SSGD: SPARSITY-PROMOTING STOCHASTIC GRADIENT DESCENT ALGORITHM FOR UNBIASED DNN PRUNING

Ching-Hua Lee[*], *Igor Fedorov*[†], *Bhaskar D. Rao*[*], *and Harinath Garudadri*[*]

[*]Department of ECE, University of California, San Diego
[†]ARM ML Research

ABSTRACT

While deep neural networks (DNNs) have achieved state-of-the-art results in many fields, they are typically over-parameterized. Parameter redundancy, in turn, leads to inefficiency. Sparse signal recovery (SSR) techniques, on the other hand, find compact solutions to over-complete linear problems. Therefore, a logical step is to draw the connection between SSR and DNNs. In this paper, we explore the application of iterative reweighting methods popular in SSR to learning efficient DNNs. By efficient, we mean sparse networks that require less computation and storage than the original, dense network. We propose a reweighting framework to learn sparse connections within a given architecture without biasing the optimization process, by utilizing the affine scaling transformation strategy. The resulting algorithm, referred to as Sparsity-promoting Stochastic Gradient Descent (SSGD), has simple gradient-based updates which can be easily implemented in existing deep learning libraries. We demonstrate the sparsification ability of SSGD on image classification tasks and show that it outperforms existing methods on the MNIST and CIFAR-10 datasets.

***Index Terms*—** Deep learning, sparse signal recovery, network pruning, iterative reweighting, affine scaling

1. INTRODUCTION

Deep neural networks (DNNs) have become extremely powerful models for many engineering tasks [1, 2, 3]. Due to the lack of principled architecture design techniques, DNNs are typically designed to be quite large, equipping the model with sufficient representation power [4]. Thus, DNNs can be over-parameterized [5], exhibiting more flexibility and complexity than strictly required by the data at hand [6]. Due to their significant redundancy, DNNs can waste storage and computational resources, restricting their deployment on real-time, embedded devices such as hearing aids [7].

Recently, compressing large DNNs into smaller ones for efficient processing has become an active research topic. Han et al. [8] have proposed a compression strategy composed of 3 stages: i) train a network to learn the importance of each connection instead of the parameter values; ii) apply hard thresholding to remove unimportant connections based on their magnitudes; iii) fine-tune the remaining connections, starting from the trained values, to regain accuracy. Since then, there has been a growing line of work on pruning DNNs [9, 10, 11, 12, 13, 14, 15]. Interestingly, sparse signal recovery (SSR) techniques [16, 17, 18, 19, 20] that have proved successful in learning compact and sparse solutions to linear problems have fueled the trend of efficient DNN research [21, 22, 23, 24, 25, 26, 27, 28].

This work was supported by NIH/NIDCD under Grants R01DC015436 and R33DC015046 and NSF/IIS under Award 1838830.

In this paper, we propose a framework for learning sparse connections of a given DNN architecture by adopting iterative reweighting methods well-known in SSR [29, 17, 19, 20]. Starting with an optimization formulation with sparsity regularization, we utilize the affine scaling transformation (AST) [30, 31] to arrive at algorithms that promote sparsity without biasing the optimization process. Existing regularization-based approaches for learning sparse DNNs [23, 24, 4, 6, 26, 27] use a sparsity-inducing regularizer such as the ℓ_1 norm weighted by a nonzero regularization coefficient λ. The proposed method, on the other hand, allows the regularization coefficient $\lambda \to 0^+$, thereby avoiding any bias incurred by the introduction of the regularizer while promoting sparsity.

The reweighting and AST strategies have recently been adopted in sparsity-aware adaptive filtering [32, 33]. Here we show that they can be successfully transferred to DNN learning. The work in [34] has explored these strategies for sparsifying pre-trained networks. We extend the idea and derive a gradient-based optimization formula for learning better network connectivity. The proposed algorithm, referred to as Sparsity-promoting Stochastic Gradient Descent (SSGD), admits simple updates in which a diagonal weighting is introduced on the gradient, and thus can be easily implemented in existing DNN libraries. We demonstrate the sparsification ability of SSGD for image classification tasks on the MNIST [35] and CIFAR-10 [36] datasets with convolutional neural networks (CNNs). The proposed method is shown to work well for both fully-connected (FC) and convolutional (CONV) layers, and be compatible with popular techniques like batch normalization [37] and dropout [38].

2. PROBLEM FORMULATION

Let D be a dataset of N input-output pairs $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=0}^{N-1}$. We consider the optimization problem:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, D) = \frac{1}{N} \sum_{n=0}^{N-1} L\left(h(\mathbf{x}_n; \boldsymbol{\theta}), \mathbf{y}_n\right), \tag{1}$$

where $J(\cdot,\cdot)$ is the empirical risk, $\boldsymbol{\theta}$ is the parameter set of the hypothesis $h(\cdot;\boldsymbol{\theta})$, and $L(\cdot,\cdot)$ corresponds to the loss function. For a DNN with M network parameters (weights and biases), we treat $\boldsymbol{\theta} = [\theta_0, \theta_1, ..., \theta_{M-1}]^T$ as a vector consisting of all M parameters.

In DNN training, stochastic gradient descent (SGD) is widely used for updating the parameters:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t, d_t), \tag{2}$$

where $\eta > 0$ is the learning rate, t is the timestep, d_t is a subset (mini-batch) of the training set D given to the model at timestep t, and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t, \cdot)$ denotes the gradient of the objective function with respect to $\boldsymbol{\theta}$ evaluated at $\boldsymbol{\theta}_t$. For the rest of the paper, we omit the dependency of $J(\cdot,\cdot)$ on the data for brevity.

3. PROPOSED FRAMEWORK

We propose to add a sparsity regularization term $G(\boldsymbol{\theta})$ to the empirical risk in (1) to promote sparse solutions $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \lambda G(\boldsymbol{\theta}), \tag{3}$$

where $G(\cdot)$ is a penalty function weighted by λ that induces sparsity in its argument. In the SSR literature, $G(\cdot)$ is usually referred to as the *general diversity measure* that serves as an alternative to the ℓ_0 "norm" for encouraging sparsity. We further define a *separable* diversity measure that has the form $G(\boldsymbol{\theta}) = \sum_{i=0}^{M-1} g(\theta_i)$, where $g(\cdot)$ has the following properties [29]:

Property 1: $g(u)$ is symmetric, i.e., $g(u) = g(-u) = g(|u|)$;
Property 2: $g(|u|)$ is monotonically increasing with $|u|$;
Property 3: $g(u)$ is finite;
Property 4: $g(u)$ is strictly concave in $|u|$ or u^2.

Any function that holds the above properties is a candidate for effective SSR algorithms.

3.1. Iterative reweighting frameworks

The iterative reweighting methods in SSR [29] are popular batch estimation algorithms for solving (3), particularly for linear models. By introducing a weighted ℓ_2 [17, 19] or ℓ_1 [20] norm term as an upper bound for $G(\boldsymbol{\theta})$ in each iteration, they form and solve for a new optimization problem to approach the optimal solution [29]. In the following, we show how the reweighting frameworks can be adapted to nonlinear DNN models.

In the iterative reweighted ℓ_2 framework, we deal with a weighted ℓ_2 norm minimization problem at timestep t:

$$\min_{\boldsymbol{\theta}} \quad J(\boldsymbol{\theta}) + \lambda \left\| \boldsymbol{\Omega}_t^{-1} \boldsymbol{\theta} \right\|_2^2, \tag{4}$$

while in the iterative reweighted ℓ_1 framework we approach the sparse solution by considering a weighted ℓ_1 norm minimization problem:

$$\min_{\boldsymbol{\theta}} \quad J(\boldsymbol{\theta}) + \lambda \left\| \boldsymbol{\Omega}_t^{-1} \boldsymbol{\theta} \right\|_1, \tag{5}$$

where $\boldsymbol{\Omega}_t = \text{diag}\{\omega_{i,t}\}$ is positive definite and each $\omega_{i,t}$ (the i-th diagonal element of $\boldsymbol{\Omega}_t$) is computed based on $\theta_{i,t}$ (the i-th entry of the current estimate $\boldsymbol{\theta}_t$), depending on which framework (reweighted ℓ_2 or ℓ_1) and diversity measure (choice of $G(\cdot)$) are used. In each timestep t, the matrix $\boldsymbol{\Omega}_t$ provides a surrogate function for the objective function in (3) to form a new minimization problem accordingly. This allows the algorithm to produce more focal estimates as optimization progresses [29]. Note that we make no assumptions about the form of the model hypothesis $h(\cdot; \boldsymbol{\theta})$ in (1).

For using the reweighted ℓ_2 framework, note that the function $g(u)$ of the diversity measure has to be concave in u^2 for Property 4; i.e., it satisfies $g(u) = f(u^2)$, where $f(z)$ is concave for $z \in \mathbb{R}_+$. Let d denote the differential operator. We have $\omega_{i,t}$ given by:

$$\omega_{i,t} = \left(\frac{\mathrm{d}f(z)}{\mathrm{d}z}\bigg|_{z=\theta_{i,t}^2} \right)^{-\frac{1}{2}}. \tag{6}$$

For using the reweighted ℓ_1 framework, the function $g(u)$ has to be concave in $|u|$ for Property 4; i.e., it satisfies $g(u) = f(|u|)$, where $f(z)$ is concave for $z \in \mathbb{R}_+$. In this case, $\omega_{i,t}$ is given by:

$$\omega_{i,t} = \left(\frac{\mathrm{d}f(z)}{\mathrm{d}z}\bigg|_{z=|\theta_{i,t}|} \right)^{-1}. \tag{7}$$

To utilize the proposed framework, we first choose an appropriate diversity measure $G(\boldsymbol{\theta})$ and then use (6) or (7) to obtain the update form of $\omega_{i,t}$. Several examples will be presented is Section 3.3.

3.2. Affine scaling transformation (AST)

Before proceeding, we reparameterize the problem in terms of the (affinely) scaled variable $\mathbf{q}$:

$$\mathbf{q} \triangleq \boldsymbol{\Omega}_t^{-1} \boldsymbol{\theta}, \tag{8}$$

in which $\boldsymbol{\Omega}_t$ is used as the *scaling matrix*. This step can be interpreted as the AST commonly employed by the interior point approach to solving optimization problems [30]. In the optimization literature, AST-based methods transform the original problem into an equivalent one, in which the current point is favorably positioned at the center of the feasible region [31].

Applying (8) to the objective functions in (4) and (5) results in the alternative forms $J(\boldsymbol{\Omega}_t \mathbf{q}) + \lambda \|\mathbf{q}\|_2^2$ and $J(\boldsymbol{\Omega}_t \mathbf{q}) + \lambda \|\mathbf{q}\|_1$ for the reweighted ℓ_2 and ℓ_1 cases, respectively. Interestingly, if we set $\lambda = 0$ and perform minimization with respect to $\mathbf{q}$, that is:

$$\min_{\mathbf{q}} J(\boldsymbol{\Omega}_t \mathbf{q}), \tag{9}$$

then we actually apply a change of coordinates to the original problem (1). Since $\boldsymbol{\Omega}_t$ is invertible, the problem of finding the $\boldsymbol{\theta}$ which minimizes $J(\boldsymbol{\theta})$ is equivalent to finding the $\mathbf{q}$ which minimizes $J(\boldsymbol{\Omega}_t \mathbf{q})$. Therefore, the advantage of solving (9) is that the solution is guaranteed to also be a solution of (1), which is not true for (3) with $\lambda > 0$. As noted in [39], the performance of gradient descent is dependent on the parameterization – a new choice may substantially alter convergence characteristics. For the case of a unique solution, introducing variable scalings may speed up convergence by altering the descent direction, while still converging to the same solution. In the case of multiple optima, an appropriate scaling matrix may push the optimizer toward sparser solutions. In other words, if there are multiple solutions to (1), then iteratively solving (9) will tend to produce sparse choices of $\boldsymbol{\theta}$. It is well known in the context of SSR that AST-based methods converge to sparse solutions [30, 17]. While we do not claim this argument is rigorous for DNNs, where multiple local minima are hard to characterize, the numerical results appear to support this observation.

To proceed, define the *a posteriori* AST variable at timestep t:

$$\mathbf{q}_{t|t} \triangleq \boldsymbol{\Omega}_t^{-1} \boldsymbol{\theta}_t \tag{10}$$

and the *a priori* AST variable at timestep t:

$$\mathbf{q}_{t+1|t} \triangleq \boldsymbol{\Omega}_t^{-1} \boldsymbol{\theta}_{t+1}. \tag{11}$$

We formulate a recursive update by using SGD in the $\mathbf{q}$ domain:

$$\mathbf{q}_{t+1|t} \leftarrow \mathbf{q}_{t|t} - \eta \nabla_{\mathbf{q}} J(\boldsymbol{\Omega}_t \mathbf{q}_{t|t}). \tag{12}$$

Using the chain rule, (8), and (10), we get:

$$\mathbf{q}_{t+1|t} \leftarrow \mathbf{q}_{t|t} - \eta \boldsymbol{\Omega}_t \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t). \tag{13}$$

Finally, premultiplying $\boldsymbol{\Omega}_t$ on both sides of (13) and applying (10) and (11), we transform the update rule back to the $\boldsymbol{\theta}$ domain:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \boldsymbol{\Omega}_t^2 \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t). \tag{14}$$

This is the Sparsity-promoting Stochastic Gradient Descent (SSGD).

In (14), the term $\mathbf{\Omega}_t^2$ provides a weighting factor $\omega_{i,t}^2$ to the learning rate η for updating the corresponding parameter $\theta_{i,t}$. This weighting $\omega_{i,t}^2$ is typically a function of $|\theta_{i,t}|$, the magnitude of the parameter. In this sense, SSGD is similar to the proportionate normalized least mean square (PNLMS) algorithm [40] well-known in sparse adaptive filtering. The main idea behind PNLMS is to update each filter coefficient using a learning rate proportional to the magnitude of the estimated coefficient to speed up convergence. In contrast, the purpose of SSGD is to promote sparse solutions.

3.3. Example diversity measures for promoting sparsity

To illustrate the flexibility of the proposed framework, we provide examples of SSGD algorithms instantiated with popular diversity measures that have proved effective in SSR.

Consider the p-norm-like diversity measure with $g(\theta_i) = |\theta_i|^p$, $0 < p \leq 2$ for the reweighted ℓ_2 framework [17, 30]. Using (6) leads to the update rule for $\mathbf{\Omega}_t$:

$$\omega_{i,t} = \left(\frac{2}{p} \left(|\theta_{i,t}| + c \right)^{2-p} \right)^{\frac{1}{2}}. \tag{15}$$

Note that we have added a small regularization constant $c > 0$ for stability purposes. The p-norm-like diversity measure can also be adopted in the reweighted ℓ_1 framework if $0 < p \leq 1$. In this case, we apply (7) to obtain the update rule for $\mathbf{\Omega}_t$:

$$\omega_{i,t} = \frac{1}{p} \left(|\theta_{i,t}| + c \right)^{1-p}. \tag{16}$$

Again, a small constant $c > 0$ is added. In general, using a smaller p for (15) and (16) promotes more sparsity.

We can consider the log-sum penalty with $g(\theta_i) = \log(\theta_i^2 + \epsilon)$, $\epsilon > 0$ for the reweighted ℓ_2 framework as well [19]. The function is readily amenable to the use of (6) to obtain the update rule for $\mathbf{\Omega}_t$ as:

$$\omega_{i,t} = \left(\theta_{i,t}^2 + \epsilon \right)^{\frac{1}{2}}. \tag{17}$$

Or consider the log-sum penalty with $g(\theta_i) = \log(|\theta_i| + \epsilon)$, $\epsilon > 0$ for the reweighted ℓ_1 framework [20]. Using (7), the update rule for $\mathbf{\Omega}_t$ becomes:

$$\omega_{i,t} = |\theta_{i,t}| + \epsilon. \tag{18}$$

For both (17) and (18), using a smaller ϵ induces stronger sparsity.

3.4. Practical implementation

In practice, we find that normalizing the $\mathbf{\Omega}_t^2$ term in (14) helps stabilize SSGD. Normalization is also performed in the PNLMS [40]. We thus propose the practical SSGD update rule:

$$\boxed{\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \mathbf{S}_t \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t),} \tag{19}$$

where $\mathbf{S}_t = \text{diag}\{s_{i,t}\}$, referred to as the *sparsity-promoting matrix*, is the normalized version of $\mathbf{\Omega}_t^2$:

$$\boxed{s_{i,t} = \frac{\omega_{i,t}^2}{\frac{1}{|\mathcal{I}^{(k)}|} \sum\limits_{j \in \mathcal{I}^{(k)}} \omega_{j,t}^2}, \quad \text{for } i \in \mathcal{I}^{(k)},} \tag{20}$$

where $\mathcal{I}^{(k)}$ denotes the index set of parameters of layer k, $\theta_{i,t}$ is in layer k, and $|\mathcal{I}^{(k)}|$ is the cardinality of $\mathcal{I}^{(k)}$. Algorithm 1 summarizes the SSGD algorithm which can be implemented using standard deep learning libraries without much effort.

Algorithm 1: The proposed SSGD algorithm for learning sparse DNN connections. $\boldsymbol{\omega}_t$ and $\mathbf{s}_t$ denote the vectors consisting of the diagonal elements of $\mathbf{\Omega}_t$ and $\mathbf{S}_t$, respectively. $\odot$ denotes element-wise multiplication.

Input: η: learning rate, d_t: training data at timestep t, and the choice of the diversity measure
Output: $\boldsymbol{\theta}_t$: estimated model parameters
Initialize: $\boldsymbol{\theta}_0$
for $t = 0, 1, 2...$ **do**
  Compute scaling factors: $\boldsymbol{\omega}_t$ according to the specified diversity measure (e.g., using (15), (16), (17), or (18))
  Compute sparsity-promoting factors: $\mathbf{s}_t$ by (20)
  Update parameters: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \cdot \mathbf{s}_t \odot \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t, d_t)$
end for

3.5. Application to DNN compression

Han et al. [8] have proposed a 3-stage compression scheme: i) learning important connections, ii) pruning unimportant parameters by hard thresholding, and iii) fine-tuning the remaining ones. We adopt the same scheme, using SSGD in stage i). In [8], it is observed that, ℓ_1 regularization leads to sparser networks after stage i), but the network loses significant accuracy after stage ii), and is not able to recover from this accuracy drop even after stage iii). The authors posit that the discrepancy between using ℓ_1 regularization during stage i) and not using it during stage iii) leads to poor performance. SSGD circumvents such issues because it finds (sparse) solutions to (1) directly, instead of switching between (3) and (1) like [8].

4. EVALUATION

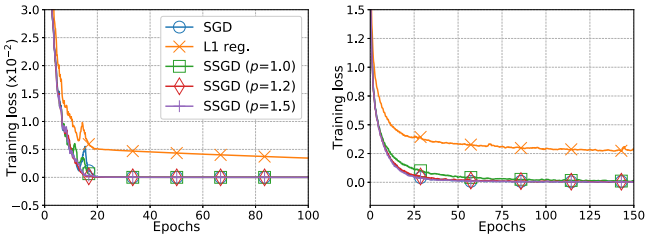
For evaluation, we use the PyTorch [41] library and consider two image classification tasks:

- *CNN-1 on MNIST database [35]*: We define a model (referred to as CNN-1) that has 2 CONV layers (# input channels $\times$ # output channels: $1 \times 32 - 32 \times 64$) using 5×5 kernels followed by 3 FC layers (# input neurons $\times$ # output neurons: $2304 \times 128 - 128 \times 64 - 64 \times 10$) for this task. Max pooling is performed after each CONV layer. A rectified linear unit (ReLU) activation is adopted for all layers and we use cross-entropy for $J(\boldsymbol{\theta})$.
- *CNN-2 on CIFAR-10 database [36]*: We define a more complicated model (referred to as CNN-2) with 6 CONV layers ($3 \times 64 - 64 \times 64 - 64 \times 128 - 128 \times 128 - 128 \times 256 - 256 \times 256$) using 3×3 kernels followed by 3 FC layers ($4096 \times 256 - 256 \times 128 - 128 \times 10$) for this task. Each of the CONV layer is followed by batch normalization [37] before activation. Max pooling is performed after the second, forth, and last CONV layers. Dropout [38] with a rate of 0.2 is applied to the first and second FC layers. ReLU activation is adopted for all layers and we use cross-entropy for $J(\boldsymbol{\theta})$.

We use (15) within SSGD, setting $c = 0.001$. We compare SSGD with SGD and SGD applied to an ℓ_1 regularized objective (denoted as 'L1 reg.'), i.e., using $\|\boldsymbol{\theta}\|_1$ for the diversity measure $G(\boldsymbol{\theta})$ in (3), with $\lambda = 10^{-6}$ and 10^{-5} for CNN-1 and CNN-2, respectively.

Fig. 1 shows the training loss vs. epochs for SGD, SGD for the ℓ_1 regularized objective, and SSGD with $p = 1.0$, 1.2, and 1.5. We train CNN-1 on MNIST for 100 epochs and CNN-2 on CIFAR-10 for 150 epochs. The same initialization is used among different algorithms for each model. A learning rate $\eta = 0.1$ and a batch size of

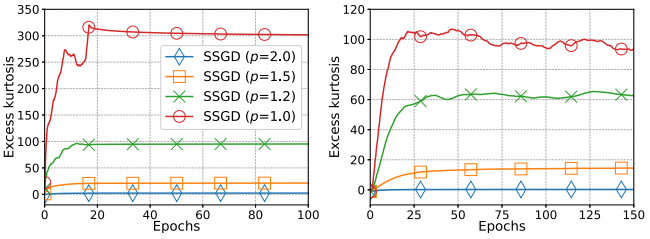
64 are used for all cases. For reference, CNN-1 and CNN-2 achieve 99.27% and 85.21% test accuracy with normal SGD training, respectively. Fig. 1 shows that SSGD is able to converge toward the same loss as SGD, supporting the argument that SSGD finds solutions to (1). The ℓ_1 regularized case, however, ends up at a higher loss due to the bias introduced by a nonzero λ.



(a) (b)

Fig. 1. Training loss vs. epochs for (a) CNN-1 on MNIST and (b) CNN-2 on CIFAR-10.

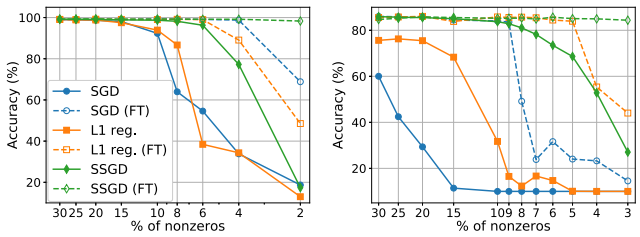
Fig. 2 monitors the excess kurtosis vs. epochs for SSGD with various p values. Distributions with excess kurtosis higher than 0 are called super-Gaussian, meaning that they have higher peaks at 0 and heavier tails compared to the Gaussian distribution, which has an excess kurtosis of 0. Excess kurtosis can thus serve as a measure of sparsity (the higher, the sparser). Fig. 2 shows that a smaller p leads to greater sparsity. Note that when using (15) with $p = 2$, SSGD reduces to normal SGD, resulting in near 0 excess kurtosis.



(a) (b)

Fig. 2. Excess kurtosis vs. epochs for (a) the first FC layer weights of CNN-1 and (b) the last CONV layer weights of CNN-2.

Fig. 3 shows the test accuracy vs. % of nonzeros after pruning for different cases. After running SSGD, we use the magnitude-based strategy from [8] to fix small weights to 0. As can be seen in Fig. 3, after pruning (solid lines), accuracy drops with decreasing % of nonzeros (more aggressive pruning). SSGD (using $p = 1$) retains the highest accuracy after pruning in both Fig. 3 (a) and Fig. 3 (b). The ℓ_1 regularized case also maintains higher accuracy than the SGD in Fig. 3 (b). As both cases are sparsity-aware training, this supports the argument that sparsity is important for learning compact connectivity of models [11, 15]. Now, to regain accuracy, fine-tuning is necessary. Compared to the iterative process suggested in [8], one-shot pruning and retraining is more desirable [11]. In addition, the retraining period should also be kept short. Therefore, we fine-tune the pruned models once (CNN-1 for 35 epochs and CNN-2 for 50 epochs only) by optimizing (1) using the Adam optimizer [42] with a learning rate of 0.001. From Fig. 3, we see that after fine-tuning (dashed lines, labeled with '(FT)'), accuracy can be regained to a certain degree for all cases. Note that the case of SGD using an ℓ_1 regularized objective is not necessarily better than normal SGD after retraining, e.g., in Fig. 3 (a). The proposed SSGD, on the other hand, achieves the highest accuracy after fine-tuning. This demonstrates the power of SSGD to learn better network connectivity in the training phase and the benefit of using the AST to avoid possible issues due to change of optimization modes in the fine-tuning stage.



(a) (b)

Fig. 3. Test accuracy vs. % of nonzeros for (a) CNN-1 on MNIST and (b) CNN-2 on CIFAR-10. 'FT' stands for 'fine-tuned.'

Table 1 compares the sparsification performance of the proposed SSGD-based approach to some recent pruning methods. We compare with [28], which also utilizes the iterative reweighting concept in their pruning framework. However, their method prunes a pre-trained network via log-sum minimization in a layer-by-layer fashion. Our approach, on the other hand, sparsifies all layers simultaneously during training. Moreover, we have a broader framework that covers the log-sum penalty as a special case. For comparison purposes, we adopt the same network architectures as in their paper, namely, a multi-layer perceptron on MNIST (referred to as MLP) which consists of 4 FC layers, and a CNN on CIRAF-10 (referred to as CNN-3) which consists of 2 CONV layers (each with batch normalization added before activation) followed by 3 FC layers. We also compare with Net-Trim [43], another pruning method also compared with in [28]. For the proposed method, we train the models with SSGD using $p = 1$. Then, we prune the models once and fine-tune using Adam. From the results, we can see that the proposed method achieves the highest sparsity with comparable, if not better, accuracy compared to existing methods.

Table 1. Comparison of sparsification results.

Model	Method	Accuracy	% of nonzeros
MLP	Original	98.62%	100.0
	Net-Trim [43]	97.70%	30.5
	Iter. Reweight. [28]	97.46%	14.8
	Proposed	98.39%	3.7
CNN-3	Original	77.44%	100.0
	Net-Trim [43]	75.92%	17.8
	Iter. Reweight. [28]	74.17%	7.9
	Proposed	74.54%	5.1

5. CONCLUSION

In this paper, we proposed an iterative reweighting framework for learning sparse connections within DNNs. The framework utilizes the AST to derive the SSGD algorithm that adopts a zero regularization coefficient λ, leading to an unbiased pruning approach. The sparsification ability of SSGD has been demonstrated on image classification tasks and shown to outperform existing methods on the MNIST and CIFAR-10 datasets.

6. REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Adv. Neural Inform. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[3] D. Wang and J. Chen, "Supervised speech separation based on deep learning: An overview," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 26, no. 10, pp. 1702–1726, 2018.

[4] J. M. Alvarez and M. Salzmann, "Learning the number of neurons in deep networks," in *Adv. Neural Inform. Process. Syst. (NIPS)*, 2016, pp. 2270–2278.

[5] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Adv. Neural Inform. Process. Syst. (NIPS)*, 2013, pp. 2148–2156.

[6] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, "Group sparse regularization for deep neural networks," *Neurocomputing*, vol. 241, pp. 81–89, 2017.

[7] L. Pisha, J. Warchall, T. Zubatiy, S. Hamilton, C.-H. Lee, G. Chockalingam, P. P. Mercier, R. Gupta, B. D. Rao, and H. Garudadri, "A wearable, extensible, open-source platform for hearing healthcare research," *IEEE Access*, vol. 7, pp. 162083–162101, 2019.

[8] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Adv. Neural Inform. Process. Syst. (NIPS)*, 2015, pp. 1135–1143.

[9] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Adv. Neural Inform. Process. Syst. (NIPS)*, 2016, pp. 1379–1387.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Int. Conf. Learn. Repres. (ICLR)*, 2016.

[11] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," in *Int. Conf. Learn. Repres. (ICLR)*, 2017.

[12] L. Mauch and B. Yang, "A novel layerwise pruning method for model reduction of fully connected deep neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2017, pp. 2382–2386.

[13] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[14] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, 2018.

[15] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vision (ECCV)*, 2018, pp. 784–800.

[16] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statist. Soc. Series B (Methodological)*, pp. 267–288, 1996.

[17] I. F. Gorodnitsky and B. D. Rao, "Sparse signal reconstruction from limited data using FOCUSS: A re-weighted minimum norm algorithm," *IEEE Trans. Signal Process.*, vol. 45, no. 3, pp. 600–616, 1997.

[18] M. E. Tipping, "Sparse Bayesian learning and the relevance vector machine," *J. Mach. Learn. Res.*, vol. 1, pp. 211–244, 2001.

[19] R. Chartrand and W. Yin, "Iteratively reweighted algorithms for compressive sensing," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 2008, pp. 3869–3872.

[20] E. J. Candès, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted ℓ_1 minimization," *J. Fourier Anal. Appl.*, vol. 14, no. 5, pp. 877–905, 2008.

[21] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proc. IEEE Conf. Compt. Vision Pattern Recog. (CVPR)*, 2015, pp. 806–814.

[22] V. Lebedev and V. Lempitsky, "Fast ConvNets using group-wise brain damage," in *Proc. IEEE Conf. Comput. Vision Pattern Recog. (CVPR)*, 2016, pp. 2554–2564.

[23] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact CNNs," in *Proc. Eur. Conf. Comput. Vision (ECCV)*, 2016, pp. 662–677.

[24] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Adv. Neural Inform. Process. Syst. (NIPS)*, 2016, pp. 2074–2082.

[25] C. Louizos, K. Ullrich, and M. Welling, "Bayesian compression for deep learning," in *Adv. Neural Inform. Process. Syst. (NIPS)*, 2017, pp. 3288–3298.

[26] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," in *Int. Conf. Learn. Repres. (ICLR)*, 2018.

[27] S. Oymak, "Learning compact neural networks with regularization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 3966–3975.

[28] T. Jiang, X. Yang, Y. Shi, and H. Wang, "Layer-wise deep neural network pruning via iteratively reweighted optimization," in *Proc. IEEE Int. Conf. Acoust., Speech, and Signal Process. (ICASSP)*, 2019, pp. 5606–5610.

[29] D. Wipf and S. Nagarajan, "Iterative reweighted ℓ_1 and ℓ_2 methods for finding sparse solutions," *IEEE J. Sel. Top. Signal Process.*, vol. 4, no. 2, pp. 317–329, 2010.

[30] B. D. Rao and K. Kreutz-Delgado, "An affine scaling methodology for best basis selection," *IEEE Trans. Signal Process.*, vol. 47, no. 1, pp. 187–200, 1999.

[31] S. G. Nash and A. Sofer, *Linear and Nonlinear Programming*, McGraw-Hill Inc., 1996.

[32] C.-H. Lee, B. D. Rao, and H. Garudadri, "Proportionate adaptive filters based on minimizing diversity measures for promoting sparsity," in *Proc. Asilomar Conf. Signals Syst. Comput. (ACSSC)*, 2019.

[33] C.-H. Lee, B. D. Rao, and H. Garudadri, "Sparsity promoting LMS for adaptive feedback cancellation," in *Proc. Eur. Signal Process. Conf. (EUSIPCO)*, 2017, pp. 226–230.

[34] I. Fedorov and B. D. Rao, "Re-weighted learning for sparsifying deep neural networks," *arXiv preprint arXiv:1802.01616*, 2018.

[35] Y. LeCun, C. Cortes, and E. J. Burges, *The MNIST Database of Handwritten Digits*, 1998.

[36] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep., Univ. Toronto, 2009.

[37] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 448–456.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[39] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, 4th edition, Springer, 2016.

[40] D. L. Duttweiler, "Proportionate normalized least-mean-squares adaptation in echo cancelers," *IEEE Trans. Speech Audio Process.*, vol. 8, no. 5, pp. 508–518, 2000.

[41] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *Neural Inform. Process. Syst. Workshop (NIPS-W)*, 2017.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Int. Conf. Learn. Repres. (ICLR)*, 2014.

[43] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, "Net-Trim: Convex pruning of deep neural networks with performance guarantee," in *Adv. Neural Inform. Process. Syst. (NIPS)*, 2017, pp. 3177–3186.