Neural Networks 119 (2019) 286–298

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Transformed ℓ_1 regularization for learning sparse deep neural networks

Rongrong Ma [a], Jianyu Miao [b], Lingfeng Niu [c,*], Peng Zhang [d]

[a] School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, 100049, China
[b] College of Information Science and Engineering, Henan University of Technology, Zhengzhou, 450001, China
[c] School of Economics and Management, University of Chinese Academy of Sciences, Beijing, 100190, China
[d] Ant Financial Services Group, Hangzhou, 310012, China

ARTICLE INFO

Article history:
Received 13 November 2018
Received in revised form 1 August 2019
Accepted 14 August 2019
Available online 27 August 2019

Keywords:
Deep neural networks
Non-convex regularization
Transformed ℓ_1
Group sparsity

ABSTRACT

Deep Neural Networks (DNNs) have achieved extraordinary success in numerous areas. However, DNNs often carry a large number of weight parameters, leading to the challenge of heavy memory and computation costs. Overfitting is another challenge for DNNs when the training data are insufficient. These challenges severely hinder the application of DNNs in resource-constrained platforms. In fact, many network weights are redundant and can be removed from the network without much loss of performance. In this paper, we introduce a new non-convex integrated transformed ℓ_1 regularizer to promote sparsity for DNNs, which removes redundant connections and unnecessary neurons simultaneously. Specifically, we apply the transformed ℓ_1 regularizer to the matrix space of network weights and utilize it to remove redundant connections. Besides, group sparsity is integrated to remove unnecessary neurons. An efficient stochastic proximal gradient algorithm is presented to solve the new model. To the best of our knowledge, this is the first work to develop a non-convex regularizer in sparse optimization based method to simultaneously promote connection-level and neuron-level sparsity for DNNs. Experiments on public datasets demonstrate the effectiveness of the proposed method.

1. Introduction

Recently, Deep Neural Networks (DNNs) have achieved remarkable success in many fields (Deng, Yu, et al., 2014; Goodfellow, Bengio, & Courville, 2016; Lecun, Bengio, & Hinton, 2015; Schmidhuber, 2015; Yu, Zhu, Zhang, Huang, & Tao, 0000). One of the key factors of the success is its expressive power, which heavily relies on the large number of parameters (Alvarez & Salzmann, 2016; Yoon & Hwang, 2017; Zhou, Alvarez, & Porikli, 2016). For example, VGG (Simonyan & Zisserman, 0000), a convolutional neural network which won the ImageNet Large Scale Visual Recognition Challenge 2014, consists of 15M neurons and up to 144M parameters. A large number of parameters put heavy burden on both memory and computation power, which make DNNs costly for training and inapplicable to resource limited platforms (Alvarez & Salzmann, 2016). Moreover, models with massive parameters are more easily overfitting when the training data are insufficient (Scardapane, Comminiello, Hussain, & Uncini, 2017; Yoon & Hwang, 2017). These challenges seriously hinder the application of DNNs (Alvarez & Salzmann, 2016). However, DNNs are known to have many redundant parameters (Alvarez & Salzmann, 2016; Cheng, Yu, Feris, Kumar, Choudhary, & Chang, 2015; Denil, Shakibi, Dinh, de Freitas, et al., 2013; Scardapane et al., 2017; Yoon & Hwang, 2017). For example, the work (Denil et al., 2013) shows that in some networks, only five percent of the parameters are enough to achieve acceptable models. A number of research works have focused on compressing and accelerating DNNs (Alvarez & Salzmann, 2016; Cheng, Wang, Zhou, & Zhang, 0000, 2018; Han, Pool, Tran, & Dally, 2015; Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 0000). Among these techniques, one branch of research directions is to promote sparsity in DNNs.

We classify the existing works on sparsity promotion for DNNs into three categories: pruning, dropout, and the sparse optimization based method. Pruning removes weight parameters which are insensitive to the performance with respect to the established dense networks. The seminal work is the Biased Weight Decay (Hanson & Pratt, 1989). Then, the works (Cun, Denker, & Solla, 1989; Hassibi & Stork, 1993; Hassibi, Stork, & Wolff, 1993) use the Hessian loss function to remove network connections. In a recent work (Han et al., 2015), connections having slight effect are removed to obtain sparse networks. There are also methods using various criteria to determine which parameters

* Corresponding author.
E-mail addresses: marongrong16@mails.ucas.ac.cn (R. Ma), jymiao@haut.edu.cn (J. Miao), niulf@ucas.ac.cn (L. Niu), hanyi.zp@alibaba-inc.com (P. Zhang).

https://doi.org/10.1016/j.neunet.2019.08.015
0893-6080/

or connections are unnecessary (Anwar, Hwang, & Sung, 2017; Narang, Diamos, Sengupta, & Elsen, 0000). However, in these approaches, the pruning criteria require manual setups of layer sensitivity and heuristic assumptions are also necessary during the pruning phase (Cheng et al., 0000).

Dropout reduces the size of networks during training by randomly dropping units along with their connections from DNNs (Hinton et al., 0000; Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014; Wan, Zeiler, Zhang, Le Cun, & Fergus, 2013). Biased dropout and crossmap dropout (Poernomo & Kang, 2018) are proposed to implement dropout on hidden units and convolutional layers respectively. These methods can reduce overfitting efficiently and improve the performance. Nonetheless, training a dropout network usually takes more time than training a standard neural network, even if they are with the same architecture (Srivastava et al., 2014). In addition, dropout can only simplify networks during training. Full-sized networks are still needed in the prediction phase. Recently, Shakeout is proposed to randomly enhance or reverse the contribution of each unit to the next layer and dropout can be considered as a special case of Kang, Li, and Tao (2017).

The sparse optimization based method promotes sparsity in networks by introducing structured sparse regularization term into the optimization model of DNNs, and zeroing out the redundant parameters during the process of training. Compared with pruning, this type of approaches does not rely on manual setups. In contrast to dropout, the simplified networks obtained by sparse optimization can also be used in the prediction stage. Moreover, different from most existing methods which compress network with negligible drop of accuracy, experiments show that some sparse optimization based methods could even achieve better performance than their original networks (Alvarez & Salzmann, 2016; Scardapane et al., 2017; Yoon & Hwang, 2017). Considering all these merits, we would construct a sparse network in the framework of optimization with sparse regularizers.

The sparse optimization method can be utilized to various tasks to produce sparse solutions. The key challenge of this approach is the design of regularization functions (Candes, Wakin, & Boyd, 2008; Donoho, 2006; Esser, Lou, & Xin, 2013; Fan & Li, 0000; Gong, Tao, Maybank, Liu, Kang, & Yang, 2016; Xu, 2010; Zhang, 2009; Zhang & Xin, 2017). The ℓ_0 norm, which counts the number of non-zero elements, is the most intuitive form of sparse regularizers and can promote the sparsest solution. However, minimizing ℓ_0 problem is combinatory and usually NP-hard (Natarajan, 1995). The ℓ_1 norm is the most commonly used surrogate (Candes et al., 2008; Donoho, 2006; Yu, Rui, & Tao, 2014), which is convex and can be solved easily. Although ℓ_1 enjoys several good properties, it is sensitive to outliers and may cause serious bias in estimation (Fan & Li, 0000, 2001). To overcome this defect, many non-convex surrogates are proposed and analyzed, including smoothly clipped absolute deviation (SCAD) (Fan & Li, 0000), log penalty (Candes et al., 2008; Mazumder, Friedman, & Hastie, 2011), capped ℓ_1 (Zhang, 2009, 2010), minimax concave penalty (MCP) (Zhang et al., 2010), ℓ_p penalty with $p \in (0, 1)$ (Krishnan & Fergus, 2009; Xu, 2010; Xu, Chang, Xu, & Zhang, 2012), the difference of ℓ_1 and ℓ_2 norms (Esser et al., 2013; Lou, Yin, He, & Xin, 2015; Yin, Lou, He, & Xin, 2015) and transformed ℓ_1 (Nikolova, 2000; Zhang & Xin, 2017, 2018). More and more works have shown the superior performance of non-convex regularizers in both theoretical analysis and real-world applications. Generally speaking, non-convex regularizers are more likely to produce unbiased models with sparser solutions than convex ones (Fan & Li, 0000, 2001; Xu, 2010; Xu et al., 2012).

When applied in DNNs, sparse regularizer is supposed to zero out redundant weights and thus remove unnecessary connections. The removal of a large number of connections can reduce enormous computation and memory requirements. Since the variables in DNNs are weights, which are usually modeled as matrices or tensors, we aim to develop a proper regularizer that can avoid augmenting excessive computation complexity. In general, solving a non-convex optimization problem is much harder and more complex than solving a convex one. Considering the fact that DNNs themselves are complicated and their solving process needs numerous calculations, we suspect that this is the main reason for the exiguity of non-convex regularizer based method for DNNs sparsification. In this work, we wish to develop a proper non-convex regularizer which can avoid these problems when applied to DNNs. After comparing the properties of the commonly used non-convex regularizers, we choose transformed ℓ_1 as the regularizer in our model. It satisfies the three desired properties that a regularizer should result in an estimator with, i.e., unbiasedness, sparsity and continuity (Fan & Li, 2001). Besides, it has a simple formula and a parameter which makes it adaptive for different tasks. In addition, although it is non-convex and non-smooth, its thresholding function has a closed-form solution, which gives a solution idea that utilizes proximal gradient descent to avoid most of increase of the computational complexity due to the introduction of non-convex regularizer component. In order to further minify the scale of the network, we also consider group sparsity as an auxiliary of transformed ℓ_1 to remove unnecessary neurons because of its remarkable performance in promoting neuron-level sparsity (Fang, Wang, Dai, & Wu, 2015; Lebedev & Lempitsky, 2016; Scardapane et al., 2017; Simon, Friedman, Hastie, & Tibshirani, 2013; Yuan & Lin, 2006; Zhou et al., 2016). By combining the transformed ℓ_1 and group sparsity, we propose a new integrated transformed ℓ_1 regularizer. Until now, there have been attempts to apply sparse optimization methods with non-convex regularization to train sparse DNNs. To the best of our knowledge, our work is the first one that utilizes a non-convex regularizer in sparse optimization based method to promote neuron-level and connection-level sparsity in DNNs simultaneously. Extensive experiments are carried out to show the effectiveness of our method. The contribution of this paper is three-fold:

- To obtain sparse DNNs, a new model with non-convex regularizer is proposed. The regularizer integrates transformed ℓ_1 and group sparsity together. To the best of our knowledge, this is the first work which uses a non-convex regularizer to induce sparsity of DNNs in neuron level and connection level at the same time.
- To train the new model, an algorithm based on proximal gradient descent is proposed. Although the transformed ℓ_1 is non-convex, the proximal operators in our algorithm have closed-form solutions and can be computed easily. Finally, most of increase of the computational cost due to the introduction of non-convex regularizer component can be avoided.
- Extensive experiments in computer vision are executed on several public datasets. Compared with popular baselines, experimental results show the effectiveness of the proposed regularizer.

The rest of the paper is organized as follows. Section 2 surveys existing sparse optimization based works which aim to promote sparsity in DNNs and some popular non-convex regularizers. Section 3 introduces the new integrated transformed ℓ_1 regularizer and proposes a proximal gradient algorithm to deal with the new model at the same time. Experiments on several public classification datasets are reported in Section 4. We conclude the paper in Section 5.

2. Related work

2.1. Sparse optimization for DNNs

Sparse optimization based approaches in DNNs achieve sparsity through introducing sparse regularization term to the objective function and turning the training process into an optimization problem. Some pruning methods are also equipped with an objective function regularized by some norms. However, these two categories of methods are inherently different. Pruning methods do not aim to learn the final values of the weights, but rather learn which connections are significant. In contrast, the final value of weights is the key criterion to remove connections in sparse optimization based approaches. Only the weights which are exactly zero will be regarded as uninformative ones and be further removed from the network.

In Collins and Kohli (0000), sparse regularizers including the ℓ_1 regularizer, the shrinkage operator and the projection to ℓ_0 balls are applied to both convolutional layers and fully-connected layers in convolutional neural networks. Nevertheless, these methods often achieve sparsity at the expense of accuracy. Zhou et al. (2016) employs two sparse constraints, including tensor low rank constraint and group sparsity, to zero out weights. Group sparsity and ℓ_1 norm are combined in Alvarez and Salzmann (2016) to zero out redundant connections and achieve sparsity of the network. The work (Scardapane et al., 2017), which exploits the similar regularization as Alvarez and Salzmann (2016), divides outgoing connections of each input neuron, outgoing connections of each neuron in hidden layer and biases into different groups and promote group-level sparsity. Group sparsity and exclusive sparsity are combined as a regularization term in a recent work (Yoon & Hwang, 2017) to enforce sparsity, by utilizing the sharing and competing relationships among various network weights. These methods can achieve sparsity with comparable or even better accuracy than the original network.

2.2. Non-convex regularization function

The work (Fan & Li, 2001) has proposed that a good penalty function which serves as the regularizer should result in an estimator with three desired properties: unbiasedness, sparsity and continuity. Obviously, the regularizers with these three properties simultaneously should be non-convex. The smoothly clipped absolute deviation (SCAD) (Fan & Li, 0000) is the first regularizer proven to fulfill these properties (Fan & Li, 2001), whose definition for vector variable $\mathbf{x} = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^n$ is given as $\mathbf{P}(\mathbf{x}; \lambda, \gamma) = \sum_{i=1}^{n} P(x_i; \lambda, \gamma)$, in which

$$P(x_i; \lambda, \gamma) = \begin{cases} \lambda|x_i|, & \text{if } |x_i| \le \lambda \\ \frac{2\gamma\lambda|x_i| - x_i^2 - \lambda^2}{2(\gamma-1)}, & \text{if } \lambda < |x_i| < \gamma\lambda \\ \lambda^2(\gamma+1)/2, & \text{if } |x_i| \ge \gamma\lambda, \end{cases} \tag{1}$$

where $\lambda > 0$ and $\gamma > 2$. It is obvious that SCAD is a two-parameter function composed of three parts. Later, a concave regularizer with two pieces, called minimax concave penalty (MCP), is proposed in Zhang et al. (2010). It is formulated as $\mathbf{P}_\gamma(\mathbf{x}; \lambda) = \sum_{i=1}^{n} P_\gamma(x_i; \lambda)$, where

$$P_\gamma(x_i; \lambda) = \begin{cases} \lambda|x_i| - x_i^2/(2\gamma), & \text{if } |x_i| \le \gamma\lambda \\ \gamma\lambda^2/2, & \text{if } |x_i| > \gamma\lambda \end{cases} \tag{2}$$

for parameter $\gamma > 1$. Log penalty is a generalization of elastic net family, which is formulated as $\mathbf{P}(\mathbf{x}; \gamma) = \sum_{i=1}^{n} P(x_i, \gamma)$ with

$$P(x_i; \gamma) = \frac{\log(\gamma|x_i| + 1)}{\log(\gamma + 1)}, \tag{3}$$

where parameter $\gamma > 0$. Through this penalty family, the entire continuum of penalties from ℓ_1 ($\gamma \to 0_+$) to ℓ_0 ($\gamma \to \infty$) can be obtained (Mazumder et al., 2011). Capped ℓ_1 is another approximation of ℓ_0 (Zhang, 2009), whose definition is

$$\mathbf{P}(\mathbf{x}; a) = \sum_{i=1}^{n} \min(|x_i|, a), \tag{4}$$

where a is a positive capped parameter. Obviously, when $a \to 0$, $\sum_i \min(|x_i|, a)/a \to \|\mathbf{x}\|_0$. Transformed ℓ_1, which is a smooth version of capped ℓ_1, is discussed in the works (Nikolova, 2000; Zhang & Xin, 2017, 2018). Some other non-convex metrics with concise form are also considered as alternatives to improve ℓ_1, including ℓ_p with $p \in (0, 1)$ (Krishnan & Fergus, 2009; Xu, 2010; Xu et al., 2012), whose formula is

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \tag{5}$$

and ℓ_{1-2} (Esser et al., 2013; Lou et al., 2015; Yin et al., 2015), which is the difference between ℓ_1 and ℓ_2 norm. Contour plots of several popular regularizers are displayed in Fig. 1.

3. DNNs with transformed ℓ_1 regularizer

Our objective is to construct a sparse neural network with fewer parameters and comparable or even better performance than the dense model. In neural networks with multiple layers, let $W^{(l)}$ represent the weight matrix of lth layer. By regularizing the weights of each layer respectively, the training objective function for supervised learning can be formulated as

$$\min_{\{W^{(l)}\}} \mathcal{L}\left(\{W^{(l)}\}, \mathcal{T}\right) + \lambda \sum_{l=1}^{L} \Omega(W^{(l)}), \tag{6}$$

where $\mathcal{T} = \{x_i, y_i\}_{i=1}^{N}$ is a training dataset that has N instances, in which $x_i \in \mathbb{R}^p$ is a p-dimension input sample and $y_i \in \{1, \ldots, K\}$ is its corresponding class label. λ is a positive hyperparameter, which controls the balance between the loss term $\mathcal{L}(\{W^{(l)}\}, \cdot)$ and the regularization term $\sum_{l=1}^{L} \Omega(W^{(l)})$. To induce sparsity in DNNs effectively, we will concentrate on constructing a proper sparse regularization function $\Omega(W^{(l)})$ in this paper.

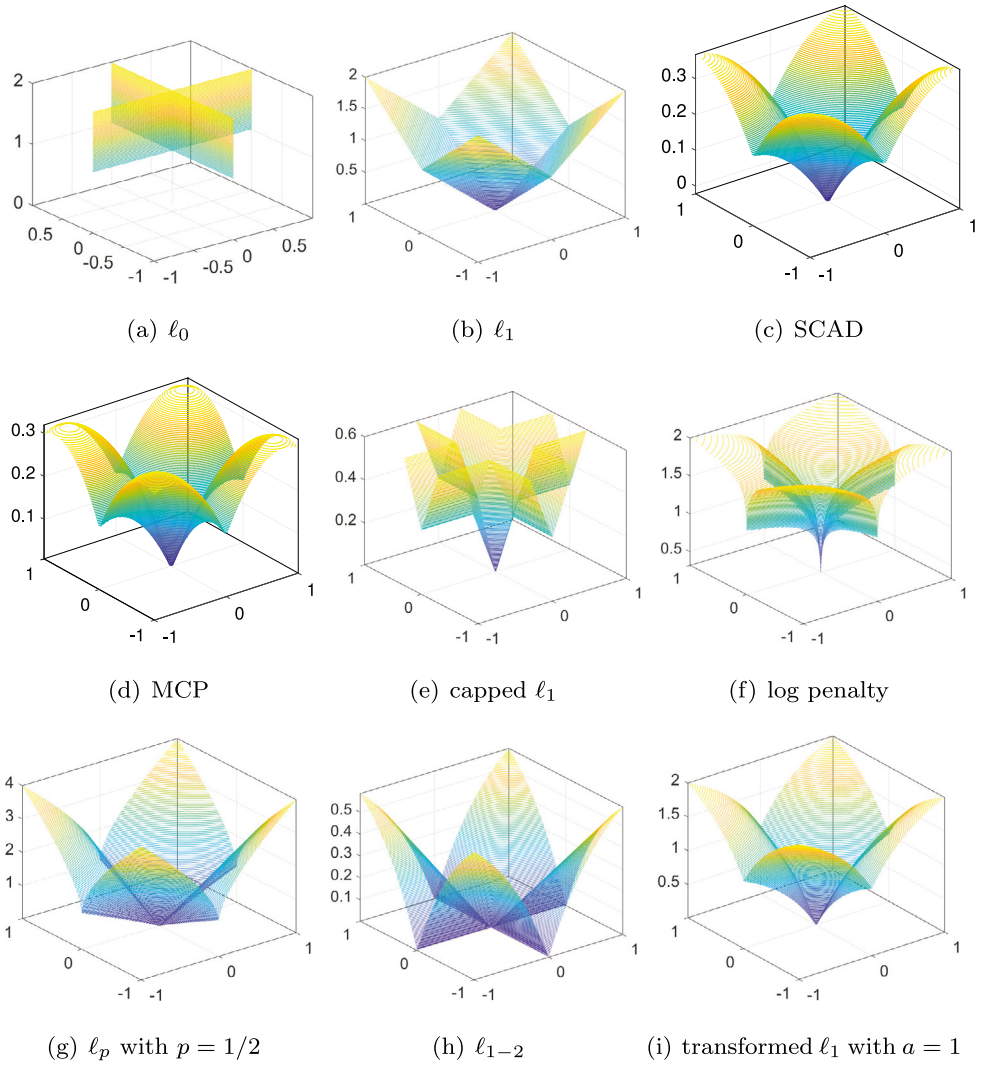
As pointed out in Yoon and Hwang (2017), sparsity-inducing problem in DNNs can also be considered from the perspective of feature selection. Sparse regularizer is designed to promote sparsity of results by inducing different weights at each layer to compete for few significant features, resulting in those remaining weights fitting to different features as many as possible and thus reducing the dependence and redundance among them. Although the regularization function in the feature selection problem can be both convex and non-convex (Gui, Sun, Ji, Tao, & Tan, 2017), it has been shown that non-convex regularizers outperform convex ones in numerous tasks (Bradley & Mangasarian, 1998; Shi, Miao, Wang, Zhang, & Niu, 0000; Xiang, Tong, & Ye, 2013; Zhang, Ding, Zhang, & Nie, 2014). Therefore, we would like to seek an appropriate non-convex regularizer to promote sparsity in DNNs.

3.1. The model

The transformed ℓ_1 ($T\ell_1$) functions are a one parameter family of bilinear transformations composed with the absolute value function (Nikolova, 2000; Zhang & Xin, 2017, 2018). Mathematically, the $T\ell_1$ function for a scalar variable x is defined as follows,

$$\rho_a(x) = \frac{(a+1)|x|}{a + |x|}, \tag{7}$$

where a is a positive parameter which controls the shape of the function. One can easily verify that when a approaches zero, $\rho_a(x)$ tends to an indicative function $I(x)$ whose definition is: $I(x) = 1$,



(a) ℓ_0 (b) ℓ_1 (c) SCAD

(d) MCP (e) capped ℓ_1 (f) log penalty

(g) ℓ_p with $p = 1/2$ (h) ℓ_{1-2} (i) transformed ℓ_1 with $a = 1$

Fig. 1. Contour plots for several popular convex and non-convex norms in two dimensions. (a) ℓ_0. (b) ℓ_1. (c) SCAD with parameters $\lambda = 0.28$ and $a = 3.7$. (d) MCP with $\lambda = 0.4$ and parameter $\gamma = 2$. (e) capped ℓ_1 with $a = 0.3$. (f) log penalty with $\gamma = 10^3$. (g) ℓ_p with $p = 1/2$. (h) ℓ_{1-2}. (i) transformed ℓ_1 with $a = 1$.

if $x \neq 0$ and $I(x) = 0$, otherwise. In contrast, when a approaches infinity, $\rho_a(x)$ tends to the absolute value function $|x|$.

When acting on vectors, the definition of $T\ell_1$ can be formulated as

$$T\ell_1(\mathbf{x}) = \sum_{i=1}^{N} \rho_a(x_i), \ \forall\ \mathbf{x} = (x_1, x_2, \ldots, x_N)^T \in \mathbb{R}^N. \tag{8}$$

With the change of parameter a, $T\ell_1$ interpolates ℓ_0 and ℓ_1 norm as,

$$\begin{aligned} \lim_{a\to 0^+} T\ell_1(\mathbf{x}) &= \sum_{i=1}^{N} I_{\{x_i\neq 0\}} = \|\mathbf{x}\|_0, \\ \lim_{a\to +\infty} T\ell_1(\mathbf{x}) &= \sum_{i=1}^{N} |x_i| = \|\mathbf{x}\|_1. \end{aligned} \tag{9}$$

As a demonstration, we plot the contours of $T\ell_1$ with $a = 10^{-2}, 1, 10^2$ in Fig. 2. From the set of figures, we can observe that $T\ell_1$ can indeed approximate $\ell_0, \ell_{1/2}$ and ℓ_1 well with the adjustment of parameter a.

The work (Zhang, Yin, & Xin, 2017) extends the $T\ell_1$ to matrix space based on the singular values in matrix completion problem, which interpolates the rank and the nuclear norm through the nonnegative parameter a. In this work, since the sparsity of neural networks is introduced in the component wise for weights of layers, we propose the $T\ell_1$ for matrix in the following form:

$$T\ell_1(X) = \sum_{i,j} \rho_a(x_{i,j}), \tag{10}$$

where $x_{i,j}$ is the element of ith row and jth column in matrix $X \in \mathbb{R}^{m\times n}$. Then, the regularization function $\Omega(W^{(l)})$ in (6) becomes $T\ell_1(W^{(l)})$. We choose $T\ell_1$ as the sparse regularizer based on the following reasons. Firstly, compared with the convex regularizers such as ℓ_1, $T\ell_1$ is unbiased (Lv & Fan, 2009) and can produce sparser solution (Zhang & Xin, 2017). Secondly, compared with non-Lipschitz regularizers such as ℓ_p, the trending rate of $T\ell_1$ can be controlled. Thirdly, compared with the piecewise regularizers such as SCAD and MCP, the formula of $T\ell_1$ is more concise. Last but not least, compared with non-parameter ℓ_{1-2}, $T\ell_1$, which relies on a parameter a, is adjustable for various tasks.

Besides removing as many unnecessary connections as possible, reducing the number of neurons also plays a powerful role in light weight neural networks. Group sparsity, which requires that the elements in one group are all zero, or none of them is, is a typical way of removing neurons and has been employed in several works (Alvarez & Salzmann, 2016; Lebedev & Lempitsky, 2016; Scardapane et al., 2017; Zhou et al., 2016). It is an

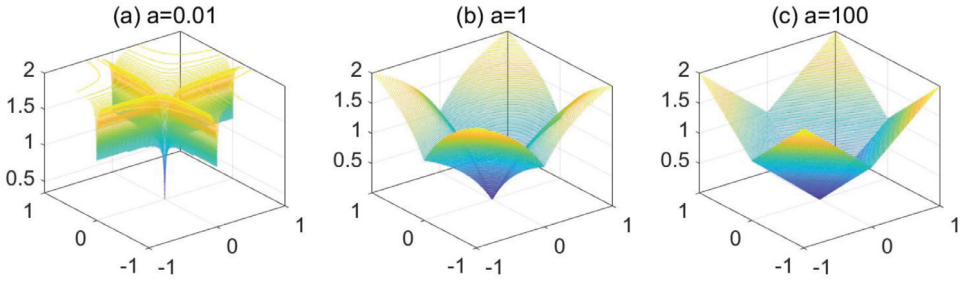


Fig. 2. Contour plots of $T\ell_1$ with different values of parameter a in two dimensions. (a) $T\ell_1$ with $a = 10^{-2}$. (b) $T\ell_1$ with $a = 1$. (c) $T\ell_1$ with $a = 10^2$. Compared with subFigs. 1(a), 1(b) and 1(g), we can see that the contour plot of $T\ell_1$ with $a = 10^{-2}$ is similar to the contour plot of ℓ_0 while $T\ell_1$ with $a = 10^2$ is similar to ℓ_1. In addition, $T\ell_1$ with $a = 1$ looks like $\ell_{1/2}$.

expansion of the $\ell_{2,1}$ norm (Gong, Tao, Chang, & Yang, 2017; Liu, Ji, & Ye, 2009; Nie, Huang, Cai, & Ding, 2010; Yang, Shen, Ma, Huang, & Zhou, 2011). In this work, to further minify the size of the networks and reduce the computation complexity, we consider using group sparsity as an assistant of $T\ell_1$ and propose the integrated transformed ℓ_1 regularizer in the following way:

$$\Omega(W^{(l)}) = \mu_l T\ell_1(W^{(l)}) + (1-\mu_l)\sum_g \|W_g^{(l)}\|_2, \tag{11}$$

where $g \in \mathcal{G}$ is the weight group obtained by dividing the weight matrix according to neurons. $W_g^{(l)}$ denotes the weight vector for group g defined on $W^{(l)}$. μ_l is a positive parameter that balances the $T\ell_1$ term and the group sparsity term. In the above regularizer, group sparsity, which uses the ℓ_1 norm to zero out variables that are grouped by the ℓ_2 norm, helps automatically decide the number of neurons in each layer. $T\ell_1$ plays the role of inducing sparsity in the connection level. Both of them work together to decide the suitable number of neurons and promote sparsity among the remaining simultaneously.

Applying the regularization function (11) to problem (6), our model of transformed ℓ_1 regularization for learning sparse deep neural networks can be formulated as:

$$\min_{\{W^{(l)}\}} \mathcal{L}\left(\{W^{(l)}\}, \mathcal{T}\right) + \lambda \sum_{l=1}^{L}\left(\mu_l T\ell_1(W^{(l)}) + (1-\mu_l)\sum_g \|W_g^{(l)}\|_2\right). \tag{12}$$

When the training process of (12) terminates, the weights of some connections will turn to zero since they have slight effect on the final performance and then be removed from the network. If all the outgoing or ingoing connections of a neuron are removed, this neuron will be removed as well. Afterwards, a sparse network with less neurons and connections can be yielded. Although there have been several works that used regularization term to promote sparsity in DNNs, to the best of our knowledge, this is the first work to use non-convex regularizer to achieve neuron-level and connection-level sparsity of DNNs simultaneously. Since non-convex regularizers tend to outperform convex ones in terms of sparsity-promoting effect, our integrated $T\ell_1$ regularizer should be able to obtain network with a sparser structure intuitively.

3.2. The optimization algorithm

In this subsection, we will focus on how to train the model (12) efficiently. The integrated regularizer proposed is non-convex and non-smooth, causing increased difficulty in solving (12). The standard stochastic gradient descent algorithm commonly used in DNNs is no longer applicable. Proximal methods, which are popularly used to handle with non-smooth problems, can solve the problem efficiently. They can be interpreted as solving optimization problems by finding fixed points of appropriate operators. They are often conceptually and mathematically simple and can work fast under extremely general conditions (Parikh, Boyd, et al., 2014). Therefore, we consider using proximal gradient method to solve the model.

The proximal gradient approach iteratively minimizes (12) layer by layer through the following formula:

$$W_{t+1}^{(l)} = prox_{\lambda\gamma\Omega}(W_t^{(l)} - \gamma\nabla\mathcal{L}(W_t^{(l)}, \mathcal{T})), \tag{13}$$

where γ is the step size. $W_t^{(l)}$ represents the variable of lth layer in tth iteration and $W_{t+1}^{(l)}$ is the variable of lth layer obtained after current iteration. The *prox* denotes proximal operator whose definition on point X with respect to function f is formulated as

$$prox_f(X) = \arg\min_Y \left\{f(Y) + (1/2)\|Y - X\|_2^2\right\}, \tag{14}$$

where $\|\cdot\|_2$ is the Euclidean norm. Therefore, (13) can be expanded as:

$$W_{t+1}^{(l)} = \arg\min_{W^{(l)}} \left\{\Omega(W^{(l)}) + \frac{1}{2\lambda\gamma}\|W^{(l)} - (W_t^{(l)} - \gamma\nabla\mathcal{L}(W_t^{(l)}, \mathcal{T}))\|_2^2\right\}. \tag{15}$$

In this work, since the problem we solve is in DNNs, it is costly to compute the gradient on whole training dataset for each update. In fact, in DNNs, stochastic gradient descent (SGD) rather than standard gradient descent is the commonly used optimization procedure (Bottou, 1991). Thus, we use SGD to replace the gradient descent step in (15). In detail, SGD involves computing the outputs and errors, calculating the average gradient on a few instances and adjusting the weights accordingly. Then, (15) turns into

$$W_{t+1}^{(l)} = \arg\min_{W^{(l)}} \left\{\Omega(W^{(l)}) + \frac{1}{2\lambda\gamma}\| W^{(l)} - (W_t^{(l)} - \gamma\sum_{i=1}^{n}\nabla\mathcal{L}(W_t^{(l)}, \{x_i, y_i\})/n)\|_2^2\right\}, \tag{16}$$

where n is the mini-batch size in SGD and $\{x_i, y_i\}_{i=1}^n$ are the n samples randomly selected from the dataset $\mathcal{T}$.

The regularization term $\Omega(W^{(l)})$ in our objective function is combined by two single regularizers and computing the proximal operator of such regularizer is not easy. Fortunately, the proximal gradient method can avoid this procedure and only proximal operator of each single regularizer is required. To be more specific, the objective function (12) can be solved simply by iteratively implementing update on the variables layer by layer through the proximal operators of two regularizers in succession after

performing a gradient step on the variables based on the loss term, i.e.,

$$W_{t+1}^{(l)} = prox_{\lambda\gamma(1-\mu_l)GS}(prox_{\lambda\gamma\mu_l T\ell_1}(W_t^{(l)} - \gamma \sum_{i=1}^{n} \nabla\mathcal{L}(W_t^{(l)}, \{x_i, y_i\})/n)). \tag{17}$$

To calculate (17), the proximal operators of $T\ell_1$ and group sparsity are required. As can be seen from (14), computing the proximal operator of a convex function turns into solving a small convex regularized optimization problem, which usually obtains a closed-form solution, for example, group sparsity in our model. The proximal operator of group sparsity on $W^{(l)}$ is formulated as:

$$prox_{\lambda\gamma(1-\mu_l)GS}(W^{(l)}) = \left(1 - \lambda\gamma(1-\mu_l)/||\mathbf{w}_g^{(l)}||_2\right)_+ w_{g,i}^{(l)}, \tag{18}$$

for all g and i, where g is a group, and i is the index in each group. $\mathbf{w}_g^{(l)}$ denotes weight vector composed of elements in gth group of lth layer and $w_{g,i}^{(l)}$ represents the ith element in gth group of lth layer. $(x)_+$ denotes the maximum between x and 0. However, for some non-convex functions, their proximal operators might not have closed forms in general, like ℓ_p penalty with $p \in (0, 1)$. In Zhang and Xin (2017), the closed-form proximal operator of $T\ell_1$ for a scalar has been calculated even though $T\ell_1$ is non-convex. Next, we will illustrate that there also exists a closed-form expression for the proximal operator of $T\ell_1$ in matrix space. As mentioned earlier, the proximal operator of $T\ell_1$ on $W^{(l)}$ can be obtained by solving the optimization problem as follows,

$$\min_{\hat{W}^{(l)}} \frac{1}{2\lambda\gamma\mu_l}\|\hat{W}^{(l)} - W^{(l)}\|_2^2 + T\ell_1(\hat{W}^{(l)}). \tag{19}$$

Expanding the optimization problem above yields:

$$\min_{\hat{W}^{(l)}} \sum_i \sum_j \left(\frac{1}{2\lambda\gamma\mu_l}(\hat{w}_{i,j}^{(l)} - w_{i,j}^{(l)})^2 + \frac{(a+1)|\hat{w}_{i,j}^{(l)}|}{a + |\hat{w}_{i,j}^{(l)}|}\right), \tag{20}$$

where $\hat{w}_{i,j}^{(l)}$ and $w_{i,j}^{(l)}$ are elements of ith row and jth column in $\hat{W}^{(l)}$ and $W^{(l)}$, respectively. Thus, (19) can be optimized for each i and j respectively, i.e., it can be solved by optimizing

$$\min_{\hat{w}_{i,j}^{(l)}} \frac{1}{2\lambda\gamma\mu_l}(\hat{w}_{i,j}^{(l)} - w_{i,j}^{(l)})^2 + \frac{(a+1)|\hat{w}_{i,j}^{(l)}|}{a + |\hat{w}_{i,j}^{(l)}|} \tag{21}$$

for each i and j. This is an unconstrained optimization problem with univariable $\hat{w}_{i,j}^{(l)}$, whose solution can be obtained by calculating its subgradient. The optimal solution of (21) is formulated as follows,

$$\hat{w}_{i,j}^{(l)} = \begin{cases} 0, & if \ \ |w_{i,j}^{(l)}| \le t \\ g_{\lambda\gamma\mu_l}(w_{i,j}^{(l)}), & otherwise \end{cases} \tag{22}$$

for all i and j, where $g_{\lambda\gamma\mu_l}(w)$ is defined as follows,

$$g_{\lambda\gamma\mu_l}(w) = sgn(w)\left\{2(a + |w|)\cos(\varphi_{\lambda\gamma\mu_l}(w)/3)/3 - 2a/3 + |w|/3\right\} \tag{23}$$

with $\varphi_{\lambda\gamma\mu_l}(w) = \arccos\left(1 - \frac{27\lambda\gamma\mu_l a(a+1)}{2(a+|w|)^3}\right)$, and t is given as follows,

$$t = \begin{cases} \lambda\gamma\mu_l(a+1)/a, & if \ \ \lambda\gamma\mu_l \le \frac{a^2}{2(a+1)} \\ \sqrt{2\lambda\gamma\mu_l(a+1)} - a/2, & otherwise. \end{cases} \tag{24}$$

More details for the solving process can be found in Zhang and Xin (2017). Therefore, the proximal operator of $T\ell_1$ can be formulated as

$$prox_{\lambda\gamma\mu_l T\ell_1}(W^{(l)}) = \begin{cases} 0, & if \ \ |w_{i,j}^{(l)}| \le t \\ g_{\lambda\gamma\mu_l}(w_{i,j}^{(l)}), & otherwise \end{cases} \tag{25}$$

for each i and j, in which $g_{\lambda\gamma\mu_l}$ and t is defined as (23) and (24), respectively.

We summarize the whole optimization process in Algorithm 1. In the algorithm, the stopping criterion is predefined and the commonly used one is the decrease of loss between two consecutive epochs is less than a threshold or the maximum number of iterations is achieved.

Algorithm 1 Stochastic Proximal Gradient Descent for Model (12).

Input: initial weight matrix W_0, regularization parameter λ, balancing parameter for each layer μ_l, learning rate γ, mini-batch size n, training dataset $\mathcal{T}$

```
t = 1
repeat
  Randomly select n samples from T
  for each layer l do
    for each sample {x_i, y_i} in the n samples selected do
      L_i^(l) := ∇L(W_{t-1}^(l), {x_i, y_i})
    end for
    L^(l) := Σ_{i=1}^n L_i^(l)/n
    W_t^(l) := W_{t-1}^(l) − γ L^(l)
    Update W_t^(l) by (25): W_t^(l) := prox_{λγμ_l Tℓ_1}(W_t^(l))
    Update W_t^(l) by (18): W_t^(l) := prox_{λγ(1−μ_l)GS}(W_t^(l))
  end for
  t := t + 1
until some stopping criterion is satisfied
```

Output: The solution W_{t-1}

4. Experiments

In this section, we evaluate the proposed combined regularizer on several real-world datasets. The regularizer is applied to all layers of the network, except the bias term.

4.1. Baselines and datasets

To demonstrate the superiority of the integrated $T\ell_1$, we compare it with several state-of-the-art baselines:

- **Network with ℓ_2.** It is a fully connected network and is utilized as a reference to illustrate the sparsity-promoting ability of the competitors. ℓ_2 cannot promote sparsity and it is only used to improve the generalization ability and the performance of the model. This model can be trained with the standard stochastic gradient descent algorithm.
- **Network regularized by ℓ_1.** ℓ_1 is a biased and commonly used convex regularizer. It can only achieve sparsity in connection level. This model is convex and Lipschitz continuous and can also be solved by the standard stochastic gradient descent algorithm.
- **Network regularized by sparse group lasso (SGL).** The SGL is a regularizer that combines group sparsity and ℓ_1 regularizer. The group sparsity is used to introduce neuron-level sparsity and ℓ_1 regularizer is still utilized to promote sparsity among connections. Such a model is convex but non-smooth, thus the proximal gradient descent algorithm is utilized to train it.

- **Network regularized by combined group and exclusive sparsity (CGES)**. As can be seen from the name of CGES, the CGES combines group sparsity and exclusive sparsity. It differs from the SGL in that the CGES uses exclusive sparsity instead of ℓ_1 to promote connection-level sparsity. This model is also non-smooth and we use the proximal gradient descent algorithm to train it.
- **Network regularized by group sparsity and ℓ_{1-2} (GSℓ_{1-2}).** This regularization term is used to illustrate the superiority of $T\ell_1$ among other non-convex regularizers. We choose ℓ_{1-2} as the non-convex competitor. Meanwhile, the group sparsity is also used to remove neurons. The proximal gradient descent algorithm is employed to train this model.

We select several public classification datasets which are commonly used in DNNs to conduct experiments,

- **DIGITS**. This is a toy dataset of handwritten digits, composed of 1,797 8 × 8 grayscale images. We use this dataset to illustrate the effect of parameter a in the integrated $T\ell_1$ regularizer and the sparsity-promoting capacity of several regularizers. The base network that we choose for this dataset is a network with one single convolutional layer and two fully connected layers.
- **MNIST**. This dataset consists of 70,000 28 × 28 grayscale images of handwritten digits, which can be classified into 10 classes. The number of training instances and test samples is 60,000 and 10,000, respectively. As for the base network, a network with two convolutional layers and two fully connected layers is chose.
- **Fashion-MNIST**. This dataset consists of a training set with 60,000 instances and a test set with 10,000 examples. Each example is a 28 × 28 grayscale image, associated with a label from 10 classes (T-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag and ankle boot). Fashion-MNIST serves as a direct drop-in replacement for the original MNIST dataset. For the base network, we use a network with two convolutional layers followed by three fully connected layers.
- **PENDIGITS**. This dataset is composed of 10,992 4 × 4 grayscale images of handwritten digits 0–9, where there are 7,494 training instances and 3,498 test samples. For the base network, we use a network with two convolutional layers and two fully connected layers.
- **Sensorless Drive Diagnosis (SDD)**. This dataset is downloaded from the UCI repository. It contains 58,508 examples obtained under 11 different operating conditions. In this dataset, we need to predict a motor with one or more defective components, starting from a set of 48 features obtained from electric drive signals of the motor. For the base network, we use a convolutional network with two convolutional layers followed by three fully connected layers.
- **CIFAR-10**. This dataset consists of 60,000 32 × 32 color images in 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck), with 6,000 images per class. The dataset is divided into one test batch with 10,000 images and five training batches with 10,000 instances in each batch. We choose a network with four convolutional layers followed by two fully connected layers as the base network.
- **CIFAR-100**. This dataset comprises 60,000 32 × 32 pixels color images as in CIFAR-10. However, these images can be divided into 100 categories instead of 10 classes and each class has 600 images. In each category, 500 images are used to train the model and 100 images are used to test the performance of the model. A wide residual network, which has 16 layers with two blocks in each convolution group and widening factor 10, is utilized as the base network.

4.2. Experimental setup

We use Tensorflow framework to implement the models. In all cases, we employ the ReLU function $f(x) = \max(0, x)$ as the activation function. As for the output layer, we apply the softmax activation function. If $\mathbf{x} \in \mathbb{R}^n$ denotes the value that is input to softmax, the i_{th} output can be obtained as,

$$f(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)}. \tag{26}$$

Besides, one-hot encoding is used to encode different classes. We initialize the weights of the network by Xavier or random initialization according to a normal distribution. The size of mini-batch is varied depending on the scale of the datasets. We choose the standard cross-entropy loss as the loss function. In the experiments, we would like group sparsity to play the leading role in the lower layers while $T\ell_1$ regularizer has more effect at the top layers, just as mentioned in Yoon and Hwang (2017). To this end, we dynamically set $\mu_l = s + (1 - 2s)(l - 1)/(L - 1)$, where L is the number of layers, $l \in \{1, 2, \ldots, L\}$ is the index of each layer and s is the lowest value that can be used for the $T\ell_1$ term. The regularization parameter λ and the parameter a in $T\ell_1$ are selected through the grid search technique, with λ varying from 10^{-6} to 10^{-4} and a in $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$. On one specific dataset, we use the same network architecture for various penalties to keep the comparison fair. The stopping criterion is the achievement of the maximum number of iterations. For MNIST, Fashion-MNIST, SDD, PENDIGITS and CIFAR-10, the training process will stop after 30,000 steps. For CIFAR-100, the training process will stop after 60,000 steps. For ImageNet, the training process will stop after 300,000 steps. The early stopping strategy is also utilized to prevent overfitting. Early stopping means that the training will stop when the improvements on the validation set cease. To obtain more reliable results, we repeatedly run the training process three times. The final results are reported as an average with standard deviations over these three times.

4.3. Performance of integrated $T\ell_1$ regularizer

In this subsection, we compare our integrated $T\ell_1$ with several baselines to verify the superiority of our model. To quantitatively measure the performance of various models, three metrics are utilized, including the prediction accuracy, the corresponding number of parameters used in the network and the corresponding number of floating point operations (FLOP). A higher accuracy means that the model can train a better network to implement classification tasks. A lower FLOP indicates that the network can reduce the computation complexity more significantly. The ability of saving memory is reflected by the parameters used in the network. Therefore, the smaller the number of parameters used is, the better the regularizer is.

We list the results in Table 1. The average ranks results of each method for all three measurements are reported in the last three rows of Table 1. The average rank is a simple ranking method, which orders the algorithms on each dataset and calculates the average rank of each algorithm for each indicator (Brazdil & Soares, 2000). Such ranking result can represent general performance of algorithms in terms of each measurement. The best results are highlighted in bold face. We also present the results on PENDIGITS and CIFAR-10 by boxplots in Fig. 3, in which we abbreviate "integrated $T\ell_1$" as "I$T\ell_1$". As seen from Table 1, the performance of our model is comparable when compared

Table 1
Performance of different methods on the datasets. The best results are highlighted in bold face.

Dataset	Measure	ℓ_2	ℓ_1	SGL	CGES	GSℓ_{1-2}	Integrated $T\ell_1$
MNIST	accuracy	0.9938 ± 0.0001	0.9889±0.0007	0.9890 ± 0.0003	0.9854±0.0007	**0.9984±0.0004**	0.9850±0.0002
	FLOP	1	0.4455±0.0023	0.7828±0.0129	0.4927±0.0053	0.1266 ± 0.0176	**0.1075±0.0039**
	parameter	1	0.0819±0.0052	0.3975±0.0257	0.1786±0.0009	0.1460 ± 0.0155	**0.0790±0.0003**
Fashion-MNIST	accuracy	**0.9117±0.0020**	0.8947 ± 0.0009	0.8927±0.0006	0.8804±0.0233	0.8802 ± 0.0012	0.8873±0.0026
	FLOP	1	0.7056±0.0034	0.7038±0.0015	0.7446±0.0021	0.5968 ± 0.0010	**0.3097±0.0099**
	parameter	1	0.2323±0.0092	**0.2275±0.0039**	0.9872 ± 0.0030	0.3362±0.0043	0.3102±0.0076
SDD	accuracy	0.9912 ± 0.0083	0.9833±0.0013		0.9909 ± 0.0034	**0.9932±0.0012**	0.9897±0.0011
	FLOP	1	0.4085±0.0429	–	0.3072±0.0619	0.2589 ± 0.0094	**0.1584±0.0494**
	parameter	1	0.4053±0.0432		0.3035±0.0622	0.2552 ± 0.0094	**0.1608±0.0480**
PENDIGITS	accuracy	**0.9755±0.0016**	0.9715±0.0014	0.9739±0.0021	0.9732±0.0015	0.9718 ± 0.0036	0.9745 ± 0.0013
	FLOP	1	0.4414±0.0265	0.7220±0.0038	0.6027±0.0075	0.3687 ± 0.0386	**0.3301±0.0089**
	parameter	1	0.4296±0.0241	0.7241±0.0034	0.5948±0.0114	0.3599 ± 0.0384	**0.3181±0.0080**
CIFAR-10	accuracy	**0.8015±0.0051**	0.7716±0.0052	0.7667±0.0006	0.7775±0.0018	0.7802±0.0055	0.7804±0.0025
	FLOP	1	0.8827±0.0007	0.6817±0.0076	0.8210±0.0180	0.8036±0.0083	**0.2830±0.0017**
	parameter	1	0.5104±0.0041	0.8076±0.0127	0.7310±0.0035	0.7506±0.0016	**0.2623±0.0030**
CIFAR-100	accuracy	**0.7157±0.0025**	0.7006 ± 0.0004		0.7041 ± 0.0035		0.7048 ± 0.0031
	FLOP	1	0.7407 ± 0.0458	–	0.8454 ± 0.0142	–	**0.5890±0.0749**
	parameter	1	**0.5409±0.0610**		0.7236 ± 0.0196		0.5749 ± 0.0878
Average rank	accuracy	**1.3**	4.0	4.3	4.0	3.5	3.3
	FLOP	5.5	3.5	4.3	3.8	2.7	**1**
	parameter	5.5	2.3	4.5	3.7	3.3	**1.5**

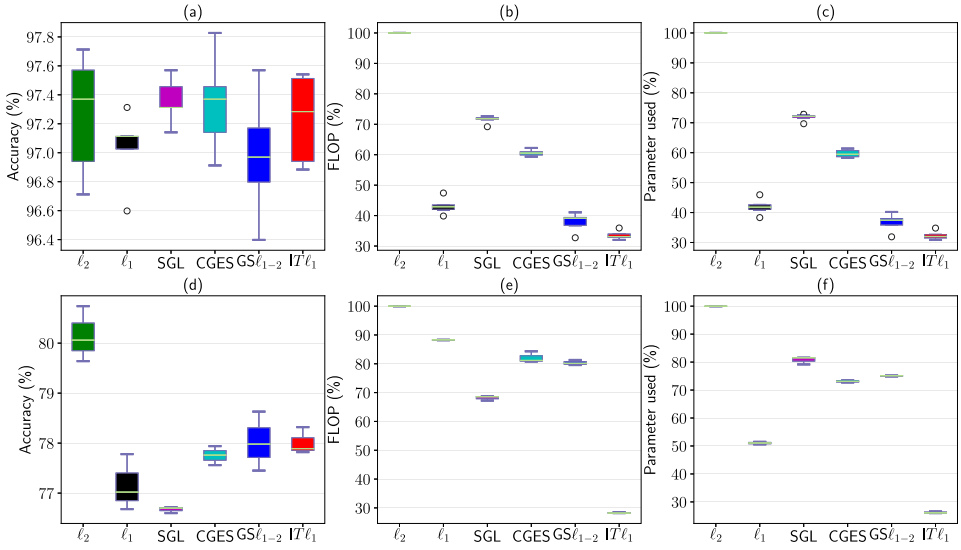


Fig. 3. Comparisons of classification and sparsity-promoting effect of six algorithms on PENDIGITS and CIFAR-10. The *x*-axis presents six algorithms: ℓ_2, ℓ_1, SGL (sparse group lasso in Scardapane et al. (2017)), CGES (combined group and exclusive sparsity in Yoon and Hwang (2017)), GSℓ_{1-2} (group sparsity and ℓ_{1-2}) and ITℓ_1 (integrated $T\ell_1$ regularizer). The *y*-axis is prediction accuracy, FLOP or parameters used in the network. Fig. (a), Fig. (b) and Fig. (c) are the results on PENDIGITS and Fig. (d), Fig. (e) and Fig. (f) are the results on CIFAR-10. The notched boxes have lines at the lower quartile, median, and upper quartile values. The whiskers are lines extending from each end of the box to the most extreme data value with 1.5·IQR (interquartile range) of the box. Outliers, whose value is beyond the ends of the whiskers, are displayed by dots.

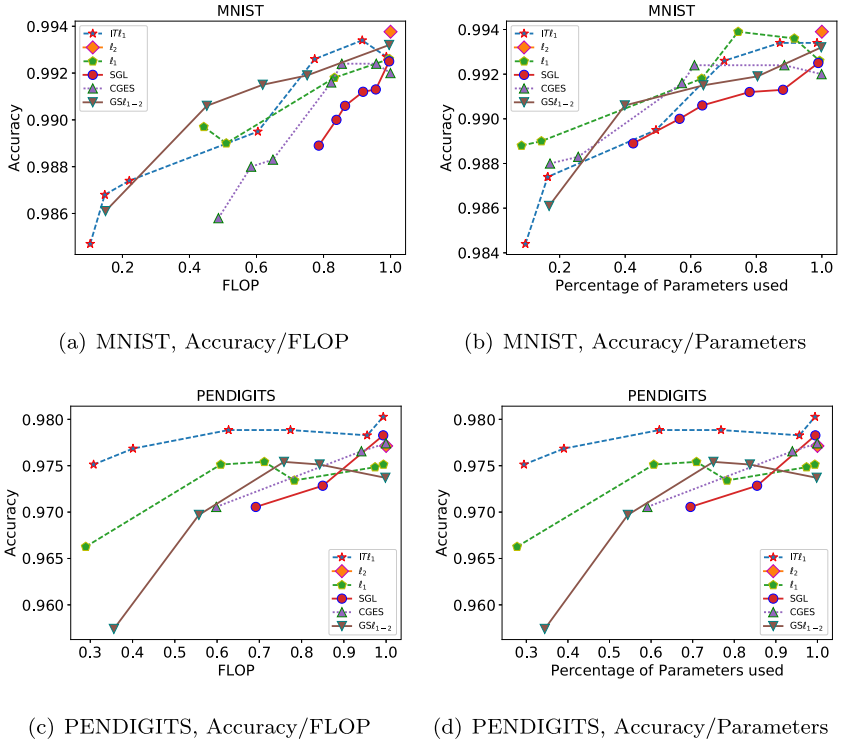
with other baselines. The average rank results illustrate that the prediction accuracy of the network with ℓ_2 is the best while the network with integrated $T\ell_1$ is sparsest. Among the networks with sparse regularizers, our model also has the highest accuracy. The reason why the network with ℓ_2 can achieve highest accuracy might be that such network is dense and can learn more information from the input data.

Generally speaking, the performance of integrated $T\ell_1$ are better than that of other regularizers. In detail, our model achieves the best results in terms of all three indicators on PENDIGITS and CIFAR-10 among the five sparse regularizers. On MNIST, although the accuracy of the network with integrated $T\ell_1$ is slightly lower than other penalties, the network with integrated $T\ell_1$ is much sparser than those with other convex regularizers. Another non-convex regularizer, i.e. GSℓ_{1-2}, has the highest accuracy while its sparsity is a little worse than that of integrated $T\ell_1$. As for the SDD dataset, the accuracy of our integrated $T\ell_1$ also has a slight decline. Like the performance on MNIST, both the FLOP and parameters used of the integrated $T\ell_1$ are lowest. GSℓ_{1-2} still achieves the highest accuracy. However, their FLOP and parameters used are 10% higher than those of integrated $T\ell_1$.

Next, we verify the acceleration effect of the introduction of sparse regularization terms on the network. We list the time it takes for the model to train a data batch (seconds per batch) and the time for the model to test the whole test dataset (second per test dataset). Since the models constructed on MNIST, Fashion-MNIST, SDD and PENDIGITS are all simple and with small size, the training and test time are short and the acceleration effect of the sparse regularizers is not obvious. Therefore, we implement this experiment on more complex dataset, including CIFAR-10 and CIFAR-100. We list the results in Table 2. As can be seen in Table 2, the training process and the test process are both

Table 2
Training and test time of networks with each regularizer.

Dataset	Measure	ℓ_2	ℓ_1	SGL	CGES	GSℓ_{1-2}	Integrated $T\ell_1$
CIFAR-10	seconds per batch	0.0170	0.0167	0.0174	0.0166	0.0165	**0.0157**
	seconds per test dataset	0.2088	0.2082	0.2094	0.2057	0.2042	**0.2040**
CIFAR-100	seconds per batch	0.4052	0.3955	–	0.3958	–	**0.3953**
	seconds per test dataset	12.235	12.1025	–	12.1073	–	**12.0902**



(a) MNIST, Accuracy/FLOP (b) MNIST, Accuracy/Parameters

(c) PENDIGITS, Accuracy/FLOP (d) PENDIGITS, Accuracy/Parameters

Fig. 4. Accuracy–efficiency trade-off. In order to explore how each regularizer affects the model accuracy at various sparsity levels, we report the accuracy over FLOP and the accuracy over the percentage of parameters used. We obtain the results by varying the regularization parameters.

accelerated due to the introduction of sparse regularization terms and the integrated $T\ell_1$ has best acceleration.

Then, we discuss how the sparsity-inducing regularizer affects the model accuracy. We change the value of regularization parameter to achieve different sparse levels. Networks with ℓ_2, ℓ_1, SGL, CGES, GSℓ_{1-2} and our integrated $T\ell_1$ are considered. In this experiment, we adopt two datasets, i.e. MNIST and PENDIGITS. Results are shown in Fig. 4. As seen from Fig. 4(a), the GSℓ_{1-2} and integrated $T\ell_1$ regularizer largely reduce FLOP with slight decline in accuracy for MNIST dataset while CGES and ℓ_1 can obtain comparable accuracy with more than 45% FLOP. There is no significant decrease in FLOP for the network with SGL, about 80% at least. When we turn to Fig. 4(b), sparsity of parameters adopted in the networks regularized by these penalties has little significant difference on the final prediction accuracy. In Figs. 4(c) and 4(d), we can observe that the performance of integrated $T\ell_1$ is obviously better than other regularizers on PENDIGITS. The FLOP and parameters used in the network regularized by integrated $T\ell_1$ can achieve a quite sparse level with a similar performance. CGES and SGL cannot obtain comparable accuracy when the network is equipped with less than 60% of parameters and 60% of FLOP. Although the integrated $T\ell_1$, GSℓ_{1-2} and ℓ_1 regularized networks all can obtain comparable accuracy when the parameters of network are only 30%, the prediction accuracy of our integrated $T\ell_1$ is obviously better than that of ℓ_1 and GSℓ_{1-2}, to be more specific, 0.975, 0.966 and 0.9574, respectively.

In real-world applications, it will be impractical if a network needs much iterations to converge or does not converge. In this

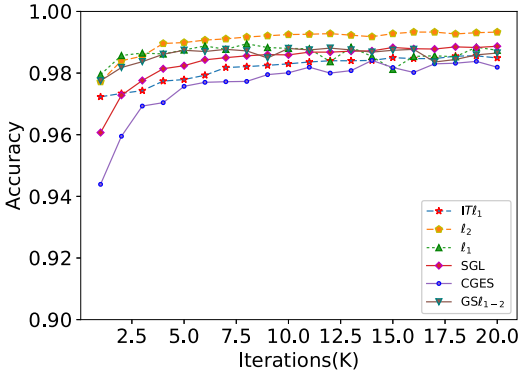


Fig. 5. Convergence speed on MNIST. Convergence of networks regularized by ℓ_2, ℓ_1, SGL, CGES, GSℓ_{1-2} and integrated $T\ell_1$.

experiment, we discuss the empirical convergence speed of the regularizers on MNIST. Fig. 5 displays the test accuracy of each algorithm on MNIST varying with the number of iteration steps. In Fig. 5, it is obvious that the network with ℓ_2 achieves the highest accuracy and the accuracy of networks with sparse regularizers is similar. The reason for this phenomenon is that ℓ_2 has no function to promote sparseness and network with ℓ_2 is dense. Such network can extract more features from the input images. Among networks with sparse regularizers, networks trained by

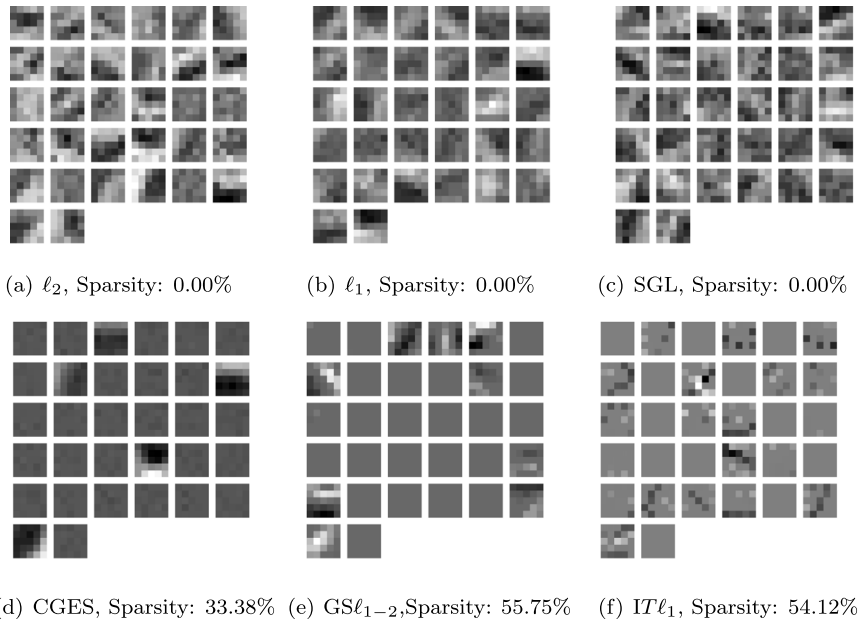


Fig. 6. Visualization of filters of the first convolutional layer for the network trained on MNIST. The ℓ_2 regularizer, ℓ_1 regularizer and SGL regularizer result in smooth non-sparse filters, while CGES obtains filters with a 33.38% sparse level. In contrast, GSℓ_{1-2} and integrated $T\ell_1$ completely remove redundant features and obtain much sharper filters.

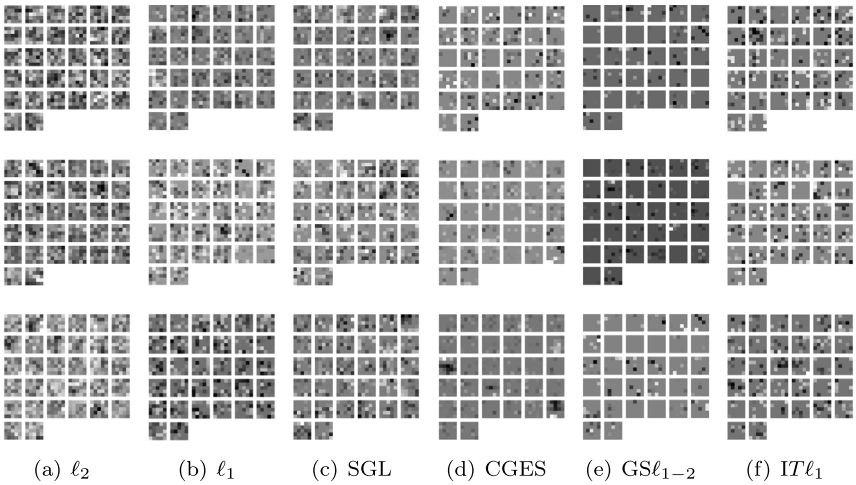
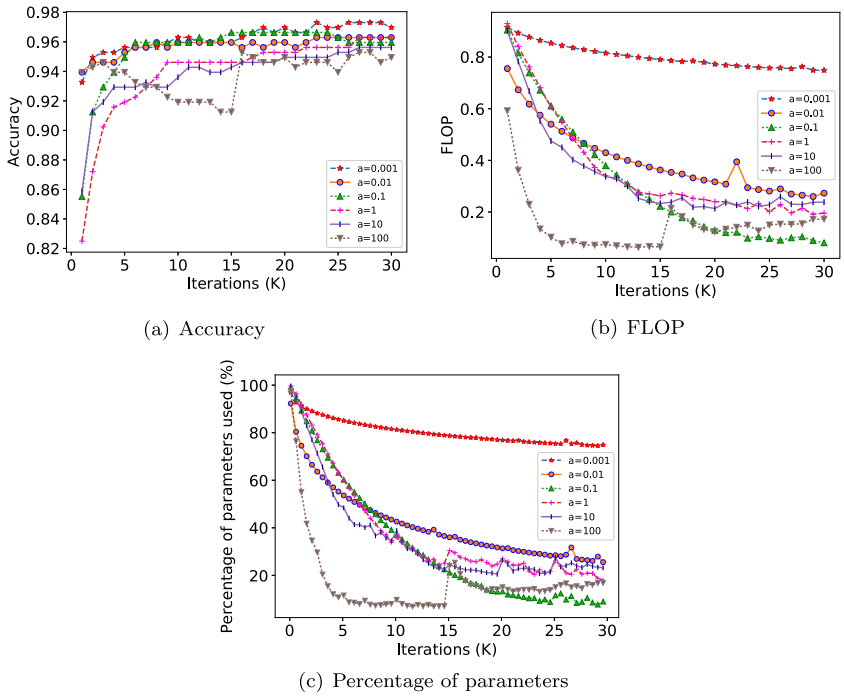


Fig. 7. Visualization of filters of the second convolutional layer for the network trained on MNIST. We flatten the convolutional filters and randomly visualize three of them. The total sparsity of filters in the second convolutional layer of networks trained by above regularizers are 0.00%, 27.18%, 27.89%, 53.17%, 60.68%, 74.66%, respectively.

ℓ_1, GSℓ_{1-2} and integrated $T\ell_1$ can obtain a good accuracy with 1,000 less steps while the accuracy of networks with CGES and SGL is only 0.9439 and 0.9607 at this time, respectively. Besides, the network trained by integrated $T\ell_1$ has the least number of parameters and floating operators, which means that numerous resources will be saved. Therefore, although there is slight drop in accuracy, the integrated $T\ell_1$ will be meaningful for those applications that are executed on resource-constrained platforms.

Next, we visualize the sparsity of filters in the first and second convolutional layers for the network trained on MNIST and display the results in Figs. 6 and 7. In our network architecture, ℓ_2 regularizer cannot promote sparsity. The ℓ_1 regularizer has little effect on the sparsity of filters in the first convolutional layer while it obtains 27.18% sparsity of filters in the second convolutional layer. The SGL also cannot promote sparsity in the first convolutional layer but results in 27.89% sparsity of filters in the second convolutional layer. The CGES results in 33.38% and 53.17% sparsity of filters in the first and second convolutional layers respectively. In contrast, the nonconvex regularizer, i.e. GSℓ_{1-2} and our integrated $T\ell_1$, zero out more spatial features, resulting in much sharper filters, compared with other regularizers. Therefore, there is less redundance among filters as the networks trained with other regularizers. Besides, we can see from Fig. 6 that the filters retained by integrated $T\ell_1$ are richer and more diverse.

In the above subsection, we have demonstrated the extraordinary performance of the proposed integrated $T\ell_1$ when comparing it with other sparse regularizers. Next, we will display that it is also effective for large-scale network architecture and dataset. We choose to train VGG-16 on ImageNet. The ImageNet dataset contains 1,281,167 images which can be classified into 1000 categories for training. 50,000 images are included in the validation dataset. The network we used is the standard VGG-16 network with 13 convolutional layers followed by 3 fully connected layers.



(a) Accuracy (b) FLOP

(c) Percentage of parameters

Fig. 8. The effect of a on the network. The prediction accuracy, FLOP, parameters used of the networks with a varying among $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^{2}\}$.

Table 3
Accuracy on ImageNet under different sparsity.

Model	Accuracy	Parameters used
ℓ_2	0.4997	1
Integrated $T\ell_1$	0.4982	0.8625
Integrated $T\ell_1$	0.4941	0.7170
Integrated $T\ell_1$	0.4883	0.3955

Table 4
Sparsity-promoting performance of each regularizer.

Regularizer	Neurons removed	Sparsity of connections (%)
Group sparsity	64	51.60%
$T\ell_1$	0	61.25%
Integrated $T\ell_1$	12	76.88%

The experimental results are listed in Table 3. As we can see in Table 3, the integrated $T\ell_1$ might result in a slight decline in accuracy because of the removal of some parameters. To be detailed, the integrated $T\ell_1$ can obtain an accuracy of 0.4982 with 86.25% parameters and an accuracy of 0.4883 with 39.55% parameters, 0.0015 and 0.0114 less than the accuracy of dense network, respectively.

4.4. Effect of a in integrated $T\ell_1$

In the experiments, the parameter a in $T\ell_1$ is set in advance. As mentioned previously, when a tends to zero, the $T\ell_1$ approaches the ℓ_0 norm, while $T\ell_1$ approaches ℓ_1 when a is close to infinity. In this subsection, we explore the effect of a in integrated $T\ell_1$ by varying the value of a in the range of $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^{2}\}$ on dataset DIGITS. In all cases, we use a network with one convolutional layer followed by two fully connected layers. For each a, we tune other parameters in the model to obtain the best performance.

We display the curves of prediction accuracy, the FLOP and the percentage of parameters used for networks with different values of a in Fig. 8(a), 8(b) and 8(c), respectively. As seen from Fig. 8(a), the network with $a = 10^{-3}$ achieves the highest accuracy, followed by $a = 10^{-2}$ and $a = 1$. Furthermore, among networks corresponding to the six values of a, the network with $a = 10^{-2}$ converges fastest. The network with $a = 10^{-1}$ achieves the least number of parameters, while it is slightly worse in accuracy than other compared networks. The FLOP and percentage of parameters of other four settings have little difference, varying from 0.2 to 0.3.

4.5. Interpretation of the regularizers

In this subsection, to quantitatively demonstrate the sparsity-inducing capacity of group sparsity, $T\ell_1$ and integrated $T\ell_1$, we study the final layer of networks with these three regularizers on dataset DIGITS. The final layer of the complete network is equipped with 128 neurons. Sparsity-promoting performance of these regularizers are listed in Table 4. As we can see from the table, group sparsity is able to remove 64 neurons and 51.60% connections, indicating that group sparsity can only achieve neuron-level sparsity and the remaining connections are still dense. The $T\ell_1$ cannot remove neurons, but it can remove connections efficiently. Although the network regularized by integrated $T\ell_1$ only removes 12 neurons, it can achieve 76.88% sparsity, which means that the integrated $T\ell_1$ is able to promote both neuron-level and connection-level sparsity simultaneously.

5. Conclusion

In this work, we introduce a new sparsity-inducing regularization called integrated transformed ℓ_1 regularizer, where a group sparsity regularizer explores structural information of neural networks and removes redundant neurons and a transformed ℓ_1 norm enforces sparsity between network connections. We verify the performance of our regularizer on several public datasets. Experimental results demonstrate the effectiveness of the proposed regularizer, when comparing it with five prominent baselines.

There is still some research that we wish to explore in the future. To begin with, in this paper, we only verify the effect of integrated $T\ell_1$ on convolutional neural networks. In the future,

we intend to test integrated $T\ell_1$ on other neural network architectures. In addition, we plan to use other regularizers to replace ℓ_2 in group sparsity to group variables. By doing this, we wish to propose a single regularizer that can remove both redundant neurons and connections simultaneously.

Acknowledgments

This work was supported by National Natural Science Foundation of China under Grant No. 11671379. Thanks to the constructive suggestions from the anonymous referees.

References

Alvarez, J. M., & Salzmann, M. (2016). Learning the number of neurons in deep networks. In *Advances in neural information processing systems* (pp. 2270–2278).

Anwar, S., Hwang, K., & Sung, W. (2017). Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *13*(3), 32.

Bottou, L. (1991). Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, *91*(8), 12.

Bradley, P. S., & Mangasarian, O. L. (1998). Feature selection via concave minimization and support vector machines. In *ICML, vol. 98* (pp. 82–90).

Brazdil, P. B., & Soares, C. (2000). A comparison of ranking methods for classification algorithm selection. In *European conference on machine learning* (pp. 63–75). Springer.

Candes, E. J., Wakin, M. B., & Boyd, S. P. (2008). Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier Analysis and Applications*, *14*(5–6), 877–905.

Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (0000). A survey of model compression and acceleration for deep neural networks. ArXiv preprint. arXiv:1710.09282.

Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2018). Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, *35*(1), 126–136.

Cheng, Y., Yu, F. X., Feris, R. S., Kumar, S., Choudhary, A., & Chang, S.-F. (2015). *An exploration of parameter redundancy in deep networks with circulant projections* (pp. 2857–2865).

Collins, M. D., & Kohli, P. (0000). Memory bounded deep convolutional networks. ArXiv preprint. arXiv:1412.1442.

Cun, Y. L., Denker, J. S., & Solla, S. A. (1989). Optimal brain damage. In *International conference on neural information processing systems* (pp. 598–605).

Deng, L., Yu, D., et al. (2014). Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, *7*(3–4), 197–387.

Denil, M., Shakibi, B., Dinh, L., de Freitas, N., et al. (2013). Predicting parameters in deep learning. In *Advances in neural information processing systems* (pp. 2148–2156).

Donoho, D. L. (2006). For most large underdetermined systems of linear equations the minimal ℓ_1–norm solution is also the sparsest solution. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, *59*(6), 797–829.

Esser, E., Lou, Y., & Xin, J. (2013). A method for finding structured sparse solutions to nonnegative least squares problems with applications. *SIAM Journal on Imaging Sciences*, *6*(4), 2010–2046.

Fan, J., & Li, R. (0000). Variable selection via penalized likelihood, Department of Statistics Ucla.

Fan, J., & Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, *96*(456), 1348–1360.

Fang, Y., Wang, R., Dai, B., & Wu, X. (2015). Graph-based learning via auto-grouped sparse regularization and kernelized extension. *IEEE Transactions on Knowledge and Data Engineering*, *27*(1), 142–154.

Gong, C., Tao, D., Chang, X., & Yang, J. (2017). Ensemble teaching for hybrid label propagation. *IEEE Transactions on Cybernetics*, *49*(2), 388–402.

Gong, C., Tao, D., Maybank, S. J., Liu, W., Kang, G., & Yang, J. (2016). Multi-modal curriculum learning for semi-supervised image classification. *IEEE Transactions on Image Processing*, *25*(7), 3249–3260.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning, vol. 1*. MIT press Cambridge.

Gui, J., Sun, Z., Ji, S., Tao, D., & Tan, T. (2017). Feature selection based on structured sparsity: A comprehensive study. *IEEE Transactions on Neural Networks Learing Systems*, *28*(7), 1490–1507.

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

Hanson, S. J., & Pratt, L. Y. (1989). Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems* (pp. 177–185).

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

Hassibi, B., Stork, D. G., & Wolff, G. J. (1993). Optimal brain surgeon and general network pruning. In *Neural networks, 1993., IEEE international conference on* (pp. 293–299). IEEE.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (0000). Improving neural networks by preventing co-adaptation of feature detectors. ArXiv preprint. arXiv:1207.0580.

Kang, G., Li, J., & Tao, D. (2017). Shakeout: A new approach to regularized deep neural network training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(5), 1245–1258.

Krishnan, D., & Fergus, R. (2009). Fast image deconvolution using hyper-laplacian priors. In *International conference on neural information processing systems* (pp. 1033–1041).

Lebedev, V., & Lempitsky, V. (2016). Fast convnets using group-wise brain damage. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2554–2564).

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436.

Liu, J., Ji, S., & Ye, J. (2009). Multi-task feature learning via efficient $\ell_{2,1}$-norm minimization. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence* (pp. 339–348). AUAI Press.

Lou, Y., Yin, P., He, Q., & 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*(1), 178–196.

Lv, J., & Fan, Y. (2009). A unified approach to model selection and sparse recovery using regularized least squares. *The Annals of Statistics*, *37*(6A), 3498–3528.

Mazumder, R., Friedman, J. H., & Hastie, T. (2011). Sparsenet: Coordinate descent with nonconvex penalties. *Journal of the American Statistical Association*, *106*(495), 1125–1138.

Narang, S., Diamos, G., Sengupta, S., & Elsen, E. (0000). Exploring sparsity in recurrent neural networks. ArXiv preprint. arXiv:1704.05119.

Natarajan, B. K. (1995). Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, *24*(2), 227–234.

Nie, F., Huang, H., Cai, X., & Ding, C. H. (2010). Efficient and robust feature selection via joint $\ell_{2,1}$–norms minimization. In *Advances in neural information processing systems* (pp. 1813–1821).

Nikolova, M. (2000). Local strong homogeneity of a regularized estimator. *SIAM Journal of Applied Mathematics*, *61*(2), 633–658.

Parikh, N., Boyd, S., et al. (2014). Proximal algorithms. *Foundations and Trends® in Optimization*, *1*(3), 127–239.

Poernomo, A., & Kang, D.-K. (2018). Biased dropout and crossmap dropout: Learning towards effective dropout regularization in convolutional neural network. *Neural Networks*, *104*, 60–67.

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

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117.

Shi, Y., Miao, J., Wang, Z., Zhang, P., & Niu, L. (0000). Feature selection with $\ell_{2,1-2}$ regularization, IEEE Transactions on Neural Networks and Learning Systems.

Simon, N., Friedman, J., Hastie, T., & Tibshirani, R. (2013). A sparse-group lasso. *Journal of Computational and Graphical Statistics*, *22*(2), 231–245.

Simonyan, K., & Zisserman, A. (0000). Very deep convolutional networks for large-scale image recognition. ArXiv preprint. arXiv:1409.1556.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, *15*(1), 1929–1958.

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., & Fergus, R. (2013). Regularization of neural networks using dropconnect. In *International conference on machine learning* (pp. 1058–1066).

Xiang, S., Tong, X., & Ye, J. (2013). Efficient sparse group feature selection via nonconvex optimization. In *International conference on machine learning* (pp. 284–292).

Xu, Z. (2010). *Data modeling: visual psychology approach and $L_{1/2}$ regularization theory*. http://dx.doi.org/10.1142/9789814324359_0184.

Xu, Z., Chang, X., Xu, F., & Zhang, H. (2012). $l_{1/2}$ Regularization: a thresholding representation theory and a fast solver. *IEEE Transactions on Neural Networks Learing Systems*, *23*(7), 1013.

Yang, Y., Shen, H. T., Ma, Z., Huang, Z., & Zhou, X. (2011). $l_{2,1}$–norm regularized discriminative feature selection for unsupervised. In *Twenty-second international joint conference on artificial intelligence*.

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

Yoon, J., & Hwang, S. J. (2017). Combined group and exclusive sparsity for deep neural networks. In *International conference on machine learning* (pp. 3958–3966).

Yu, J., Rui, Y., & Tao, D. (2014). Click prediction for web image reranking using multimodal sparse coding. *IEEE Transactions on Image Processing*, *23*(5), 2019–2032.

Yu, J., Zhu, C., Zhang, J., Huang, Q., & Tao, D. (0000). Spatial pyramid-enhanced NetVLAD with weighted triplet loss for place recognition, IEEE Transactions on Neural Networks and Learning Systems.

Yuan, M., & Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *68*(1), 49–67.

Zhang, T. (2009). Multi-stage convex relaxation for learning with sparse regularization. In *Advances in neural information processing systems* (pp. 1929–1936).

Zhang, T. (2010). Analysis of multi-stage convex relaxation for sparse regularization. *Journal of Machine Learning Research (JMLR)*, *11*(Mar), 1081–1107.

Zhang, M., Ding, C. H., Zhang, Y., & Nie, F. (2014). Feature selection at the discrete limit. In *AAAI* (pp. 1355–1361).

Zhang, S., & Xin, J. (2017). Minimization of transformed ℓ_1 penalty: Closed form representation and iterative thresholding algorithms. *Communications in Mathematical Sciences*, *15*(2), 511–537.

Zhang, S., & Xin, J. (2018). Minimization of transformed ℓ_1 penalty: Theory, difference of convex function algorithm, and robust application in compressed sensing. *Mathematical Programming*, *169*(1), 307–336.

Zhang, S., Yin, P., & Xin, J. (2017). Transformed schatten-1 iterative thresholding algorithms for low rank matrix completion. *Communications in Mathematical Sciences*, *15*(3), 839–862.

Zhang, C.-H., et al. (2010). Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, *38*(2), 894–942.

Zhou, H., Alvarez, J. M., & Porikli, F. (2016). Less is more: Towards compact CNNs. In *European conference on computer vision* (pp. 662–677). Springer.