Proceedings of Machine Learning Research 95:614-629, 2018 ACML 2018

Optimization Algorithm Inspired Deep Neural Network Structure Design

Huan Li LIHUANSS@PKU.EDU.CN
Key Lab. of Machine Perception (MOE), School of EECS, Peking University, Beijing, China
Yibo Yang IBO@PKU.EDU.CN
Key Lab. of Machine Perception (MOE), School of EECS, Peking University, Beijing, China
Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, China
Dongmin Chen DONGMINCHEN@PKU.EDU.CN
Academy for Advanced Interdisciplinary Studies, Peking University, Beijing, China
Zhouchen Lin[*] ZLIN@PKU.EDU.CN
Key Lab. of Machine Perception (MOE), School of EECS, Peking University, Beijing, China

Editors: Jun Zhu and Ichiro Takeuchi

Abstract

Deep neural networks have been one of the dominant machine learning approaches in recent years. Several new network structures are proposed and have better performance than the traditional feedforward neural network structure. Representative ones include the skip connection structure in ResNet and the dense connection structure in DenseNet. However, it still lacks a unified guidance for the neural network structure design. In this paper, we propose the hypothesis that the neural network structure design can be inspired by optimization algorithms and a faster optimization algorithm may lead to a better neural network structure. Specifically, we prove that the propagation in the feedforward neural network with the same linear transformation in different layers is equivalent to minimizing some function using the gradient descent algorithm. Based on this observation, we replace the gradient descent algorithm with the heavy ball algorithm and Nesterov's accelerated gradient descent algorithm, which are faster and inspire us to design new and better network structures. ResNet and DenseNet can be considered as two special cases of our framework. Numerical experiments on CIFAR-10, CIFAR-100 and ImageNet verify the advantage of our optimization algorithm inspired structures over ResNet and DenseNet.

Keywords: Deep neural network structure design, Optimization algorithms inspiration, Heavy ball algorithm, Nesterov's accelerated gradient descent algorithm, ResNet, DenseNet

1. Introduction

Deep neural networks have become a powerful tool in machine learning and have achieved remarkable success in many computer vision and image processing tasks, including classification (Krzhevsky et al., 2012), semantic segmentation (Long et al., 2015) and object detection (Girshick, 2015; Ren et al., 2016). After the breakthrough result in the ImageNet classification challenge (Krzhevsky et al., 2012), different kinds of neural network architectures have been proposed and the performance is improved year by year. GoogLeNet (Szegedy et al., 2015) seems to achieve the bottleneck of performance in 2014 with the traditional

[*] H. Li and Y. Yang have equal contributions and Z. Lin is the corresponding author.

feedforward neural network structure, where the units are connected only with the ones in the following layer. After that, new neural network structures have been proposed. Examples include ResNet (He et al., 2015b), DenseNet (Huang et al., 2017) and CliqueNet (Yang et al., 2018), where skip connections and dense connections are adopted to ease the network training and further push the state of the arts.

Although the new structures lead to a significant improvement compared with the traditional feedforward structure, it seems to require profound understandings of practical neural networks and substantial trials of experiments to design effective neural network structures. Thus we believe that the design of neural network structure needs a unified guidance. This paper serves as a preliminary trial towards this goal.

1.1. Related Work

There has been extensive work on the neural network structure design. Generic algorithm (Schaffer et al., 1992; Lam et al., 2003) based approaches were proposed to find both architectures and weights in the early stage of neural network design. However, networks designed with the generic algorithm perform worse than the hand-crafted ones (Verbancsics and Harguess, 2013). Saxena and Verbeek (2016) proposed a "Fabric" to sidestep the CNN model architecture selection problem and it performs close to the hand-crafted networks. Domhan et al. (2015) used Bayesian optimization for network architecture selection and Bergstra et al. (2013) used a meta-modeling approach to choose the type of layers and hyper parameters. Kwok and Yeung (1997), Ma and Khorasani (2003) and Cortes et al. (2017) used the adaptive strategy that grows the network structure layer by layer from a small network based on some principles, e.g., Cortes et al. (2017) minimized some loss value to balance the model complexity and the empirical risk minimization. Baker et al. (2016) and Zoph and Le (2017) used the reinforcement learning to search the neural network architecture. All of these are basically heuristic search based approaches. They are difficult to produce effective neural networks if the computing power is insufficient or the search strategy is inefficient as the search space is huge. DNN structure designed via minimizing some loss values is only best for given data. It may not generalize to other datasets if no *regular* structure exists in the network. Although some recently proposed methods that utilize a recurrent neural network and reinforcement learning scheme also achieve impressive results (Zoph and Le, 2017; Zhong et al., 2018), they differ from us due to the lack of explicit guidance to indicate where the connections should appear.

1.2. Motivation

In this paper, we design the neural network structures based on the inspiration from optimization algorithms. Our idea is motivated by the recent work in the compressive sensing community. Traditional methods for compressive sensing solve a well-defined problem $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}-\mathbf{y}\|^2+\alpha\|\mathbf{x}\|_1$[1] and employ iterative algorithms to solve it, e.g., the ISTA algorithm (Beck and Teboulle, 2009) with iterations $\mathbf{x}_{k+1} = \mathrm{prox}_{\frac{\alpha}{L}\|\cdot\|_1}(\mathbf{x}_k - \frac{1}{L}\mathbf{A}^T(\mathbf{A}\mathbf{x}_k-\mathbf{y}))$, where $\mathrm{prox}_{\alpha\|\cdot\|_1}(\mathbf{x}) = \mathrm{argmin}_{\mathbf{z}}\frac{1}{2}\|\mathbf{z}-\mathbf{x}\|^2 + a\|\mathbf{z}\|_1$. The iterative algorithms often need many iterations to converge and thus suffer from high computational complexity. Gregor and LeCun

[1] We denote $\|\mathbf{x}\| = \sqrt{\sum_i \mathbf{x}_i^2}$ and $\|\mathbf{x}\|_1 = \sum_i |\mathbf{x}_i|$.

(2010), Xin et al. (2016), Kulkarni et al. (2016), Zhang and Ghanem (2017) and Yang et al. (2016) developed a series of neural network based methods for compressive sensing. Their main idea is to train a non-linear feedforward neural network with a fixed depth. At each layer, a linear transformation is applied to the input $\mathbf{x}_k$ and then a nonlinear transformation follows, which can be described as $\mathbf{x}_{k+1} = \Phi(\mathbf{W}_k\mathbf{x}_k)$. In the traditional optimization based compressive sensing, the linear transformation $\mathbf{A}$ is fixed. As a comparison, in the neural network based compressive sensing, $\mathbf{W}_k$ is learnable so that each layer has a different linear transformation matrix. The neural network based compressive sensing often needs much less computation compared with the optimization based ones.

Since ISTA is almost the most popular algorithm for compressive sensing, most of the existing neural network based methods (Gregor and LeCun, 2010; Xin et al., 2016; Kulkarni et al., 2016) are inspired by ISTA and thus have the feedforward structure. Zhang and Ghanem (2017) proposed a FISTA-net (Beck and Teboulle, 2009) by adding a skip connection to the feedforward structure. However, all these networks are for *image reconstruction*, based on the compressive sensing model. The design methodology of deep neural networks for *image recognition* tasks is still lacking.

1.3. Contributions

In this paper, we study the design of the neural network structures for *image recognition* tasks[2]. To make our network structure easy to generalize to other datasets, our methodology separates the structure design and weights search, i.e., we do not consider the optimal weights in the structure design stage. The optimal weights will be searched via training after the structure design. Our methodology is inspired by optimization algorithms. Specifically, our contributions include:

1. For the standard feedforward neural network that shares the same linear transformation and nonlinear activation function at different layers, we prove that the propagation in the neural network is equivalent to using the gradient descent algorithm to minimize some function $F(\mathbf{x})$. As a comparison, the neural network based compressed sensing only studied the soft thresholding as the nonlinear activation function and the goal of designing the network is to solve the compressive sensing problem as accurately as possible.

2. Based on the above observation, we propose the hypothesis that a faster optimization algorithm may inspire a better neural network structure. Especially, we give the neural network structures inspired by the heavy ball algorithm and Nesterov's accelerated gradient algorithm, which include ResNet and DenseNet as two special cases.

3. Numerical experiments on CIFAR-10, CIFAR-100 and ImageNet verify that the optimization algorithm inspired neural network structures outperform ResNet and DenseNet. These show that our methodology is very promising.

Our methodology is still preliminary. Although we have shown in some degree the connection between faster optimization algorithms and better deep neural networks, currently we haven't

[2] We only focus on the part before SoftMax as SoftMax will be connected to all networks in order to produce label information.

revealed the connection between optimization algorithm and DNN structure design in a theoretically rigorous way. It is an analogy. However, analogy does *not* mean unsolid. For example, DNN is inspired by brain. It is also an analogy and has no strict connections to brain either. However, no one can say that DNN is insignificant or ineffective.

2. Reviews of Some Optimization Algorithms

In this section, we review the gradient descent (GD) algorithm (Bertsekas, 1999), the heavy ball (HB) algorithm (Polyak, 1964), Nesterov's accelerated gradient descent (AGD) algorithm (Nesterov, 1983) and the Alternating Direction Method of Multipliers (ADMM) (Gabay, 1983; Lin et al., 2011) to solve the general optimization problem $\min_{\mathbf{z}} f(\mathbf{z})$.

The gradient descent algorithm is one of the most popular algorithms in practice. It consists of the following iteration[3]:

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \nabla f(\mathbf{z}_k). \tag{1}$$

The heavy ball algorithm is a variant of the gradient descent algorithm, where a momentum is added after the gradient descent step:

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \nabla f(\mathbf{z}_k) + \beta(\mathbf{z}_k - \mathbf{z}_{k-1}). \tag{2}$$

Nesterov's accelerated gradient algorithm has the similar idea with the heavy ball algorithm, but it uses the momentum in another way:

$$\begin{aligned} \mathbf{y}_k &= \mathbf{z}_k + \frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}(\mathbf{z}_k - \mathbf{z}_{k-1}), \\ \mathbf{z}_{k+1} &= \mathbf{y}_k - \nabla f(\mathbf{y}_k). \end{aligned} \tag{3}$$

where θ_k is computed via $\frac{1-\theta_k}{\theta_k^2} = \frac{1}{\theta_{k-1}^2}$ and $\theta_0 = 1$[4]. When $f(\mathbf{z})$ is μ-strongly convex[5] and its gradient is L-Lipschitz continuous[6], the heavy ball algorithm and Nesterov's accelerated gradient algorithm can find an ϵ-accuracy solution in $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$ iterations, while the gradient descent algorithm needs $O\left(\frac{L}{\mu}\log\frac{1}{\epsilon}\right)$ iterations. Iteration (3) has an equivalent form of

$$\mathbf{y}_{k+1} = \mathbf{y}_k - \sum_{j=0}^{k} h_{k+1,j}\nabla f(\mathbf{y}_j), \tag{4}$$

where

$$h_{k+1,j} = \begin{cases} \frac{\theta_{k+1}(1-\theta_k)}{\theta_k}h_{k,j}, & j = 0,\cdots,k-2, \\ \frac{\theta_{k+1}(1-\theta_k)}{\theta_k}(h_{k,k-1}-1), & j = k-1, \\ 1 + \frac{\theta_{k+1}(1-\theta_k)}{\theta_k}, & j = k. \end{cases} \tag{5}$$

[3] For direct use in our network design, we fix the stepsize to 1. It can be obtained by scaling the objective function $f(\mathbf{z})$ such that the Lipschitz constant of $\nabla f(\mathbf{z})$ is 1.

[4] When $f(\mathbf{z})$ is μ-strongly convex and its gradient is L-Lipschitz continuous, $\frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}$ is fixed at $\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$.

[5] I.e., $f(\mathbf{y}) \geq f(\mathbf{x}) + \langle\nabla f(\mathbf{x}), \mathbf{y}-\mathbf{x}\rangle + \frac{\mu}{2}\|\mathbf{y}-\mathbf{x}\|^2$.

[6] I.e., $\|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\| \leq L\|\mathbf{y}-\mathbf{x}\|$.

ADMM and its linearized version can also be used to minimize $f(\mathbf{z})$ by reformulating it as $\min_{\mathbf{y},\mathbf{z}} f(\mathbf{z}) + f(\mathbf{y}), \quad s.t. \quad \mathbf{y} - \mathbf{z} = 0$. Linearized ADMM consists the following steps[7]:

$$\begin{aligned}
&\mathbf{z}_{k+1} = \operatorname{argmin}_{\mathbf{z}} \langle \nabla f(\mathbf{z}_k), \mathbf{z}\rangle + \frac{1}{2}\|\mathbf{z} - \mathbf{z}_k\|^2 + \langle \lambda_k, \mathbf{z}\rangle + \frac{1}{2}\|\mathbf{z} - \mathbf{y}_k\|^2,\\
&\mathbf{y}_{k+1} = \operatorname{argmin}_{\mathbf{y}} \langle \nabla f(\mathbf{y}_k), \mathbf{y}\rangle + \frac{1}{2}\|\mathbf{y} - \mathbf{y}_k\|^2 - \langle \lambda_k, \mathbf{y}\rangle + \frac{1}{2}\|\mathbf{z}_{k+1} - \mathbf{y}\|^2,\\
&\lambda_{k+1} = \lambda_k + (\mathbf{z}_{k+1} - \mathbf{y}_{k+1}).
\end{aligned} \tag{6}$$

3. Modeling the Propagation in Feedforward Neural Network

In the standard feedforward neural network, the propagation from the first layer to the last layer can be expressed as:

$$\mathbf{x}_{k+1} = \Phi(\mathbf{W}_k \mathbf{x}_k), \tag{7}$$

where $\mathbf{x}_k$ is the output of the k-th layer, Φ is the activation function such as the sigmoid and ReLU, and $\mathbf{W}_k$ is a linear transformation. As claimed in Section 1.3, we do not consider the optimal weights during the structure design stage. Thus, we fix the matrix $\mathbf{W}_k$ as $\mathbf{W}$ to simplify the analysis.

We want to relate (7) with the gradient descent procedure (1). The critical step is to find an objective $F(\mathbf{x})$ to minimize.

Lemma 1 *Suppose* $\mathbf{W}$ *is a symmetric and positive definite matrix*[8]. *Let* $\mathbf{U} = \sqrt{\mathbf{W}}$. *Then there exists a function* $f(\mathbf{x})$ *such that (7) is equivalent to minimizing* $F(\mathbf{x}) = f(\mathbf{U}\mathbf{x})$ *using the following steps:*

1. *Define a new variable* $\mathbf{z} = \mathbf{U}\mathbf{x}$,
2. *Using (1) to minimize* $f(\mathbf{z})$,
3. *Recovering* $\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_k$ *from* $\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_k$ *via* $\mathbf{x} = \mathbf{U}^{-1}\mathbf{z}$.

Proof We can find $\Psi(z)$ such that $\Psi'(z) = \Phi(z)$ for the commonly used activation function $\Phi(z)$. Then we can have that $\nabla_{\mathbf{z}} \sum_i \Psi(\mathbf{U}_i^T \mathbf{z}) = \mathbf{U}\Phi(\mathbf{U}^T\mathbf{z}) = \mathbf{U}\Phi(\mathbf{U}\mathbf{z})$. So if we let

$$f(\mathbf{z}) = \frac{\|\mathbf{z}\|^2}{2} - \sum_i \Psi(\mathbf{U}_i^T \mathbf{z}), \tag{8}$$

where $\mathbf{U}_i$ is the i-th column of $\mathbf{U}$, then we have

$$\nabla f(\mathbf{z}_k) = \mathbf{z}_k - \mathbf{U}\Phi(\mathbf{U}\mathbf{z}_k). \tag{9}$$

Using (1) to minimize (8), we have

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \nabla f(\mathbf{z}_k) = \mathbf{U}\Phi(\mathbf{U}\mathbf{z}_k). \tag{10}$$

[7] For direct use in our network design, we fix the penalty parameter to 1.

[8] This assumption is just for building the connection between network design and optimization algorithms. $\mathbf{W}$ will be learnt from data once the structure of network is fixed.

	Activation function	Optimization objective $f(\mathbf{x})$
Sigmoid	$\frac{1}{1+e^{-x}}$	$\frac{\lVert\mathbf{x}\rVert^2}{2} - \sum_i \left[\mathbf{U}_i^T\mathbf{x} + \log\left(\frac{1}{e^{\mathbf{U}_i^T\mathbf{x}}} + 1\right)\right]$
tanh	$\frac{1-e^{-2x}}{1+e^{-2x}}$	$\frac{\lVert\mathbf{x}\rVert^2}{2} - \sum_i \left[\mathbf{U}_i^T\mathbf{x} + \log\left(\frac{1}{e^{2\mathbf{U}_i^T\mathbf{x}}} + 1\right)\right]$
Softplus	$\log(e^x + 1)$	$\frac{\lVert\mathbf{x}\rVert^2}{2} - \sum_i \left[C - \mathrm{polylog}(2, -e^{\mathbf{U}_i^T\mathbf{x}})\right]$
Softsign	$\frac{x}{1+\lvert x\rvert}$	$\frac{\lVert\mathbf{x}\rVert^2}{2} - \sum_i \phi_i(\mathbf{x})$, where $\phi_i(\mathbf{x}) = \begin{cases} \mathbf{U}_i^T\mathbf{x} - \log(\mathbf{U}_i^T\mathbf{x} + 1), & \text{if } \mathbf{U}_i^T\mathbf{x} > 0, \\ -\mathbf{U}_i^T\mathbf{x} - \log(\mathbf{U}_i^T\mathbf{x} - 1), & \text{otherwise} \end{cases}$
ReLU	$\begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases}$	$\frac{\lVert\mathbf{x}\rVert^2}{2} - \sum_i \phi_i(\mathbf{x})$, where $\phi_i(\mathbf{x}) = \begin{cases} \frac{(\mathbf{U}_i^T\mathbf{x})^2}{2}, & \text{if } \mathbf{U}_i^T\mathbf{x} > 0, \\ 0, & \text{otherwise} \end{cases}$
Leaky ReLU	$\begin{cases} x, & \text{if } x > 0, \\ \alpha x, & \text{if } x \leq 0. \end{cases}$	$\frac{\lVert\mathbf{x}\rVert^2}{2} - \sum_i \phi_i(\mathbf{x})$, where $\phi_i(\mathbf{x}) = \begin{cases} \frac{(\mathbf{U}_i^T\mathbf{x})^2}{2}, & \text{if } \mathbf{U}_i^T\mathbf{x} > 0, \\ \frac{\alpha(\mathbf{U}_i^T\mathbf{x})^2}{2}, & \text{otherwise} \end{cases}$
ELU	$\begin{cases} x, & \text{if } x > 0, \\ a(e^x - 1), & \text{if } x \leq 0. \end{cases}$	$\frac{\lVert\mathbf{x}\rVert^2}{2} - \sum_i \phi_i(\mathbf{x})$, where $\phi_i(\mathbf{x}) = \begin{cases} \frac{(\mathbf{U}_i^T\mathbf{x})^2}{2}, & \text{if } \mathbf{U}_i^T\mathbf{x} > 0, \\ a(e^{\mathbf{U}_i^T\mathbf{x}} - \mathbf{U}_i^T\mathbf{x}), & \text{otherwise} \end{cases}$
Swish	$\frac{x}{1+e^{-x}}$	$\frac{\lVert\mathbf{x}\rVert^2}{2} - \sum_i \left[\frac{(\mathbf{U}_i^T\mathbf{x})^2}{2} + \mathbf{U}_i^T\mathbf{x}\log\left(\frac{1}{e^{\mathbf{U}_i^T\mathbf{x}}} + 1\right) - \mathrm{polylog}\left(2, -\frac{1}{e^{\mathbf{U}_i^T\mathbf{x}}}\right)\right]$

Table 1: The optimization objectives for the common activation functions.

Now we define a new function

$$F(\mathbf{x}) = f(\mathbf{U}\mathbf{x}).$$

Let $\mathbf{z} = \mathbf{U}\mathbf{x}$ for variable substitution and minimize $f(\mathbf{z})$ to obtain a sequence of $\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_k$. Then we use this sequence to recover $\mathbf{x}$ by $\mathbf{x} = \mathbf{U}^{-1}\mathbf{z}$, which leads to

$$\mathbf{x}_{k+1} = \mathbf{U}^{-1}\mathbf{z}_{k+1} = \mathbf{U}^{-1}\mathbf{U}\Phi(\mathbf{U}\mathbf{z}_k) = \Phi(\mathbf{U}\mathbf{z}_k) = \Phi(\mathbf{U}^2\mathbf{x}_k) = \Phi(\mathbf{W}\mathbf{x}_k).$$

■

We list the objective function $f(\mathbf{x})$ for the commonly used activation functions in Table 3.

4. From GD to Other Optimization Algorithms

As shown in Section 3, the propagation in the general feedfroward neural network can be seen as using the gradient descent algorithm to minimize some function $F(\mathbf{x})$. In this section, we consider to use other algorithms to minimize the same function $F(\mathbf{x})$.

The Heavy Ball Algorithm. We first consider iteration (2). Similar to the proof in Section 3, we use the following three steps to minimize $F(\mathbf{x}) = f(\mathbf{U}\mathbf{x})$:

1. Variable substitution $\mathbf{z} = \mathbf{U}\mathbf{x}$.
2. Using (2) to minimize $f(\mathbf{z})$, which is defined in (8). Then (2) becomes

   $$\mathbf{z}_{k+1} = \mathbf{z}_k - \nabla f(\mathbf{z}_k) + \beta(\mathbf{z}_k - \mathbf{z}_{k-1}) = \mathbf{U}\Phi(\mathbf{U}\mathbf{z}_k) + \beta(\mathbf{z}_k - \mathbf{z}_{k-1}),$$

   where we use (9) in the second equation.
3. Recovering $\mathbf{x}$ from $\mathbf{z}$ via $\mathbf{x} = \mathbf{U}^{-1}\mathbf{z}$:

   $$\begin{aligned} \mathbf{x}_{k+1} =& \mathbf{U}^{-1}\mathbf{z}_{k+1} = \Phi(\mathbf{U}\mathbf{z}_k) + \beta(\mathbf{U}^{-1}\mathbf{z}_k - \mathbf{U}^{-1}\mathbf{z}_{k-1}) \\ =& \Phi(\mathbf{U}^2\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1}) = \Phi(\mathbf{W}\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1}). \end{aligned} \tag{11}$$

Nesterov's Accelerated Gradient Descent Algorithm. Now we consider to use iteration (3). Following the same three steps, we have:

$$\mathbf{x}_{k+1} = \Phi\left(\mathbf{W}\left(\mathbf{x}_k + \beta_k(\mathbf{x}_k - \mathbf{x}_{k-1})\right)\right), \tag{12}$$

where $\beta_k = \frac{\theta_k(1-\theta_{k-1})}{\theta_{k-1}}$. Then we use iteration (4) to minimize $F(\mathbf{x})$. Following the same three steps, we have

$$\mathbf{x}_{k+1} = \sum_{j=0}^{k} h_{k+1,j}\Phi(\mathbf{W}\mathbf{x}_j) + \mathbf{x}_k - \sum_{j=0}^{k} h_{k+1,j}\mathbf{x}_j. \tag{13}$$

ADMM. At last, we use iteration (6) to minimize $F(\mathbf{x})$, which consists of the following steps:

$$\begin{aligned} \mathbf{x}'_{k+1} &= \frac{1}{2}\left(\Phi(\mathbf{W}\mathbf{x}'_k) + \mathbf{x}_k - \sum_{t=1}^{k}(\mathbf{x}'_t - \mathbf{x}_t)\right), \\ \mathbf{x}_{k+1} &= \frac{1}{2}\left(\Phi(\mathbf{W}\mathbf{x}_k) + \mathbf{x}'_{k+1} + \sum_{t=1}^{k}(\mathbf{x}'_t - \mathbf{x}_t)\right). \end{aligned} \tag{14}$$

Comparing (11), (12) (13) and (14) with (7), we can see that the new algorithms keep the basic $\Phi(\mathbf{W}\mathbf{x})$ operation but use some additional side paths, which inspires the new network structure design in Section 6.

5. Hypothesis: Faster Optimization Algorithm May Imply Better Network

In this section, we consider the general representation learning task: Given a set of data points $\{\{\mathbf{x}_0^i, l_i\} : i = 1, \cdots, m\}$, where $\mathbf{x}_0^i$ is the i-th data and l_i is its label, we want to find a deep neural network which can learn the best feature $\mathbf{f}_i$ for each $\mathbf{x}_0^i$, which exists in theory but actually unknown in reality, such that $\mathbf{f}_i$ can perfectly predict l_i. For simplicity, we assume that $\mathbf{x}_0^i$ and $\mathbf{f}_i$ have the same dimension. As clarified at the beginning of Section 1.3, in this paper, we only consider the learning model form $\mathbf{x}_0^i$ to $\mathbf{f}_i$ and do not consider the prediction model from $\mathbf{f}_i$ to l_i. We do not consider the optimal weights during the structure design stage and thus, we study a simplified neural network model with the same linear transformation $\mathbf{W}\mathbf{x}$ in different layers. Actually, this corresponds to the recurrent neural networks (Putzky and Welling, 2017). In this section we use $\{\mathbf{x}_0, \mathbf{f}\}$ instead of $\{\mathbf{x}_0^i, \mathbf{f}_i\}$ for simplicity.

5.1. Same Linear Transformation in Different Layers

We first consider the simplified neural network model with the same linear transformation $\mathbf{W}\mathbf{x}$ in different layers. As stated in Section 3, the propagation in the standard feedforward network can be seen as using the gradient descent algorithm to minimize some function $F(\mathbf{x})$. Assume that there exists a special function $F(\mathbf{x})$ with some parameter $\mathbf{U}$ dependent on $\{\mathbf{x}_0, \mathbf{f}\}$ such that $\mathbf{f} = \text{argmin}_{\mathbf{x}} F(\mathbf{x})$. Now we use different algorithms to minimize this $F(\mathbf{x})$ and we want to find the minimizer of $F(\mathbf{x})$ via as few iterations as possible.

When we use the gradient descent algorithm to minimize this $F(\mathbf{x})$ with initializer $\mathbf{x}_0$, the iterative procedure is equivalent to (7), which corresponds to the propagation in the feedforward neural network characterized by the parameter $\mathbf{U}$ discussed above. Let $\hat{\mathbf{f}}$ be the output of this feedforward neural network. As is known, the gradient descent algorithm needs $O\left(\frac{L}{\mu}\log\frac{1}{\epsilon}\right)$ iterations to reach an ϵ-accuracy solution, i.e., $\|\hat{\mathbf{f}}-\mathbf{f}\|\leq\epsilon$. In other words, this feedforward neural network needs $O\left(\frac{L}{\mu}\log\frac{1}{\epsilon}\right)$ layers for an ϵ-accuracy prediction.

When we use some faster algorithm to minimize this $F(\mathbf{x})$, e.g., the heavy ball algorithm and Nesterov's accelerated gradient algorithm, their iterative procedures are equivalent to (11) and (13), respectively and the new algorithms will need $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$ iterations for $\|\hat{\mathbf{f}}-\mathbf{f}\|\leq\epsilon$. That is, the networks corresponding to faster algorithms (also characterized by the same $\mathbf{U}$ but has different structures) will need fewer layers than the feedforward neural network discussed above.

We define a network with fewer layers to reach the same approximation accuracy as a better network. As is known, training a deep neural network model is a nonconvex optimization problem and it is NP-hard to reach its global minima. The training precess becomes more difficult when the network becomes deeper. So if we can find a network with fewer layers and no loss of approximation accuracy, it will make the training process much easier.

5.2. Different Linear Transformation in Different Layers

In the previous discussion, we require the linear transformation in different layers to be the same. This is not true except in recurrent neural networks and is only for theoretical explanation. Now we allow each layer to have a different transformation.

Assume that we have a network that is inspired by using some optimization algorithm to minimize $F(\mathbf{x})$, where its k-th layer has the operation of (7), (11), (12), (13) or (14). Denote its final output as $\mathrm{Net}_{\mathbf{U}}(\mathbf{x}_0)$. Then for a network with finite layers, we have $\|\mathbf{f}-\mathrm{Net}_{\mathbf{U}}(\mathbf{x}_0)\|\leq\epsilon$.

Now we relax the parameter $\mathbf{U}$ to be different in different layers. Then the output can be rewritten as $\mathrm{Net}_{\mathbf{U}_1,\cdots,\mathbf{U}_n}(\mathbf{x})$. Now we can use the following model to learn the parameters $\mathbf{U}_1,\cdots,\mathbf{U}_n$ with a fixed network structure:

$$\min_{\mathbf{U}_1,\cdots,\mathbf{U}_n}\|\mathbf{f}-\mathrm{Net}_{\mathbf{U}_1,\cdots,\mathbf{U}_n}(\mathbf{x}_0)\|^2 \tag{15}$$

and denote $\mathbf{U}_1^*,\cdots,\mathbf{U}_n^*$ as the solution. Then we have $\|\mathbf{f}-\mathrm{Net}_{\mathbf{U}_1^*,\cdots,\mathbf{U}_n^*}(\mathbf{x}_0)\|\leq\|\mathbf{f}-\mathrm{Net}_{\mathbf{U}}(\mathbf{x}_0)\|$, which means that different linear transformation in different layers will not make the network worse. In fact, model (15) is the general training model for a neural network with a fixed structure.

5.3. Simulation Experiment

In this section, we verify our hypothesis that the network structures inspired by faster algorithms may be better than the ones inspired by slower algorithms.

We compare five neural network structures, which are inspired by the gradient descent algorithm, the heavy ball algorithm, ADMM and two variants of Nesterov's accelerated

depth	ADMM (14)	GD (7)	HB (11)	AGD (12)	AGD2 (13)
10	1.07576	1.00644	1.00443	1.00270	1.00745
20	1.07495	1.00679	1.00449	1.00215	1.00227
30	1.07665	1.00652	1.00455	1.00204	1.00086
40	1.07749	1.00653	1.00457	1.00213	0.99964

Table 2: MSE comparisons of different optimization algorithm inspired neural network structures.

gradient algorithm, which have the operation of (7), (11), (14), (12) and (13) at each layer, respectively. We set $\beta = 0.3$ for (11) and the parameters of (12) and (13) are exactly the same with their corresponding optimization algorithms. We use the sigmoid function for Φ and $\mathbf{W}\mathbf{x}$ is a full-connection linear transformation. Then we use model (15)[9] to train the parameters of each layer under the fixed network structures. We generate 10,000 random pairs of $\{\mathbf{x}_0^i, \mathbf{f}_i\}$ in $N(\mathbf{0}, \mathbf{I})$ as the training data. Each $\mathbf{x}_0^i$ and $\mathbf{f}_i$ has a dimension of 100. We use $\mathbf{x}_0^i$ as the input of the network and use its output to fit $\mathbf{f}_i$. We report the Mean Squared Error (MSE) loss value of the five aforementioned models with different depths after training 1,000 epoches.

Table 2 shows the experimental results. We can see that HB, AGD and AGD2 inspired neural network structures perform better than GD inspired network structure. This corresponds to the fact that the HB algorithm and AGD algorithm have a better theoretical convergence rate than the GD algorithm. The ADMM inspired network performs the worst. In fact, although ADMM has been widely used in practice, it does not have a faster theoretical convergence rate than GD. We also observe that the MSEs of GD, HB and AGD inspired network will not always decrease when the depth increases. This means that the deeper networks with GD, HB and AGD inspired structures are harder to train. However, the AGD2 inspired networks can still be efficiently trained with a larger depth when GD, HB and AGD inspired structures fail. This AGD2 is better may be because it has better numerical stability, although it is theoretically equivalent to AGD if there is no numerical error. Such a phenomenon is yet to be further explored.

6. Engineering Implementation

In the above section, we hypothesize and verify that faster optimization algorithm inspired neural network structure may need fewer layers without accuracy loss. In this section, we consider the practical implementations in engineering. Specifically, we consider the network structures inspired by algorithm iterations (7), (11), (12), (13) and (14).

We define the following three meta operations for practical implementation.

Relax Φ and $\mathbf{W}$. We use $\mathbf{W}\mathbf{x}$ as the linear transformation with full-connection in Section 4, which is the product of a matrix $\mathbf{W}$ and a vector $\mathbf{x}$. We may relax it to the convolution operation, which is also a linear transformation. Moreover, different layers may

[9] The optimization algorithms we present in Section 2 are not related to what algorithm we use to train model (15). The algorithms in Section 2 are for designing network structures.

have different weight matrix $\mathbf{W}$ and $\mathbf{W}$ may not be a square matrix, thus the dimensions of input and output may be different. Φ is the nonlinear transformation defined by the activation function. It can also be relaxed to pooling and batch normalization (BN). Moreover, $\Phi(\cdot)$ can be a composite of nonlinear activation, pooling, BN, convolution or full-connection linear transformation. Using the different combinations of these operations, the network structure (7) covers many famous CNNs, e.g., LeNet LeCun et al. (1998) and VGG Simonyan and Zisserman (2015). The activation function can also be different for different layers, e.g., be learnable Jin et al. (2016).

In the following discussions, we replace $\Phi(\mathbf{W}\mathbf{x})$ with the operator $T(\mathbf{x})$ for more flexibility.

Adaptive Coefficients. Inspired by (11), (12) and (13), we can design some practical neural network structures. However, the coefficients in these formulations may be impractical.So we keep the structure inspired by these formulations but allow the coefficient to have other values or even be learnable. Specifically, we rewrite (11) as

$$\mathbf{x}_{k+1} = T(\mathbf{x}_k) + \beta_1 \mathbf{x}_k + \beta_2 \mathbf{x}_{k-1}, \tag{16}$$

where β_1 and β_2 can be set as any constants, e.g., 0, which means that we drop the corresponding term. It can also be co-optimized with the training of network's weights.

The structure of (16) is illustrated in Figure 1(a). The symbol $\bigoplus$ means that $\mathbf{x}_{k+1}$ is a combination of $T(\mathbf{x}_k)$, $\mathbf{x}_k$ and $\mathbf{x}_{k-1}$.

Now we consider (12) and (13). Rewrite them as

$$\mathbf{x}_{k+1} = T(\beta_1 \mathbf{x}_k + \beta_2 \mathbf{x}_{k-1}) \tag{17}$$

and

$$\mathbf{x}_{k+1} = \sum_{j=0}^{k} \alpha_{k+1}^{j} T(\mathbf{x}_j) + \sum_{j=0}^{k} \beta_{k+1}^{j} \mathbf{x}_j. \tag{18}$$

All the coefficients α and β can be set as any constants, e.g., 0 or following (3) or (5). They can also be co-trained with the weights of the network.

We demonstrate the structures of (17) and (18) in Figure 1(b) and 1(c). From Figure 1(b) we can see that it first makes a combination of $\mathbf{x}_k$ and $\mathbf{x}_{k-1}$ is then the operation T follows. In Figure 1(c), $\mathbf{x}_{k+1}$ combines all of $T(\mathbf{x}_1), \cdots, T(\mathbf{x}_k)$ and $\mathbf{x}_1, \cdots, \mathbf{x}_k$.

It is known that ResNet adds an additional skip connection that bypasses the non-linear transformations with an identity transformation:

$$\mathbf{x}_{k+1} = T(\mathbf{x}_k) + \mathbf{x}_k.$$

The structure of ResNet can be recovered from (16) by setting $\beta_2 = 0$. If we also set $\beta_2 = 0$ in (17), then it is similar to the structure of ResNet. The difference is that (16) performs the operation T before adding the skip connection while (17) do it after the skip connection.

DenseNet is an extension of ResNet, which connects each layer with all its following layers. Consequently, the k-th layer receives the feature-maps of all preceding layers as its input, and produces

$$\mathbf{z}_{k+1} = T([\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_k]), \tag{19}$$

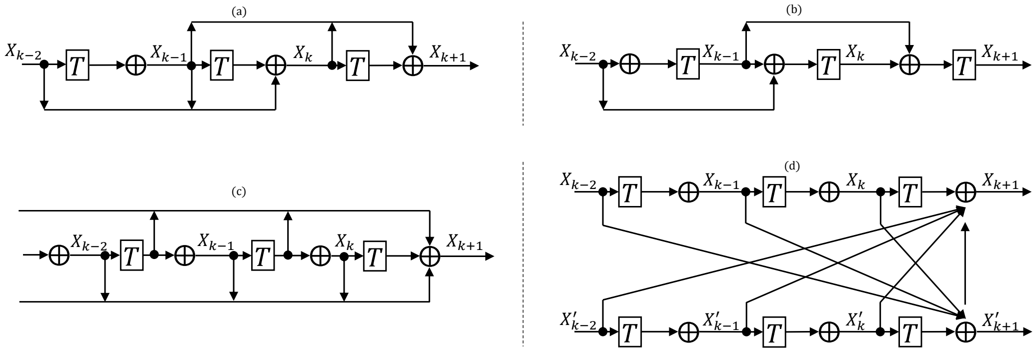

Figure 1: Demonstrations of network structures (16), (17), (18) and (20) in (a), (b), (c) and (d), respectively.

where [ ] refers to the concatenating operation. (18) recovers the structure of DenseNet by setting $\beta_k^j = 0, \forall k, j$.

We can also rewrite (14) as

$$\begin{aligned} \mathbf{x}'_{k+1} =& T(\mathbf{x}'_k) + \sum_{t=1}^{k} \alpha_t \mathbf{x}'_t + \sum_{t=1}^{k} \beta_t \mathbf{x}_t, \\ \mathbf{x}_{k+1} =& T(\mathbf{x}_k) + \sum_{t=1}^{k} \alpha_t \mathbf{x}'_t + \sum_{t=1}^{k} \beta_t \mathbf{x}_t. \end{aligned} \tag{20}$$

Figure 1(d) demonstrates the structure of (20). From the figure we can see that two paths interact with each other. Different from the previous network structures, the ADMM inspired network has two parallel paths, which makes the network wider. It also recovers the DMRNet in Zhao et al. (2017), which can be expressed as

$$\begin{aligned} \mathbf{x}'_{k+1} =& T(\mathbf{x}'_k) + \mathbf{x}'_k/2 + \mathbf{x}_k/2, \\ \mathbf{x}_{k+1} =& T(\mathbf{x}_k) + \mathbf{x}'_k/2 + \mathbf{x}_k/2. \end{aligned}$$

We list the optimization algorithms that relate to the commonly used existing network structures in Table 3.

Block Based Structure. (16), (17) and (18) are not available when the size of feature-maps changes, especially when down-sampling is used. To facilitate down-sampling, we can divide the network into multiple connected blocks. The formulationtions (16), (17) and (18) are used in each block. Such an operation is used in both ResNet and DenseNet. In the following sections, as examples we will give the explicit implementation of HB inspired network structure (16) and AGD inspired network structure (18).

6.1. HB Inspired Network

In this section, we describe the practical implementation of the neural network structure inspired by the heavy ball algorithm (2), which is used in our experiments. Specifically, we

Algorithm	Network Structure	Transforming Setting
GD (1)	CNN	$\mathbf{Wx} \rightarrow$convolution
HB (2)	ResNet	$\beta_2 = 0$ in (16)
AGD (4)	DenseNet	$\beta = 0$, $\alpha = 1$ in (18)
ADMM (6)	DMRNet	$\alpha_k = \beta_k = \frac{1}{2}$ in (20)

Table 3: Optimization algorithms and their inspired network structures.

implement the HB inspired network by setting $\beta = 1$ directly in (2):

$$\mathbf{x}_{k+1} = T(\mathbf{x}_k) + \mathbf{x}_k - \mathbf{x}_{k-1},$$

here T is a composite function including two weight layers. According to the residual structure in ResNet, the first layer is composed of three consecutive operations: convolution, batch normalization, and ReLU, while the second one is performed only with convolution and batch normalization. The feature maps are down-sampled at the first layer of each block by convolution with a stride of 2.

6.2. AGD Inspired Network

In line with the analysis above, we introduce the AGD inspired network of (18) as follow, which is easy to implement.

$$\mathbf{z}_{k+1} = \sum_{j=0}^{k} \alpha_{k+1}^{j} T(\mathbf{z}_j) + \beta \left(\mathbf{z}_k - \sum_{j=0}^{k} h_{k+1}^{j} \mathbf{z}_j \right), \tag{21}$$

where T is the composite function including batch normalization, ReLU and convolution, following DenseNet. Different from DenseNet, where all preceding layers $(\mathbf{z}_j, j < k+1)$ are concatenated first and then mapped by T to produce $\mathbf{z}_{k+1}$, AGD inspired network makes each preceding layer $\mathbf{z}_j$ produce its own output first, and then sum the outputs by weights α_{k+1}^{j}. The weights α_{k+1}^{j} of the first term, are co-optimized with the network, and the weights h_{k+1}^{j} of the third term, are calculated by formulation (5). The parameter β is set to be 0.1 in our experiments.

7. Experiments

7.1. Datasets and Training Details

CIFAR Both CIFAR-10 and CIFAR-100 datasets consist of 32 × 32 colored natural images. The CIFAR-10 dataset has 60,000 images in 10 classes, while the CIFAR-100 dataset has 100 classes, each of which containing 600 images. Both are split into 50,000 training images and 10,000 testing images. For image preprocessing, we normalize the images by subtracting the mean and dividing by the standard deviation. Following common practice, we adopt a standard scheme for data augmentation. The images are padded by 4 pixels on each side, filled with 0, resulting in 40 × 40 images, and then a 32 × 32 crop is randomly sampled from each image or its horizontal flip.

Model	CIFAR-10	CIFAR-100	CIFAR-10(+)	CIFAR-100(+)
ResNet $(n = 9)$	10.05	39.65	5.32	26.03
HB-Net (16) $(n = 9)$	10.17	38.52	5.46	26
ResNet $(n = 18)$	9.17	38.13	5.06	24.71
HB-Net (16) $(n = 18)$	8.66	36.4	5.04	23.93
DenseNet $(k = 12, L = 40)^*$	7	27.55	5.24	24.42
AGD-Net (18) $(k = 12, L = 40)$	6.44	26.33	5.2	24.87
DenseNet $(k = 12, L = 52)$	6.05	26.3	5.09	24.33
AGD-Net (18) $(k = 12, L = 52)$	5.75	24.92	4.94	23.84

Table 4: Error rates (%) on CIFAR-10 and CIFAR-100 datasets and their augmented versions. '*' denotes the result reported by (Huang et al., 2017). Others are implemented by ourselves. '+' denotes datasets with standard augmentation. We compare AGD-Net and HB-Net with DenseNet and ResNet, respectively, because they have very similar structures.

Model	top-1(%)	top-5(%)
ResNet-34	26.73	8.74
HB-Net-34	26.33	8.56
DenseNet-121	25.02	7.71
AGD-Net-121	24.62	7.39

Table 5: Error rates (%) on ImageNet when HB-Net and AGD-Net have the same depth as their baselines.

ImageNet We also test the vadility of our models on ImageNet, which contains 1.2 million training images, 50,000 validation images, and 100,000 test images with 1000 classes. We adopt standard data augmentation for the training sets. A 224 × 224 crop is randomly sampled from the images or horizontal flips. The images are normalized by mean values and standard deviations. We report the single-crop error rate on the validation set.

Training Details For fair comparison, we train our ResNet based models and DenseNet based models using training strategies adopted in the DenseNet paper (Huang et al., 2017). Concretely, the models are trained by stochastic gradient descent (SGD) with 0.9 Nesterov momentum and 10^{-4} weight decay. We adopt the weight initialization method in (He et al., 2015a), and use Xavier initialization (Glorot and Bengio, 2010) for the fully connected layer. For CIFAR, we train 300 epoches in total with a batchsize of 64. The learning rate is set to be 0.1 initially, and divided by 10 at 50% and 75% of the training procedure. For ImageNet, we train 100 eoches and drop learning rate at epoch 30, 60, and 90. The batchsize is 256 among 4 GPUs.

7.2. Comparison with State of The Arts

The experimental results on CIFAR are shown in Table 4, where the first two blocks are for ResNet based models and the last two blocks are for DenseNet based models. For ResNet based models, we conduct experiments with parameter n of 9 and 18, corresponding to a depth of 56 and 110. For DenseNet based models, we consider two cases where the growth rate k is 12 and the depth L equals to 40 and 52, respectively. We do not consider a larger DenseNet model (*e.g.* $L = 100$) due to the memory constraint of our single GPU. We can see that our proposed AGD-Net and HB-Net have a better performance than their respective baseline. For DenseNet based models, when $k = 12$ and $L = 40$, our model has an improvement on all datasets except for augmented CIFAR-100. When $k = 12$ and $L = 50$, AGD-Net's superiority is obvious on all datasets. Similar to DenseNet based models, the superiority of HB-Net over ResNet increases as the model capacity goes larger from $n = 9$ to $n = 18$. Besides, as reported by the ResNet paper (He et al., 2015b), when $n = 18$, the standard training strategy is difficult to converge and a warming-up is necessary for training. Our reimplementation of ResNet ($n = 18$) in Table 4 indeed needs repeating the experiments to get a converged result. But for HB-net, we can use exactly the same training strategy adopted by other models and get a converged performance with training only once. Therefore, the training procedure of HB-Net is more stable than original ResNet when the network goes deeper.

As shown in Table 5, our proposed structures are also effective on the ImageNet dataset. Both HB-ResNet and AGD-Net has the better performance than their baselines. All the above experiments show that our design methodology is very promising.

8. Conclusion and Future Work

In this paper, we use the inspiration from optimization algorithms to design neural network structures. We propose the hyphothesis that a faster algorithm may inspire us to design a better neural network. We prove that the propagation in the standard feedforward network with the same linear transformation in different layers is equivalent to minimizing some functions using the gradient descent algorithm. Based on this observation, we replace the gradient descent algorithm with the faster heavy ball algorithm and Nesterov's accelerated gradient algorithm to design better network structures, where ResNet and DenseNet are two special cases of our design.

Our methodology is still preliminary and not conclusive as many engineering tricks may also affect the performance of neural networks greatly. Nonetheless, our methodology can serve as the start point of network design. Practitioners can easily make changes to optimization algorithm inspired networks out of various insights and integrate various engineering tricks to produce even better results. Such a practice should be much easier than designing from scratch.

Acknowledgments

Z. Lin was supported by National Basic Research Program of China (973 Program) (grant no. 2015CB352502), National Science Foundation (NSF) of China (grant nos. 61625301 and 61731018), Qualcomm, and Microsoft Research Asia.

References

B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcemen learning. In *arxiv:1611.02167*, 2016.

A. Beck and M. Teboulle. A fast iterative shrinkage thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sciences*, 2(1):183–202, 2009.

J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *ICML*, 2013.

D. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, Ma, 1999.

C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang. AdaNet: Adaptive structure learning of artificial nerual networks. In *ICML*, 2017.

T. Domhan, J. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

D. Gabay. Applications of the method of multipliers to variational inequalities. *Studies in Mathematics and its applications*, 15:299–331, 1983.

R. Girshick. Fast R-CNN. In *ICCV*, 2015.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *ICML*, 2010.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015a.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2015b.

G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan. Deep learning with s-shaped rectified linear activation units. In *AAAI*, 2016.

A. Krzhevsky, I. Sutshever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

K. Kulkarni, S. Lohit, P. Turaga, R. Kerviche, and A. Ashok. Reconnet: Non-iterative reconstruction of images from compressively sensed mmeasuremets. In *CVPR*, 2016.

T. Kwok and D. Yeung. Constructive algorithms for structure learning feedforward nerual networks for regression problems. *IEEE Trans. on Neural Networks*, 8(3):630–645, 1997.

H. Lam, F. Leung, and P. Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Trans. on Neural Networks*, 14:79–88, 2003.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

Z. Lin, R. Liu, , and Z. Su. Linearized alternating direction method with adaptive penalty for low-rank representation. In *NIPS*, 2011.

J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

L. Ma and K. Khorasani. A new strategy for adaptively constructing multiplayer feedforward neural networks. *Neurocomputing*, 51:361–385, 2003.

Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

B. Polyak. ome methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

P. Putzky and M. Welling. Recurrent inference machines for solving inverse problems. In *arXiv:1706.04008*, 2017.

S. Ren, R. Girshick K. He, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Trans. PAMI*, 39:1137–1149, 2016.

S. Saxena and J. Verbeek. Convolutional neural fabrics. In *NIPS*, 2016.

J. Schaffer, D. Whitley, and L. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 1992.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

P. Verbancsics and J. Harguess. Generative neuroevolution for deep learning. In *arxiv:1312.5355*, 2013.

B. Xin, Y. Wang, W. Gao, B. Wang, and D. Wipf. Maximal sparsity with deep networks? In *NIPS*, 2016.

Y. Yang, J. Sun, H. Li, and Z. Xu. Deep admm-net for compressive sensing mri. In *NIPS*, 2016.

Y. Yang, Z. Zhong, T. Shen, and Z. Lin. Convolutional neural networks with alternately updated clique. In *CVPR*, 2018.

J. Zhang and B. Ghanem. ISTA-Net: Iterative shrinkage-thresholding algorithm inspired deep network for image compressie sensing. In *arxiv:1706.01929*, 2017.

L. Zhao, J. Wang, X. Li, Z. Tu, and W. Zeng. Deep convolutional neural networks with merge-and-run mappings. In *arxiv:1611.07718*, 2017.

Z. Zhong, J. Yan, W. Wu, J. Shao, and C. Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018.

B. Zoph and Q. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.