Structured Sparsity of Convolutional Neural Networks via Nonconvex Sparse Group Regularization

Kevin Bui¹, Fredrick Park², Shuai Zhang¹, Yingyong Qi¹ and Jack Xin¹∗

¹Department of Mathematics, University of California, Irvine, Irvine, CA, United States, ²Department of Mathematics and Computer Science, Whittier College, Whittier, CA, United States

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 \( \ell_1 \), \( \ell_1 - \ell_2 \), and \( \ell_0 \)) 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 [28, 39, 77, 95], semantic segmentation [13, 49], and object detection [30, 72, 73]. 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 [6, 17] 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 [25, 54, 66, 81] 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 Refs. 14 and 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 \( \ell_1 \) or \( \ell_2 \) 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 sparsity [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 \( \ell_0 \) regularization on CNNs. Ma et al. [61] proposed integrated transformed \( \ell_1 \), a convex combination of transformed \( \ell_1 \) 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 \( \ell_1 \) penalty term with a nonconvex penalty term. The nonconvex penalty terms considered are \( \ell_0, \alpha \ell_2 \), transformed \( \ell_1 \), and SCAD. The proposed family is supposed to yield a more accurate and/or more compressed network than sparse group lasso since \( \ell_1 \) suffers various weaknesses due to being a convex relaxation of \( \ell_0 \). 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:

\[
\min_{W} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(h(x_i, W), y_i) + \lambda R(W),
\]

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.
- \( R(\cdot) \) is the regularizer on the set of weight parameters \( W \).
- \( \lambda > 0 \) is a regularization parameter for \( R(\cdot) \).

The most common regularizer used for DNNs is \( \ell_2 \) regularization \( \| \cdot \|_2 \), 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 \( \ell_1 \) norm \( \| \cdot \|_1 \), also known as the lasso penalty [78]. The \( \ell_1 \) norm is the tightest convex relaxation of the \( \ell_0 \) 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 \( \ell_1 \) norm’s ability to reconstruct sparse solution in compressed sensing. When a sensing matrix satisfies the restricted isometry property, the \( \ell_1 \) 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 \( \ell_1 \) minimization to guarantee exact recovery of sparse solutions [16, 23]. Being able to yield sparse solutions, the \( \ell_1 \) norm has gained popularity in other types of inverse problems such as compressed imaging [33, 57] and image segmentation [34, 35, 42] and in various fields of applications such as geoscience [74], medical imaging [33, 57], machine learning [10, 36, 67, 78, 89], and traffic flow network [91]. Unfortunately, element-wise sparsity by \( \ell_1 \) or \( \ell_2 \) 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],...
The regularization for exclusive sparsity is

\[ R_{GL}(W_l) = \sum_{g=1}^{N_l} \sqrt{\#w_{lg}} \|w_{lg}\|_2 \]  

(2)

where \( w_{lg} \) corresponds to the weight parameter with index \( i \) in group \( g \) in layer \( l \) and the term \( \#w_{lg} \) denotes the number of weight parameters in group \( g \) in layer \( l \). Because group sizes vary, the constant \( \sqrt{\#w_{lg}} \) is multiplied in order to rescale the \( \ell_2 \) norm of each group with respect to the group size, ensuring that each group is weighed uniformly [65, 76, 93].

The group lasso regularizer imposes the \( \ell_2 \) 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 \( \ell_2 \) norms are negligible, resulting in a highly compact network compared to element-wise sparsity.

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

\[ \frac{1}{2} \sum_{g=1}^{N_l} \|w_{lg}\|_1^2 = \frac{1}{2} \sum_{g=1}^{N_l} \left( \sum_{i=1}^N \#w_{lg,i} \right)^2 . \]  

(3)

Now, within each group, sparsity is enforced. Because exclusivity cannot guarantee the optimal features since some features do not 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

\[ R_{CGES} = \sum_{g=1}^{N_l} \left( 1 - \mu_i \right) \sum_{i=1}^N \#w_{lg,i} + \mu_i \left( \frac{1}{2} \sum_{g=1}^{N_l} \|w_{lg}\|_1 \right)^2 . \]  

(4)

where \( \mu_i \in (0, 1) \) 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

\[ R_{SGL}(W_l) = R_{GL}(W_l) + \|W_l\|_1 \]  

(5)

where

\[ \|W_l\|_1 = \sum_{g=1}^{N_l} \sum_{i=1}^N |w_{lg,i}| . \]

Sparse group lasso simultaneously enforces group sparsity by having the regularizer \( R_{GL}(\cdot) \) and element-wise sparsity by having the \( \ell_1 \) 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 zeroed 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 zeroed out. Combined with \( \ell_1 \) regularization on the individual weights, sparse group lasso allows for more weights in the remaining two neurons to be pruned.

2.2 Nonconvex Sparse Group Lasso

We recall that the \( \ell_1 \) norm is the tightest convex relaxation of the \( \ell_0 \) penalty, given by

\[ \|W_l\|_0 = \sum_{g=1}^{N_l} \sum_{i=1}^N |w_{lg,i}|_0 \]  

(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 \( \ell_0 \) penalty is non-convex and discontinuous. In addition, any \( \ell_0 \)-regularized problem is NP-hard [23]. These properties make developing convergent and tractable algorithms for \( \ell_0 \)-regularized problems difficult, thereby making \( \ell_1 \)-regularized problems better alternatives to solve. However, the \( \ell_0 \)-regularized problems have been shown to recover better solutions in terms of sparsity and/or accuracy than do \( \ell_1 \)-regularized problems in various applications, such as compressed sensing [56], image restoration [8, 12, 19, 55, 102], MRI reconstruction [80], and machine learning [56, 94]. In particular, \( \ell_0 \)-regularized inverse problems were demonstrated to be more robust against Poisson noise than are \( \ell_1 \)-regularized inverse problems [100].

A continuous alternative to the \( \ell_0 \) penalty is the SCAD penalty term [22, 58], given by

\[ \lambda \|W_l\|_{SCAD(a)} = \sum_{g=1}^{N_l} \sum_{i=1}^N \lambda |w_{lg,i}|_{SCAD(a)} \]  

(7)

where

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

for \( \lambda > 0 \) and \( a > 2 \). This penalty term enjoys three properties—unbiasedness, sparsity, and continuity—while the \( \ell_1 \) norm, on the other hand, has only sparsity and continuity [22]. In linear and logistic regression, SCAD was shown to outperform \( \ell_1 \) in variable selection [22]. SCAD has been applied to wavelet approximation [5], bioinformatics [9, 84], and compressed sensing [64].
The transformed $\ell_1$ 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
\[
||W||_{\ell_1} = \sum_{g=1}^{N_l} \sum_{i=1}^{\#} |w_{g,i}|_{\ell_1(a)}
\]
where
\[
|w|_{\ell_1(a)} = \frac{(a + 1)|w|}{a + |w|}.
\]
In addition, it interpolates the $\ell_0$ and $\ell_1$ penalties through the parameter $a$ \[98\] because
\[
\lim_{a \to 0} |w|_{\ell_1(a)} = |w|_0 \quad \text{and} \quad \lim_{a \to \infty} |w|_{\ell_1(a)} = |w|.
\]
The transformed $\ell_1$ penalty term was investigated and was shown to outperform $\ell_1$ in compressed sensing \[79, 97, 98\], deep learning \[45, 61, 87\], matrix completion \[99\], and epidemic forecasting \[45\].

Another Lipschitz continuous, nonconvex regularizer is the $\ell_1 - a\ell_2$ penalty given by
\[
||W||_{\ell_1 - a\ell_2} = ||W||_1 - a||W||_2
\]
where $a \in (0, 1]$. In a series of works \[50-52, 90\], the penalty term $\ell_1 - a\ell_2$ with $a = 1$ yields better solutions than does $\ell_1$ 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, $\ell_1 - a\ell_2$ only requires a relaxed variant of the null space property \[79\]. Furthermore, $\ell_1 - a\ell_2$ is more robust against impulsive noise in yielding sparse, accurate solutions for inverse problems than is $\ell_1$ \[44\].

Besides compressed sensing, it has been utilized in image

\[\text{FIGURE 1} \quad \text{Comparison between lasso, group lasso, and sparse group lasso applied to a weight matrix. Entries in white are zero’ed out or removed; entries in gray remain.}\]
denoising and deblurring [53], image segmentation [71], image inpainting [63], and hyperspectral demixing [21]. In deep learning applications, the $\ell_1 - \ell_2$ regularization was used to learn permutation matrices [59] for ShuffleNet [60, 101].

Due to the advantages and recent successes of the aforementioned nonconvex regularizers, we propose to replace the $\ell_1$ norm in Eq. 5 with nonconvex penalty terms. Hence, we propose a family of group regularizers called nonconvex sparse group lasso. The family includes the following:

\[
R_{SGL} (W_l) = R_{GL} (W_l) + ||W_l||^2
\]

(10)

\[
R_{SGSCL} (W_l) = R_{GL} (W_l) + ||W_l||_{SCAD(a)}
\]

(11)

\[
R_{SGTL} (W_l) = R_{GL} (W_l) + ||W_l||_{TL(a)}
\]

(12)

\[
R_{SGL1-ad} (W_l) = R_{GL} (W_l) + ||W_l||_{1-ad}
\]

(13)

Using these regularizers, we expect to obtain a sparser and/or more accurate network than from using the original sparse group lasso. The $\ell_1$ norm can also be replaced with other nonconvex penalties not mentioned in this paper. Refer to Refs. 3 and 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 $(V, W) := (\{V_l\}_{l=1}^L, \{W_l\}_{l=1}^L) = (V_1, \ldots, V_L, W_1, \ldots, W_L)$
- $V^* := V_{K+1}^*$
- $\tilde{L}(W) := \sum_{i=1}^{N} \mathcal{L}(h(x_i, W), y_i)$

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

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

for $\lambda > 0$.

2.4 Numerical Optimization

We develop a general algorithm framework to solve

\[
\min_{W} \tilde{L}(W) + \lambda \sum_{l=1}^{L} \mathcal{R} (W_l) = \tilde{L}(W) + \sum_{l=1}^{L} [\lambda R_{GL} (W_l) + \lambda r (W_l)]
\]

(14)

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

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

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

\[
\min_{V, W} \tilde{L}(W) + \sum_{l=1}^{L} (\lambda R_{GL} (W_l) + \lambda r (V_l))
\]

s.t. $V_l = W_l$ \quad $l = 1, \ldots, L$

(15)

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

\[
\min_{V, W} F_{\beta} (V, W) := \tilde{L}(W) + \sum_{l=1}^{L} [\lambda R_{GL} (W_l) + \lambda r (V_l) + \frac{\beta}{2} ||V_l - W_l||^2]
\]

(16)

With $\beta$ fixed, Eq. 16 can be solved by alternating minimization:

\[
W^{k+1} = \arg \min_{W} F_{\beta} (V^k, W)
\]

(17a)

\[
V^{k+1} = \arg \min_{V} F_{\beta} (V, W^{k+1})
\]

(17b)

To solve Eq. 17a, we simultaneously update $W_l$ for $l = 1, \ldots, L$ by gradient descent

\[
W^{k+1}_l = W^k_l - \gamma \left[ \nabla_{W_l} \tilde{L}(W^k) + \lambda \partial_{W_l} R_{GL} (W^k_l) - \beta (V^k_l - W^k_l) \right]
\]

(18)

where $\gamma > 0$ is the learning rate and $\partial_{W_l} R_{GL}$ is the subdifferential of $R_{GL}$ with respect to $W_l$. In practice, Eq. 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 Eq. 17b can be rewritten as

\[
V^{k+1} = \arg \min_{V} \sum_{l=1}^{L} \left( \frac{\lambda}{\beta} r(V_l) + \frac{1}{2} ||V_l - W_l||^2 \right)
\]

(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.

Incorporating the algorithm that solves the quadratic penalty problem Eq. 16, we now develop a general algorithm to solve Eq. 14. We solve a sequence of quadratic penalty problems Eq. 16 with $\beta \in [\beta_0, \infty)$ where $\beta_0 > 0$. This will yield a sequence $\{(V, W)_{\beta}^k\}_{\beta=1}^{\infty}$ so that $W^k \rightarrow W^*$, 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 Eq. 14 is proximal gradient descent [70]. By this method, the update for $W_l$, $l = 1, \ldots, L$, is
\[ \lambda \| \mathbf{x} \|_{\ell_1} = \lambda \sum_{i=1}^{n} |x_i| \]

| Regularizer Name | Penalty Formulation | Proximal Operator |
|------------------|---------------------|------------------|
| \( \ell_1 \)     | \( \lambda \| \mathbf{x} \|_{\ell_1} = \lambda \sum_{i=1}^{n} |x_i| \) | \( \text{prox}_{\lambda \| \cdot \|_{\ell_1}}(\mathbf{x}) = [S_1(x_1), \ldots, S_1(x_n)] \) |
| \( \ell_0 \)     | \( \lambda \| \mathbf{x} \|_{\ell_0} = \lambda \sum_{i=1}^{n} |x_i| \) | \( \text{prox}_{\lambda \| \cdot \|_{\ell_0}}(\mathbf{x}) = [S_0(x_1), \ldots, S_0(x_n)] \) |
| SCAD(\( a \))   | \( \lambda \| \mathbf{x} \|_{\text{SCAD}(a)} = \sum_{i=1}^{n} \lambda \| x_i \|_{\text{SCAD}(a)} \) | \( \text{prox}_{\lambda \| \cdot \|_{\text{SCAD}(a)}}(\mathbf{x}) = (T_1(x_1), \ldots, T_n(x_n)) \) |
| TL1(\( a \))    | \( \lambda \| \mathbf{x} \|_{\text{TL1}(a)} = \lambda \sum_{i=1}^{n} \| x_i \|_{\text{TL1}(a)} \) | \( \text{prox}_{\lambda \| \cdot \|_{\text{TL1}(a)}}(\mathbf{x}) = (T_1(x_1), \ldots, T_n(x_n)) \) |

Where
\( g_{a\lambda}(t) = \frac{2}{3} a + \frac{t}{3} \cos \left( \frac{\phi_{a\lambda}(0)}{3} - \frac{2a}{3} + \frac{|t|}{3} \right) \)
\( \phi_{a\lambda}(0) = \frac{2\pi}{3} a (a + 1) \)
\( \rho(a, \lambda) = \begin{cases} \sqrt{2a(a+1)} - \frac{a}{2} & \text{if } \lambda > \frac{a^2}{2(a+1)} \\ \frac{a + 1}{a} & \text{if } \lambda \leq \frac{a^2}{2(a+1)} \end{cases} \)

and
\( \text{prox}_{\lambda \| \cdot \|_{\ell_1}}(\mathbf{x}) = \left[ \begin{array}{c} \frac{|x_1| + \lambda}{|x_1|} z_1 \\ \frac{|x_2| + \lambda}{|x_2|} z_2 \\ \vdots \\ \frac{|x_n| + \lambda}{|x_n|} z_n \end{array} \right] \)

with \( z_1 = S_1(x) \) and
\( z_k = \begin{cases} 0 & \text{if } i \neq k \\ \text{sign}(x_i) |x_i|_{\text{\iota}} & \text{if } i = k \end{cases} \)

where \( k = \arg \min_{1 \leq k \leq n} |x_k| = |x_{\text{\iota}}| \).
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
   4. Set $k := 1$.
   5. Set $V^j, 1 = W^j$ and $V^j, 1 = V^j$.
   6. while stopping criterion for inner loop not satisfied
      7. Update $W^{j,k+1}$ by Eq. (16).
      8. Update $V^{j,k+1}$ by Eq. (17).
      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 $V^j$.

\[ W^{k+1}_i = \text{prox}_{\lambda L_i} \left[ W^k_i - \gamma \left( \nabla_W L_i(W^k) + \lambda \partial W_i R_{GL}(W^k_i) \right) \right]. \] (20)

Using this algorithm results in weights parameters with some already zeroed out.

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

\[ W^{k+1}_i = \text{arg} \min_{W_i} \tilde{L}(W) + R_{GL}(W) + \frac{\beta}{2} \| V_i - W_i \|^2_2 \]

\[ = \text{arg} \min_{W_i} \tilde{L}(W) + R_{GL}(W) + \frac{\beta}{2} \sum_{j=1}^{\#(W_i)} (V_i - W_i)^2. \] (21)

We see that this step performs exact weight decay or $\ell_2$ regularization on weights $W_i$ whenever $V_i = 0$. On the other hand, when $V_i \neq 0$, the effect of $\ell_2$ regularization is mitigated on the corresponding weight $W_i$ based on the absolute difference $|V_i - W_i|$. Using $\ell_2$ 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 $\ell_2$ regularization method, where Eq. 17b identifies which weights to perform exact $\ell_2$ regularization on and Eq. 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 Eqs 17a and 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 Eq. 15 because the loss function $L_{\ell_2}(\cdot)$ is nonconvex. However, it was shown in [18] that a similar algorithm to Algorithm 1, but for fixed $\beta$ 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 Eqs. 17a and 17b, where $r(\cdot)$ is $\ell_0, \ell_1, \ell_1 - a\ell_2$, 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 Eq. 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 Eq. 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

\[ M \geq \max \left\{ \tilde{L}(W^{\text{feas}}) + \lambda \sum_{k=1}^{n} R(W_i^{\text{feas}}), \min_{W} F_{\beta_k}(V^k, W) \right\}. \]

If $\min_{W} F_{\beta_k}(V^k, W) > M$, then we set $V^{k+1} = W^{\text{feas}}$. This strategy is based on Ref. 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), CGES proposed in Ref. 92, and the group variant of $\ell_0$ regularization (denoted as $\ell_0$ for simplicity) proposed in Ref. 54. SGL is optimized using the same algorithm proposed for...
Nonconvex Sparse Group Regularization of CNN

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 (α), we set α = 3.7 as suggested in [22]; for SGTL (α), we set α = 1.0 as suggested in Ref. 99; and for SGL (α), we set α = 1.0 as suggested by the literatures [50–52, 90]. For CGES, we have μ/L = U. 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^−3, 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: \( \lambda = \mu/60000 \) for \( \alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\} \). For SGL and nonconvex sparse group lasso, we set \( \beta = 2\mu/60000 \), and for every 40 epochs, \( \beta \) increases by a factor of \( \alpha = 1.25 \). 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, SGL0 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 \( \alpha \)’s tested. Generally, SGL and nonconvex sparse group lasso outperform \( \ell_0 \) nonconvex sparse group lasso.
### 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.

| $\ell_0$ | CGES | GL | SGL | SGL0 | SGSCAD | SGTL | SGL1 | SGL1 − $L_2$ |
|---------|------|----|-----|------|--------|------|------|-------------|
| $\alpha = 0.1$ | 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 (0.026) | 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) | 0.576 (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) |

### 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.

| $\ell_0$ | CGES | GL | SGL | SGL0 | SGSCAD | SGTL | SGL1 | SGL1 − $L_2$ |
|---------|------|----|-----|------|--------|------|------|-------------|
| $\alpha = 0.1$ | 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 (9.90×10^{-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 (6.80×10^{-3}) |
| $\alpha = 0.4$ | 0.614 (1.71×10^{-3}) | 0.482 (0.020) | 0.586 (0.164) | 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 5 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 SGLα at \( \alpha = 0.3, 0.5 \); SGL1 - L2 at \( \alpha = 0.2 \); and CGES at \( \alpha = 0.1, 0.4 \). For average weight sparsity, SGLα 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 SGLL1 at \( \alpha = 0.2 ; \) by SGL1 at \( \alpha = 0.3 \); and by SGL0 at \( \alpha = 0.4, 0.5 \). Although SGLα 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 \( \ell_0 \) 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 SGL1 and nonconvex sparse group lasso, we set \( \beta = 5\alpha/60000 \) and for every 40 epochs, \( \beta \) increases by a factor of 6. 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, SGL1, and nonconvex group lasso when \( \alpha \geq 0.6 \). \( \ell_0 \) has the highest neuron sparsity for all \( \alpha \)’s given, but its test errors are much greater. When \( \alpha \leq 0.6 \), SGLSCAD yields the most accurate models at \( \alpha = 0.2, 0.6 \) while SGL1 yields one at \( \alpha = 0.4 \). Overall, we see that nonconvex group lasso has comparable weight sparsity and neuron sparsity as group lasso and SGL1. 

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 \( \alpha = 0.2 \), SGL1 and SGLSCAD have the lowest test errors, but their weight sparsity are exceeded by CGES and their neuron sparsity are exceeded by \( \ell_0 \). At \( \alpha = 0.4 \), SGL1 - L2 has the lowest test error, but its weight sparsity and neuron sparsity are exceeded by CGES and \( \ell_0 \) respectively. At \( \alpha = 0.6 \), SGL1 has the lowest test error, but SGLSCAD 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, SGL1, 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, SGL1, and the nonconvex group lasso regularizers. At \( \alpha = 1.0 \), group lasso has the highest neuron sparsity, but nonconvex group

regularization proposed by Louizos et al. [54] and group lasso by average weight and neuron sparsity.

### Table 5: Test Error, Weight Sparsity, and Neuron Sparsity of 4-Layer CNN Models

| \( \alpha \) | Test Error (\%) | Weight Sparsity | Neuron Sparsity |
|---|---|---|---|
| 0.2 | 0.9160 (0.0110) | 0.4520 (0.0323) | 0.0160 (0.0014) |
| 0.4 | 1.1410 (0.0120) | 0.4560 (0.0322) | 0.0430 (0.0012) |
| 0.6 | 1.2860 (0.0128) | 0.4650 (0.0310) | 0.0630 (0.0014) |
| 0.8 | 1.4850 (0.0120) | 0.4760 (0.0310) | 0.1200 (0.0014) |
| 1.0 | 1.7840 (0.0120) | 0.4860 (0.0301) | 0.1800 (0.0014) |

---

Frontiers in Applied Mathematics and Statistics | www.frontiersin.org 10 February 2021 | Volume 6 | Article 529564
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     | SGL1   | SGL2   | SGSCAD  | SGLT1   | SGL1−L2 |
|---------------------|--------|--------|--------|--------|---------|---------|---------|
| \(\alpha = 1.0\)    | 6.932 (0.154) | 6.154 (0.199) | 6.442 (0.065) | 6.456 (0.178) | 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 (0.178) | 7.450 (0.192) | 7.386 (0.143) | 7.284 (0.122) |
| \(\alpha = 2.5\)    | 7.590 (0.148) | 7.296 (0.105) | 7.760 (0.079) | 8.146 (0.178) | 8.226 (0.196) | 8.066 (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) |

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     | SGL1   | SGL2   | SGSCAD  | SGLT1   | SGL1−L2 |
|---------------------|--------|--------|--------|--------|---------|---------|---------|
| \(\alpha = 1.0\)    | 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) | 0.300 (0.020) | 0.343 (0.008) |
| \(\alpha = 2.0\)    | 0.395 (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) |
| \(\alpha = 2.5\)    | 0.388 (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) | 0.515 (0.011) | 0.594 (0.009) | 0.568 (0.009) | 0.598 (0.012) | 0.569 (0.014) | 0.585 (0.006) |

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, \(\beta\) increases by a factor of \(\alpha = 1.25\). The networks are
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 | SGL1 | SGL2 | SGSCAD | SGLT1 | SGLT2 - L2 |
|---------------------|------|----|------|------|--------|-------|-----------|
| α = 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 (0.069) |
| α = 0.05            | 3.856 (0.089) | 3.946 (0.106) | 3.874 (0.029) | 3.838 (0.067) | 3.862 (0.078) | 3.812 (0.097) | 3.872 (0.110) |
| α = 0.1             | 4.000 (0.076) | 3.960 (0.062) | 3.784 (0.082) | 3.824 (0.086) | 3.832 (0.047) | 3.800 (0.062) | 3.792 (0.113) |
| α = 0.2             | 4.146 (0.052) | 3.928 (0.115) | 3.824 (0.054) | 3.874 (0.093) | 3.789 (0.085) | 3.764 (0.129) | 3.962 (0.078) |
| α = 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) |

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 | SGL1 | SGL2 | SGSCAD | SGLT1 | SGLT2 - L2 |
|---------------------|------|----|------|------|--------|-------|-----------|
| α = 0.01            | 18.696 (0.184) | 19.792 (0.064) | 19.494 (0.241) | 19.498 (0.189) | 19.536 (0.188) | 19.474 (0.051) | 19.632 (0.182) |
| α = 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) |
| α = 0.1             | 19.120 (0.387) | 19.168 (0.067) | 18.648 (0.269) | 18.690 (0.181) | 18.446 (0.108) | 18.689 (0.292) | 18.724 (0.084) |
| α = 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 (0.026) |
| α = 0.5             | 21.370 (0.259) | 19.604 (0.107) | 19.648 (0.203) | 19.732 (0.147) | 19.488 (0.262) | 19.552 (0.168) | 19.732 (0.156) |

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, SGL1, and nonconvex sparse group lasso are higher by at most 1.1%. When α ≤ 1.5, CGES has the largest weight sparsity while SGSCAD, SGLT1, SGL1 − SGL2 have larger weight sparsity than does group lasso. At α = 2.0, 2.5, SGSCAD has the largest weight sparsity. At α = 3.0, SGL1 has the largest weight sparsity with comparable test error as the nonconvex group lasso regularizers. For neuron sparsity, SGL1 − L2 has the largest at α = 1.0 while SGSCAD has the largest at α = 1.5, 2.0. However, at α = 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 SGL1 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 trained for 200 epochs across 5 runs. We excluded ℓ0 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.
The models are trained with different algorithms. Standard deviations are in parentheses. (SGD is stochastic gradient descent).

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 SGTL1 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 $\alpha$. 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 $\alpha$ increases. When $\alpha = 0.5$, the best weight sparsity is attained by $SGSCAD$, but the other methods have comparable weight sparsity.
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 $\alpha$ 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 among 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

| Table 11 | Average test error, weight sparsity, and neuron sparsity of SGL1-regularized Lenet-5 models trained on MNIST with lowest test errors across 5 runs. |
|------------|------------------|------------------|------------------|
| Avg. Test Error (%) | direct SGD | proximal SGD | proposed |
| $\alpha = 0.1$ | 0.594 (0.032) | 1.152 (0.028) | 0.568 (0.021) |
| $\alpha = 0.2$ | 0.634 (0.031) | 2.320 (0.042) | 0.582 (0.039) |
| $\alpha = 0.3$ | 0.692 (0.028) | 3.960 (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) |

The models are trained with different algorithms. Standard deviations are in parentheses. (SGD is stochastic gradient descent).

---

**FIGURE 3** Mean results of algorithms applied to SGL1 for Lenet-5 models trained on MNIST with lowest test errors across 5 runs when varying the regularization parameter $\lambda = 160000$ 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.
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 $\ell_1$ 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. 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 SGL1 yields a satisfactorily sparse network with lower test error than do the other two algorithms.

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 matroid 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 Eqs 17a and 17b, for each $k \in \mathbb{N}$, we have

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

for all $W$, and

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

for all $V$. By Eq. 23, we have

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

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

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

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 $\lim_{k \to \infty} F_{\beta}(V^k, W^k)$ exists. From Eqs. 22–24, we have

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

Taking the limit gives us

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

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

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

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

$$F_{\beta}(V, W^k) \geq F_{\beta}(V, W^k)$$

$$\geq \tilde{C}(W) + \sum_{l=1}^{L} \lambda(R_{\text{calc}}(W_l) + r(V_l)) + \frac{\beta}{2} \|W_l - W_{l}^k\|_2^2$$

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

$$F_{\beta}(V, W^k) \geq \tilde{C}(W) + \sum_{l=1}^{L} \lambda(R_{\text{calc}}(W_l) + r(V_l)) + \frac{\beta}{2} \|W_l - W_{l}^k\|_2^2$$

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

For notational convenience, let

$$R_{\beta}(V, W) := \sum_{i=1}^{L} \lambda R_{\text{calc}}(W_i) + \frac{\beta}{2} \|V_i - W_i\|_2.$$

By Eq. 22, we have

$$\tilde{C}(W) + \tilde{R}_{\beta}(V^k, W) = F_{\beta}(V^k, W) - \lambda \sum_{i=1}^{L} r(V_i)$$

$$\geq F_{\beta}(V^k, W^k) - \lambda \sum_{i=1}^{L} r(V_i^*) = \tilde{C}(W^*) + \tilde{R}_{\beta}(V^k, W^k).$$

Because $\lim_{k \to \infty} V^k$ exists, the sequence $(V^k)_{k=1}^{\infty}$ is bounded. If $r(\cdot)$ is $\ell_0$, transformed $\ell_1$, or SCAD, then $r(V^k)_{k=1}^{\infty}$ is bounded. If $r(\cdot)$ is $\ell_1$, then $r(\cdot)$ is coercive. If $r(\cdot)$ is $\ell_1 - a\ell_2$, then $r(\cdot)$ is
bounded above by $\ell_1$. Overall, this follows that $\{r(V^k)\}_{k \in K}$ bounded as well. Hence, there exists a further subsequence $K \subset K$ such that $\lim_{k \to \infty} r(V^k)$ exists. So, we obtain

$$\lim_{k \to \infty} \tilde{L}(W^*) + \tilde{R}_{\lambda, \beta}(V^k, W^*) = \lim_{k \to \infty} F_{\beta}(V^k, W^*) - \lambda \sum_{i=1}^{L} r(V^k_i)$$

$$= \lim_{k \to \infty} F_{\beta}(V^k, W^*) - \lambda \sum_{i=1}^{L} r(V^*)$$

$$= \tilde{L}(W^*) + \tilde{R}_{\lambda, \beta}(V^k, W^*)$$

(31)

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

Taking the limit over the subsequence $K$ in Eq. 30 and applying Eq. 31, we obtain

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

(32)

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

$$F_{\beta}(V^*, W) \geq F_{\beta}(V^*, V^*),$$

(33)

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

5.2 Proof of Theorem 3
Because $(V^*, W^*)$ is an accumulation point, there exists a subsequence $K$ such that $\lim_{k \to \infty} (V^k, W^k) = (V^*, W^*)$. If $\{F_{\beta}(V^k, W^k)\}_{k \to \infty}$ is uniformly bounded, there exists $M$ such that $F_{\beta}(V^k, W^k) \leq M$ for all $k \in \mathbb{N}$. Then we have

$$M \geq F_{\beta}(V^k, W^k) = \tilde{L}(W) + \sum_{i=1}^{L} \lambda R_{\text{cl}}(W_i) + \lambda r(V^*_i)$$

$$+ \frac{\beta}{2} \|V_j - W_j\|_2^2 \geq \frac{\beta}{2} \sum_{i=1}^{L} \|V_j - W_j\|_2^2$$

As a result,

$$\sum_{j=1}^{L} \|V^k_j - W_j\|_2^2 \leq \frac{2}{\beta} M.$$  \hspace{1cm} (34)

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

$$\sum_{j=1}^{L} \|V^*_j - W_j\|_2^2 = 0,$$

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

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.

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, DMS-1952644 and the Qualcomm Faculty Award.

ACKNOWLEDGMENTS

The authors would like to thank 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.

REFERENCES

1. Aghasi A, Abdi A, Nguyen N, Romberg J. Net-trim: convex pruning of deep neural networks with performance guarantee. In: Advances in Neural Information Processing Systems; 2017 Nov 23; Long Beach, CA. Pasadena, CA: NeurIPS (2017) p. 3177–86. doi:10.5555/3294996.3295077
2. Aghasi A, Abdi A, Romberg J. Fast convex pruning of deep neural networks. SIAM J Math Data Sci (2020) 2:158–188. doi:10.1137/19m1246468
3. Ahn M, Pang J-S, Xin J. Difference-of-convex learning: directional stationarity, optimality, and sparsity. SIAM J Optim. (2017) 27:1637–1665. doi:10.1137/16m1084754
4. Alvarez IM, Salzmann M. Learning the number of neurons in deep networks In: Advances in Neural Information Processing Systems; 2018 Oct 11; Barcelona, Spain. Pasadenas, CA: NeurIPS (2016) p. 2270–8.
5. Antoniadis A, Fan J. Regularization of wavelet approximations. J Am Stat Assoc. (2001) 96:939–67. doi:10.1198/016214501753208942
6. Ba J, Caruana R. Do deep nets really need to be deep? Adv Neural Inf Process Syst. (2014) 2:2654–62. doi:10.5555/2969033.2969123
7. Bach FR. Consistency of the group lasso and multiple kernel learning. J Mach Learn Res. (2008) 9:1179–223. doi:10.5555/1390681.1390721
8. Bao C, Dong B, Hou L, Shen Z, Zhang X, Zhang X. Image restoration by minimizing zero norm of wavelet frame coefficients. Inverse Problems. (2016) 32:115004. doi:10.1088/0266-5611/32/11/115004
103. Zhou H, Sehl ME, Sinsheimer JS, Lange K. Association screening of common and rare genetic variants by penalized regression. *Bioinformatics*. (2010) 26: 2375. doi:10.1093/bioinformatics/btq448.

104. Zhou Y, Jin R, Hoi S. Exclusive lasso for multi-task feature selection. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics; Montreal, Canada. JMLR: W&CP (2010) p. 988–95.

105. Zhuang Z, Tan M, Zhuang B, Liu J, Guo Y, Wu Q, et al. Discrimination-aware channel pruning for deep neural networks. In: Advances in Neural Information Processing Systems; 2018 Dec 2–8; Sardinia, Italy. San Diego, CA: NeurIPS (2018) p. 875–86.

**Conflict of Interest:** 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.

Copyright © 2021 Bui, Park, Zhang, Qi and Xin. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.