89.59	91.89
layer2.1.conv2.0	49.85	63.01	71.87	77.04	81.44	84.69	87.65	90.72	91.80
layer3.0.conv1.0	48.98	60.98	71.23	75.30	80.16	83.48	85.91	88.49	90.02
layer3.0.conv2.0	56.54	68.36	77.14	81.23	86.21	88.38	91.20	93.05	94.38
layer3.0.downsample.0.0	27.11	37.56	48.76	54.21	61.76	66.29	70.50	75.62	79.10
layer3.1.conv1.0	61.25	72.95	80.88	83.91	87.98	89.96	91.59	93.66	94.48
layer3.1.conv2.0	60.16	71.00	79.03	82.63	86.38	88.59	90.49	92.50	93.61
layer4.0.conv1.0	58.92	70.02	78.17	82.08	86.23	88.38	90.44	92.65	93.86
layer4.0.conv2.0	64.63	74.90	82.40	85.93	89.74	91.58	93.27	95.14	95.96
layer4.0.downsample.0.0	31.91	42.72	53.25	58.97	65.57	69.29	72.34	75.55	77.45
layer4.1.conv1.0	67.33	77.95	85.81	89.57	93.42	95.61	96.99	97.89	98.38
layer4.1.conv2.0	63.04	72.57	80.14	84.08	88.71	91.79	94.04	95.35	96.01
fc	33.00	52.40	71.29	79.66	86.71	89.96	92.52	94.95	96.27

Table 15: Sparsity budgets of SEW ResNet-18 using our implementation of original STDS (STDS + Sine scheduler) on ImageNet.

Final threshold D	0.6	0.8	1.5	3.0	5.0
Layer(s)	Sparsity (%)				
Overall	76.06	79.91	85.91	90.62	93.19
conv1	48.07	53.99	68.15	79.49	85.71
layer1.0.conv1.0	61.37	69.80	81.65	88.07	91.37
layer1.0.conv2.0	56.20	62.73	73.62	79.45	83.24
layer1.1.conv1.0	56.73	64.55	79.27	84.19	87.55
layer1.1.conv2.0	58.62	65.86	73.35	79.39	84.28
layer2.0.conv1.0	58.34	66.05	75.50	80.44	85.06
layer2.0.conv2.0	66.05	72.17	80.13	85.38	88.55
layer2.0.downsample.0.0	28.70	36.28	49.60	58.87	64.78
layer2.1.conv1.0	68.32	74.54	82.47	87.28	89.96
layer2.1.conv2.0	68.65	73.85	80.92	86.52	89.55
layer3.0.conv1.0	68.04	73.05	80.11	86.17	88.89
layer3.0.conv2.0	74.94	79.00	85.20	89.69	92.33
layer3.0.downsample.0.0	44.28	50.82	62.14	70.68	76.31
layer3.1.conv1.0	78.62	82.50	87.89	91.36	93.57
layer3.1.conv2.0	76.93	80.73	86.32	90.13	92.28
layer4.0.conv1.0	76.41	80.41	86.30	90.59	92.95
layer4.0.conv2.0	80.60	83.84	88.43	92.07	93.92
layer4.0.downsample.0.0	49.57	55.31	63.96	71.63	75.94
layer4.1.conv1.0	82.60	85.81	90.74	94.64	96.62
layer4.1.conv2.0	77.17	79.68	84.09	88.96	92.32
fc	46.12	56.43	77.07	90.62	95.18