SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

June 2023, Vol. 66 162101:1–162101:16
https://doi.org/10.1007/s11432-021-3442-2

Newton design: designing CNNs with the family of Newton's methods

Zhengyang SHEN[1,2], Yibo YANG[3], Qi SHE[2], Changhu WANG[2], Jinwen MA[1*] & Zhouchen LIN[4,5*]

[1]*School of Mathematical Sciences, Peking University, Beijing* 100871, *China;*
[2]*Bytedance AI Lab, Haidian District, Beijing* 100871, *China;*
[3]*JD Explore Academy, Beijing* 100176, *China;*
[4]*Key Laboratory of Machine Perception, School of Intelligence Science and Technology, Peking University, Beijing* 100871, *China;*
[5]*Pazhou Lab, Guangzhou* 510320, *China*

Received 31 March 2021/Revised 4 August 2021/Accepted 26 January 2022/Published online 22 May 2023

Abstract Nowadays, convolutional neural networks (CNNs) have led the developments of machine learning. However, most CNN architectures are obtained by manual design, which is empirical, time-consuming, and non-transparent. In this paper, we aim at offering better insight into CNN models from the perspective of optimization theory. We propose a unified framework for understanding and designing CNN architectures with the family of Newton's methods, which is referred to as Newton design. Specifically, we observe that the standard feedforward CNN model (PlainNet) solves an optimization problem via a kind of quasi-Newton method. Interestingly, residual network (ResNet) can also be derived if we use a more general quasi-Newton method to solve this problem. Based on the above observations, we solve this problem via a better method, the Newton-conjugate-gradient (Newton-CG) method, which inspires Newton-CGNet. In the network design, we translate binary-value terms in the optimization schemes to dropout layers, so dropout modules naturally appear in the derived CNN structures with specific locations, rather than being an empirical training strategy. Extensive experiments on image classification and text categorization tasks verify that Newton-CGNets perform very competitively. Particularly, Newton-CGNets surpass their counterparts ResNets by over 4% on CIFAR-10 and over 10% on CIFAR-100, respectively.

Keywords CNN, dropout, optimization method, network design, Newton's method

Citation Shen Z Y, Yang Y B, She Q, et al. Newton design: designing CNNs with the family of Newton's methods. Sci China Inf Sci, 2023, 66(6): 162101, https://doi.org/10.1007/s11432-021-3442-2

1 Introduction

In recent years, convolutional neural networks (CNNs) have become the leading machine learning methods in several real-world application domains, e.g., image recognition [1–6] and text processing [7–10]. In all, the structure of a CNN model determines its performance, thus designing CNNs is a key problem. However, most CNN structures, such as ResNet [4] and DenseNet [5], are obtained by manual design, which is empirical, time-consuming, and lacking theoretical support.

In order to reduce the requirements for human expertise and labor, researchers are increasingly interested in designing neural networks automatically. One main strategy is network architecture search (NAS) [11–15], which searches for network architectures in a given search space. However, NAS uses a search strategy, and usually requires some extra computing power for search. In addition, these architectures are inherently obtained by learning from data, and still cannot provide any theoretical insight into the neural networks either.

Besides, there exist many studies [16–21] devoted to designing neural networks from theoretical derivation, such as optimization algorithms, which are much more transparent and interpretable compared with manual design and NAS. These studies are mainly focused on the sparse coding or compressive sensing

* Corresponding author (email: jwma@math.pku.edu.cn, zlin@pku.edu.cn)

(CS) problems, including signal/image recovery. Mathematically, the purposes of these problems are to infer the original signal x from its randomized measurements $y = \Phi x$, where Φ is a linear projection. Traditional methods for CS solve a well-defined problem, e.g., $\min_x \|\Phi x - y\|_2^2 + \lambda\|x\|_1$, where $\lambda\|x\|_1$ is the regularization, and employ iterative algorithms to solve it, e.g., the iterative shrinkage-thresholding algorithm (ISTA) [22], with iteration $x_{k+1} = \mathcal{T}_{\lambda t}(x_k - 2t\Phi^{\mathrm{T}}(\Phi x_k - y))$, where $\mathcal{T}_{\lambda t}$ is the soft-thresholding operator. Noting that this iteration resembles a network layer quite well when $\mathcal{T}_{\lambda t}$ is viewed as an activation function and Φ is made learnable. Zhang et al. [20] unfolded ISTA iterations and proposed ISTA-Net.

Generally, we need to point out that this CNN design methodology inspired by optimization algorithms is an important part in differential programming. A common practice is firstly using an iterative algorithm to solve a well-defined problem, and then mapping the iterations to a data flow graph, which may correspond to a deep neural network. After the network structure is obtained, the parameters can be made learnable to increase the capacity. However, this CNN design methodology is limited to the above-mentioned sparse coding or CS problems, and cannot be directly applied to more general applications, where neural networks are used to extract features, such as the image recognition task. This is mainly because it is difficult to establish a well-defined optimization problem for feature extraction like CS problems, let alone we wish the derived optimization iterations to resemble network layers in form. Some studies [23–25] addressed this issue by viewing the forward pass of CNN as sparse coding. However, these architectures cannot help understand some common CNN architectures, like ResNets, and have a high computational cost.

Li et al. [26] proposed another approach: they prove that computing with a standard feedforward neural network (PlainNet), when the weights are fixed and positive semi-definite, is equivalent to minimizing an objective function using the gradient descent algorithm, and assume that a better optimization algorithm may correspond to a better neural network architecture. With this new understanding, they use faster first-order optimization algorithms to minimize this objective function and design better neural network structures. However, the assumption that weight matrices are positive semi-definite is too strong, whereas we only need to assume weight matrices to be symmetric in this work.

Specifically, we observe that the PlainNet, when the weights are fixed and symmetric, can also be viewed as a kind of quasi-Newton method solving a well-defined optimization problem. Furthermore, we find that residual network (ResNet) can be derived by using an improved quasi-Newton method to solve this problem. Then, we utilize a better method, the Newton-conjugate-gradient (Newton-CG) method, to solve the problem, and propose Newton-CGNet, which contains branch structures and dropout modules naturally. In all, our theory proposes a unified framework for understanding and designing CNNs. Since that our theory understands some existing CNNs and designs new CNNs with the family of Newton's methods, we refer to it as Newton design.

We evaluate Newton-CGNets on both image classification and text categorization tasks. As for image classification, our models achieve lower classification error rates while using comparable numbers of parameters with the counterpart ResNets. Furthermore, the results are still competitive even compared with some advanced variants of ResNet. Without data augmentation, Newton-CGNets perform better than ResNets and its variants by a large margin. For text categorization, Newton-CGNets outperform VDCNNs [9] using fewer parameters, of which two versions exactly correspond to the counterparts PlainNets and ResNets.

Our contributions are as follows:

• We propose a unified framework for understanding and designing CNNs with the family of Newton's methods, which are mainly the second-order optimization methods. ResNet can be derived from our methodology.

• With our methodology, we translate binary-value terms in the optimization schemes to dropout layers. Then the specific locations of the dropout modules are naturally determined, rather than being positioned manually.

• Newton-CGNets perform very competitively on both image recognition and text processing applications.

2 Related work

There have been extensive studies on the neural network design. The main design strategies include manual design, NAS, and theoretical derivation, including optimization algorithms and ordinary differential equations (ODEs).

Manual design. The most common design strategy is manual design. As for image recognition, AlexNet [1] and VGG [2] achieved breakthrough results in the ImageNet classification challenge, with feedforward CNN structures. Also, many new neural network structures have been proposed, such as GoogLeNet [3], which contains several branches. ResNet [4] is the first ultra-deep CNN model, where skip connections are applied to avoid gradient vanishing. Moreover, Huang et al. [5] proposed DenseNet, where each layer connects to all latter layers, in order to improve the information flow. Nevertheless, manual design is empirical, imposing high demand for human skills, and always time-consuming.

NAS. In the early stage of neural network design, genetic algorithm [27, 28] based approaches were taken to find both architectures and weights. However, they perform worse than the hand-crafted ones [29]. Also, Domhan et al. [30] used Bayesian optimization for network architecture selection. First adopted in [11], reinforcement learning is the main mechanism to assign a better structure with a higher reward. Follow-up studies [12, 13] focused on reducing the search space and computational cost. But they are still time-consuming. Liu et al. [14] proposed differentiable architecture search (DARTS) and showed remarkable efficiency improvement. However, it is still unable to offer theoretical insight into the CNN architectures. In addition, NAS uses a search strategy and usually requires some computing power, while our method does not use any search strategy and computing power.

Derivation from optimization theory. CNNs derived by optimization algorithms are mainly for image restoration and reconstruction. The ISTA [22] is a popular method for CS. Most of the existing neural network based methods [16, 18] induced by ISTA have the feedforward structures. Particularly, Zhang and Ghanem [20] proposed ISTA-Net inspired by ISTA and FISTA-Net inspired by the fast iterative shrinkage-thresholding algorithm (FISTA). Interestingly, the acceleration in FISTA naturally leads to skip connections in the network design, and FISTA-Net outperforms ISTA-net in experiments, consistent with the performance of their related optimization methods. Besides, the alternating direction method of multipliers (ADMM) is an efficient algorithm for CS magnetic resonance imaging models. Sun et al. [19] defined the ADMM-Net over a data flow graph inspired by ADMM. In conclusion, all the studies mentioned here unfolded optimization iterations to final networks with the practice of differential programming.

Later, some studies have proposed the interpretations of deep networks as unrolling optimization algorithms. Papyan et al. [23] showed that the forward pass of the CNN is, in fact, the thresholding pursuit serving the multi-layer convolutional sparse coding model, and Sun et al. [24] proposed a supervised deep sparse coding network for image classification. However, it remains unclear why such low-level sparse coding is needed for the high-level classification task. Chan et al. [25] pointed out that for high-dimensional multi-class data, the optimal linear discriminative representation maximizes the coding rate difference between the whole dataset and the average of all the subsets, and proposed ReduNet, which is derived by using a gradient ascent scheme for optimizing the rate reduction objective. However, they should use a large batch size for training and have a high computational cost. Also, ReduNet performs much worse than common models, e.g., ResNets. The closest work to ours is [26], which viewed the PlainNet as the gradient descent algorithm minimizing an objective function. Then they designed better neural network structures induced by employing faster first-order optimization algorithms to solve this objective. However, the assumption on the weight matrices being positive semi-definite is too strong, and they only used first-order algorithms.

Derivation from ODE. The connection between neural networks and ODEs may be first observed by [31], where the forward propagation of ResNet can be seen as an Euler discretization of a continuous transformation. Lu et al. [32] proposed a linear multi-step architecture (LM-architecture) which is inspired by the linear multi-step method solving ODEs. Haber and Ruthotto [33] used this connection to analyze the stability and well-posedness of deep learning, and developed more stable network architectures. Furthermore, Chen et al. [34] introduced a continuous neural network. Instead of specifying a discrete sequence of hidden layers, they parameterized the derivative of the hidden state using a neural network. However, it cannot induce some operations naturally, such as dropout.

Table 1 Summary of notations in this paper

Notation	Description	Notation	Description
x_k	The output of the k-th layer	W_k	The weight matrix
A	The fixed weight matrix	Φ	The ReLU function
$\mathbb{S}^n$	The space of symmetric matrices	$\mathbb{R}^n$	n-dimensional Euclidean space
$\mathbb{S}^n_{++}$	The space of positive definite matrices	$P(x)$	$P'(x)=\Phi(x)$
A^{T}	The transpose of a matrix (vector)	H_k	The approximate inverse Hessian matrix
$\|\cdot\|_2$	The 2-norm	$\nabla F(x)$	The gradient of $F(x)$
$\nabla^2 F(x)$	The Hessian matrix of $F(x)$	Diag$[\cdot]$	The generated diagonal matrix
$\mathcal{P}$	The projection operator	$\mathcal{C}$	$\mathcal{C}=\{x\|x\succeq 0\}$
I	The identity matrix	U	$U=I-\mathrm{Diag}[\Phi'(Ax_k)]$
r	$r=\Phi(Ax_k)-x_k$	Q	$Q=U^{\mathrm{T}}U$
b	$b=U^{\mathrm{T}}r$	g_t	The gradient
d_t	The conjugate gradient	α_t,β_t	The scalars
N	The number of CG iterations (blocks)	L	The depth of the Newton-CGNet

3 Newton design

3.1 CNN as iterations of optimization

For differential programming, people may firstly use an iterative algorithm to solve a well-defined problem. Then they map the iterations into a data flow graph that may correspond to a neural network. Finally, the parameters in iterations can be made variable and learnable. However, for the image recognition task, we do not have a well-defined optimization problem in advance. Thus, we have to translate a known CNN structure to the optimization iterations solving an optimization problem firstly, in order to get a well-defined problem. All the notations in this paper are summarized in Table 1.

The most classic CNN structure is feedforward structures, such as AlexNet [1], which establishes the dominant status of CNNs in the computer vision field. Excluding the final softmax layer, the propagation from the first layer to the last layer, i.e., the process of extracting features (see Figure 1(a)), can be expressed as

$$x_{k+1}=\Phi(W_k x_k), \tag{1}$$

where x_k is the output of the k-th layer, Φ is an activation function and we set it as an ReLU. W_k is a linear transformation implemented by a convolution operation. We call model (1) PlainNet in this paper. Actually, many neural networks, which implement linear transformations using some special convolutions, can be naturally categorized into the PlainNets. For instance, VGG [2] uses 3×3 convolutions, while MobileNet [35] uses depthwise and pointwise convolutions. In this work, we focus on designing CNN architectures (i.e., the patterns of stacking convolutions) from the perspective of optimization theory, rather than the specific forms of convolutions. Thus, we uniformly denote the linear transformations as W_k without any distinction in the theoretical derivation.

Following [26], we fix the matrix W_k as A to simplify the analysis, and get the iteration

$$x_{k+1}=\Phi(Ax_k). \tag{2}$$

Furthermore, we have the following observations.

Proposition 1. If $A\in\mathbb{S}^n$, $x\in\mathbb{R}^n$, Φ is an ReLU, where $\mathbb{S}^n$ denotes the space of n-order symmetric matrices, then the iteration $x_{k+1}=\Phi(Ax_k)$ solves the optimization problem

$$\min_{x\succeq 0}\quad F(x)\equiv\frac{1}{2}x^{\mathrm{T}}Ax-1^{\mathrm{T}}P(Ax), \tag{3}$$

via a kind of quasi-Newton method (see the explanation in the Appendix A), where A^{-1} approximates the inverse Hessian of $F(x)$. $P'(x)=\Phi(x)$.

Proof.

$$\nabla F(x)=A[x-\Phi(Ax)], \tag{4}$$

where $\nabla F(x)$ is the gradient of $F(x)$, then

$$x_{k+1}=x_k-A^{-1}\nabla F(x_k)$$

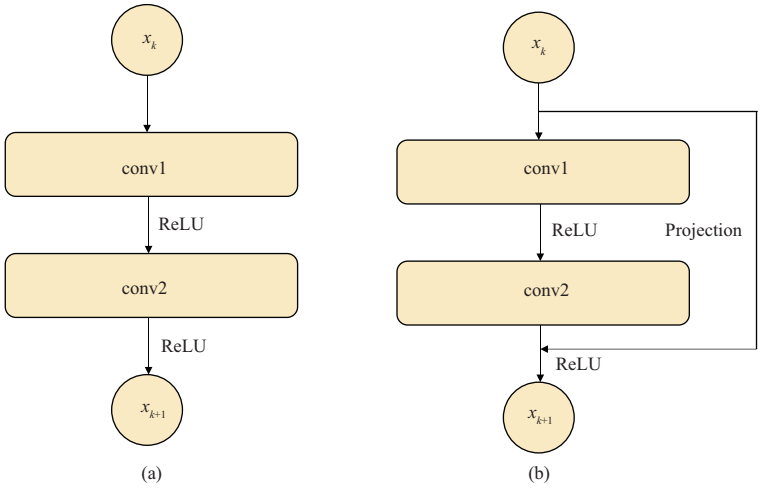

Figure 1 (Color online) (a) PlainNet; (b) Non-bottleneck ResNet. The block with two convolution operations is also called a non-bottleneck Residual Block in our paper.

$$\begin{aligned} &= x_k - A^{-1}[Ax_k - A\Phi(Ax_k)] \\ &= \Phi(Ax_k). \end{aligned} \tag{5}$$

Since Φ is an ReLU, $x_{k+1} \succeq 0$ is satisfied. Thus iteration (2) does solve the optimization problem (3) with a kind of quasi-Newton method, where A^{-1} approximates the inverse Hessian of $F(x)$.

$F(x)$ may not be the only objective function that the iteration (2) minimizes, but choosing $F(x)$ as (3) seems very natural in form.

To get better insight into our theory, we analyze the gap between A^{-1} and the inverse Hessian of $F(x)$. We assume $\|A\|_2 < 1$. Since

$$\nabla^2 F(x) = A - A\mathrm{Diag}[\Phi'(Ax)]A, \tag{6}$$

where $\nabla^2 F(x)$ is the Hessian matrix of $F(x)$, then

$$\begin{aligned} [\nabla^2 F(x)]^{-1} &= [I - \mathrm{Diag}[\Phi'(Ax)]A]^{-1}A^{-1} \\ &= \sum_{n=0}^{\infty} (\mathrm{Diag}[\Phi'(Ax)]A)^n A^{-1} \\ &= A^{-1} + \sum_{n=1}^{\infty} (\mathrm{Diag}[\Phi'(Ax)]A)^n A^{-1}. \end{aligned} \tag{7}$$

The Neumann series can be expanded because $\|\mathrm{Diag}[\Phi'(Ax)]A\|_2 \leqslant \|A\|_2 < 1$.

Obviously, the remaining term $\sum_{n=1}^{\infty}(\mathrm{Diag}[\Phi'(Ax)]A)^n A^{-1}$ cannot be neglected. On the other hand, the above quasi-Newton method only has a linear convergence rate (see the proof in the Appendix B), whereas a good quasi-Newton method may achieve a quadratic convergence rate. Thus A^{-1} is not a good enough approximation for the inverse Hessian. Instead, we can approximate the inverse Hessian by matrices H_k that change over iteration (e.g., $H_k = A^{-1} + \sum_{n=1}^{m}(\mathrm{Diag}[\Phi'(Ax_k)]A)^n A^{-1}$, where m is a given integer). As a result, the iteration scheme becomes

$$\begin{aligned} x_{k+1} &= \mathcal{P}_{\mathcal{C}}[x_k - H_k \nabla F(x_k)] \\ &= \Phi[x_k + H_k A(\Phi(Ax_k) - x_k)] \\ &= \Phi[(I - H_k A)x_k + H_k A\Phi(Ax_k)], \end{aligned} \tag{8}$$

where $\mathcal{P}$ is a projection operator and $\mathcal{C} = \{x | x \succeq 0\}$.

We can obtain the computation structure shown in Figure 1(b), which corresponds to the following iteration:

$$x_{k+1} = \Phi\left[W_s^{(k)} x_k + W_1^{(k)} \Phi\left(W_2^{(k)} x_k\right)\right]. \tag{9}$$

Eq. (9) is obtained by making the coefficient matrices in (8) learnable and variable. The structure in Figure 1(b) is non-bottleneck ResNet [4][1)].

So far, we have translated the PlainNet (1) to an optimization method solving the problem (3), building a bridge linking CNN models and optimization theory together. We use a more general quasi-Newton method to solve it, and derive ResNet. From this new understanding, we are able to design more promising and transparent CNN structures with optimization theory: optimize (3) with better optimization methods and then inspire better CNN structures[2)]. Our theory explains and designs CNNs with the family of Newton's methods, so we call it Newton design.

3.2 Newton-CG method

Observing the problem (3), we notice that the first term of $F(x)$, $x^{\mathrm{T}}Ax/2$, is a quadratic term. Particularly, the Newton's method is very suitable to solve a quadratic problem, which only takes one iteration to obtain the solution. Thus we speculate that Newton's method would optimize (3) better, and it will be verified in Subsection 4.1.3. The iteration scheme of the Newton's method is as follows:

$$\begin{aligned} x_{k+1} &= \mathcal{P}_{\mathcal{C}}\{x_k - [\nabla^2 F(x_k)]^{-1}\nabla F(x_k)\} \\ &= \Phi\{x_k + [I - \mathrm{Diag}\,[\Phi'(Ax_k)]\,A]^{-1}[\Phi(Ax_k) - x_k]\}. \end{aligned} \tag{10}$$

Noting that it is difficult to compute the inverse $[I-\mathrm{Diag}[\Phi'(Ax_k)]A]^{-1}$ directly, we adopt the conjugate gradient (CG) method to compute it indirectly. We denote $U = I-\mathrm{Diag}[\Phi'(Ax_k)]A$ and $r = \Phi(Ax_k)-x_k$. Then we just need to compute

$$y = U^{-1}r, \tag{11}$$

and y is the solution of the optimization problem

$$\min_y \ h(y), \tag{12}$$

where

$$h(y) = \frac{1}{2}(Uy-r)^{\mathrm{T}}(Uy-r) = \frac{1}{2}y^{\mathrm{T}}U^{\mathrm{T}}Uy - r^{\mathrm{T}}Uy + \frac{1}{2}r^{\mathrm{T}}r. \tag{13}$$

Again, we denote $Q = U^{\mathrm{T}}U$ and $b = U^{\mathrm{T}}r$, and the problem can be rewritten as

$$\min_y \ h(y) \equiv \frac{1}{2}y^{\mathrm{T}}Qy - b^{\mathrm{T}}y + \frac{1}{2}r^{\mathrm{T}}r. \tag{14}$$

We use the CG method to solve the problem. The procedure is shown in Algorithm 1, where g_t and d_t denote the gradient and the conjugate gradient, respectively.

Algorithm 1 Solving the optimization problem (14) via the CG method

Require: The parameters of the problem: Q and b; the number of iterations: N.
Ensure: The solution of problem (14), $y^\star$.
1: Select the initial point y_0;
2: $g_0 = \nabla h(y_0) = Qy_0 - b$;
3: set $d_0 = -g_0$;
4: **for** $t = 0, 1, \ldots, N-1$ **do**
5: $\quad \alpha_t = -\frac{g_t^{\mathrm{T}} d_t}{d_t^{\mathrm{T}} Q d_t}$;
6: $\quad y_{t+1} = y_t + \alpha_t d_t$;
7: $\quad g_{t+1} = \nabla h(y_{t+1}) = Qy_{t+1} - b$;
8: $\quad \beta_t = \frac{g_{t+1}^{\mathrm{T}} Q d_t}{d_t^{\mathrm{T}} Q d_t}$;
9: $\quad d_{t+1} = -g_{t+1} + \beta_t d_t$;
10: **end for**
11: **return** y_N.

Theoretically, the CG method needs at most n iterations to get the solution, where n is the dimension of the matrix Q. However, n is always very large, thus we always iterate N times ($N < n$) to approximate the solution (see line 4).

1) Although W_s is treated as an identity projection in most studies, the original work [4] showed that treating it as a learnable linear projection also works, and it is a more general form.

2) Li et al. [26] proposed the hypothesis that a better optimization algorithm may correspond to a better neural network architecture.

Table 2 Iteration numbers of using quasi-Newton methods and Newton-CG methods to solve the problem (3)

Method	Setting	Iteration
Quasi-Newton method	$m=0$	72
	$m=1$	36
	$m=2$	24
Newton-CG method	$N=3$	157
	$N=4$	36
	$N=6$	14.5
	$N=8$	9
	$N=10$	7

3.3 Numerical experiments

Before unfolding Newton-CG iterations to a CNN architecture, we show the numerical performance of the above-mentioned three different optimization methods solving the optimization problem (3).

We generate a positive definite matrix $\tilde{A}\in\mathbb{S}^n_{++}$ using $\tilde{A}=C^{\mathrm{T}}C$, where $C\in\mathbb{R}^{n\times n}$ have i.i.d. elements drawn from $\mathcal{N}(0,1)$. Then we get $A=\gamma\frac{\tilde{A}}{\|\tilde{A}\|_2}$, where $\gamma<1$. As a result, we obtain a positive definite matrix A and $\|A\|_2=\gamma<1$. On this condition, the zero vector $\mathbf{0}$ is a local minimum because $\nabla F(\mathbf{0})=\mathbf{0}$ and the Hessian matrix $\nabla^2F(\mathbf{0})=A\succ 0$. The initial point $x_0\in\mathbb{R}^n$ has i.i.d. elements drawn from $\mathcal{N}(0,1)$. We terminate iterations when $\|F(x_k)-F(\mathbf{0})\|<\epsilon$. We set $n=2048$, $\gamma=0.9999$, and $\epsilon=10^{-20}$.

As for the quasi-Newton method

$$x_{k+1}=\mathcal{P}_{\mathcal{C}}[x_k-H_k\nabla F(x_k)], \tag{15}$$

we artificially take the approximation of inverse Hessian $H_k=A^{-1}+\sum_{n=1}^{m}(\mathrm{Diag}[\Phi'(Ax_k)]A)^nA^{-1}$, where m is a given integer. Then, the optimization method

$$x_{k+1}=x_k-A^{-1}\nabla F(x_k) \tag{16}$$

is equivalent to the case that we set $m=0$.

As for the Newton-CG method, we simply use x_k to initialize the initial point y_0 in Algorithm 1. Among all the iterative algorithms, we run 10 times and report the median iteration numbers. The results are listed in Table 2.

The results show that the iterative algorithm (16) takes 72 iterations to solve the given optimization problem. And when we use matrices H_k, which change over iteration, to approximate the inverse Hessian, it takes fewer iterations. Concretely, quasi-Newton methods take 36 and 24 iterations when $m=1$ and 2, respectively.

When it comes to the Newton-CG method, we find that when $N=3$, it needs 157 iterations, which is far more than that of quasi-Newton methods. This phenomenon is mainly because that if CG method only iterates a few times, it cannot obtain a good enough solution to approximate the inverse Hessian, consequently, Newton-CG method performs badly. When we increase N, we find that Newton-CG methods perform significantly better than quasi-Newton methods: Newton-CG methods take only 14.5, 9, and 7 iterations when $N=6,8$, and 10, respectively.

It seems that quasi-Newton methods will also perform better if we further increase m. However, it is impossible to increase m in the derived ResNet artificially, because the approximate inverse Hessian H_k in (8) is obtained by learning, rather than being calculated using a given expression. By contrast, it is very easy to increase N in our derived Newton-CGNet, which will be introduced in the next subsection. This makes a big difference between our derived CNNs and ResNets.

3.4 Unfolding Newton-CG iterations to Newton-CGNet

We have shown how to use a better iterative algorithm, Newton-CG method, to optimize (3), and now we will unfold the iterations to a CNN architecture. Meanwhile, we need to increase the model capacity with differential programming. Firstly, we consider α_t and β_t in lines 5 and 8 of Algorithm 1. Here, we treat them as two learnable scalars, which will be tuned according to the loss function, rather than calculate them with the given expressions. Obviously, it will enhance the expressive power of the network. Besides, we simply use x_k to initialize the initial point y_0 (see line 1 of Algorithm 1).

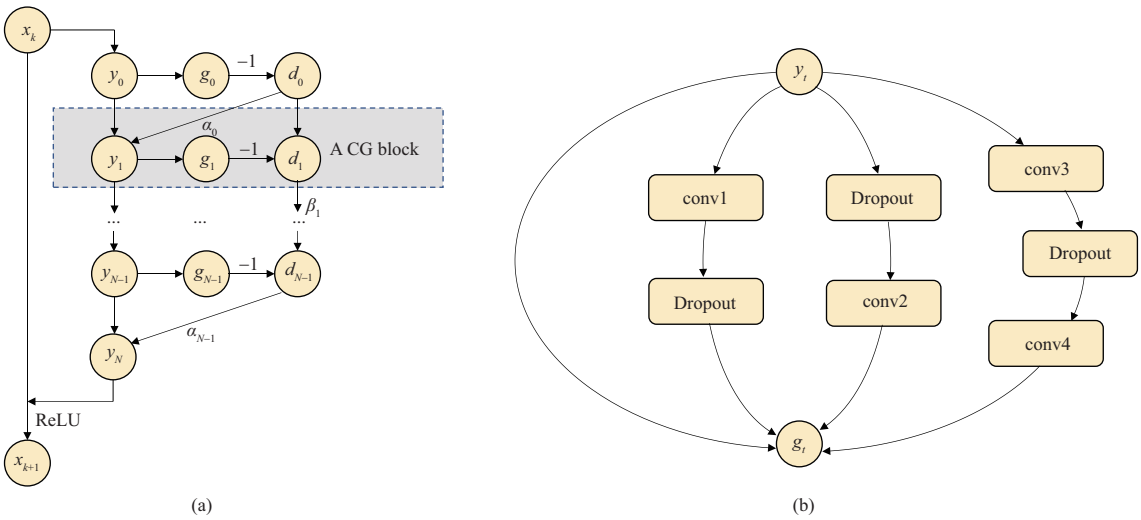

Figure 2 (Color online) (a) A Newton-CG Block. It is the forward propagation from x_k to x_{k+1}, related to one step of the Newton's method. The grey region is a CG Block, related to one step of the CG method. The forward propagation from y_t to g_t is in (b). (b) The forward propagation from y_t to g_t. It is the implementation of (17) that computes the gradient, with some parameters made learnable.

In addition, computing the gradient is an important operation (see lines 2 and 7 of Algorithm 1). Concretely,

$$\begin{aligned} g &= Qy - b = U^{\mathrm{T}}Uy - b \\ &= [I - A\mathrm{Diag}[\Phi'(Ax_k)]][I - \mathrm{Diag}[\Phi'(Ax_k)]A]y - b \\ &= y - \mathrm{Diag}[\Phi'(Ax_k)]Ay - A\mathrm{Diag}[\Phi'(Ax_k)]y + A\mathrm{Diag}[\Phi'(Ax_k)]Ay - b. \end{aligned} \tag{17}$$

Naturally, the forms like Ay can be viewed as convolution operations and the last term b can be simply viewed as a threshold. As for $\mathrm{Diag}[\Phi'(Ax_k)]$, since Φ is an ReLU, $\Phi'(Ax_k)$ is a binary-valued function, taking 0 or 1 as its values. Moreover, since $\mathrm{Diag}[\Phi'(Ax_k)] \cdot y = \Phi'(Ax_k) \odot y$, where $\odot$ is a Hadamard product, we can view the term $\mathrm{Diag}[\Phi'(Ax_k)] \cdot y$ as a binary mask, which will be further translated to a dropout layer in implementation, as will discussed in Subsection 4.1.2.

Based on the above discussion, we are able to map the Newton-CG iterations to a data flow graph. Then we make the parameters in the data flow graph learnable and variable and get Newton-CGNet. For ease of presentation, we call the forward propagation from x_k to x_{k+1}, related to one step of Newton's method, a Newton-CG Block. Also, we call the forward propagation related to one step of CG method a CG Block. The forward propagation of a Newton-CG Block is shown in Figure 2, where a CG Block is denoted in the grey region. In all, a Newton-CGNet is stacked by multiple Newton-CG Blocks, and each Newton-CG Block is consisted of multiple CG Blocks. Formally, the forward propagation of a Newton-CG Block (from x_k to x_{k+1}) is shown in Algorithm 2.

As shown in Algorithm 2, we implement (17) in lines 3–6, and this schema is shown in Figure 2(b). Particularly, as the convolution kernels are learnable, we replace some "–" in (17) with "+" in line 6 for ease of presentation. We use "*" to denote the convolution operation. The total learnable parameters in a Newton-CG Block are $\Theta_k = \{W_t^{(1)}, W_t^{(2)}, W_t^{(3)}, W_t^{(4)}, 0 \leqslant t \leqslant N-1; \alpha_0, \alpha_1, \ldots, \alpha_{N-1}; \beta_0, \beta_1, \ldots, \beta_{N-2}\}$. Each CG Block is 2-layer deep using 4 convolution kernels. Correspondingly, each Newton-CG block is $2N$-layer deep using $4N$ convolution kernels.

4 Experiments

We evaluate our models on both image classification and text categorization tasks.

Algorithm 2 The forward propagation of a Newton-CG Block (from x_k to x_{k+1})

Require: The input: x_k; the number of CG Blocks: N; the convolution kernels: $W_t^{(1)}, W_t^{(2)}, W_t^{(3)}$, and $W_t^{(4)}, 0 \leqslant t \leqslant N-1$; the scalars: $\alpha_0, \alpha_1, \ldots, \alpha_{N-1}, \beta_0, \beta_1, \ldots, \beta_{N-2}$;

$y_0 = x_k$;
for $t = 0, 1, \ldots, N-1$ **do**
  $g_t^{(1)} = \mathrm{Dropout}(W_t^{(1)} * y_t)$;
  $g_t^{(2)} = W_t^{(2)} * \mathrm{Dropout}(y_t)$;
  $g_t^{(3)} = W_t^{(4)} * \mathrm{Dropout}(W_t^{(3)} * y_t)$;
  $g_t = y_t + g_t^{(1)} + g_t^{(2)} + g_t^{(3)}$;
  if $t = 0$ **then**
    $d_t = -g_t$;
  else
    $d_t = -g_t + \beta_{t-1} d_{t-1}$;
  end if
  $y_{t+1} = y_t + \alpha_t d_t$;
end for
$x_{k+1} = \mathrm{ReLU}(x_k + y_N)$;
return x_{k+1}.

4.1 Image classification

4.1.1 *Datasets*

CIFAR. The two CIFAR datasets [36] consist of colored natural images with 32×32 pixels. CIFAR-10 (C10) consists of images drawn from 10 classes and CIFAR-100 (C100) from 100. The training and the test set contain 50000 and 10000 images, respectively, and we randomly hold out 5000 training images as a validation set. We select the model with the lowest validation error during training. We adopt a standard data augmentation scheme (mirroring/shifting) [37] that is widely used for these two datasets. We denote this augmentation scheme by a "+" mark at the end of the dataset name (e.g., C10+). For preprocessing, we normalize the images using the channal means and standard deviations. We report the test errors.

SVHN. The street view house numbers (SVHN) dataset [38] contains 32×32 colored digit images. There are 73257 images in the training set, 26032 images in the test set, and 531131 images for additional training. Following common practice [37,39], we use all the training data without any data augmentation, and a validation set with 6000 images is randomly split from the training set. We select the model with the lowest validation error during training. We follow [40] and divide the pixel values by 255 so they are in the $[0, 1]$ range. We report the test errors.

ImageNet. We also conduct experiments on ILSVRC 2012 dataset [41], which contains 1.2 million training images, 50000 validation images, and 100000 test images with 1000 classes. Following [4, 42], we adopt the standard data augmentation for the training sets. A 224×224 crop is randomly sampled from the images or its horizonal flip. Following [40], we report the top-1 and top-5 single-crop error rates on the validation set.

4.1.2 *Architectures and training details for CIFAR and SVHN*

Dropout. Firstly, we show the necessity that we should translate the binary mask $\mathrm{Diag}\Phi'(Ax_k) \cdot y$ to a dropout layer. As for the forward propagation, simply treating this term as a parameterized binary mask does not affect inference. However, for the backward propagation, since $\Phi'(Ax_k)$ is a binary-valued function, the gradient w.r.t. the parameters A is always zero, so that A is difficult to update. Consequently, it does not make sense to simply translate $\mathrm{Diag}[\Phi'(Ax_k)] \cdot y$ to a learnable binary mask.

From another point of view, during the training phase with stochastic gradient descent (SGD), the input x_k is random, so $\mathrm{Diag}[\Phi'(Ax_k)]$ is also a random binary mask, which resembles the dropout module quite well. Thus in implementation, we alternatively translate the term $\mathrm{Diag}[\Phi'(Ax_k)]$ to a dropout layer for ease of training.

In order not to cause misunderstanding, we have to emphasize that the term $\mathrm{Diag}[\Phi'(Ax_k)]$ is not exactly the same as a dropout layer, because this term is inherently determined by A and x_k, whereas the dropout is a completely random binary mask. In all, we provide a novel approach to deal with the binary masks derived in the differential programming field, rather than derive the dropout modules theoretically. One advantage of our methodology is in that dropout can naturally be introduced into CNN structures by analogy, and have specific locations from the theoretical derivation. By contrast, Zagoruyko et al. [40]

also added dropout modules to Wide ResNet at similar locations. However, their strategy is empirical.

Architectures. In order to faciliate the analysis and comparison, we design the architecture of Newton-CGNets based on ResNets. As shown in Figure 2, the Newton-CGNet contains branch structures. Thus with the same depth, it contains twice more convolution kernels than the ResNet. Naturally, we modify two Residual Blocks $\left[\begin{smallmatrix}\text{conv1}\\ \text{conv2}\end{smallmatrix}\right] \times 2$ to one CG Block $\left[\text{conv1}; \text{conv2}; \begin{smallmatrix}\text{conv3}\\ \text{conv4}\end{smallmatrix}\right]$ (corresponding to the topology in Figure 2(b)).

Generally, we use L-$\{N_1, N_2, N_3\}$ to denote the Newton-CGNet architecture which contains 3 Newton-CG Blocks with L-layer depth totally. And each Newton-CG Block contains N_1, N_2, and N_3 CG Blocks, respectively.

Training details. All the models are trained using SGD and a Nesterov momentum [43] of 0.9 without dampening. During the training phase, we find that our models can converge stably when we adopt common settings used in existing CNNs. Specifically, we adopt the weight initialization method in [44] for convolutional layer and use Xavier initialization [45] for the fully connected layer. On CIFAR and SVHN we train our models using batch size 128 for 300 and 40 epochs, weight decay of 5×10^{-4} and 10^{-4}, respectively. The initial learning rate is set to 0.1 and is divided by 10 at 50% and 75% of the total number of training epochs. We add an ReLU after conv3 of each CG Block to supplement some nonlinearity[3)]. We adopt batch normalization (BN) [46] after each convolution kernel. Following [44], we perform a linear projection to match the dimensions for addition operation whenever in need, with a 1×1 convolution kernel. We use 0.2 dropout rate on C10+ and C100+, 0.3 dropout rate on SVHN, and 0.4 dropout rate on C10 and C100, respectively. Since the dropout module is an integral part of the Newton-CGNet, rather than merely a training strategy, we can adopt dropout fairly. All the learnable scalars are initialized as 1.0. We report the median of 5 runs.

4.1.3 *Newton-CGNets vs. ResNets*

As we have shown in Table 2, compared with quasi-Newton methods, Newton-CG methods perform worse than quasi-Newton methods when the interior CG methods iterate only a few times and perform better when the CG methods iterate enough times. Naturally, it is interesting to explore how their derived CNNs perform. In fact, the number of the CG Blocks in each Newton-CG Block relates to the number of the CG iterations, thus we explore the performance of Newton-CGNets via changing the number of the CG Blocks.

We now evaluate our models on C10+. We take L-$\{N_1, N_2, N_3\}$ as 10-$\{1, 1, 2\}$, 16-$\{2, 2, 3\}$, 22-$\{3, 3, 4\}$, 28-$\{4, 4, 5\}$, 56-$\{9, 9, 9\}$, and 82-$\{13, 13, 14\}$, and get Newton-CGNet-10, 16, 22, 28, 56, and 82, respectively. On one TITAN Xp GPU, these models take 9, 15, 21, 26, 52, and 75 s for training for one epoch, 1, 2, 2.5, 3, 7, and 11 s for inference, respectively. For fair comparison, we compare the performance of Newton-CGNets with its counterpart ResNets using comparable numbers of parameters and also run ResNets for 300 epochs. The results are listed in Table 3. The error rates resulted from ResNets are slightly better than that reported in [4], due to more training epochs.

We plot the results in Figure 3(a) and observe that when using a few CG Blocks ($N_3 \leqslant 5$), Newton-CGNets perform worse than its counterpart ResNets. And when we use more CG Blocks ($N_3 \geqslant 9$), Newton-CGNets perform better. To be specific, with comparable numbers of parameters, Newton-CGNet-56 and 82 surpass ResNet-110 and 164 by 0.38% and 0.67%, respectively. This phenomenon is very similar to that shown in Table 2. To conclude, as for this image recognition task, iterative algorithms (quasi-Newton methods and Newton-CG methods) and their derived CNN models (ResNets and Newton-CGNets) show the similar pattern: the better an iterative algorithm approximates the inverse Hessian, the better the iterative algorithm solves the optimization problem, also, the better its derived CNN model performs.

In addition, we investigate the sensitivity of some important hyperparameters for model training, including learning rate and weight decay, based on Newton-CGNet-56. As shown in Figure 4, when the learning rate is 0.1, our model performs well when the weight decay is between 10^{-4} and 10^{-3}. When the weight decay is 5×10^{-4}, our model performs well when the learning rate is between 0.2 and 0.02. To conclude, our model can perform stably when the weight decay and learning rate are around 5×10^{-4} and 0.1, respectively.

3) This modification is necessary, because a Newton-CG Block is nearly a linear model otherwise. We make the modification as minor as possible in order to maintain the derived scheme.

Table 3 Test error rates (%) on C10+ using ResNets and Newton-CGNets (median of 5 runs)

Method	Depth	Params	Error (%)	Method	Depth	Params	Error (%)
ResNet	20	0.27M	7.67	Newton-CGNet	10	0.29M	9.25
	32	0.46M	6.83		16	0.48M	7.21
	44	0.66M	6.31		22	0.68M	6.44
	56	0.85M	6.02		28	0.87M	6.14
	110	1.73M	5.73		56	1.70M	5.35
	164	2.62M	5.57		82	2.62M	4.90

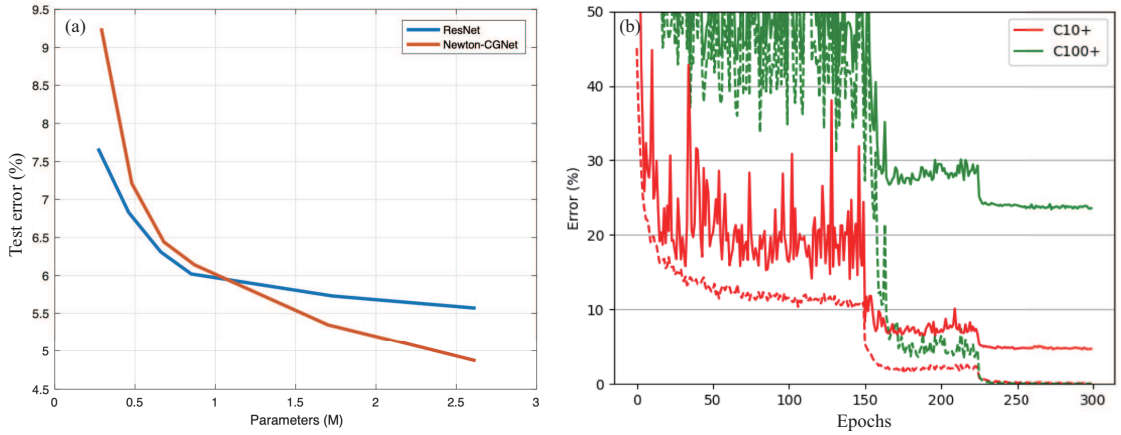

Figure 3 (Color onine) (a) Newton-CGNets perform worse than ResNets using a few CG Blocks, and outperform ResNets using more CG Blocks; (b) training curves on C10+ and C100+. Dashed lines denote training errors, and solid lines denote test errors.

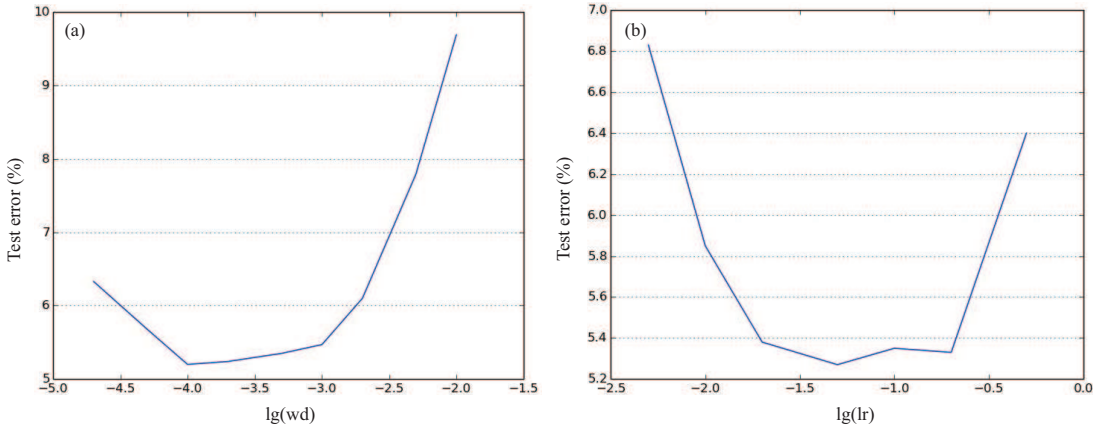

Figure 4 (Color onine) (a) Test error rates of Newton-CGNet-56 with the learning rate of 0.1 and different weight decay (wd); (b) test error rates of Newton-CGNet-56 with the weight decay of 5×10^{-4} and different learning rates (lrs).

Actually, since that the performance of a CNN model is dependent on multiple factors, such as datasets and training strategies, it is difficult to accurately predict how the derived CNN models perform. However, Newton design provides an optimization perspective to help analyze and explain the performance of the derived CNN models qualitatively, and then guide us to use the derived CNNs more efficiently, e.g., using enough CG Blocks. By contrast, we cannot analyze the CNN models obtained by manual design or NAS in this way.

4.1.4 *Newton-CGNets vs. competitive models*

Using enough CG Blocks, we compare our derived Newton-CGNets with some more competitive models on three datasets, C10, C100, and SVHN, respectively. The test error rates are listed in Table 4.

Newton-CGNets vs. advanced variants of ResNets. On the dataset with data augmentation, Newton-CGNet-82 results in 4.90% on C10+ and 23.87% on C100+, outperforming its counterpart ResNet-164 by 0.67% on C10+ and 2.74% on C100+, respectively. And the results are at least compara-

Table 4 Test error rates (%) on CIFAR and SVHN datasets (median of 5 runs)[a)]

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
ResNet [4]	110	1.73M	12.06	5.73	41.83	26.93	1.80
	164	2.62M	11.68	5.57	38.65	26.61	1.75
ResNet with stochastic depth [39]	110	1.7M	11.66	5.23	37.80	24.58	1.75
Wide ResNet with dropout [40]	16	2.7M	–	5.24	–	23.91	1.64
ResNet (pre-activation) [42]	164	2.62M	11.26	5.46	35.58	24.33	–
MobileNet [35]	15	3.26M	8.70	6.89	28.96	27.33	1.77
FractalNet with dropout/drop-path [47]	21	38.6M	7.33	**4.60**	28.20	23.73	1.87
HB-Net [26]	110	1.7M	8.66	5.04	36.4	23.93	–
ReduNet [26]	18	11.5M	–	7.00	–	–	–
Newton-CGNet (ours)	56	1.7M	7.06	5.35	28.23	24.55	1.65
	82	2.62M	**6.80**	4.90	**27.06**	23.87	**1.57**
	110	3.45M	–	4.66	–	**23.44**	–

a) The best results are highlighted in bold.

ble with all the listed variants of ResNet, including Wide ResNet [40], ResNet with stochastic depth [39] or pre-activation [42]. We furthermore increase the number of CG Blocks and obtain a CNN architecture over 100-layer deep. Concretely, we take L-$\{N_1, N_2, N_3\}$ as 110-$\{18, 18, 18\}$ and get Newton-CGNet-110. The behaviors of Newton-CGNet-110 are shown in Figure 3(b), indicating that this model can be optimized without difficulty.

On the dataset without data augmentation, our models perform even better. To be specific, Newton-CGNet-82 results in 6.80% on C10, 27.06% on C100, and 1.57% on SVHN, significantly surpassing its counterpart ResNet-164 by 4.88% on C10, 11.59% on C100, and 0.18% on SVHN, with comparable numbers of parameters, respectively. In addition, our models outperform all the listed variants of ResNet significantly.

Newton-CGNets vs. MobileNets. Inherently, the MobileNet [35] can be naturally categorized into PlainNets, where the linear transformation is implemented using much more efficient depth-wise separable convolutions. Considering that the reported MobileNet architecture uses 5 downsampling layers (convolutions of stride 2) to process the input size of 224×224, whereas the images in CIFAR and SVHN are only of size 32×32, directly employing that setting will result in very low resolution (1×1) after the last convolution. So we remove the first three dowmsampling and preserve the last two, in consistent with the setting of Newton-CGNets for fair comparison. As shown in Table 4, Newton-CGNet-82 perform better than MobileNet on all tasks using fewer parameters (2.62M vs. 3.26M), even though we only employ conventional convolutions, which shows great superiority of our architecture.

Newton-CGNets vs. FractalNets. FractalNet [47] employs fractal architecture instead of residual connections to build DNNs, and achieve much more competitive results. Compared with it, Newton-CGNets perform comparably when using data augmentation (4.66% vs. 4.60% on C10+ and 23.44% vs. 23.73% on C100+), and better without data augmentation (6.80% vs. 7.33% on C10, 27.06% vs. 28.20% on C100, and 1.57% vs. 1.87% on SVHN), using less than 10% parameters, which indicates great parameter efficiency of our method.

Newton-CGNets vs. other optimization-inspired networks. Our method significantly outperforms ReduNet (4.66 vs. 7.00 on C10+), which is designed by solving a rate reduction objective. Also, ReduNet needs to use a large batch size (about 1000), so it has a much higher computational cost. Compared with HB-Nets, our models achieve comparable results on C10+ and C100+, and perform much better on C10 and C100. Noting that HB-Nets are essentially inspired by a first-order optimization method, while ours are by a second-order method, the better performance also indicates that a faster optimization method would help design a better network architecture.

4.1.5 *Training details and results for ImageNet*

Also, we get the Newton-CGNet architecture for ImageNet by modifying ResNet. Particularly, the ResNets with 50, 101, and 152 layers are stacked by multiple "bottleneck" building blocks, different from the non-bottleneck residual blocks derived in our paper (see Figure 1(b)). As for ResNet-18, each group of convolution kernels only contains 2 residual blocks. As discussed in Subsection 4.1.3, Newton-CGNets with a few CG Blocks do not perform well. Thus it dose not make sense to modify ResNet-18 to the

Table 5 The top-1 and top-5 single-crop error rates (%) on the validation set of ImageNet dataset (median of 5 runs)

Method	Depth	Params	Top-1 error (%)	Top-5 error (%)
Wide ResNet	18	25.9M	27.06	9.00
ResNet	34	21.8M	26.73	8.65
Newton-CGNet	34	21.7M	25.98	8.23

Table 6 Large-scale text categorization datasets used in our experiments

Dataset	Train	Test	Classes	Average words	Categorization task
AG news	120k	7.6k	4	45	English news categorization
Sogou news	450k	60k	5	578	Chinese news categorization
DBPedia	560k	70k	14	55	Ontology classification
Yelp review polarity	560k	38k	2	153	Sentiment analysis
Yelp review full	650k	50k	5	155	Sentiment analysis
Yahoo! answers	1400k	60k	10	112	Topic classification
Amazon review full	3000k	650k	5	93	Sentiment analysis
Amazon review polarity	3600k	400k	2	91	Sentiment analysis

Newton-CGNet. Consequently, we choose ResNet-34 as our basic model.

For ResNet-34, the numbers of the residual blocks with different output sizes are $3, 4, 6$, and 3, respectively. In order not to get too shallow Newton-CG blocks, we only modify the third group of convolution kernels to a Newton-CG block, with the other parts unchanged. In addition, in order to utilize enough CG blocks without introducing more parameters, we specifically reduce the parameters in each CG block. Concretely, the residual blocks $\left[\begin{smallmatrix}3\times3,256\\3\times3,256\end{smallmatrix}\right]\times6$ are modified to the CG blocks $\left[3\times3,64;3\times3,64;\begin{smallmatrix}3\times3,256\\3\times3,128\end{smallmatrix}\right]\times6$, composing a Newton-CG block. Correspondingly, the method of computing g_t is modified to

$$g_t = y_t + [g_t^{(1)}, g_t^{(2)}, g_t^{(3)}], \tag{18}$$

where $[g_t^{(1)}, g_t^{(2)}, g_t^{(3)}]$ refers to the concatenation of the feature-maps produced in three branches. It is not contradictory to the derivation in lines 3–6 of Algorithm 2, because this equals to the case that some channels of $W_t^{(1)}, W_t^{(2)}$, and $W_t^{(4)}$ are fixed to be zeros.

We initialize the learning rate as 0.1, with the batch size of 256. We set the dropout rate as 0.2. For using dropout, we train our model for 100 epochs and drop the learning rate by 0.1 at epoch $30, 60$, and 90. The other training details are the same as that for CIFAR and SVHN. We report the median of 5 runs, and the results are shown in Table 5. The top-1 and top-5 single-crop error rates resulted from Newton-CGNet-34 are 25.98% and 8.23%. With comparable numbers of parameters and the same depth, Newton-CGNet-34 surpasses ResNet-34 by 0.75% and 0.42% for top-1 and top-5 single-crop error rates, respectively. This indicates that our proposed models can also be applied on large datasets. In future work, we will study how to modify the bottleneck structure to our Newton-CG block.

4.2 Text categorization

As for text processing, we evaluate our Newton-CGNet on 8 freely available large-scale datasets introduced by [8] which cover several text categorization tasks, including English/Chinese news categorization, ontology classification, sentiment analysis and topic classification. The number of training examples varies from 120k to 3.6M, and the number of classes is comprised between 2 and 14. The more detailed description is listed in Table 6.

VDCNN [9] is a representative model that utilizes deep CNNs (over 6 layers) for text processing. All processing is done at the character level which is the atomic representation of a sentence, same as pixels for images. Particularly, two versions of VDCNN (with and without short connections) can be naturally categorized into ResNets and PlainNets, where the linear transformation is implemented using temporal convolutions with kernel size 3. Because PlainNets and ResNets are exactly involved in our theoretical framework, we set VDCNN as the baseline.

For fair comparison with VDCNN, we also process texts at the character level, and design the architecture of Newton-CGNet based on VDCNN. To be specific, our model begins with a look-up table that generates a 2D tensor of size (f_0, s) that contain the embeddings of the s characters, where s is fixed to 1024. We first apply one layer of 64 convolutions of size 3, followed by a stack of temporal Newton-CG

Table 7 Test error rates (%) on the 8 datasets (median of 5 runs)[a)]

Method	Depth	Params	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
VDCNN (without shortcut) [9]	20	9.1M	8.88	3.54	1.4	4.5	36.07	27.51	37.39	4.41
	32	9.5M	8.73	3.36	1.29	4.28	35.74	26.57	37.00	4.31
VDCNN (with shortcut) [9]	20	9.1M	8.47	4.09	1.27	4.62	37.07	27.65	38.11	4.38
	32	9.5M	8.26	3.96	1.26	4.43	36.57	27.36	37.86	4.22
Newton-CGNet (ours)	12	8.2M	**7.87**	**3.31**	**0.98**	**4.11**	**35.41**	**26.28**	**36.94**	**3.92**

a) The best results are highlighted in bold. All the models use max pooling for downsampling between convolutional layers.

blocks for the feature maps of different resolutions. The networks contain 3 max pooling operations (halving the temporal resolution each time by 2), resulting in 3 levels of 128, 256 and 512 feature maps. The output of the final Newton-CG block is downsampled using k-max pooling, and then the resulting features are feed into a three layer fully connected (FC) classifier with softmax outputs. The number of hidden units is set to 2048, and k to 8 in all experiments. We use temporal batch normalization to regularize our network.

We train our models using SGD and a Nesterov momentum of 0.9 without dampening for 100 epochs, with a batch size of 128, and a weight decay of 5×10^{-5}. The initial learning rate is set to 0.1 and is divided by 10 at 50% and 75% of the total number of training epochs. The dropout rate is set to 0.1.

In experiments, we observe that only setting one CG Block for each resolution of feature maps is enough to result in very competitive results, showing great superiority of our method. Specifically, as shown in Table 7, Newton-CGNets outperform two versions of VDCNN on all the listed categorization tasks yet use fewer parameters (8.2M vs. 9.1M+). That a 12-layer deep model (9 convolutional layers + 3 FC layers) is enough is because that it has been very deep for the text categorization task.

5 Conclusion

In this work, we have proposed a unified framework for understanding and designing CNNs with the family of Newton's methods, which is referred to as Newton design. The core of our theory is using better methods to solve a given optimization problem, and then design CNNs with the iterative algorithms. Extensive experiments on image classification and text categorization have shown the superiority of our method. In addition, our theory generates dropout layers naturally, enriching the diversity of inspired CNNs.

In our work, we only design the Newton-CGNet with the Newton-CG method. Actually, there are many kinds of Newton's methods in optimization theory. For example, we can use a rank one correction formula, DFP or BFGS algorithm to approximate the inverse Hessian, and adopt damped Newton's method instead of Newton's method. In future work, we plan to explore more CNN structures with Newton design.

Actually, this optimization theory based design methodology can be naturally extended to other applications with various objectives, as it is easy to derive their iterative algorithms. However, the difficulty is in that most derived iterations do not resemble network layers quite well like the iterations of CS problems, thus it is difficult to unfold them to neural networks. This is the main limitation of optimization theory based design and accounts for why most related works are focused on CS problems. We will explore more possibilities in the future.

Acknowledgements This work was supported by National Key R&D Program of China (Grant No. 2022ZD0160302), Major Key Project of PCL, China (Grant No. PCL2021A12), and National Natural Science Foundation of China (Grant No. 62276004).

Supporting information Appendixes A and B. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems, 2012. 1097–1105

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

3 Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2015. 1–9

4 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, 2016. 770–778
5 Huang G, Liu Z, Maaten L, et al. Densely connected convolutional networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2017. 4700–4708
6 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, 2018. 2413–2422
7 Kim Y. Convolutional neural networks for sentence classification. In: Proceedings of Conference on Empirical Methods in Natural Language Processing, 2014. 1746–1751
8 Zhang X, Zhao J B, LeCun Y. Character-level convolutional networks for text classification. InProceedings of the 28th International Conference on Neural Information Processing Systems, 2015. 649–657
9 Conneau A, Schwenk H, Barrault L, et al. Very deep convolutional networks for text classification. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, 2017. 1107–1116
10 Johnson R, Zhang T. Deep pyramid convolutional neural networks for text categorization. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, 2017. 562–570
11 Baker B, Gupta O, Naik N, et al. Designing neural network architectures using reinforcement learning. In: Proceedings of International Conference on Learning Representations, 2017
12 Liu C, Zoph B, Neumann M, et al. Progressive neural architecture search. In: Proceedings of European Conference on Computer Vision, 2018. 19–34
13 Pham H, Guan M Y, Zoph B, et al. Efficient neural architecture search via parameter sharing. In: Proceedings of the 35th International Conference on Machine Learning, 2018. 4095–4104
14 Liu H, Simonyan K, Yang Y. Darts: differentiable architecture search. In: Proceedings of International Conference on Learning Representations, 2018
15 Chen X, Xie L X, Wu J, et al. Progressive differentiable architecture search: bridging the depth gap between search and evaluation. In: Proceedings of IEEE/CVF International Conference on Computer Vision, 2019. 1294–1303
16 Gregor K, LeCun Y. Learning fast approximations of sparse coding. In: Proceedings of the 27th International Conference on International Conference on Machine Learning, 2010. 399–406
17 Liu Q, Wang J. A one-layer projection neural network for nonsmooth optimization subject to linear equalities and bound constraints. IEEE Trans Neural Netw Learn Syst, 2013, 24: 812–824
18 Xin B, Wang Y, Gao W, et al. Maximal sparsity with deep networks? In: Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016. 4340–4348
19 Yang Y, Sun J, Li H B, et al. Deep ADMM-Net for compressive sensing MRI. In: Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016. 10–18
20 Zhang J, Ghanem B. ISTA-Net: interpretable optimization-inspired deep network for image compressive sensing. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2018. 1828–1837
21 Giryes R, Eldar Y C, Bronstein A M, et al. Tradeoffs between convergence speed and reconstruction accuracy in inverse problems. IEEE Trans Signal Process, 2018, 66: 1676–1690
22 Beck A, Teboulle M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM J Imag Sci, 2009, 2: 183–202
23 Papyan V, Romano Y, Elad M. Convolutional neural networks analyzed via convolutional sparse coding. J Mach Learn Res, 2017, 18: 2887–2938
24 Sun X, Nasrabadi N M, Tran T D. Supervised deep sparse coding networks for image classification. IEEE Trans Image Process, 2020, 29: 405–418
25 Chan K H R, Yu Y, You C, et al. ReduNet: a white-box deep network from the principle of maximizing rate reduction. 2021. ArXiv:2105.10446
26 Li H, Yang Y, Chen D, et al. Optimization algorithm inspired deep neural network structure design. In: Proceedings of the 10th Asian Conference on Machine Learning, 2018. 614–629
27 Schaffer J D, Whitley D, Eshelman L J. 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. 1–37
28 Leung F H F, Lam H K, Ling S H, et al. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. IEEE Trans Neural Netw, 2003, 14: 79–88
29 Verbancsics P, Harguess J. Generative neuroevolution for deep learning. 2013. ArXiv:1312.5355
30 Domhan T, Springenberg J T, Hutter F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Proceedings of the 24th International Conference on Artificial Intelligence, 2015
31 E W. A proposal on machine learning via dynamical systems. Commun Math Stat, 2017, 5: 1–11
32 Lu Y, Zhong A, Li Q, et al. Beyond finite layer neural networks: bridging deep architectures and numerical differential equations. In: Proceedings of the 35th International Conference on Machine Learning, 2018. 3276–3285
33 Haber E, Ruthotto L. Stable architectures for deep neural networks. Inverse Probl, 2018, 34: 014004
34 Chen T Q, Rubanova Y, Bettencourt J, et al. Neural ordinary differential equations. In: Proceedings of the 32nd Conference on Neural Information Processing System, 2018. 6571–6583
35 Howard A G, Zhu M, Chen B, et al. Mobilenets: efficient convolutional neural networks for mobile vision applications. 2017. ArXiv:1704.04861
36 Krizhevsky A, Hinton G. Learning Multiple Layers of Features from Tiny Images. Technical Report, 2009
37 Lee C Y, Xie S, Gallagher P, et al. Deeply-supervised nets. In: Proceedings of the 18th International Conference on Artificial

Intelligence and Statistics, 2015. 562–570
38 Netzer Y, Wang T, Coates A, et al. Reading digits in natural images with unsupervised feature learning. In: Proceedings of Conference on Neural Information Processing Systems, 2011
39 Huang G, Sun Y, Liu Z, et al. Deep networks with stochastic depth. In: Proceedings of European Conference on Computer Vision, 2016. 646–661
40 Zagoruyko S, Komodakis N. Wide residual networks. 2016. ArXiv:1605.07146
41 Deng J, Dong W, Socher R, et al. Imagenet: a large-scale hierarchical image database. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2009. 248–255
42 He K, Zhang X, Ren S, et al. Identity mappings in deep residual networks. In: Proceedings of European Conference on Computer Vision, 2016. 630–645
43 Sutskever I, Martens J, Dahl G, et al. On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on International Conference on Machine Learning, 2013. 1139–1147
44 He K, Zhang X, Ren S, et al. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of IEEE/CVF International Conference on Computer Vision, 2015. 1026–1034
45 Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the 13th International Conference on Artificial Intelligence and Statistic, 2010. 249–256
46 Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. 2015. ArXiv:1502.03167
47 Larsson G, Maire M, Shakhnarovich G. Fractalnet: ultra-deep neural networks without residuals. In: Proceedings of International Conference on Learning Representations, 2017