Annals of Data Science (2022) 9(4):695–722
https://doi.org/10.1007/s40745-022-00389-6

A Survey for Sparse Regularization Based Compression Methods

Anda Tang[1] · Pei Quan[2] · Lingfeng Niu[3] · Yong Shi[4]

Received: 10 October 2021 / Revised: 28 February 2022 / Accepted: 2 March 2022 /
Published online: 16 April 2022

Abstract

In recent years, deep neural networks (DNNs) have attracted extensive attention due to their excellent performance in many fields of vision and speech recognition. With the increasing scale of tasks to be solved, the network used is becoming wider and deeper, which requires millions or even billions of parameters. The deep and wide network with many parameters brings the problems of memory requirement, computing overhead and over fitting, which seriously hinder the application of DNNs in practice. Therefore, a natural idea is to train sparse networks and floating-point operators with fewer parameters while maintaining considerable performance. In the past few years, people have done a lot of research in the field of neural network compression, including sparse-inducing methods, quantization, knowledge distillation and so on. And the sparse-inducing methods can be roughly divided into pruning, dropout and sparse regularization based optimization. In this paper, we briefly review and analyze the sparse regularization optimization methods. For the model and optimization method

✉ Lingfeng Niu
niulf@ucas.ac.cn

✉ Yong Shi
yshi@ucas.ac.cn

Anda Tang
tanganda17@mails.ucas.ac.cn

Pei Quan
quanpei17@mails.ucas.ac.cn

1 School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100190, China

2 School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049, China

3 School of Economics and Management, University of Chinese Academy of Sciences, Beijing 100190, China

4 Research Center on Fictitious Economy and Data Science, Chinese Academy of Sciences, Beijing 100190, China

of sparse regularization based compression, we discuss both the different advantages and disadvantages. Finally, we provide some insights and discussions on how to make sparse regularization fit within the compression framework.

Keywords Deep neural networks · Sparsity learning · Compression

1 Introduction

Now we have entered the big data era. Big data provides great opportunities for academia and industry. It is a reality that no one can ignore and our environment whenever we need to make decision [1]. The meaning of big data contains both data science and applications such as business data mining or artificial intelligence [2, 3], where big data analysis, across data science and applications, is also a subset of big data [4]. Facing the challenges of big data, the core of big data is intelligence still where deep learning is a focus [5].

Deep learning is a branch of machine learning. It automatically extracts data features layer by layer through multi-layer network structure, and uses parameters and network structure design algorithms to make the network have the ability to fit any complex nonlinear mapping. So far, deep learning has made an important progress in pattern recognition [6], computer vision [7], speech recognition and natural language processing [8], etc. Nowadays, the research on deep learning is in the ascendant. The major technology companies in the industry are competing to occupy the commanding height of deep learning technology [9]. The excellent performance of deep learning is inseparable from its powerful expression ability. In order to achieve a strong representation ability, deep learning needs to rely on the multi-layer structure and a large number of parameters. With the increase of data, the improvement of learning accuracy and the complexity of learning tasks, the number of network layers and the amount of parameters are increasing. Taking the famous Imagenet image recognition contest as an example, the champion algorithm AlexNet [7] in 2012 only constructed five convolution layers and three fully connected layers, which required more than 60 million parameters; while the champion algorithm ResNet [10] in 2015 achieved 96.43% accuracy through 152 layers of deep neural network, which successfully exceeded the human visual ability (94.9%) in the field of visual recognition. In the language model, the Bert [11] proposed by Google in 2019 has more than 300 million parameters, and the gpt-3 proposed by openai in 2020 has 175 billion parameters. Despite the rapid development of deep learning, the state-of-the-art algorithm in a series of fields has made great progress, but it is still restricted by time and space in practical application. Deep networks have a large amount of computation, even with the help of graphics processing unit (GPU) acceleration, it still can not meet the needs of many application scenarios in time. In addition, large-scale model also take up a lot of memory space, which is not suitable for mobile devices such as mobile phones. Therefore, how to compress the deep neural network and improve the efficiency of deep learning is the key problem that must be solved in the process of deep learning.

There are many kinds of methods to compress the network, among which sparsity is the main method of compression. Iterative pruning [12] is proposed to achieve sparsity

by iteratively remove connections which are uncritical. The work [13] makes neurons sparse by employing sparsity-inducing priors for hidden units. Structured Sparsity Learning (SSL) method [14] is proposed to directly learn a compressed structure of DNNs by group Lasso regularization during the training. The work [15] studies Variational Dropout [16] in the case when each weight of a model has its individual dropout rate and proposes Sparse Variational Dropout that extends Variational Dropout to all possible values of dropout rates and leads to a sparse solution. The work [17] proposes a solution through the inclusion of a collection of non-negative stochastic gates for ℓ_0 norm regularization for neural networks. A lot of sparse methods to compress the network have been proposed. We classify these methods into three categories, including pruning, dropout and sparse regularization based compression. Among these methods, this article focuses on sparse regularization based compression which is with both highly optimized implementations and approximate methods investigated [18]. The sparse regularization based compression has many advantages. On the one hand, this methodology does not need too many assumptions and is easy to apply. On the other hand, the sparse regularization has a good theoretical basis of statistical learning, which transforms the maximum likelihood into a posteriori probability estimation, and introduces a reasonable prior in the case of insufficient data, which can improve the effect of the model. We represent the details of sparse regularization based compression, including their general model, sparse-inducing regularizer and optimization methods in the following sections.

The rest of our paper is organized as follows. In next section, we review the literature of network compression from a global approach Sect. 2. In Sect. 3, we will represent the details of sparse regularization based compression, especially sparse-inducing regularizers. In Sect. 4, commonly-used optimization methods for corresponding sparse regularization based compression are introduced.

2 Compression for DNNs

Recently, deep neural networks have become the dominant models for heterogeneous computer tasks including but not limited to pattern recognition [6], computer vision [7], speech recognition and natural language processing [8]. To deal with various difficult tasks, one habitual method is to deepen the networks to confront more complex data or to improve performance, resulting in a large number of parameters in the deep network model. The network needs a lot of storage space and computing resources, so the model can not be applied to devices with limited resources or real-time requirements. Therefore, the compression and acceleration of neural network is a very important research direction. Compression requires joint knowledge from multiple disciplines, such as machine learning, optimization, computer architecture, signal processing and hardware design, etc. The existing neural network model compression methods can be divided into four categories: low-rank decomposition, compact convolution filter design, knowledge distillation and compression from a parametric perspective. Then, we further review the literature of these four categories of compression in the following content (Fig. 1).

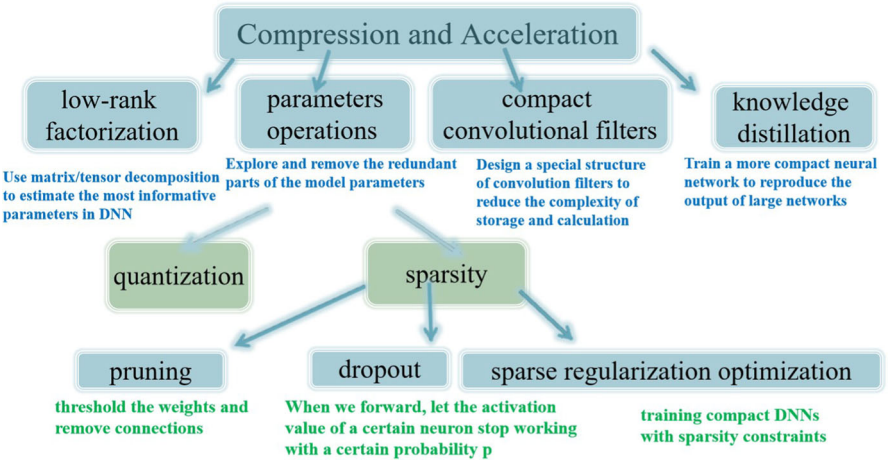

Fig. 1 Network-compression

2.1 Low-Rank Decomposition

The low rank decomposition method mainly uses the decomposition of matrix or tensor to estimate the meaningful parameters in neural network. In deep convolution neural networks, most of the computation comes from convolution operation. For example, if a convolution network has l convolution layers, the number of filters in layer l is n_l, the size of the filter is s_l, the size of the active graph is m_l, the sum of the computational complexity of all the convolution layers can be denoted as $O(\sum_{l=1}^{L} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2)$ [19]. Therefore, reducing the amount of computation of the convolution layer can speed up the overall computing speed of the network. Convolution kernel can be regarded as a 4-dimensional tensor which usually has a great redundancy. Tensor decomposition is an effective method to remove this redundancy. The fully connected layer can be regarded as a two-dimensional matrix, and matrix decomposition is also effective for it.

Based on this methodology, many decomposition techniques for matrices and tensors are applied to compress and accelerate neural networks, including SVD [20–22], Canonical Polyadic Decomposition [23], Tucker Decomposition [24], Block Component Decomposition [25, 26] and many other tenser decomposition [27, 28]. Sainath et al. [29] decompose the last weight layer into low rank matrices. Denton et al. [20] approximate tensors in convolutional layers by SVD(singular value decomposition). Jaderberg et al. [30] exploit cross-channel or filter redundancy and compress filters by constructing rank-1 filters. Ioannou et al. [31] train base filters and combine these base filters into complex ones. Wen et al. [32] compress CNNs using a low-rank regularization. Yu et al. [33] approximates the weight matrix as the sum of a low rank matrix and a sparse matrix.

The low rank decomposition based method is a very direct way for model compression and acceleration. However, decomposition consumes a lot of calculation and is not simple to implement. The model needs fine-tuning and retraining to achieve con-

vergence. In addition, it can not perform global parameter compression since different layers have different information. At last, it does not work for some networks with small convolution kernel.

2.2 Compact Filter Design

The compact filter design is to save parameters by designing structural convolution filter, so as to economize storage. Taco [34] points out that the convolution layer can be effectively used in deep networks because these layers are translation invariant. According to the translation invariance, some algorithms design specific transformations and apply them to layers or filters to compress the network. Zhai et al. [35] define the transformation as a set of transform functions which are applied to 2-dimensional filters. Li et al. [36] introduce a multi-bias non-linear activation to generate more modes without adding too much calculation through the transformation. Shang et al. [37] also use a new activation called CReLU. CReLU function saves the positive and negative linear returns after convolution, so that each filter can effectively represent their direction. Sander et al. [38] propose four operations to establish rotation invariant neural networks, including slicing, pooling, rolling and stacking. However, the application of this kind of method is limited in some wide network structures such as VGGNet, but can not achieve comparable performance on narrow or some special networks such as GoogLeNet and ResNet. At the same time, the assumptions of this method make the results obtained on some data sets unstable [39]. Therefore, some studies [40, 41] consider designing compact convolution filter to replace the loose filter with redundant parameters, which can directly reduce the computational consumption and improve the computational speed. Christian et al. [40] replace per 3×3-size filter with two 1×1 filters. Wu et al. [41] propose SqueezeDet with two 1×1 filters.

2.3 Knowledge Distillation

The basic idea of knowledge distillation is to extract knowledge from a large network which is called teacher network and learn a small network which is called student network, and makes the student networks have the knowledge extracted from the teacher networks. The seminal work [42] compresses the networks exploiting knowledge transfer, but this method can only be applied to shallow networks. Jimmy et al. [43] prove that the shallow network can also learn the complex functions learned by the deep network, and obtains the accuracy that only the deep network can achieve before. It adopts the idea of knowledge transfer to mimick the deep networks by shallower ones, which is referred to as knowledge distillation. Hinton et al. [44] introduce a student-teacher paradigm as knowledge distillation compression framework. The soft labels corresponding to the training samples are obtained by training the teacher networks. Combined with the real label, the soft labels are used to calculate the loss function of the student networks. Romero et al. [45] divide the training process into two stages. In the first stage, the middle layer information of teacher networks are used to train the parameters in the student networks. In the second stage, the final output of the teacher networks to train all the parameters of the student networks to

get a deeper but narrower network. Luo et al. [46] compress the model in the face recognition tasks. It uses the characteristics of the data sets, and exploit the network output of the previous layer of softmax as information to select neurons. Chen et al. [47] first trains a small network, and then transfer the knowledge to a larger network. It uses the trained weight of the small network to accelerate the speed of training a large network. Zagoruyko et al. [48] improve FitNets and merge the attention mechanism to transfer the attention map as knowledge from the teacher model to the student model. Yim et al. [49] represent the data flow relationship between the two network layers as a flow of solution procedure (FSP) matrix and transfer it as knowledge. And the knowledge is added to the optimization process to reduce the FSP difference between the corresponding layers between teacher networks and student networks. In the object detection problem, Chen et al. [50] uses fitnets combined with a new loss function to extract knowledge dealing with sample imbalance and other problems. Huang et al. [51] utilize the consistency of activation distribution of student network and teacher network to conduct the training of student network.

2.4 Quantization

The model quantization method compresses the original network by reducing the number of bits required to represent each weight. The parameters of the network are generally represented by 32-bit floating-point numbers, but in fact, it is not necessary to retain such high bit-number. The bit-number required for stochastic gradient descent method is only 6 to 8 bits. The quantization neural network compresses the scale of the network and speeds up the training and inference process by using a small number of bits, such as 8 bits or even 1 bit. The calculation operation in quantization neural network is usually bit-by-bit operation, which is performed by arithmetic logic unit (ALU), which consumes much less energy than floating-point unit (FPU). Therefore, for mobile devices whose power consumption is limited, such as mobile phones, quantitative neural network is better than full precision network.

In neural networks, there are three components that can be quantified: weight, activation and gradient. By quantifying the weight and activation, a low-bit network model can be obtained. In the distributed training environment, the communication cost can be saved by quantifying the gradient. Generally speaking, quantifying the gradient is more difficult than quantifying the weight and activating the output, because a high-precision gradient is required to converge the optimization algorithm.

Gong et al. [52] propose vector quantization to quantify the neural networks. The basic idea of vector quantization is to cluster the weights into several groups, and then use the cluster center to replace other weights in each group. In this way, only indexes and cluster centers need to be stored, which reduces the memory requirement. However, the accuracy loss caused by k-means cluster can not be controlled, and compression ratio constraints can not be imposed either. In order to solve these problems, Choi et al. [53] propose a k-means cluster method weighted by Hessian matrix. The basic idea of this method is to use Hessian to measure the performance degradation caused by weight quantization. In this way, the weights that have a great importance on the network performance can be prevented from excessively deviating from their

full-precision values. Wu et al. [54] also adopt k-means cluster method to quantify the filters and the weights in fully connected layers. The responses of the convolution layer and the fully connected layer are estimated by inner product approximation. The performance is improved by reducing the estimation error of the responses of each layer during quantization process. Park et al. [55] propose a method of weights cluster based on weight entropy. In this method, there are more clusters around the important weights so that a more automatic and flexible multi-bit quantization can be realized. These methods represent parameters in the cluster through the cluster center, so as to reduce the required storage. However, one disadvantage of quantization using k-means cluster is that the calculation is very expensive due to the large number of network weights in deep neural networks.

Some methods focus on reducing the number of bits used to represent per parameter, so as to reduce the memory requirements and computation of the model. Vanhoucke et al. [56] employ 8-bit quantization for the activation, and limit the activation to the range of 0 to 1 by a sigmoid function. Gupta et al. [57] take 16-bit representation in the convolution neural network training based on stochastic rounding, which significantly reduces the memory requirements and floating-point operations. Micikevicius et al. [58] still save the weights as 32 bits, but truncates them to 16 bits for calculation during training. Wu et al. [59] propose a heuristic rounding function to quantify the full-precision floating-point weight into a k bits integer. Seide et al. [60] propose a subgradient representation with 1 bit quantization, so as to speed up the calculation. The extreme case of 1-bit quantization of weights is weight binarization. Courbariaux et al. [61] propose BinaryConnect which trains the network with binary weights during forward and back propagation. In the forward propagation stage, the algorithm quantifies the real weight through the symbolic function, and uses the quantized weight to generate the output. In the back propagation stage, the symbolic function can no longer be used to back propagate the loss, because the gradient is almost zero everywhere. The usual approach is to use a heuristic method to estimate the gradient. Courbariaux et al. [62] further propose BinaryNet based on BinaryConnect which quantifies the network weights and limits the weights to -1 or $+1$. Storm et al. [63] propose a threshold quantization method to quantify the gradient, that is, with a fixed threshold selected in advance, the gradient above the threshold is quantified as $+1$, the gradient below the threshold is quantified as 0. Rastegari et al. [64] propose the Binary-Weight-Network(BWN) and the XNOR-Network. BWN only binarizes the network weights, while XNOR network binarizes the network weights and network inputs. And XNOR has significantly more acceleration effect than BWN. Hou et al. [65] consider the influence of binarization on the loss function, and proposes a loss-aware binarization. Kim et al. [66] propose a two-step binarization. In the first step, the weight is compressed to the range of -1 to 1. In the second step, the compressed weight is used to initialize the parameters of a binary network. Hu et al. [67] train binary weight networks via hashing. Lin et al. [68] believe that the binary representation is not enough to comprehensively represent the weights of neural networks, and propose ternary weight networks added by the value of 0. Li et al. [69] propose ternary weight networks with -1, 0 and 1. Zhu et al. [70] introduce two full-precision scaling coefficients for each layer, and then quantify the weights to one of the two coefficients and 0, rather than between $+1$, -1 and 0. Mellempudi et al. [71] utilize multiple scaling factors to con-

sider the asymmetry between positive and negative weights in the ternary networks. Deng et al. [72] propose gated XNOR networks which train deep neural networks with ternary weights and activations. The calculation trigger determined by weight and activation turns on the calculation as a control signal or control gate in the network. Wang et al. [73] propose a semi discrete decomposition of weight matrix. It decomposes $W \in \mathbb{R}$ into the multiplication of $X \in \{-1, 0, +1\}^{m\times k}$, $Y \in \{-1, 0, +1\}^{n\times k}$ and non-negative diagonal matrix $D \in \mathbb{R}^{k\times k_+}$. By choosing a different k, a trade-off can be achieved between compression ratio and performance loss. Wen et al. [74] introduce the TernGrad method, which quantifies the gradient into $\{-1, 0, 1\}$

In addition, some other quantitative methods have been proposed. Muller et al. [75] draw lessons from the idea of integer programming and propose a stochastic rounding function to map the real value to the nearest discrete point or the second nearest discrete point based on the distance from the corresponding point. Polino et al. [76] propose a uniform quantization. Given a parameter s in advance. Uniform quantization generates $s + 1$ equally spaced points between 0 and 1, and quantifies each real value as the nearest quantization point. Koster et al. [77] quantify networks in the training process, share the exponential bits of the stored data, and transforms the floating-point operation in the training process into the integer operation of the mantissa, so as to accelerate the network training. Cai et al. [78] propose low precision Relu by half-wave gaussian quantization. Mishra et al. [79] propose wide reduced precision networks (WRPN) to quantify the activations and weights. This paper points out that the activations actually occupies more memory than the weight, so it adopts a strategy, that is, increasing the number of filters in each layer to compensate for the decline in accuracy caused by quantization operation.

2.5 Sparse-Inducing Methods

The sparse-inducing methods seek the sparsity in dense networks. It trains a sparse network with smaller number of weights, connections, filters or neurons compared with the original network to reduce the memory demand and computation. Denil et al. [80] points out that most parameters in neural networks are redundant. Therefore, it is feasible to train sparse networks with comparable performace. The neural networks with a sparse structure and fewer connections need less memory space to store these weight data, and the compressed networks require less computation when applied to new data. In this subsection, the existing network sparsity methods are mainly divided into three categories: pruning, dropout and sparse optimization methods.

2.5.1 Pruning

The main idea of pruning method is to remove those unimportant connections that have little impact on the final output of the network in the process of network training. Assuming the neural network architecture is $f(x, \cdot)$, where x is the input sample. Such an architecture includes the parameters of the network and the operators it uses, such as convolution, pooling, batch normalization, etc. For the specific parameter W, the neural network is denoted as $f(x, W)$. The pruning of neural networks actually

takes $f(x, W)$ as input and generates a new model $f(x, \hat{W})$ as output. $\hat{W}$ is the parameter set after pruning. In a pruning operation, it can be denoted as $M \odot W$, where $M \in \{0, 1\}^{|W|}$ is a binary mask to change some specific parameters to 0, $\odot$ is an element-wise multiplication operation. In practice, instead of using a mask, the parameter is directly set to 0 or removed. Generally speaking, pruning methods follow such a framework,

1. Initializing parameters W, $M \in \{1\}^{|W|}$ and training $f(x, W)$ to convergence.
2. According to an evaluation criterion $S(\cdot)$, scoring the existing parameter set, and update the next pruning rule according to the current score and pruning rule, that is $M \leftarrow P(M, S(W))$.
3. Fine-tuning the network $f(x, M \odot W)$.

In the last century, several early pruning methods were proposed. Biased weight decay [81] is an early work for pruning. A magnitude based pruning method is proposed to apply weight decay related to its absolute value to each hidden neuron in the network to reduce the number of hidden units. Subsequently, optimal brain damage(OBD) [82] and optimal brain surgery(OBS) [83, 84] take the Hessian matrix of the loss function into consideration. These two methods get higher accuracy than biased weight decay. However, the complexity of calculating the Hessian matrix is $O(|W|^2)$ so that it is not feasible for current networks which are deeper and with more parameters and neurons.

In recent years, pruning methods have regained attention. Many works have been proposed to prune redundant network connections which do not provide useful information for the network output. Srinivas et al. [85] observe that similar neurons are redundant. It calculates a saliency value between two neurons to represent similarity. And then it removes one of the neurons with the smallest saliency. Han et al. [86] consider that the weights whose value are less than a certain threshold are redundant. It removes these weights from the network to obtain a sparse network. Finally, the network obtained is retrained and the weights are fine-tuned to ensure that the model performance will not be affected by the removal of some connections. They also combine this pruning method with quantization method and Huffman coding to compress the networks [87]. Molchanov et al. [88] propose a pruning standard based on Taylor expansion, which approximates the change of loss caused by pruning network through first-order Taylor expansion. Anwar et al. [89] introduce the particle filter method into structural pruning. The importance of each particle is determined by calculating the error classification rate. And it removes the particle with lower importance to achieve structurally pruned networks. And It is becoming increasingly popular to focus on setting different pruning standards on neuron-level. Narang et al. [90] propose to prune with a monotonically increasing threshold and the sparsity rate can be controlled by setting the shape of the threshold function. Lin et al. [91] propose a global and dynamic pruning strategy. Specifically, the model uses a global discriminant function established based on a priori knowledge to globally prune the filter that is not significant, and then dynamically update the significance value of the filter based on the pruned sparse network, recover the proper filter, and then retrain the model to improve the accuracy of the model. Molchanov et al. [92] propose an adaptable structural pruning without requiring per-layer sensitivity analysis.

2.5.2 Dropout Methods

Dropout methods randomly discard some neurons and connections of networks in the training process. It is worth noting that dropout is proposed not only as a sparse technique of neural network, but also as a regularization to reduce the overfitting problem in the process of network training.

Dropout was first proposed by Hinton et al. [93]. In each training process, each hidden neuron is randomly removed from the network with a probability of 0.5, so that the existing neurons cannot rely on removed hidden neurons. Dropout can also be regarded as training a large number of different networks in a reasonable time. There are different networks for each representation of each training case. Extensive stochastic methods are inspired by this original dropout and we refer to them by dropout method in general. DropConnect [94] extends Dropout [93] to the connections of the fully connected layers. Adaptive dropout [95] substitute the constant dropout rate by the adaptive rate in Dropout [93]. Srivastava et al. [96] comprehensively analyse dropout. Dropout is found to promote a sparse distribution of weights. Kingma et al. [16] propose variational dropout which sets an independent inactivation rate for each layer, each neuron and even each weight. Molchanov et al. [15] extend variational dropout [16] and build a bridge between pruning and dropout via sparse distribution. Poernomo et al. [97] propose Biased Dropout and Crossmap Dropout on hidden layers and convolutional layers.

In general, dropout methods are originally an algorithmic mechanism used to avoid overfitting. On one hand, its algorithm framework background is conducive to being applied to a wide range of neural network topologies. On the other hand, since it has good mathematical and statistical properties, it can be expanded to a compression applications. It can be seen as a sparse distribution of weights, which narrows the gap between pruning and sparse regularizaiton.

2.5.3 Sparse Optimization

The method based on sparse optimization is to transform the sparse learning process of neural network into an optimization problem by introducing sparse regularization terms. By solving an unconstrained sparse optimization problem, the weight of some network connections will be set to zero. And the training objective function can be denoted as,

$$\min_{W} \ \mathcal{L}(f(\{W^{(l)}\}_{l=1}^{L}), D) + \lambda \sum_{l=1}^{L} \Omega(W^{(l)}) \tag{1}$$

where $W^{(l)}$ is the weights matrix of l-th layer, $\{W^{(l)}\}$ is the set of all weights matrices. $D = \{x_i, y_i\}_{i=1}^{N}$ is the dataset containing N samples, x_i is the i-th input sample $y_i \in \{1, \ldots, K\}$ ia the corresponding label. $\lambda^{(l)}$ is the hyper-parameter to balance the effect between $\mathcal{L}\left(\{W^{(l)}\}, \cdot\right)$ and regularization term $\Omega(W^{(l)})$.

The method based on sparse optimization mainly designs different sparse regularizaiton terms to maintain the prediction accuracy and sparse the network at the same

time. [98] studies the use of three sparse regularization terms as sparse constraints to sparse neural networks, including ℓ_1, shrinkage operator and projection to ℓ_0 ball. The model training process using these regular functions can still be performed by the stochastic gradient descent method, which makes it easy for the algorithm to call the existing code. Zhou et al. [99] propose a method which tensor low rank constraint and group sparsity are applied to the objective function of deep neural network to remove redundant neurons. The tensor low rank constraint is realized by the trace norm of the tensor. The trace norm of tensor is proposed by Liu et al. [100]. The average rank of tensor is approximated by calculating the average value of trace norm of different expansion modes of tensor. For the tensor connected to a neuron $\widetilde{\mathbf{w}}_{Neu}$, whose trace norm can be denoted as follows,

$$\Omega(W^{(l)}) = \left((1-\alpha)\sqrt{P^{(l)}} \sum_{n=1}^{N^{(l)}} \|\omega_n^{(l)}\|_2 + \alpha \|W^{(l)}\|_1 \right), \tag{2}$$

where $P^{(l)}$ is the neuron size in l-th layer, $N^{(l)}$ is the neuron number in l-th layer, $\omega_n^{(l)}$ is the corresponding weight matrix of n-th neuron in l-th layer, α is a balance parameter. And then a majority of regularization based methods focus on utilizing various regularizer $\Omega(\cdot)$ [99, 101–105], which we will further introduce regularizer in the next section.

Sparse regularization based compression has a strong mathematical and optimization background. It can be related with sparse priori distributions. Therefore, it has a certain interpretability in theory. Moreover, the capability to prevent overfitting make regularization based compression can achieve comparable accuracy for model. In addition, it is easy to practical application.

3 Sparse Regularizers

Compression with sparse regularizer based approaches in DNNs obtain sparsity through turning the training process into an optimization problem. The compression is obtained by introducing sparse regularization to the objective function [99, 106–109]. In general, optimization problem for network compression can be formulated as follows,

$$\min_{W} \ \mathcal{L}(f(W), D) + \lambda \Omega(W), \tag{3}$$

where $f(\cdot)$ is the neural network, D is the dataset, W is the weights, $\mathcal{L}(\cdot, \cdot)$ is the data fidelity term, $\Omega(\cdot)$ is the regularization term and $\lambda > 0$ is the hyper-parameter.

Many studies focus on applying sparse-regularization based compression by designing the regularization term $\Omega(\cdot)$ [99, 101–105]. ℓ_0 norm is the most intuitive sparse constraint. When it acts on a vector, its output is the number of non-zero elements. Intuitivly, ℓ_0 norm is the strictest sparse-inducing constraint, and the most sparse solution can be obtained. However, minimize a ℓ_0 norm constraint problem is usually NP

hard. ℓ_1 norm is a convex relaxation of ℓ_0 norm. It is commonly used as it is convex and easy to be operated. The ℓ_1 norm [108, 110, 111] of a vector $x \in \mathbb{R}^N$ is defined by

$$\|x\|_1 = \sum_{i=1}^{N} |x_i|. \tag{4}$$

In [98], sparse regularizers including the ℓ_1 regularizer are applied to both convolutional and fully-connected structures. Some methods [108, 111] use ℓ_1 to promoted weight sparsity, which is to remove connections of a well-trained network.

Though the ℓ_1 norm has many advantages, it is sensitive to outliers and may cause serious bias in estimation [112]. That means these methods may need to sacrifice accuracy to achieve a comparable compression rate. After that, some works focus on improving the performance of regularization, which can be categorized as two types. The first kind of methods pay attention to the properties of regularization terms, so as to select better regularization terms. It is pointed out in [112] that a good regularization term should obtain an estimation with the following three characteristics: unbiasedness, sparsity and continuity. The so-called unbiasedness means that for variables with a large proportion (such as the previous parameters), the estimated value should be asymptotically unbiased, so as to avoid excessive model deviation. Sparsity means that small variables can be automatically estimated to zero to obtain sparse solutions, which can reduce the complexity of the model. Continuity means that the obtained estimation should maintain continuity, so as to maintain the stability of the model. These characteristics provide a reference for the selection of regularization functions. Obviously, regularization functions with such characteristics should be nonconvex.

Smooth clipped absolute deviation (SCAD) is the first regularization function [112] proved to meet these characteristics in order to improve ℓ_1 and hard threshold functions are proposed. It effectively combines the soft threshold function with the hard threshold function. Its definition on the vector $\mathbf{x} = \{x_1, x_2, \ldots, x_N\} \in \mathbb{R}^N$ is as follows,

$$\mathbf{P}(\mathbf{x}; \lambda, \gamma) = \sum_{i=1}^{n} P(x_i; \lambda, \gamma),\ P(x_i; \lambda, \gamma) = \begin{cases} \lambda |x_i|, & if\ |x_i| \le \lambda \\ \frac{2\gamma\lambda|x_i| - x_i^2 - \lambda^2}{2(\gamma - 1)}, & if\ \lambda < |x_i| < \gamma\lambda \\ \lambda^2(\gamma + 1)/2, & if\ |x_i| \ge \gamma\lambda, \end{cases} \tag{5}$$

where $\lambda > 0$and $\gamma > 2$ are two adjustment parameters, usually $\gamma = 3.7$. As can be seen from the above formula, for scalar variable $|x| \le \lambda$, SCAD is equivalent to ℓ_1 , and then it smoothly transforms into a quadratic function until $|x| = \gamma\lambda$. Then, for all $|x| > \gamma\lambda$, it is equal to a constant, so as to meet the approximate unbiased estimation of variables.

SCAD has been proved the following two properties:

(a) it can select the correct subset of variables.
(b) parameter estimation is asymptotically normal, and the variable estimation can be unbiased by controlling the parameters.

And then the minimum maximum concave penalty (MCP) is proposed in [113] and is defined as follows,

$$\mathbf{P}(\mathbf{x};\lambda,\gamma)=\sum_{i=1}^{n}P(x_i;\lambda,\gamma),\ P(x_i;\lambda,\gamma)=\begin{cases}\lambda|x_i|-x_i^2/(2\gamma), & if\ |x_i|\le\gamma\lambda\\ \gamma\lambda^2/2, & if\ |x_i|>\gamma\lambda\end{cases} \tag{6}$$

where $\lambda > 0$and $\gamma > 1$. It can be seen from (6) that when $\gamma \to \infty$, MCP function tends to ℓ_1 regularization, the sparsity-inducing ability becomes weaker; When $\gamma \to 1$, MCP function approaches ℓ_0 regularization, and the sparsity-inducing ability becomes stronger. Similar to SCAD, when the value of the variable is greater than a certain value, the value of MCP will become a constant. And MCP can also obtain unbiasedness, sparsity and continuous variable estimates. Different from SCAD, MCP directly relaxes the penalty rate to zero in the later stage, while the penalty rate of SCAD remains unchanged for a period before it decreases.

Although SCAD and MCP can obtain unbiased estimation, they are in the segmented form and the model is relatively complex. As a result, they undoubtedly increases the amount of calculation in practice. Especially when the applied model itself is complex and the amount of calculation is large, the processing of these two parameters will increase the amount of calculation to the model.

Capped ℓ_1 function is another approximate form of ℓ_0 to better reduce the influence of noise and outliers [114],

$$\mathbf{P}(\mathbf{x};a)=\sum_{i=1}^{n}\min(|x_i|,a),a>0. \tag{7}$$

It consists of two segments. As seen in Eq. (7), when $a \to 0 \sum_i \min(|x_i|, a)/a \to \|\mathbf{x}\|_0$. It can better approximates ℓ_0 than ℓ_1. When the absolute value of the variable is less than the parameter a, Capped ℓ_1 function corresponding to ℓ_1. And when the absolute value of the variable is greater than the parameter a, Capped ℓ_1 function corresponding to a constant, which means that the noise term with large error will be truncated by Capped ℓ_1. Thus Capped ℓ_1 is able to avoid the bias of variable estimation, which is more robust to noise.

The elastic net is a combination of ℓ_1 regularization function and ℓ_2 regularization function [115],

$$\mathbf{P}(\mathbf{x};\alpha)=\sum_{i=1}^{n}\left((\alpha-1)x_i^2/2+(2-\alpha)|x_i|\right),\ 1\le\alpha\le 2, \tag{8}$$

where α is a non negative parameter. When $\alpha = 1$, the elastic net function becomes ℓ_1 function, and when $\alpha = 2$, the elastic net function becomes the ℓ_2 function. Logarithmic penalty function is a generalized form of elastic nets [116],

$$\mathbf{P}(\mathbf{x};\gamma)=\sum_{i=1}^{n}P(x_i,\gamma),\ P(x_i;\gamma)=\frac{\log(\gamma|x_i|+1)}{\log(\gamma+1)}, \tag{9}$$

where the parameter $\gamma > 0$. And by changing the value of γ, the continuum from ℓ_1 ($\gamma \to 0_+$) to ℓ_0 ($\gamma \to \infty$) can be obtained.

The ℓ_1/ℓ_2 regularization function has been applied as a sparsity-inducing regularizer in many fields, such as non-negative matrix factorization [117], blind deconvolution [118], image deblurring [119]. Without the range constraint, this regularization in these works adopts the form as follows,

$$\mathbf{P}(\mathbf{x}) = \frac{\sum_{i=1}^{n} |x_i|}{\sqrt{\sum_{i=1}^{n} x_i^2}} \tag{10}$$

It satisfies five desired heuristic criteria of sparsity measures [120]. And It is differentiable almost everywhere and scale-invariant so that it is utilized in learning sparser neural networks [121].

The ℓ_{1-2} [122] regularization is the difference of ℓ_1 and ℓ_2 norm and widely used in many problems. Its relationship with ℓ_1/ℓ_2 regularization function can be denoted as $\|x\|_1 - \|x\|_2 = \|x\|_2(\frac{\|x\|_1}{\|x\|_2} - 1)$. The ℓ_{1-2} is nonconvex and Lipschitz continuous, and the corresponding optimization algorithms are effective. It performs very well in spectral imaging [122], compressed sensing [123], sparse signal recovery [124]. This regularization function has no hyper-parameters and is simple in form, but it is also lack of adaptability to different tasks due to the absence of hyper-parameters.

Transformed ℓ_1(Tℓ_1) regularization function [125, 126] is a smooth form of Capped ℓ_1, which has many good properties, such as unbiasedness, sparsity, Lipschitz continuity and so on. For a single variable x, its definition formula as follows,

$$\rho_a(x) = \frac{(a+1)|x|}{a + |x|} \tag{11}$$

where $a > 0$ is a parameter which controls the shape of the function. When $a \to 0$, $\rho_a(x)$ approaches the indicator function as follows,

$$I(x) = \begin{cases} 1, & if\ x \neq 0 \\ 0, & if\ otherwise. \end{cases} \tag{12}$$

and when $a \to \infty$, $\rho_a(x)$ approaches the ℓ_1 function. And Zhang extends Tℓ_1 to vector space, for a vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\} \in \mathbb{R}^n$, T$\ell_1$ is defined as follows,

$$T\ell_1(\mathbf{x}) = \sum_{i=1}^{n} \rho_a(x_i). \tag{13}$$

and noticing that

$$\lim_{a \to 0^+} T\ell_1(\mathbf{x}) = \sum_{i=1}^{N} I_{\{x_i \neq 0\}} = \|\mathbf{x}\|_0, \quad \lim_{a \to +\infty} T\ell_1(\mathbf{x}) = \sum_{i=1}^{N} |x_i| = \|\mathbf{x}\|_1. \tag{14}$$

Table 1 Commonly used sparse-inducing regularizer

Regularizer	Formula
ℓ_1	$\|\mathbf{x}\|_1 = \sum_{i=1}^{N} \|x_i\|$
SCAD	$\mathbf{P}(\mathbf{x};\lambda,\gamma) = \sum_{i=1}^{n} P(x_i;\lambda,\gamma)\ \ P(x_i;\lambda,\gamma) = \begin{cases} \lambda\|x_i\|, & if\ \|x_i\| \le \lambda \\ \frac{2\gamma\lambda\|x_i\| - x_i^2 - \lambda^2}{2(\gamma-1)}, & if\ \lambda < \|x_i\| < \gamma\lambda \\ \lambda^2(\gamma+1)/2, & if\ \|x_i\| \ge \gamma\lambda, \end{cases}$
MCP	$\mathbf{P}(\mathbf{x};\lambda,\gamma) = \sum_{i=1}^{n} P(x_i;\lambda,\gamma)\ \ P(x_i;\lambda,\gamma) = \begin{cases} \lambda\|x_i\| - x_i^2/(2\gamma), & if\ \|x_i\| \le \gamma\lambda \\ \gamma\lambda^2/2, & if\ \|x_i\| > \gamma\lambda \end{cases}$
Capped ℓ_1	$\mathbf{P}(\mathbf{x};a) = \sum_{i=1}^{n} \min(\|x_i\|, a), a > 0$
Elastic Net	$\mathbf{P}(\mathbf{x};\alpha) = \sum_{i=1}^{n} \left((\alpha-1)x_i^2/2 + (2-\alpha)\|x_i\|\right),\ 1 \le \alpha \le 2,$
ℓ_{1-2}	$\|\mathbf{x}\|_1 - \|\mathbf{x}\|_2$
Tℓ_1	$T\ell_1(\mathbf{x}) = \sum_{i=1}^{n} \rho_a(x_i)\ \ \rho_a(x) = \frac{(a+1)\|x\|}{a+\|x\|}$
ℓ_p	$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} \|x_i\|^p\right)^{1/p}$

The non-convexity and the hyper-parameter property of it make it better sparsify networks [127, 128].

ℓ_p $(0 < p < 1)$ has attracted a great deal of attention in recent years [129–131, 131–135], which is defined by

$$\|x\|_p = \left(\sum_{i=1}^{N} |x_i|^p\right)^{\frac{1}{p}}. \tag{15}$$

In theory, It fulfills unbiasedness, sparsity and oracle properties [129]. And when the value of p approaching to 0, ℓ_p can interpolate ℓ_0.

$$\lim_{p\to 0^+} \|x\|_p^p = \|x\|_0 . \tag{16}$$

The changing rate of the ℓ_p function value is unbounded at zero because it is non-Lipschitz continuous at $x = 0$. From these two perspectives, the mutation between zero and non-zero is a imitation of the discreteness of ℓ_0.

Some commonly used sparse-inducing regularizer are shown in Table 1.

The second type of regularizer-based approach is to use the composite regularizers [99, 101–105], as the following form,

$$\Omega(W) = \mu\Omega_1(W) + (1-\mu)\Omega_2(W), \tag{17}$$

where $\Omega_1(\cdot)$ aims to compress networks on connection-level, $\Omega_2(\cdot)$ is used to enhance the compression capability on neuron-level, and $\mu \in [0, 1]$ is the parameter to keep balance. For example, Zhou et al. [99] utilized two sparse constraints, group sparsity for inducing sparsity on neuron-level and low rank constraint for tensor to promote

Table 2 Commonly used sparse-inducing regularizer

Regularizer	Formula
SSL [14]	$\sum_l \|\|W^{(l)}\|\|_F + \alpha \sum_l \|\|W^{(l)}\|\|_{2,1}$
SGL [136]	$\sum_l \|\|W^{(l)}\|\|_1 + \alpha \sum_l \|\|W^{(l)}\|\|_{2,1}$
TLR& GS [99]	$\sum_l \|\|W^{(l)}\|\|_{tr} + \alpha \sum_l \|\|W^{(l)}\|\|_{2,1}$
CGES [104]	$\sum_l \alpha \|\|W^{(l)}\|\|_{1,1}^2 + (1-\alpha)\|\|W^{(l)}\|\|_{2,1}$
ITℓ_1 [101]	$\sum_l \alpha \|\|W^{(l)}\|\|_{T\ell_1} + (1-\alpha)\|\|W^{(l)}\|\|_{2,1}$
Hoyer-square [121]	$\sum_l \alpha H_s(W^{(l)}) + \beta \|\|W^{(l)}\|\|_2, where H_s(W^l) = \frac{(\sum_i \|w_i\|)^2}{\sum_i w_i^2}$
Group-Hoyer-square [121]	$\sum_l \alpha G_H(W^{(l)}) + \beta \|\|W^{(l)}\|\|_2, where G_H(W^l) = \frac{(\sum_g \|\|w^{(g)}\|\|)^2}{\|\|W^{(l)}\|\|_2^2}$
Polarization [137]	$t\|\|\gamma\|\|_1 - \|\|\gamma - \frac{\sum_{i=1}^n \gamma_i}{n}\mathbf{1}_n\|\|_1$ where γ is the scaling vector for neuron

competition between weights at each layer. The work [102] use the $\ell_{2,1}$ norm, which was used to obtain a sparse network at neuron-level, together with the ℓ_1 norm, which tends to remove connections. Yoon et al. [104] combined group sparsity and exclusive sparsity to promote sharing and competition for different features. In our previous work [101], to explore the strength of these two methodologies, we proposed integrating the non-convex transformed ℓ_1 with group sparsity. Commonly used composite sparse-inducing regularizers for DNNs are shown in Table 2.

Although the combination of the two regularizers can achieve good results in terms of accuracy and compression ratio, this structure makes the optimization process alternate. This alternative optimization method may lead to sawtooth phenomenon and reduce the training speed. In order to overcome the numerical difficulties caused by the composite regularizer, some work design a single regularizer to compress the network without losing accuracy. In order to ensure the compression effect, the regularizer should be able to compress the connection and neuron at the same time.

4 Optimization

$$\min_W \ \mathcal{L}(f(\{W^{(l)}\}_{l=1}^L), D) + \lambda \sum_{l=1}^{L} \mu_l \Omega(W^{(l)}), \tag{18}$$

where $\Omega(W^{(l)})$ is a regularization term, $\lambda > 0$ is the regularization parameter, $\mu_l > 0$ is the balancing parameter used to control the sparse level of each layer l, D is a training set. By coordinating the value of λ, $W^{(l)} \in \mathbb{R}^{m_l \times m_{l-1}}$ represents the weights at the l-th layer, $l \in \{1, 2, \ldots, L\}$, L is the number of layers, m_l denotes the neuron number of layer l.

we can compress the connections of the network during the process of solving problem (18). After obtain the solution to (18), the neurons, whose weights of connections are all zero, would be removed from the network. Then the sparse network is constructed.

4.1 Proximal Algorithm

In this subsection, we introduce the proximal algorithm. It is a very universal method under many circumstances, such as nonsmooth objective function and so on. At the same time, the speed of this algorithm can be very satisfactory in practice. The proximal method is also widely used in many fields, such as kernel norm problem [138], sparse problem [139], maximum a posteriori probability estimation in graph model [140], empirical or structural risk minimization [141–143] and signal processing [144]. It is worth noting that the proximal algorithm is often very suitable for solving sparse optimization problems. In addition, it is simple in both mathematical form and operation, which is very easy to understand and operate. Such an algorithm depends on the use of proximal operator.

4.1.1 Proximal Operator

Firstly, we introduce the concept of proximal operator. A proximal operator $prox_f : \mathbb{R}^n \to \mathbb{R}^n$ on f is defined as follows,

$$prox_f(y) = \arg\min_x \left(f(x) + \frac{1}{2}\|x - y\|_2^2 \right), \tag{19}$$

where $\|\cdot\|_2$ is ℓ_2 norm. The proximal operator of a function is actually solving the optimization problem. Generally, f is accompanied by a scaling coefficient λ, and its proximal operator becomes as follows,

$$prox_{\lambda f}(y) = \arg\min_x \left(f(x) + \frac{1}{2\lambda}\|x - y\|_2^2 \right). \tag{20}$$

From the perspective of gradient, when f is differentiable and $f(x) + \frac{1}{2\lambda}\|x - y\|_2^2$ has an minimum,

$$\nabla f(x) + \frac{x - y}{\lambda} = 0 \Rightarrow x = y - \lambda \nabla f(x) \approx y - \lambda \nabla f(y) \tag{21}$$

that means, $prox_{\lambda f}(y)$ is approximately the gradient descent at the point of y.

There is also an illustrative example for the proximal method when f is $I_{\mathcal{C}}(x)$.

$$I_{\mathcal{C}}(x) = \begin{cases} 0, & x \in \mathcal{C} \\ +\infty, & x \notin \mathcal{C} \end{cases} \tag{22}$$

where $\mathcal{C}$ is a closed nonempty convex set, and its proximal operator is

$$\arg\min_{x}\left(I_{\mathcal{C}}(x)+\frac{1}{2}\|x-y\|_2^2\right). \tag{23}$$

In fact, it is an optimization problem of projection under Euclidean norm,

$$\arg\min_{x\in\mathcal{C}}\|x-y\|_2^2. \tag{24}$$

The solution of Eq. (24) can be seen as a projection of y onto set $\mathcal{C}$. Therefore, the proximal operator can be regarded as a projection. And if f can be divided into $f(x)=\sum_{i=1}^{n} f_i(x_i)$, then the proximal operator has $(prox_f(v))_i = prox_{f_i}(v_i)$. Such property can be applied in parallel computing design.

4.1.2 Proximal Gradient Algorithm

Given a optimization problem as follows,

$$\min_{x} f(x)+g(x). \tag{25}$$

where $f:\mathbb{R}^n\to R$ and $g:\mathbb{R}^n\to R\cup\{+\infty\}$ are two closed proper functions, f is differentiable. The proximal gradient method uses the following iteration to solve the problem,

$$x^{k+1}:=prox_{\lambda^k g}(x^k-\lambda\nabla f(x^k)), \tag{26}$$

4.1.3 Applied in Compression for DNNs

Proximal gradient algorithm is the commom and useful method to train compression model (18). During the process of training DNNs, a stochastic framework is adopted to deal with the computational cost brought by the extremely large training dataset. And stochastic gradient method cannot be applied directly for a majority of regularization. Thus, a stochastic proximal gradient algorithm is uitilized in our previous work. In detail, the proximal operator is presented layer by layer as follow,

$$W_{t+1}^{(l)}=prox_{\lambda\gamma\Omega}\left(W_t^{(l)}-\nabla\mathcal{L}\big(f(W_t^l),D\big)\right), \tag{27}$$

where $W_t^{(l)}$ is the weight matrix of the l-th layer. Furthermore, (27) can be rewritten as the following optimization problem,

$$W_{t+1}^{(l)}=\arg\min_{W}\frac{1}{2\lambda}\left\|W-(W_t^{(l)}-\gamma\nabla\mathcal{L}(f(W_t^l),D))\right\|_F^2+\Omega(W). \tag{28}$$

Using the stochastic gradient to replace the gradient in above Eq. (28), we obtain,

$$W_{t+1}^{(l)} = \arg\min_{W} \left\{ \frac{1}{2\lambda} \left\| W - \left(W_t^{(l)} - \frac{\gamma}{m_0} \sum_{i=1}^{m_0} \nabla \mathcal{L}(f(W_t^l), \{x_i, y_i\}) \right) \right\|_F^2 + \Omega(W) \right\}, \tag{29}$$

where m_0 is the mini-batch size in SGD.

It is a general framework to solve regularized compression model. However, ther is a few of differences for the pipeline of solving a combined regularized objective function. As we known, the success of the application of proximal methods depends on whether there is a solution of the proximal operator for the regularization term. It is difficult to directly calculate the proximal operator of such a combined regularization term. Under this circumstace, calculating the proximal operators of two regular functions separately is a pragmatic method [101, 104, 136]. For a regularized compression model as follows,

$$\min_{W} \ \mathcal{L}(f(\{W^{(l)}\}_{l=1}^{L}), D) + \lambda \sum_{l=1}^{L} (\mu_l \Omega_1(W^{(l)}) + (1 - \mu_l)\Omega_2(W^{(l)})), \tag{30}$$

calculation results for per regularization term are utilized ralatively to update the gradient steps only on the loss function iteratively.

$$W_{t+1}^{(l)} = prox_{\lambda^{(l)}\gamma(1-\mu_l)\Omega_2} \left(prox_{\lambda^{(l)}\gamma\mu_l\Omega_1}(W_t^{(l)} - \gamma \sum_{i=1}^{n} \nabla \mathcal{L}(W_t^{(l)}, \{x_i, y_i\})/n) \right). \tag{31}$$

As seen from the Eq. (27) and the Eq. (31), we can find some differences in single and combined regularization from the perspective of optimization process. Noticing that the optimization pipeline of the these combined regularizers, we optimized alternatively for the two-term structure. It in practice results in slows down the convergence. Meanwhile, each part of these two-term regularizers can be seen a single regularization. Thus, the alternative optimization of per term regularization may cause the computational difficulties.

4.2 Subgradient Based Method

4.2.1 Subgradient

$\partial f(x)$ is called the subgradient of f at the point x, if it satisfies the condition,

$$f(y) \geq f(x) + \partial f(x)(y - x) \tag{32}$$

Table 3 Subgradients and proximal operator of Several sparse regularizers

Regularizer $R(x)$	$\partial R(x)$	Proximal operator $prox_{\lambda\Omega}$
ℓ_1	$\begin{cases} sgn(x_i), & if x_i \neq 0 \\ [-1,1], & if\ x_i = 0, \end{cases}$	$sgn(x_i)(\lvert x_i\rvert - \lambda)_+$
ℓ_p	$\begin{cases} \frac{p \cdot sgn(x_i)}{\lvert x_i\rvert^{1-p}}, & if x_i \neq 0 \\ \mathbb{R}, & if\ x_i = 0, \end{cases}$	
$\mathrm{T}\ell_1$	$\begin{cases} \frac{a(a+1)sgn(x_i)}{(a+\lvert x_i\rvert)^2}, & if x_i \neq 0 \\ 0, & if\ x_i = 0, \end{cases}$	$\begin{cases} 0, & if\ \lvert\hat{\omega}_{g,i}\rvert \leq t \\ g_\lambda(\hat{\omega}_{g,i}), & otherwise \end{cases}$ where $g_\lambda(\omega) = sgn(\omega)\left\{\frac{2}{3}(a+\lvert\omega\rvert)\cos(\frac{\phi_\lambda(\omega)}{3}) - \frac{2a}{3} + \frac{\lvert\omega\rvert}{3}\right\},$ $\phi_\lambda(\omega) = arc\cos\left(1 - \frac{27\lambda a(a+1)}{2(a+\lvert\omega\rvert)^3}\right),$ $t = \begin{cases} \lambda\frac{a+1}{a}, & if\ \lambda \leq \frac{a^2}{a(a+1)} \\ \sqrt{2\lambda(a+1)} - \frac{a}{2}, & otherwise \end{cases}$
$\ell_{2,1}$	$\begin{cases} \frac{\mathbf{x}}{\lvert\lvert\mathbf{x}\rvert\rvert_2}, & if \mathbf{x} \neq \mathbf{0} \\ \{\mathbf{x} \in \mathbb{R}^n : \lvert\lvert\mathbf{x}\rvert\rvert_2 \leq 1\}, & if \mathbf{x} = \mathbf{0}, \end{cases}$	$x_i(1 - \lambda/\lvert\lvert\mathbf{x}\rvert\rvert_2)_+$

4.2.2 Subgradient Descent

Due to the non-smoothness of the commonly used sparse-inducing regularization, gradient descent cannot be applied for our model (18) directly. To minimize (18) in general, subgradient descent is also utilized [145], where stochastic gradient descent is applied to the first term while subgradient descent is applied to the regularization term. The mathematical form is as follows

$$w_{t+1}^{(l)} = w_t^{(l)} - \nabla\mathcal{L}(f(w_t^l), D) \tag{33}$$

$$w_{t+2}^{(l)} = w_{t+1}^{(l)} - \alpha_{t+1} g_{t+1}^{(l)} \tag{34}$$

where $w_t^{(l)}$ is the weight of l-th layer in t-th iteration, g^t is any subgradient of Ω at w^t, and $\alpha_t > 0$ is the step size in t-th iteration.

4.3 Discussion

We compare subgradients and proximal operators of several sparse regularizers in Table 3. From these two optimization algorithms and Table 3, we find that the compression by proximal gradient algorithm usually obtain a threshold-form proximal operator from which the unsignificant weights are exactly zero through the optimization process. The sparsity in the network results from this. In the network trained by subgradient descent method, some weights may become very small in the optimization process, but it is difficult to accurately become zero. The compression of the network needs post-processing, and the sparsity is obtained by pruning. It may need retraining to keep the accuracy.

We used to focus on the sparsity-iducing capability of various sparse regularizer, which is supposed to make weights in the network zero directly. We often focus on setting the parameter to zero and the threshold value setting parameters to zero. However, pruning a network without a threshold produced by regularizer is actually possible, and it can also control the compression ratio. Compression with subgradient algorithm inspires us. In regularization based compression using subgradient descent method, post-operation, pruning, is needed. They don't spend too much effort on the threshold, but they still get a compressed network with good accuracy. The sparsity brought by the regularizer, which can be seen as the anti-disturbance ability brought by it to the model. In other words, the ability of regularizers is the ability to select the optimal subnetwork for the original network. It will pick some of the more important positions of the weight and keep them down.

5 Avenues for Future Research

Considering the enormous interest in neural network compression, we briefly review and analyze the sparse regularization based compression methods for DNNs in this work. After carefully studying the literature, we overview sparse regularizations and optimization methods for DNNs compression. We discussed both the different advantages and disadvantages, and provided some insights and discussions on how to make sparse regularization fit within the compression framework to help future research endeavors to produce the kinds of results. In this section, we indicate some avenues for future research so that regularization will harmonize the networks compression and develop better in this area.

First, regularizer designing has attracted great attention of researchers. They design regularizers with different desired properties, such as unbiasedness, sparsity, scale-invariant, continuity and Lipschitz-continuity. A good regularizer with desired properties should result in an estimator with preference. Moreover, to obtain structural compression effect, state-of-the-art works focus on removing neurons utilizing $\ell_{2,1}$ group sparse regularizer. Although $\ell_{2,1}$ performs well, there is only $\ell_{2,1}$ widely used. Therefore, it is significant to design or utilize a novel group sparse regularizer which can better capture intra-group information or better promote group sparsity, so that the regularized networks can adapt to different tasks.

In addition, to obtain good compression effect, composite regularizers are utilized to sparsify connections and neurons at the same time. One part of a composite regularizer removes spare connections and the other part removes unnecessary neurons. Although the composite regularizers can obtain accurate compressed networks, they are usually difficult to solve. Because the algorithms need alternately optimize the two regularized objective. Such an alternative optimization usually leads to the zigzagging phenomenon and slow down the training. Therefore, it is necessary to design a simpler regularizer or algorithm in order to overcome the computational difficulty.

Furthermore, it will be very promising to study from the perspective of Bayesian. Noticing that explicit regularization can be considered as introducing priors into posteriors. For example, the commonly used prior distribution is Laplacian prior distribution, which corresponds to ℓ_1 regularization. In the future, it is promising to extend

the commonly used priors to more general priors. From a compression point of view, sparse priors can be placed over connection-wise or neuron-wise structures. some sparser prior distribution can be utilized on weights and some priors such as Bernoulli priors can be introduced over structures such as convolutional filters or ResNet units to promote sturctural sparsity.

Second, it is meaningful to introduce regularization based compression methods on other neural network architectures. With the increasing scale of tasks to be solved, the network used is becoming wider and deeper. Although there are various network architectures, a majority of methods verify the effect on convolutional neural networks. In the future, it is meaningful to test the performance on other neural network architectures. Furthermore, it could be interesting to study the pipeline of compression, such as training models from scratch or in a student-teacher setting.

Last but not least, it is promising to explore the effect of explicit and implicit regularizations. Generally speaking, existing works on DNN model compression include pruning, dropout, quantization and optimization with explicit regularization. In addition to the explicit regularization based method, other methods may impose a regularization effect by the training algorithm rather than introducing explicit regularizer to the objective function. These algorithmic methods can be regarded as a implicit regularization. For instance, pruning is to remove non-informative weights which are insensitive to the performance in a pretrained network. It may remove bad noise to improve the generalization or robustness of the networks. Dropout is proposed to reduce over-fitting and improve the performance by randomly removing neurons, which is definitely a regularization. It could be interesting to explore the effect of explicit and implicit regularizations.

Acknowledgements This work was supported by the National Natural Science Foundation of China [No.71932008], UCAS, China Grant [No. Y55202LY00].

Author Contributions Anda Tang contributed to the conception of the study,the survey for compression methods, analysis and wrote the manuscript; Pei Quan contributed to analysis and manuscript preparation; Lingfeng Niu, Yong Shi helped perform the analysis with constructive discussions.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. The authors declare the following financial interests/personal relationships which may be considered as potential competing interests.

Ethical Statement for Annals of Data Science I testify on behalf of all co-authors that our article submitted to Annals of Data Science: Title: A survey for sparse regularization based compression methods All authors: Anda Tang , Pei Quan , Lingfeng Niu , Yong Shi 1. This material has not been published elsewhere. 2. The manuscript is not currently being considered for publication in another journal. 3. All authors have been personally and actively involved in substantive work leading to the manuscript, and will hold themselves jointly and individually responsible for its content

References

1. Tien JM (2017) Internet of things, real-time decision making, and artificial intelligence. Ann Data Sci 4(2):149–178
2. Olson DL, Shi Y, Shi Y (2007) Introduction to business data mining, vol 10. McGraw-Hill/Irwin
3. Liu F, Shi Y (2020) Investigating laws of intelligence based on AI IQ research. Ann Data Sci 7(3):399–416
4. Shi Y (2022) Advances in Big Data Analytics: Theory. Algorithms and Practices. Springer
5. Wang P, Ouyang H, Zhong Y, He H (2016) Cognition math based on factor space. Ann Data Sci 3(3):281–303
6. Geoffrey H, Li D, Dong Y, George ED, Mohamed A (2012) Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Process Mag 29(6):82–97
7. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp 1097–1105
8. Dauphin YN, Fan A, Auli M, Grangier D (2017) Language modeling with gated convolutional networks. In: Proceedings of the 34th international conference on machine learning-Volume 70, JMLR. org, pp 933–941
9. Learning D. Deep learning, High-Dimensional Fuzzy Clustering
10. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. pp 770–778
11. Devlin J, Chang M-W, Lee K, Toutanova K, Bert (2019) Pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT
12. Han S (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems, pp 1135–1143
13. . Christos Louizos MW, Ullrich K (2017) Bayesian compression for deep learning. In: Advances in neural information processing systems. pp 3288–3298
14. Wen W, Wu C, Wang Y, Chen Y, Li H (2016) Learning structured sparsity in deep neural networks. In: Advances in neural information processing systems. pp 2074–2082
15. Molchanov AA, Dmitry DV (2017) In: ICML (Ed) Variational dropout sparsifies deep neural networks
16. Diederik MW, Kingma P, Salimans T (2015) Variational dropout and the local reparameterization trick. In: Advances in neural information processing systems
17. Christos Louizos K, Welling Max Learning sparse neural networks through l0 regularization. ArXiv: abs/1712.01312
18. Shi Y, Tian Y, Kou G, Peng Y, Li J (2011) Optimization based data mining: theory and applications. Springer
19. He K, Sun J (2015) Convolutional neural networks at constrained time cost. pp 5353–5360
20. Denton EL, Zaremba W, Bruna J, LeCun Y, Fergus R (2014) Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in neural information processing systems, pp 1269–1277
21. Xue J, Li J, Yu D, Seltzer M, Gong Y (2014) Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network. pp 6359–6363
22. Zhang X, Zou J, He K, Sun J (2016) Accelerating very deep convolutional networks for classification and detection. IEEE Trans Pattern Anal Mach Intell 38(10):1943–1955
23. Rigamonti R, Sironi A, Lepetit V, Fua P (2013) Learning separable filters. pp 2754–2761
24. Kim YD, Park E, Yoo S, Choi T, Yang L, Shin D (2015) Compression of deep convolutional neural networks for fast and low power mobile applications. arXiv preprint arXiv:1511.06530
25. Wang P, Cheng J (2016) Accelerating convolutional neural networks for mobile applications. pp 541–545
26. Ye J, Wang L, Li G, Chen D, Zhe S, Chu X, Xu Z (2018) Learning compact recurrent neural networks with block-term tensor decomposition. pp 9378–9387
27. Novikov A, Podoprikhin D, Osokin A, Vetrov DP (2015) Tensorizing neural networks. In: Advances in neural information processing systems. pp 442–450
28. Wang W, Sun Y, Eriksson B, Wang W, Aggarwal V (2018) Wide compression: Tensor ring nets. pp 9329–9338
29. Sainath TN, Kingsbury B, Sindhwani V, Arisoy E, Ramabhadran B (2013) Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: IEEE international conference on acoustics, speech and signal processing. IEEE 2013:6655–6659

30. Jaderberg M, Vedaldi A, Zisserman A Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866
31. Ioannou Y, Robertson D, Shotton J, Cipolla R, Criminisi A Training cnns with low-rank filters for efficient image classification. arXiv preprint arXiv:1511.06744
32. Wen W, Xu C, Wu C, Wang Y, Chen Y, Li H (2017) Coordinating filters for faster deep neural networks. pp 658–666
33. Yu X, Liu T, Wang X, Tao D (2017) On compressing deep models by low rank and sparse decomposition. pp 7370–7379
34. Cohen T, Welling M (2016) Group equivariant convolutional networks. In: International conference on machine learning, pp 2990–2999
35. Zhai S, Cheng Y, Zhang ZM, Lu W (2016) Doubly convolutional neural networks. In: Advances in neural information processing systems, pp 1082–1090
36. Li H, Ouyang W, Wang X (2016) Multi-bias non-linear activation in deep neural networks. pp 221–229
37. Shang W, Sohn K, Almeida D, Lee H (2016) Understanding and improving convolutional neural networks via concatenated rectified linear units. pp 2217–2225
38. Dieleman S, De Fauw J, Kavukcuoglu K (2016) Exploiting cyclic symmetry in convolutional neural networks. arXiv preprint arXiv:1602.02660
39. Cheng Y, Wang D, Zhou P, Zhang T (2017) A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282
40. Szegedy C, Ioffe S, Vanhoucke V, Alemi AA (2017) Inception-v4, inception-resnet and the impact of residual connections on learning. In: 31st AAAI conference on artificial intelligence
41. Wu B, Iandola F, Jin PH, Keutzer K, Squeezedet, (2017) Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. pp 129–137
42. Buciluǎ C, Caruana R, Niculescu-Mizil A (2006) Model compression. pp 535–541
43. Ba J, Caruana R(2014) Do deep nets really need to be deep? In: Advances in neural information processing systems, pp 2654–2662
44. Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531
45. Romero A, Ballas N, Kahou SE, Chassang A, Gatta C, Bengio Y (2014) Fitnets: Hints for thin deep nets. arXiv preprint arXiv:1412.6550
46. Luo P, Zhu Z, Liu Z, Wang X, Tang X (2016) Face model compression by distilling knowledge from neurons. In: 13th AAAI Conference on Artificial Intelligence
47. Chen T, Goodfellow I, Shlens J (2015) Net2net: Accelerating learning via knowledge transfer. arXiv preprint arXiv:1511.05641
48. Zagoruyko S, Komodakis N (2016) Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer. arXiv preprint arXiv:1612.03928
49. Yim J, Joo D, Bae J, Kim J (2017) A gift from knowledge distillation: fast optimization. Network minimization and transfer learning. pp 4133–4141
50. Chen G, Choi W, Yu X, Han T, Chandraker M (2017) Learning efficient object detection models with knowledge distillation. In: Advances in neural information processing systems. pp 742–751
51. Huang M, Wang N (2017) Like what you like: Knowledge distill via neuron selectivity transfer. arXiv preprint arXiv:1707.01219
52. Gong Y, Liu L, Yang M, Bourdev L (2014) Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115
53. Choi Y, El-Khamy M, Lee J (2016) Towards the limit of network quantization. arXiv preprint arXiv:1612.01543
54. Wu J, Leng C, Wang Y, Hu Q, Cheng J (2016) Quantized convolutional neural networks for mobile devices. pp 4820–4828
55. Park E, Ahn J, Yoo S (2017) Weighted-entropy-based quantization for deep neural networks. pp 5456–5464
56. Vanhoucke V, Senior A, Mao MZ (2011) Improving the speed of neural networks on CPUs
57. Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P (2015) Deep learning with limited numerical precision. In: International Conference on Machine Learning. pp 1737–1746
58. Micikevicius P, Narang S, Alben J, Diamos G, Elsen E, Garcia D, Ginsburg B, Houston M, Kuchaiev O, Venkatesh G, et al. (2017) Mixed precision training. arXiv preprint arXiv:1710.03740
59. Wu S, Li G, Chen F, Shi L (2018) Training and inference with integers in deep neural networks. arXiv preprint arXiv:1802.04680

60. Seide F, Fu H, Droppo J, Li G, Yu D (2014) 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs
61. Courbariaux M, Bengio Y, David J-P (2015) Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems, 2015, pp 3123–3131
62. Courbariaux M, Bengio Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1
63. Strom N (2015) Scalable distributed dnn training using commodity gpu cloud computing, in
64. Rastegari M, Ordonez V, Redmon J, Farhadi A (2016) Xnor-net: Imagenet classification using binary convolutional neural networks. In: European Conference on Computer Vision. Springer, pp 525–542
65. Hou L, Yao Q, Kwok JT (2016) Loss-aware binarization of deep networks. arXiv:1611.01600
66. Kim M, Smaragdis P (2016) Bitwise neural networks. arXiv preprint arXiv:1601.06071
67. Hu Q, Wang P, Cheng J (2018) From hashing to cnns: Training binary weight networks via hashing. In: 32nd AAAI Conference on Artificial Intelligence
68. Lin Z, Courbariaux M, Memisevic R, Bengio Y (2015) Neural networks with few multiplications. arXiv preprint arXiv:1510.03009
69. Li F, Zhang B, Liu B (2016) Ternary weight networks. arXiv preprint arXiv:1605.04711
70. Zhu C, Han S, Mao H, Dally WJ (2016) Trained ternary quantization. arXiv preprint arXiv:1612.01064
71. Mellempudi N, Kundu A, Mudigere D, Das D, Kaul B, Dubey P (2017) Ternary neural networks with fine-grained quantization. arXiv preprint arXiv:1705.01462
72. Deng L, Jiao P, Pei J, Wu Z, Li G (2018) Gxnor-net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework. Neural Netw 100:49–58
73. Wang P, Cheng J (2017) Fixed-point factorized networks. pp 4012–4020
74. Wen W, Xu C, Yan F, Wu C, Wang Y, Chen Y, Li H (2017) Terngrad: ternary gradients to reduce communication in distributed deep learning. In: Advances in neural information processing systems, pp 1509–1519
75. Muller LK, Indiveri G (2015) Rounding methods for neural networks with low resolution synaptic weights. arXiv preprint arXiv:1504.05767
76. Polino A, Pascanu R, Alistarh D (2018) Model compression via distillation and quantization. arXiv preprint arXiv:1802.05668
77. Koster U, Webb T, Wang X, Nassar M, Bansal AK, Constable W, Elibol O, Gray S, Hall S, Hornof L, et al (2017) Flexpoint: an adaptive numerical format for efficient training of deep neural networks. In: Advances in neural information processing systems, pp 1742–1752
78. Cai Z, He X, Sun J, Vasconcelos N (2017) Deep learning with low precision by half-wave gaussian quantization. pp 5918–5926
79. Mishra A, Nurvitadhi E, Cook JJ, Marr D (2017) Wrpn: wide reduced-precision networks. arXiv preprint arXiv:1709.01134
80. 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
81. Hanson SJ, Pratt LY (1989) Comparing biases for minimal network construction with back-propagation. In: Advances in neural information processing systems, pp 177–185
82. LeCun Y, Denker JS, Solla SA (1990) Optimal brain damage. In: Advances in neural information processing systems pp 598–605
83. Hassibi B, Stork DG (1993) Second order derivatives for network pruning: Optimal brain surgeon. In: Advances in neural information processing systems, pp 164–171
84. Hassibi B, Stork DG, Wolff GJ (1993) Optimal brain surgeon and general network pruning. In: IEEE international conference on neural networks. IEEE, pp 293–299
85. Srinivas S, Babu RV, Data-free parameter pruning for deep neural networks. arXiv preprint arXiv:1507.06149
86. 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
87. Han S, Mao H, Dally WJ (2015) Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149
88. Molchanov P, Tyree S, Karras T, Aila T, Kautz J (2016) Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440

89. Anwar S, Hwang K, Sung W (2017) Structured pruning of deep convolutional neural networks. ACM J Emerg Technol Comput Syst (JETC) 13(3):32
90. Narang S, Elsen E, Diamos G, Sengupta S (2017) Exploring sparsity in recurrent neural networks. arXiv preprint arXiv:1704.05119
91. Lin S, Ji R, Li Y, Wu Y, Huang F, Zhang B (2018) Accelerating convolutional networks via global & dynamic filter pruning. In: IJCAI, pp 2425–2432
92. Molchanov P, Mallya A, Tyree S, Frosio I, Kautz J (2019) Importance estimation for neural network pruning. pp 11264–11272
93. Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR (2012) Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580
94. Wan L, Zeiler M, Zhang S, Le Cun Y, Fergus R (2013) Regularization of neural networks using dropconnect. pp 1058–1066
95. Ba J, Frey B (2013) Adaptive dropout for training deep neural networks. In: Advances in Neural Information Processing Systems, pp 3084–3092
96. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
97. Poernomo A, Kang D-K (2018) Biased dropout and crossmap dropout: learning towards effective dropout regularization in convolutional neural network. Neural Netw 104:60–67
98. Collins MD, Kohli P (2014) Memory bounded deep convolutional networks. arXiv:1412.1442
99. Zhou H, Alvarez JM, Porikli F (2016) Less is more: towards compact cnns. In: European Conference on Computer Vision. Springer, pp 662–677
100. Liu J, Musialski P, Wonka P, Ye J (2013) Tensor completion for estimating missing values in visual data. IEEE Trans Pattern Anal Mach Intell 35(1):208–220
101. Ma R, Miao J, Niu L, Zhang P (2019) Transformed ℓ_1 regularization for learning sparse deep neural networks. Neural Netw 119:286–298
102. Alvarez JM, Salzmann M (2016) Learning the number of neurons in deep networks. In: Advances in neural information processing systems, pp 2270–2278
103. Lebedev V, Lempitsky V (2016) Fast convnets using group-wise brain damage, pp 2554–2564
104. Yoon J, Hwang SJ (2017) Combined group and exclusive sparsity for deep neural networks. In: Proceedings of the 34th international conference on machine learning-volume 70, JMLR. org, pp 3958–3966
105. Li X, Grandvalet Y, Davoine F (2019) A baseline regularization scheme for transfer learning with convolutional neural networks. Pattern Recogn 98:107049
106. Aslan Ö, Zhang X, Schuurmans D (2014) Convex deep learning via normalized kernels. In: Advances in neural information processing systems, pp 3275–3283
107. Bengio Y, Roux NL, Vincent P, Delalleau O, Marcotte P (2006) Convex neural networks. In: Advances in neural information processing systems, pp 123–130
108. Aghasi A, Abdi A, Nguyen N, Romberg J (2017) Net-trim: convex pruning of deep neural networks with performance guarantee. In: Advances in neural information processing systems, pp. 3177–3186
109. Gu J, Wang Z, Kuen J, Ma L, Shahroudy A, Shuai B, Liu T, Wang X, Wang G, Cai J et al (2018) Recent advances in convolutional neural networks. Pattern Recogn 77:354–377
110. Yu J, Rui Y, Tao D (2014) Click prediction for web image reranking using multimodal sparse coding. IEEE Trans Image Process 23(5):2019–2032
111. Glorot X, Bordes A, Bengio Y (2011) Deep sparse rectifier neural networks, pp 315–323
112. Fan J, Li R (2001) Variable selection via nonconcave penalized likelihood and its oracle properties. J Am Stat Assoc 96(456):1348–1360
113. Zhang C-H et al (2010) Nearly unbiased variable selection under minimax concave penalty. Ann Stat 38(2):894–942
114. Zhang T (2009) Multi-stage convex relaxation for learning with sparse regularization. In: Advances in Neural Information Processing Systems, pp 1929–1936
115. Zou H, Hastie T (2005) Regularization and variable selection via the elastic net. J R Stat Soc: Ser B (Stat Methodol) 67(2):301–320
116. Mazumder R, Friedman JH, Hastie T (2011) Sparsenet: Coordinate descent with nonconvex penalties. J Am Stat Assoc 106(495):1125–1138
117. Hoyer PO, Non-negative matrix factorization with sparseness constraints. J Mach Learn Res 5(9):1
118. Repetti A, Pham MQ, Duval L, Chouzenoux E, Pesquet J-C (2014) Euclid in a taxicab: Sparse blind deconvolution with smoothed ℓ_1/ℓ_2 regularization. IEEE Signal Process Lett 22(5):539–543

119. Krishnan D, Tay T, Fergus R (2011) Blind deconvolution using a normalized sparsity measure. In: CVPR 2011. IEEE, pp 233–240
120. Hurley N, Rickard S (2009) Comparing measures of sparsity. IEEE Trans Inf Theory 55(10):4723–4741
121. Yang H, Wen W, Li H (2019) Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. arXiv preprint arXiv:1908.09979
122. Esser E, Lou Y, Xin J (2013) A method for finding structured sparse solutions to nonnegative least squares problems with applications. SIAM J Imag Sci 6(4):2010–2046
123. Yin P, Lou Y, He Q, Xin J (2015) Minimization of ℓ_{1-2} for compressed sensing. SIAM J Sci Comput 37(1):A536–A563
124. 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. J Sci Comput 64(1):178–196
125. Nikolova M (2000) Local strong homogeneity of a regularized estimator. SIAM J Appl Math 61(2):633–658
126. Zhang S, Xin J (2018) Minimization of transformed ℓ_1 penalty: closed form representation and iterative thresholding algorithms. arXiv preprint arXiv:1412.5240
127. Zhang S, Xin J (2017) Minimization of transformed l_1 penalty: Closed form representation and iterative thresholding algorithms. Commun Math Sci 15(2):511–537
128. Xue F, Xin J (2019) Learning sparse neural networks via ell_0 and transformed ell_1 by a relaxed variable splitting method with application to multi-scale curve classification. In: World congress on global optimization. Springer, pp 800–809
129. Xu Z, Zhang H, Wang Y, Chang X, Liang Y (2010) $l_{1/2}$ regularization, Science China. Inf Sci 53(6):1159–1169
130. Chartrand R, Staneva V (2008) Restricted isometry properties and nonconvex compressive sensing. Inverse Problems 24(3):035020
131. Krishnan D, Fergus R (2009) Fast image deconvolution using hyper-laplacian priors. In: Advances in neural information processing systems 1033–1041
132. Xu Z, Chang X, Xu F, Zhang H (2012) $l_{1/2}$ regularization: a thresholding representation theory and a fast solver. IEEE Trans Neural Netw Learn Syst 23(7):1013–1027
133. Xu ZX, Guo H-L, Wang Y, Zhang L (2012) Representative of $l_{1/2}$ regularization among l_q $(0<q \leq 1)$ regularizations: an experimental study based on phase diagram. Acta Automatica Sinica 38(7):1225–1228
134. Xu Z (2010) Data modeling: Visual psychology approach and $l_{1/2}$ regularization theory, in: Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures. World Scientific, pp 3151–3184
135. Chartrand R, Yin W (2008) Iteratively reweighted algorithms for compressive sensing. pp 3869–3872
136. Scardapane S, Comminiello D, Hussain A, Uncini A (2017) Group sparse regularization for deep neural networks. Neurocomputing 241:81–89
137. Zhuang T, Zhang Z, Huang Y, Zeng X, Shuang K, Li X (2020) Neuron-level structured pruning using polarization regularizer. Adv Neural Inf Process Syst 33:1
138. Toh K-C, Yun S (2010) An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems. Pac J Optim 6(615–640):15
139. Scheinberg K, Ma S, Goldfarb D (2010)Sparse inverse covariance selection via alternating linearization methods. In: Advances in neural information processing systems, pp 2101–2109
140. Ravikumar P, Agarwal A, Wainwright MJ (2010) Message-passing for graph-structured linear programs: Proximal methods and rounding schemes. J Mach Learn Res 11:1043–1080
141. Boyd S, Parikh N, Chu E, Peleato B, Eckstein J et al (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends®. Mach Learn 3(1):1–122
142. Do CB, Le QV, Foo C-S (2009) Proximal regularization for online and batch learning. pp 257–264
143. Jenatton R, Mairal J, Obozinski G, Bach FR (2010) Proximal methods for sparse hierarchical dictionary learning
144. Combettes PL, Pesquet J-C (2011) Proximal splitting methods in signal processing. Fixed-point algorithms for inverse problems in science and engineering. Springer, pp 185–212
145. Bui K, Park F, Zhang S, Qi Y, Xin J (2020) Nonconvex regularization for network slimming:compressing cnns even more. Adv Vis Comput