Exploring Structural Sparsity of Deep Networks Via Inverse Scale Spaces

Yanwei Fu®, Chen Liu®, Donghao Li®, Zuyuan Zhong, Xinwei Sun, Jinshan Zeng®, and Yuan Yao®

Abstract—The great success of deep neural networks is built upon their over-parameterization, which smooths the optimization landscape without degrading the generalization ability. Despite the benefits of over-parameterization, a huge amount of parameters makes deep networks cumbersome in daily life applications. On the other hand, training neural networks without over-parameterization faces many practical problems, e.g., being trapped in the local optimal. Though techniques such as pruning and distillation are developed, they are expensive in fully training a dense network as backward selection methods; and there is still a void on systematically exploring forward selection methods for learning structural sparsity in deep networks. To fill in this gap, this paper proposes a new approach based on differential inclusions of inverse scale spaces. Specifically, our method can generate a family of models from simple to complex ones along the dynamics via coupling a pair of parameters, such that over-parameterized deep models and their structural sparsity can be explored simultaneously. This kind of differential inclusion scheme has a simple discretization, dubbed Deep structure splitting Linearized Bregman Iteration (DessiLIBI), whose global convergence in learning deep networks could be established under the Kurdyka-Łojasiewicz framework. Particularly, we explore several applications of DessiLIBI, including finding sparse structures of networks directly via the coupled structure parameter and growing networks from simple to complex ones progressively. Experimental evidence shows that our method achieves comparable and even better performance than the competitive optimizers in exploring the sparse structure of several widely used backbones on the benchmark datasets. Remarkably, with early stopping, our method unveils “winning tickets” in early epochs: the effective sparse network structures with comparable test accuracy to fully trained over-parameterized models, that are further transferable to similar alternative tasks. Furthermore, our method is able to grow networks efficiently with adaptive filter configurations, demonstrating the good performance with much less computational cost. Codes and models can be downloaded at https://github.com/DessiLIBI2020/DessiLIBI.

Index Terms—Early stopping, growing network, inverse scale space, linearized bregman iteration, lottery ticket hypothesis, network pruning, structural sparsity

1 INTRODUCTION

 Nowadays deep neural networks have shown great expressive power in many research areas such as image recognition [1], object detection [2], and point cloud estimation [3]. Such power is attributed to an avalanche of network parameters learned by supervision on large-scale datasets, i.e., model over-parameterization. Typically, the total number of parameters is orders of the magnitude higher than the number of training samples. And the over-parameterized neural networks can be trained with the loss functions by Stochastic Gradient Descent (SGD) [4] or modified optimization methods with adaptive stepsize, e.g., Adam [5], accompanied by early stopping.

The over-parameterization can benefit the training process of deep neural networks (DNN), and not necessarily result in a bad generalization or overfitting [6], especially when some weight size dependent complexities are controlled [7], [8], [9], [10]. Particularly, some recent empirical works show that model over-parameterization may help both optimization and generalization of networks, by simplifying the optimization landscape of empirical risks toward locating global optima [11], [12], [13], [14], and improving the generalization ability of deep neural networks for both discriminative [6] and generative models [15].

However, compressive networks are desired in many real world applications, e.g., robotics, self-driving cars, and augmented reality. For instance, the inference of large DNN models typically demands the support of GPUs, which are expensive for many real-world applications. Thus, it is essential to produce compressive networks. For this purpose, the classical way is to employ the norm-based regularization such as $L_1$ regularization [16] and enforce the
sparsity on weights toward the compact, and memory efficient networks. This type of methods, unfortunately, may cause the decline of expressive power as empirically validated in [17]. This is because that the weights learned in neural networks are highly correlated, and $L_1$ regularization on such weights violates the incoherence or irresponsible condition needed for sparse model selection [18], [19], [20], leading to spurious selections with poor generalization. On the other hand, the general type of regularization such as $L_2$ norm typically takes the function of low-pass filtering, sometimes in the form of weight decay [21] or early stopping [22], [23]. Sparsity has not been explicitly enforced on the models in this regularization, which may not produce a compressive model directly. Alternatively, Group Lasso [24] has also been utilized for finding sparse structures in DNN [25], and exerting good data locality with structured sparsity [25].

The difficulty of efficiently training a sparse network without over-parameterization results in the common practice in the community resorting to the backward selection, i.e., starting from training a big model using common task datasets like ImageNet, and then conduct the pruning [26], [27], [28], [29] or distilling [30] such big models to small ones without sacrificing too much of the performance. In particular, the recent Lottery Ticket Hypothesis (LTH) proposed in [31] made the following key empirical observation: dense, randomly-initialized, feed-forward networks contain small, sparse subnetworks, i.e., “winning tickets” structures, capable of being trained to comparable performance as the original network at a similar speed. To find such winning tickets, LTH works in backward selection, relying on the methods of one-shot or iterative pruning, which however, demands expensive computations and rewinding from initializations in [32], [33].

Is there any alternative approach to find effective subnetworks without fully training a dense network? In this paper, we pursue the methodology in a reverse order, forward selection, a sharp contrast to the backward selection methods above. Particularly, we design some dynamics that starts from simple yet interpretable models, delving into complex models progressively, and simultaneously exploits over-parameterized models and structural sparsity. Our forward selection method enables finding important structural sparsity even before fully training a dense, over-parameterized model, avoiding the expensive computations in backward selection.

To achieve this goal, the Inverse Scale Space (ISS) method in applied mathematics [34], [35] is introduced to training deep neural networks, for the first time up to our knowledge. The “inverse scale space” method, was first proposed in [36] with Total-Variation sparsity for image reconstruction. The name comes from the fact that the features in the inverse scale space shown early in small scales are coarse-grained shapes, while fine details appeared later, in a reverse order of wavelet scale space where coarse-grained features appear in large scale spaces. Recently the ISS was shown as sparse regularization paths with statistical model selection consistency in high dimensional linear regression [35] and generalized linear models [37]. Moreover, Huang et al. [38], [39] further improved this by relaxing model selection consistency conditions using variable splitting.

Our inverse scale space dynamics of training neural networks can be described as differential inclusions, where important network parameters are learned at a faster speed than unimportant ones. Specifically, original network parameters are lifted to a coupled pair, with one weight set $W$ of parameters following the standard gradient descend to explore the over-parameterized model space, while the other set of parameters $\Gamma$ learning structure sparsity in an inverse scale space. The two sets of parameters are coupled in an $L_2$ regularization. The ISS follows the gradient descent flow when the coupling regularization is weak, while reduces to a sparse mirror descent flow when the coupling is strong. During the training process, the parameters $\Gamma$ plays the role of exploring the sparse structure of the model parameters in inverse scale space, where important structures are learned faster than unimportant ones. Such differential inclusion dynamics enjoy a simple discretization even in a highly non-convex setting of training deep neural networks, where we call such a discretization as Deep structure splitting Linearized Bregman Iteration (DessiLBI). A proof is provided to guarantee the global convergence of DessiLBI under the Kurdyka-Lojasiewicz framework. DessiLBI is a natural extension of SGD with sparse structure exploration in an inverse scale space. Critically, DessiLBI finds the important structure faster than unimportant ones, which enables a totally new way of exploring and exploiting the compact structure in DNNs. This paper presents the applications of DessiLBI in network sparsification, finding winning tickets, and growing networks. Particularly, we address: (1) how to find sparse network structures directly from our augmented variables $\Gamma$ computed by DessiLBI, where in particular, DessiLBI can help find the winning tickets without the expensive rewinding; and (2) an effective way to grow networks from a simple seed network to complex ones.

Network Sparsification. DessiLBI may find sparse network structures that effective subnetworks can be rapidly learned via the structural sparsity parameter along the early iterative dynamics without fully training a dense network first. The support set of structural sparsity parameter learned in the early stage of this inverse scale space discloses important sparse subnetworks, including important weights, filters, and even layers. After obtaining the important sparse structures, both fine-tuning and retraining can be selected as post-processing. The priority of fine-tuning and retraining received wide discussions recently [40], [41], while we conduct extensive experiments to compare their performances. Our experimental results illustrate that the sparse structure found by DessiLBI is relatively robust to post-processing. Sparse structure found along the regularization path shows good performance on several widely-used network structures compared with their dense counterparts. In addition, training with DessiLBI does no harm to, or even enhances the performance of the dense model. As a result, the structural sparsity parameter may enable us to rapidly find sparse structure in early training epochs which saves plenty of training time and computational cost.

Finding Winning Tickets. DessiLBI also demonstrates new inspiring performance on finding a winning ticket structure in LTH [31]. We conduct several experiments to explore the performance of winning ticket subnetworks found by our
methods. These experiments show that our method can find a winning ticket subnetwork at an early stage, while having similar or even better generalization ability if compared against fully trained dense models. Besides, experiments also show that the winning tickets obtained by our method generalize across different natural image datasets, exhibiting transferability as studied in [33].

Growing Networks. DessiLBI can result in an elegant way to grow network dynamically. Here we propose to use regularization paths in inverse scale spaces to construct a lite growing method. In detail, we start with a small seed network with only a few filters for each layer. During the exploration of inverse scale spaces, important parameters are selected at an early stage. When the majority of the filters in one layer are selected, we assume that the complexity of this layer should be increased to enhance the model capacity, and more filters will be added to this layer. The early stopping property of DessiLBI will greatly reduce computational cost while maintain the model performance.

Contributions. We highlight the contributions in this paper. (1) The Inverse Scale Space method is, for the first time, applied to explore the structural sparsity of over-parameterized deep networks. DessiLBI can be interpreted as the discretization of solution paths of differential inclusion dynamics for the inverse scale spaces. (2) Global convergence of DessiLBI in such a nonconvex optimization is established based on the Kurdyka–Lojasiewicz framework, that the whole iterative sequence converges to a critical point of the empirical loss function from arbitrary initializations. (3) Stochastic variants of DessiLBI demonstrate comparable and even better performance than other training algorithms on ResNet-18 in large scale training such as ImageNet-2012, among other datasets, jointly exploring structural sparsity with interpretability. (4) Structural sparsity parameters in DessiLBI provide important information about subnetwork architecture with comparable or even better accuracies than dense models after retraining or fine-tuning – DessiLBI with early stopping can provide fast winning tickets without fully training dense, over-parameterized models. (5) By using DessiLBI, we present two elegant ways to explore compact model: selecting important structures in the original network and expanding a seed network to ones with sufficient capacity.

Extensions. We explain the extension from our conference paper [42]. (1) Fundamentally, despite the essential idea is still the same as [42], we equip DessiLBI with a new magnitude scaling update strategy, that significantly alleviates the imbalance of magnitude scales across different layers in DNNs, as empirically validated in our experiments. (2) By using the DessiLBI updated from our conference version, we further propose a series of ways to pruning the neural network including weight pruning, filter pruning and our novel layer pruning. (3) We further study the properties of early stopping and transferability of the winning tickets found by DessiLBI. (4) We propose an elegant way to dynamically grow a network via exploring the inverse scale space which needs much less training time and computational cost compared with other methods. (5) Extensive new experiments and ablation studies that are added in addition to our conference version, further reveal the insights and efficacy of our methods.

2 RELATED WORKS

Our DessiLBI is built upon the Linearized Bregman Iterations and has a tight relationship to the classical mirror descent algorithm and ADMM. Some other related topics of finding sparse networks are also discussed.

2.1 Mirror Descent Algorithm

Mirror Descent Algorithm (MDA). first proposed by [43] to solve constrained convex optimization \( L^*: = \min_{W} \mathcal{L}(W) \) \((K \text{ is convex and compact}), can be understood as a generalized projected gradient descent [44] with respect to Bregman distance \( B_{\Omega}(u_0, u_1) = \Omega(u_0) - \Omega(u_1) - \langle \nabla \Omega(u_1), u_0 - u_1 \rangle \) induced by a convex and differentiable function \( \Omega(\cdot) \),

\[
V_{t+1} = V_t - \alpha \nabla \mathcal{L}(W_t) \\
W_{t+1} = \nabla \Omega^* (V_{t+1}),
\]

where the conjugate function of \( \Omega(\cdot) \) is defined as \( \Omega^*(V) = \sup_{W} \langle W, V \rangle - \Omega(W) \).

At the \( k \) iteration, Eq. (1) uses two steps to optimize \( W_{k+1} = \arg \min_{W} \langle v, \alpha \nabla \mathcal{L}(W) \rangle + B_{\Omega}(v, W_0) \) [45] : Eq (1a) implements the gradient descent on \( V \) that is an element in dual space \( V_k = \nabla \Omega(W_k) \); and Eq (1b) projects it back to the primal space. As step size \( \alpha \to 0 \), MDA has the following limit dynamics as ordinary differential equation (ODE) [43]:

\[
\dot{V}_t = - \nabla \mathcal{L}(W_t), \\
\dot{W}_t = \nabla \Omega^* (V_t),
\]

where \( \dot{V}_t \) denotes the right derivative of \( V_t \) at the time \( t > 0 \).

Convergence analysis with rates for convex loss has been well studied. Researchers also extend the analysis to stochastic version [46], [47] and Nesterov acceleration scheme [48], [49]. In deep learning, we have to deal with highly non-convex loss, recent work [50] has established the convergence to global optima for overparameterized under two assumptions: (i) the initial point is close enough to the manifold of global optima; (ii) the \( \Omega(\cdot) \) is strongly convex and differentiable.

For non-differentiable \( \Omega \) such as the Elastic Net penalty in compressed sensing and high dimensional statistics, Equation (1) is studied as the Linearized Bregman Iteration (LBI) in applied mathematics [35], [51] that follows a discretized solution path of differential inclusions, to be discussed below. Such solution paths play a role of sparse regularization path where early stopped solutions are often better than the convergent ones when noise is present. In this paper, we investigate a varied form of LBI for the highly non-convex loss in deep learning models, exploiting the sparse paths, and establishing its convergence to a Karush–Kuhn–Tucker (KKT) point for general networks from arbitrary initializations. Furthermore, our method is a natural extension of SGD with sparse structure exploration. It reduces to the standard gradient descent method when the coupling regularization is weak, while reduces to a sparse mirror descent when the coupling is strong.
2.2 Linearized Bregman Iteration

Linearized Bregman Iteration (LBI), was proposed in [51], [52] that first studies Eq. (1) when $\Omega(W)$ involves $\ell_1$ or total variation non-differentiable penalties met in compressed sensing and image denoising. Beyond convergence for convex loss [51], [53], Osher et al. [35] and Huang et al. [37], particularly showed that LBI is a discretization of differential inclusion dynamics whose solutions generate iterative sparse regularization paths, and established the statistical model selection consistency for high-dimensional generalized linear models. Moreover, Huang et al. [38], [39] further improved this by proposing SplitLBI, incorporating into LBI a variable splitting strategy such that the restricted Hessian with respect to augmented variable ($\Gamma$ in Eq. (6)) is orthogonal. This can alleviate the multicollinearity problem when the features are highly correlated; and thus can relax the irrepresentable condition, i.e., the necessary condition for Lasso to have model selection consistency [19], [20], [54]. A variety of applications (e.g., [55], [56], [57], [58], [59], [60]) have been found for this algorithm since its inception.

However, existing work on SplitLBI is restricted to convex problems in generalized linear modes. It remains unknown whether the algorithm can exploit the structural sparsity in highly non-convex deep networks. To fill in this gap, in this paper, we propose the Deep structure splitting LBI that simultaneously explores the overparameterized networks and the structural sparsity of parameters in such networks, which enables us to generate an iterative regularization path of deep models whose important sparse architectures are unveiled in early stopping.

2.3 Alternating Direction Method of Multipliers

Alternating Direction Method of Multipliers (ADMM), which also adopted variable splitting strategy, breaks original complex loss into smaller pieces with each one can be easily solved iteratively [61], [62]. Recent works [63] established the convergence result of ADMM in convex, stochastic and non-convex setting, respectively. Wang et al. [64] studied convergence analysis with respect to Bregman distance. Training neural networks by ADMM has been studied in [65], [66]. Recently, Wang et al. [67] established the convergence of ADMM in a very general nonconvex setting, and Franca et al. [68] derived the limit ODE dynamics of ADMM for convergent analysis.

However, one should distinguish the LBI dynamics from ADMM that LBI should be viewed as a discretization of differential inclusion of inverse scale space that generalizes a sparse regularization solution path from simple to complex models where early stopping helps find important sparse models; in contrast, the ADMM, as an optimization algorithm for a given objective function, focuses on the convergent property of the iterations.

2.4 Early Stopping Regularization of Gradient Method

To optimize the not differentiable target function, subgradient is typically utilized as the generalization of the gradients in classical optimization textbooks [69], [70].

The early stopping is commonly used as a regularization technique to avoid overfitting. It has been studied in mathematics as a regularization method in inverse problems [71]. In statistical machine learning, early stopping has been studied in Boosting as gradient descent method, e.g., $L_2$-Boost [72] and Boosting in classification [73], [74]. In particular, $L_2$-Boost is generalized by [22] to gradient descent learning of regression functions in Reproducing Kernel Hilbert Spaces (RKHS) with random designs, showing that early stopping is a polynomial regularization better than Tikhonov or Ridge regularization for avoiding the saturation issue of the latter. A generalization of such kernel boosting algorithms to convex losses is given in [23], [75] using localized Rademacher complexities. Nonetheless, these works do not take sparsity constraints into consideration as high dimensional statistics and our work here. To handle sparse linear regressions, [76], establishes the the inverse scale space approach using differential inclusions, showing the model selection consistency for discovering causal variables by early stopping under the Irreprentable or Incoherence condition equivalent to that of LASSO [20], [54]. Such results are later extended by [77] to general convex losses including logistic regression and various graphical models. For general structural sparsity where parameters are sparse under a linear transformation, [39], [78] proposes Split LBI and its limit differential inclusions, establishing their model selection consistency under weaker conditions than the incoherence condition of generalized LASSO. This last work lays down a foundation of current exploration of early stopping to find important structure in deep neural networks.

2.5 Finding Compact Networks

Pruning Networks. In real world applications, limited computational resource makes compact models more demanding. In the manner of backward selection, pruning [79] is one of the most direct ways of producing a light model. Network pruning can be roughly categorized as weight and filter pruning, by whether removing some structural parameters such as convolutional fillers. For unstructural weight pruning, Han et al. [26] drops small weights of a well-trained dense network. The filter pruning methods take into consideration the network sparsity, memory footprint, and computational cost. Several works [27], [80] attempt to train a network first and then prune the network according to specific metrics such as $L_2$ norm of weights. Centripetal-SGD [81] groups the weight by their initialization and forces the weights inner one group to have the same values. LEGR [82] studies the scale variation across layers and suggests that using genetic algorithm can find decent affine transformation, making $L_2$ normalization more reasonable. In contrast to these backward selection methods that are expensive in both memory and computational cost, forward selection methods have been widely used in traditional statistical machine learning. Examples of such forward selection or parsimonious learning include boosting as functional gradient descent [83], [84] or coordinate descent method to solve LASSO [16], regularization paths associated with $L_2$ (Tikhonov regularization or Ridge regression) [85] and/or $L_1$ (Lasso) penalties [86], etc. None of these methods has been systematically studied in deep learning. In [40], the authors investigated a greedy forward
selection method to sparsify filters. In contrast, we investigate the inverse scale space method to network sparsification and our DessiLBI can select structural sparsity in weight level, filter level, and even layer level with improved efficiency.

Lottery Ticket Hypothesis (LTH). LTH [31] states that one can find effective subnets in a well-trained dense model, i.e., winning tickets. When training a winning ticket subnet in isolation from the initialization of the dense network, its test accuracy can match the dense model test accuracy with at most the same number of iterations. By finding a winning ticket using backward-selection-based algorithms, one can obtain an extremely sparse network with comparable or even better generalization ability.

Frankle et al. [32] studied the LTH in the even deeper neural networks by retraining, which means retrain from a very early stage of training (0.1% to 7%). LTH also appeared in reinforcement learning and natural language processing [87], [88], [89]. Additionally, winning ticket networks for filter pruning are also studied in several works [41], [42]. The transferability is studied in [33] that the winning tickets can be transferred across datasets as well as optimizers. Recently it found that utilizing gradient information can unveil a winning ticket in a randomly initialized network without modifying the weight values [90]. Furthermore, Malach et al. [91] proved that there exists a subnet with similar accuracy in a sufficiently over-parameterized neural network with bounded random weights. More theoretical analysis appeared in [92], [93], [94].

Although LTH has been widely studied, generating winning tickets could be computationally expensive. The one shot pruning and iterative pruning proposed in [31] are the de facto algorithms in winning ticket generation. One shot pruning utilizes a backward selection that first trains a network and prunes it according to weight magnitudes. Iterative pruning improves one shot pruning by conducting one shot pruning multiple times with a smaller pruning rate for each time which may cost unbearable GPU hours when finding extremely sparse winning tickets.

In this paper, our novel way to get winning tickets is built upon the forward selection, in the iterative procedure of DessiLBI. Critically, we show that by using early stopping, our DessiLBI can successfully discover the winning ticket structure without fully training a dense network. Empirically, we also show that our winning ticket structure has the nice property of transferability to the new dataset in the same domain of natural images.

Searching Network Structure. Plenty of recent efforts are made on searching for a good sparse network, rather than pruning. For example, Neural Architecture Search (NAS) [95], [96], [97], [98], [99] aims to search better network architectures automatically. The early works [95], [100] of NAS use reinforcement learning or evolutionary algorithms to search the whole network architectures, which need huge amount of computation cost. For example, Zoph et al. [95] spend more than 2000 GPU days in searching a network architecture on CIFAR10 dataset. Many works try to reduce the searching cost. Instead of searching the whole network architectures, Zoph et al. [96] and Zhong et al. [97] search cells and stack the searched cells into the complete networks, which greatly reduces the search space. However, these improved methods still need nearly 100 GPU days [97]. There are other strategies to improve the efficiency of the searching process. ENAS [101] and one-shot NAS [102], [103] share the parameters between child models, which significantly accelerates the searching process. DARTS [98] proposes gradient based methods that just use one or several GPU days for searching. Different from these NAS works, this paper presents a lite network growing method, benefiting from the forward selection in training networks by DessiLBI. In our method, we jointly grow the network structures, and train the network parameters by using our DessiLBI. It thus saves significant computational cost, while still maintains reasonably good performance.

3 INVERSE SCALE SPACES AND DESSI-LBI

Supervised learning aims to learn a mapping

$$\Phi_W : \mathcal{X} \rightarrow \mathcal{Y}$$

from input space $\mathcal{X}$ to output space $\mathcal{Y}$, with a parameter $W$ such as weights in neural networks, by minimizing certain empirical loss function on training samples

$$\widehat{L}_n(W) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \Phi_W(x_i)).$$

For example, a neural network of $l$-layer is defined as

$$\Phi_W(x) = \sigma_l(W^l) \sigma_{l-1}(W^{l-1}) \cdots \sigma_1(W^1)x)$$

where $W = \{W^l\}_{l=1}^L$ and $\sigma_l$ is the nonlinear activation function of the $l$th layer.

3.1 Differential Inclusion of Inverse Scale Space

Differential Inclusion of Inverse Scale Space. Consider the following dynamics adapted to neural network training,

$$\dot{W}_t = -\nabla W \widehat{L}(W_t, \Gamma_t)$$

and

$$\dot{V}_t = -\nabla \Gamma \widehat{L}(W_t, \Gamma_t)$$

where $W_t$ and $V_t$ are the right derivatives of $W_t$ and $V_t$, individually. The $V$ is a sub-gradient of

$$\Omega(\Gamma) := \Omega_x(\Gamma) + \frac{1}{2\kappa} \|\Gamma\|^2$$

where for some sparsity-enforced, often non-differentiable regularization, we have $\Omega_x(\Gamma) = \lambda \Omega(\Gamma)$ ($\lambda \in \mathbb{R}_+$) such as Lasso or group Lasso penalties for $\Omega_x(\Gamma)$; $\kappa > 0$ is a damping parameter such that the solution path is continuous, and the augmented loss function is

$$\widetilde{L}(W, \Gamma) = \widehat{L}_n(W) + \frac{1}{2\nu} \|W - \Gamma\|^2_2,$$
Differential inclusion system Eq. (6) is a coupling of gradient descent on $W$ with non-convex loss and mirror descent (LBI) of $\Gamma$ Eq. (2) with non-differentiable sparse penalty. It may explore dense over-parameterized models $W_t$ in the proximity of structural parameter $\Gamma_t$ with gradient descent, while $\Gamma_t$ records important sparse model structures. Specifically, the solution path of $\Gamma_t$ exhibits the following property in the separation of scales: starting at the zero, important parameters of large scale will be learned fast, popping up to be nonzeros early, while unimportant parameters of small scale will be learned slowly, appearing to be nonzeros late. In fact, Eq. (7) takes the $\Omega_1(\Gamma) = ||\Gamma||_1$ and $\kappa \to \infty$ for simplicity, $V_t$ as the subgradient of $\Omega_t$, undergoes a gradient descent flow before reaching the $\ell_\infty$-unit box, which implies that $\Gamma_t = 0$ in this stage. The earlier a component in $V_t$ reaches the $\ell_\infty$-unit box, the earlier a corresponding component in $\Gamma_t$ becomes nonzero and rapidly evolves toward a critical point of $\mathcal{L}$ under gradient flow. On the other hand, the $W_t$ follows the gradient descent with a standard $\ell_2$-regularization. Therefore, $W_t$ closely follows the dynamics of $\Gamma_t$ whose important parameters are selected.

Compared with directly enforcing a penalty function such as $\ell_1$ or $\ell_2$ regularization

$$\min_W \hat{R}_\lambda(W) := \mathcal{L}_n(W) + \Omega_\lambda(W), \quad \lambda \in \mathbb{R}_+.$$  

Eq. (6) can relax the irrepresentable conditions for model selection by Lasso [38], which can be violated for highly correlated weight parameters. The weight $W$, instead of directly being imposed with $\ell_1$-sparsity, adopts $\ell_2$-regularization in the proximity of the sparse path of $\Gamma$ that admits simultaneously exploring highly correlated parameters in over-parameterized models and sparse regularization.

The key insight lies in that differential inclusion of Eq. (6c) drives the important features in $\Gamma_t$ that earlier reaches the $\ell_\infty$-unit box to be selected earlier. Hence, the importance of features is related to the “time scale” of dynamic hitting time to the $\ell_\infty$ unit box, and such a time scale is inversely proportional to lasso regularization parameter $\lambda = 1/\kappa$ [35]. Such a differential inclusion is first studied in [34] with Total-Variation (TV) sparsity reconstruction, where important features in early dynamics are coarse-grained shapes with fine details appeared later. This is in contrast to wavelet scale space that coarse-grained features appear in large scale spaces, thus named “inverse scale space”. In this paper, we shall see that Eq. (6) inherits such an inverse scale space property empirically even for the highly nonconvex neural network training. Fig. 1 shows a LeNet trained on MNIST by the discretized dynamics, where important sparse filters are selected in early epochs while the popular SGD returns dense filters.

3.2 Deep Structure Splitting LBI

Deep Structure Splitting Linearized Bregman Iteration. Eq. (6) admits an extremely simple discrete approximation, using Euler forward discretization of dynamics and called DessiLBI in the sequel:

$$W_{k+1} = W_k - \kappa \alpha_k \cdot \nabla_W \mathcal{L}(W_k, \Gamma_k),$$  

$$V_{k+1} = V_k - \alpha_k \cdot \nabla_\Gamma \mathcal{L}(W_k, \Gamma_k),$$  

$$\Gamma_{k+1} = \kappa \cdot \text{Prox}_{\Omega_\lambda}(V_{k+1}),$$  

where $\alpha_k$ is the step size at the $k$ iteration; $V_0 = \Gamma_0 = 0, W_0$ can be small random numbers such as Gaussian initialization. Here we add interpretation for $\alpha$ and $\kappa$ in Appendix. F, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPAMI.2022.3168881. For some complex networks, it can be initialized as common setting. The proximal map in Eq. (10c) that controls the sparsity of $\Gamma$,

$$\text{Prox}_{\Omega_\lambda}(V) = \arg \min_{\Gamma} \left\{ \frac{1}{2} ||\Gamma - V||_2^2 + \Omega_\lambda(\Gamma) \right\}.$$  

Such an iterative procedure returns a sequence of sparse networks from simple to complex ones whose global convergence condition to be shown below, while solving Eq. (9)
at various levels of \( \lambda \) might not be tractable, especially for over-parameterized networks.

Structural Sparsity. Our DessiLBI explores structural sparsity in fully connected and convolutional layers, which can be unified in framework of group lasso penalty, \( \Omega_g(\Gamma) = \sum_g ||\Gamma^g||_2^2 \), where \( ||\Gamma^g||_2 = \sqrt{\sum_{i=1}^{\text{size}} (\Gamma^g_i)^2} \) and \( \text{size} \) is the number of weights in \( \Gamma^g \). Thus Eq. (10c) has a closed form solution \( \Gamma^g = \kappa \cdot \text{max}(0, 1 - 1/||V^g||_2) \). Typically,

a) For a convolutional layer, \( \Gamma^g = \Gamma^g(c_{in}, c_{out}, \text{size}) \) denote the convolutional filters where \( \text{size} \) denotes the kernel size and \( c_{in} \) and \( c_{out} \) denote the numbers of input channels and output channels, respectively. When we regard each group as each convolutional filter, \( g = c_{out} \); otherwise for weight sparsity, \( g \) can be every element in the filter that reduces to the Lasso.

b) For a fully connected layer, \( \Gamma = \Gamma(c_{in}, c_{out}) \) where \( c_{in} \) and \( c_{out} \) denote the numbers of inputs and outputs of the fully connected layer. Each group \( g \) corresponds to each element \((i, j)\), and the group Lasso penalty degenerates to the Lasso penalty.

4 Global Convergence of DessiLBI

We present a theorem that guarantees the global convergence of DessiLBI, i.e., from any initialization, the DessiLBI sequence converges to a critical point of \( \mathcal{L} \). Our treatment extends the block coordinate descent (BCD) studied in [105], with a crucial difference being the mirror descent involved in DessiLBI. Instead of the splitting loss in BCD, a new Lyapunov function is developed here to meet the Kurdyka-Łojasiewicz property [106], [107] studied the convergence of variable splitting method for single hidden layer networks with Gaussian inputs.

Let \( P := (W, \Gamma) \). Following [37], the DessiLBI algorithm in Eq. (10a)–(10c) can be rewritten as the following standard Linearized Bregman Iteration,

\[
P_{k+1} = \arg\min_P \{ \langle P - P_k, \alpha \nabla \mathcal{L}(P_k) \rangle + B^\Psi(P, P_k) \}
\]

where

\[
\Psi(P) = \Omega_\lambda(\Gamma) + \frac{1}{2\kappa} ||P||_2^2
\]

\[
= \Omega_\lambda(\Gamma) + \frac{1}{2\kappa} ||W||_2^2 + \frac{1}{2\kappa} ||\Gamma||_2^2,
\]

(13)

\( p_k \in \partial \Psi(P_k) \), and \( B^\Psi \) is the Bregman divergence associated with convex function \( \Psi \), defined by

\[
B^\Psi(P, Q) := \Psi(P) - \Psi(Q) - \langle q, P - Q \rangle.
\]

for some \( q \in \partial \Psi(Q) \). Without the loss of generality, consider \( \lambda = 1 \) in the sequel. One can establish the global convergence of DessiLBI under the following assumptions.

Assumption 1. Suppose that:

a) \( \mathcal{L}_n(W) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, \Phi(W(x_i))) \) is continuous differentiable and \( \nabla \mathcal{L}_n \) is Lipschitz continuous with a positive constant \( Lip \);

b) \( \mathcal{L}_n(W) \) has bounded level sets;

c) \( \mathcal{L}_n(W) \) is lower bounded (without loss of generality, we assume that the lower bound is 0);

d) \( \Omega \) is a proper lower semi-continuous convex function and has locally bounded subgradients, that is, for every compact set \( S \subseteq \mathbb{R}^n \), there exists a constant \( C > 0 \) such that for all \( \Gamma \in S \) and all \( g \in \partial \Omega(\Gamma) \), there holds \( ||g|| \leq C \);

e) the Lyapunov function

\[
F(P, \tilde{g}) := \alpha \mathcal{L}(W, \Gamma) + B^\Omega(\Gamma, \tilde{g}).
\]

(15)

is a Kurdyka-Łojasiewicz function [105], [108] on any bounded set, where \( B^\Omega(\Gamma, \tilde{g}) := \Omega(\Gamma) - \Omega(\tilde{g}) - (\tilde{g}, \Gamma - \tilde{g}) \), \( \tilde{g} \in \partial \Omega^*(\tilde{g}) \), and \( \Omega^* \) is the conjugate of \( \Omega \) defined as

\[
\Omega^*(\tilde{g}) := \sup_{u \in \mathbb{R}^n} \{ \langle u, \tilde{g} \rangle - \Omega(U) \}.
\]

Remark 1. Assumption 1(a)-(c) are regular in the analysis of nonconvex algorithm (see, [108] for instance), while Assumption 1(d) is also mild including all Lipschitz continuous convex function over a compact set. Some typical examples satisfying Assumption 1(d) are the \( \ell_1 \) norm, group \( \ell_1 \) norm, and every continuously differentiable penalties. By Eq. (15) and the definition of conjugate, the Lyapunov function \( F \) can be rewritten as follows,

\[
F(W, \Gamma, g) := \alpha \mathcal{L}(W, \Gamma) + \Omega(\Gamma) + \Omega^*(g) - \langle \Gamma, g \rangle.
\]

(16)

Now we are ready to present the main theorem.

Theorem 1. [Global Convergence of DessiLBI] Suppose that Assumption 1 holds. Let \( \{W_k, \Gamma_k\} \) be the sequence generated by DessiLBI (Eqs. (10a)–(10c)) with a finite initialization. If \( 0 < \alpha_k = \mu < \frac{2}{\kappa(Lip + \nu^{-1})} \), then \( \{W_k, \Gamma_k\} \) converges to a critical point of \( \mathcal{L} \) defined in Eq. (8), and \( \{W^k\} \) converges to a critical point of \( \mathcal{L}_n(W) \).

Applying to the neural networks, typical examples are summarized in the following corollary.

Corollary 1. Let \( \{W_k, \Gamma_k, g_k\} \) be a sequence generated by DessiLBI (26a–26c) for neural network training where \( (a) \ell \) is any smooth definable loss function [109], such as the square loss \( t^2 \), exponential loss \( e^t \), logistic loss \( \log(1 + e^{-t}) \), and cross-entropy loss; \( (b) \sigma_t \) is any smooth definable activation [109], such as linear activation \( t \), sigmoid \( \frac{1}{1 + e^{-t}} \), hyperbolic tangent \( \tanh(t) \), and softplus \( \frac{1}{\ln(1 + e^t)} \) for some \( c > 0 \) as a smooth approximation of ReLU; \( (c) \Omega \) is the group Lasso. Then the sequence \( \{W_k\} \) converges to a stationary point of \( \mathcal{L}_n(W) \) under the conditions of Theorem 1.

5 Implementation of DessiLBI

The DessiLBI is a natural extension of SGD with exploring sparse structures of network. As referring to the variants of SGD, we further introduce several variants of our DessiLBI, by learning with data batches, using momentum to
accelerate the convergence, adding weight decay on the parameters \( W \), as well as the magnitude scaling updates.

**Batch DessiLBI.** To train networks on large datasets, stochastic approximation of the gradients in DessiLBI over the mini-batch \((X, Y)_{\text{batch}}\) is adopted to update the parameter \( W \),

\[
\nabla_W W = \nabla_W \mathcal{L}(W) |_{X, Y_{\text{batch}}},
\]

**DessiLBI with Momentum (Mom).** Inspired by the variants of SGD, the momentum term can be also incorporated to the standard DessiLBI that leads to the following updates of \( W \) by replacing Eq (10a) with,

\[
\begin{align*}
 v_{t+1} & = \tau v_t + \nabla_W \mathcal{L}(W_t, \Gamma_t) \\
 W_{t+1} & = W_t - \kappa v_{t+1}
\end{align*}
\]

where \( \tau \) is the momentum factor, empirically setting as 0.9.

**DessiLBI with Momentum and Weight Decay (Mom-Wd).** The update formulation for Eq. (10a) is

\[
\begin{align*}
 v_{t+1} & = \tau v_t + \nabla_W \mathcal{L}(W_t, \Gamma_t) \\
 W_{t+1} & = W_t - \kappa v_{t+1} - \beta W_t
\end{align*}
\]

**Magnitude Scaling Update Strategy.** The imbalance of weight scales across different layers during training may degrade the performance of network sparsification and winning ticket generation. We visualize the distribution of \( V \) in Figs. 6 and 5. The magnitude scale varies significantly across layers. It adds to the difficulties of finding good structure. On the other hand, for ReLU-based neural networks, the parameter magnitude scales of different layers do not negatively affect the network training, as the positive homogeneity of ReLU [110] guarantees that the scale has no difference to the class prediction. To this end, we introduce a magnitude scaling update strategy to improve our DessiLBI, such that the magnitude scales of \( V \) are comparable across different layers. The detailed updating rule is to replace Eqs. (10b), (10c) by the following:

\[
\begin{align*}
 V_{t+1} & = V_t - \alpha_t \beta_k \circ \nabla_{V_t} \mathcal{L}(W_t, \Gamma_t) \\
 \Gamma_{k+1} & = \kappa \epsilon_k \odot \text{Prox}_{\Omega_k}(V_{t+1}),
\end{align*}
\]

where \( \beta_k = [\beta_{k,i}] \), and \( \epsilon_k \) are scaling factor tensors. For the \( i \)th layer, scaling factors are defined by

\[
\beta_{k,i} = \min \left( 1, \frac{1}{\|W_{k,i}\|} \right) \cdot \left( 1 - \frac{\#(\text{supp}(\Gamma_{k,i}))}{\#(\text{supp}(W_{k,i}))} \right),
\]

and \( \epsilon_k = \||W_{k,i}\|_1 \|. \) Here, we introduce the notation of \( W_{k,i}^\supset \) in Eq (21) to denote the \( j \)th weight/filter of \( i \)th iteration and \( i \)th layer; for the \( j \)th weight or filter in the \( i \)th layer, \( \|W_{k,i}^\supset\|_1 = \|W_{k,i}^\supset\|_1 \) or \( \|W_{k,i}^\supset\|_2 = \|W_{k,i}^\supset\|_2 \), respectively; \# denotes the number of elements; and \( \text{supp} \) is the support of a set. Note that for \( j \)th filter of \( i \)th layer, we will expand the size of \( \beta_{k,i} \) to match the size of the filter. To avoid \( \beta_{k,i} = 0 \), we will set a minimum value for it.

For the parameter \( V \), its gradient is determined by the magnitude of the residue between corresponding parameter \( W \) and \( \Gamma \). Here, we normalize the gradient for \( V \) by the magnitude of \( W \), so that the gradient contains more direction information. For instance, one filter may be more important if the updating direction altered in lower frequency. To avoid the gradient explosion, we set the maximum of \( \beta_k \) to be 1. In addition, we use the ratio between numbers of selected parameters and numbers of whole parameters in one layer as the penalty.

This idea is enlightened by Lipschitz renormalization/ scaling for ReLu-activation networks [110], [111] which is of positive homogeneity, i.e., invariant under a multiplication of a positive constant on the weight matrix such that its norm is renormalized to be close to a unit ball. These works attempt to avoid driving the model weights to infinity via the renormalization. Different from their motivation, we aim to alleviate the scale imbalance problem in neural networks when selecting structure. As mentioned in [110], [112], relu-based neural networks own the property called positive homogeneity or rescaling invariance. It means that we can modify the weights of relu-based neural networks in an appropriate way that the prediction of the model is not changed. So the scale of magnitude for different layers can vary significantly and this property sets some obstacles for selecting important structures for the scale imbalance. In our work, we attempt to normalize the gradient to \( V \) so that the update relies more on the direction instead of magnitude.

6 APPLICATIONS OF DESSILBI

6.1 Network Sparsification
In the process of training networks by DessiLBI, we are able to produce the sparse network directly. This enables us to explore the over-parameterization and structural sparsity simultaneously. We take the coupled parameters \( \Gamma \) to record the structural sparsity information along the regularization paths. The support set of \( \Gamma \) is utilized to conduct the pruning. The key principle is that it will be taken as less important parameters if the convolutional filters have the corresponding coupled parameter \( \Gamma = 0 \) in the inverse scale space. We have three levels for pruning parameters in network, described below. After getting the compact one, some post-processing steps such as fine-tuning or retraining are used to improve the performance of a pruned network. In [41] the authors point out that the performance of pruned network relies on the structure heavily; and retraining them from scratch can get even better results.¹ In our work, we will validate this in the experiment.

**Pruning Weights.** We define \( M' \) as the mask for each parameter of the \( i \)th layer by using support set \( \Gamma' \),

\[
M' = |\text{sign}(\Gamma')| = 1_{\text{supp}(\Gamma')}
\]

We directly use the mask to remove some weight parameters by Hadamard (elementwise) product as \( W' = M' \odot W \). This will enable us to get a sparse network. Typically, such pruning is commonly adopted to prune the weights of fully connected layers.

**Pruning Filters.** We can remove the convolutional filters by

\[
M'^g = |\text{sign}(\Gamma'^g)| = 1_{\text{supp}(\Gamma'^g)}
\]

¹. Potentially this is an arguable point, which is challenged by [40]
where \( \text{supp}(\Gamma^g) \) is the support set of filters \( \Gamma^g \) on the \( i \)th layer. Accordingly, the filters are changed as \( \tilde{W} = M^{\text{supp}} \odot W \). Generally, the pruning is conducted on the convolutional layers to remove the filters.

**Pruning Layers.** Benefiting from the magnitude scaling update of DessiLBI, the weight magnitudes of different layers learned by our DessiLBI are comparable or balanced now. This enables us to select the sparse structure at a higher level of whole layers than the level of individual filters or weights. Specifically if \( \text{supp}(\Gamma^g) \) or \( \text{supp}(\Gamma^i) \) is the empty set for the \( i \)th layer, we remove this layer totally from the network. Note that we will utilize the necessary transition layers, including pooling and concatenation operations, to make the consistent feature dimension after removing layers. Heavy redundancy of deep neural networks results in that several layers sometimes can be removed completely.

### 6.2 Finding Winning Tickets

As shown in LTH [31], finding winning tickets as subnets in a fully trained over-parameterized network is quite computationally expensive, especially using iterative pruning to unveil extremely sparse networks. In contrast, our DessiLBI empowers a much easier way to generate winning ticket structure. Here a forward selection method is implemented through exploring the Inverse Scale Space. By using the augmented variables of DessiLBI, we can obtain the winning ticket structure of a network in a more efficient way. Particularly, we search for the subnetwork in the inverse scale space computed by DessiLBI: the \( \Gamma \) is utilized as the metric for evaluating the importance of weights. That is, the mask \( M^i \) of each \( i \)th layer, derived from the support set of \( \Gamma^i \) is utilized to define the winning ticket structure, where the winning ticket can be obtained without fully training the network but using early stopping of DessiLBI. In fact, only a few epochs are required to get the winning ticket via DessiLBI in our experiments below. Fine-tuning or retraining after receiving the winning ticket structure exhibit comparable or better performance than the fully trained networks in a similar speed. Hence in a forward selection manner, our DessiLBI offers a more efficient and elegant way to get the winning ticket.

Transferability of winning tickets found in traditional ways has been observed in the empirical experiments in [33]. Here the transferability of winning tickets found by DessiLBI is further studied. In our experiment, for each target dataset, we find winning ticket structures by DessiLBI on different source datasets and show their performance close to the winning tickets generated on the same target dataset.

In a summary, we highlight the two key points in our winning ticket generation.

- **Early stopping.** We can define the winning ticket structures by exploiting the structural sparsity parameter computed in the early stage during the training process by DessiLBI, known as the early stopping regularization.
- **Transferability of winning ticket structure.** The winning ticket structure discovered by early stopping of DessiLBI can generalize across a variety of datasets in the natural image domain.

### 6.3 Growing Networks by DessiLBI

By exploiting the inverse scale space by DessiLBI, the structure of our network can be dynamically altered during the training procedure. We present a novel network growing process. Specifically, we start with a simple initialized network and gradually increase its capacity by adding parameters along with the training process by DessiLBI. Starting from very few filters of each convolutional layer, our growing method requires not only efficiently optimizing the parameters of filters, but also adding more filters if the existing filters do not have enough capacity to model the distribution of training data. The key idea is to measure whether the network at the current training iteration has enough capacity.

To this end, we monitor in training, the difference between support set of model parameters \( \text{supp}(W^i) \) and support set \( \text{supp}(\Gamma^i) \) of its augmented parameter \( \Gamma^i \) on the \( i \)th layer of network,

\[
\text{ratio} = \frac{\#(\text{supp}(\Gamma^i))}{\#(\text{supp}(W^i))}
\]  

where \( \text{supp} \) is the support set and \( \# \) denotes the number of elements in the set. When the ratio becomes close to 1, it means most of the filters in \( W^i \) are important and they are selected into \( \Gamma^i \), and thus we should expand the network capacity by adding new filters to the corresponding layer. In our network growing process, we start from a small number of filters (e.g. 4) of convolutional layers, more and more filters tend to be with non-zero values as the training iterates. Every \( J \) epochs, we can compute the ratio of Eq. (24) and consider the following two scenarios.

- **a)** If this ratio is larger than a pre-set threshold (denote as \( \tau \)), we add some new filters to existing filters \( W \) randomly initialized as [113], and \( \Gamma \) will add corresponding dimensions, initialized as zeros;
- **b)** Otherwise, we will not grow any filter in this epoch.

Then we continue DessiLBI to learn all the weights from training data. This process is repeated until the loss does not change much, or the maximum number of epoch is reached. After stopping the growing process, the network model is trained for extra few epochs (typically 30) with a smaller learning rate to finish the training process.

### 7 Experiments

**Dataset.** In this section, four widely used datasets are selected: CIFAR10, CIFAR100, SVHN, and ImageNet. For CIFAR10, it owns 10 categories and each category has 5000 and 1000 images for training and testing respectively. CIFAR100 has 100 classes containing 500 training images and 100 testing images each. SVHN contains real-world number images from 0 to 9. In SVHN, there are 73257 images for training and 26032 images for testing. For CIFAR10, CIFAR100, and SVHN, the images have a spatial size of \( 32 \times 32 \). For ImageNet-2012, there are 1.28 million training images and 50 k validation images. These images are sampled from 1000 classes.
This section introduces some stochastic variants of DessiLBI, followed by four sets of experiments revealing the key insights and properties of DessiLBI in exploring the structural sparsity of deep networks.

Implementation. Experiments are conducted over various backbones, e.g., LeNet, AlexNet, VGG, and ResNet. For MNIST and CIFAR10, the default hyper-parameters of DessiLBI are $\kappa = 1$, $\nu = 10$ and $\alpha_k$ is set as 0.1, decreased by $1/10$ every 30 epochs. In ImageNet-2012, the DessiLBI utilizes $\kappa = 1$, $\nu = 1000$, and $\alpha_k$ is initially set as 0.1, decays $1/10$ every 30 epochs. We set $\lambda = 1$ in Eq. (11) by default, unless otherwise specified. On MNIST and CIFAR10, we have batch size as 128; and for all methods, the batch size of ImageNet 2012 is 256. The standard data augmentation implemented in pytorch is applied to CIFAR10 and ImageNet-2012, as [114]. The weights of all models are initialized as [113]. In the experiments, we define the sparsity as percentage of non-zero parameters, i.e., the number of non-zero weights dividing the total number of weights in consideration. Runnable codes can be downloaded.\(^2\)

7.1 Image Classification Experiments

Settings. By referring the settings in [114], we conduct the experiments on ImageNet-2012. We compare variants of DessiLBI with that of SGD and Adam in the experiments. By default, the learning rate of competitors is set as 0.1 for SGD and its variant and 0.001 for Adam and its variants, and gradually decreased by $1/10$ every 30 epochs. (1) Naive SGD: the standard SGD with batch input. (2) SGD with $l_1$ penalty (Lasso). The $l_1$ norm is applied to penalize the weights of SGD by encouraging the sparsity of the learned model, with the regularization parameter of the $l_1$ penalty term being set as $1e^{-3}$. (3) SGD with moment (Mom): we utilize momentum 0.9 in SGD. (4) SGD with momentum and weight decay (Mom-Wd): we set the momentum 0.9 and the standard $l_2$ weight decay with the coefficient weight $1e^{-4}$. (5) SGD with Nesterov (Nesterov): the SGD uses nesterov momentum 0.9. (6) Naive Adam: it refers to standard Adam\(^3\).

As a natural extension of SGD with exploring sparse structure, DessiLBI can serve as a network training method. To validate this point, we training different backbones by DessiLBI on the large-scale dataset – ImageNet 2012. The results of image classification are shown in Table 1. Our DessiLBI variants may achieve comparable or even better performance than SGD variants in 100 epochs, indicating the efficacy in learning dense, over-parameterized models. It verifies that our DessiLBI can explore the structural sparsity while keeping the performance of the dense model.

Learning Sparse Filters for Interpretation. In DessiLBI, the structural sparsity parameter $\Gamma$, explores important sub-network architectures that contribute significantly to the loss or error reduction in early training stages. Through the $\ell_2$-coupling, structural sparsity parameter $\Gamma$ may guide the weight parameter to explore those sparse models in favour of improved interpretability. Fig. 1 visualizes some sparse filters learned by DessiLBI of LeNet-5 trained on MNIST (with $\kappa = 10$ and weight decay every 40 epochs), in comparison with dense filters learned by SGD. The activation patterns of improved interpretability. Fig. 1 visualizes some sparse filters learned by DessiLBI of LeNet-5 trained on MNIST (with $\kappa = 10$ and weight decay every 40 epochs), in comparison with dense filters learned by SGD. The activation pattern of such sparse filters favour high order global correlations between pixels of input images. Further visualization of convolutional filters learned by ImageNet is shown in Fig. 8 in Appendix, available in the online supplemental material, demonstrating the texture bias in ImageNet training [115].

7.2 Experiments on Network Sparsification

In this section, we conduct experiments on CIFAR10 dataset. Our DessiLBI is utilized for network sparsification. Empirically, with the augmented $\Gamma$, we can find sparse structure in the inverse scale space. In Appendix, Figs. 6 and 5, available in the online supplemental material, illustrate the distribution of filter norm for $W$ and $V$. It is clear that the magnitude scale of the augmented variable $V$ is balanced and is decoupled with arbitrary value of $W$.

To validate the efficacy of the structure, we use several experiments. By exploiting the magnitude scaling update strategy in computing $\Gamma$ for convolutional filters, and connections in convolutional layers and fully connected layers respectively, we introduce the filter and weight pruning in our experiments. For filter pruning, we aim to prune redundant convolutional filters, while the target of weight pruning is to prune connections in the network.

Common Setting. We follow the setting of LeGr [82], training the network for 200 epochs and fine-tuning the network for 400 epochs. For the training process, the learning rate is decayed by $5\times$ every 60 epochs with an initial learning rate 0.1. In post-processing, we compare fine-tuning and retraining. For fine-tuning, the learning rate is decayed by $5\times$ every 120 epochs. The initial learning rate for fine-tuning is 0.01. For retraining, the setting is the same as training. For all the processes, the weight decay is set as $5e^{-4}$.

Pruning Weights Setting. We conduct experiments on CIFAR10 and select two widely used network structure VGGNet [119] and ResNet [114]. Several methods such Iterative-Pruning-A [116] are selected for comparison. The

\[\text{Table 1: Top-1/Top-5 accuracy(%) on ImageNet-2012} \]

| Dataset | Models | Variants | AlexNet | ResNet-18 |
|---------|--------|----------|---------|-----------|
| ImageNet-2012 | Naive | -/- | 60.76/79.18 |
| SGD | $l_1$ | 46.49/65.45 | 51.49/72.45 |
| | Mom | 55.14/78.09 | 66.98/86.97 |
| | Mom-Wd | 56.55/79.09 | 69.76/89.18 |
| | Nesterov | -/- | 70.19/89.30 |
| DessiLBI | Naive | -/- | 59.66/83.28 |
| | Mom | 55.06/77.69 | 65.26/86.57 |
| | Mom-Wd | 56.23/78.48 | 68.55/87.85 |
| | 57.09/79.86 | 70.55/89.56 |

All models are run in 100 epochs. *: results from the official pytorch website. we use the official pytorch codes to run the competitors. more results on MNIST/CIFAR10, please refer Table 6 in supplementary, available online.

2. https://github.com/DessiLBI2020/DessiLBI
3. In Table 6 of Appendix, available in the online supplemental material, we further give more results for Adabound, Adagrad, Amsgrad, and Radam, which, we found, are hard to be trained on ImageNet-2012 in practice.
training setting is the same as filter pruning. Here, the threshold proximal mapping is altered: 0.002 for ResNet50 and 0.0006 for VGGNet16.

Results. For ResNet50, it is relatively compact, so the extreme sparsity leads to a relative decrease in accuracy. However, our results are often better than our competitors. For VGGNet16, our method can find subnets with sparsity 0.06 and the accuracy is still comparable or even improved. Here fine-tuning is often better than retraining, often with an increase in terms of accuracy. The results are shown in Table 2.

Results for fine-tuning with only 10 epochs are also present, which shows that the pruned network can get decent performance with just a few epochs of fine-tuning. The keep ratio for each layer is visualized in Fig. 9 in Appendix, available in the online supplemental material. For VGGNet16, most of the weight in the middle of the network can be pruned without hurting the performance. For the input conv layer and output fc layer, a high percent of weights are kept and the total number of weights for them is also smaller than other layers. For ResNet50, we can find an interesting phenomenon, that most layers inside the block can be pruned to a very sparse level. Meanwhile, for the input and output of a block, a relatively high percentage of weights should be kept.

Pruning Filters. Setting: We conduct experiments of filter pruning, and select several methods for comparison. For ResNet56, we pick up LEGR [82], HRank [118], AMC [120] and SFP [27] for comparison. And for VGGNet16, we choose L1 [121], HRank [118], Variational [122] and Hinge [123]. Here we set \( n = 500 \) and \( k = 1 \). The batch size is 128.

Results. The results are shown in Table 3. To control the whole sparsity, we set an upper bound for every layer which is 60% for both VGGNet16 and ResNet56. For both of the two networks, we can find that our pruning method achieves good results. For ResNet56, our model can reduce the FLOPs to 62% without hurting the performance. For fine-tuning setting with full (pre-)training, the pruned model can even increase the accuracy by about 0.5. Our early stopping experiments show that our method indeed

### TABLE 2

The Performance of the Sparse Structure Under Weight Pruning Setting Found Via Our Method

| Method                  | ResNet50                     | VGG16                      |
|-------------------------|------------------------------|----------------------------|
|                         | Acc  | Sparse Acc | Sparsity | Acc  | Sparse Acc | Sparsity |
| Iterative-Pruning-A [116]| 92.67| 93.08      | 0.40     | Iterative-Pruning-A [116]| 93.64| 93.58      | 0.40     |
| Iterative-Pruning-B [117]| 92.67| 85.16      | 0.10     | Iterative-Pruning-A [116]| 93.64| 91.88      | 0.10     |
| Iterative-Pruning-B [117]| 93.02| 92.14      | 0.40     | Iterative-Pruning-B [117]| 93.77| 93.74      | 0.40     |
| Iterative-Pruning-B [117]| 93.02| 73.75      | 0.10     | Iterative-Pruning-B [117]| 93.77| 91.09      | 0.10     |

DessiLBI

| FT 60 | 94.13 | 91.70 | 0.09 |
| RT 60 | 94.13 | 90.68 | 0.09 |
| FT 120| 94.13 | 92.64 | 0.11 |
| FT 120| 94.13 | 91.35 | 0.11 |
| FT 200| 94.13 | 92.54 | 0.11 |
| FT* 200| 94.13 | 90.03 | 0.11 |
| RT 200| 94.13 | 91.48 | 0.11 |

DessiLBI

| FT 60 | 94.13 | 91.70 | 0.09 |
| RT 60 | 94.13 | 90.68 | 0.09 |
| FT 120| 94.13 | 92.64 | 0.11 |
| FT 120| 94.13 | 91.35 | 0.11 |
| FT 200| 94.13 | 92.54 | 0.11 |
| FT* 200| 94.13 | 90.03 | 0.11 |
| RT 200| 94.13 | 91.48 | 0.11 |

Here \( \text{FT} \) and \( \text{RT} \) \( n \) refer to fine-tuning and retraining using the weights and structure after training with \( n \) epoch, respectively. In particular, \( \text{FT}^* \) 200 means training for 200 epochs and using fine-tune for only 10 epochs. The definition of sparsity is the number of non-zero parameters divided by the number of whole parameters. Sparse Acc denotes the accuracy of pruned model.

### TABLE 3

The Performance of the Sparse Structure Under Filter Pruning Setting Found Via Our Method

| Method                  | ResNet56                     | VGG16                      |
|-------------------------|------------------------------|----------------------------|
|                         | Acc  | Sparse Acc | MFLOP COUNT | Acc  | Sparse Acc | MFLOP COUNT |
| LEGR [82]               | 93.90| 93.70      | 59 (47%)    | Hinge [123]| 93.25| 92.91      | 191 (61%)   |
| AMC [120]               | 92.80| 91.90      | 63 (50%)    | Variational [122]| 93.25| 93.18      | 190 (61%)   |
| SFP [27]                | 93.60| 93.40      | 59 (47%)    | L1 [121]| 93.25| 93.11      | 200 (64%)   |
| HRank [118]             | 93.26| 93.17      | 63 (50%)    | HRank [118]| 93.96| 93.43      | 146 (54%)   |
| FT 60                  | 93.46| 93.30      | 77 (61%)    | FT 60 | 93.43| 93.00      | 91 (29%)    |
| RT 60                  | 93.46| 93.31      | 77 (61%)    | RT 60 | 93.43| 93.01      | 91 (29%)    |
| FT 120                 | 93.46| 93.59      | 78 (62%)    | FT 120| 93.43| 93.21      | 97 (31%)    |
| FT 200                 | 93.46| 93.16      | 78 (62%)    | FT 200| 93.43| 92.92      | 97 (31%)    |
| FT* 200                | 93.46| 89.97      | 78 (62%)    | FT* 200| 93.43| 89.76      | 100 (32%)   |
| RT 200                 | 93.46| 93.15      | 78 (62%)    | RT 200| 93.43| 93.01      | 100 (32%)   |
| Layer Selection         | 93.73| 93.47      | 56 (45%)    | Layer Selection | 93.47| 94.06      | 106 (34%)   |

DessiLBI

Here \( \text{FT} \) and \( \text{RT} \) \( n \) refer to fine-tuning and retraining using the weights and structure after training with \( n \) epoch, respectively. In particular, \( \text{FT}^* \) 200 means training for 200 epochs and using fine-tune for only 10 epochs. Layer selection contains experiments of pruning layers. MFLOP COUNT indicates the total number of floating-point operations executed in millions. The percentage means the ratio between the current MFLOP count and the MFLOP count of the full model. Sparse Acc denotes the accuracy of pruned model.
finds good sparse structure at the early stage. By viewing the comparison between fine-tuning and retraining, we can find that for structure found by our method, fine-tuning slightly outperforms retraining. And both of the post-processing steps show a very decent performance, i.e., our method is robust to the post-processing. Similar results can be observed on VGGNet16 in Table 3.

Pruning Layers. Setting: To push ahead the performance of our network sparsification, we alter the hyperparameters to get a further sparsification by using pruning layers as in Section 6.1. Here we use $k = 1$. We select $\nu = 1200$ for VGGNet16 and $\nu = 1500$ for ResNet56.

Results. The results are shown in Table 3. For ResNet56, our method reduces the FLOP count to about 45%, while the performance is only dropped by 0.26%. The detailed structure is shown in Fig. 2. Interestingly, the sparsified VGG16 network actually has improved performance over the original VGG16 significantly. Note that although the Flop counts of VGGNet16 are similar for pruning filters and pruning layers, the network of pruning layers is much sparser than that of filter pruning, with 10.95% of the parameters remained, in contrast to the sparsity for pruning filters at about 42.82%. For VGG16, most of the filters close to the input layer are selected by our network sparsification and much of the pruning occurs near the output layer. By viewing the structure of VGGNet, it is clear that redundancy exists in the layers close to the output layer. It is in accordance with our results. For ResNet56, we drop two layers in the middle of the corresponding blocks. The whole structure shows a dense selection in the beginning and end of channel alternating stage and a sparse selection inside each stage. A more detailed table is shown in Table 7 in Appendix, available in the online supplemental material.

7.3 Experiments on Winning Ticket

In this section, with early stopping, $\Gamma_t$ may learn effective subnetworks (i.e., “winning tickets” in [31]) in the inverse scale space. After retraining, these subnetworks achieve comparable or even better performance than the dense network. Our method is comparable to existing pruning strategies with an improved efficiency.

Settings. We find winning tickets on CIFAR10 dataset and two network structures VGGNet16 and ResNet50. When DessiLBI is used for finding winning tickets, we set $\mu = 500, \lambda = 0.05$ for ResNet50 and $\mu = 200, \lambda = 0.03$ for VGG16 respectively. $k = 1, \alpha = 0.1$ and training epoch $T = 160$ is used in both case. Then, we retrain the winning tickets using SGD with 160 epochs from the end of the second epoch in the first stage. The initial learning rate for SGD is 0.1 and decayed by 10 at the 80 and 120 epoch. To make a fair comparison, in one shot pruning and iterative pruning, we also use SGD with this setting for training and retraining.

Winning Tickets in Early Stopping. The results are shown in Fig. 3. DessiLBI gives a sequence of winning tickets with increasing sparsity. All the winning tickets found by our method achieve similar or even better performance than the winning tickets found by one shot pruning and iterative pruning. All the winning tickets outperform the two baseline methods, especially in extremely sparse cases. It is worth noting that DessiLBI with early stopping at a very early epoch (for example, the end of second epoch) is computationally effective and gives extremely sparse winning tickets. Comparing with one shot pruning, we can avoid complete training in the first stage and save up to training 158 epochs. Since iterative pruning consists of 10 iterative pruning, We can save more computation by using early stopped DessiLBI followed by retraining once. The high efficiency of DessiLBI is due to exploring subnets via inverse scale space in which, important subnets come out first.

Transferability of Winning Tickets. We further explore the generalization of winning tickets found by DessiLBI cross different datasets like [33]. The results are shown in Fig. 4. Specifically, we search winning tickets on source datasets and retrain with target datasets.

Similar to the observations in [33], our winning tickets discovered by DessiLBI in early stopping are transferable in
We first try to verify our method with a plain \( L_1 \)-prune and model learned by our growing method on CIFAR10 dataset. \( L_1 \)-P is the \( L_1 \) pruning

| VGG16 | Output size | Model | \( L_1 \)-P [121] | Ours |
|-------|-------------|-------|-----------------|------|
| Conv_1 | 32×32 | Maps | Maps | Maps |
| Conv_2 | 32×32 | 64 | 32 | 32 |
| Conv_3 | 16×16 | 128 | 128 | 126 |
| Conv_4 | 16×16 | 128 | 128 | 220 |
| Conv_5 | 8×8 | 256 | 256 | 242 |
| Conv_6 | 8×8 | 256 | 256 | 200 |
| Conv_7 | 8×8 | 256 | 256 | 182 |
| Conv_8 | 4×4 | 512 | 512 | 261 |
| Conv_9 | 4×4 | 512 | 512 | 151 |
| Conv_10 | 4×4 | 512 | 512 | 105 |
| Conv_11 | 2×2 | 512 | 512 | 88 |
| Conv_12 | 2×2 | 512 | 512 | 88 |
| Conv_13 | 2×2 | 512 | 512 | 80 |
| FC1 | - | 512 | 512 | 512 |
| FC2 | - | 10 | 10 | 10 |
| Params | - | 15 M | 5.40 M | 2.96 M |
| FLOPs | - | 313 M | 206 M | 217 M |
| Acc. | - | 93.25% | 93.40% | 93.80% |

The fully connected layer is altered according to [121].

The sense that they generalize across a variety of natural image datasets, such as CIFAR10, CIFAR100, and SVHN, often achieving performance close to the winning tickets generated on the same dataset. Moreover, winning tickets of DessiLBI may significantly outperform random tickets in very sparse cases. This shows that our algorithm can find transferable winning tickets.

We try to give some tentative explanations about the transferability. Despite that different datasets are used here, they all come from the same domain, i.e., natural images. In that sense, the images and models on these datasets shall share some common features; and winning tickets found on different datasets may encode the shared inductive bias and benefit the performance of sparse networks trained on another dataset.

7.4 Experiments on Network Growing

In this section, we grow a network to explore compact network structures via Inverse Scale Space. In detail, our method attempts to find a better filter configuration for each layer during training. To verify its efficacy, we conduct several experiments with CIFAR10 dataset.

Settings. We first try to verify our method with a plain network structure which means we do not use some special structure such as residue connection [114]. Here we utilize VGG16 [119] as the backbone. The backbone is initialized with eight filters in each convolutional layer and there are two fully connected layers following the setting of [121]. The hyper-parameters of DessiLBI here are \( \kappa = 1, \nu = 100 \) and the learning rate is set as 0.01. And the batch size is 128. We set the hyper-parameters of our method as \( J = 10, \tau = 0.8 \). After finishing growing structure, we decrease the learning rate by 10× and continue training for 30 epochs.

Results. Table 4 illustrates the filter configuration of models found by \( L_1 \)-prune [121] and our method. We can observe that among the 13 convolutional layers, \( L_1 \)-prune only reduces filters in the first convolution layer and the last 6 convolution layers by half, with the total number of final parameters being 5.40 M. By comparison, our method obtains a more compact model with only 2.96 M parameters.

A crucial observation is that our growing method prefers to filters of large feature spatial size, which helps our model with a better performance. Such results suggest that our growing method may learn better model structures by learning better filter configurations. Specifically, pruning has the limitation that it only tries to reduce the number of filters in each layer based on the originally designed number, while our method gets rid of this limitation and adapts to better configurations.

Note that our growing method is different from network architecture search (NAS) [95], [98], [99]. NAS searches the overall structure of the new network in the predefined space and spends much more time and computational cost on this searching process. However, our method is much more light and can find good network structure during training. Besides, our model adds little extra memory cost and is more device-friendly.

To further validate our method, AutoGrow [124] is picked for comparison. ResNet [114] is selected as the backbone. For AutoGrow, the results are from the original paper. For our method, we use ResNet56 with initial channel number 8. The results are in Table 5. It is clear that the networks grown by our method are more compact while having a better performance than AutoGrow. For more sufficient comparison, we try to compare our approach with several recent method, such as NASH [125], Splitting [126], Energy-aware Splitting [127], Firefly growing [128]. Please refer to the Appendix, available in the online supplemental material. E for more experimental comparison on our growing algorithm.

8 CONCLUSION

In this paper, a parsimonious deep learning method is proposed based on the dynamics of inverse scale spaces. Its simple discretization – DessiLBI has a provable global convergence on DNNs and thus is employed to train deep networks. Our DessiLBI can explore the structural sparsity without sacrificing the performance of the dense, over-parameterized models, in favour of interpretability. It helps identify effective sparse structure at the early stage which is verified with several experiments on network sparsification and finding winning ticket subnets which are transferable across different natural image datasets. Particularly, our method can select the structure in the level of layers which is not well studied in the existing literature. Furthermore, the proposed method can be applied to grow a network by simultaneously learning both network structure and parameters.
D. Erhan, Y. Bengio, A. Courville, and P. Vincent, “Visualizing higher-layer features of a deep network,” Univ. Montreal, Tech. Rep. 3, vol. 1341, p. 1, 2009.

J. Zheng, T. T.-K. Lau, S.-B. Lin, and Y. Yao, “Global convergence of block coordinate descent in deep learning,” in Proc. 36th Int. Conf. Mach. Learn., 2019, pp. 733–741.

S. Lojasiewicz, “Une propriete topologique des sous-ensembles analytiques reels,” in “Les Equations Aux Derrives Partielles,” Editions du centre National de la Recherche Scientifique, Paris, vol. 117, pp. 87–89, 1963.

F. Xue and J. Xin, “Convergence of a relaxed variable splitting method for learning sparse neural networks via $\ell_1$, $\ell_0$, and transformed-$\ell_1$ penalties,” 2018, arXiv:1807.03719v2.

J. Attouch, J. Bolte, and B. F. Svaiter, “Convergence of descent methods for semi-algebraic and tame problems: Proximal algorithms, forward-backward splitting, and regularized Gauss-Seidel methods,” Math. Program., vol. 137, pp. 91–129, 2013.

R. T. Rockafellar and J.-B. Wets, “Variational analysis,” Grundlehren Math. Wiss, Berlin, Germany: Springer, 1998.

W. Zhu, Y. Haung, and Y. Yao, “Rethinking breiman’s dilemma in neural networks: Phase transitions of margin dynamics,” Front. Appl. Math. Statist., vol. 45, 2020.

Q. Liao, B. Miranda, A. Banburski, J. Hidary, and T. Poggio, “A surprising linear relationship predicts test performance in deep networks,” 2018, arXiv:1807.09659.

B. Neyshabur, R. Salakhutdinov, and N. Srivastava, “Path-SGD: Path-normalized optimization in deep neural networks,” 2018, arXiv:1805.12639.

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., 2015, pp. 1026–1034.

K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

R. Goult, P. Reutenauer, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, “Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness,” in Proc. Int. Conf. Learn. Representations, 2019.

S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” in Proc. Int. Conf. Learn. Representations, 2015.

M. Zhu and S. Gupta, “To prune, or not to prune: Exploring the efficacy of pruning for model compression,” 2017, arXiv:1710.01878.

M. Lin et al., “HRank: Filter pruning using high-rank feature map,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 1529–1538.

K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in Proc. Int. Conf. Learn. Representations, 2015.

Y. He et al., “AMC: Automl for model compression and acceleration on mobile devices,” in Proc. Eur. Conf. Comput. Vis., 2018, pp. 784–800.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” in Proc. Int. Conf. Learn. Representations, 2017.

C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, “Variational convolutional neural network pruning,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2019, pp. 2780–2789.

Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, “Group sparsity: The hinge between filter pruning and decomposition for network compression,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., 2020, pp. 8018–8027.

W. Wen, F. Yan, Y. Chen, and H. Li, “Autogrow: Automatic layer growing in deep convolutional networks,” in Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2020, pp. 833–841.

T. Elsken, J.-H. Metzen, and F. Hutter, “Simple and efficient architecture search for convolutional neural networks,” 2017, arXiv:1711.04538.

L. Wu, D. Wang, and Q. Liu, “Splitting steepest descent for growing neural architectures,” Adv. Neural Inf. Process. Syst., vol. 32, pp. 656–666, 2019.

D. Wang, M. Li, L. Wu, V. Chandra, and Q. Liu, “Energy-aware neural architecture optimization with fast splitting steepest descent,” 2019, arXiv:1910.03103.

L. Wu, B. Liu, P. Stone, and Q. Liu, “Firefly neural architecture descent: A general approach for growing neural networks,” 2021, arXiv:2102.08574.
Zuyuan Zhong received the BS degree in statistics from Fudan University, Shanghai, China, in 2019. He is currently working toward the master degree with the School of Data Science, Fudan University, Shanghai, China. His current research interests include network pruning, adversarial example, etc.

Xinwei Sun received the PhD degree from the School of Mathematical Sciences, Peking University, in 2018. He is a tenure-track Associate Professor with Fudan University. His research interests mainly focus on statistical machine learning, causal learning, with their applications on medical imaging and few-shot learning.

Jinshan Zeng received the PhD degree in mathematics from Xi’an Jiaotong University, Xi’an, China, in 2015. He is currently a distinguished professor with the School of Computer and Information Engineering, Jiangxi Normal University, Nanchang, China, and serves as the director of the Department of Data Science and Big Data. He has published more than 40 papers in high-impact journals and conferences such as the IEEE Transactions on Pattern Analysis and Machine Intelligence, Journal of Machine Learning Research, IEEE Transactions on Signal Processing, IEEE Transactions on Geoscience and Remote Sensing, ICML etc. He has had two papers co-authored with collaborators that received the International Consortium of Chinese Mathematicians (ICCM) Best Paper Award (2018, 2020). His research interests include nonconvex optimization, machine learning, and remote sensing.

Yuan Yao received the BSE and MSE degrees in control engineering from the Harbin Institute of Technology, China, in 1996 and 1998, respectively, the MPhil degree in mathematics from the City University of Hong Kong, in 2002, and the PhD degree in mathematics from the University of California, Berkeley, in 2006. Since then, he has been with Stanford University and in 2009, he joined the Department of Probability and Statistics, School of Mathematical Sciences, Peking University, Beijing, China. He is currently a professor of mathematics, chemical and biological engineering, and by courtesy, Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong SAR, China. His current research interests include topological and geometric methods for high dimensional data analysis and statistical machine learning, with applications in computational biology, computer vision, and information retrieval.