Automatic Sparse Connectivity Learning for Neural Networks

Zhimin Tang, Linkai Luo, Bike Xie, Yiyu Zhu, Rujie Zhao, Lvqing Bi, and Chao Lu

Abstract—Since sparse neural networks usually contain many zero weights, these unnecessary network connections can potentially be eliminated without degrading network performance. Therefore, well-designed sparse neural networks have the potential to significantly reduce the number of floating-point operations (FLOPs) and computational resources. In this work, we propose a new automatic pruning method—sparse connectivity learning (SCL). Specifically, a weight is reparameterized as an elementwise multiplication of a trainable weight variable and a binary mask. Thus, network connectivity is fully described by the binary mask, which is modulated by a unit step function. We theoretically prove the fundamental principle of using a straight-through estimator (STE) for network pruning. This principle is that the proxy gradients of STE should be positive, ensuring that mask variables converge at their minima. After finding Leaky ReLU, Softplus, and identity STEs can satisfy this principle, we propose to adopt identity STE in SCL for discrete mask relaxation. We find that mask gradients of different features are very unbalanced; hence, we propose to normalize mask gradients of each feature to optimize mask variable training. In order to automatically train sparse masks, we include the total number of each feature to optimize mask variable training. In order to very unbalanced; hence, we propose to normalize mask gradients of each feature to optimize mask variable training. In order to automatically train sparse masks, we include the total number of each feature to optimize mask variable training. In order to automatically train sparse masks, we include the total number of each feature to optimize mask variable training. In order to automatically train sparse masks, we include the total number of each feature to optimize mask variable training.

Index Terms—Model compression, model pruning, neural networks, sparse connectivity learning (SCL), trainable binary mask.

I. INTRODUCTION

DESPITE the great success in improving neural networks and learning systems [1]–[6], state-of-the-art deep neural networks usually consist of dozens of stacked layers and a huge number of parameters. It is difficult to deploy these overparameterized and over-redundant neural networks on resource-constrained computing platforms [7]. To address this challenge, network pruning has received great attention. Network pruning tends to remove unimportant trainable parameters in a neural network architecture while maintaining its high accuracy. Effective network pruning leads to less computing operations, memory usage, and power consumption with little performance degeneration. After performing network pruning, sparse models are often implemented in hardware (e.g., GPUs, ASICs\(^1\) or FPGAs\(^2\)) for AI acceleration.

Network pruning can be divided into two categories: human-designed pruning and automatic pruning. The former requires some pruning criteria or hyperparameters defined by designers (e.g., importance measure or pruning threshold), while automatic pruning generates optimized sparse networks with little human intervention. Human-designed pruning usually consists of three steps: 1) training weight parameters in a selected baseline neural network; 2) eliminating unimportant network connections based on designer-defined criteria or hyperparameters; and 3) training weight parameters again in this pruned network architecture. Human-designed network pruning can be further divided into two types: unstructured [7]–[13] or structured [14]–[26]. It has been reported that performing unstructured pruning on deep neural networks does not cause much loss of accuracy. On the other hand, structured pruning, especially filter [14]–[19] and channel [20]–[24] pruning, has been used to accelerate neural networks in general hardware platforms (i.e., GPUs). Since zero-masked feature maps can be deleted, the less computational cost is required after network pruning. Note that some existing neural networks with multibranch or multigroup structures may be considered as human-designed sparse architectures (e.g., inception [1], ShuffleNet [27], MobileNet [28], and ResNeXt [29]), even though these sparse architectures do not involve network pruning.

Up to date, three major bottlenecks have hindered the use of human-designed pruning methods to generate sparse networks. First, although human-designed pruning can provide good network performance under a low pruning rate, the network performance under a high pruning rate is severely degraded. Second, there is a lack of appropriate methods to effectively prune network connections for high-compression and high-performance neural networks. In most human-designed pruning methods, network connections are pruned based on the assumption of “smaller-norm-less-important.”

\(^1\)https://www.xilinx.com/applications/megatrends/machine-learning.html
\(^2\)https://www.kneron.com/solutions/soc/

Manuscript received 16 October 2019; revised 4 August 2020, 1 March 2021, and 4 October 2021; accepted 2 January 2022. Date of publication 25 January 2022; date of current version 6 October 2023. This work was supported in part by the China Natural Science Foundation under Grant 62171391 and in part by the Opening Foundation of Yulin Research Institute of Electronic Engineering and Telecommunication, Hebei Key Laboratory of Cloud Computing Technology and Application, and Hebei Key Laboratory of IoT Network Security and Technology. (Corresponding authors: Linkai Luo; Chao Lu.)

Zhimin Tang was with the Department of Electrical and Computer Engineering, Southern Illinois University Carbondale, Carbondale, IL 62901 USA. He is now with the Department of Automation, Xiamen University, Xiamen 361102, China.

Linkai Luo is with the Department of Electrical and Computer Engineering, Southern Illinois University Carbondale, Carbondale, IL 62901 USA. He is also with the Department of Automation, Xiamen University, Xiamen 361102, China (e-mail: luokl@xmu.edu.cn).

Bike Xie and Yiyu Zhu are with Kneron Inc., San Diego, CA 92121 USA. Rujie Zhao and Chao Lu are with the Department of Electrical and Computer Engineering, Southern Illinois University Carbondale, Carbondale, IL 62901 USA (e-mail: chaolu@siu.edu).

Lvqing Bi is with the Research Center for Intelligent Information and Communication Technology, School of Physics and Telecommunication Engineering, Yulin Normal University, Yulin, Guangxi 537000, China.

Digital Object Identifier 10.1109/TNNLS.2022.3141665

See https://www.ieee.org/publications/rights/index.html for more information.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3141665.
Network connections with smaller weights are generally considered trivial and eliminated during network pruning. However, researchers have found that sometimes the amount of information in smaller weights is important and cannot be ignored [30]. Therefore, considering the variability of loss function sensitivity with respect to different weights, eliminating network connections with smaller weights does not guarantee a slight decrease in network performance. Third, the biggest weakness is the need for designer-defined pruning criteria. For example, let $L_1$ or $L_2$ norm of filters and hyperparameters (e.g., pruning threshold or ratio) for each network layer during network pruning. Because the selection of pruning criteria or hyperparameters is manually determined, it heavily depends on the designer's prior experience and varies from application to application. Consequently, since the pruning criteria and hyperparameters are not guaranteed to be the best choice, the resulting network connectivity and performance are not optimal. Note that even though weight importance estimation methods have been proposed in [31] and [32], layerwise hyperparameters are still needed to determine a pruning threshold or ratio for all layers during network pruning. Unfortunately, as the optimal pruning threshold or ratio varies with local network structures, the use of only one hyperparameter for all layers leads to inferior pruning performance. Recently, it is proposed that a proper criterion may be selected from a set of designer-defined criteria to learn pruning criteria for each network layer [33]. Yet, the pruning performance of this method still depends on the quality of candidate criteria defined by designers.

To get rid of shortcomings of human-designed pruning, researchers have investigated automatic pruning, which trains sparse network connectivity through task-aware loss or a sparse regularization term [34]–[38]. Note that automatic pruning does not require designer-defined pruning criteria or layerwise hyperparameters. Louizos et al. [34] train sparse neural networks through $L_0$-norm regularization. They use the Gumbel-Softmax trick (also known as a concrete distribution) [39], [40] and apply gates to weights for connectivity training. Note a stochastic gate produces zero connectivity only when the probability is zero, which is almost impossible to reach during network training. To address this problem, threshold operation is applied on gates to make zero connectivity. The drawback of [34] is that the expected $L_0$-norm of stochastic training does not reflect the $L_0$-norm of deterministic inference. Therefore, as will be discussed in Section V-D3, although the expected $L_0$ norm of weights is significantly reduced, it is still not low enough to produce sparse connections. Kang and Han [35] propose soft channel pruning (SCP), which assumes that feature maps follow a Gaussian distribution and features are pruned if the cumulative density function is larger than a certain threshold. Hence, threshold operation is relaxed to a continuous form, and the Gumbel-Softmax trick is used to tackle the nondifferentiable Bernoulli distribution sampling. Unfortunately, the assumption of Gaussian distribution for feature maps is too strict to derive accurate gradients. Moreover, SCP shows good pruning performance in network layers that are followed by both batch normalization (BN) and ReLU. According to evaluation results in [35], its pruning performance is severely degraded if only BN exists. Since many neural networks do not include both BN and ReLU, the application scope of SCP is limited. Herrmann et al. [36] jointly consider conditional computation and network pruning. The Gumbel-Softmax trick is used to relax the discrete masks to a continuous form. The researchers focus on dynamic inference using conditional computation. Depending on network inputs, masks are dynamically applied on channels. Unfortunately, these masks are data-dependent, rather than being fixed, and the dynamically pruned structures are not friendly to hardware implementation. Huang and Wang [37] introduce a series of nonnegative scaling factors that are associated with neural network connectivity. To encourage sparse network connectivity, these scaling factors are penalized by an $L_1$-norm regularization term. During the training process, stochastic gradient descent (SGD) and a proposed stochastic accelerated proximal gradient (APG) are used to train weight parameters and scaling factor parameters, respectively. The scaling factor parameters are converted into scaling factors through a soft-threshold operation. Xiao et al. [38] utilize straight-through estimator (STE) to relax binary masks. Even though empirical experiments have shown potential, the fundamental principle of using STE in network pruning has not been explored.

From the above-mentioned discussion, it is clear that existing network pruning methods, either human-designed or automatic, do not fully address the requirement of highly effective sparse connectivity learning. It is attractive to develop new pruning methods to overcome the drawbacks and limitations of existing pruning methods. In this work, we propose a new automatic neural network pruning technique—sparse connectivity learning (SCL). This work makes the following contributions.

1) We theoretically prove the fundamental principle of using STE for network pruning. After finding Leaky ReLU, Softplus, and identity STEs can satisfy this principle, we propose to adopt identity STE in SCL for discrete mask relaxation. Thus, SCL guarantees the convergence of mask variables at their minima.

2) We observe that mask gradients on different features have a wide range of magnitudes and hence are unbalanced. Therefore, we propose to normalize mask gradients of each feature to optimize mask variable training.

3) The pruning principle of our proposed SCL method is the significance of weight, instead of the magnitude of weight. SCL can automatically learn and determine critical network connections of baseline networks [e.g., DenseNets, ResNets, Visual Geometry Groups (VGGs), EfficientNets, and recurrent neural networks (RNNs)].

4) SCL enables highly effective weight-level sparsity learning on neural networks under high pruning rates. Experimental results in the Modified National Institute of Standards and Technology (MNIST), Canadian Institute for Advanced Research-10 (CIFAR-10), CIFAR-100, ImageNet, and WikiText-2 datasets demonstrate the enhanced performance of SCL-induced sparse neural networks than the state-of-the-art network pruning methods in the literature.

5) SCL automatically learns optimized neural network connectivity in a task-aware manner, ensuring that performance-sensitive network connections are ultimately preserved. As it does not need any designer-defined pruning criteria or layerwise pruning hyperparameters, SCL gets rid of the limitation of human designers. Compared with the state-of-the-art pruning methods, experimental results demonstrate that SCL results in high-performance neural networks with higher sparsity and fewer the number of floating-point operations (FLOPs).

Since this work focuses on the algorithm aspect of network pruning, the hardware implementation of sparse models
generated from our proposed pruning algorithm is beyond the scope of this article. Sparse neural network models generated from SCL can be implemented in FPGAs or Kerreson edge AI hardware products. This article is organized as follows. Section II introduces related works about pruning criteria and binary mask relaxation. Section III describes the proposed automatic SCL theory. Section IV introduces the baseline network architectures and experimental setups. Section V demonstrates various experimental results and comparisons with the state-of-the-art works in the literature. Section VI concludes this article.

II. RELATED WORK

A. Human-Designed Pruning Criteria

The performance of human-designed pruning methods depends to a large extent on the quality of pruning criteria defined by designers. Unimportant network connections are removed based on the important measure, which usually follows the assumption that smaller norms are less important. The criterion for unstructured pruning is the absolute value of weights [7], [8]. The pruning criterion for structured pruning is the $L1$-norm [14] or $L2$-norm [16] of filter or channel weights. The magnitude of scaling factors in BN is also proposed as the channel pruning criterion [20]. He et al. [19] do not agree that smaller filters are less important, but propose that the contribution of median filters is relatively small because they can be represented by other filters.

B. Gumbel-Softmax Trick

Automatic pruning involves training binary masks. To address the nondifferentiation problem, the Gumbel-Softmax trick [39], [40] has been used to relax binary masks [34]–[36]. This trick uses gradient methods to train discrete random variables. In this trick, discrete values produced by argmax are encoded in a one-hot vector, and the random sampling $f(U)$ is expressed as

$$f(U) = \text{argmax}(\log \alpha + G(U))$$

$$G(U) = -\log(-\log U), U \sim \mathcal{U}(0, 1)$$

(1) (2)

where the argmax function finds the argument corresponding to the maximum value, $\alpha$ is a set of unnormalized parameters $\alpha_k$, and the probability of outcome $k$ is $\alpha_k / \sum \alpha_k$. Each element in the vector $G(U)$ obeys a Gumbel distribution. As a binary mask has two possible discrete values, mask reparameterization is regarded as a special case. Due to the sparsity consideration, there is a high possibility that mask sampling results should be trained to be zero. As a result, sparse binary masks are obtained. However, the discrete argmax operation cannot be trained by gradient methods. Therefore, one way to circumvent this is to relax the argmax operation by replacing it with softmax [39] and [40] as

$$f(U) = \text{softmax}(\log \alpha + G(U)) / \tau$$

(3)

where $\tau$ is a hyperparameter of the softmax function to control relaxation. When $\tau \to 0$, the softmax function becomes the argmax function. Thus, (3) provides a differentiable form, which is able to train and optimize by gradient methods.

The Gumbel-Softmax trick has two limitations. First, its gradient estimations are biased with respect to the gradients of discrete connectivity. According to (3), this trick leads to unbiased gradients only when the hyperparameter $\tau$ tends to 0. However, because the value of $\tau$ is used to balance bias and variance in practice, $\tau$ is rarely chosen to be close to 0. As a result, the studies in [34]–[36] use biased gradients, which may not meet the essential condition for convergence (i.e., low-biased gradient). Second, the training process is stochastic, but the inference is deterministic. Therefore, the expected number of connections of stochastic training does not reflect the number of connections of deterministic inference. As shown in Section V-D3, the reduction of $L_0$-norm during training does not mean sparser network connections.

C. Straight-Through Estimator Trick

Due to the derivatives of a unit step function are mostly zero, training variables cannot update using gradient-based optimization methods. STE is a trick of using proxy gradients in backpropagation [41]. In this trick, zero gradients of a discrete function are replaced by the derivative of a (sub)differentiable function. Derivatives of ReLU STE and Clipped ReLU STE were used for network quantization [42]–[44]. Yin et al. [45] referred to proxy gradients as coarse gradients and studied the STE properties for network quantization. Yin et al. [45] assumes that network inputs obey a normal distribution, yet, this assumption has not been validated for network pruning. Later, based on [45], Xiao et al. [38] proposed to use leaky ReLU STE and Softplus STE to relax binary masks. Instead of performing network pruning, Hinton [46] proposed to use identity STE for binary neuron training. The (sub)differentiable functions of five existing STEs are expressed as

$$\sigma_{\text{relu}}(x) = \max(0, x)$$

$$\sigma_{\text{clipped relu}}(x) = \min(\max(0, x), \alpha), \alpha > 0$$

$$\sigma_{\text{leaky relu}}(x) = \max(x, \alpha \cdot x), 0 < \alpha < 1$$

$$\sigma_{\text{softplus}}(x) = \log(1 + \exp(x))$$

$$\sigma_{\text{identity}}(x) = x$$

(4) (5) (6) (7) (8)

The STE trick for network pruning has two limitations. First, as Xiao et al. [38] treat connectivity as a hyperparameter, bilevel optimization is used to update weights and masks separately. Because it involves two-pass calculations to complete an update, the pruning optimization method is too cumbersome and computationally expensive. Second, although the use of STE for network pruning has achieved empirical success in [38], yet, it is based on the assumption that network inputs obey a normal distribution. Unfortunately, so far, the fundamental principle of using STE for network pruning is still unclear, which is the focus of this work.

III. AUTOMATIC SPARSE CONNECTIVITY LEARNING

The proposed automatic SCL is briefly described in the following and shown in Fig. 1. First, we represent the weight as a multiplication of a weight variable and a binary mask (0 or 1). The value 1 indicates a network connection, while 0 indicates no network connection. Thus, the connectivity of a neural network can be fully described by the binary mask, which will be further modulated by a unit step function on a mask variable. Second, during the training process, we will decay the network connectivity to gradually push the elements of the binary mask toward zero (i.e., no network connection). Finally, without the need for designer-defined criteria or layerwise pruning hyperparameters, unimportant
network connections will be automatically discovered and removed. The design details of each step will be elaborated in Sections III-A–III-G.

A. Weight Reparameterization

We design to learn a binary mask that is a standard convolution layer and is related to network connectivity. A fully connected layer is a special case of an input with a feature size of $1 \times 1$ convolved with a kernel with a size of $1 \times 1$. The base cell of an RNN is actually composed of several fully connected modules. We model this convolution operation as

$$y = w \ast x$$  \hspace{1cm} (9)

where the convolutional kernel $w$ is weight, $x$ is input, and $y$ is output. $\ast$ annotates the convolution operation. As shown in Fig. 1, a weight is reparameterized by a weight variable $\tilde{w}$ and a binary mask $m$. Weight reparameterization is formulated as

$$w = \tilde{w} \odot m$$  \hspace{1cm} (10)

where $\odot$ is the elementwise multiplication. Mask is binarized from the trainable mask variable $\tilde{m}$ using a unit step function. The binarization is formulated as

$$m = H(\tilde{m}) = \begin{cases} 0, & \tilde{m} \leq 0 \\ 1, & \tilde{m} > 0. \end{cases}$$  \hspace{1cm} (11)

Therefore, the elements of $\tilde{w}$ will be zero-masked if the corresponding elements in $\tilde{m}$ is nonpositive.

B. Gradient Redefinition for Weight Variables

According to the derivative chain rule and (10), the gradient of loss function $L$ with respect to $\tilde{w}$ is written as

$$\frac{\partial L}{\partial \tilde{w}} = \frac{\partial L}{\partial w} \odot m.$$  \hspace{1cm} (12)

As $m$ is a sparse tensor, many elements of $\partial L/\partial \tilde{w}$ are zero, indicating that these masked variables are not updated. Even though network connections that contribute less to precision can be ignored in the current global structure, they may not be negligible in future global structures. As a result, temporary zero masks do not mean unimportant, and it is worth keeping training for these zero-masked weight variables. In this work, the gradient of loss function $L$ with respect to a weight variable $\tilde{w}$ is redefined as

$$\frac{\partial L}{\partial \tilde{w}} := \frac{\partial L}{\partial w}.$$  \hspace{1cm} (13)

where $\partial L/\partial w$ is obtained through backpropagation. Similar to [8] and [16], even if some weight variables are temporarily zero-masked during training, they will update later.

C. Gradient Redefinition for Mask Variables

Due to the derivatives of a unit step function are mostly zero, mask variables cannot update using gradient-based optimization methods. To solve this problem, we investigate the fundamental principle of using STE for network pruning. Let us analyze a realizable case that uses the following loss function to update mask variables $\tilde{m}$:

$$\min_{\tilde{m}} \ell(\tilde{m}) = \frac{1}{2} (H(\tilde{m}) - m^*)^2$$  \hspace{1cm} (14)

where $H(\tilde{m})$ and $m^*$ represent the binarized mask values and optimal mask values, respectively. Note that the optimal solution of mask variables $\tilde{m}$ is a region, rather than a point. The gradient of $\tilde{m}$ is expressed as

$$\frac{\partial \ell}{\partial \tilde{m}} = \frac{\partial H(\tilde{m})}{\partial \tilde{m}} (H(\tilde{m}) - m^*).$$  \hspace{1cm} (15)

As $H(\tilde{m}_i)$ and $m^*_i$ are binary values, there are three possible values (i.e., 0, -1, and +1) for their difference $H(\tilde{m}_i) - m^*_i$. Hence, the gradient of $\tilde{m}_i$ is expressed as

$$\frac{\partial \ell}{\partial \tilde{m}_i} = \begin{cases} \frac{\partial H(\tilde{m}_i)}{\partial \tilde{m}_i} \cdot 0, & \text{if } H(\tilde{m}_i) = m^*_i \\ \frac{\partial H(\tilde{m}_i)}{\partial \tilde{m}_i} \cdot (-1), & \text{if } \tilde{m}_i \leq 0 \text{ and } m^*_i = 1 \\ \frac{\partial H(\tilde{m}_i)}{\partial \tilde{m}_i} \cdot (+1), & \text{if } \tilde{m}_i > 0 \text{ and } m^*_i = 0. \end{cases}$$  \hspace{1cm} (16)

When STE is used to relax the binarization, $\partial H(\tilde{m}_i)/\partial \tilde{m}_i$ is replaced by proxy gradients. Correct proxy gradients should move $\tilde{m}_i$ toward their optimal values during network pruning. The gradient should be zero when the optimal value is reached, indicating no further update. Based on the mechanism of gradient descent optimizers (i.e., a variable is adjusted in the opposite direction of its gradient), $\tilde{m}_i$ should move toward the negative direction of proxy gradients. Consequently, when STE is used for network pruning, positive values of $\partial H(\tilde{m}_i)/\partial \tilde{m}_i$ in (16) ensure mask variables converge at their minima. This fundamental principle is derived without any assumptions and is just based on the mechanism of gradient descent optimizers. In contrast, a normal distribution is assumed for inputs in [38].

Fig. 1. Automatic SCL. Weight reparameterization for sparse convolution, identity STE for masked weight training, identity STE for binary mask relaxation, and mask gradient normalization. The elliptical symbols, “Weight_V” and “Mask_V,” refer to trainable variables, the rectangular symbols indicate intermediate tensors, and the circles indicate calculation operators. Dotted lines mean gradient process during backward propagation.
Fig. 2. Proxy gradients of five existing STEs used for binarization relaxation. The x-axis and y-axis represent mask variables $\tilde{m}$ and proxy gradients $\partial H(\tilde{m})/\partial \tilde{m}$, respectively. The dead zones for ReLU and clipped ReLU STEs are marked. (a) ReLU STE. (b) Clipped ReLU STE. (c) Leaky ReLU STE. (d) Softplus STE. (e) Identity STE.

Fig. 2 plots proxy gradients of the existing five STEs. We can see that three STEs (i.e., Leaky ReLU, Softplus, and identity) can satisfy the fundamental principle of positive proxy gradients. In this work, the identity STE is selected for simplicity. Since its proxy gradient is always 1, as shown in Fig. 2(e), (16) is expressed as

$$\frac{\partial \ell}{\partial \tilde{m}_i} = \begin{cases} 0, \text{ if } H(\tilde{m}_i) = m^*_i \\ -1, \text{ if } \tilde{m}_i \leq 0 \text{ and } m^*_i = 1 \\ +1, \text{ if } \tilde{m}_i > 0 \text{ and } m^*_i = 0. \end{cases}$$

As indicated in (17), when $\tilde{m}_i$ reaches its optimal value, the gradient is zero. As a result, $\tilde{m}_i$ does not update further. When $\tilde{m}_i \leq 0$ and $m^*_i = 1$, the gradient of $-1$ pushes $\tilde{m}_i$ to be more positive. When $\tilde{m}_i > 0$ and $m^*_i = 0$, the gradient of $+1$ pushes $\tilde{m}_i$ to be more negative. Thus, mask variables and network connectivity (from 1 to 0 or vice versa) can update during training. The Identity STE ensures that mask variables move toward and finally stabilize at their optimal values. According to (11) and Fig. 2(e), we obtain

$$\frac{\partial m}{\partial \tilde{m}} = \frac{\partial H(\tilde{m})}{\partial \tilde{m}} = 1$$

which indicates that the differential of $\tilde{m}$ is redefined as that of $m$. Thus, the gradient of loss function $L$ with respect to mask variables $\tilde{m}$ is redefined as

$$\frac{\partial L}{\partial m} = \frac{\partial L}{\partial \tilde{m}} \frac{\partial \tilde{m}}{\partial m} := \frac{\partial L}{\partial \tilde{m}} \odot \tilde{w}.$$ 

D. Mask Gradient Normalization

We observe a wide range of mask gradient magnitudes in various layers and channels of common neural networks. To visualize this observation, a pretrained ResNet-110 is used as an example to plot mask gradient magnitudes in Fig. 3. In Fig. 3(a), layerwise results show that the average mask gradient magnitude of the first and last layers (e.g., $10^{-5}$) is much higher than other layers (e.g., $10^{-7}$). Then, channelwise results of two typical layers in Fig. 3(b) and (c) show that mask gradient magnitudes fluctuate greatly between channels.

The influence of a wide range of mask gradient magnitudes is analyzed in the following. Let us review the training process of mask variables during network pruning. Through binarization (i.e., (11)), a positive mask variable means a mask state of 1, indicating that there is a network connection. If this network connection is pruned, the binary mask state should become 0. According to (11), the final value of this mask variable should be zero or negative. Assume two mask variables are initialized to the same positive value and eventually become zero. That is, two initially established network connections are eliminated after pruning. Under the same learning rate for all layers, the mask variable with a smaller gradient magnitude requires more training iterations to update until convergence. Therefore, it is difficult for mask variables with a wide range of mask gradient magnitudes to converge to their optimal solution, thereby degrading network pruning performance. To mitigate this problem, we propose to normalize mask gradients on different features. Mask gradients of each feature are normalized to the unit variance in each mini-batch. As shown in (20), mask gradients obtained through backpropagation are divided by a gradient scale $s$

$$\text{Norm} \left( \frac{\partial L}{\partial m_j} \right) = \frac{\partial L}{\partial w_j} \odot \tilde{w}_j / (s + \epsilon), \text{ where}$$

$$s = \sqrt{\sum_{x_b \in B} \sum_{w_j} \left( \frac{\partial L(x_b)}{\partial w_k} \odot \tilde{w}_k \right)^2} / \|B\| \cdot |w_j|$$

where $B$ and $|B|$ denote the sampled mini-batch data and batch size, respectively. $w_j$ and $|w_j|$ represent the weight and weight number of the $j$th feature, respectively. $\epsilon$ is a small positive constant to avoid division by zero. Since it is not a zero-mean normalization, the sign of each mask gradient does not change; therefore, the analysis derived in Section III-C is still valid. Because the same gradients are produced on all masks by the connectivity decay introduced in the Section III-E, the gradient normalization excludes mask gradients caused by the connectivity decay [i.e., normalization in (20) processes the gradients produced by the first and third terms of (23)].

E. Connectivity Decay for Sparse Connectivity Learning

Through the proposed weight reparameterization, the information of network connectivity is completely represented by the elements in the binary mask $m$. The degree of network connectivity is equal to the sum of all element values in $m$. We incorporate the degree of connectivity into an objective function to optimize network connectivity. The training objective function is expressed as

$$L = \mathcal{C} + \lambda_1 \cdot \sum_{l=0}^{L} \sum_{i=0}^{\#m^{(l)}} m_i^{(l)}$$

where $\mathcal{C}$ is the criteria to measure the performance loss, $m^{(l)}_i$ is the mask of $l$th layer, $i$ is the element index, $\#m^{(l)}$ is the total number of elements in $l$th layer, and $L$ is the total number of layers. $\lambda_1$ is a hyperparameter for connectivity decay. As the second term includes the information of network
connectivity, so during training this term attenuation indicates connectivity decay. Connectivity decay can be viewed as a way of \(L_0\) regularization of weight. There are two gradients that affect the training of mask variables. One gradient is due to the connectivity decay, which pushes mask variables moving toward the negative infinity direction all the time. The other gradient is due to the performance loss minimization, which usually pushes mask variables moving toward the positive infinity direction according to the significance of network connectivity. When training converges, both gradients on the mask variables will reach equilibrium at important network connections. Therefore, an optimized neural network with an expected level of sparsity is learned and determined through training.

F. Proposed SCL Algorithm

Algorithm 1 describes the details of our proposed automatic SCL method. Through repeatedly calculating gradients of weight variables and mask variables, this algorithm updates them through SGD. When the network pruning process is complete, a sparse neural network is obtained. The well-trained sparse weights are calculated as

\[
\mathbf{w}^* = \tilde{\mathbf{w}} \odot H(\tilde{\mathbf{m}}). \tag{22}
\]

G. Comparison of SCL With Existing Automatic Pruning

It is necessary to comprehensively compare our SCL algorithm with existing automatic pruning methods in the literature (i.e., [34]–[38]). Therefore, conceptual comparison and experimental results discussion of [34], [35], [37], and [38] will be provided in Sections V-D and V-E. Since [36] does not report the proper experimental result to compare, we only discuss their algorithm differences as follows. Compared with [36], SCL has the following differences. First, the way of dealing with the nondifferentiation problem of discrete masks is different. SCL uses a deterministic STE to estimate the gradient of discrete masks, whereas [36] uses the Gumbel-Softmax trick to relax the discrete masks to a continuous form. Second, the way to reduce the computational workload is different. Herrmann et al. [36] focuses on conditional computation using dynamic inference; therefore, masks vary with network inputs. Besides, since the efficiency of data handling is greatly affected, the conditional computational graph in [36] is not friendly to hardware accelerators. In contrast, SCL focuses on network pruning that is static during inference; therefore, the computational graph is fixed after training. Therefore, the feature of static computation reduction in SCL is more appropriate than [36] for hardware acceleration.

Algorithm 1 Automatic SCL

\begin{enumerate}
\item \textbf{Input:} A training dataset \(D\), a \(L\)-layer neural network, weight variables \(\mathcal{W} = \{\mathbf{w}^{(l)}\}_{l=1}^{L}\), mask variables \(\mathcal{M} = \{\tilde{\mathbf{m}}^{(l)}\}_{l=1}^{L}\), a connectivity decay coefficient \(\lambda_1\), an \(L_2\) regularization coefficient \(\lambda_2\), and the total number of training epochs \(T\).
\item \textbf{Output:} Well-trained sparse weights \(\mathbf{w}^* = \{\mathbf{w}^{(l)*}\}_{l=1}^{L}\).
\item \textbf{Initialization:} Initial values of mask variables are positive. Initial values of weight variables are random according to [47].
\item \textbf{repeat}
\item Sample a mini-batch from \(D\) as input data.
\item \textbf{Forward pass:} First, calculate all the masks using Eq. (11). Next, compute all weights using Eq. (10). Then, Calculate all layers like a normal neural network with weights and sampled input.
\item \textbf{Backward pass:} First, calculate weight gradients using Eq. (13). Next, relax mask gradients through the Identity STE using Eq. (19). Then, normalize mask gradients using Eq. (20).
\item \textbf{Update weights:} Use gradients obtained from the backward pass to update all weight variables and mask variables via SGD.
\item \textbf{until} \(T\) epochs complete
\item \textbf{7: Compute the well-trained sparse weights using Eq. (22) and output a sparse neural network.}
\end{enumerate}

IV. Baseline Network Architectures and Experimental Setups

We conduct the experiments on four NVIDIA Titan XP GPUs using PyTorch.\footnote{https://pytorch.org} The evaluation codes and trained models can be downloaded from Google Drive.\footnote{https://drive.google.com/drive/folders/1tuddm577qw88sOlATOF7Ms04m6Vlhf6zg?usp=sharing} We choose VGGs [3], ResNets [4], DenseNets [5], and EfficientNets [48] as our baseline CNN architectures, because most of the state-of-the-art neural networks are based on them. ResNets and DenseNets are outstanding due to their high accuracy and fewer numbers of trainable parameters. EfficientNet is a lightweight high-accuracy convolutional neural network architecture with much fewer parameters. In CNN experiments, the proposed SCL technique is evaluated using the datasets of MNIST [49], CIFAR [50], and ImageNet (i.e., full data of ImageNet2012 classification) [51]. In RNN experiments,
SCL is evaluated on a language model using the WikiText-2 dataset [52].

For the experiments on the MNIST dataset, the baseline is a DenseNet-based network, where only fully connected layers are implemented (i.e., no convolutional networks involved) to evaluate the proposed SCL method. Then, two-dimension 28 × 28 image samples are flattened before being fed into the neural network. The input images are normalized using the channel mean and standard deviation. Within this baseline network, 16 fully connected layers with a growth rate of 8 are applied to feature extraction, followed by a softmax function for object classification. For the experiments on the CIFAR dataset, convolutional networks are evaluated. The VGGs and ResNets are adapted from the codes of [53]. The codes of DenseNets are adapted from the codes of [53]. The codes of DenseNets are slightly different from the description in this article of [5], which is referred to as official codes. For the experiments on the ImageNet dataset, we only evaluate SCL on VGG-16, ResNet-50, and EfficientNet-B0 due to the limitation of computational resources. The language model used in the RNN experiments consists of an encoder embedding, two LSTM layers, and a decoder embedding. As encoder and decoder embeddings are tied to improve the perplexity results [54], we only account for one of the encoder/decoder embeddings in sparsity statistics. The vocabulary size and LSTM hidden layer size are 33 278 and 1500, respectively. The language model used in the RNN experiments is adapted from a PyTorch word language model.

All the networks are trained by the SGD optimizer. We adopt the weight initialization method in [47], and BN in [55] for fast training. The objective function of an \( L \)-layer network is expressed as

\[
\mathcal{L} = \mathcal{C} + \lambda_1 \sum_{l=0}^{L} \sum_{i=0}^{m(l)} m^{(l)}_i + \lambda_2 \sum_{l=0}^{L} \sum_{i=0}^{\tilde{m}(l)} (\tilde{w}^{(l)}_i)^2 \tag{23}
\]

where \( \mathcal{C} \) is the cross entropy loss for classification, and \( \lambda_1 \) and \( \lambda_2 \) are the coefficients of connectivity decay and \( L_2 \) regularization, respectively. \( m^{(l)}_i \) and \( \tilde{w}^{(l)}_i \) refer to the number of elements in \( m^{(l)} \) and \( \tilde{w}^{(l)} \), respectively. Note that \( \lambda_1 \) and \( \lambda_2 \) are not designer-defined pruning criteria or layerwise pruning hyperparameters (e.g., pruning threshold or ratio), which are indispensable in existing pruning methods in the literature. In our proposed SCL method, the layerwise sparsity is automatically determined by the learning algorithm itself, without the intervention of designers. In our SCL method, tuning \( \lambda_1 \) can adjust the tradeoff between network sparsity and accuracy. Therefore, designers can adjust \( \lambda_1 \) to reach the desired sparsity-accuracy balance. For a given value of \( \lambda_1 \), we do not run full training iterations to check the resulting sparsity. Instead, we run a small portion (e.g., 10%) of the full training iterations to see if the target sparsity can be achieved. In this way, a proper \( \lambda_1 \) for the target sparsity can be found with several attempts. To further reduce the trial time, we start with a larger value of \( \lambda_1 \), which leads to faster network pruning and hence shorter running time. Then, we decrease the value of \( \lambda_1 \) until the target sparsity is obtained. Therefore, the running time for tuning \( \lambda_1 \) is not huge. A similar process for tuning \( \lambda_1 \) is applied to other datasets and baseline architectures in the experiments of Section V. To make a concise comparison with existing works, we list the experimental results in terms of sparsity, the number of parameters, FLOPs, and accuracy in Tables in Section V. Note \( \lambda_1 \) is the coefficient of \( L_2 \) regularization, which has nothing with network pruning. \( L_2 \) regularization is required to ensure an effective learning rate when BN is applied [56].

For experiments on the MNIST and CIFAR datasets, mini-batch size and initial learning rate are set to 64 and 0.1, respectively. For experiments on the MNIST dataset, a simple training schedule is used, and mask variables do not update in the first and last 15 epochs. For experiments on the CIFAR datasets, mask variables do not update in the first 150 epochs to facilitate weight convergence. Then, mask variables are updated to obtain the corresponding sparse network connectivity. The learning rate is adjusted to 0.01 when the target sparsity is almost achieved. Finally, the mask variables continue to update for 20 epochs and achieve stable network connectivity. In order to achieve ultimate convergence stability, the weights continue to update for 80 epochs with a learning rate of 0.01 and then another 80 epochs with a learning rate of 0.001. Initial values of mask variables are positive. Weight variables are randomly initialized according to [47] and trained from scratch. A larger \( \lambda_1 \) usually leads to less number of training epochs for realizing the same sparsity expectation. For experiments on the ImageNet dataset, all the models are pretrained before sparse training for saving time. Mini-batch size and initial learning rate are set to 128 and 0.005, respectively. We use 90 epochs for sparse connectivity training and another 60 epochs to achieve stable training convergence. In the RNN experiments, we follow the training procedure of Distill [10].

V. EXPERIMENTAL RESULTS AND COMPARISON

A. Effect of STE and Mask Gradient Normalization

An input–output mapping experiment has been designed to evaluate the effect of STE and mask gradient normalization. A three-layer fully connected network is built as a baseline for input–output mapping. Each network layer consists of 64 neurons, followed by BN and ReLU activation. In order to ensure output convergence, weights are randomly initialized according to [47]. A set of inputs and mask variables are randomly selected and applied to this baseline. These inputs and mask variables follow a normal distribution with zero-mean and unit standard deviation. Thus, without any training, output results of the baseline are obtained as a benchmark.

Next, the baseline architecture is reused to perform output fitting experiments, where inputs and weights are the same as the baseline, but mask variables are randomly initialized and then trained through gradient descent to match the benchmark (i.e., baseline outputs). These fitting experiments involve different STEs (i.e., proxy mask gradients) for binary mask relaxation with or without mask gradient normalization. In these experiments, the ideal fitting result is that after training, mask variables finally converge and produce the same outputs as the benchmark, indicating an accurate reproduction of the baseline input–output mapping.

Fig. 4 shows the mean squared errors (MSEs) obtained from the above-mentioned experiments. MSE tells how close
As shown in Table I, a larger connectivity decay network with the MNIST database, as described in Section IV, method, we carried out experiments using a DenseNet-based MNIST B. Sparse Connectivity of Fully Connected Network on
validating the necessity of normalizing mask gradients. Mask gradient normalization reduces MSEs for all STEs, descent direction to push mask variables toward the minima. Positive proxy gradients of these three STEs guarantee a loss consistent with (16) and theoretical analysis in Section III-C. STEs achieve better and similar results. This observation is ReLU and ReLU STEs, leaky ReLU, Softplus, and identity zone, there is no chance to get out. Compared with the Clipped ReLU STE leads to the worst fitting results, because its gradient is 0 when a mask variable is negative or larger than a threshold, as shown in Fig. 2(b). The results of ReLU STE are better, because it has fewer dead zones, as shown in Fig. 2(a). Note that once a mask variable falls into a dead zone, there is no chance to get out. Compared with the Clipped ReLU and ReLU STEs, leaky ReLU, Softplus, and identity STEs achieve better and similar results. This observation is consistent with (16) and theoretical analysis in Section III-C. Positive proxy gradients of these three STEs guarantee a loss descent direction to push mask variables toward the minima. Mask gradient normalization reduces MSEs for all STEs, validating the necessity of normalizing mask gradients.

B. Sparse Connectivity of Fully Connected Network on MNIST

To quickly evaluate the effectiveness of the proposed SCL method, we carried out experiments using a DenseNet-based network with the MNIST database, as described in Section IV. As shown in Table I, a larger connectivity decay $\lambda_1$ corresponds to more sparse network connectivity. Even if $\lambda_1$ is set to 0, our experimental results show that the two gradients due to performance loss minimization and $L_2$ regularization in (23) can push some mask variables to negative values, when their corresponding network connections do not contribute to accuracy. As a result, our SCL-induced neural network has a good sparsity of 34%, and its object classification accuracy of 98.47% outperforms the DenseNet-based baseline network (i.e., 98.35%). From a system accuracy perspective, this means that the best network connection for this example is inherently sparse. In addition, when $\lambda_1$ is 0.03, our SCL-induced sparse networks can achieve very high accuracy of above 98% and sparsity of about 96.2%. Note that in this fully connected layer experiment, the sparsity is equal to the reduction in FLOPs. As a result, the SCL-induced DenseNet-based network achieves a 96.2% reduction in FLOPs (i.e., approximately $26.3 \times$ lower than the baseline with an accuracy loss of only 0.34%). This experiment validates that our proposed SCL method supports effective network pruning on fully connected layer architectures.

Based on these experimental results, we add all input-related binary mask connections and normalize them. The normalized connections for three typical $\lambda_1$ values are shown in Fig. 5. It is clear that network connections are mainly concentrated in central areas; therefore, our SCL training ignores edges. As a result, the pixels of digits are trained to be connected in the central area. When $\lambda_1$ is set to 0.1 (i.e., very high sparsity), only a few central pixels are connected, as shown in Fig. 5(a).

C. Connectivity Learning Analysis on CIFAR-10

In order to demonstrate the automatically learned network connectivity, we report and discuss the density profiles (i.e., the percentage of remaining network connections) of several SCL-induced networks on the CIFAR-10 dataset. Fig. 6 shows that the proposed SCL method can automatically learn and determine the corresponding density profile for each network layer. In Fig. 6(a), there are three layers with relatively high densities, which are the layers located before the three dense blocks. Since it is most often reused in this low-density DenseNet network, the three important network layers should retain more connections. In Fig. 6(b), when this DenseNet network has a high targeted density of about 30%, most of the network layers obtained by our SCL method have a similar density. Note that the density of the last network layer is completely different in Fig. 6(a) and (c). This is due to different types of redundancy between DenseNet and ResNet structures. In DenseNet-40, the fact that previously extracted features are channelwise concatenated to subsequent layers leads to 456 channel connections to the last layer. In ResNet-110, previously extracted features are elementwise added to subsequent layers; therefore, the last layer contains only 64 channel connections and there is less redundancy. Therefore, DenseNet-40 removes most weights in the last layer, whose density is about 7% in Fig. 6(a), while ResNet-110 retains most weights in the last layer, whose density is about 65% in Fig. 6(c). In Fig. 6(c) and (d), since ResNet-110 is too deep, network connections in some layers are completely removed. In Fig. 6(d), even though 70% of weights and some computational expensive shallow layers are removed, our SCL-induced network is superior to the baseline networks (94.82% versus 93.57% in Table III). In summary,
we find that unlike conventional pruning methods, which require designer-defined pruning criteria or hyperparameters for each layer, our proposed SCL method can automatically learn and select important network connections for given baseline structures.

Fig. 6. Connection density profiles for all layers in DenseNet-40 and ResNet-110 for different overall densities. (a) DenseNet-40 profile (overall density $\approx 10\%$). (b) DenseNet-40 profile (overall density $\approx 30\%$). (c) ResNet-110 profile (overall density $\approx 10\%$). (d) ResNet-110 profile (overall density $\approx 30\%$).

D. Comparison With the State-of-the-Art Pruning Methods on CIFAR-10 and CIFAR-100

In order to comprehensively evaluate the proposed SCL method, we compare it with the state-of-the-art pruning methods in the literature, including both nonstructured and structured methods on the baseline networks of VGGs, ResNets, and DenseNets.

1) Comparison With Nonstructured Pruning Methods:
We compare the proposed SCL with nonstructured pruning methods, e.g., Frankle and Carbin [13] and Han et al. [7]. The results of Frankle and Carbin [13] (i.e., lottery ticket hypothesis) are copied from Liu et al. [53]. As shown in Fig. 7, we use a very high sparse target ($97\%$) to learn sparse VGGs. Our SCL-induced VGGs are superior to the state-of-the-art pruning methods of [7], [13], [14], and [20] in both sparsity and accuracy. For DenseNet-BC-100, with the same sparsity of $80\%$, our SCL method results in an accuracy of $95.40\%$, which is much higher than the accuracy of $95.04\%$ in Han et al. [7]. Even though we do not compare FLOPs here because we are unable to access pretrained models in these previous works, the reduction in FLOPs is positively correlated with the sparsity for nonstructured sparse models. Compared with the human-determined criterion of nonstructured pruning

Fig. 7. Sparsity comparisons with nonstructured pruning methods. (a) VGG-16, comparisons with Frankle and Carbin [13] and Li et al. [14]. (b) VGG-19, comparisons with Han et al. [7] and Liu et al. [20]. (c) DenseNet-BC-100, comparison with Han et al. [7].
of weights. Hence, this work is simple and efficient. Third, the actual reduction of sparse connectivity and FLOPs is different. Both training and testing stages in this work are deterministic, whereas training and testing stages in [34] are stochastic and deterministic, respectively. Note a mask variable implies no connectivity only when its probability is zero. In this work, the probability of connectivity is either 0% or 100%, a reduction of \( L_0 \)-norm of masks (i.e., probability of certain connectivity is regulated from 100% to 0%) in the training stage indicates the same reduction in the testing stage. In contrast, the probability of connectivity is continuous from 0% to 100% in the training stage of [34]. Hence, in [34], the reduction of expected \( L_0 \)-norm during training does not necessarily mean a reduction of \( L_0 \)-norm during testing. Besides, the discrete mask function is used in [34] for hard pruning in the testing stage. Discrete mask function potentially has a large discrepancy from the probabilistic continuous mask function during training time [35]. Furthermore, the use of stochastic sampling in [34] leads to a huge gap between the expected \( L_0 \) in the stochastic training phase and the actual \( L_0 \) in the deterministic testing phase. In [34], although the hard concrete distribution trick allows zero gates to be produced, the expected \( L_0 \) cannot reflect the actual \( L_0 \) during inference.

Our proposed SCL takes advantage of STE [41] to redefine gradients of mask variables. Even though STE involves the use of STE in stochastic neurons, we think STE is also applicable to deterministic neurons, because deterministic neurons are regarded as a special case of stochastic neurons with a probability of either 0% (i.e., no connection) or 100% (i.e., with connection). The use of STE on deterministic problems has been empirically verified by the deep learning community [38], [45]. Then, \( L_0 \) norm of weights is integrated as a regularization term of the objective function. Despite the role of SCL in training is similar to that of \( L_0 \) regularization, we think SCL is advantageous because it is efficient processing of mask variables helps sparsity training. As a result, the proposed SCL method is more direct and effective in encouraging sparse network connections. Table V lists the experimental results of \( L_0 \) regularization [34] and SCL on the CIFAR-10 and CIFAR-100 datasets. The observation that both of them outperform the baseline model [58] indicates that the best network connections should be sparse. Moreover, the experimental results of SCL with three sparsity levels (i.e., 20%, 50%, and 90%) are provided. Their corresponding accuracy results are better than those of \( L_0 \) regularization, even though the sparsity results are not reported in [34]. In fact, when we use the hyperparameters and codes in [34] to repeat the experiment for Table V, we find that the obtained network is not sparse. Although the expected \( L_0 \) norm of weights has been significantly reduced, it is still not low enough to generate sparse connections.

### Table II

| Scheme   | # Param. | Sparsity | FLOPs ↓ | Accuracy |
|----------|----------|----------|---------|----------|
| Baseline [4]       | 0.268M  | 0%       | 0%      | 91.25%   |
| He et al. [16]    | 0.241M  | 10%      | 15%     | 92.24%   |
| He et al. [16]    | 0.214M  | 20%      | 29%     | 91.20%   |
| He et al. [16]    | 0.185M  | 30%      | 42%     | 90.83%   |
| SCL                | 0.133M  | 50%      | 40%     | 92.61%   |
| SCL                | 0.080M  | 70%      | 69%     | 92.35%   |

### Table III

| Scheme   | # Param. | Sparsity | FLOPs ↓ | Accuracy |
|----------|----------|----------|---------|----------|
| Baseline [4]       | 1.72M   | 0%       | 0%      | 93.57%   |
| He et al. [16]    | 1.20M   | 30%      | 41%     | 93.86%   |
| Yu et al. [57]    | 0.98M   | 43%      | 44%     | 93.39%   |
| Yu et al. [57]    | 1.17M   | 32%      | 39%     | 93.34%   |
| Li et al. [14]    | 1.17M   | 32%      | 39%     | 93.36%   |
| Prankle et al. [13]| 1.17M  | 32%      | 39%     | 93.15%   |
| SCL                | 0.51M   | 70%      | 73%     | 94.82%   |
| SCL                | 0.17M   | 90%      | 90%     | 94.56%   |

### Table IV

| Scheme | # Param. | Sparsity | FLOPs ↓ | Accuracy |
|--------|----------|----------|---------|----------|
| Baseline [5] | 1.04M | 0%      | 0%      | 94.76%   |
| Liu et al. [20] | 0.66M | 36%     | 28%     | 94.81%   |
| Liu et al. [20] | 0.35M | 65%     | 55%     | 94.35%   |
| SCL    | 0.62M   | 40%      | 38%     | 94.81%   |
| SCL    | 0.30M   | 71%      | 70%     | 94.66%   |
| SCL    | 0.10M   | 90%      | 88%     | 94.53%   |
TABLE V
COMPARISON WITH \( L_0 \) REGULARIZATION [34] ON CIFAR DATASETS.
WRN-28-10 [58] IS USED AS THE BASELINE. “-” INDICATES RESULTS NOT REPORTED.

| Scheme       | CIFAR-10 Sparsity | CIFAR-10 Accuracy | CIFAR-100 Sparsity | CIFAR-100 Accuracy |
|--------------|-------------------|------------------|-------------------|-------------------|
| Baseline [58]| 0%                | 96.06%           | 0%                | 80.75%            |
| Louizos et al. [34] | - | 96.07%           | -                 | 80.96%            |
| Louizos et al. [34] | - | 96.17%           | -                 | 81.25%            |
| SCL          | 20%               | 96.36%           | 20%               | 81.79%            |
| SCL          | 51%               | 96.53%           | 50%               | 81.87%            |
| SCL          | 91%               | 96.33%           | 90%               | 81.51%            |

TABLE VI
COMPARISON RESULTS OF DENSENET-BASED NETWORKS ON MNIST

| Scheme | Sparsity | Leaky ReLU | Softplus | Identity |
|--------|----------|------------|----------|----------|
| w/o norm | 95.0%   | 98.25% [38] | 98.31% [38] | 98.37% |
| w/ norm  | 95.0%   | 98.33%    | 98.35%   | 98.39% (SCL) |

4) Comparison With Existing STE-Based Pruning Method:
In addition to evaluating the effect of STE and mask gradient normalization in a mapping experiment in Section V-A, we also conduct experiments with DenseNet-based networks on MNIST and with ResNet-20 and ResNet-110 on CIFAR-10, respectively. We compare the results of SCL with the existing STE-based pruning method [38], which uses leaky ReLU or Softplus STEs without mask gradient normalization. As shown in Tables VI and VII, under the same 95% sparsity, the result of SCL (i.e., 90.08%) is superior to [38] (i.e., 88.01% or 87.95%). Table VII also provides the comparison results of VGG-16 on CIFAR-10, as reported in [38].

E. Comparison With State-of-the-Art Pruning Methods on ImageNet

In neural networks, most of the weights are in fully connected layers, while most FLOPs are in convolutional layers. The number of fully connected layers for VGG-16 and ResNet is three and one, respectively. In Tables VIII and IX, except for Han et al. [7], the performance results of existing pruning methods on ImageNet are copied from Lin et al. [18].

As shown in Table VIII, the structured pruning methods of [15], [18], and [59] on VGG-16 lead to low sparsity and significant computational cost reduction, because they only prune the convolutional layers for fewer FLOPs and do not prune fully connected layers. On the other hand, the nonstructured pruning methods, i.e., Han et al. [7] and our proposed SCL in this study, can obtain higher weight compression ratios, mainly due to the pruning of fully connected layers. Specifically, the method of [7] achieves a weight sparsity of 92.5% over its baseline. At a sparsity level of around 90%, even though the performance (i.e., TOP-1 and TOP-5 drop) of our proposed SCL is worse than [7], our SCL method can achieve a much higher accuracy of 69.20% over 68.66% in [7] and a more FLOPs reduction of 87% over 79% in [7]. The difference between our proposed SCL and [7] is as followed. The pruning efforts in [7] are mainly for fully connected layers, but less for pruning convolutional layers. In contrast, depending on the weight significance, the proposed SCL can efficiently perform pruning on both fully connected layers and convolutional layers. We also observe that the baseline of [7] may not converge, because its baseline TOP-1 accuracy is reported as 68.5%, and our TOP-1 accuracy is larger than 71.5%. The performance gain in [7] is attributed to the use of hundreds of training epochs during pruning, which leads to a much better convergence.

As shown in Table IX, for low-sparisty pruning, these cutting-edge pruning methods [14], [15], [18], [25], [37], [59] show significant performance degradation (i.e., TOP-1, TOP-5) than our SCL-induced results. Furthermore, our SCL method achieves a high sparsity of 74% with a FLOPs reduction of 79%, while retaining a slight performance degradation.

Compared with [33], this work has three main differences. First, our SCL method is pruning criterion-free, whereas [33] intends to learn proper layerwise pruning criterion from a set of designer-defined criteria. Due to the criterion-free nature, our SCL method does not need hyperparameters in criterion. In contrast, [33] needs hyperparameters to determine the number of filters to keep. Second, the pruning performance of [33] heavily depends on human experience, including designer-defined hyperparameters and criteria set. In contrast, our SCL method automatically learns the optimized network connectivity in a task-aware manner. Third, binary masks and weight parameters are jointly updated in SCL, whereas mask and weight parameters are trained separately in [33]. As shown in Table IX, this work shows better network pruning results than [33]. SCL achieves a much lower drop of TOP-1 accuracy (i.e., 0.23%) than [33] (i.e., 1.69%) and a much lower drop of TOP-5 accuracy (i.e., 0.40%) than [33] (i.e., 0.83%), meanwhile largely reducing the number of FLOPs (i.e., 79%) than [33] (i.e., 61%).

Compared with [37], this work has three main differences. First, our SCL method uses binary masks to represent network connectivity, whereas [37] uses continuous scaling factors to represent network connectivity. In the proposed SCL method, no threshold is needed to train binary masks. In contrast, soft-threshold needs to be trained to obtain nonnegative scaling factors in [37]. In [37], if the value of a scaling factor is zero, it indicates no network connection. In contrast, our SCL method automatically learns the optimized network connectivity in a task-aware manner. Second, our SCL method uses scheduled SGD to train the binary masks, whereas [37] uses APG to train the scaling factor parameters. Third, [37] only prunes convolutional layers to reduce FLOPs, while our SCL method prunes convolutional layers and fully connected layers to compress the weight size. As a result, the SCL method leads to much higher sparsity in network connectivity, and therefore, fewer trainable parameters. As shown in Table IX, this work shows better network pruning performance than [37]. SCL achieves a much lower drop of TOP-1 accuracy (i.e., 0.23%) and a much lower drop of TOP-5 accuracy (i.e., 0.40%) than [37], meanwhile reducing the number of parameters by 58% (i.e., 6.6M versus 15.6M).

Even though this work and [38] are all inspired by the general concept of STE, the objective function optimization,
update rule for mask parameters, and coarse gradient estimation used in this work are significantly different from [38]. In this work, the weight and mask parameters are updated in the same optimization iteration. Yet, the weight and mask parameters are updated separately in [38], which corresponds to high computational complexity. As a result, the training calculation cost of SCL is significantly reduced. In this work, the update rule is not based on assumptions, and the update rule for mask parameters is derived through the backpropagation algorithm. In contrast, gradients in [38] are modified by dividing true mask gradients (i.e., gradients obtained through backpropagation) by the absolute value of weights elementwisely. Two coarse gradient estimations (i.e., gradients of leaky ReLU and Softplus function) are used in [38], whereas the straight-through gradient estimation (i.e., gradient of identity function) is used in this work. As shown in Table IX, in addition to the significant reduction of FLOPs (i.e., 79% in this work versus 55% in [38]), our SCL pruning method achieves a much lower drop of TOP-1 accuracy (i.e., 0.23% in this work versus 0.40% in [38]). These experimental results demonstrate that our SCL method is better than [38].

Compared with [35], this work has two main differences. First, the SCP method in [35] assumes that feature maps follow a Gaussian distribution. Yet, this assumption is too strict to derive accurate gradients. In contrast, our SCL method does not rely on any assumptions. Our SCL method trains network connectivity through STE gradient estimation and gradient backpropagation. Second, the SCP method in [35] has a good pruning performance in network layers that are followed by BN and ReLU. However, there are no BN and ReLU in many deep neural networks. For example, BN does not exist in the VGGs network. In the architectures of MobileNets or EfficientNets, some convolutional layers are only followed by BN, rather than BN and ReLU. As shown in [35, Table 5], the network pruning performance is significantly degraded if only BN exists for channel pruning. As a result, the SCP method in [35] does not show good pruning performance for many neural networks. In contrast, experimental results in this work show that our SCL method is applicable to various neural networks, including VGGs, DenseNets, ResNets, EfficientNets, and RNNs. As shown in Table IX, SCL shows better network pruning performance than [35]. SCL achieves a much lower drop of TOP-1 accuracy (i.e., 0.23%) than [35] (i.e., 1.69%) and a much lower drop of TOP-5 accuracy (i.e., 0.40%) than [35] (i.e., 0.98%), meanwhile largely reducing the number of FLOPs (i.e., 79%) than [35] (i.e., 54%).

We also apply SCL to the EfficientNet [48] architecture to learn sparse connections. Unlike VGG, ResNet, and DenseNet architectures that are designed by humans, the EfficientNet architecture is automatically determined by a neural architecture search technique. Depthwise convolution is widely used in EfficientNet architectures to improve efficiency. As shown in Table X, three pruning polices are used by us for depthwise convolution in the experiments of [11] on the ImageNet dataset. In the first pruning policy, the pruning rates of the convolution, depthwise convolution, and classifier layers are set to 0.2, 0.1, and 0.2, respectively. In the second pruning policy, the pruning rates of the convolution, depthwise convolution, and classifier layers are set to 0.5, 0.5, and 0.5, respectively. In the third pruning policy, the pruning rates of the convolution, depthwise convolution, and classifier layers are set to 0.5, 0.125, and 0.6, respectively. Compared with the baseline EfficientNet-B0, Table X demonstrates that when the sparsity is low (such as 19.5% and 25.8%), both the pruning method of Zhu and Gupta [11] and SCL show a negligible accuracy degradation. According to the magnitude of weights, the work of Zhu and Gupta [11] forces smaller weights to zero. As a result, although it is an elementwise pruning method, if the difference of weight magnitudes between depthwise channels is large, the pruning method of Zhu and Gupta [11] tends to completely remove certain depthwise convolution channels. The second policy of Zhu and Gupta [11] sets the weights of all layers to be pruned by 50%, experimental results show that FLOPs are reduced by 60% with a big TOP-1 accuracy drop of 0.79%. The third policy of Zhu and Gupta [11] prunes less weights of depthwise layers and obtains a better accuracy. These pruning results of Zhu and Gupta [11] show that it is difficult to find an appropriate pruning policy to obtain satisfactory pruning results. In contrast, since the network connectivity learned by SCL is determined by the significance of weight, SCL automatically retains necessary depthwise channels even if their magnitudes of weight are small. As a result, when SCL prunes more weights than the pruning method of Zhu and Gupta [11] (i.e., a sparsity of

### Table VIII

| Scheme | # Param. | Sparsity | FLOPs ↓ | TOP-1 ↓ (TOP-1) | TOP-5 ↓ (TOP-5) |
|--------|----------|----------|---------|----------------|----------------|
| Li et al. [14] | 126.7M | 8.38% | 71% | 0.29% | -0.05% |
| Luo et al. [15] | 131.5M | 4.92% | 68% | -1.46% | -1.09% |
| Hu et al. [59] | 126.7M | 8.38% | 71% | -0.64% | -0.43% |
| Lin et al. [18] | 126.2M | 8.75% | 71% | -1.65% | -0.97% |
| Han et al. [7] | 10.3M | 92.5% | 79% | -0.26% (68.66%) | 0.44% (89.12%) |
| SCL | 60.2M | 56.5% | 50% | -1.11% (72.84%) | -0.54% (90.88%) |
| SCL | 36.9M | 73.3% | 71% | -0.33% (72.05%) | -0.25% (90.60%) |
| SCL | 13.9M | 89.9% | 87% | 2.49% (69.24%) | 1.11% (89.24%) |

### Table IX

| Scheme | # Param. | Sparsity | FLOPs ↓ | TOP-1 ↓ (TOP-1) | TOP-5 ↓ (TOP-5) |
|--------|----------|----------|---------|----------------|----------------|
| Li et al. [14] | 15.9M | 38% | 54% | 3.36% | 2.08% |
| Li et al. [14] | 12.2M | 52% | 59% | 4.31% | 2.42% |
| Luo et al. [15] | 16.8M | 34% | 41% | 3.09% | 1.63% |
| Luo et al. [15] | 12.2M | 52% | 59% | 4.12% | 2.28% |
| Wen et al. [25] | 13.2M | 48% | 49% | 4.58% | 2.68% |
| Hu et al. [59] | 15.9M | 38% | 54% | 3.47% | 2.39% |
| Hu et al. [59] | 12.2M | 52% | 59% | 4.25% | 2.41% |
| Lin et al. [18] | 15.5M | 39% | 54% | 2.83% | 1.57% |
| Lin et al. [18] | 12.0M | 53% | 59% | 3.65% | 2.11% |
| He et al. [33] | - | - | - | 0.19% | 8.39% |
| Huang et al. [37] | 15.6M | 39% | 43% | 4.30% | 2.07% |
| Xiao et al. [38] | - | - | - | 0.40% | - |
| Kang et al. [35] | - | - | - | 0.12% | 0.16% |
| SCL | 17.9M | 30% | 24% | -0.30% | -0.16% |
| SCL | 6.8M | 74% | 79% | 0.23% | 0.40% |
The results indicate that SCL can effectively output models that outperform the baseline in terms of perplexity on the test set. The 80% sparse RNN obtained by these existing state-of-the-art pruning methods and the baseline model that has a sparsity of 0%, sparse models are automatically found by SCL. Compared with Zhu and Gupta [11] and Narang [60], SCL achieves higher perplexity values. The lower the perplexity, the better the RNN model. Table XI lists the experimental results of existing pruning methods (i.e., Zhu and Gupta [11] and Narang [60]) and SCL. To obtain these results, the pruning method of Zhu and Gupta [11] has to find an appropriate combination of sparsity rates for each weight tensor through a lot of trials and errors, while the sparsity for each weight is automatically found by SCL. Compared with the baseline model that has a sparsity of 0%, sparse models obtained by these existing state-of-the-art pruning methods and SCL achieve higher perplexity values. The 80% sparse RNN model trained by Zhu and Gupta [11] or SCL reduces the number of weight parameters by almost 80% and meanwhile outperforms the baseline in terms of perplexity on the test dataset. The results indicate that SCL can effectively output excellent sparse connectivity for RNNs. Besides, when the expected sparsity is 95%, the perplexity results of SCL are better than those of Zhu and Gupta [11] and Narang et al. [60].

### F. Sparse RNN Learning on WikiText-2

When SCL prunes weight matrices in RNNs (i.e., setting some weight values to zero), it does not necessarily produce zero gradients. In addition, the network connectivity of RNNs is not sparsely initialized in SCL. During the training process, sparse connections are gradually produced by SCL. Hence, SCL does not cause the exploding/vanishing gradient problem for RNNs. In this work, SCL is applied to a word-level language model WikiText-2 [52] for verifying its effectiveness on RNNs. The perplexity of the language model is evaluated. In this work, SCL is applied to a word-level language model WikiText-2 [52] for verifying its effectiveness on RNNs. The perplexity of the language model is evaluated.

\[
h_t = \sigma(Wx_t + u \odot h_{t-1} + b)\]

(24)

where \(x_t \in \mathbb{R}^m\) and \(h_t \in \mathbb{R}^n\) are the input states and hidden states at a time step \(t\), respectively. \(W \in \mathbb{R}^{n \times m}\), \(U \in \mathbb{R}^{n \times n}\), and \(b \in \mathbb{R}^n\) represent the weight for the current input, recurrent input, and bias of neurons, respectively. \(u \in \mathbb{R}^n\) is a weight vector. When the weight matrix \(U\) happens to be a diagonal matrix, the vector \(u\) can be regarded as the diagonal vector of matrix \(U\). \(N\) represents the number of neurons in this layer and \(\sigma\) represents an activation function. We can see that \(U_{h_{t-1}}\) in the RNN is reformulated as Hadamard product \(u \odot h_{t-1}\) in the IndRNN. Therefore, the gradient of the \(n\)th neuron at the time step \(t\) is changed from the RNN gradient to the IndRNN gradient.

\[
\frac{\partial J}{\partial h_t} = \frac{\partial J}{\partial h_t} \prod_{k=0}^{T-1} \text{diag}(\sigma(h_{k+1})) U^T
\]

(26)

Next, let us analyze the impact of our SCL method on the exploding/vanishing gradient problem in IndRNN neural networks. For IndRNN networks, the proposed SCL method is applicable to prune weights in the matrix \(W\) in the IndRNN without affecting the exploding/vanishing gradient problem in training. Moreover, the weights in the vector \(u\) should be excluded from being pruned by SCL, because if some weights in \(u\) are pruned, some values of \(u_{n,k-1}^\prime\) will be set to zero, thus causing the vanishing gradient problem in the IndRNN. In fact, by comparing the dimension of weight matrix \(W \in \mathbb{R}^{n \times m}\) and weight vector \(u \in \mathbb{R}^n\), we see that the majority of weight parameters are located in the matrix \(W\). As a result, when SCL prunes the weight matrix \(U\) and meanwhile prevents the vector \(u\) from being pruned, the pruning ability and space in the IndRNN networks are not significantly reduced. Since we have shown the effectiveness of the proposed SCL on RNNs, both of whose weight matrices \(W\) and \(U\) are pruned by SCL, it is expected that when only pruning the weight matrix \(W\), and the proposed SCL method can achieve highly sparse IndRNN networks without affecting the exploding/vanishing gradient problem.

### VI. Conclusion

We present an SCL method to automatically explore and optimize sparse network connectivity. As the number of neural network connections is incorporated into the objective function, the network connectivity can be optimized for a given sparsity expectation to achieve the best performance. Our proposed SCL method has the task-aware ability, which does not require designer-defined pruning criteria or hyperparameters for each network layer. As a result, the SCL-induced...
sparse networks are explored in a larger hypothesis space, and they have the potential to generate optimized network connections. The proposed SCL is applicable to various neural network architectures, including fully connected networks, convolutional neural networks (VGGs, ResNets, DenseNets, and EfficientNets), and RNNs. Experiments on the MNIST, CIFAR-10, CIFAR-100, ImageNet, and WikiText-2 datasets demonstrate that the proposed SCL method achieves highly efficient learning of sparse network connectivity in network compression ratio, FLOP reduction, and accuracy over existing state-of-the-art-pruning methods in the literature. So far, SCL supports convolutions, fully connected layers, and RNN layers. We will explore SCL on other operators in the future.

ACKNOWLEDGMENT
The authors would like to thank Dr. Chun-Chen Liu from Kneron Inc., who provided insight and expertise that greatly assisted the research. They would also like to thank Cheng-Hung Hsieh and Jia-En Hsieh from National Chiao Tung University for their helpful discussion.

REFERENCES
[1] C. Szegedy et al., “Going deeper with convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 1–9.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105.
[3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. ICLR, 2015, pp. 1–14.
[4] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2016, pp. 770–778.
[5] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 4700–4708.
[6] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature, vol. 521, no. 7553, p. 436, 2015.
[7] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in Proc. Adv. Neural Inf. Process. Syst., 2015, pp. 1135–1143.
[8] Y. Guo, A. Yao, and Y. Chen, “Dynamic network surgery for efficient dnns,” in Proc. Adv. In Neural Inf. Process. Syst., 2016, pp. 1379–1387.
[9] K. Ullrich, E. Meeds, and M. Welling, “Soft weight-sharing for neural network compression,” in ICLR, 2018.
[10] D. Molchanov, A. Ashukha, and D. Vetrov, “Variational dropout sparsifies deep networks,” in Proc. ICML, 2017, pp. 1–10.
[11] M. Zhu and S. Gupta, “To prune, or not to prune: Exploring the efficacy for mobile vision applications,” 2018, arXiv:1704.04861.
[12] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 1492–1500.
[13] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, “Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers,” 2018, arXiv:1802.00124.
[14] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” in Proc. ICLR, 2017, pp. 1–17.
[15] L. Theis, L. Korshunova, A. Tejani, and F. Huszár, “Faster gaze prediction with dense networks and Fisher pruning,” 2018, arXiv:1801.05787.
[16] Y. He, Y. Ding, P. Liu, L. Zhu, H. Zhang, and Y. Yang, “Learning filter pruning criteria for deep convolutional neural networks acceleration,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 2099–2108.
[17] C. Louizos, M. Welling, and D. P. Kingma, “Learning sparse neural networks through L0 regularization,” in Proc. ICLR, 2018, pp. 1–13.
[18] M. Kang and B. Han, “Operation-aware soft channel pruning using differentiable masks,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 5122–5131.
[19] C. Hermann, R. S. Bowen, and R. Zabih, “Channel selection using gumbel softmax,” in Proc. Eur. Conf. Comput. Vis. Glasgow, U.K.: Springer, 2020, pp. 241–257.
[20] Z. Huang and N. Wang, “Data-driven sparse structure selection for deep neural networks,” in Proc. ECCV, 2018, pp. 304–320.
[21] X. Xiao, Z. Wang, and Z. Luan, “Neurouze: Automatic network pruning by regularizing auxiliary parameters,” in Proc. Adv. Neural Inf. Process. Syst., vol. 32, 2019, pp. 1–11.
[22] E. Jang, S. Gu, and B. Poole, “Categorical reparameterization with gumbel-softmax,” in Proc. ICLR, 2017, pp. 1–13.
[23] C. J. Maddison, A. Mnih, and Y. W. Teh, “The concrete distribution: A continuous relaxation of discrete random variables,” in Proc. ICLR, 2017, pp. 1–20.
[24] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” 2013, arXiv:1308.5432.
[25] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binary neural networks: Training deep neural networks with weights and activations constrained to +1 or −1,” 2016, arXiv:1602.02830.
[26] P. Yin, S. Zhang, J. Lu, S. Osher, Y. Qi, and J. Xin, “Blended coarse gradient descent for full quantization of deep neural networks,” Res. Math. Sci., vol. 6, no. 1, pp. 1–23, 2019.
[27] Z. Cai, X. He, J. Sun, and V. Vasconcelos, “Deep learning with low precision by half-wave Gaussian quantization,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 5918–5926.
[28] P. Yin, J. Lu, S. Zhang, S. Osher, Y. Qi, and J. Xin, “Understanding straightforward estimate in training activation quantized neural nets,” 2019, arXiv:1903.05662.
[29] G. Hinton, “Natural networks for machine learning coursea videos lectures,” 2012. [Online]. Available: https://www.youtube.com/playlist?list=PLoRl3Ht4JOcdU872GhiYWf6jwrk_SNhz9
[30] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 1026–1034.
[31] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” in Proc. ICLR, 2019, pp. 1–10.
Zhimin Tang received the B.S. degree from Hunan Normal University, Changsha, China, in 2012, and the M.S. degree from Hunan University, Changsha, in 2015. He is currently pursuing the Ph.D. degree with the Department of Automation, Xiamen University, Xiamen, China.

He was a Visiting Scholar with the Department of Electrical and Computer Engineering, Southern Illinois University Carbondale, Carbondale, IL, USA, from 2018 to 2020. His research interests include machine learning, pattern recognition, data mining, computer vision, biological information processing, and financial data analysis.

Bike Xie received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2005, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2006 and 2010, respectively.

He worked on a broad range of research topics including capacity regions and encoding schemes for broadcast channels, downlink-time regions for peer-to-peer networks, packet coding and exchange for broadcast networks, universal turbo codes for space-time channels, and channel code design for optical communications at UCLA. He joined Marvell Semiconductor Inc., Santa Clara, CA, USA, in 2010, where he led a system team designing physical layer systems and specifications for 6–112 Gbps SerDes systems and capacitive touchscreen systems. In 2017, he joined Kneron Inc., San Diego, CA, USA, where he is currently the Director of engineering. He is/has been working on system architecture design for AI chips, deep learning model compression, and algorithm design for deep learning applications.

Mr. Lu was a recipient of the Best Paper Award of the International Symposium on Low Power Electronics and Design in 2007, the Best Paper Award Nomination of the IEEE System-on-Chip Conference in 2016, and the top five teams in the International Hardware Design Contest of the IEEE Design Automation Conference in 2017.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.