Soft Threshold Weight Reparameterization for Learnable Sparsity

Aditya Kusupati [1]
Vivek Ramanujan [*2] **Raghav Somani** [*1] **Mitchell Wortsman** [*1]
Prateek Jain [3] **Sham Kakade** [1] **Ali Farhadi** [1]

Abstract

Sparsity in Deep Neural Networks (DNNs) is studied extensively with the focus of maximizing prediction accuracy given an overall parameter budget. Existing methods rely on uniform or heuristic non-uniform sparsity budgets which have sub-optimal layer-wise parameter allocation resulting in a) lower prediction accuracy or b) higher inference cost (FLOPs). This work proposes Soft Threshold Reparameterization (STR), a novel use of the soft-threshold operator on DNN weights. STR smoothly induces sparsity while *learning* pruning thresholds thereby obtaining a non-uniform sparsity budget. Our method achieves state-of-the-art accuracy for unstructured sparsity in CNNs (ResNet50 and MobileNetV1 on ImageNet-1K), and, additionally, learns non-uniform budgets that empirically reduce the FLOPs by up to 50%. Notably, STR boosts the accuracy over existing results by up to 10% in the ultra sparse (99%) regime and can also be used to induce low-rank (structured sparsity) in RNNs. In short, STR is a simple mechanism which learns effective sparsity budgets that contrast with popular heuristics. Code, pretrained models and sparsity budgets are at https://github.com/RAIVNLab/STR.

1. Introduction

Deep Neural Networks (DNNs) are the state-of-the-art models for many important tasks in the domains of Computer Vision, Natural Language Processing, etc. To enable highly accurate solutions, DNNs require large model sizes resulting in huge inference costs, which many times become the main bottleneck in the real-world deployment of the solutions. During inference, a typical DNN model stresses the following aspects of the compute environment: 1) RAM - working memory, 2) Processor compute - Floating Point Operations (FLOPs[1]), and 3) Flash - model size. Various techniques are proposed to make DNNs efficient including model pruning (sparsity) (Han et al., 2015), knowledge distillation (Buciluǎ et al., 2006), model architectures (Howard et al., 2017) and quantization (Rastegari et al., 2016).

Sparsity of the model, in particular, has potential for impact across a variety of inference settings as it reduces the model size and inference cost (FLOPs) without significant change in training pipelines. Naturally, several interesting projects address inference speed-ups via sparsity on existing frameworks (Liu et al., 2015; Elsen et al., 2019) and commodity hardware (Ashby et al.). On-premise or Edge computing is another domain where sparse DNNs have potential for deep impact as it is governed by billions of battery limited devices with single-core CPUs. These devices, including mobile phones (Anguita et al., 2012) and IoT sensors (Patil et al., 2019; Roy et al., 2019), can benefit significantly from sparsity as it can enable real-time on-device solutions.

Sparsity in DNNs, surveyed extensively in Section 2, has been the subject of several papers where new algorithms are designed to obtain models with a given parameter budget. But state-of-the-art DNN models tend to have a large number of layers with highly non-uniform distribution both in terms of the number of parameters as well as FLOPs required per layer. Most existing methods rely either on uniform sparsity across all parameter tensors (layers) or on heuristic non-uniform sparsity budgets leading to a sub-optimal weight allocation across layers and can lead to a significant loss in accuracy. Furthermore, if the budget is set at a global level, some of the layers with a small number of parameters would be fully dense as their contribution to the budget is insignificant. However, those layers can have significant FLOPs, e.g., in an initial convolution layer, a simple tiny 3×3 kernel would be applied to the entire image. Hence, while such models might decrease the number of non-zeroes significantly, their FLOPs could still be large.

[*]Equal contribution [1]University of Washington, USA [2]Allen Institute for Artificial Intelligence, USA [3]Microsoft Research, India. Correspondence to: Aditya Kusupati <kusupati@cs.washington.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

[1]One Multiply-Add is counted as one FLOP

Motivated by the above-mentioned challenges, this works addresses the following question: "*Can we design a method to learn non-uniform sparsity budget across layers that is optimized per-layer, is stable, and is accurate?*".

Most existing methods for learning sparse DNNs have their roots in the long celebrated literature of high-dimension statistics and, in particular, sparse regression. These methods are mostly based on well-known Hard and Soft Thresholding techniques, which are essentially projected gradient methods with explicit projection onto the set of sparse parameters. However, these methods require a priori knowledge of sparsity, and as mentioned above, mostly heuristic methods are used to set the sparsity levels per layer.

We propose Soft Threshold Reparameterization (STR) to address the aforementioned issues. We use the fact that the projection onto the sparse sets is available in closed form and propose a novel reparameterization of the problem. That is, for forward pass of DNN, we use soft-thresholded version (Donoho, 1995) of a weight tensor $\mathbf{W}_l$ of the l-th layer in the DNN: $\mathcal{S}(\mathbf{W}_l, \alpha_l) := \text{sign}\,(\mathbf{W}_l) \cdot \text{ReLU}(|\mathbf{W}_l| - \alpha_l)$ where α_l is the pruning threshold for the l-th layer. As the DNN loss can be written as a continuous function of α_l's, we can use backpropagation to learn layer-specific α_l to smoothly induce sparsity. Typically, each layer in a neural network is distinct unlike the interchangeable weights and neurons making it interesting to learn layer-wise sparsity.

Due to layer-specific thresholds and sparsity, STR is able to achieve state-of-the-art accuracy for unstructured sparsity in CNNs across various sparsity regimes. STR makes even small-parameter layers sparse resulting in models with significantly lower inference FLOPs than the baselines. For example, STR for 90% sparse MobileNetV1 on ImageNet-1K results in a 0.3% boost in accuracy with 50% fewer FLOPs. Empirically, STR's learnt non-uniform budget makes it a very effective choice for ultra (99%) sparse ResNet50 as well where it is $\sim$10% more accurate than baselines on ImageNet-1K. STR can also be trivially modified to induce structured sparsity, demonstrating its generalizability to a variety of DNN architectures across domains. Finally, STR's learnt non-uniform sparsity budget transfers across tasks thus discovering an efficient sparse backbone of the model.

The 3 major contributions of this paper are:

- Soft Threshold Reparameterization (STR), for the weights in DNNs, to induce sparsity via learning the per-layer pruning thresholds thereby obtaining a better non-uniform sparsity budget across layers.
- Extensive experimentation showing that STR achieves the state-of-the-art accuracy for sparse CNNs (ResNet50 and MobileNetV1 on ImageNet-1K) along with a significant reduction in inference FLOPs.
- Extension of STR to structured sparsity, that is useful for the direct implementation of fast inference in practice.

2. Related Work

This section covers the spectrum of work on sparsity in DNNs. The sparsity in the discussion can be characterized as (a) unstructured and (b) structured while sparsification techniques can be (i) dense-to-sparse, and (ii) sparse-to-sparse. Finally, the sparsity budget in DNNs can either be (a) uniform, or (b) non-uniform across layers. This will be a key focus of this paper, as different budgets result in different inference compute costs as measured by FLOPs. This section also discusses the recent work on learnable sparsity.

2.1. Unstructured and Structured Sparsity

Unstructured sparsity does not take the structure of the model (e.g. channels, rank, etc.,) into account. Typically, unstructured sparsity is induced in DNNs by making the parameter tensors sparse directly based on heuristics (e.g. weight magnitude) thereby creating sparse tensors that might not be capable of leveraging the speed-ups provided by commodity hardware during training and inference. Unstructured sparsity has been extensively studied and includes methods which use gradient, momentum, and Hessian based heuristics (Evci et al., 2020; Lee et al., 2019; LeCun et al., 1990; Hassibi & Stork, 1993; Dettmers & Zettlemoyer, 2019), and magnitude-based pruning (Han et al., 2015; Guo et al., 2016; Zhu & Gupta, 2017; Frankle & Carbin, 2019; Gale et al., 2019; Mostafa & Wang, 2019; Bellec et al., 2018; Mocanu et al., 2018; Narang et al., 2019; Kusupati et al., 2018; Wortsman et al., 2019). Unstructured sparsity can also be induced by L_0, L_1 regularization (Louizos et al., 2018), and Variational Dropout (VD) (Molchanov et al., 2017).

Gradual Magnitude Pruning (GMP), proposed in (Zhu & Gupta, 2017), and studied further in (Gale et al., 2019), is a simple magnitude-based weight pruning applied gradually over the course of the training. Discovering Neural Wirings (DNW) (Wortsman et al., 2019) also relies on magnitude-based pruning while utilizing a straight-through estimator for the backward pass. GMP and DNW are the state-of-the-art for unstructured pruning in DNNs (especially in CNNs) demonstrating the effectiveness of magnitude pruning. VD gets accuracy comparable to GMP (Gale et al., 2019) for CNNs but at a cost of $2\times$ memory and $4\times$ compute during training making it hard to be used ubiquitously.

Structured sparsity takes structure into account making the models scalable on commodity hardware with the standard computation techniques/architectures. Structured sparsity includes methods which make parameter tensors low-rank (Jaderberg et al., 2014; Alizadeh et al., 2020; Lu et al., 2016), prune out channels, filters and induce block/group sparsity (Liu et al., 2019; Wen et al., 2016; Li et al., 2017; Luo et al., 2017; Gordon et al., 2018; Yu & Huang, 2019). Even though structured sparsity can leverage speed-ups provided by parallelization, the highest levels of model pruning

are only possible with unstructured sparsity techniques.

2.2. Dense-to-sparse and Sparse-to-sparse Training

Until recently, most sparsification methods were dense-to-sparse i.e., the DNN starts fully dense and is made sparse by the end of the training. Dense-to-sparse training in DNNs encompasses the techniques presented in (Han et al., 2015; Zhu & Gupta, 2017; Molchanov et al., 2017; Frankle & Carbin, 2019; Renda et al., 2020).

The lottery ticket hypothesis (Frankle & Carbin, 2019) sparked an interest in training sparse neural networks end-to-end. This is referred to as sparse-to-sparse training and a lot of recent work (Mostafa & Wang, 2019; Bellec et al., 2018; Evci et al., 2020; Lee et al., 2019; Dettmers & Zettlemoyer, 2019) aims to do sparse-to-sparse training using techniques which include re-allocation of weights to improve accuracy.

Dynamic Sparse Reparameterization (DSR) (Mostafa & Wang, 2019) heuristically obtains a global magnitude threshold along with the re-allocation of the weights based on the non-zero weights present at every step. Sparse Networks From Scratch (SNFS) (Dettmers & Zettlemoyer, 2019) utilizes momentum of the weights to re-allocate weights across layers and the Rigged Lottery (RigL) (Evci et al., 2020) uses the magnitude to drop and the periodic dense gradients to regrow weights. SNFS and RigL are state-of-the-art in sparse-to-sparse training but fall short of GMP for the same experimental settings. It should be noted that, even though sparse-to-sparse can reduce the training cost, the existing frameworks (Paszke et al., 2019; Abadi et al., 2016) consider the models as dense resulting in minimal gains.

DNW (Wortsman et al., 2019) and Dynamic Pruning with Feedback (DPF) (Lin et al., 2020) fall between both as DNW uses a fully dense gradient in the backward pass and DPF maintains a copy of the dense model in parallel to optimize the sparse model through feedback. Note that DPF is complementary to most of the techniques discussed here.

2.3. Uniform and Non-uniform Sparsity

Uniform sparsity implies that all the layers in the DNN have the same amount of sparsity in proportion. Quite a few works have used uniform sparsity (Gale et al., 2019), given its ease and lack of hyperparameters. However, some works keep parts of the model dense, including the first or the last layers (Lin et al., 2020; Mostafa & Wang, 2019; Zhu & Gupta, 2017). In general, making the first or the last layers dense benefits all the methods. GMP typically uses uniform sparsity and achieves state-of-the-art results.

Non-uniform sparsity permits different layers to have different sparsity budgets. Weight re-allocation heuristics have been used for non-uniform sparsity in DSR and SNFS. It can be a fixed budget like the ERK (Erdos-Renyi-Kernel) heuristic described in RigL (Evci et al., 2020). A global pruning threshold (Han et al., 2015) can also induce non-uniform sparsity and has been leveraged in Iterative Magnitude Pruning (IMP) (Frankle & Carbin, 2019; Renda et al., 2020). A good non-uniform sparsity budget can help in maintaining accuracy while also reducing the FLOPs due to a better parameter distribution. The aforementioned methods with non-uniform sparsity do not reduce the FLOPs compared to uniform sparsity in practice. Very few techniques like AMC (He et al., 2018), using expensive reinforcement learning, minimize FLOPs with non-uniform sparsity.

Most of the discussed techniques rely on intelligent heuristics to obtain non-uniform sparsity. Learning the pruning thresholds and in-turn learning the non-uniform sparsity budget is the main contribution of this paper.

2.4. Learnable Sparsity

Concurrent to our work, (Savarese et al., 2019; Liu 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. The reader is urged to review these works for a more complete picture of the field. Note that, while STR is proposed to induce layer-wise unstructured sparsity, it can be easily adapted for global, filter-wise, or per-weight sparsity as discussed in Appendix A.5.

3. Method - STR

Optimization under sparsity constraint on the parameter set is a well studied area spanning more than three decades (Donoho, 1995; Candes et al., 2007; Jain et al., 2014), and is modeled as:

$$\min_{\mathcal{W}} \mathcal{L}(\mathcal{W}; \mathcal{D}), \text{ s.t. } \|\mathcal{W}\|_0 \leq k,$$

where $\mathcal{D} \coloneqq \left\{\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}\right\}_{i \in [n]}$ is the observed data, $\mathcal{L}$ is the loss function, $\mathcal{W}$ are the parameters to be learned and $\|\cdot\|_0$ denotes the L_0-norm or the number of non-zeros, and k is the parameter budget. Due to non-convexity and combinatorial structure of the L_0 norm constraint, it's convex relaxation L_1 norm has been studied for long time and has been at the center of a large literature on high-dimensional learning. In particular, several methods have been proposed to solve the two problems including projected gradient descent, forward/backward pruning etc.

Projected Gradient Descent (PGD) in particular has been popular for both the problems as the projection onto both L_0 as well as the L_1 ball is computable in almost closed form (Beck & Teboulle, 2009; Jain et al., 2014); L_0 ball projection is called Hard Thresholding while L_1 ball projection is known as Soft Thresholding. Further, these methods have been the guiding principle for many modern DNN

model pruning (sparsity) techniques (Han et al., 2015; Zhu & Gupta, 2017; Narang et al., 2019).

However, projection-based methods suffer from the problem of dense gradient and intermediate parameter structure, as the gradient descent iterate can be arbitrarily out of the set and is then projected back onto L_0 or L_1 ball. At a scale of billions of parameters, computing such dense gradients and updates can be daunting. More critically, the budget parameter k is set at the global level, so it is not clear how to partition the budget for each layer, as the importance of each layer can be significantly different.

In this work, we propose a reparameterization, Soft Threshold Reparameterization (STR) based on the soft threshold operator (Donoho, 1995), to alleviate both the above mentioned concerns. That is, instead of first updating $\mathcal{W}$ via gradient descent and then computing its projection, we directly optimize over projected $\mathcal{W}$. Let $\mathcal{S}_g(\mathcal{W}; s)$ be the projection of $\mathcal{W}$ parameterized by s and function g. $\mathcal{S}$ is applied to each element of $\mathcal{W}$ and is defined as:

$$\mathcal{S}_g(w, s) := \text{sign}\,(w) \cdot \text{ReLU}(|w| - g(s)), \tag{1}$$

where s is a learnable parameter, $g : \mathbb{R} \to \mathbb{R}$, and $\alpha = g(s)$ is the pruning threshold. $\text{ReLU}(a) = \max(a, 0)$. That is, if $|w| \leq g(s)$, then $\mathcal{S}_g(w, s)$ sets it to 0.

Reparameterizing the optimization problem with $\mathcal{S}$ *modifies* (note that it is not equivalent) it to:

$$\min_{\mathcal{W}} \mathcal{L}(\mathcal{S}_g(\mathcal{W}, \boldsymbol{s}), \mathcal{D}). \tag{2}$$

For L-layer DNN architectures, we divide $\mathcal{W}$ into: $\mathcal{W} = [\mathbf{W}_l]_{l=1}^{L}$ where $\mathbf{W}_l$ is the parameter tensor for the l-th layer. As mentioned earlier, different layers of DNNs are unique can have significantly different number of parameters. Similarly, different layers might need different sparsity budget for the best accuracy. So, we set the trainable pruning parameter for each layer as s_l. That is, $\boldsymbol{s} = [s_1, \ldots, s_L]$.

Now, using the above mentioned reparameterization for each $\mathbf{W}_l$ and adding a standard L_2 regularization per layer, we get the following Gradient Descent (GD) update equation at the t-th step for $\mathbf{W}_l$, $\forall\, l \in [L]$:

$$\begin{aligned}&\mathbf{W}_l^{(t+1)} \leftarrow (1 - \eta_t \cdot \lambda)\mathbf{W}_l^{(t)} \\ &- \eta_t \nabla_{\mathcal{S}_g(\mathbf{W}_l, s_l)} \mathcal{L}(\mathcal{S}_g(\mathcal{W}^{(t)}, \boldsymbol{s}), \mathcal{D}) \odot \nabla_{\mathbf{W}_l} \mathcal{S}_g(\mathbf{W}_l, s_l),\end{aligned} \tag{3}$$

where η_t is the learning rate at the t-th step, and λ is the L_2 regularization (weight-decay) hyper-parameter. $\nabla_{\mathbf{W}_l} \mathcal{S}_g(\mathbf{W}_l, s_l)$ is the gradient of $\mathcal{S}_g(\mathbf{W}_l, s_l)$ w.r.t. $\mathbf{W}_l$.

Now, $\mathcal{S}$ is non-differentiable, so we use sub-gradient which leads to the following update equation:

$$\begin{aligned}&\mathbf{W}_l^{(t+1)} \leftarrow (1 - \eta_t \cdot \lambda)\mathbf{W}_l^{(t)} \\ &- \eta_t \nabla_{\mathcal{S}_g(\mathbf{W}_l, s_l)} \mathcal{L}(\mathcal{S}_g(\mathcal{W}^{(t)}, \boldsymbol{s}), \mathcal{D}) \odot \mathbf{1}\left\{\mathcal{S}_g(\mathbf{W}_l^{(t)}, s_l) \neq 0\right\},\end{aligned} \tag{4}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function and $A \odot B$ denotes element-wise (Hadamard) product of tensors A and B.

Now, if g is a continuous function, then using the STR (2) and (1), it is clear that $\mathcal{L}(\mathcal{S}_g(\mathcal{W}, \boldsymbol{s}), \mathcal{D})$ is a continuous function of $\boldsymbol{s}$. Further, sub-gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{s}$, can be computed and uses for gradient descent on $\boldsymbol{s}$ as well; see Appendix A.2. Algorithm 1 in the Appendix shows the implementation of STR on 2D convolution along with extensions to global, per-filter & per-weight sparsity. STR can be modified and applied on the eigenvalues of a parameter tensor, instead of individual entries mentioned above, resulting in low-rank tensors; see Section 4.2.1 for further details. Note that $\boldsymbol{s}$ also has the same weight-decay parameter λ.

Naturally, g plays a critical role here, as a sharp g can lead to an arbitrary increase in threshold leading to poor accuracy while a flat g can lead to slow learning. Practical considerations for choice of g are discussed in Appendix A.1. For the experiments, g is set as the Sigmoid function for unstructured sparsity and the exponential function for structured sparsity. Typically, $\{s_l\}_{l\in[L]}$ are initialized with s_{init} to ensure that the thresholds $\{\alpha_l = g(s_l)\}_{l\in[L]}$ start close to 0. Figure 1 shows that the thresholds' dynamics are guided by a combination of gradients from $\mathcal{L}$ and the weight-decay on $\boldsymbol{s}$. Further, the overall sparsity budget for STR is not set explicitly. Instead, it is controlled by the weight-decay parameter (λ), and can be further fine-tuned using s_{init}. Interestingly, this curve is similar to the handcrafted heuristic for thresholds defined in (Narang et al., 2019). Figure 2 shows the overall learnt sparsity budget for ResNet50 during training. The curve looks similar to GMP (Zhu & Gupta, 2017) sparsification heuristic, however, STR learns it via backpropagation and SGD.

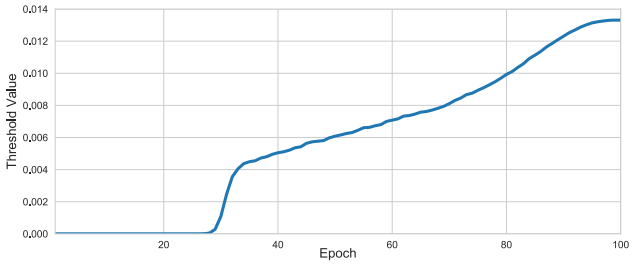

Figure 1. The learnt threshold parameter, $\alpha = g(s)$, for layer 10 in 90% sparse ResNet50 on ImageNet-1K over the course of training.

Finally, each parameter tensor learns a different threshold value, $\{\alpha_l\}_{l\in[L]}$, resulting in unique final thresholds across

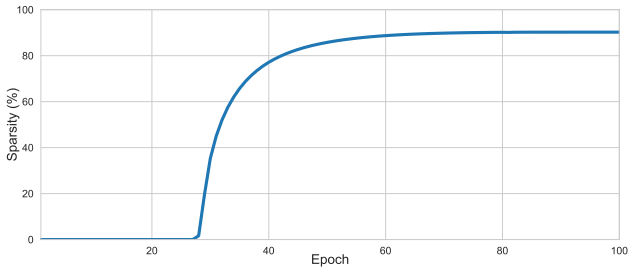

Figure 2. The progression of the learnt overall budget for 90% sparse ResNet50 on ImageNet-1K over the course of training.

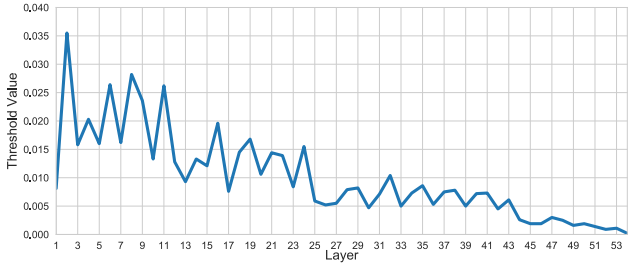

Figure 3. The final learnt threshold values, $[\alpha_l]_{l=1}^{54} = [g(s_l)]_{l=1}^{54}$, for all the layers in 90% sparse ResNet50 on ImageNet-1K.

the layers, as shown in Figure 3 for ResNet50. This, in turn, results in the non-uniform sparsity budget (see Figure 6) which is empirically shown to be effective in increasing prediction accuracy while reducing FLOPs. Moreover, (4) shows that the gradient update itself is sparse as gradient of $\mathcal{L}$ is multiplied with an indicator function of $\mathcal{S}_g(\mathbf{W}_l) \neq 0$ which gets sparser over iterations (Figure 2). So STR addresses both the issues with standard PGD methods (Hard/-Soft Thresholding) that we mentioned above.

3.1. Analysis

The reparameterization trick using the projection operator's functional form can be used for standard constrained optimization problems as well (assuming the projection operator has a closed-form). However, it is easy to show that in general, such a method need not converge to the optimal solution even for convex functions over convex sets. This raises a natural question about the effectiveness of the technique for sparse weights learning problem. It turns out that for sparsity constrained problems, STR is very similar to backward pruning (Hastie et al., 2009) which is a well-known technique for sparse regression. Note that, similar to Hard/-Soft Thresholding, standard backward pruning also does not support differentiable tuning thresholds which makes it challenging to apply it to DNNs.

To further establish this connection, let's consider a standard sparse regression problem where $\mathbf{y} = \mathbf{X}\mathbf{w}^*$, $\mathbf{X}_{ij} \sim \mathcal{N}(0,1)$, and $\mathbf{X} \in \mathbb{R}^{n\times d}$. $\mathbf{w}^* \in \{0,1\}^d$ has $r \ll d$ non-zeros, and $d \gg n \gg r\log d$. Due to the initialization, $g(s) \approx 0$ in initial few iterations. So, gradient descent converges to the least ℓ_2-norm regression solution. That is, $\mathbf{w} = \mathbf{U}\mathbf{U}^T\mathbf{w}^*$ where $\mathbf{U} \in \mathbb{R}^{d\times n}$ is the right singular vector matrix of $\mathbf{X}$ and is a random n-dimensional subspace. As $\mathbf{U}$ is a random subspace. Since $n \gg r\log d$, $\mathbf{U}_S\mathbf{U}_S^T \approx \frac{r}{d}\cdot I$ where $S = \text{supp}(\mathbf{w}^*)$, and $\mathbf{U}_S$ indexes rows of $\mathbf{U}$ corresponding to S. That is, $\min_{j\in S}\left|\mathbf{U}_j\cdot\mathbf{U}^T\mathbf{w}^*\right| \geq 1 - o(1)$. On the other hand, $\left|\mathbf{U}_j\cdot\mathbf{U}_S^T\mathbf{w}^*\right| \lesssim \frac{\sqrt{nr}}{d}\sqrt{\log d}$ with high probability for $j \notin S$. As $n \gg r\log d$, almost all the elements of $\text{supp}(\mathbf{w}^*)$ will be in top $\mathcal{O}(n)$ elements of $\mathbf{w}$. Furthermore, $\mathbf{X}\mathcal{S}_g(\mathbf{w}, s) = \mathbf{y}$, so $|s|$ would decrease significantly via weight-decay and hence $g(s)$ becomes large enough to prune all but say $\mathcal{O}(n)$ elements. Using a similar argument as above, leads to further pruning of $\mathbf{w}$, while ensuring recovery of almost all elements in $\text{supp}(\mathbf{w}^*)$.

4. Experiments

This section showcases the experimentation followed by the observations from applying STR for (a) unstructured sparsity in CNNs and (b) structured sparsity in RNNs.

4.1. Unstructured Sparsity in CNNs

4.1.1. Experimental Setup

ImageNet-1K (Deng et al., 2009) is a widely used large-scale image classification dataset with 1K classes. All the CNN experiments presented are on ImageNet-1K. ResNet50 (He et al., 2016) and MobileNetV1 (Howard et al., 2017) are two popular CNN architectures. ResNet50 is extensively used in literature to show the effectiveness of sparsity in CNNs. Experiments on MobileNetV1 argue for the generalizability of the proposed technique (STR). Dataset and models' details can be found in Appendix A.7.

STR was compared against strong state-of-the-art baselines in various sparsity regimes including GMP (Gale et al., 2019), DSR (Mostafa & Wang, 2019), DNW (Wortsman et al., 2019), SNFS (Dettmers & Zettlemoyer, 2019), RigL (Evci et al., 2020) and DPF (Lin et al., 2020). GMP and DNW always use a uniform sparsity budget. RigL, SNFS, DSR, and DPF were compared in their original form. Exceptions for the uniform sparsity are marked in Table 1. The "+ ERK" suffix implies the usage of ERK budget (Evci et al., 2020) instead of the original sparsity budget. Even though VD (Molchanov et al., 2017) achieves state-of-the-art results, it is omitted due to the 2× memory and 4× compute footprint during training. Typically VD and IMP use a global threshold for global sparsity (GS) (Han et al., 2015) which can also be learnt using STR. The unstructured sparsity experiments presented compare the techniques which induce layer-wise sparsity. Note that STR is generalizable to other scenarios as well. Open-source implementations,

pre-trained models, and reported numbers of the available techniques were used as the baselines. Experiments were run on a machine with 4 NVIDIA Titan X (Pascal) GPUs.

All baselines use the hyperparameter settings defined in their implementations/papers. The experiments for STR use a batch size of 256, cosine learning rate routine and are trained for 100 epochs following the hyperparameter settings in (Wortsman et al., 2019) using SGD + momentum. STR has weight-decay (λ) and s_{init} hyperparameters to control the overall sparsity in CNNs and can be found in Appendix A.6. GMP$_{1.5\times}$ (Gale et al., 2019) and RigL$_{5\times}$ (Evci et al., 2020) show that training the networks longer increases accuracy. However, due to the limited compute and environmental concerns (Schwartz et al., 2019), all the experiments were run only for around 100 epochs ($\sim$3 days each). Unstructured sparsity in CNNs with STR is enforced by learning one threshold per-layer as shown in Figure 3. PyTorch STRConv code can be found in Algorithm 1 of Appendix.

4.1.2. ResNet50 on ImageNet-1K

A fully dense ResNet50 trained on ImageNet-1K has 77.01% top-1 validation accuracy. STR is compared extensively to other baselines on ResNet50 in the sparsity ranges of 80%, 90%, 95%, 96.5%, 98%, and 99%. Table 1 shows that DNW and GMP are state-of-the-art among the baselines across all the aforementioned sparsity regimes. As STR might not be able to get exactly to the sparsity budget, numbers are reported for the models which nearby. Note that the 90.23% sparse ResNet50 on ImageNet-1K with STR is referred to as the 90% sparse ResNet50 model learnt with STR.

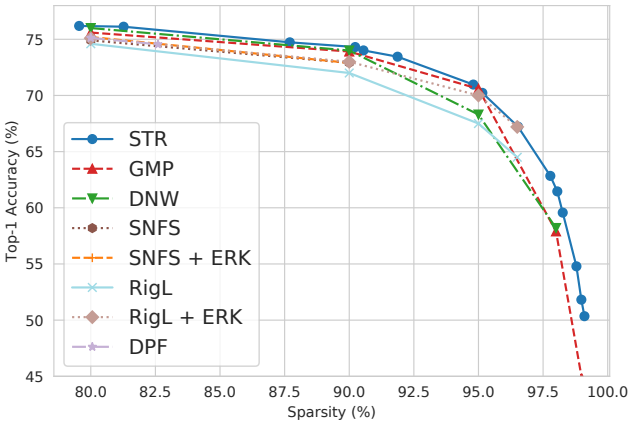

Figure 4. STR forms a frontier curve over all the baselines in all sparsity regimes showing that it is the state-of-the-art for unstructured sparsity in ResNet50 on ImageNet-1K.

STR comfortably beats all the baselines across all the sparsity regimes as seen in Table 1 and is the state-of-the-art for unstructured sparsity. Figure 4 shows that STR forms a frontier curve encompassing all the baselines at all the levels of sparsity. Very few methods are stable in the ultra sparse regime of 98-99% sparsity and GMP can achieve

Table 1. STR is the state-of-the-art for unstructured sparsity in ResNet50 on ImageNet-1K while having lesser inference cost (FLOPs) than the baselines across all the sparsity regimes. * and $^\#$ imply that the first and last layer are dense respectively. Baseline numbers reported from their respective papers/open-source implementations and models. FLOPs do not include batch-norm.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
ResNet-50	77.01	25.6M	0.00	4.09G
GMP	75.60	5.12M	80.00	818M
DSR$^{*\#}$	71.60	5.12M	80.00	1.23G
DNW	76.00	5.12M	80.00	818M
SNFS	74.90	5.12M	80.00	-
SNFS + ERK	75.20	5.12M	80.00	1.68G
RigL*	74.60	5.12M	80.00	920M
RigL + ERK	75.10	5.12M	80.00	1.68G
DPF	75.13	5.12M	80.00	818M
STR	**76.19**	5.22M	79.55	**766M**
STR	**76.12**	**4.47M**	**81.27**	**705M**
GMP	73.91	2.56M	90.00	409M
DNW	74.00	2.56M	90.00	409M
SNFS	72.90	2.56M	90.00	1.63G
SNFS + ERK	72.90	2.56M	90.00	960M
RigL*	72.00	2.56M	90.00	515M
RigL + ERK	73.00	2.56M	90.00	960M
DPF$^\#$	74.55	4.45M	82.60	411M
STR	**74.73**	3.14M	87.70	**402M**
STR	**74.31**	**2.49M**	**90.23**	**343M**
STR	**74.01**	**2.41M**	**90.55**	**341M**
GMP	70.59	1.28M	95.00	204M
DNW	68.30	1.28M	95.00	204M
RigL*	67.50	1.28M	95.00	317M
RigL + ERK	70.00	1.28M	95.00	$\sim$600M
STR	**70.97**	1.33M	94.80	**182M**
STR	70.40	**1.27M**	**95.03**	**159M**
STR	70.23	**1.24M**	**95.15**	**162M**
RigL*	64.50	0.90M	96.50	257M
RigL + ERK	67.20	0.90M	96.50	$\sim$500M
STR	**67.78**	0.99M	96.11	**127M**
STR	**67.22**	**0.88M**	**96.53**	**117M**
GMP	57.90	0.51M	98.00	82M
DNW	58.20	0.51M	98.00	82M
STR	**62.84**	0.57M	97.78	**80M**
STR	**61.46**	**0.50M**	**98.05**	**73M**
STR	**59.76**	**0.45M**	**98.22**	**68M**
GMP	44.78	0.26M	99.00	41M
STR	**54.79**	0.31M	98.79	54M
STR	**51.82**	0.26M	98.98	47M
STR	**50.35**	**0.23M**	**99.10**	44M

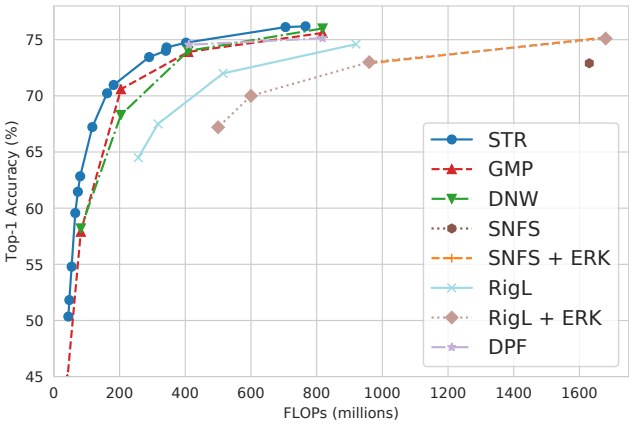

Figure 5. STR results in ResNet50 models on ImageNet-1K which have the lowest inference cost (FLOPs) for any given accuracy.

99% sparsity. STR is very stable even in the ultra sparse regime, as shown in Table 1 and Figure 4, while being up to 10% higher in accuracy than GMP at 99% sparsity.

STR induces non-uniform sparsity across layers, Table 1 and Figure 5 show that STR produces models which have lower or similar inference FLOPs compared to the baselines while having better prediction accuracy in all the sparsity regimes. This hints at the fact that STR could be redistributing the parameters thereby reducing the FLOPs. In the 80% sparse models, STR is at least 0.19% better in accuracy than the baselines while having at least 60M (6.5%) lesser FLOPs. Similarly, STR has state-of-the-art accuracy in 90%, 95%, and 96.5% sparse regimes while having at least 68M (16.5%), 45M (22%) and 140M (54%) lesser FLOPs than the best baselines respectively. In the ultra sparse regime of 98% and 99% sparsity, STR has similar or slightly higher FLOPs compared to the baselines but is up to 4.6% and 10% better in accuracy respectively. Table 1 summarizes that the non-uniform sparsity baselines like SNFS, SNFS+ERK, and RigL+ERK can have up to 2-4× higher inference cost (FLOPs) due to non-optimal layer-wise distribution of the parameter weights.

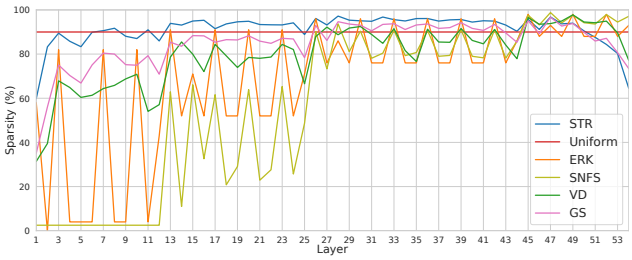

Figure 6. Layer-wise sparsity budget for the 90% sparse ResNet50 models on ImageNet-1K using various sparsification techniques.

Observations: STR on ResNet50 shows some interesting

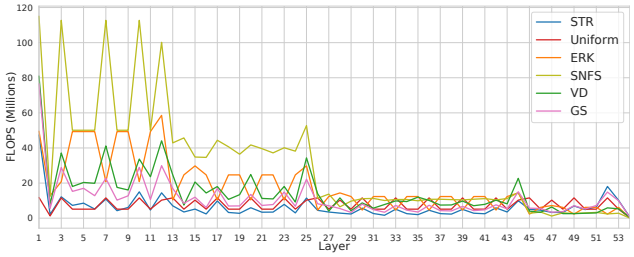

Figure 7. Layer-wise FLOPs budget for the 90% sparse ResNet50 models on ImageNet-1K using various sparsification techniques.

observations related to sparsity and inference cost (FLOPs). These observations will be further discussed in Section 5:

1. STR is state-of-the-art for unstructured sparsity.
2. STR minimizes inference cost (FLOPs) while maintaining accuracy in the 80-95% sparse regime.
3. STR maximizes accuracy while maintaining inference cost (FLOPs) in 98-99% ultra sparse regime.
4. STR learns a non-uniform layer-wise sparsity, shown in Figure 6, which shows that the initial layers of the CNN can be sparser than that of the existing non-uniform sparsity methods. All the learnt non-uniform budgets through STR can be found in Appendix A.3.
5. Figure 6 also shows that the last layers through STR are denser than that of the other methods which is contrary to the understanding in the literature of non-uniform sparsity (Mostafa & Wang, 2019; Dettmers & Zettlemoyer, 2019; Evci et al., 2020; Gale et al., 2019). This leads to a sparser backbone for transfer learning. The backbone sparsities can be found in Appendix A.3.
6. Figure 7 shows the layer-wise FLOPs distribution for the non-uniform sparsity methods. STR adjusts the FLOPs across layers such that it has lower FLOPs than the baselines. Note that the other non-uniform sparsity budgets lead to heavy compute overhead in the initial layers due to denser parameter tensors.

STR can also induce global sparsity (GS) (Han et al., 2015) with similar accuracy at $\sim 2\times$ FLOPs compared to layer-wise for 90-98% sparsity (details in Appendix A.5.1).

4.1.3. MobileNetV1 on ImageNet-1K

MobileNetV1 was trained on ImageNet-1K for unstructured sparsity with STR to ensure generalizability. Since GMP is the state-of-the-art baseline as shown earlier, STR was only compared to GMP for 75% and 90% sparsity regimes. A fully dense MobileNetV1 has a top-1 accuracy of 71.95% on ImageNet-1K. GMP (Zhu & Gupta, 2017) has the first layer and depthwise convolution layers dense for MobileNetV1 to ensure training stability and maximize accuracy.

Table 2 shows the STR is at least 0.65% better than GMP for

Table 2. STR is up to 3% higher in accuracy while having 33% lesser inference cost (FLOPs) for MobileNetV1 on ImageNet-1K.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
MobileNetV1	71.95	4.21M	0.00	569M
GMP	67.70	1.09M	74.11	163M
STR	**68.35**	**1.04M**	**75.28**	**101M**
STR	66.52	**0.88M**	**79.07**	**81M**
GMP	61.80	0.46M	89.03	82M
STR	**64.83**	0.60M	85.80	**55M**
STR	**62.10**	0.46M	89.01	**42M**
STR	61.51	**0.44M**	**89.62**	**40M**

75% sparsity, while having at least 62M (38%) lesser FLOPs. More interestingly, STR has state-of-the-art accuracy while having up to 50% (40M) lesser FLOPs than GMP in the 90% sparsity regime. All the observations made for ResNet50 hold for MobileNetV1 as well. The sparsity and FLOPs distribution across layers can be found in Appendix A.4.

4.2. Structured Sparsity in RNNs

4.2.1. Experimental Setup

Google-12 is a speech recognition dataset that has 12 classes made from the Google Speech Commands dataset (Warden, 2018). HAR-2 is a binarized version of the 6-class Human Activity Recognition dataset (Anguita et al., 2012). These two datasets stand as compelling cases for on-device resource-efficient machine learning at the edge. Details about the datasets can be found in Appendix A.7.

FastGRNN (Kusupati et al., 2018) was proposed to enable powerful RNN models on resource-constrained devices. FastGRNN relies on making the RNN parameter matrices low-rank, sparse and quantized. As low-rank is a form of structured sparsity, experiments were done to show the effectiveness of STR for structured sparsity. The input vector to the RNN at each timestep and hidden state have D & $\hat{D}$ dimensionality respectively. FastGRNN has two parameter matrices, $\mathbf{W} \in \mathbb{R}^{D\times\hat{D}}$, $\mathbf{U} \in \mathbb{R}^{\hat{D}\times\hat{D}}$ which are reparameterized as product of low-rank matrices, $\mathbf{W} = \mathbf{W}_1\mathbf{W}_2$, and $\mathbf{U} = \mathbf{U}_1\mathbf{U}_2$ where $\mathbf{W}_1 \in \mathbb{R}^{D\times r_W}$, $\mathbf{W}_2 \in \mathbb{R}^{r_W\times\hat{D}}$, and $(\mathbf{U}_1)^\top, \mathbf{U}_2 \in \mathbb{R}^{r_U\times\hat{D}}$. r_W, r_U are the ranks of the respective matrices. In order to apply STR, the low-rank reparameterization can be changed to $\mathbf{W} = (\mathbf{W}_1 \odot \mathbf{1}\mathbf{m}_\mathbf{W}^\top)\mathbf{W}_2$, and $\mathbf{U} = (\mathbf{U}_1 \odot \mathbf{1}\mathbf{m}_\mathbf{U}^\top)\mathbf{U}_2$ where $\mathbf{m}_\mathbf{W} = \mathbf{1}_D$, and $\mathbf{m}_\mathbf{U} = \mathbf{1}_{\hat{D}}$, $\mathbf{W}_1 \in \mathbb{R}^{D\times D}$, $\mathbf{W}_2 \in \mathbb{R}^{D\times\hat{D}}$, and $\mathbf{U}_1, \mathbf{U}_2 \in \mathbb{R}^{\hat{D}\times\hat{D}}$. To learn the low-rank, STR is applied on the $\mathbf{m}_\mathbf{W}$, and $\mathbf{m}_\mathbf{U}$ vectors. Learning low-rank with STR on $\mathbf{m}_\mathbf{W}$, $\mathbf{m}_\mathbf{U}$ can be thought as inducing unstructured sparsity on the two trainable vectors aiming for the right r_W, and r_U.

The baseline is low-rank FastGRNN where the ranks of the matrices are preset (Kusupati et al., 2018). EdgeML (Dennis et al.) FastGRNN was used for the experiments with the hyperparameters suggested in the paper and is referred to as vanilla training. Hyperparameters for the models can be found in Appendix A.6.

4.2.2. FastGRNN on Google-12 and HAR-2

Table 3 presents the results for low-rank FastGRNN with vanilla training and STR. Full-rank non-reparameterized FastGRNN has an accuracy of 92.60% and 96.10% on Google-12 and HAR-2 respectively. STR outperforms vanilla training by up to 1.67% in four different model-size reducing rank settings on Google-12. Similarly, on HAR-2, STR is better than vanilla training in all the rank settings by up to 2.47%. Note that the accuracy of the low-rank models obtained by STR is either better or on-par with the full rank models while being around 50% and 70% smaller in size (low-rank) for Google-12 and HAR-2 respectively.

Table 3. STR can induce learnt low-rank in FastGRNN resulting in up to 2.47% higher accuracy than the vanilla training.

Google-12			HAR-2		
(r_W, r_U)	Accuracy (%)		(r_W, r_U)	Accuracy (%)	
	Vanilla Training	STR		Vanilla Training	STR
Full rank (32, 100)	92.30	-	Full rank (9, 80)	96.10	-
(12, 40)	92.79	**94.45**	(9, 8)	94.06	**95.76**
(11, 35)	92.86	**94.42**	(9, 7)	93.15	**95.62**
(10, 31)	92.86	**94.25**	(8, 7)	94.88	**95.59**
(9, 24)	93.18	**94.45**			

These experiments for structured sparsity in RNNs show that STR can be applied to obtain low-rank parameter tensors. Similarly, STR can be extended for filter/channel pruning and block sparsity (He et al., 2017; Huang & Wang, 2018; Liu et al., 2019) and details for this adaptation can be found in Appendix A.5.2.

5. Discussion and Drawbacks

STR's usage for unstructured sparsity leads to interesting observations as noted in Section 4.1.2. It is clear from Table 1 and Figures 4, 5 that STR achieves state-of-the-art accuracy for all the sparsity regimes and also reduces the FLOPs in doing so. STR helps in learning non-uniform sparsity budgets which are intriguing to study as an optimal non-uniform sparsity budget can ensure minimization of FLOPs while maintaining accuracy. Although it is not clear why STR's learning dynamics result in a non-uniform budget that minimizes FLOPs, the reduction in FLOPs is due to the better redistribution of parameters across layers.

Non-uniform sparsity budgets learnt by STR have the ini-

tial and middle layers to be sparser than the other methods while making the last layers denser. Conventional wisdom suggests that the initial layers should be denser as the early loss of information would be hard to recover, this drives the existing non-uniform sparsity heuristics. As most of the parameters are present in the deeper layers, the existing methods tend to make them sparser while not affecting the FLOPs by much. STR, on the other hand, balances the FLOPs and sparsity across the layers as shown in Figures 6, 7 making it a lucrative and efficient choice. The denser final layers along with sparser initial and middle layers point to sparser CNN backbones obtained using STR. These sparse backbones can be viable options for efficient representation/transfer learning for downstream tasks.

Table 4. Effect of various layer-wise sparsity budgets when used with DNW for ResNet50 on ImageNet-1K.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
Uniform	74.00	2.56M	90.00	409M
ERK	**74.10**	2.56M	90.00	960M
Budget from STR	74.01	**2.49M**	**90.23**	**343M**
Uniform	68.30	1.28M	95.00	204M
Budget from STR	**69.72**	1.33M	94.80	182M
Budget from STR	68.01	**1.24M**	**95.15**	**162M**

Table 4 shows the effectiveness/transferability of the learnt non-uniform budget through STR for 90% sparse ResNet50 on ImageNet-1K using DNW (Wortsman et al., 2019). DNW typically takes in a uniform sparsity budget and has an accuracy of 74% for a 90% sparse ResNet50. Using ERK non-uniform budget for 90% sparsity results in a 0.1% increase in accuracy at the cost 2.35× inference FLOPs. Training DNW with the learnt budget from STR results in a reduction of FLOPs by 66M (16%) while maintaining accuracy. In the 95% sparsity regime, the learnt budget can improve the accuracy of DNW by up to 1.42% over uniform along with a reduction in FLOPs by at least 22M (11%).

Table 5. Effect of various layer-wise sparsity budgets when used with GMP for ResNet50 on ImageNet-1K.

Method	Top-1 Acc (%)	Params	Sparsity (%)	FLOPs
Uniform	73.91	2.56M	90.00	409M
Budget from STR	**74.13**	**2.49M**	**90.23**	**343M**
Uniform	57.90	0.51M	98.00	82M
Budget from STR	**59.47**	**0.50M**	**98.05**	**73M**

Similarly, these budgets can also be used for other methods like GMP (Zhu & Gupta, 2017). Table 5 shows that the learnt sparsity budgets can lead to an increase in accuracy by 0.22% and 1.57% in 90% and 98% sparsity regimes respectively when used with GMP. Accuracy gains over uniform sparsity are also accompanied by a significant reduction in inference FLOPs. Note that the learnt non-uniform sparsity budgets can also be obtained using smaller representative datasets instead of expensive large-scale experiments.

The major drawback of STR is the tuning of the weight-decay parameter, λ and finer-tuning with s_{init} to obtain the targeted overall sparsity. One way to circumvent this issue is to freeze the non-uniform sparsity distribution in the middle of training when the overall sparsity constraints are met and train for the remaining epochs. This might not potentially give the best results but can give a similar budget which can be then transferred to methods like GMP or DNW. Another drawback of STR is the function g for the threshold. The stability, expressivity, and sparsification capability of STR depends on g. However, it should be noted that sigmoid and exponential functions work just fine, as g, for STR.

6. Conclusions

This paper proposed Soft Threshold Reparameterization (STR), a novel use of the soft-threshold operator, for the weights in DNN, to smoothly induce sparsity while learning layer-wise pruning thresholds thereby obtaining a non-uniform sparsity budget. Extensive experimentation showed that STR is state-of-the-art for unstructured sparsity in CNNs for ImageNet-1K while also being effective for structured sparsity in RNNs. Our method results in sparse models that have significantly lesser inference costs than the baselines. In particular, STR achieves the same accuracy as the baselines for 90% sparse MobileNetV1 with 50% lesser FLOPs. STR has ∼10% higher accuracy than the existing methods in ultra sparse (99%) regime for ResNet50 showing the effectiveness of the learnt non-uniform sparsity budgets. STR can also induce low-rank structure in RNNs while increasing the prediction accuracy showing the generalizability of the proposed reparameterization. Finally, STR is easy to adapt and the learnt budgets are transferable.

Acknowledgments

We are grateful to Keivan Alizadeh, Tapan Chugh, Tim Dettmers, Erich Elsen, Utku Evci, Daniel Gordon, Gabriel Ilharco, Sarah Pratt, James Park, Mohammad Rastegari and Matt Wallingford for helpful discussions and feedback. Mitchell Wortsman is in part supported by AI2 Fellowship in AI. Sham Kakade acknowledges funding from the Washington Research Foundation for Innovation in Data-intensive Discovery, and the NSF Awards CCF-1637360, CCF-1703574, and CCF-1740551. Ali Farhadi acknowledges funding from the NSF Awards IIS 1652052, IIS 17303166, DARPA N66001-19-2-4031, 67102239 and gifts from Allen Institute for Artificial Intelligence.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

Alizadeh, K., Farhadi, A., and Rastegari, M. Butterfly transform: An efficient fft based neural architecture design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International Workshop on Ambient Assisted Living*, pp. 216–223. Springer, 2012. URL https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones.

Ashby, M., Baaij, C., Baldwin, P., Bastiaan, M., Bunting, O., Cairncross, A., Chalmers, C., Corrigan, L., Davis, S., van Doorn, N., et al. Exploiting unstructured sparsity on next-generation datacenter hardware.

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

Beck, A. and Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

Candes, E., Tao, T., et al. The dantzig selector: Statistical estimation when p is much larger than n. *The annals of Statistics*, 35(6):2313–2351, 2007.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dennis, D. K., Gaurkar, Y., Gopinath, S., Gupta, C., Jain, M., Kumar, A., Kusupati, A., Lovett, C., Patil, S. G., and Simhadri, H. V. EdgeML: Machine Learning for resource-constrained edge devices. URL https://github.com/Microsoft/EdgeML.

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

Donoho, D. L. De-noising by soft-thresholding. *IEEE transactions on information theory*, 41(3):613–627, 1995.

Elsen, E., Dukhan, M., Gale, T., and Simonyan, K. Fast sparse convnets. *arXiv preprint arXiv:1911.09723*, 2019.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, 2020.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 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., Yang, T.-J., and Choi, E. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1586–1595, 2018.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015.

Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pp. 164–171, 1993.

Hastie, T., Tibshirani, R., and Friedman, J. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, 2018.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Huang, Z. and Wang, N. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 304–320, 2018.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference. BMVA Press*, 2014.

Jain, P., Tewari, A., and Kar, P. On iterative hard thresholding methods for high-dimensional m-estimation. In *Advances in Neural Information Processing Systems*, pp. 685–693, 2014.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., and Varma, M. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, pp. 9017–9028, 2018.

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

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

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 *International Conference on Learning Representations*, 2017.

Lin, T., Stich, S. U., Barba, L., Dmitriev, D., and Jaggi, M. Dynamic model pruning with feedback. In *International Conference on Learning Representations*, 2020.

Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 806–814, 2015.

Liu, J., Xu, Z., Shi, R., Cheung, R. C. C., and So, H. K. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2020.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

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

Lu, Z., Sindhwani, V., and Sainath, T. N. Learning compact recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5960–5964. IEEE, 2016.

Luo, J.-H., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pp. 5058–5066, 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*, 9 (1):2383, 2018.

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2498–2507. JMLR. org, 2017.

Mostafa, H. and Wang, X. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655, 2019.

Narang, S., Elsen, E., Diamos, G., and Sengupta, S. Exploring sparsity in recurrent neural networks. In *International Conference on Learning Representations*, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

Patil, S. G., Dennis, D. K., Pabbaraju, C., Shaheer, N., Simhadri, H. V., Seshadri, V., Varma, M., and Jain, P. Gesturepod: Enabling on-device gesture-based interaction for white cane users. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pp. 403–415, 2019.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11893–11902, 2020.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Renda, A., Frankle, J., and Carbin, M. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020.

Roy, D., Srivastava, S., Kusupati, A., Jain, P., Varma, M., and Arora, A. One size does not fit all: Multi-scale, cascaded rnns for radar classification. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pp. 1–10, 2019.

Savarese, P., Silva, H., and Maire, M. Winning the lottery with continuous sparsification. *arXiv preprint arXiv:1912.04427*, 2019.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. Green AI. *arXiv preprint arXiv:1907.10597*, 2019.

Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018. URL http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pp. 2074–2082, 2016.

Wortsman, M., Farhadi, A., and Rastegari, M. Discovering neural wirings. In *Advances In Neural Information Processing Systems*, pp. 2680–2690, 2019.

Xiao, X., Wang, Z., and Rajasekaran, S. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Advances in Neural Information Processing Systems*, pp. 13681–13691, 2019.

Yu, J. and Huang, T. Network slimming by slimmable networks: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 2019.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pp. 3592–3602, 2019.

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