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), and 3) Flash - model size. Various techniques are proposed to make DNNs efficient including model pruning (sparsity) (Han et al., 2015), knowledge distillation (Bucilă 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 $3 \times 3$ 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 1University of Washington, USA 2Allen Institute for Artificial Intelligence, USA 3Microsoft Research, India. Correspondence to: Aditya Kusupati <kusupati@cs.washington.edu>.


---

1One Multiply-Add is counted as one FLOP
Motivated by the above-mentioned challenges, this work 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 $W_l$ of the $l$-th layer in the DNN: $S(W_l, \alpha_l) := \text{sign}(W_l) \cdot \text{ReLU}(|W_l| - \alpha_l)$ where $\alpha_l$ is the pruning threshold for the $l$-th layer. As the DNN loss can be written as a continuous function of $\alpha_l$’s, we can use backpropagation to learn layer-specific $\alpha_l$ 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 version (Donoho, 1995) of a weight tensor $W_l$.

Structured sparsity takes structure into account 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; Evcı 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 Reparaterization (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) (Evcı 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.

DNN (Wortsman et al., 2019) and Dynamic Pruning with Feedback (DPF) (Lin et al., 2020) fall between both as DNN 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; Candès et al., 2007; Jain et al., 2014), and is modeled as:

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

where $D := \{x_i \in \mathbb{R}^d, y_i \in \mathbb{R} \}_{i \in [n]}$ is the observed data, $\mathcal{L}$ is the loss function, $W$ 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 \( S_g(\mathcal{W}; s) \) be the projection of \( \mathcal{W} \) parameterized by \( s \) and function \( g \). \( S \) is applied to each element of \( \mathcal{W} \) and is defined as:

\[
S_g(w, s) := \text{sign}(w) \cdot \text{ReLU}(|w| - g(s)),
\]

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

Reparameterizing the optimization problem with \( S \) modifies (note that it is not equivalent) it to:

\[
\min_{S_g(\mathcal{W}; s)} \mathcal{L}(S_g(\mathcal{W}; s), D).
\]

For \( L \)-layer DNN architectures, we divide \( \mathcal{W} \) into: \( \mathcal{W} = \left[ W_l \right]_{l=1}^L \) where \( W_l \) is the parameter tensor for the \( l \)-th layer. As mentioned earlier, different layers of DNNs are unique and 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, \( s = \{s_1, \ldots, s_L\} \).

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

\[
W_l^{(t+1)} \leftarrow (1 - \eta_t \cdot \lambda)W_l^{(t)} - \eta_t \nabla_{S_g(W_l, s_l)} \mathcal{L}(S_g(W_l^{(t)}, s_l), D) \odot \nabla_{W_l} S_g(W_l, s_l),
\]

where \( \eta_t \) is the learning rate at the \( t \)-th step, and \( \lambda \) is the \( L_2 \) regularization (weight-decay) hyper-parameter. \( \nabla_{W_l} S_g(W_l, s_l) \) is the gradient of \( S_g(W_l, s_l) \) w.r.t. \( W_l \).

Now, \( S \) is non-differentiable, so we use sub-gradient which leads to the following update equation:

\[
W_l^{(t+1)} \leftarrow (1 - \eta_t \cdot \lambda)W_l^{(t)} - \eta_t \nabla_{S_g(W_l, s_l)} \mathcal{L}(S_g(W_l^{(t)}, s_l), D) \odot 1 \{ S_g(W_l^{(t)}, s_l) \neq 0 \},
\]

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, \( \{\alpha_l\}_{l=1}^L \), resulting in unique final thresholds across
Soft Threshold Weight Reparameterization for Learnable Sparsity

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 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,
Soft Threshold Weight Reparameterization for Learnable Sparsity

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 ($\lambda$) and $s_{\text{init}}$ 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$_{1.5\times}$ (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 (∼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.

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.

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

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.
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 $2\times$ higher inference cost (FLOPs) due to non-optimal layer-wise distribution of the parameter weights.

Observations: STR on ResNet50 shows some interesting 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
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, $W \in \mathbb{R}^{D \times D}$, $U \in \mathbb{R}^{\hat{D} \times \hat{D}}$ which are reparameterized as product of low-rank matrices, $W = W_1 W_2$, and $U = U_1 U_2$ where $W_1 \in \mathbb{R}^{D \times r_W}$, $W_2 \in \mathbb{R}^{r_W \times \hat{D}}$, and $(U_1)^T$, $U_2 \in \mathbb{R}^{\hat{D} \times \hat{D}}$, $r_W$, $r_U$ are the ranks of the respective matrices. In order to apply STR, the low-rank reparameterization can be changed to $W = (W_1 \odot 1_{mW}) W_2$, and $U = (U_1 \odot 1_{mU}) U_2$ where $mW = 1_D$, and $mU = 1_{\hat{D}}$. $W_1 \in \mathbb{R}^{D \times D}$, $W_2 \in \mathbb{R}^{D \times \hat{D}}$, and $U_1, U_2 \in \mathbb{R}^{\hat{D} \times \hat{D}}$. To learn the low-rank, STR is applied on the $mW$, and $mU$ vectors. Learning low-rank with STR on $mW$, $mU$ 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.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Acc (%)</th>
<th>Params</th>
<th>Sparsity (%)</th>
<th>FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>73.91</td>
<td>2.56M</td>
<td>90.00</td>
<td>409M</td>
</tr>
<tr>
<td>Budget from STR</td>
<td>74.13</td>
<td>2.49M</td>
<td>90.23</td>
<td>343M</td>
</tr>
<tr>
<td>Uniform</td>
<td>57.90</td>
<td>0.51M</td>
<td>98.00</td>
<td>82M</td>
</tr>
<tr>
<td>Budget from STR</td>
<td>59.47</td>
<td>0.50M</td>
<td>98.05</td>
<td>73M</td>
</tr>
</tbody>
</table>

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.

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. ST 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.
Soft Threshold Weight Reparameterization for Learnable Sparsity

References


Soft Threshold Weight Reparameterization for Learnable Sparsity


