SCIENCE CHINA
Information Sciences

• **RESEARCH PAPER** •

Optimization-inspired Manual Architecture Design and Neural Architecture Search

Yibo YANG[1], Zhengyang SHEN[2], Huan LI[3] & Zhouchen LIN[4,5,6*]

[1]*JD Explore Academy, Beijing* 100176*, China;*
[2]*School of Mathematical Sciences, Peking University, Beijing* 100871*, China;*
[3]*Institute of Robotics and Automatic Information Systems, College of Artificial Intelligence, Nankai University, Tianjin* 300071*, China;*
[4]*Key Laboratory of Machine Perception (MOE), School of AI, Peking University, Beijing* 100871*, China;*
[5]*Institute for Artificial Intelligence, Peking University, Beijing* 100871*, China;*
[6]*Pazhou Lab, Guangzhou* 510335*, China*

Abstract Neural architecture has been a research focus in recent years due to its importance in deciding the performance of deep networks. Representative ones include the ResNet with skip connections and the DenseNet with dense connections. However, a theoretical guidance for manual architecture design and neural architecture search is still lacking. In this paper, we propose a manual architecture design framework which is inspired by optimization algorithms, based on the hypothesis that a faster optimization algorithm may lead to a better neural architecture. Specifically, we prove that the propagation in the feedforward neural network with the same linear transformation in different layers is equivalent to minimizing a cost 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 neural architectures. As a result, we find that ResNet and DenseNet can be considered as two special cases of our developed architectures. The optimization-inspired manual architecture design offers not only theoretical guidance, but also better performances of image recognition on the CIFAR-10, CIFAR-100, and ImageNet datasets. Besides, we show that our method is also useful for neural architecture search by offering a good initial search point or guiding the search space.

Keywords deep neural network, manual architecture design, neural architecture search, image recognition, optimization algorithms, learning-based optimization

Citation Yang Y, Shen Z, Li H, et al. Optimization-inspired Manual Architecture Design and Neural Architecture Search. Sci China Inf Sci, for review

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 [1], semantic segmentation [2] and object detection [3, 4]. After the breakthrough result in the ImageNet classification challenge [1], different kinds of neural architectures have been proposed and the performance is improved gradually by increasing the network depth and capacity [5–7]. Highway networks [8] and ResNets [9] introduce the skip connections and successfully train very deep networks. Later manual architecture design adopts more complex topologies and connections [10–12]. All these neural architectures are designed by humans mainly based on empirical analyses and experimental observations. However, they may not be the optimal choice of architecture and have the best performance overhead trade-off, which inspires a series of exploration on neural architecture search in recent literature. Popular methods include reinforcement learning that assigns a better architecture on the validation set with a higher reward [13–20], evolution algorithms [21–27], and gradient-based methods [28–34]. Despite the success of these architectures, neither manual architecture design nor neural architecture search has a clear understanding on deep network behaviors. How to

*Corresponding author (email: zlin@pku.edu.cn) develop a theoretical guidance for neural architectures to benefit both manual architecture design and neural architecture search is the core challenge that we target on in this study.

In order to improve the interpretability of neural architectures, some studies try to build connections to the areas with rich theories and analytical tools, such as numerical methods of ODEs [35–38] and optimization algorithms [39–44]. Specifically, the forward propagation of a neural network can be deemed as the iterative process of an optimization algorithm, so that the final layer of the network outputs an optimal solution $\mathbf{x}^*$ that minimizes a cost function, known as learning-based optimization. The architecture of learning-based optimization is transparent and explainable because it is translated from a well-developed algorithm in optimization theory. However, current optimization-inspired architecture is only limited to solving optimization problems, such as compressed sensing. We cannot use these methods with *known* objective functions to construct architectures for general purposes, such as image recognition. If we can move a step forward to build the connection between optimization and the neural architecture used for general feature learning, a broader range of neural architectures can be explainable. Moreover, the connection may inspire us to improve manual architecture design and neural architecture search following the guidance from optimization theory.

Motivated by the analysis above, in this study, we generalize the idea of learning-based optimization. We prove that the forward propagation in a network for image recognition also corresponds to a gradient-based optimization algorithm. Bridging the connection between general neural architecture and optimization algorithm, we hypothesize that a faster optimization algorithm may lead to a better neural architecture. Accordingly, we develop new architectures inspired by traditional optimization algorithms. Our optimization-inspired manual architecture design not only has a theoretical guidance, but also performs better than traditional architectures. Moreover, we show that our method is also useful for neural architecture search (NAS) by offering a good initial search point or guiding the search space.

Our methodology is inspired by optimization algorithms and is applied to manual architecture design and neural architecture search that both lack theoretical guidance before our study. Specifically, our contributions include:

1. In the general case, for any 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 a cost function $F(\mathbf{x})$. As a comparison, the learning-based optimization architectures only adopt the soft thresholding as the nonlinear function and can be used only for the compressed sensing problem.

2. Based on the above observation, we propose the hypothesis that a faster optimization algorithm may correspond to a better neural architecture. Especially, we give the architectures inspired by the heavy ball algorithm (HBNet) and the Nesterov's accelerated gradient algorithm (AGDNet), respectively. They turn out to include ResNet and DenseNet as two special cases.

3. Experimental results on multiple datasets including CIFAR-10, CIFAR-100, and ImageNet verify that the optimization-inspired neural architectures are competitive with or even outperform their counterparts, ResNet and DenseNet. Further, in experiments of neural architecture search, we adopt the architecture of HBNet as the initial search point, and use the connection pattern of HBNet and AGDNet to modify the search space. Both show improvements over the original settings at low costs. These results show that our optimization-inspired methodology is promising for both manual architecture design and neural architecture search.

2 Related Work

Manual architecture design. Manual architecture design denotes the neural architectures designed by humans based on their empirical understanding on the tasks that the DNNs are applied to. In the early stage of manual architecture design, genetic algorithm [45, 46] based approaches were proposed to find both architectures and weights. However, networks designed by the genetic algorithms perform worse than only designing architecture [47]. [48] proposed a "Fabric" to sidestep the CNN architecture selection problem and it performs close to manual architecture design. [49] used Bayesian optimization for architecture selection and [50] used a meta-modeling approach to choose the type of layers and hyperparameters. [51], [52] and [53] used the adaptive strategy that grows the network layer by layer from a small network based on some principles, e.g., [53] minimized a loss value to balance the model complexity and the empirical risk minimization. In recent years, manual architecture design has been popular and revolutionized a wide range of computer vision tasks [1, 5–10, 12], but designing architectures suffers from relying on empirical knowledge and exhaustive engineering trials.

Neural architecture search. Because manual architecture design cannot ensure an optimal architecture, recent studies turn to use the Auto-ML technique to automatically produce satisfactory architectures, *a.k.a.* neural architecture search. The key difference between manual architecture design and neural architecture search is that design needs all detailed human specification on the architecture, while search only requires human to give an overall search space and then computer searches for the details. The current methods can be categorized into three frameworks. Reinforcement learning based searching introduces an agent to generate architectures and assign a better one with a higher reward [13, 14]. Follow-up studies focus on proposing better search space or algorithm to reduce the search cost [18–20]. Evolution-based searching uses evolutionary algorithms for optimizing neural architectures [21–27]. But the computation and time required by these methods are still not acceptable. Another line of searching is one-shot based methods that significantly reduce the search cost by treating all potential architectures as subgraphs of a super-net and sparing the efforts of evaluating each candidate [16, 29–34, 54–60]. In DARTS [29], the search space is relaxed to be differentiable, so that the super-net is trained jointly with the architecture variables. Despite its simplicity, DARTS-based methods suffer from instability and the gap between the architectures in search and evaluation. Some later studies adopt the Gumbel Softmax strategy to reduce the architectural gap [30, 34]. In [31], a progressive shrinking method is proposed to bridge the depth gap between search and evaluation. DenseNAS [33] and PC-DARTS [32] introduce another set of trainable parameters to model path probabilities. A recent study formulates NAS as a sparse coding problem [61]. However, these search methods lack theoretical guidance. Similar to manually designed ones, the searched architectures are not interpretable either. In this study, we show that our optimization-inspired methodology is beneficial to neural architecture search with clear guidance.

Optimization-inspired networks. In order to make a theoretical guidance for neural architecture and improve the interpretability, researchers build connections between neural network and some well-developed areas, such as numerical methods of ODEs and optimization methods, to analyze and specify neural architectures using their rich analytical tools. In particular, optimization-inspired networks refer to the architectures designed from the optimization perspective. The early optimization-inspired network [39] is derived by unrolling the traditional iterative optimization method, ISTA [62], 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 $\mathrm{prox}_{\alpha\|\cdot\|_1}(\mathbf{x}) = \mathrm{argmin}_{\mathbf{z}} \frac{1}{2}\|\mathbf{z}-\mathbf{x}\|^2 + a\|\mathbf{z}\|_1$, to solve the compressed sensing problem defined as $\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x}-\mathbf{y}\|^2 + \alpha\|\mathbf{x}\|_1$[1]. The main idea is to train a non-linear neural network where learnable weights $\mathbf{W}_k$ replace the fixed transformation matrix $\mathbf{A}$ in the iterative optimization method, the non-linear activation is achieved by the proximal operator, and the connection pattern corresponds to the calculation of each iteration, so the architecture is decided by the optimization method. Compared with the traditional iteration-based ISTA, the learning-based ISTA (LISTA) enjoys better convergence speed, and inspires a series of later studies [40–43, 63, 64]. In [65], a theoretical understanding is offered and the linear convergence of LISTA is proved. Apart from unrolling ISTA, there are also many works that unroll other optimization algorithms, such as iterative hard thresholding [40], approximate message passing [66], and alternating direction method of multipliers (ADMM) [43, 44].

Albeit these optimization-inspired networks have theoretical guidance and achieve great success, they can only be used to solve given optimization problems, such as sparse coding and compressed sensing for image *reconstruction*, instead of general *feature learning* tasks, such as image *recognition*. Our preliminary study[2] [67] aimed at generalizing the optimization-inspired network for image reconstruction into the general feature learning purpose. It established the connection between a faster optimization algorithm and its corresponding better neural architecture, and proposed to design architectures based on this connection[3]. However, it only focused on manual architecture design. In this study, we move a step forward to generalize this idea. We show that our methodology can also benefit neural architecture search in two ways, using the optimization-inspired networks as the initial search point or using the optimization-inspired connection pattern as a better search space. It brings theoretical interpretability for the searched architecture and improves the search results as well.

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

2) The preliminary version of this paper has appeared on conference ACML 2018 and a patent has been filed.

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

Concretely, the improvements and differences of this study over our conference version can be listed as:

1. We re-write and polish the contents that have been in the conference version. We add more experiments of the optimization-inspired networks on the ImageNet dataset in Section 8 of this journal version.

2. In Section 9, we extend our optimization-inspired methodology from manual architecture design to neural architecture search. We find that our method is also useful for neural architecture search via two ways, *i.e.* offering a good initial search point and guiding the search space.

3. We conduct experiments of neural architecture search on CIFAR and ImageNet to demonstrate our method in Section 9. We show that the optimization-inspired networks, HBNet and AGDNet, correspond to better search spaces, where better architectures can be searched from. The HBNet itself can also be used as a good initial search point, to make the search process more efficient. In contrast, our conference version only focused on optimization-inspired manual architecture design and did not contribute to neural architecture search.

4. Our all experiments, including the original results in the conference version and the newly added results of both manual architecture design and neural architecture search indicate that the optimization-inspired methodology can serve as a good way to design or search for high-performance neural architectures, which is much easier and more interpretable than designing manually from scratch or searching without any guidance. Based on our all contributions, we further conclude that the connection between optimization algorithm and DNN architecture has been empirically validated, and the proposed methodology is able to offer a good guidance for both manual architecture design and neural architecture search, which is beyond our conference version that only validates the benefit on manual architecture design with limited experimental results on ImageNet.

3 Reviews of Some Optimization Algorithms

In this section, we review some optimization algorithms, including the gradient descent (GD) [68], the heavy ball (HB) [69], Nesterov's accelerated gradient descent (AGD) [70] and the Alternating Direction Method of Multipliers (ADMM) [71, 72] that 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 iterates as[4]:

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

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 is derived from the similar idea to the heavy ball algorithm, but 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 $\theta_0 = 1$[5]. When $f(\mathbf{z})$ is μ-strongly convex[6] and its gradient is L-Lipschitz continuous[7], the heavy ball algorithm and Nesterov's accelerated gradient algorithm is able

4) 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.

5) 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 $\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$.

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

7) I.e., $\|\nabla f(\mathbf{y}) - \nabla f(\mathbf{x})\| \leqslant L\|\mathbf{y} - \mathbf{x}\|$.

to 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) can be reformulated as 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

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

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 of the following steps[8]:

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

where $\boldsymbol{\lambda}$ is the Lagrange multiplier.

4 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, $\mathbf{W}_k$ is a linear transformation, and Φ is the activation function such as the sigmoid and ReLU. We do not consider the optimal weights during the manual architecture design stage, since this stage only concerns the *architecture* of the network. Thus, we can fix the matrix $\mathbf{W}_k$ as $\mathbf{W}$ to simplify the analysis.

Our goal is 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[9]. 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:

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

8) For direct use in our network design, we fix the penalty parameter to 1.

9) This assumption is just for building the connection between network design and optimization algorithms. $\mathbf{W}$ will be learnt from data once the architecture of network is determined.

Table 1 The optimization objectives for the common activation functions.

	Activation function	Optimization objective $f(\mathbf{x})$
Sigmoid	$\frac{1}{1+e^{-x}}$	$\frac{\|\mathbf{x}\|^2}{2} - \sum_i \left[\mathbf{U}_i^T\mathbf{x} + \log\left(\frac{1}{e^{\mathbf{U}_i^T\mathbf{x}}} + 1\right)\right]$
tanh	$\frac{1-e^{-2x}}{1+e^{-2x}}$	$\frac{\|\mathbf{x}\|^2}{2} - \sum_i \left[\mathbf{U}_i^T\mathbf{x} + \log\left(\frac{1}{e^{2\mathbf{U}_i^T\mathbf{x}}} + 1\right)\right]$
Softplus	$\log(e^x+1)$	$\frac{\|\mathbf{x}\|^2}{2} - \sum_i \left[C - \text{polylog}(2, -e^{\mathbf{U}_i^T\mathbf{x}})\right]$
Softsign	$\frac{x}{1+\|x\|}$	$\frac{\|\mathbf{x}\|^2}{2} - \sum_i \phi_i(\mathbf{x})$, 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	$\begin{cases} x, & \text{if } x > 0, \\ 0, & \text{if } x \leqslant 0. \end{cases}$	$\frac{\|\mathbf{x}\|^2}{2} - \sum_i \phi_i(\mathbf{x})$, 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	$\begin{cases} x, & \text{if } x > 0, \\ \alpha x, & \text{if } x \leqslant 0. \end{cases}$	$\frac{\|\mathbf{x}\|^2}{2} - \sum_i \phi_i(\mathbf{x})$, 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	$\begin{cases} x, & \text{if } x > 0, \\ a(e^x-1), & \text{if } x \leqslant 0. \end{cases}$	$\frac{\|\mathbf{x}\|^2}{2} - \sum_i \phi_i(\mathbf{x})$, 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	$\frac{x}{1+e^{-x}}$	$\frac{\|\mathbf{x}\|^2}{2} - \sum_i \left[\frac{(\mathbf{U}_i^T\mathbf{x})^2}{2} + \mathbf{U}_i^T\mathbf{x}\log\left(\frac{1}{e^{\mathbf{U}_i^T\mathbf{x}}} + 1\right) - \text{polylog}\left(2, -\frac{1}{e^{\mathbf{U}_i^T\mathbf{x}}}\right)\right]$

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

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

Now we define a new function:

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

Let $\mathbf{z} = \mathbf{U}\mathbf{x}$ be 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:

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

As shown in Table 1, we list the objective function $f(\mathbf{x})$ for the commonly used activation functions.

5 From GD to Other Optimization Algorithms

In Section 4, the propagation in the general feedforward neural network can be deemed as using the gradient descent algorithm to optimize a cost 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. First, we consider the Iteration (2). Similar to the proof in Section 4, 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:

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

   where we use (9) in the second equation.

3. Recovering $\mathbf{x}$ from $\mathbf{z}$ via $\mathbf{x} = \mathbf{U}^{-1}\mathbf{z}$:

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

The Nesterov's Accelerated Gradient Descent Algorithm. Now we consider to use the 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 the 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 the 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 observe that the new algorithms keep the basic $\Phi(\mathbf{Wx})$ operation but use some additional side paths, which inspires our new manual architecture design in Section 7. The architecture derived by optimization algorithm X is referred to X-inspired architecture in our paper.

6 Hypothesis: Faster Optimization Algorithm May Imply Better Network

In this section, we deal with 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 aim to find a deep neural network that learns 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 in Section 1, we only consider the learning model from $\mathbf{x}_0^i$ to $\mathbf{f}_i$ and do not consider the prediction model from $\mathbf{f}_i$ to l_i. During our architecture design stage, the optimal weights are also not considered. Thus, we study a simplified neural network model with the same linear transformation $\mathbf{Wx}$ in different layers. Actually, this corresponds to the recurrent neural networks [73]. In this section we use $\{\mathbf{x}_0, \mathbf{f}\}$ instead of $\{\mathbf{x}_0^i, \mathbf{f}_i\}$ for simplicity.

6.1 Same Linear Transformation in Different Layers

We first assume that the simplified neural network model has the same linear transformation $\mathbf{Wx}$ in different layers. As stated in Section 4, the propagation in the standard feedforward neural network can be seen as an optimization process that minimizes a cost function $F(\mathbf{x})$ by the gradient descent algorithm. Assuming that there exists a special function $F(\mathbf{x})$ with the parameter $\mathbf{U}$ dependent on $\{\mathbf{x}_0, \mathbf{f}\}$ such that $\mathbf{f} = \text{argmin}_{\mathbf{x}} F(\mathbf{x})$, we use different algorithms to minimize $F(\mathbf{x})$ 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 feature of this feedforward neural network. It is known that the gradient descent algorithm needs $O\left(\frac{L}{\mu}\log\frac{1}{\epsilon}\right)$ iterations to reach an ϵ-accuracy solution, i.e., $\|\hat{\mathbf{f}} - \mathbf{f}\| \leqslant \epsilon$. In another word, $O\left(\frac{L}{\mu}\log\frac{1}{\epsilon}\right)$ layers are needed for this feedforward neural network to have an ϵ-accuracy prediction.

If we use a 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. In this way, the new algorithms will need $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$ iterations for $\|\hat{\mathbf{f}} - \mathbf{f}\| \leqslant \epsilon$. Therefore, the networks corresponding to faster algorithms (also characterized by the same $\mathbf{U}$ but has different architectures) will need fewer layers than the feedforward neural network discussed above.

Table 2 MSE comparisons of different optimization algorithm-inspired neural architectures.

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

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

6.2 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 practical except in recurrent neural networks and is only introduced for theoretical analyses. Now we allow each layer to have a different transformation.

Suppose that we have a network that is inspired by an optimization algorithm minimizing a cost function $F(\mathbf{x})$, where its k-th layer has the operation of (7), (11), (12), (13) or (14). We denote its final output as $\text{Net}_{\mathbf{U}}(\mathbf{x}_0)$. Then for a network with finite layers, we have $\|\mathbf{f} - \text{Net}_{\mathbf{U}}(\mathbf{x}_0)\| \leqslant \epsilon$.

Now we relax the parameter $\mathbf{U}$ to be different in different layers. As a result, the output can be rewritten as $\text{Net}_{\mathbf{U}_1,\cdots,\mathbf{U}_n}(\mathbf{x})$. Accordingly we can use the following model to learn the parameters $\mathbf{U}_1,\cdots,\mathbf{U}_n$ with a fixed neural architecture:

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

Denote $\mathbf{U}_1^*,\cdots,\mathbf{U}_n^*$ as the optimal solution. Then we have $\|\mathbf{f} - \text{Net}_{\mathbf{U}_1^*,\cdots,\mathbf{U}_n^*}(\mathbf{x}_0)\| \leqslant \|\mathbf{f} - \text{Net}_{\mathbf{U}}(\mathbf{x}_0)\|$, 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 neural architecture.

6.3 Simulation Experiment

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

We compare five neural architectures inspired by the gradient descent algorithm, the heavy ball algorithm, ADMM, and two variants of the Nesterov's accelerated gradient algorithm, corresponding to the operations in (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 as their corresponding optimization algorithms. We use the sigmoid function for Φ and $\mathbf{W}\mathbf{x}$ is a fully-connected layer. Then we use model (15)[10)] to train the parameters of each layer under the fixed neural architecture. We generate 10,000 random pairs of $\{\mathbf{x}_0^i, \mathbf{f}_i\}$ from the Gaussian distribution $N(\mathbf{0}, \mathbf{I})$ 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)[11)] loss value of the five aforementioned models with different depths after training 1,000 epoches.

Table 2 shows the experimental results. It is shown that HB, AGD, and AGD2-inspired architectures perform better than GD-inspired architecture. This is in line with 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, despite that 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 architectures do not always decrease as the depth increases. This indicates that the deeper

10) 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 neural architectures.

11) The MSE here does not represent the convergence speed of these optimization algorithms, but reflects the relative quality of their inspired neural architectures.

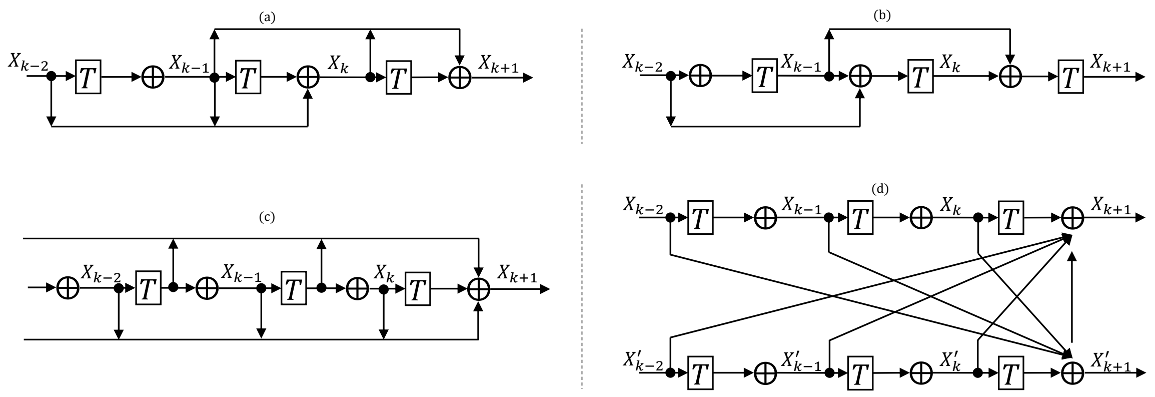

Figure 1 (Figure 1 in [67]) Illustrations of neural architectures (16), (17), (18) and (20) in (a), (b), (c) and (d), respectively.

networks with GD, HB, and AGD-inspired architectures are harder to train. However, the AGD2 inspired networks can still be efficiently trained with a larger depth when GD, HB, and AGD-inspired architectures fail. This may be because AGD2 has a 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.

7 Engineering Implementation

In the above section, we hypothesize and verify that faster optimization algorithm-inspired architecture needs fewer layers without accuracy loss. In this section, we consider the practical implementations in engineering. Concretely, we introduce the neural architectures inspired by algorithm iterations (7), (11), (12), (13), and (14).

We define the following three meta operations for practical implementation.

Relax Φ and W. We use $\mathbf{Wx}$ as the linear transformation with fully-connected in Section 5, which performs the product of a matrix $\mathbf{W}$ and a vector $\mathbf{x}$. We now relax it to the convolution operation, which is also a linear transformation. Moreover, different layers may have different weight matrix $\mathbf{W}$ and $\mathbf{W}$ may not be a square matrix, thus the dimensions of input and output can be different. Φ is the nonlinear transformation defined by the activation function. Note that it can also be relaxed to pooling and batch normalization (BN). Moreover, $\Phi(\cdot)$ can be a composite function of nonlinear activation, pooling, BN, convolution, or fully-connected layer. Using different combinations of these operations, the neural architecture (7) actually covers many known CNNs, e.g., LeNet [74] and VGG [6]. The activation function can also be different for different layers, e.g., be learnable in [75].

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

Adaptive Coefficients. Inspired by (11), (12), and (13), we can design some practical neural architectures. However, the coefficients in these formulations may be impractical. So we keep the architecture inspired by these formulations but allow the coefficients to have other values or 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 network parameters.

The architecture of (16) is shown in Figure 1(a), where $\bigoplus$ 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

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

Table 3 Optimization algorithms and their inspired neural architectures.

Algorithm	Neural Architecture	Transforming Setting
GD (1)	CNN	$\mathbf{Wx} \rightarrow$convolution
HB (2)	ResNet [9]	$\beta_2 = 0$ in (16)
AGD (4)	DenseNet [10]	$\beta = 0$, $\alpha = 1$ in (18)
ADMM (6)	DMRNet [76]	$\alpha_k = \beta_k = \frac{1}{2}$ in (20)

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-optimized with the network parameters.

We demonstrate the architectures 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 $\mathbf{x}_{k-1}$, 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 $\mathbf{x}_1, \cdots, \mathbf{x}_k$.

We know that ResNet adds an additional skip connection that bypasses the non-linear transformations with an identity transformation:

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

Actually the architecture 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 architecture of ResNet. The difference is that (16) performs the operation T before adding the skip connection, while (17) does it after the skip connection.

As an extension of ResNet, DenseNet connects different layers by concatenation. Consequently, the k-th layer receives the feature-maps of all preceding layers as its input, and produces:

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

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

We can also rewrite (14) as:

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

The architecture of (20) is illustrated in Figure 1(d), from which we can see that the two paths interact with each other.

Different from the previous neural architectures, the ADMM-inspired network has two parallel paths, which makes the network wider. Note that it also recovers the DMRNet in [76], 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 summarize the optimization algorithms that relate to the existing neural architectures in Table 3.

Block Based Architecture. (16), (17), and (18) are not available when the size of feature-maps changes, especially when down-sampling is used. To enable down-sampling, we can divide the network into multiple connected blocks. The formulations of (16), (17), and (18) are used in each block. Such an operation is used in both ResNet and DenseNet.

7.1 HB-Inspired Architecture (HB-Net)

In this section, we describe how to implement the neural architecture inspired by the heavy ball algorithm (2). Specifically, we implement the HB-inspired architecture by directly setting $\beta = 1$ in (2):

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

where T is a composite function including two weight layers. According to the residual branch 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. When down-sampling, the feature maps are down-sampled at the first layer by convolution with a stride of 2.

Table 4 Error rates (%) on CIFAR-10 and CIFAR-100 datasets and their augmented versions. '+' denotes datasets with standard augmentation. We compare AGD-Net and HB-Net with DenseNet and ResNet, respectively, because they are special cases of AGD-Net and HB-Net, respectively. '*' denotes that deep ResNet is not stable on CIFAR, so we implement for multiple times and report the converged result, while our HB-Net needs training only once.

Model	Params	CIFAR-10	CIFAR-100	CIFAR-10(+)	CIFAR-100(+)
Network in Network [77]	-	10.41	35.68	8.81	-
All-CNN [78]	-	9.08	-	7.25	33.71
Deeply Supervised Net [79]	-	9.69	-	7.97	34.57
Highway Network [8]	-	-	-	7.72	32.39
ResNet ($n = 9$)	0.85M	10.05	39.65	5.32	26.03
HB-Net (16) ($n = 9$)	0.85M	10.17	38.52	5.46	26
ResNet ($n = 18$)*	1.7M	9.17	38.13	5.06	24.71
HB-Net (16) ($n = 18$)	1.7M	8.66	36.4	5.04	23.93
DenseNet ($k = 12, L = 40$)	1.0M	7	27.55	5.24	24.42
AGD-Net (18) ($k = 12, L = 40$)	1.0M	6.44	26.33	5.2	24.87
DenseNet ($k = 12, L = 52$)	1.4M	6.05	26.3	5.09	24.33
AGD-Net (18) ($k = 12, L = 52$)	1.4M	5.75	24.92	4.94	23.84

7.2 AGD-Inspired Architecture (AGD-Net)

In line with our analyses, we introduce the AGD-inspired architecture of (18) as follows,

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

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 architecture makes each preceding layer $\mathbf{z}_j$ first produce its own output, 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 parameters, 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.

In the following sections, we will give the explicit implementation of HB-inspired architecture (16) and AGD-inspired architecture (18).

8 Experiments on Manual Architecture Design

8.1 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.

ImageNet We also test the validity of our models on ImageNet, which contains 1.2 million training images, 50,000 validation images, and 100,000 test images with 1,000 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 [10]. Concretely, the models are trained by stochastic gradient descent (SGD) with 0.9 Nesterov momentum and 10^{-4} weight decay. We adopt the

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

Model	top-1(%)	top-5(%)	Params (M)	FLOPs (G)
ResNet-34	26.73	8.74	21.80	3.66
HB-Net-34	26.33	8.56	21.80	3.66
ResNet-50	24.24	7.29	25.56	3.86
HBNet-50	23.93	7.06	25.56	3.87
ResNet-101	22.42	6.46	44.55	7.58
HBNet-101	21.91	6.07	44.55	7.59
DenseNet-121	25.02	7.71	7.98	2.88
AGD-Net-121	24.62	7.39	7.98	2.92

Table 6 Candidate operations.

	Candidate operations
1	3×3 separable conv
2	5×5 separable conv
3	3×3 dilated conv
4	5×5 dilated conv
5	3×3 max pooling
6	3×3 average pooling
7	Identity
8	Negative identity
9	Zero

weight initialization method in [80], and use the Xavier initialization [81] for fully connected layers. For CIFAR, we train all models for 300 epoches 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 all models for 100 eoches and drop learning rate at epoch 30, 60, and 90. The batchsize is 256 among 4 GPUs.

8.2 Comparison with State of The Arts

The experimental results on CIFAR are shown in Table 4, where the middle 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 single GPU. We can see that our proposed AGD-Net and HB-Net have a better performance than their corresponding baselines. 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 [9], 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 our HB-net, we can use exactly the same training strategy and get a converged performance with training only once. Therefore, the training procedure of HB-Net is more stable than the original ResNet when the network goes deeper.

As shown in Table 5, our proposed architectures are also effective on the ImageNet dataset. Both HB-Net and AGD-Net have better performances than their baselines. All the above experiments show that our optimization-inspired manual architecture design is very promising.

9 Experiments on Neural Architecture Search

Recently, DARTS has been widely used for NAS. But still, DARTS-based methods suffer from instability and the gap between the performance of the architecture in search and that in evaluation. In our work, we show that our optimization-inspired architectures can be used to improve the performance of DARTS. Specifically, we will show that HB-Net can inspire a search space that reduces the time consumption and can be a good initial search point to help search for a better architecture, while AGD-Net inspires us to exploit a general and better search space.

In DARTS, a standard convolutional cell consists of $N = 7$ nodes, among which the output node c_k is defined as the depthwise concatenation of all the intermediate nodes (input nodes excluded), i.e.,

$$c_k = [x_0, x_1, x_2, x_3], \tag{22}$$

where $[\cdot]$ denotes the concatenation. A network is then formed by stacking multiple cells together. The first and sencond nodes of cell k are set equal to the outputs of cell $k-2$ and cell $k-1$, i.e., c_{k-2} and c_{k-1}, respectively. 1×1 convolutions are inserted when necessary.

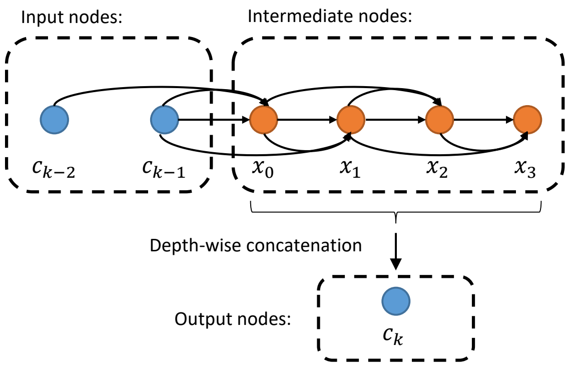

Figure 2 The diagram for the HB-inspired search space.

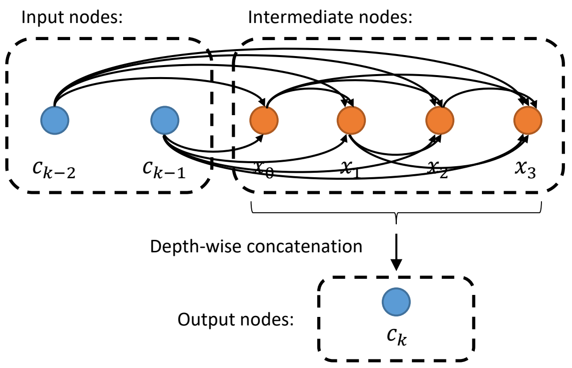

Figure 3 The diagram for the AGD-inspired search space.

Table 7 Experimental results on CIFAR-10 with our optimization-inspired initialization or search space and some competitive baselines in NAS, including DARTS. The mark † denotes using HBNet as initialization.

Methods	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC [10]	3.46	25.6	-	manual
NASNet-A + cutout [19]	2.65	3.3	1800	RL
ENAS + cutout [16]	2.89	4.6	0.5	RL
AmoebaNet-B +cutout [23]	2.55	2.8	3150	evolution
Hierarchical Evolution [22]	3.75	15.7	300	evolution
NASONet-WS [28]	3.53	3.1	0.4	NAO
DARTS (1st order) + cutout [29]	3.00	3.3	0.4	gradient-based
DARTS (2nd order) + cutout [29]	2.76	3.3	1	gradient-based
SNAS (moderate) + cutout [30]	2.85	2.8	1.5	gradient-based
PC-DARTS + cutout [32]	2.57	3.6	0.1	gradient-based
P-DARTS + cutout [31]	2.50	3.4	0.3	gradient-based
CNAS + cutout [59]	2.60	3.7	0.3	gradient-based
Search space-HB + cutout	2.82	2.92	0.3	gradient-based
Search space-HB† + cutout	2.68	3.35	0.3	gradient-based
Search space-AGD + cutout	2.56	3.3	0.4	gradient-based

9.1 Search Space and Initialization Inspired by HB-Net

We examine a simple search space which is consistent with HB-Net, where the intermediate node x_t is computed based on its two predecessors, and the specific expression is:

$$x_t = o_1^{(t-1,t)}(x_{t-1}) + o_2^{(t-1,t)}(x_{t-1}) + o^{(t-2,t)}(x_{t-2}), \tag{23}$$

where $o_1^{(t-1,t)}$ and $o_2^{(t-1,t)}$ denote the two operations applied to x_{t-1}, and $o^{(t-2,t)}$ denotes the operation applied to x_{t-2}. For ease of presentation, we denote c_{k-2} and c_{k-1} as x_{-2} and x_{-1} as well, respectively. The diagram for the HB-inspired search space is shown in Fig. 2.

We use $\mathcal{O}$ to denote a set of 9 candidate operations, each of which is an operation $o(\cdot)$ to be applied to x, and the candidate operations are listed in Table 6. To make the search space differentiable, the choice of a particular operation $o(\cdot)$ from node i to j is relaxed to a softmax over all possible operations:

$$\overline{o}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} w_o^{(i,j)} o(x), \tag{24}$$

where

$$w_o^{(i,j)} = \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} \exp(\alpha_{o'}^{(i,j)})}, \tag{25}$$

Table 8 Comparison between our performance on ImageNet using the architecture searched by Search space-AGD on CIFAR-10, and some competitive results in NAS, including DARTS, SNAS, and BayesNAS.

Methods	Top-1 Error (%)	Top-5 Error (%)	Params (M)	Search Method
Inception-v1 [5]	30.2	10.1	6.6	manual
MobileNet [82]	29.4	10.5	4.2	manual
ShuffleNet 2× [83]	25.1	-	∼5	manual
NASNet-A [19]	26.0	8.4	5.3	RL
MnasNet-92 [84]	25.2	8.0	5.5	RL
AmoebaNet-C [23]	24.3	7.6	6.4	evolution
DARTS (2nd order) [29]	26.7	8.7	4.7	gradient-based
SNAS [30]	27.3	9.2	4.3	gradient-based
BayesNAS [85]	26.5	8.9	3.9	gradient-based
Ours (Search space-AGD on CIFAR-10)	25.2	7.6	4.83	gradient-based

and the weights to mix operations for a pair of nodes (i,j) are parameterized by a vector $\boldsymbol{\alpha}^{(i,j)}$ of dimension $|\mathcal{O}|$. The task of architecture search then reduces to learning a set of continuous variables $\boldsymbol{\alpha}^{\mathrm{HB}} = \{\boldsymbol{\alpha}_1^{(t-1,t)}, \boldsymbol{\alpha}_2^{(t-1,t)}, \boldsymbol{\alpha}^{(t-2,t)}\}$, and we name this HB-inspired search space as Search space-HB in experiments.

Our optimization-inspired initialization particularly encodes the HB-Net into the architecture parameters $\boldsymbol{\alpha}$ at the beginning of architecture search. To be specific, we initialize the coefficient vectors $\boldsymbol{w}_2^{(t-1,t)}$ for $\bar{o}_2^{(t-1,t)}$ and $\boldsymbol{w}^{(t-2,t)}$ for $\bar{o}^{(t-2,t)}$ by constructing one-hot vectors. For $\bar{o}_2^{(t-1,t)}$, the coefficient for identity operation is 1; for $\bar{o}^{(t-2,t)}$, the coeficient for negative operation is 1. We initialize $\boldsymbol{w}_1^{(t-1,t)}$ for $\bar{o}_1^{(t-1,t)}$ using the softmax of a randomly initialized set of variables $\boldsymbol{\alpha}_{\mathrm{HBinit}} = \{\boldsymbol{\alpha}_{\mathrm{HBinit}}^{(t-1,t)}\}$. In this way, our optimization-inspired initialization leads to the following propagation:

$$x_t = F(x_{t-1}) + x_{t-1} - x_{t-2}, \tag{26}$$

where $F(\cdot)$ is a linear combination of 9 candidate operations listed in Table 6, and this is consistent with the HB-Net architecture. Noting that in Section 7.1, we specifically take $F(\cdot)$ as a composite function including two convolution layers, which restricts the HB-Net architecture. By contrast, the architecture given in (26) is more flexible and general.

As a comparable baseline, all the parameters are randomly initialized by $\boldsymbol{\alpha}_{\mathrm{init}} = \{\boldsymbol{\alpha}_{1,\mathrm{init}}^{(t-1,t)}, \boldsymbol{\alpha}_{2,\mathrm{init}}^{(t-1,t)}, \boldsymbol{\alpha}_{\mathrm{init}}^{(t-2,t)}\}$.

9.2 Search Space Inspired by AGD-Net

As for AGD-Net, each intermediate node is computed based on its all predecessors. Using DARTS, this kind of propagation is always simply described as:

$$x_t = \sum_{i<t} o^{(i,t)}(x_i). \tag{27}$$

Similarly, we use $\bar{o}(\cdot)$ defined in (24) to make the search space differentiable, and the search space becomes $\boldsymbol{\alpha}^{\mathrm{DARTS}} = \{\boldsymbol{\alpha}^{(i,j)}, i<j\}$.

However, different from the conventional DARTS, the intermediate node in (21) is actually calculated based on its predecessors with the linear weighted sum method, i.e.,

$$x_t = \sum_{i<t} p_{it} o^{(i,t)}(x_i), \tag{28}$$

where p_{it} are the weight coefficients. Particularly, some operations like identity and negative identity that are explicitly expressed in (21) are omitted because they are involved in the relaxed operation $\bar{o}(\cdot)$.

Considering that the number of the predecessors changes across iterations, which could cause undesired fluctuation in the resultant neural architecture, we introduce edge normalization to mitigate this problem.

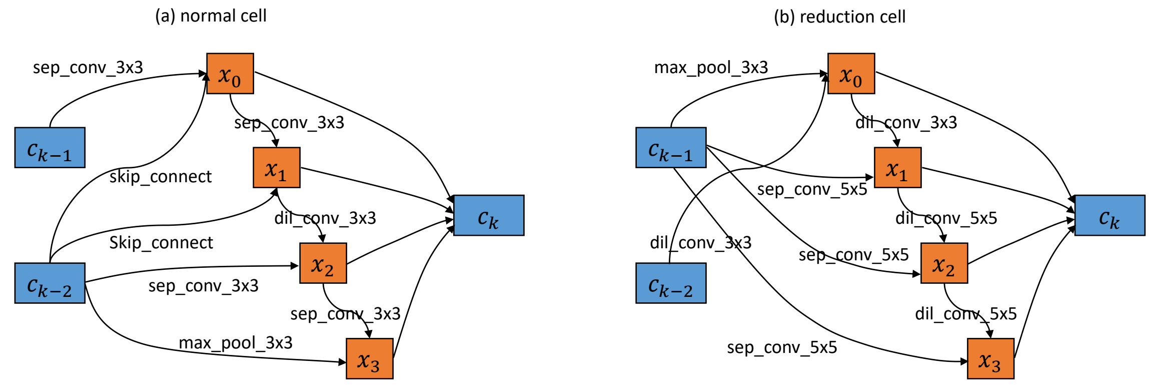

Figure 4 Our searched cells by Search space-AGD on CIFAR 10. (a): normal cell, (b): reduction cell.

Then, the coefficients p_{it} become path probabilites, and they are computed using a softmax function over paths to node t:

$$p_{it} = \frac{\exp(\beta_{it})}{\sum_{j<t} \exp(\beta_{jt})}, \tag{29}$$

and we denote the introduced parameters as $\boldsymbol{\beta}^{\mathrm{AGD}} = \{\beta_{ij}, i < j\}$. As a result, the task of architecture search is modified to learning a larger set of variables $\boldsymbol{\Theta}^{\mathrm{AGD}} = \{\boldsymbol{\alpha}^{\mathrm{DARTS}}, \boldsymbol{\beta}^{\mathrm{AGD}}\}$, which is inspired by AGD-Net. We name this AGD-inspired search space as Search space-AGD in experiments, and the diagram for the AGD-inspired search space is shown in Fig. 3.

9.3 Search Algorithm and Architecture Evaluation.

We evaluate our methods on CIFAR-10. To carry out architecture search, we randomly set 25,000 training samples as the training set and the other 25,000 training samples as the validation set. We denote the training and the validation loss as $\mathcal{L}_{\mathrm{train}}$ and $\mathcal{L}_{\mathrm{val}}$, respectively. We optimize the model weights by descending $\nabla_w \mathcal{L}_{\mathrm{train}}(\boldsymbol{w}, \boldsymbol{\alpha})$ or $\nabla_w \mathcal{L}_{\mathrm{train}}(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ on the training set, and optimize the architecture parameters by descending $\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\mathrm{val}}(\boldsymbol{w}, \boldsymbol{\alpha})$ or $\nabla_{\boldsymbol{\alpha},\boldsymbol{\beta}} \mathcal{L}_{\mathrm{val}}(\boldsymbol{w}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ on the validation set. We train a small network with 8 cells for 50 epochs, with batch size 64 (for both the training and validation sets) and the initial number of channels is 16. We use momentum SGD to optimize the weights $\boldsymbol{w}$, with initial learning rate $\eta_w = 0.025$ (annealed down to zero following a cosine schedule without restart [86]), momentum 0.9, and weight decay 3×10^{-4}. We use zero initialization for architecture variables $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, and use Adam [87] as the optimizer with initial learning rate $\eta_\alpha = \eta_\beta = 3 \times 10^{-4}$, momentum $(0.5, 0.999)$ and weight decay 10^{-3}.

When the searching is finished, for Search space-HB, we retain the top-3 strongest non-zero operations from x_{k-1} and x_{k-2} to x_k to derive the final architecture. For Search space-AGD, we retain the top-2 strongest non-zero operations from all previous nodes to x_k.

To evaluate the selected architecture, we randomly initialize its weights (weights learned during the search process are discarded), train it from scratch, and report its performance on the test set. Specifically, we train a large network of 20 cells for 600 epochs with batch size 96. The initial number of channels is increased from 16 to 36. Other hyperparameters remain the same as the ones used for architecture search. Additional enhancements include cutout [88], path dropout of probability 0.2, and auxiliary heads with weight 0.4. Note that the test set is never used for architecture search or architecture selection. The search cost and the test error are listed in Table 7.

Compared with DARTS, our method with HB-inspired search space takes less search cost (0.3 GPU days vs. 0.4 GPU days), because HB-inspired search space contains fewer edges (see Fig. 2). But still, our method results in a lower test error (2.82% vs. 3.00%). Furthermore, we use HB-inspired initialization and get a lower test error, 2.68%, which indicates that HB-Net can be a good initial point to help search for a better architecture. We use the Search space-AGD and get the cells shown in Fig. 4. Compared with its counterpart DARTS, our searched architecture performs better (2.56% vs. 3.00%) using nearly the same search cost. We transfer the architecture searched in the Search space-AGD from CIFAR-10 to ImageNet.

As shown in Table 8, our method performs significantly better than DARTS (25.2% vs. 26.7% for top-1 error and 7.6% vs. 8.7% for top-5 error), which indicates the superiority of the Search space-AGD.

Our results do not surpass P-DARTS [31], CNAS [59], and PC-DARTS [32] on ImageNet. Note that we just perform our method upon DARTS to test the effectiveness. Our method brings interpretability by offering optimization-inspired initial search point and search space, but does not change the search method, while P-DARTS, CNAS, and PC-DARTS propose better search methods to reduce the architectural gap, alleviate the space explosion problem, and improve the search efficiency, respectively. So it is easy to integrate our proposed interpretable initial search point and search space onto these improved search methods. They are compatible and are expected to further improve the search results.

10 Conclusion and Future Work

In this paper, we use the inspiration from optimization algorithms to design neural architectures. 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 neural network with the same linear transformation in different layers is equivalent to minimizing a cost function 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 neural architectures, where ResNet and DenseNet are two special cases of our case.

Note that our methodology is still preliminary and not conclusive as many engineering tricks may affect the performance of neural architectures greatly. Although we have shown the connection between faster optimization algorithms and better deep neural architectures, currently we haven't revealed the connection between optimization algorithm and manual architecture design or neural architecture search in a theoretically rigorous way. It is just an analogy. However, analogy does *not* mean unsolid or irrational. For example, DNN is inspired by brain. It is also an analogy and has no strict connections to brain either. But DNN has been proved to be very useful and effective in many areas. Accordingly, our optimization-inspired methodology can serve as a starting point to explain neural architectures. Practitioners can easily make changes to optimization-inspired architectures out of various insights and integrate various engineering tricks to produce even better results. Such a practice should be much easier and more explainable, than designing manually from scratch or searching without any guidance.

Acknowledgements Z. Lin was supported by the NSF China (No. 61731018), NSFC Tianyuan Fund for Mathematics (No. 12026606) and Project 2020BD006 supported by PKU-Baidu Fund.

References

1 Krzhevsky A, Sutshever I, Hinton G. ImageNet classification with deep convolutional neural networks. In: Proceedings of Advances in Neural Information Processing Systems, 2012

2 Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2015

3 Girshick R. Fast R-CNN. In: Proceedings of IEEE International Conference on Computer Vision, 2015

4 Ren S, Girshick R, He K, et al. Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell, 2016, 39: 1137-1149

5 Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2015

6 Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In: Proceedings of International Conference on Learning Representations, 2015

7 Szegedy C, Vanhoucke V, Ioffe S, et al. Rethingking the inception architecture for computer vison. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2016

8 Srivastava RK, Greff K, Schmidhuber J. Training very deep networks. In: Proceedings of Advances in Neural Information Processing Systems, 2015. 2377-2385

9 He K, Zhang X, Ren S, et al. Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2015

10 Huang G, Liu Z, Van der Maaten L, et al. Densely connected convolutional networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2017

11 Chen Y, Li J, Xiao H, et al. Dual path networks. In: Proceedings of Advances in Neural Information Processing Systems, 2017

12 Yang Y, Zhong Z, Shen T, et al. Convolutional neural networks with alternately updated clique. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, June 2018

13 Baker B, Gupta O, Naik N, et al. Designing neural network architectures using reinforcemen learning. In: Proceedings of International Conference on Learning Representations, 2017

14 Zoph B, Le Q. Neural architecture search with reinforcement learning. In: Proceedings of International Conference on Learning Representations, 2017

15 Liu C, Zoph B, Neumann M, et al. Progressive neural architecture search. In: Proceedings of European Conference on Computer Vision, September 2018

16 Pham H, Guan MY, Zoph B, et al. Efficient neural architecture search via parameter sharing. In: Proceedings of International Conference on Machine Learning, 2018

17 Zhong Z, Yan J, Wu W, et al. Practical block-wise neural network architecture generation. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, June 2018

18 Zhong Z, Yang Z, Deng B, et al. Blockqnn: efficient block-wise neural network architecture generation. IEEE Trans Pattern Anal Mach Intell, 2020

19 Zoph B, Vasudevan V, Shlens J, et al. Learning transferable architectures for scalable image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, June 2018

20 Cai H, Yang J, Zhang W, et al. Path-level network transformation for efficient architecture search. In: Proceedings of International Conference on Machine Learning, 2018. 678-687

21 Real E, Moore S, Selle A, et al. Large-scale evolution of image classifiers. In: Proceedings of International Conference on Machine Learning, 2017. 2902-2911

22 Liu H, Simonyan K, Vinyals O, et al. Hierarchical representations for efficient architecture search. In: Proceedings of International Conference on Learning Representations, 2018

23 Real E, Aggarwal A, Huang Y, et al. Regularized evolution for image classifier architecture search. In: Proceedings of AAAI Conference on Artificial Intelligence, 2019. 4780-4789

24 Elsken T, Metzen JH, Hutter F. Efficient multi-objective neural architecture search via lamarckian evolution. In: Proceedings of International Conference on Learning Representations, 2019

25 Lu Z, Whalen I, Boddeti V, et al. Nsga-net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, 2019. 419-427

26 Lu Z, Deb K, Goodman E, et al. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In: European Conference on Computer Vision, 2020. 35-51

27 Fang J, Chen Y, Zhang X, et al. EAT-NAS: elastic architecture transfer for accelerating large-scale neural architecture search. Sci China Inf Sci, 2021, 64: 1-13

28 Luo R, Tian F, Qin T, et al. Neural architecture optimization. In: Proceedings of Advances in Neural Information Processing Systems, 2018. 7816-7827

29 Liu H, Simonyan K, Yang Y. Darts: Differentiable architecture search. In: Proceedings of International Conference on Learning Representations, 2019

30 Xie S, Zheng H, Liu C, et al. Snas: stochastic neural architecture search. In: Proceedings of International Conference on Learning Representations, 2019

31 Chen X, Xie L, Wu J, et al. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019. 1294-1303

32 Xu Y, Xie L, Zhang X, et al. PC-DARTS: Partial channel connections for memory-efficient differentiable architecture search. In: Proceedings of International Conference on Learning Representations, 2020

33 Fang J, Sun Y, Zhang Q, et al. Densely connected search space for more flexible neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020. 10628-10637

34 Wu B, Dai X, Zhang P, et al. FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019. 10734-10742

35 E W. A proposal on machine learning via dynamical systems. Communications in Mathematics and Statistics, 2017, 5: 1-11

36 Haber E, Ruthotto L. Stable architectures for deep neural networks. Inverse Problems, 2017, 34: 014004

37 Chen TQ, Rubanova Y, Bettencourt J, et al. Neural ordinary differential equations. In: Proceedings of Advances in Neural Information Processing Systems, 2018. 6571-6583

38 Yang Y, Wu J, Li H, et al. Dynamical system inspired adaptive time stepping controller for residual network families. In: Proceedings of AAAI Conference on Artificial Intelligence, 2020

39 Gregor K, LeCun Y. Learning fast approximations of sparse coding. In: Proceedings of International Conference on Machine Learning, 2010

40 Xin B, Wang Y, Gao W, et al. Maximal sparsity with deep networks. In: Proceedings of Advances in Neural Information Processing Systems, 2016

41 Kulkarni K, Lohit S, Turaga P, et al. Reconnet: non-iterative reconstruction of images from compressively sensed mmeasuremets. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2016

42 Zhang J, Ghanem B. ISTA-Net: Iterative shrinkagethresholding algorithm inspired deep network for image compressie sensing. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2018

43 Yang Y, Sun J, Li H, et al. Deep admm-net for compressive sensing mri. In: Proceedings of Advances in Neural Information Processing Systems, 2016

44 Xie X, Wu J, Zhong Z, et al. Differentiable linearized admm, In: Proceedings of International Conference on Machine Learning, 2019

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

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

47 Verbancsics P, Harguess J. Generative neuroevolution for deep learning. arxiv:1312.5355, 2013

48 Saxena S, Verbeek J. Convolutional neural fabrics. In: Proceedings of Advances in Neural Information Processing Systems, 2016

49 Domhan T, Springenberg J, Hutter F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Proceedings of International Joint Conference on Artificial Intelligence, 2015

50 Bergstra J, Yamins D, Cox D. Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. In: Proceedings of International Conference on Machine Learning, 2013

51 Kwok T, Yeung D. Constructive algorithms for structure learning feedforward nerual networks for regression problems. IEEE Trans. on Neural Networks, 1997, 8: 630-645

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

53 Cortes C, Gonzalvo X, Kuznetsov V, et al. AdaNet: Adaptive structure learning of artificial nerual networks. In: Proceedings of International Conference on Machine Learning, 2017

54 Brock A, Lim T, Ritchie JM, et al. Smash: one-shot model architecture search through hypernetworks. In: Proceedings of

International Conference on Learning Representations, 2018
55 Cai H, Zhu L, Han S. Proxylessnas: direct neural architecture search on target task and hardware. In: Proceedings of International Conference on Learning Representations, 2019
56 Bender G, Kindermans PJ, Zoph B, et al. Understanding and simplifying one-shot architecture search. In: Proceedings of International Conference on Machine Learning, 2018. 550-559
57 Guo Z, Zhang X, Mu H, et al. Single path one-shot neural architecture search with uniform sampling. In: Proceedings of European Conference on Computer Vision, 2020. 544-560
58 Stamoulis D, Ding R, Wang D, et al. Single-path NAS: Designing hardware-efficient convnets in less than 4 hours. In: Proceedings of Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2019. 481-497
59 Guo Y, Chen Y, Zheng Y, et al. Breaking the curse of space explosion: Towards efficient nas with curriculum search. In: Proceedings of International Conference on Machine Learning, 2020. 3822-3831
60 Yang Y, You S, Li H, et al. Towards Improving the Consistency, Efficiency, and Flexibility of Differentiable Neural Architecture Search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021. 6667-6676
61 Yang Y, Li H, You S, et al. ISTA-NAS: Efficient and consistent neural architecture search by sparse coding. In: Proceedings of Advances in Neural Information Processing Systems, 2020. 10503-10513
62 Beck A, Teboulle M. A fast iterative shrinkage thresholding algorithm for linear inverse problems. SIAM J. Imaging Sciences, 2009, 2: 183-202
63 Sprechmann P, Bronstein AM, Sapiro G. Learning efficient sparse and low rank models. IEEE Trans Pattern Anal Mach Intell, 2015, 37: 1821-1833
64 Zhou JT, Di K, Du J, et al. Sc2net: sparse lstms for sparse coding. In: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018
65 Chen X, Liu J, Wang Z, et al. Theoretical linear convergence of unfolded ista and its practical weights and thresholds. In: Proceedings of Advances in Neural Information Processing Systems, 2018. 9061-9071
66 Metzler C, Mousavi A, Baraniuk R. Learned d-amp: principled neural network based compressive image recovery. In: Proceedings of Advances in Neural Information Processing Systems, 2017. 1772-1783
67 Li H, Yang Y, Chen D, et al. Optimization algorithm inspired deep neural network structure design. In: Proceedings of Asian Conference on Machine Learning, 2018. 614-629
68 Bertsekas D. Nonlinear Programming. Athena Scientific, Belmont, Ma, 1999
69 Polyak B. Ome methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 1964, 4: 1-17
70 Nesterov Y. A method for unconstrained convex minimization problem with the rate of convergence O(1=k2). Soviet Mathematics Doklady, 1983, 27: 372-376
71 Gabay D. Applications of the method of multipliers to variational inequalities. Studies in Mathematics and its applications, 1983, 15: 299-331
72 Lin Z, Liu R, Su Z. Linearized alternating direction method with adaptive penalty for low-rank representation. In: Proceedings of Advances in Neural Information Processing Systems, 2011
73 Putzky P, Welling M. Recurrent inference machines for solving inverse problems. arXiv:1706.04008, 2017
74 LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 1998, 86: 2278-2324
75 Jin X, Xu C, Feng J, et al. Deep learning with s-shaped rectified linear activation units. In: Proceedings of AAAI Conference on Artificial Intelligence, 2016
76 Zhao L, Li M, Meng D, et al. Deep convolutional neural networks with merge-and-run mappings. In: Proceedings of International Joint Conference on Artificial Intelligence, 2018
77 Lin M, Chen Q, Yan S. Network in network. In: Proceedings of International Conference on Learning Representations, 2014
78 Springenberg JT, Dosovitskiy A, Brox T, et al. Striving for simplicity: the all convolutional net. In: Proceedings of International Conference on Learning Representations Workshop, 2015
79 Lee CY, Xie S, Gallagher P, et al. Deeplysupervised nets. In: Proceedings of Artificial Intelligence and Statistics, 2015. 562-570
80 He K, Zhang X, Ren S, et al. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of IEEE International Conference on Computer Vision, 2015
81 Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of Artificial Intelligence and Statistics, 2010
82 Howard AG, Zhu M, Chen B, et al. Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861, 2017
83 Ma N, Zhang X, Zheng HT, et al. Shufflenet v2: practical guidelines for efficient cnn architecture design. In: Proceedings of European Conference on Computer Vision, 2018. 116-131.
84 Tan M, Chen B, Pang R, et al. Mnasnet: platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019. 2820-2828
85 Zhou H, Yang M, Wang J, et al. BayesNAS: a bayesian approach for neural architecture search. In: Proceedings of International Conference on Machine Learning, 2019. 7603-7613
86 Loshchilov I, Hutter F. Sgdr: stochastic gradient descent with warm restarts. In: Proceedings of International Conference on Learning Representations, 2017
87 Kingma DP, Ba J. Adam: a method for stochastic optimization. In: Proceedings of International Conference on Learning Representations, 2015
88 DeVries T, Taylor GW. Improved regularization of convolutional neural networks with cutout. arXiv:1708.04552, 2017