Structured Sparsity of Convolutional Neural Networks via Nonconvex Sparse Group Regularization

Kevin Bui[1], **Fredrick Park**[2], **Shuai Zhang**[1], **Yingyong Qi**[1], **and Jack Xin**[1,*]

[1] *Department of Mathematics; University of California, Irvine; Irvine, CA 92697, United States*
[2] *Department of Mathematics & Computer Science; Whittier College; Whittier, CA 90602, United States*

Correspondence*:
Corresponding Author
jack.xin@uci.edu

ABSTRACT

Convolutional neural networks (CNN) have been hugely successful recently with superior accuracy and performance in various imaging applications, such as classification, object detection, and segmentation. However, a highly accurate CNN model requires millions of parameters to be trained and utilized. Even to increase its performance slightly would require significantly more parameters due to adding more layers and/or increasing the number of filters per layer. Apparently, many of these weight parameters turn out to be redundant and extraneous, so the original, dense model can be replaced by its compressed version attained by imposing inter- and intra-group sparsity onto the layer weights during training. In this paper, we propose a nonconvex family of sparse group lasso that blends nonconvex regularization (e.g., transformed ℓ_1, $\ell_1 - \ell_2$, and ℓ_0) that induces sparsity onto the individual weights and $\ell_{2,1}$ regularization onto the output channels of a layer. We apply variable splitting onto the proposed regularization to develop an algorithm that consists of two steps per iteration: gradient descent and thresholding. Numerical experiments are demonstrated on various CNN architectures showcasing the effectiveness of the nonconvex family of sparse group lasso in network sparsification and test accuracy on par with the current state of the art.

Keywords: deep learning, sparsity, nonconvex optimization, sparse group lasso, feature selection

1 INTRODUCTION

Deep neural networks (DNNs) have proven to be advantageous for numerous modern computer vision tasks involving image or video data. In particular, convolutional neural networks (CNNs) yield highly accurate models with applications in image classification [39, 77, 28, 95], semantic segmentation [49, 13], and object detection [73, 30, 72]. These large models often contain millions of weight parameters that often exceed the number of training data. This is a double-edged sword since on one hand, large models allow for high accuracy, while on the other, they contain many redundant parameters that lead to overparametrization.

Overparametrization is a well-known phenomenon in DNN models [17, 6] that results in overfitting, learning useless random patterns in data [96], and having inferior generalization. Additionally, these models also possess exorbitant computational and memory demands during both training and inference. Consequently, they may not be applicable for devices with low computational power and memory.

Resolving these problems requires compressing the networks through sparsification and pruning. Although removing weights might affect the accuracy and generalization of the models, previous works [54, 25, 81, 66] demonstrated that many networks can be substantially pruned with negligible effect on accuracy. There are many systematic approaches to achieving sparsity in DNNs, as discussed extensively in [14, 15].

Han *et al.* [26] proposed to first train a dense network, prune it afterward by setting the weights to zeroes if below a fixed threshold, and retrain the network with the remaining weights. Jin *et al.* [32] extended this method by restoring the pruned weights, training the network again, and repeating the process. Rather than pruning by thresholding, Aghasi *et al.* [1, 2] proposed Net-Trim, which prunes an already trained network layer by layer using convex optimization in order to ensure that the layer inputs and outputs remain consistent with the original network. For CNNs in particular, filter or channel pruning is preferred because it significantly reduces the amount of weight parameters required compared to individual weight pruning. Li *et al.* [43] calculated the sums of absolute weights of the filters of each layer and pruned the ones with the smallest sums. Hu *et al.* [29] proposed a metric called average percentage of zeroes for channels to measure their redundancies and pruned those with highest values for each layer. Zhuang *et al.* [105] developed discrimination-aware channel pruning that selects channels that contribute to the network's discriminative power.

An alternative approach to pruning a dense network is learning a compressed structure from scratch. A conventional approach is to optimize the loss function equipped with either the ℓ_1 or ℓ_2 regularization, which drives the weights to zeroes or to very small values during training. To learn which groups of weights (e.g., neurons, filters, channels) are necessary, group regularization, such as group lasso [93] and sparse group lasso [76], are equipped to the loss function. Alvarez and Salzmann [4] and Scardapane *et al.* [75] applied group lasso and sparse group lasso to various architectures and obtained compressed networks with comparable or even better accuracy. Instead of sharing features among the weights as suggested by group sparsity, exclusive sparsiy [104] promotes competition for features between different weights. This method was investigated by Yoon and Hwang [92]. In addition, they combined it with group sparsity and demonstrated that this combination resulted in compressed networks with better performance than their original counterparts. Non-convex regularization has also been examined. Louizos *et al.* [54] proposed a practical algorithm using probabilistic methods to perform ℓ_0 regularization on CNNs. Ma *et al.* [61] proposed integrated transformed ℓ_1, a convex combination of transformed ℓ_1 and group lasso, and compared its performance against the aforementioned group regularization methods.

In this paper, we propose a family of group regularization methods that balances both group lasso for group-wise sparsity and nonconvex regularization for element-wise sparsity. The family extends sparse group lasso by replacing the ℓ_1 penalty term with a nonconvex penalty term. The nonconvex penalty terms considered are ℓ_0, $\ell_1 - \alpha\ell_2$, transformed ℓ_1, and SCAD. The proposed family is supposed to yield a more accurate and/or more compressed network than sparse group lasso since ℓ_1 suffers various weaknesses due to being a convex relaxation of ℓ_0. We develop an algorithm to optimize loss functions equipped with the proposed nonconvex, group regularization terms for DNNs.

2 MODEL AND ALGORITHM

2.1 Preliminaries

Given a training dataset consisting of N input-output pairs $\{(x_i, y_i)\}_{i=1}^N$, the weight parameters of a DNN are learned by optimizing the following objective function:

$$\min_W \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(h(x_i, W), y_i) + \lambda \mathcal{R}(W), \tag{1}$$

where

- W is the set of weight parameters of the DNN.
- $h(\cdot, \cdot)$ is the output of the DNN used for prediction.
- $\mathcal{L}(\cdot, \cdot) \geq 0$ is the loss function that compares the prediction $h(x_i, W)$ with the ground-truth output y_i. Examples include cross-entropy loss function for classification and mean-squared error for regression.
- $\mathcal{R}(\cdot)$ is the regularizer on the set of weight parameters W.
- $\lambda > 0$ is a regularization parameter for $\mathcal{R}(\cdot)$.

The most common regularizer used for DNNs is ℓ_2 regularization $\|\cdot\|_2^2$, also known as weight decay. It prevents overfitting and improves generalization because it enforces the weights to decrease proportionally to their magnitudes [40]. Sparsity can be imposed by pruning weights whose magnitudes are below a certain threshold at each iteration during training. However, an alternative regularizer is the ℓ_1 norm $\|\cdot\|_1$, also known as the lasso penalty [78]. The ℓ_1 norm is the tightest convex relaxation of the ℓ_0 penalty [20, 23, 82] and it yields a sparse solution that is found on the corners of the 1-norm ball [27, 52]. Theoretical results justify the ℓ_1 norm's ability to reconstruct sparse solution in compressed sensing. When a sensing matrix satisfies the restricted isometry property, the ℓ_1 norm recovers the sparse solution exactly with high probability [11, 23, 82]. On the other hand, the null space property is a necessary and sufficient condition for ℓ_1 minimization to guarantee exact recovery of sparse solutions [16, 23]. Being able to yield sparse solutions, the ℓ_1 norm has gained popularity in other types of inverse problems such as compressed imaging [33, 57] and image segmentation [35, 34, 42] and in various fields of applications such as geoscience [74], medical imaging [33, 57], machine learning [10, 78, 36, 67, 89], and traffic flow network [91]. Unfortunately, element-wise sparsity by ℓ_1 or ℓ_2 regularization in CNNs may not yield meaningful speedup as the number of filters and channels required for computation and inference may remain the same [86].

To determine which filters or channels are relevant in each layer, group sparsity using the group lasso penalty [93] is considered. The group lasso penalty has been utilized in various applications, such as microarray data analysis [62], machine learning [7, 65], and EEG data [46]. Suppose a DNN has L layers, so the set of weight parameters W is divided into L sets of weights: $W = \{W_l\}_{l=1}^L$. The weight set of each layer W_l is divided into N_l groups (e.g., channels or filters): $W_l = \{w_{l,g}\}_{g=1}^{N_l}$. The group lasso penalty applied to W_l is formulated as

$$\mathcal{R}_{GL}(W_l) = \sum_{g=1}^{N_l} \sqrt{\# w_{l,g}} \|w_{l,g}\|_2 = \sum_{g=1}^{N_l} \sqrt{\# w_{l,g}} \sqrt{\sum_{i=1}^{\# w_{l,g}} w_{l,g,i}^2}, \tag{2}$$

where $w_{l,g,i}$ corresponds to the weight parameter with index i in group g in layer l and the term $\#w_{l,g}$ denotes the number of weight parameters in group g in layer l. Because group sizes vary, the constant $\sqrt{\#w_{l,g}}$ is multiplied in order to rescale the ℓ_2 norm of each group with respect to the group size, ensuring that each group is weighed uniformly [93, 76, 65]. The group lasso regularizer imposes the ℓ_2 norm on each group, forcing weights of the same groups to decrease altogether at every iteration during training. As a result, the groups of weights are pruned when their ℓ_2 norms are negligible, resulting in a highly compact network compared to element-sparse networks.

As an alternative to group lasso that encourages feature sharing, exclusive sparsity [104] enforces the model weight parameters to compete for features, making the features discriminative for each class in the context of classification. The regularization for exclusive sparsity is

$$\frac{1}{2}\sum_{g=1}^{N_l} \|w_{l,g}\|_1^2 = \frac{1}{2}\sum_{g=1}^{N_l}\left(\sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|\right)^2. \tag{3}$$

Now, within each group, sparsity is enforced. Because exclusivity cannot guarantee the optimal features since some features do need to be shared, exclusive sparsity can be combined with group sparsity to form combined group and exclusive sparsity (CGES) [92]. CGES is formulated as

$$\mathcal{R}_{CGES} = \sum_{g=1}^{N_l}\left[(1-\mu_l)\sqrt{\sum_{i=1}^{\#w_{l,g}} w_{l,g,i}^2} + \frac{\mu_l}{2}\left(\sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|\right)^2\right], \tag{4}$$

where $\mu_l \in (0,1)$ is a parameter for balancing exclusivity and sharing among features.

To obtain an even sparser network, element-wise sparsity and group sparsity can be combined and applied together to the training of DNNs. One regularizer that combines these two types of sparsity is the sparse group lasso penalty [76], which is formulated as

$$\mathcal{R}_{SGL_1}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_1 \tag{5}$$

where

$$\|W_l\|_1 = \sum_{g=1}^{N_l}\sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|.$$

Sparse group lasso simultaneously enforces group sparsity by having the regularizer $\mathcal{R}_{GL}(\cdot)$ and element-wise sparsity by having the ℓ_1 norm. This regularizer has been used in machine learning [83], bioinformatics [48, 103], and medical imaging [47].

Figure 1 demonstrates the differences between lasso, group lasso, and sparse group lasso applied to a weight matrix connecting a 5-dimensional input layer to a 10-dimensional output layer. In white, the entries are zero'ed out; in gray; the entries are not. Unlike lasso, group lasso results in a more structured method of pruning since three of the five neurons can be zero'ed out. Combined with ℓ_1 regularization on the individual weights, sparse group lasso allows for more weights in the remaining two neurons to be pruned.

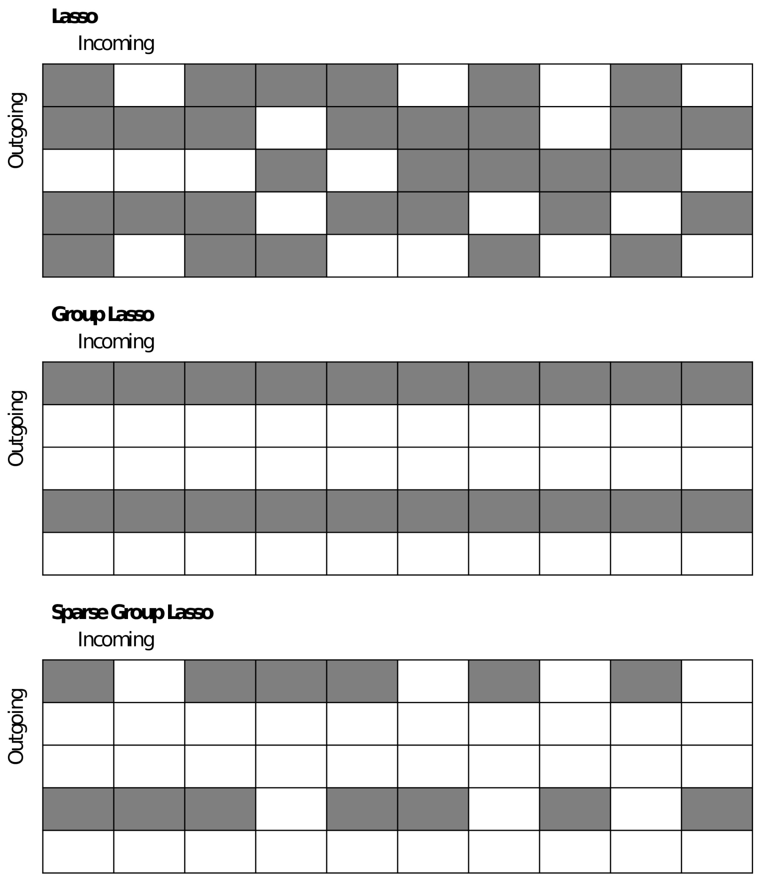

Figure 1. Comparison between lasso, group lasso, and sparse group lasso applied to a weight matrix. Entries in white are zero'ed out or removed; entries in gray remain.

107 2.2 Nonconvex Sparse Group Lasso

We recall that the ℓ_1 norm is the tightest convex relaxation of the ℓ_0 penalty, given by

$$\|W_l\|_0 = \sum_{g=1}^{N_l} \sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|_0 \tag{6}$$

where

$$|w|_0 = \begin{cases} 1 \text{ if } w \neq 0 \\ 0 \text{ if } w = 0 \end{cases}$$

when applied to the weight set W_l of layer l. The ℓ_0 penalty is non-convex and discontinuous. In addition, any ℓ_0-regularized problem is NP-hard [23]. These properties make developing convergent and tractable algorithms for ℓ_0-regularized problems difficult, thereby making ℓ_1-regularized problems better alternatives to solve. However, the ℓ_0-regularized problems have been shown to recover better solutions in terms of sparsity and/or accuracy than do ℓ_1-regularized problems in various applications, such as compressed sensing [56], image restoration [8, 12, 19, 102, 55], MRI reconstruction [80], and machine learning [56, 94]. In particular, ℓ_0-regularized inverse problems were demonstrated to be more robust against Poisson noise than are ℓ_1-regualarized inverse problems [100].

A continuous alternative to the ℓ_0 penalty is the SCAD penalty term [22, 58], given by

$$\lambda \|W_l\|_{\text{SCAD}(a)} = \sum_{g=1}^{N_l} \sum_{i=1}^{\#w_{l,g}} \lambda |w_{l,g,i}|_{\text{SCAD}(a)} \tag{7}$$

where

$$\lambda |w|_{\text{SCAD}(a)} := \begin{cases} \lambda|w| & \text{if } |w| < \lambda \\ \frac{2a\lambda|w| - w^2 - \lambda^2}{2(a-1)} & \text{if } \lambda \leq |w| < a\lambda \\ (a+1)\lambda^2/2 & \text{if } |w| \geq a\lambda \end{cases}$$

for $\lambda > 0$ and $a > 2$. This penalty term enjoys three properties – unbiasedness, sparsity, and continuity – while the ℓ_1 norm, on the other hand, has only sparsity and continuity [22]. In linear and logistic regression, SCAD was shown to outperform ℓ_1 in variable selection [22]. SCAD has been applied to wavelet approximation [5], bioinformatics [9, 84], and compressed sensing [64].

The transformed ℓ_1 penalty term [68] also enjoys the properties of unbiasedness, sparsity, and continuity [58]. In fact, the regularizer is not just continuous but Lipschitz continuous [98]. The term is given by

$$\|W_l\|_{\text{TL1}(a)} = \sum_{g=1}^{N_l} \sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|_{\text{TL1}(a)} \tag{8}$$

where

$$|w|_{\text{TL1}(a)} = \frac{(a+1)|w|}{a+|w|}.$$

In addition, it interpolates the ℓ_0 and ℓ_1 penalties through the parameter a [98] because

$$\lim_{a \to 0^+} |w|_{\text{TL1}(a)} = |w|_0 \quad \text{and} \quad \lim_{a \to \infty} |w|_{\text{TL1}(a)} = |w|.$$

The transformed ℓ_1 penalty term was investigated and was shown to outperform ℓ_1 in compressed sensing [97, 98, 79], deep learning [61, 87, 45], matrix completion [99], and epidemic forecasting [45].

Another Lipschitz continous, nonconvex regularizer is the $\ell_1 - \alpha\ell_2$ penalty given by

$$\|W_l\|_{\ell_1-\alpha\ell_2} = \|W_l\|_1 - \alpha\|W_l\|_2 = \sum_{g=1}^{N_l}\sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}| - \alpha\sqrt{\sum_{g=1}^{N_l}\sum_{i=1}^{\#w_{l,g}} |w_{l,g,i}|^2}, \tag{9}$$

where $\alpha \in (0,1]$. In a series of works [52, 90, 50, 51], the penalty term $\ell_1 - \ell_2$ with $\alpha = 1$ yields better solutions than does ℓ_1 in various compressed sensing applications especially when the sensing matrix is highly coherent or it violates the restricted isometry property condition. To guarantee exact recovery of sparse solution, $\ell_1 - \ell_2$ only requires a relaxed variant of the null space property [79]. Furthermore, $\ell_1 - \alpha\ell_2$ is more robust against impulsive noise in yielding sparse, accurate solutions for inverse problems than is ℓ_1 [44]. Besides compressed sensing, it has been utilized in image denoising and deblurring [53], image segmentation [71], image inpainting [63], and hyperspectral demixing [21]. In deep learning application, the $\ell_1 - \ell_2$ regularization was used to learn permutation matrices [59] for ShuffleNet [101, 60].

Due to the advantages and recent successes of the aforementioned nonconvex regularizers, we propose to replace the ℓ_1 norm in (5) with nonconvex penalty terms. Hence, we propose a family of group regularizers called nonconvex sparse group lasso. The family includes the following:

$$\mathcal{R}_{SGL_0}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_0 \tag{10}$$

$$\mathcal{R}_{SGSCAD(a)}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_{\text{SCAD}(a)} \tag{11}$$

$$\mathcal{R}_{SGTL_1(a)}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_{\text{TL1}(a)} \tag{12}$$

$$\mathcal{R}_{SGL_1-\alpha L_2}(W_l) = \mathcal{R}_{GL}(W_l) + \|W_l\|_{\ell_1-\alpha\ell_2}. \tag{13}$$

Using these regularizers, we expect to obtain a sparser and/or more accurate network than from using the original sparse group lasso. The ℓ_1 norm can also be replaced with other nonconvex penalties not mentioned in this paper. Refer to [3, 85] to see other nonconvex penalties. However, we focus on the aforementioned nonconvex regularizers because they have closed-form proximal operators required by our proposed algorithm described in the next section.

2.3 Notations and Definitions

Before discussing the algorithm, we summarize notations that we will use to save space. They are the following:

- If $V = \{V_l\}_{l=1}^L$ and $W = \{W_l\}_{l=1}^L$, then $(V, W) := (\{V_l\}_{l=1}^L, \{W_l\}_{l=1}^L) = (V_1, \dots, V_L, W_1, \dots, W_L)$.
- $V^+ := V^{k+1}$.
- $\tilde{\mathcal{L}}(W) := \frac{1}{N}\sum_{i=1}^N \mathcal{L}(h(x_i, W), y_i)$.

In addition, we define the proximal operator for the regularization function $r(\cdot)$ as follows:

$$\text{prox}_{\lambda r}(y) = \arg\min_x \lambda\, r(x) + \frac{1}{2}\|x - y\|_2^2$$

for $\lambda > 0$.

2.4 Numerical Optimization

We develop a general algorithm framework to solve

$$\min_{W} \tilde{\mathcal{L}}(W) + \lambda \sum_{l=1}^{L} \mathcal{R}(W_l) = \tilde{\mathcal{L}}(W) + \sum_{l=1}^{L} \left(\lambda \mathcal{R}_{GL}(W_l) + \lambda r(W_l)\right) \tag{14}$$

where $W = \{W_l\}_{l=1}^{L}$, $\mathcal{R}$ is either $\mathcal{R}_{SGL_1}$ or one of the nonconvex regularizers (10)-(13), and $r(\cdot)$ is the corresponding sparsity-inducing regularizer. Throughout the paper, our assumption on (14) is the following:

ASSUMPTION 1. *The function $\tilde{\mathcal{L}}$ is continuously differentiable with respect to W_l for each $l = 1, \dots, L$.*

By introducing an auxiliary variable $V = \{V_l\}_{l=1}^{L}$ for (14), we have a constrained optimization problem:

$$\begin{aligned} \min_{V,W} \quad & \tilde{\mathcal{L}}(W) + \sum_{l=1}^{L} \left(\lambda \mathcal{R}_{GL}(W_l) + \lambda r(V_l)\right) \\ \text{s.t.} \quad & V_l = W_l \qquad l = 1, \dots, L. \end{aligned} \tag{15}$$

The constraints can be relaxed by adding the quadratic penalty terms with $\beta > 0$ so that we have

$$\min_{V,W} F_{\beta}(V, W) := \tilde{\mathcal{L}}(W) + \sum_{l=1}^{L} \left[\lambda \mathcal{R}_{GL}(W_l) + \lambda r(V_l) + \frac{\beta}{2}\|V_l - W_l\|_2^2\right]. \tag{16}$$

With β fixed, (16) can be solved by alternating minimization:

$$W^{k+1} = \arg\min_{W} F_{\beta}(V^k, W) \tag{17a}$$

$$V^{k+1} = \arg\min_{V} F_{\beta}(V, W^{k+1}). \tag{17b}$$

To solve (17a), we simultaneously update W_l for $l = 1, \dots L$ by gradient descent

$$W_l^{k+1} = W_l^k - \gamma\Big(\nabla_{W_l} \tilde{\mathcal{L}}(W^k) + \lambda \partial_{W_l} \mathcal{R}_{GL}(W_l^k) - \beta(V_l^k - W_l^k)\Big) \tag{18}$$

where $\gamma > 0$ is the learning rate and $\partial_{W_l} \mathcal{R}_{GL}$ is the subdifferential of $\mathcal{R}_{GL}$ with respect to W_l. In practice, (18) is performed using stochastic gradient descent (or one of its variants) with mini-batches due to the large-size computation dealing with the amount of data and weight parameters that a typical DNN has.

To update V, we see that (17b) can be rewritten as

$$V^{k+1} = \arg\min_{V} \sum_{l=1}^{L} \left(\frac{\lambda}{\beta} r(V_l) + \frac{1}{2}\|V_l - W_l\|_2^2\right) = \left(\text{prox}_{\frac{\lambda}{\beta} r}(W_1), \dots, \text{prox}_{\frac{\lambda}{\beta} r}(W_L)\right). \tag{19}$$

The proximal operators for the considered regularizers are thresholding functions as their closed-form solutions, and as a result, the V update simplifies to thresholding W. The regularization functions and their corresponding proximal operators are summarized in Table 1.

Algorithm 1: Algorithm for Nonconvex Sparse Group Lasso Regularization

```
1  Initialize $V^1$ and $W^1$ with random entries; learning rate $\gamma$; regularization parameters $\lambda$ and $\beta$; and
   multiplier $\sigma > 1$.
2  Set $j := 1$.
3  while stopping criterion for outer loop not satisfied do
4      Set $k := 1$.
5      Set $W^{j,1} = W^j$ and $V^{j,1} = V^j$.
6      while stopping criterion for inner loop not satisfied do
7          Update $W^{j,k+1}$ by Eq. (18).
8          Update $V^{j,k+1}$ by Eq. (19).
9          $k := k + 1$
10     end
11     Set $W^{j+1} = W^{j,k}$ and $V^{j+1} = V^{j,k}$.
12     Set $\beta := \sigma\beta$.
13     Set $j := j + 1$.
14 end
15 Output: $W^j$ and $V^j$.
```

Incorporating the algorithm that solves the quadratic penalty problem (16), we now develop a general algorithm to solve (14). We solve a sequence of quadratic penalty problems (16) with $\beta \in \{\beta_j\}_{j=1}^{\infty}$ where $\beta_j \uparrow \infty$. This will yield a sequence $\{(V^j, W^j)\}_{j=1}^{\infty}$ so that $W^j \uparrow W^*$, a solution to (14). This algorithm is based on the quadratic penalty method [69] and the penalty decomposition method [56]. The algorithm is summarized in Algorithm 1.

An alternative algorithm to solve (14) is proximal gradient descent [70]. By this method, the update for $W_l, l = 1, \dots, L$, is

$$W_l^{k+1} = \text{prox}_{\gamma\lambda r}\left(W_l^k - \gamma\left(\nabla_{W_l}\tilde{\mathcal{L}}(W^k) + \lambda\partial_{W_l}\mathcal{R}_{GL}(W_l^k)\right)\right). \tag{20}$$

Using this algorithm results in weight parameters with some already zero'ed out.

However, the advantage of our proposed algorithm lies in (17a), written more specifically as

$$\begin{aligned} W_l^{k+1} &= \arg\min_{W_l} \tilde{\mathcal{L}}(W) + \mathcal{R}_{GL}(W_l) + \frac{\beta}{2}\|V_l - W_l\|_2^2 \\ &= \arg\min_{W_l} \tilde{\mathcal{L}}(W) + \mathcal{R}_{GL}(W_l) + \frac{\beta}{2}\sum_{i=1}^{\#W_l}(v_{l,i} - w_{l,i})^2. \end{aligned} \tag{21}$$

We see that this step performs exact weight decay or ℓ_2 regularization on weights $w_{l,i}$ whenever $v_{l,i} = 0$. On the other hand, when $v_{l,i} \neq 0$, the effect of ℓ_2 regularization is mitigated on the corresponding weight $w_{l,i}$ based on the absolute difference $|v_{l,i} - w_{l,i}|$. Using ℓ_2 regularization was shown to give superior pruning results in terms of accuracy by Han et al. [26]. Our proposed algorithm can be perceived as an adaptive ℓ_2 regularization method, where (17b) identifies which weights to perform exact ℓ_2 regularization on and (17a) updates and regularizes the weights accordingly.

2.5 Convergence Analysis

To establish convergence for the proposed algorithm, the results below state that the accumulation point of the sequence generated by (17a)-(17b) is a block-coordinate minimizer, and an accumulation point generated by Algorithm 1 is a sparse feasible solution to (15). Proofs are provided in Section 5. Unfortunately, the feasible solution generated may not be a local minimizer of (15) because the loss function $\mathcal{L}(\cdot, \cdot)$ is nonconvex. However, it was shown in [18] that a similar algorithm to Algorithm 1, but for fixed β in a bounded interval, generates an approximate global solution with high probability for a one-layer CNN with ReLu activation function.

THEOREM 2. *Let $\{(V^k, W^k)\}_{k=1}^{\infty}$ be a sequence generated by the alternating minimization algorithm* (17a)-(17b)*, where $r(\cdot)$ is ℓ_0, ℓ_1, transformed ℓ_1, $\ell_1 - \alpha\ell_2$, or SCAD. If (V^*, W^*) is an accumulation point of $\{(V^k, W^k)\}_{k=1}^{\infty}$, then (V^*, W^*) is a block-coordinate minimizer of* (16)*. that is*

$$V^* \in \arg\min_{V} F_\beta(V, W^*)$$

$$W^* \in \arg\min_{W} F_\beta(V^*, W).$$

THEOREM 3. *Let $\{(V^k, W^k, \beta_k)\}_{k=1}^{\infty}$ be a sequence generated by Algorithm 1. Suppose that $\{F_{\beta_k}(V^k, W^k)\}_{k=1}^{\infty}$ is uniformly bounded. If (V^*, W^*) is an accumulation point of $\{V^k, W^k)\}_{k=1}^{\infty}$, then (V^*, W^*) is a feasible solution to* (15)*, that is $V^* = W^*$.*

Remark: To safely ensure that $\{F_{\beta_k}(V^k, W^k)\}_{k=1}^{\infty}$ is uniformly bounded in practice, we can find a feasible solution $(V^{\text{feas}}, W^{\text{feas}})$ to (15) and impose a bound M such that

$$M \geq \max\left\{\tilde{L}(W^{\text{feas}}) + \lambda \sum_{l=1}^{L} \mathcal{R}(W_l^{\text{feas}}), \min_{W} F_{\beta_0}(V^1, W)\right\}.$$

If $\min_W F_{\beta_{k+1}}(V^k, W) > M$, then we set $V^{k+1} = W^{\text{feas}}$. This strategy is based on [56]. However, in our numerical experiments, we have not yet encountered $F_{\beta_k}(V^k, W^k)$ to diverge.

3 NUMERICAL EXPERIMENTS

3.1 Application to Deep Neural Networks

We compare the proposed nonconvex sparse group lasso against four other methods as baselines: group lasso, sparse group lasso (SGL_1), CGES proposed in [92], and the group variant of ℓ_0 regularization (denoted as ℓ_0 for simplicity) proposed in [54]. SGL_1 is optimized using the same algorithm proposed for nonconvex sparse group lasso. For the group terms, the weights are grouped together based on the filters or output channels, which we will refer to as neurons. We trained various CNN architectures on MNIST [41] and CIFAR 10/100 [38]. The MNIST dataset consists of 60k training images and 10k test images. MNIST is trained on two simple CNN architectures: LeNet-5-Caffe [31, 41] and a 4-layer CNN with two convolutional layers (32 and 64 channels, respectively) and an intermediate layer of 1000 fully connected neurons. CIFAR 10/100 is a dataset that has 10/100 classes split into 50k training images and 10k test images. It is trained on Resnets [28] and wide Resnets [95]. Throughout all of our experiments, for $SGSCAD(a)$, we set $a = 3.7$ as suggested in [22]; for $SGTL_1(a)$, we set $a = 1.0$ as suggested in [99]; and for $SGL_1 - L_2$, we set $\alpha = 1.0$ as suggested by the literatures [52, 90, 50, 51]. For CGES, we have $\mu_l = l/L$. Because the optimization algorithms do not drive most, if not all, the weights and neurons to zeroes, we have to set them to zeroes when their values are below a certain threshold. In our experiments, if the absolute weights are below 10^{-5}, we set them to zeroes. Then, **weight sparsity** is defined to be *the percentage of zero weights with respect to the total number of weights trained in the network.* If the normalized sum of the absolute values of the weights of the neuron is less than 10^{-5}, then the weights of the neuron are set to zeroes. **Neuron sparsity** is defined to be *the percentage of neurons whose weights are zeroes with respect to the total number of neurons in the network.*

3.1.1 MNIST Classification

MNIST is trained on Lenet-5-Caffe, which has four layers with 1,370 total neurons and 431,080 total weight parameters. All layers of the network are applied with strictly the same type of regularization. No other regularization methods (e.g., dropout and batch normalization) are used. The network is optimized using Adam [37] with initial learning rate 0.001. For every 40 epochs, the learning rate decays by a factor of 0.1. We set the regularization parameter to the following values: $\lambda = \alpha/60000$ for $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. For SGL_1 and nonconvex sparse group lasso, we set $\beta = 25\alpha/60000$, and for every 40 epochs, β increases by a factor of $\sigma = 1.25$. The network is trained for 200 epochs across 5 runs.

Table 2 reports the mean results for test error, weight sparsity, and neuron sparsity across five runs of Lenet-5-Caffe trained after 200 epochs. We see that although CGES has the lowest test errors at $\alpha \in \{0.1, 0.3, 0.4\}$ and the largest weight sparsity for all $\alpha \in \{0.1, 0.2, \ldots, 0.5\}$, nonconvex sparse group lasso's test errors and weight sparsity are comparable. Additionally, nonconvex sparse group lasso's neuron sparsity is nearly two times larger than the neuron sparsity attained by CGES. Across all parameters and methods, SGL_0 with $\alpha = 0.5$ attains the best average test error of 0.630 with average weight sparsity 95.7% and neuron sparsity 80.7%. Furthermore, its test error is lower than the test errors of other nonconvex sparse group lasso regularization methods for all α's tested. Generally, SGL_1 and nonconvex sparse group lasso outperform ℓ_0 regularization proposed by Louizos et al. [54] and group lasso by average weight and neuron sparsity.

Table 3 reports the mean results for test error, weight sparsity, and neuron sparsity of the Lenet-5-Caffe models with the lowest test errors from the five runs. According to the results, the best test errors are attained by SGL_0 at $\alpha = 0.3, 0.5$; $SGL_1 - L_2$ at $\alpha = 0.2$; and CGES at $\alpha = 0.1, 0.4$. For average weight sparsity, SGL_0 attains the largest weight sparsity at $\alpha \in \{0.2, 0.3, 0.4, 0.5\}$. For average neuron sparsity, the largest values are attained by $SGTL_1$ at $\alpha = 0.1, 0.2$; by SGL_1 at $\alpha = 0.3$; and by SGL_0 at $\alpha = 0.4, 0.5$. Although SGL_0 does not outperform all the other methods across the board, its results are still comparable to the best results. Overall, we see that nonconvex sparse group lasso outperforms ℓ_0 in test error, weight sparsity, and neuron sparsity and group lasso in weight and neuron sparsity.

MNIST is also trained on a 4-layer CNN with two convolutional layers with 32 and 64 channels, respectively, and an intermediate layer with 1000 neurons. Each convolutional layer has a 5×5 convolutional filters. The 4-layer CNN has 2,120 total neurons and 1,087,010 total weight parameters. All layers of the network are applied with strictly the same type of regularization. The network is optimized with the same settings as Lenet-5-Caffe. However, the regularization parameter is different: we have $\lambda = \alpha/60000$ for $\alpha \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. For SGL_1 and nonconvex sparse group lasso, we set $\beta = 5\alpha/60000$ and for every 40 epochs, β increases by a factor of $\sigma = 1.25$. The network is trained for 200 epochs across 5 runs.

Table 4 reports the mean results for test error, weight sparsity, and neuron sparsity across five runs of the 4-layer CNN models trained after 200 epochs. Although CGES consistently has the highest weight sparsity, it does not yield the most accurate models until when $\alpha \geq 0.8$. Moreover, its neuron sparsity is smaller than the neuron sparsity by group lasso, SGL_1, and nonconvex group lasso when $\alpha \geq 0.6$. ℓ_0 has the highest neuron sparsity for all α's given, but its test errors are much greater. When $\alpha \leq 0.6$, $SGSCAD$

yields the most accurate models at $\alpha = 0.2, 0.6$ while SGL_1 yields one at $\alpha = 0.4$. Overall, we see that nonconvex group lasso has comparable weight sparsity and neuron sparsity as group lasso and SGL_1.

Table 5 reports the mean results for test error, weight sparsity, and neuron sparsity of the 4-layer CNN models with the lowest test errors from the five runs. At $\alpha = 0.2$, SGL_1 and $SGSCAD$ have the lowest test errors, but their weight sparsity are exceeded by CGES and their neuron sparsity are exceed by ℓ_0. At $\alpha = 0.4$, $SGL_1 - L_2$ has the lowest test error, but its weight sparsity and neuron sparsity are exceeded by CGES and ℓ_0, respectively. At $\alpha = 0.6$, SGL_1 has the lowest test error, but $SGSCAD$ has the largest weight sparsity with comparable test error. At $\alpha \geq 0.8$, CGES has the lowest test error, but its weight sparsity is exceeded by group lasso, SGL_1, and the nonconvex group lasso regularizers, which all have slightly higher test error. At $\alpha = 0.8$, the neuron sparsity of CGES is comparable to the neuron sparsity of group lasso, SGL_1, and the nonconvex group lasso regularizers. At $\alpha = 1.0$, group lasso has the highest neuron sparsity, but nonconvex group lasso has slightly lower neuron sparsity. In general, weight sparsity of nonconvex group lasso is comparable to or larger than the weight sparsity of group lasso and SGL_1.

3.1.2 CIFAR Classification

CIFAR 10/100 is trained on Resnet-40 and wide Resnet with depth 28 and width 10 (WRN-28-10). Resnet-40 has approximately 570,000 weight parameters and 1520 neurons while WRN-28-10 has approximately 36,500,000 weight parameters and 10,736 neurons. The networks are optimized using stochastic gradient descent with initial learning rate 0.1. After every 60 epochs, learning rate decays by a factor of 0.2. Strictly the same type of regularization is applied to the weights of the hidden layer where dropout is utilized in the residual block. We vary the regularization parameter $\lambda = \alpha/50000$. For Resnet-40, we have $\alpha \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$ for CIFAR 10 and $\alpha \in \{2.0, 2.5, 3.0, 3.5, 4.0\}$ for CIFAR 100. For SGL_1 and nonconvex sparse group lasso, we set $\beta = 15\alpha/50000$ for Resnet-40 and $\beta = 25\alpha/50000$ for WRN-28-10. For every 20 epochs, β increases by a factor of $\sigma = 1.25$. The networks are trained for 200 epochs across 5 runs. We excluded ℓ_0 regularization by Louizos *et al.* [54] because it was unstable for the provided α's. Furthermore, we only analyze the models with the lowest test errors since the test errors did not stabilize by the end of the 200 epochs in our experiments.

Table 6 reports mean test error, weight sparsity, and neuron sparsity across the Resnet-40 models trained on CIFAR 10 with the lowest test errors from the five runs. Group lasso has the lowest test errors for all α's provided while CGES, SGL_1, and nonconvex sparse group lasso are higher by at most 1.1%. When $\alpha \leq 1.5$, CGES has the largest weight sparsity while $SGSCAD$, $SGTL_1$ $SGL_1 - SGL_2$ have larger weight sparsity than does group lasso. At $\alpha = 2.0, 2.5$, $SGSCAD$ has the largest weight sparsity. At $\alpha = 3.0$, SGL_1 has the largest weight sparsity with comparable test error as the nonconvex group lasso regularizers. For neuron sparsity, $SGL_1 - L_2$ has the largest at $\alpha = 1.0$ while $SGSCAD$ has the largest at $\alpha = 1.5, 2.0$. However, at $\alpha = 2.5, 3.0$, group lasso has the largest neuron sparsity. For all α's tested, $SGSCAD$ has higher weight sparsity and neuron sparsity than does SGL_1 but with comparable test error.

Table 7 reports mean test error, weight sparsity, and neuron sparsity across the Resnet-40 models trained on CIFAR 100 with the lowest test errors from the five runs. Group lasso has the lowest test errors for $\alpha \leq 3.5$ while CGES has the lowest test error at $\alpha = 4.0$. However, the weight sparsity and the neuron sparsity of group lasso are lower than the sparsity of SGL_1 and some of the nonconvex sparse group lasso regularizers. CGES has the lowest neuron sparsity across all α's. Among the nonconvex group lasso penalties, $SGSCAD$ has the best test errors, which are lower than the test errors of SGL_1 for all α's except 2.5.

Table 8 reports mean test error, weight sparsity, and neuron sparsity across the WRN-28-10 models trained on CIFAR 10 with the lowest test errors from the five runs. The best test errors are attained by

$SGTL_1$ at $\alpha = 0.05, 0.2, 0.5$; by CGES at $\alpha = 0.01$; and by SGL_1 at $\alpha = 0.1$. Weight sparsity of CGES outperforms the other methods only when $\alpha = 0.01, 0.05, 0.1$, but it underperforms when $\alpha \geq 0.2$. Weight sparsity levels between group lasso and nonconvex group lasso are comparable across all α. For neuron sparsity, $SGL_1 - L_2$ attains the largest values at $\alpha = 0.02, 0.1, 0.2$. Nevertheless, the other nonconvex sparse group lasso methods have comparable neuron sparsity. Overall, SGL_1, SGL_0, $SGSCAD$, and $SGTL_1$ outperform group lasso in test error while having similar or higher weight and neuron sparsity.

Table 9 reports mean test error, weight sparsity, and neuron sparsity across the WRN-28-10 models trained on CIFAR 100 with the lowest test errors from the five runs. According to the results, the best test errors are attained by CGES when $\alpha = 0.01, 0.05$; by $SGSCAD$ when $\alpha = 0.1, 0.5$; and by $SGTL_1$ when $\alpha = 0.2$. Although CGES has the largest weight sparsity for $\alpha = 0.01, 0.05, 0.1, 0.2$, we see that its test error increases as α increases. When $\alpha = 0.5$, the best weight sparsity is attained by $SGSCAD$, but the other methods have comparable weight sparsity. The best neuron sparsity is attained by CGES at $\alpha = 0.01, 0.02$; by $SGL_1 - L_2$ at $\alpha = 0.1, 0.2$; and by $SGSCAD$ at $\alpha = 0.5$. The neuron sparsity among the nonconvex sparse group lasso methods are comparable. For $\alpha \leq 0.2$, we see that SGL_1 and nonconvex sparse group lasso outperform group lasso in test error across α while having comparable weight and neuron sparsity.

3.2 Algorithm Comparison

We compare the proposed Algorithm 1 with direct stochastic gradient descent, where the gradient of the regularizer is approximated by backpropagation, and proximal gradient descent, discussed in Section 2.4, by applying them to SGL_1 on Lenet-5 trained on MNIST. The parameter setting for this CNN is discussed in Section 3.1.1. Table 10 reports the mean results for test error, weight sparsity, and neuron sparsity across five models trained after 200 epochs while Figure 2 provides visualizations. Table 11 and Figure 3 record mean statistics for models with the lowest test errors from the five runs. According to the results, proximal stochastic gradient descent attains the highest level of weight sparsity and neuron sparsity for models trained after 200 epochs and models with the lowest test error. However, their test errors are the highest amongst the three algorithms. On the other hand, our proposed algorithm attains the lowest test errors. For models trained after 200 epochs, the weight sparsity and neuron sparsity attained by Algorithm 1 are comparable to the sparsity attained by direct stochastic gradient descent. For models with the lowest test errors generated from their respective runs, the weight sparsity and neuron sparsity by the proposed algorithm are better than the sparsity by direct stochastic gradient descent. Therefore, our proposed algorithm generates the most accurate model with satisfactory sparsity among the three algorithms for sparse regularization.

4 CONCLUSION AND FUTURE WORK

In this work, we propose nonconvex sparse group lasso, a nonconvex extension of sparse group lasso. The ℓ_1 norm in sparse group lasso on the weight parameters is replaced with a nonconvex regularizer whose proximal operator is a thresholding function. Taking advantage of this property, we develop a new algorithm to optimize loss functions regularized with nonconvex sparse group lasso for CNNs in order to attain a sparse network with competitive accuracy. We compare the proposed family of regularizers with various baseline methods on MNIST and CIFAR 10/100 on different CNNs. The experimental results demonstrate that in general, nonconvex sparse group lasso generates a more accurate and/or more compressed CNN than does group lasso. In addition, we compare our proposed algorithm to direct stochastic gradient descent and proximal gradient descent on Lenet-5 trained on MNIST. The results show that the proposed algorithm to solve SGL_1 yields a satisfactorily sparse network with lower test error than do the other two algorithms.

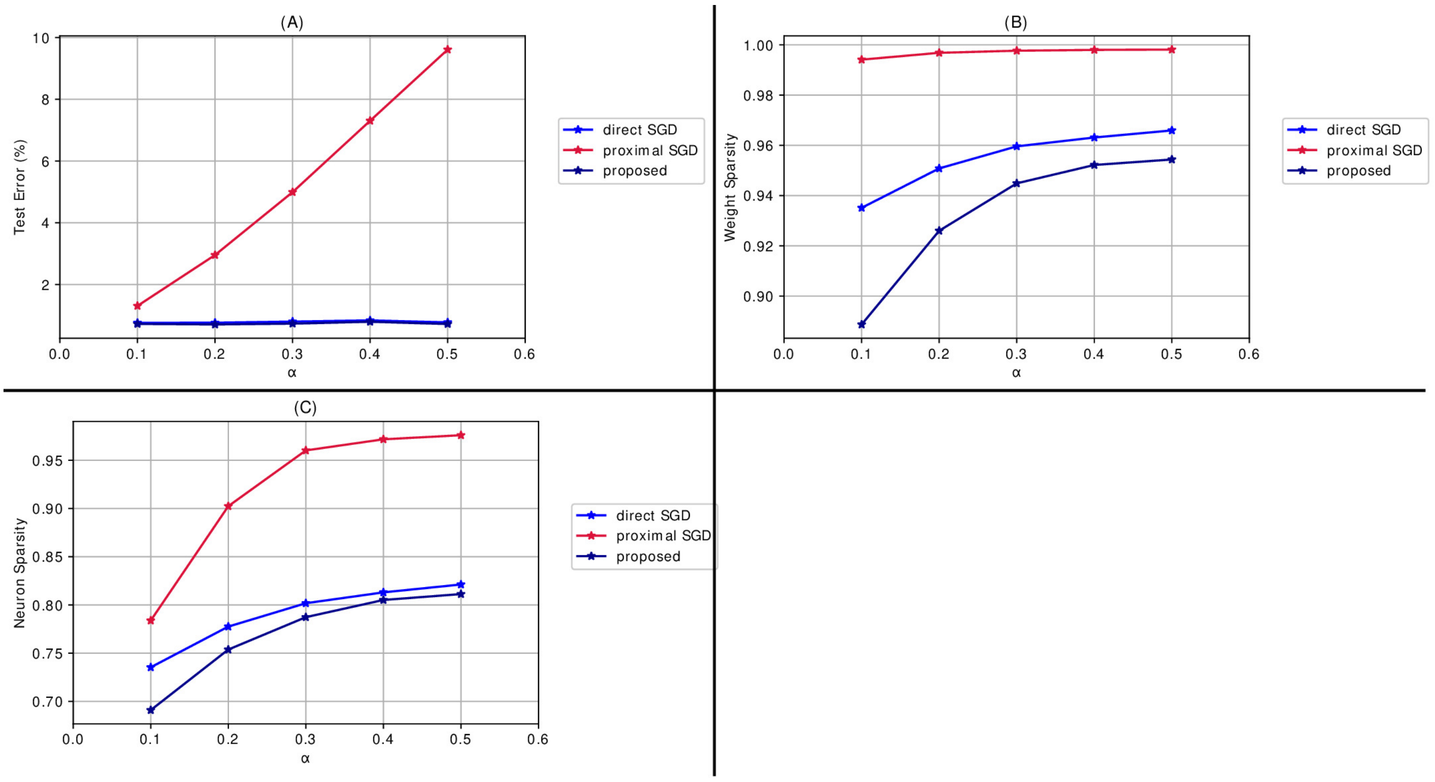

Figure 2. Mean results of algorithms applied to SGL$_1$ for Lenet-5 models trained on MNIST for 200 epochs across 5 runs when varying the regularization parameter $\lambda = \alpha/60000$ when $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. **(A)** Mean test error. **(B)** Mean weight sparsity. **(C)** Mean neuron sparsity.

According to the numerical results, there is no single sparse regularizer that outperforms all other on any CNN trained on a given dataset. One regularizer may perform well in one case while it may perform worse on a different case. Due to the myriad of sparse regularizers to select from and the various parameters to tune, especially for one CNN trained on a given dataset, one direction is to develop an automatic machine learning framework that efficiently selects the right regularizer and parameters. In recent works, automatic machine learning can be represented as a matrix completion problem [88] and a statistical learning problem [24]. These frameworks can be adapted for selecting the best sparse regularizer, thus saving time for users who are training sparse CNNs.

5 PROOFS

We provide proofs for the results discussed in Section 2.5.

5.1 Proof of Theorem 2

By (17a)-(17b), for each $k \in \mathbb{N}$, we have

$$F_\beta(V^k, W^{k+1}) \leq F_\beta(V^k, W) \tag{22}$$

for all W, and

$$F_\beta(V^{k+1}, W^{k+1}) \leq F_\beta(V, W^{k+1}) \tag{23}$$

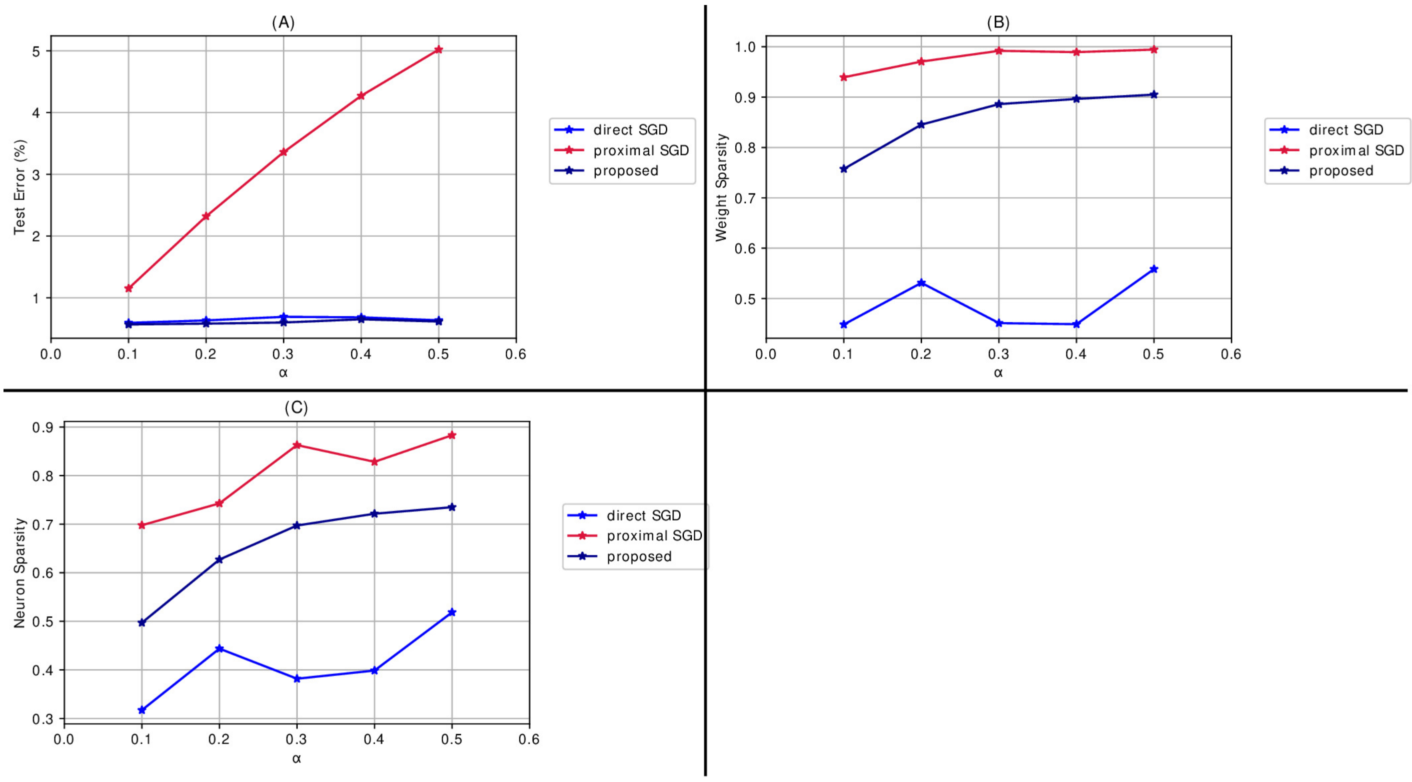

Figure 3. Mean results of algorithms applied to SGL$_1$ for Lenet-5 models trained on MNIST with lowest test errors across 5 runs when varying the regularization parameter $\lambda = \alpha/60000$ when $\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. **(A)** Mean test error. **(B)** Mean weight sparsity. **(C)** Mean neuron sparsity.

for all V. By (23), we have

$$F_\beta(V^+, W^+) \leq F_\beta(V^k, W^+) \tag{24}$$

for each $k \in \mathbb{N}$. Altogether, we have

$$F_\beta(V^+, W^+) \leq F_\beta(V^k, W^k) \tag{25}$$

for each $k \in \mathbb{N}$, so $\{F_\beta(V^k, W^k)\}_{k=1}^\infty$ is nonincreasing. Since $F_\beta(V^k, W^k) \geq 0$ for all $k \in \mathbb{N}$, its limit $\lim\limits_{k \to \infty} F_\beta(V^k, W^k)$ exists. From (22)-(24), we have

$$F_\beta(V^+, W^+) \leq F_\beta(V^k, W^+) \leq F_\beta(V^k, W^k).$$

Taking the limit gives us

$$\lim_{k \to \infty} F_\beta(V^k, W^+) = \lim_{k \to \infty} F_\beta(V^k, W^k). \tag{26}$$

Since (V^*, W^*) is an accumulation point of $\{(V^k, W^k)\}_{k=1}^\infty$, there exists a subsequence K such that

$$\lim_{k \in K \to \infty} (V^k, W^k) = (V^*, W^*). \tag{27}$$

Because $r(\cdot)$ is lower semicontinuous and $\lim_{k\in K\to\infty} V^k = V^*$, there exists $k' \in K$ such that $k \geq k'$ implies $r(V_l^k) \geq r(V_l^*)$ for each $l = 1, \ldots, L$. Using this result along with (23), we obtain

$$\begin{aligned} F_\beta(V, W^k) &\geq F_\beta(V^k, W^k) \\ &= \tilde{\mathcal{L}}(W^k) + \sum_{l=1}^{L} \left[\lambda\left(\mathcal{R}_{GL}(W_l^k) + r(V_l^k)\right) + \frac{\beta}{2}\|V_l^k - W_l^k\|_2^2\right] \\ &\geq \tilde{\mathcal{L}}(W^k) + \sum_{l=1}^{L} \left[\lambda\left(\mathcal{R}_{GL}(W_l^k) + r(V_l^*)\right) + \frac{\beta}{2}\|V_l^k - W_l^k\|_2^2\right] \end{aligned}$$

for $k \geq k'$. As $k \in K \to \infty$, we have

$$F_\beta(V, W^*) \geq \tilde{\mathcal{L}}(W^*) + \sum_{l=1}^{L} \left[\lambda\left(\mathcal{R}_{GL}(W_l^*) + r(V_l^*)\right) + \frac{\beta}{2}\|V_l^* - W_l^*\|_2^2\right] = F_\beta(V^*, W^*) \tag{28}$$

338 by continuity, so it follows that $V^* \in \arg\min_V F_\beta(V, W^*)$.

For notational convenience, let

$$\tilde{\mathcal{R}}_{\lambda,\beta}(V, W) := \sum_{l=1}^{L} \left[\lambda \mathcal{R}_{GL}(W_l) + \frac{\beta}{2}\|V_l - W_l\|_2^2\right]. \tag{29}$$

By (22), we have

$$\begin{aligned} \tilde{\mathcal{L}}(W) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^k, W) &= F_\beta(V^k, W) - \lambda \sum_{i=1}^{L} r(V_l^k) \\ &\geq F_\beta(V^k, W^+) - \lambda \sum_{i=1}^{L} r(V_l^k) = \tilde{\mathcal{L}}(W^+) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^k, W^+). \end{aligned} \tag{30}$$

Because $\lim_{k\in K\to\infty} V^k$ exists, the sequence $\{V^k\}_{k\in K}$ is bounded. If $r(\cdot)$ is ℓ_0, transformed ℓ_1, or SCAD, then $\{r(V^k)\}_{k\in K}$ is bounded. If $r(\cdot)$ is ℓ_1, then $r(\cdot)$ is coercive. If $r(\cdot)$ is $\ell_1 - \alpha\ell_2$, then $r(\cdot)$ is bounded above by ℓ_1. Overall, this follows that $\{r(V^k)\}_{k\in K}$ bounded as well. Hence, there exists a further subsequence

$\overline{K} \subset K$ such that $\lim\limits_{k\in\overline{K}\to\infty} r(V^k)$ exists. So, we obtain

$$\begin{aligned}
\lim_{k\in\overline{K}\to\infty} \tilde{\mathcal{L}}(W^+) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^k, W^+) &= \lim_{k\in\overline{K}\to\infty} F_\beta(V^k, W^+) - \lambda\sum_{i=1}^{L} r(V_l^k) \\
&= \lim_{k\in\overline{K}\to\infty} F_\beta(V^k, W^+) - \lim_{k\in\overline{K}\to\infty} \lambda\sum_{i=1}^{L} r(V_l^k) \\
&= \lim_{k\in\overline{K}\to\infty} F_\beta(V^k, W^k) - \lim_{k\in\overline{K}\to\infty} \lambda\sum_{i=1}^{L} r(V_l^k) \qquad (31)\\
&= \lim_{k\in\overline{K}\to\infty} F_\beta(V^k, W^k) - \lambda\sum_{i=1}^{L} r(V_l^k) \\
&= \lim_{k\in\overline{K}\to\infty} \tilde{\mathcal{L}}(W^k) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^k, W^k) \\
&= \tilde{\mathcal{L}}(W^*) + \tilde{\mathcal{R}}_{\lambda,\beta}(W^*, V^*)
\end{aligned}$$

after applying (26) in the third inequality and by continuity in the last equality.

Taking the limit over the subsequence $\overline{K}$ in (30) and applying (31), we obtain

$$\tilde{\mathcal{L}}(W) + \tilde{\mathcal{R}}_{\lambda,\beta}(V^*, W) \geq \tilde{\mathcal{L}}(W^*) + \tilde{\mathcal{R}}_{\lambda,\beta}(W^*, V^*) \qquad (32)$$

by continuity. Adding $\sum_{l=1}^{L} r(V_l^*)$ on both sides yields

$$F_\beta(V^*, W) \geq F_\beta(V^*, W^*), \qquad (33)$$

which follows that $W^* \in \arg\min_W F_\beta(V^*, W)$. This completes the proof.

5.2 Proof of Theorem 3

Because (V^*, W^*) is an accumulation point, there exists a subsequence K such that $\lim\limits_{k\in K\to\infty}(V^k, W^k) = (V^*, W^*)$. If $\{F_{\beta_k}(V^k, W^k)\}_{k=1}^{\infty}$ is uniformly bounded, there exists M such that $F_{\beta_k}(V^k, W^k) \leq M$ for all $k \in \mathbb{N}$. Then we have

$$M \geq F_{\beta_k}(V^k, W^k) = \tilde{\mathcal{L}}(W) + \sum_{l=1}^{L}\left[\lambda\mathcal{R}_{GL}(W_l) + \lambda r(V_l) + \frac{\beta_k}{2}\|V_l - W_l\|_2^2\right] \geq \frac{\beta_k}{2}\sum_{l=1}^{L}\|V_l - W_l\|_2^2$$

As a result,

$$\sum_{l=1}^{L}\|V_l^k - W_l^k\|_2^2 \leq \frac{2}{\beta_k}M. \qquad (34)$$

Taking the limit over $k \in K$, we have

$$\sum_{l=1}^{L}\|V_l^* - W_l^*\|_2^2 = 0,$$

which follows that $V^* = W^*$. As a result, (V^*, W^*) is a feasible solution to (15).

PERMISSION TO REUSE AND COPYRIGHT

CONFLICT OF INTEREST STATEMENT

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

AUTHOR CONTRIBUTIONS

KB and FP performed the experiments and analysis. All authors contributed to the design, evaluation, discussions and production of the manuscript.

FUNDING

The work was partially supported by NSF grants IIS-1632935, DMS-1854434, DMS-1924548 and the Qualcomm Faculty Award.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Thu Dinh for helpful conversations. They also thank Christos Louizos for answering our questions we had regarding his work in [54]. Lastly, the authors thank AWS Cloud Credits for Research and Google Cloud Platform (GCP) for providing cloud based computational resources for this work.

DATA AVAILABILITY STATEMENT

The datasets MNIST and CIFAR 10/100 for this study are available through the Pytorch package in Python. Codes for the numerical experiments in Section 3 are available at `https://github.com/kbui1993/Official_Nonconvex_SGL`.

REFERENCES

[1] Aghasi, A., Abdi, A., Nguyen, N., and Romberg, J. (2017). Net-trim: Convex pruning of deep neural networks with performance guarantee. In *Advances in Neural Information Processing Systems*. 3177–3186

[2] Aghasi, A., Abdi, A., and Romberg, J. (2020). Fast convex pruning of deep neural networks. *SIAM Journal on Mathematics of Data Science* 2, 158–188

[3] Ahn, M., Pang, J.-S., and Xin, J. (2017). Difference-of-convex learning: directional stationarity, optimality, and sparsity. *SIAM Journal on Optimization* 27, 1637–1665

[4] Alvarez, J. M. and Salzmann, M. (2016). Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*. 2270–2278

[5] Antoniadis, A. and Fan, J. (2001). Regularization of wavelet approximations. *Journal of the American Statistical Association* 96, 939–967

[6] Ba, J. and Caruana, R. (2014). Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*. 2654–2662

[7] Bach, F. R. (2008). Consistency of the group lasso and multiple kernel learning. *Journal of Machine Learning Research* 9, 1179–1225

[8] Bao, C., Dong, B., Hou, L., Shen, Z., Zhang, X., and Zhang, X. (2016). Image restoration by minimizing zero norm of wavelet frame coefficients. *Inverse Problems* 32, 115004

[9] Breheny, P. and Huang, J. (2011). Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *The Annals of Applied Statistics* 5, 232

[10] Candès, E. J., Li, X., Ma, Y., and Wright, J. (2011). Robust principal component analysis? *Journal of the ACM (JACM)* 58, 1–37

[11] Candès, E. J., Romberg, J. K., and Tao, T. (2006). Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics* 59, 1207–1223

[12] Chan, R. H., Chan, T. F., Shen, L., and Shen, Z. (2003). Wavelet algorithms for high-resolution image reconstruction. *SIAM Journal on Scientific Computing* 24, 1408–1432

[13] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 834–848

[14] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2017). A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*

[15] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2018). Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine* 35, 126–136

[16] Cohen, A., Dahmen, W., and DeVore, R. (2009). Compressed sensing and best k-term approximation. *Journal of the American mathematical society* 22, 211–231

[17] Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*. 1269–1277

[18] Dinh, T. and Xin, J. (September 3-4, 2020). Convergence of a relaxed variable splitting method for learning sparse neural networks via ℓ_1,ℓ_0, and transformed-ℓ_1 penalties. *arXiv preprint arXiv:1812.05719, in Proceedings of Intelligent Systems Conference (IntelliSys), Amsterdam, The Netherlands*

[19] Dong, B. and Zhang, Y. (2013). An efficient algorithm for ℓ_0 minimization in wavelet frame based image restoration. *Journal of Scientific Computing* 54, 350–368

[20] Donoho, D. L. and Elad, M. (2003). Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proceedings of the National Academy of Sciences* 100, 2197–2202

[21] Esser, E., Lou, Y., and Xin, J. (2013). A method for finding structured sparse solutions to nonnegative least squares problems with applications. *SIAM Journal on Imaging Sciences* 6, 2010–2046

[22] Fan, J. and Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association* 96, 1348–1360

[23] Foucart, S. and Rauhut, H. (2013). An invitation to compressive sensing. In *A mathematical introduction to compressive sensing* (Springer). 1–39

[24] Gupta, R. and Roughgarden, T. (2017). A pac approach to application-specific algorithm selection. *SIAM Journal on Computing* 46, 992–1017

[25] Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*

[26] Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*. 1135–1143

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

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

[29] Hu, H., Peng, R., Tai, Y.-W., and Tang, C.-K. (2016). Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*

[30] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., et al. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7310–7311

[31] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia* (ACM), 675–678

[32] Jin, X., Yuan, X., Feng, J., and Yan, S. (2016). Training skinny deep neural networks with iterative hard thresholding methods. *arXiv preprint arXiv:1607.05423*

[33] Jung, H., Ye, J. C., and Kim, E. Y. (2007). Improved k–t blast and k–t sense using focuss. *Physics in Medicine & Biology* 52, 3201

[34] Jung, M. (2017). Piecewise-smooth image segmentation models with L^1 data-fidelity terms. *Journal of Scientific Computing* 70, 1229–1261

[35] Jung, M., Kang, M., and Kang, M. (2014). Variational image segmentation models involving non-smooth data-fidelity terms. *Journal of scientific computing* 59, 277–308

[36] Kim, C. and Klabjan, D. (2019). A simple and fast algorithm for L1-norm kernel pca. *IEEE transactions on pattern analysis and machine intelligence*

[37] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*

[38] Krizhevsky, A. and Hinton, G. (2009). *Learning multiple layers of features from tiny images*. Tech. rep., Citeseer

[39] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105

[40] Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In *Advances in neural information processing systems*. 950–957

[41] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 2278–2324

[42] Li, F., Osher, S., Qin, J., and Yan, M. (2016). A multiphase image segmentation based on fuzzy membership functions and l1-norm fidelity. *Journal of Scientific Computing* 69, 82–106

[43] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*

[44] Li, P., Chen, W., Ge, H., and Ng, M. K. (2020). $\ell_1 - \alpha\ell_2$ minimization methods for signal and image reconstruction with impulsive noise removal. *Inverse Problems* 36, 055009

[45] Li, Z., Luo, X., Wang, B., Bertozzi, A. L., and Xin, J. (2019). A study on graph-structured recurrent neural networks and sparsification with application to epidemic forecasting. In *World Congress on Global Optimization* (Springer), 730–739

[46] Lim, M., Ales, J. M., Cottereau, B. R., Hastie, T., and Norcia, A. M. (2017). Sparse EEG/MEG source estimation via a group lasso. *PloS one* 12, e0176835

[47] Lin, D., Calhoun, V. D., and Wang, Y.-P. (2014). Correspondence between fMRI and SNP data by group sparse canonical correlation analysis. *Medical image analysis* 18, 891–902

[48] Lin, D., Zhang, J., Li, J., Calhoun, V. D., Deng, H.-W., and Wang, Y.-P. (2013). Group sparse canonical correlation analysis for genomic data integration. *BMC bioinformatics* 14, 1–16

[49] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3431–3440

[50] Lou, Y., Osher, S., and Xin, J. (2015). Computational aspects of constrained $L_1 - L_2$ minimization for compressive sensing. In *Modelling, Computation and Optimization in Information Systems and Management Sciences* (Springer). 169–180

[51] Lou, Y. and Yan, M. (2018). Fast $L_1 - L_2$ minimization via a proximal operator. *Journal of Scientific Computing* 74, 767–785

[52] Lou, Y., Yin, P., He, Q., and Xin, J. (2015). Computing sparse representation in a highly coherent dictionary based on difference of L_1 and L_2. *Journal of Scientific Computing* 64, 178–196

[53] Lou, Y., Zeng, T., Osher, S., and Xin, J. (2015). A weighted difference of anisotropic and isotropic total variation model for image processing. *SIAM Journal on Imaging Sciences* 8, 1798–1823

[54] Louizos, C., Welling, M., and Kingma, D. P. (2017). Learning sparse neural networks through l_0 regularization. *International Conference on Learning Representations, 2018; CoRR* abs/1712.01312

[55] Lu, J., Qiao, K., Li, X., Lu, Z., and Zou, Y. (2019). ℓ_0-minimization methods for image restoration problems based on wavelet frames. *Inverse Problems* 35, 064001

[56] Lu, Z. and Zhang, Y. (2013). Sparse approximation via penalty decomposition methods. *SIAM Journal on Optimization* 23, 2448–2478

[57] Lustig, M., Donoho, D., and Pauly, J. M. (2007). Sparse mri: The application of compressed sensing for rapid mr imaging. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine* 58, 1182–1195

[58] Lv, J., Fan, Y., et al. (2009). A unified approach to model selection and sparse recovery using regularized least squares. *The Annals of Statistics* 37, 3498–3528

[59] Lyu, J., Zhang, S., Qi, Y., and Xin, J. (2020). Autoshufflenet: Learning permutation matrices via an exact Lipschitz continuous penalty in deep convolutional neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 608–616

[60] Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*. 116–131

[61] Ma, R., Miao, J., Niu, L., and Zhang, P. (2019). Transformed ℓ_1 regularization for learning sparse deep neural networks. *arXiv preprint arXiv:1901.01021*

[62] Ma, S., Song, X., and Huang, J. (2007). Supervised group lasso with applications to microarray data analysis. *BMC bioinformatics* 8, 60

[63] Ma, T.-H., Lou, Y., Huang, T.-Z., and Zhao, X.-L. (2017). Group-based truncated L_{1-2} model for image inpainting. In *2017 IEEE International Conference on Image Processing (ICIP)* (IEEE), 2079–2083

[64] Mehranian, A., Rad, H. S., Rahmim, A., Ay, M. R., and Zaidi, H. (2013). Smoothly Clipped Absolute Deviation (SCAD) regularization for compressed sensing MRI using an augmented lagrangian scheme. *Magnetic resonance imaging* 31, 1399–1411

[65] Meier, L., Van De Geer, S., and Bühlmann, P. (2008). The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 70, 53–71

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

[67] Nie, F., Wang, H., Huang, H., and Ding, C. (2011). Unsupervised and semi-supervised learning via ℓ_1-norm graph. In *2011 International Conference on Computer Vision* (IEEE), 2268–2273

[68] Nikolova, M. (2000). Local strong homogeneity of a regularized estimator. *SIAM Journal on Applied Mathematics* 61, 633–658

[69] Nocedal, J. and Wright, S. (2006). *Numerical optimization* (Springer Science & Business Media)

[70] Parikh, N., Boyd, S., et al. (2014). Proximal algorithms. *Foundations and Trends® in Optimization* 1, 127–239

[71] Park, F., Lou, Y., and Xin, J. (2016). A weighted difference of anisotropic and isotropic total variation for relaxed mumford-shah image segmentation. In *2016 IEEE International Conference on Image Processing (ICIP)* (IEEE), 4314–4318

[72] Parkhi, O. M., Vedaldi, A., Zisserman, A., et al. (2015). Deep face recognition. In *bmvc*. 6

[73] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99

[74] Santosa, F. and Symes, W. W. (1986). Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing* 7, 1307–1330

[75] Scardapane, S., Comminiello, D., Hussain, A., and Uncini, A. (2017). Group sparse regularization for deep neural networks. *Neurocomputing* 241, 81–89

[76] Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. (2013). A sparse-group lasso. *Journal of Computational and Graphical Statistics* 22, 231–245

[77] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *CoRR* abs/1409.1556

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

[79] Tran, H. and Webster, C. (2019). A class of null space conditions for sparse recovery via nonconvex, non-separable minimizations. *Results in Applied Mathematics* 3, 100011

[80] Trzasko, J., Manduca, A., and Borisch, E. (2007). Sparse mri reconstruction via multiscale L_0-continuation. In *2007 IEEE/SP 14th Workshop on Statistical Signal Processing* (IEEE), 176–180

[81] Ullrich, K., Meeds, E., and Welling, M. (2017). Soft weight-sharing for neural network compression. *stat* 1050, 9

[82] Vershynin, R. (2018). *High-dimensional probability: An introduction with applications in data science*, vol. 47 (Cambridge university press)

[83] Vincent, M. and Hansen, N. R. (2014). Sparse group lasso and high dimensional multinomial classification. *Computational Statistics & Data Analysis* 71, 771–786

[84] Wang, L., Chen, G., and Li, H. (2007). Group scad regression analysis for microarray time course gene expression data. *Bioinformatics* 23, 1486–1494

[85] Wen, F., Chu, L., Liu, P., and Qiu, R. C. (2018). A survey on nonconvex regularization-based sparse and low-rank recovery in signal processing, statistics, and machine learning. *IEEE Access* 6, 69883–69906

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

[87] Xue, F. and Xin, J. (2019). Learning sparse neural networks via ℓ_0 and Tℓ_1 by a relaxed variable splitting method with application to multi-scale curve classification. In *World Congress on Global*

Optimization (Springer), 800–809

[88] Yang, C., Akimoto, Y., Kim, D. W., and Udell, M. (2019). Oboe: Collaborative filtering for automl model selection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1173–1183

[89] Ye, Q., Zhao, H., Li, Z., Yang, X., Gao, S., Yin, T., et al. (2017). L1-norm distance minimization-based fast robust twin support vector k-plane clustering. *IEEE transactions on neural networks and learning systems* 29, 4494–4503

[90] Yin, P., Lou, Y., He, Q., and Xin, J. (2015). Minimization of ℓ_{1-2} for compressed sensing. *SIAM Journal on Scientific Computing* 37, A536–A563

[91] Yin, P., Sun, Z., Jin, W.-L., and Xin, J. (2017). ℓ_1-minimization method for link flow correction. *Transportation Research Part B: Methodological* 104, 398–408

[92] Yoon, J. and Hwang, S. J. (2017). Combined group and exclusive sparsity for deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (JMLR. org), 3958–3966

[93] Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68, 49–67

[94] Yuan, X.-T., Li, P., and Zhang, T. (2017). Gradient hard thresholding pursuit. *Journal of Machine Learning Research* 18, 166–1

[95] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*

[96] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*

[97] Zhang, S. and Xin, J. (2017). Minimization of transformed l_1 penalty: Closed form representation and iterative thresholding algorithms. *Communications in Mathematical Sciences* 15, 511 – 537

[98] Zhang, S. and Xin, J. (2018). Minimization of transformed l_1 penalty: theory, difference of convex function algorithm, and robust application in compressed sensing. *Mathematical Programming* 169, 307–336

[99] Zhang, S., Yin, P., and Xin, J. (2017). Transformed schatten-1 iterative thresholding algorithms for low rank matrix completion. *Communications in Mathematical Sciences* 15, 839 – 862

[100] Zhang, X., Lu, Y., and Chan, T. (2012). A novel sparsity reconstruction method from poisson data for 3d bioluminescence tomography. *Journal of scientific computing* 50, 519–535

[101] Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 6848–6856

[102] Zhang, Y., Dong, B., and Lu, Z. (2013). ℓ_0 minimization for wavelet frame based image restoration. *Mathematics of Computation* 82, 995–1015

[103] Zhou, H., Sehl, M. E., Sinsheimer, J. S., and Lange, K. (2010). Association screening of common and rare genetic variants by penalized regression. *Bioinformatics* 26, 2375

[104] Zhou, Y., Jin, R., and Hoi, S. C.-H. (2010). Exclusive lasso for multi-task feature selection. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 988–995

[105] Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., et al. (2018). Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*. 875–886

Table 1. Regularization penalties and their corresponding proximal operators with $\lambda > 0$.

Regularizer Name	Penalty Formulation	Proximal Operator
ℓ_1	$\lambda\lVert x\rVert_1 = \lambda\sum_{i=1}^{n}\lvert x_i\rvert$	$\operatorname{prox}_{\lambda\lVert\cdot\rVert_1}(x) = (\mathcal{S}_\lambda(x_1),\dots,\mathcal{S}_\lambda(x_n))\,,$ with $\mathcal{S}_\lambda(t) = \operatorname{sign}(t)\max\{\lvert t\rvert-\lambda, 0\}$
ℓ_0	$\lambda\lVert x\rVert_0 = \lambda\sum_{i=1}^{n}\lvert x_i\rvert_0$	$\operatorname{prox}_{\lambda\lVert\cdot\rVert_0}(x) = (\mathcal{H}_\lambda(x_1),\dots,\mathcal{H}_\lambda(x_n))\,,$ with $\mathcal{H}_\lambda(t) = \begin{cases} 0 & \text{if } \lvert t\rvert \le \sqrt{2\lambda} \\ t & \text{if } \lvert t\rvert > \sqrt{2\lambda} \end{cases}$
SCAD(a)	$\lambda\lVert x\rVert_{\mathrm{SCAD}(a)} = \sum_{i=1}^{n}\lambda\lvert x_i\rvert_{\mathrm{SCAD}(a)}$ with $\lambda\lvert t\rvert_{\mathrm{SCAD}(a)} = \begin{cases} \lambda\lvert t\rvert & \text{if } \lvert t\rvert < \lambda \\ \frac{2a\lambda\lvert t\rvert - t^2 - \lambda^2}{2(a-1)} & \text{if } \lambda \le \lvert t\rvert < a\lambda \\ (a+1)\lambda^2/2 & \text{if } \lvert t\rvert \ge a\lambda \end{cases}$	$\operatorname{prox}_{\lambda\lVert\cdot\rVert_{\mathrm{SCAD}(a)}}(x) = (\mathscr{S}_{a,\lambda}(x_1),\dots,\mathscr{S}_{a,\lambda}(x_n))\,,$ with $\mathscr{S}_{a,\lambda}(t) = \begin{cases} \mathcal{S}_\lambda(t) & \text{if } \lvert t\rvert \le 2\lambda \\ \frac{(a-1)t - \operatorname{sign}(t)a\lambda}{a-2} & \text{if } 2\lambda < \lvert t\rvert \le a\lambda \\ t & \text{if } \lvert t\rvert > a\lambda. \end{cases}$
TL1(a)	$\lambda\lVert x\rVert_{\mathrm{TL1}(a)} = \lambda\sum_{i=1}^{n}\frac{(a+1)\lvert x_i\rvert}{a+\lvert x_i\rvert}$	$\operatorname{prox}_{\lambda\lVert\cdot\rVert_{\mathrm{TL1}(a)}}(x) = (\mathcal{T}_{a,\lambda}(x_1),\dots,\mathcal{T}_{a,\lambda}(x_n))\,,$ with $\mathcal{T}_{a,\lambda}(t) = \begin{cases} 0 & \text{if } \lvert t\rvert \le \tau(a,\lambda) \\ g_{a,\lambda}(t) & \text{if } \lvert t\rvert > \tau(a,\lambda) \end{cases}$ where $g_{a,\lambda}(t) = \operatorname{sign}(t)\left(\frac{2}{3}(a+\lvert t\rvert)\cos\left(\frac{\phi_{a,\lambda}(t)}{3}\right) - \frac{2a}{3} + \frac{\lvert t\rvert}{3}\right),$ $\phi_{a,\lambda}(t) = \arccos\left(1 - \frac{27\lambda a(a+1)}{2(a+\lvert t\rvert)^3}\right),$ and $\tau(a,\lambda) = \begin{cases} \sqrt{2\lambda(a+1)} - \frac{a}{2} & \text{if } \lambda > \frac{a^2}{2(a+1)} \\ \lambda\frac{a+1}{a} & \text{if } \lambda \le \frac{a^2}{2(a+1)} \end{cases}$
$\ell_1 - \ell_2$	$\lambda\lVert x\rVert_{\ell_1-\ell_2} = \lambda\left(\sum_{i=1}^{n}\lvert x_i\rvert - \sqrt{\sum_{i=1}^{n}x_i^2}\right)$	$\operatorname{prox}_{\lambda\lVert\cdot\rVert_{\ell_1-\ell_2}}(x) = \begin{cases} \frac{\lVert z_1\rVert_2+\lambda}{\lVert z_1\rVert_2}z_1 & \text{if } \lVert x\rVert_\infty > \lambda \\ z_2 & \text{if } 0 \le \lVert x\rVert_\infty \le \lambda \end{cases}$ with $z_1 = \mathcal{S}_\lambda(x)$ and $(z_2)_i = \begin{cases} 0 & \text{if } i \ne k \\ \operatorname{sign}(x_i)\lVert x\rVert_\infty & \text{if } i = k, \end{cases}$ where $k = \underset{1\le k\le n}{\arg\min}\{\lvert x_i\rvert = \lVert x\rVert_\infty\}$.

Table 2. Average test error, weight sparsity, and neuron sparsity of Lenet-5 models trained on MNIST after 200 epochs across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.1$	0.816 (0.024)	**0.644** (0.039)	0.742 (0.030)	0.722 (0.028)	0.682 (0.044)	0.734 (0.039)	0.716 (0.048)	0.688 (0.034)
$\alpha = 0.2$	0.914 (0.029)	0.718 (0.044)	0.772 (0.031)	**0.704** (0.031)	0.712 (0.042)	0.788 (0.045)	0.718 (0.025)	0.746 (0.031)
$\alpha = 0.3$	1.032 (0.045)	**0.678** (0.007)	0.782 (0.035)	0.732 (0.045)	0.686 (0.048)	0.760 (0.037)	0.728 (0.034)	0.712 (0.061)
$\alpha = 0.4$	1.062 (0.030)	**0.662** (0.024)	0.820 (0.054)	0.792 (0.034)	0.704 (0.033)	0.786 (0.045)	0.766 (0.045)	0.756 (0.014)
$\alpha = 0.5$	1.098 (0.035)	0.696 (0.016)	0.834 (0.033)	0.720 (0.039)	**0.630** (0.024)	0.728 (0.044)	0.684 (0.024)	0.750 (0.017)

Avg. Weight Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.1$	2.12×10^{-4} (1.54×10^{-5})	**0.940** (1.51×10^{-3})	0.885 (2.25×10^{-3})	0.889 (4.30×10^{-3})	0.894 (3.81×10^{-3})	0.894 (3.61×10^{-3})	0.901 $(1.57\times10^{-3}$	0.893 (2.77×10^{-3})
$\alpha = 0.2$	2.16×10^{-4} (3.76×10^{-6})	**0.952** (1.51×10^{-3})	0.922 (2.07×10^{-3})	0.926 (1.19×10^{-3})	0.926 (1.75×10^{-3})	0.926 (3.31×10^{-3})	0.930 (2.37×10^{-3})	0.923 (2.86×10^{-3})
$\alpha = 0.3$	2.24×10^{-4} (5.35×10^{-6})	**0.956** (1.41×10^{-3})	0.933 (1.03×10^{-3})	0.945 (1.43×10^{-3})	0.941 (1.73×10^{-3})	0.941 (2.52×10^{-3})	0.941 (1.28×10^{-3})	0.943 (1.04×10^{-3})
$\alpha = 0.4$	2.06×10^{-4} (6.27×10^{-6})	**0.960** (1.05×10^{-3})	0.943 $1.63\times10^{-3})$	0.952 (1.21×10^{-3})	0.951 (1.82×10^{-3})	0.950 (1.64×10^{-3})	0.952 (1.91×10^{-3})	0.952 (1.14×10^{-3})
$\alpha = 0.5$	2.27×10^{-4} (1.53×10^{-5})	**0.963** (1.85×10^{-3})	0.946 (1.43×10^{-3})	0.954 (1.63×10^{-3})	0.957 (9.21×10^{-4})	0.956 (1.37×10^{-3})	0.956 (2.00×10^{-3})	0.956 (2.43×10^{-3})

Avg. Neuron Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.1$	0.531 (3.79×10^{-4})	0.387 (9.13×10^{-3})	0.696 (2.42×10^{-3})	0.691 (7.38×10^{-3})	0.682 (6.27×10^{-3})	**0.704** (3.94×10^{-3})	0.703 (5.09×10^{-3})	0.697 (3.93×10^{-3})
$\alpha = 0.2$	0.578 (1.19×10^{-3})	0.449 (1.26×10^{-2})	0.756 (3.39×10^{-3})	0.754 (2.72×10^{-3})	0.740 (4.01×10^{-3})	**0.758** (5.78×10^{-3})	0.757 (3.93×10^{-3})	0.749 (6.50×10^{-3})
$\alpha = 0.3$	0.602 (4.42×10^{-4})	0.476 (1.17×10^{-2})	0.776 (3.18×10^{-3})	**0.787** (2.55×10^{-3})	0.769 (4.44×10^{-3})	0.785 (4.97×10^{-3})	0.774 (4.11×10^{-3})	0.783 (3.78×10^{-3})
$\alpha = 0.4$	0.616 (7.58×10^{-4})	0.518 (9.72×10^{-3})	0.795 (3.44×10^{-3})	**0.805** (3.89×10^{-3})	0.791 (5.40×10^{-3})	0.803 (3.35×10^{-3})	0.799 (3.56×10^{-3})	0.804 (2.69×10^{-3})
$\alpha = 0.5$	0.626 (1.07×10^{-3})	0.539 (1.27×10^{-2})	0.799 (2.59×10^{-3})	0.811 (4.07×10^{-3})	0.807 (3.15×10^{-3})	**0.819** (2.79×10^{-3})	0.811 (6.29×10^{-3})	0.815 (6.10×10^{-3})

Table 3. Average test error, weight sparsity, and neuron sparsity of Lenet-5 models trained on MNIST with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.1$	0.682 (0.023)	**0.532** (0.031)	0.568 (0.026)	0.568 (0.021)	0.576 (0.027)	0.602 (0.027)	0.582 (0.028)	0.554 (0.056)
$\alpha = 0.2$	0.846 (0.033)	0.584 (0.038)	0.630 (0.017)	0.582 (0.035)	0.584 (0.049)	0.616 (0.021)	0.592 (0.026)	**0.578** (0.032)
$\alpha = 0.3$	0.980 (0.033)	0.590 (0.028)	0.642 (0.013)	0.600 (0.030)	**0.588** (0.019)	0.618 (0.037)	0.594 (0.022)	0.596 (0.039)
$\alpha = 0.4$	1.014 (0.019)	**0.562** (0.015)	0.680 (0.038)	0.652 (0.025)	0.604 (0.033)	0.630 (0.035)	0.630 (0.048)	0.628 (0.020)
$\alpha = 0.5$	1.066 (0.024)	0.598 (0.027)	0.682 (0.043)	0.616 (0.052)	**0.572** (0.012)	0.654 (0.015)	0.586 (0.034)	0.670 (0.026)

Avg. Weight Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.1$	2.38×10^{-4} (1.97×10^{-5})	0.541 (0.024)	0.661 (0.073)	0.757 (0.015)	0.768 (0.019)	0.680 (0.167)	**0.773** (7.48×10^{-3})	0.719 (0.066)
$\alpha = 0.2$	2.26×10^{-4} (9.43×10^{-6})	0.583 (0.017)	0.728 (0.170)	0.845 (4.79×10^{-3})	**0.857** (6.15×10^{-3})	0.821 (0.041)	0.854 (5.60×10^{-3})	0.836 (6.76×10^{-3})
$\alpha = 0.3$	2.19×10^{-4} (1.36×10^{-5})	0.603 (0.020)	0.810 (0.078)	0.886 (3.69×10^{-3})	**0.889** (3.62×10^{-3})	0.878 (9.43×10^{-4})	0.827 (0.115)	0.879 (3.97×10^{-3})
$\alpha = 0.4$	2.22×10^{-4} (1.47×10^{-5})	0.627 (0.019)	0.845 (0.040)	0.896 (3.57×10^{-3})	**0.905** (3.66×10^{-3})	0.846 (0.097)	0.899 (4.23×10^{-3})	0.852 (0.097)
$\alpha = 0.5$	2.24×10^{-4} (1.02×10^{-5})	0.633 (0.013)	0.886 (6.40×10^{-3})	0.905 (2.87×10^{-3})	**0.922** (0.015)	0.902 (2.64×10^{-3})	0.871 (0.084)	0.848 (0.080)

Avg. Neuron Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.1$	0.363 (0.047)	0.315 (0.030)	0.389 (0.120)	0.497 (0.014)	0.496 (0.030)	0.426 (0.172)	**0.513** (9.57×10^{-3})	0.440 (0.107)
$\alpha = 0.2$	0.574 (2.22×10^{-3})	0.392 (0.016)	0.498 (0.185)	0.627 (0.011)	0.631 (0.012)	0.549 (0.169)	**0.634** (9.30×10^{-3})	0.608 (0.015)
$\alpha = 0.3$	0.599 (2.61×10^{-3})	0.418 (0.021)	0.570 (0.154)	**0.697** (9.73×10^{-3})	0.692 (8.19×10^{-3})	0.684 (5.69×10^{-3})	0.613 (0.154)	0.686 (8.60×10^{-3})
$\alpha = 0.4$	0.614 (1.71×10^{-3})	0.482 (0.020)	0.586 (0.184)	0.721 (8.16×10^{-3})	**0.725** (9.97×10^{-3})	0.642 (0.151)	0.724 (0.015)	0.655 (0.150)
$\alpha = 0.5$	0.625 (1.55×10^{-3})	0.492 (0.024)	0.708 (8.94×10^{-3})	0.735 (3.73×10^{-3})	**0.759** (0.020)	0.733 (8.59×10^{-3})	0.683 (0.143)	0.570 (0.216)

Table 4. Average test error, weight sparsity, and neuron sparsity of 4-layer CNN models trained on MNIST after 200 epochs across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.2$	0.962 (0.041)	0.470 (0.036)	0.486 (0.030)	0.418 (0.010)	0.432 (0.023)	**0.408** (0.013)	0.418 (0.026)	0.436 (0.012)
$\alpha = 0.4$	1.454 (0.070)	0.486 (0.030)	0.502 (0.035)	**0.436** (0.026)	0.49 (0.017)	0.456 (0.016)	0.47 (0.035)	0.446 (0.031)
$\alpha = 0.6$	2.396 (0.066)	0.512 (0.035)	0.510 (0.028)	0.494 (0.031)	0.500 (0.023)	**0.488** (0.019)	0.498 (0.025)	0.522 (0.019)
$\alpha = 0.8$	3.396 (0.096)	**0.502** (0.020)	0.544 (0.026)	0.542 (0.025)	0.536 (0.037)	0.524 (0.015)	0.536 (0.014)	0.524 (0.015)
$\alpha = 1.0$	4.74 (0.148)	**0.524** (0.26)	0.568 (0.004)	0.566 (0.041)	0.576 (0.014)	0.544 (0.024)	0.552 (0.017)	0.556 (0.022)
Avg. Weight Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.2$	5.99×10^{-5} (9.28×10^{-6})	**0.655** (4.10×10^{-3})	0.284 (6.47×10^{-3})	0.302 (6.68×10^{-3})	0.306 (0.014)	0.297 (5.42×10^{-3})	0.298 (8.63×10^{-3}	0.299 (7.74×10^{-3})
$\alpha = 0.4$	5.84×10^{-5} (7.95×10^{-6})	**0.710** (2.45×10^{-3})	0.489 (7.38×10^{-3})	0.510 (1.85×10^{-3})	0.502 (8.01×10^{-3})	0.507 (8.80×10^{-3})	0.510 (0.011)	0.505 (7.25×10^{-3})
$\alpha = 0.6$	6.06×10^{-5} (1.22×10^{-5})	**0.737** (2.13×10^{-3})	0.593 (5.67×10^{-3})	0.606 (5.41×10^{-3})	0.603 (7.61×10^{-3})	0.605 (5.46×10^{-3})	0.599 (0.012)	0.609 (6.96×10^{-3})
$\alpha = 0.8$	7.18×10^{-5} (6.24×10^{-6})	**0.755** (5.67×10^{-3})	0.661 (6.11×10^{-3})	0.660 (6.42×10^{-3})	0.663 (7.30×10^{-3})	0.661 (8.74×10^{-3})	0.665 (3.95×10^{-3})	0.661 (5.72×10^{-3})
$\alpha = 1.0$	6.90×10^{-5} (7.33×10^{-6})	**0.767** (2.92×10^{-3})	0.695 (5.08×10^{-3})	0.696 (4.68×10^{-3})	0.697 (2.38×10^{-4})	0.698 (6.51×10^{-3})	0.699 (4.27×10^{-3})	0.689 (9.47×10^{-3})
Avg. Neuron Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.2$	**0.472** (7.10×10^{-4})	0.299 (2.40×10^{-3})	0.153 (4.06×10^{-3})	0.160 (4.54×10^{-3})	0.164 (8.58×10^{-3})	0.158 (3.68×10^{-3})	0.158 (5.20×10^{-3})	0.159 (5.87×10^{-3})
$\alpha = 0.4$	**0.494** (1.01×10^{-3})	0.329 (2.10×10^{-3})	0.280 (5.64×10^{-3})	0.287 (7.55×10^{-4})	0.280 (6.57×10^{-3})	0.281 (5.05×10^{-3})	0.285 (8.48×10^{-3})	0.284 (7.22×10^{-3})
$\alpha = 0.6$	**0.506** (7.23×10^{-4})	0.343 (1.78×10^{-3})	0.351 (4.72×10^{-3})	0.354 (2.47×10^{-3})	0.35 (7.17×10^{-3})	0.352 (3.99×10^{-3})	0.347 (9.65×10^{-3})	0.353 (5.88×10^{-3})
$\alpha = 0.8$	**0.516** (6.72×10^{-4})	0.355 (8.23×10^{-3})	0.404 (6.20×10^{-3})	0.391 (4.66×10^{-3})	0.396 (7.60×10^{-3})	0.395 (9.59×10^{-3})	0.399 (3.89×10^{-3})	0.398 (6.39×10^{-3})
$\alpha = 1.0$	**0.526** (9.45×10^{-4})	0.361 (5.36×10^{-3})	0.432 (5.02×10^{-3})	0.424 (5.62×10^{-3})	0.427 (2.64×10^{-3})	0.427 (7.36×10^{-3})	0.430 (6.37×10^{-3})	0.417 (0.011)

Table 5. Average test error, weight sparsity, and neuron sparsity of 4-layer CNN models trained on MNIST with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.2$	0.916 (0.010)	0.452 (0.033)	0.440 (0.021)	**0.384** (0.015)	0.404 (0.019)	**0.384** (0.020)	0.392 (0.023)	0.398 (0.015)
$\alpha = 0.4$	1.414 (0.073)	0.448 (0.012)	0.456 (0.024)	0.414 (0.021)	0.426 (0.016)	0.426 (0.017)	0.428 (0.034)	**0.412** (0.012)
$\alpha = 0.6$	1.890 (0.033)	0.464 (0.022)	0.472 (0.013)	**0.434** (0.010)	0.460 (0.026)	0.440 (0.017)	0.452 (0.016)	0.454 (0.024)
$\alpha = 0.8$	1.966 (0.010)	**0.478** (0.007)	0.506 (0.014)	0.484 (0.019)	0.504 (0.015)	0.482 (0.019)	0.488 (0.016)	0.492 (0.007)
$\alpha = 1.0$	2.046 (0.019)	**0.492** (0.024)	0.530 (0.014)	0.514 (0.026)	0.520 (0.035)	0.506 (0.019)	0.514 (0.014)	**0.492** (0.016)

Avg. Weight Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.2$	5.86×10^{-5} (4.32×10^{-6})	**0.384** (0.112)	0.201 (0.005)	0.248 (0.012)	0.249 (0.017)	0.254 (0.013)	0.250 (0.013)	0.244 (0.006)
$\alpha = 0.4$	6.45×10^{-5} (9.15×10^{-6})	**0.541** (0.155)	0.424 (0.006)	0.467 (0.007)	0.449 (0.012)	0.466 (0.011)	0.460 0.020)	0.468 (0.015)
$\alpha = 0.6$	1.41×10^{-4} (1.74×10^{-5})	0.502 (0.157)	0.541 (0.010)	0.563 (0.016)	0.563 (0.016)	**0.568** (0.011)	0.559 (0.015)	0.565 (0.008)
$\alpha = 0.8$	1.39×10^{-4} (1.06×10^{-6})	0.576 (0.166)	0.619 (0.012)	0.620 (0.012)	0.625 (0.014)	0.624 (0.014)	**0.628** (0.007)	0.626 (0.012)
$\alpha = 1.0$	1.47×10^{-4} (7.84×10^{-6})	0.518 (0.169)	0.658 (0.010)	0.661 (0.007)	0.658 (0.007)	**0.664** (0.006)	0.659 (0.007)	0.653 (0.008)

Avg. Neuron Sparsity	ℓ_0	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 0.2$	**0.470** (5.97×10^{-4})	0.293 (2.61×10^{-3})	0.099 (3.77×10^{-3})	0.122 (7.25×10^{-3})	0.123 (9.71×10^{-3})	0.126 (8.39×10^{-3})	0.123 (7.86×10^{-3})	0.120 (4.93×10^{-3})
$\alpha = 0.4$	**0.494** (6.51×10^{-4})	0.328 (1.43×10^{-3})	0.224 (4.23×10^{-3})	0.243 (6.85×10^{-3})	0.231 (0.011)	0.241 (3.74×10^{-3})	0.238 (0.015)	0.249 (0.014)
$\alpha = 0.6$	0.198 (6.25×10^{-5})	**0.343** (4.82×10^{-3})	0.296 (9.94×10^{-3})	0.305 (0.013)	0.307 (0.014)	0.311 (6.32×10^{-3})	0.303 (0.010)	0.306 (9.24×10^{-3})
$\alpha = 0.8$	0.217 (2.03×10^{-5})	0.353 (3.37×10^{-3})	0.357 (0.012)	0.343 (0.015)	0.350 (0.011)	0.348 (0.013)	0.356 (4.78×10^{-3})	**0.358** (0.016)
$\alpha = 1.0$	0.229 (3.98×10^{-5})	0.359 (2.78×10^{-3})	**0.387** (0.010)	0.379 (3.75×10^{-3})	0.382 (5.85×10^{-3})	0.385 (6.37×10^{-3})	0.383 (4.66×10^{-3})	0.373 (9.97×10^{-3})

Table 6. Average test error, weight sparsity, and neuron sparsity of Resnet-40 models trained on CIFAR 10 with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 1.0$	6.932 (0.154)	**6.154** (0.199)	6.442 (0.065)	6.456 (0.176)	6.618 (0.128)	6.500 (0.158)	6.512 (0.126)
$\alpha = 1.5$	7.248 (0.145)	**6.504** (0.122)	6.850 (0.078)	7.108 (0.084)	6.948 (0.124)	6.958 (0.158)	6.820 (0.177)
$\alpha = 2.0$	7.306 (0.206)	**6.860** (0.174)	7.494 (0.092)	7.642 (0.176)	7.450 (0.192)	7.388 (0.140)	7.384 (0.122)
$\alpha = 2.5$	7.590 (0.148)	**7.298** (0.105)	7.760 (0.079)	8.146 (0.178)	8.026 (0.196)	8.096 (0.137)	7.968 (0.190)
$\alpha = 3.0$	7.672 (0.082)	**7.542** (0.135)	8.424 (0.081)	8.740 (0.166)	8.426 (0.192)	8.624 (0.083)	8.598 (0.144)

Avg. Weight Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 1.0$	**0.350** (0.009)	0.201 (0.018)	0.189 (0.007)	0.191 (0.008)	0.213 (0.015)	0.205 (0.015)	0.224 (0.016)
$\alpha = 1.5$	**0.371** (0.012)	0.322 (0.008)	0.345 (0.013)	0.313 (0.008)	0.354 (0.029)	0.330 (0.020)	0.343 (0.008)
$\alpha = 2.0$	0.385 (0.009)	0.431 (0.013)	0.457 (0.012)	0.422 (0.014)	**0.466** (0.015)	0.428 (0.013)	0.451 (0.012)
$\alpha = 2.5$	0.386 (0.010)	0.509 (0.017)	0.525 (0.010)	0.507 (0.011)	**0.534** (0.012)	0.522 (0.026)	0.537 (0.013)
$\alpha = 3.0$	0.401 (0.008)	0.551 (0.015)	**0.594** (0.009)	0.568 (0.009)	0.598 (0.012)	0.569 (0.014)	0.585 (0.006)

Avg. Neuron Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 1.0$	0.035 (0.003)	0.096 (0.011)	0.087 (0.004)	0.082 (0.005)	0.102 (0.008)	0.093 (0.010)	**0.105** (0.012)
$\alpha = 1.5$	0.040 (0.006)	0.154 (0.006)	0.159 (0.008)	0.144 (0.009)	**0.168** (0.013)	0.151 (0.009)	0.155 (0.004)
$\alpha = 2.0$	0.048 (0.004)	0.207 (0.005)	0.203 (0.008)	0.188 (0.006)	**0.217** (0.015)	0.195 (0.009)	0.209 (0.009)
$\alpha = 2.5$	0.045 (0.005)	**0.247** (0.010)	0.232 (0.010)	0.225 (0.017)	0.245 (0.011)	0.233 (0.008)	0.244 (0.006)
$\alpha = 3.0$	0.048 (0.007)	**0.274** (0.012)	0.271 (0.008)	0.249 (0.004)	0.272 (0.016)	0.259 (0.008)	0.268 (0.011)

Table 7. Average test error, weight sparsity, and neuron sparsity of Resnet-40 models trained on CIFAR 100 with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 2.0$	30.102 (0.234)	**28.636** (0.140)	29.260 (0.306)	29.610 (0.275)	29.044 (0.155)	29.316 (0.154)	29.274 (0.249)
$\alpha = 2.5$	30.326 (0.272)	**29.322** (0.144)	30.140 (0.180)	30.454 (0.295)	30.180 (0.175)	30.426 (0.253)	30.204 (0.159)
$\alpha = 3.0$	30.378 (0.154)	**29.750** (0.258)	31.134 (0.099)	31.482 (0.361)	31.048 (0.118)	31.164 (0.236)	31.108 (0.129)
$\alpha = 3.5$	30.666 (0.267)	**30.588** (0.285)	31.966 (0.260)	32.438 (0.272)	31.930 (0.156)	31.984 (0.182)	31.822 (0.365)
$\alpha = 4.0$	**30.982** (0.277)	31.436 (0.069)	33.106 (0.281)	33.210 (0.230)	32.758 (0.279)	33.240 (0.171)	33.094 (0.219)
Avg. Weight Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 2.0$	**0.286** (0.002)	0.129 (0.024)	0.182 (0.018)	0.164 (0.010)	0.198 (0.012)	0.162 (0.017)	0.187 (0.015)
$\alpha = 2.5$	**0.299** (0.005)	0.233 (0.010)	0.283 (0.005)	0.251 (0.021)	0.292 (0.010)	0.271 (0.015)	0.284 (0.016)
$\alpha = 3.0$	0.303 (0.003)	0.321 (0.008)	0.365 (0.009)	0.355 (0.018)	**0.377** (0.012)	0.363 (0.023)	0.372 (0.010)
$\alpha = 3.5$	0.306 (0.004)	0.409 (0.013)	0.441 (0.014)	0.418 (0.012)	**0.444** (0.014)	0.418 (0.016)	0.442 (0.006)
$\alpha = 4.0$	0.313 (0.010)	0.456 (0.014)	**0.511** (0.015)	0.461 (0.011)	0.501 (0.013)	0.480 (0.017)	0.507 (0.012)
Avg. Neuron Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	SGL_1-L_2
$\alpha = 2.0$	0.001 (0.001)	0.054 (0.007)	0.074 (0.007)	0.064 (0.008)	**0.083** (0.005)	0.063 (0.004)	0.078 (0.007)
$\alpha = 2.5$	0.003 (0.001)	0.092 (0.005)	0.113 (0.004)	0.093 (0.010)	**0.116** (0.005)	0.103 (0.004)	0.111 (0.005)
$\alpha = 3.0$	0.004 (0.001)	0.126 (0.004)	0.140 (0.005)	0.133 (0.007)	0.145 (0.003)	0.138 (0.009)	**0.146** (0.003)
$\alpha = 3.5$	0.002 (0.001)	0.157 (0.006)	0.166 (0.005)	0.158 (0.005)	**0.182** (0.017)	0.156 (0.004)	0.171 (0.005)
$\alpha = 4.0$	0.005 (0.002)	0.177 (0.007)	**0.195** (0.005)	0.176 (0.007)	0.193 (0.004)	0.180 (0.011)	0.193 (0.004)

Table 8. Average test error, weight sparsity, and neuron sparsity of WRN-28-10 models trained on CIFAR 10 with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.01$	**3.822** (0.054)	4.092 (0.159)	4.050 (0.058)	4.036 (0.074)	4.004 (0.104)	3.994 (0.039)	4.152 (0.089)
$\alpha = 0.05$	3.856 (0.089)	3.946 (0.106)	3.874 (0.029)	3.838 (0.067)	3.862 (0.076)	**3.812** (0.097)	3.872 (0.110)
$\alpha = 0.1$	4.000 (0.076)	3.960 (0.062)	**3.784** (0.082)	3.824 (0.088)	3.832 (0.047)	3.800 (0.082)	3.792 (0.113)
$\alpha = 0.2$	4.146 (0.092)	3.928 (0.115)	3.824 (0.034)	3.874 (0.093)	3.780 (0.096)	**3.764** (0.129)	3.962 (0.078)
$\alpha = 0.5$	4.524 (0.090)	4.486 (0.077)	4.444 (0.086)	4.408 (0.063)	4.448 (0.084)	**4.340** (0.115)	4.382 (0.068)

Avg. Weight Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.01$	**0.362** (0.016)	0.045 (0.001)	0.040 (0.002)	0.044 (0.002)	0.039 (0.002)	0.040 (0.001)	0.043 (0.001)
$\alpha = 0.05$	**0.464** (0.003)	0.117 (0.003)	0.145 (0.006)	0.156 (0.005)	0.145 (0.007)	0.145 (0.004)	0.161 (0.006)
$\alpha = 0.1$	**0.483** (0.003)	0.417 (0.005)	0.438 (0.004)	0.450 (0.005)	0.441 (0.005)	0.428 (0.004)	0.446 (0.013)
$\alpha = 0.2$	0.495 (0.003)	0.673 (0.002)	0.669 (0.005)	0.672 (0.003)	0.679 (0.003)	0.666 (0.004)	**0.688** (0.003)
$\alpha = 0.5$	0.503 (0.003)	**0.868** (0.001)	0.864 (0.002)	0.857 (0.001)	0.865 (0.001)	0.858 (0.002)	0.867 (0.001)

Avg. Neuron Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.01$	**0.033** (0.002)	0.018 (0.001)	0.015 (0.001)	0.018 (0.001)	0.014 (0.001)	0.015 (0.001)	0.017 (0.001)
$\alpha = 0.02$	0.050 (0.002)	0.056 (0.001)	0.068 (0.003)	0.074 (0.003)	0.069 (0.004)	0.069 (0.003)	**0.077** (0.002)
$\alpha = 0.1$	0.055 (0.002)	0.178 (0.002)	0.189 (0.002)	0.190 (0.002)	0.188 (0.002)	0.182 (0.003)	**0.191** (0.006)
$\alpha = 0.2$	0.059 (0.001)	0.297 (0.002)	0.294 (0.005)	0.293 (0.001)	0.299 (0.001)	0.289 (0.002)	**0.307** (0.003)
$\alpha = 0.5$	0.061 (0.001)	**0.440** (0.002)	0.434 (0.002)	0.428 (0.001)	0.435 (0.001)	0.429 (0.003)	0.436 (0.001)

Table 9. Average test error, weight sparsity, and neuron sparsity of WRN-28-10 models trained on CIFAR 100 with lowest test errors across 5 runs. Standard deviations are in parentheses.

Avg. Test Error (%)	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.01$	**18.696** (0.184)	19.792 (0.084)	19.494 (0.241)	19.498 (0.189)	19.368 (0.188)	19.474 (0.051)	19.632 (0.182)
$\alpha = 0.05$	**18.714** (0.203)	19.284 (0.134)	18.816 (0.141)	19.106 (0.277)	18.936 (0.085)	18.846 (0.082)	19.094 (0.272)
$\alpha = 0.1$	19.120 (0.387)	19.168 (0.067)	18.648 (0.268)	18.690 (0.181)	**18.446** (0.108)	18.680 (0.292)	18.724 (0.084)
$\alpha = 0.2$	20.298 (0.078)	18.902 (0.130)	18.440 (0.115)	18.694 (0.150)	18.502 (0.108)	**18.290** (0.107)	18.614 (0.326)
$\alpha = 0.5$	21.370 (0.259)	19.604 (0.107)	19.648 (0.203)	19.732 (0.147)	**19.488** (0.262)	19.552 (0.186)	19.732 (0.156)
Avg. Weight Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.01$	**0.281** (0.017)	0.013 (0.001)	0.011 (0.001)	0.013 (<0.001)	0.011 (0.001)	0.011 (0.001)	0.013 (0.001)
$\alpha = 0.05$	**0.412** (0.004)	0.014 (0.001)	0.015 (0.002)	0.017 (0.001)	0.014 (0.001)	0.015 (0.001)	0.018 (0.001)
$\alpha = 0.1$	**0.440** (0.013)	0.054 (0.002)	0.070 (0.003)	0.069 (0.001)	0.073 (0.002)	0.066 (0.002)	0.080 (0.001)
$\alpha = 0.2$	**0.458** (0.016)	0.332 (0.004)	0.356 (0.005)	0.346 (0.002)	0.355 (0.004)	0.345 (0.003)	0.361 (0.003)
$\alpha = 0.5$	0.478 (0.003)	0.697 (0.001)	0.693 (0.004)	0.685 (0.002)	**0.700** (0.002)	0.686 (0.001)	0.698 (0.002)
Avg. Neuron Sparsity	CGES	GL	SGL_1	SGL_0	$SGSCAD$	$SGTL_1$	$SGL_1 - L_2$
$\alpha = 0.01$	**0.008** (0.001)	0.002 (<0.001)	0.002 (<0.001)	0.003 (<0.001)	0.001 (<0.001)	0.002 (<0.001)	0.002 (<0.001)
$\alpha = 0.02$	**0.030** (0.001)	0.003 (<0.001)	0.005 (0.001)	0.006 (<0.001)	0.005 (0.001)	0.005 (0.001)	0.006 (<0.001)
$\alpha = 0.1$	0.037 (0.001)	0.033 (0.001)	0.044 (0.002)	0.041 (<0.001)	0.046 (0.001)	0.040 (0.001)	**0.050** (0.001)
$\alpha = 0.2$	0.043 (0.003)	0.153 (0.002)	0.157 (0.002)	0.150 (0.001)	0.157 (0.002)	0.148 (0.001)	**0.160** (0.001)
$\alpha = 0.5$	0.052 (0.001)	0.303 (0.001)	0.298 (0.001)	0.294 (0.004)	**0.304** (0.002)	0.293 (0.002)	0.303 (0.001)

Table 10. Average test error, weight sparsity, and neuron sparsity of SGL_1-regularized Lenet-5 models trained on MNIST after 200 epochs across 5 runs. The models are trained with different algorithms. Standard deviations are in parentheses. (SGD is stochastic gradient descent.)

Avg. Test Error (%)	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.758 (0.029)	1.306 (0.031)	**0.722** (0.028)
$\alpha = 0.2$	0.760 (0.006)	2.954 (0.051)	**0.704** (0.031)
$\alpha = 0.3$	0.798 (0.023)	4.992 (0.161)	**0.732** (0.045)
$\alpha = 0.4$	0.836 (0.034)	7.304 (0.147)	**0.792** (0.034)
$\alpha = 0.5$	0.772 (0.019)	9.610 (0.170)	**0.720** (0.039)
Avg. Weight Sparsity	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.935 (0.001)	**0.994** (<0.001)	0.889 (0.004)
$\alpha = 0.2$	0.951 (0.002)	**0.997** (<0.001)	0.926 (0.001)
$\alpha = 0.3$	0.960 (<0.001)	**0.998** (<0.001)	0.945 (0.001)
$\alpha = 0.4$	0.963 (0.001)	**0.998** (<0.001)	0.952 (0.001)
$\alpha = 0.5$	0.966 (0.001)	**0.998** (<0.001)	0.954 (0.002)
Avg. Neuron Sparsity	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.735 (0.003)	**0.784** (0.004)	0.691 (0.007)
$\alpha = 0.2$	0.778 (0.004)	**0.902** (0.005)	0.754 (0.003)
$\alpha = 0.3$	0.802 (0.001)	**0.960** (0.002)	0.787 (0.003)
$\alpha = 0.4$	0.813 (0.003)	**0.972** (0.001)	0.805 (0.004)
$\alpha = 0.5$	0.821 (0.004)	**0.976** (0.002)	0.811 (0.004)

Table 11. Average test error, weight sparsity, and neuron sparsity of SGL_1-regularized Lenet-5 models trained on MNIST with lowest test errors across 5 runs. The models are trained with different algorithms. Standard deviations are in parentheses. (SGD is stochastic gradient descent.)

Avg. Test Error (%)	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.594 (0.032)	1.152 (0.026)	**0.568** (0.021)
$\alpha = 0.2$	0.634 (0.031)	2.320 (0.042)	**0.582** (0.035)
$\alpha = 0.3$	0.692 (0.028)	3.360 (0.075)	**0.600** (0.030)
$\alpha = 0.4$	0.684 (0.014)	4.272 (0.051)	**0.652** (0.025)
$\alpha = 0.5$	0.636 (0.022)	5.020 (0.094)	**0.616** (0.052)

Avg. Weight Sparsity	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.449 (0.172)	**0.939** (0.011)	0.757 (0.015)
$\alpha = 0.2$	0.531 (0.012)	**0.971** (0.005)	0.845 (0.005)
$\alpha = 0.3$	0.451 (0.217)	**0.992** (<0.001)	0.886 (0.004)
$\alpha = 0.4$	0.449 (0.213)	**0.989** (0.005)	0.896 (0.004)
$\alpha = 0.5$	0.559 (0.007)	**0.994** (<0.001)	0.905 (0.003)

Avg. Neuron Sparsity	direct SGD	proximal SGD	proposed
$\alpha = 0.1$	0.317 (0.139)	**0.698** (0.024)	0.497 (0.014)
$\alpha = 0.2$	0.444 (0.015)	**0.743** (0.021)	0.627 (0.011)
$\alpha = 0.3$	0.382 (0.185)	**0.863** (0.003)	0.697 (0.010)
$\alpha = 0.4$	0.399 (0.196)	**0.828** (0.061)	0.721 (0.008)
$\alpha = 0.5$	0.519 (0.013)	**0.883** (0.003)	0.735 (0.004)