SCIENCE CHINA
Information Sciences

• **Supplementary File** •

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*

Appendix A

The objective functions $f(\mathbf{z})$ for most widely adopted non-linear activations are listed in A1.

Table A1 The optimization objectives for most non-linear activations.

	Non-linear activation	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	$\frac{x}{1+e^{-x}}$	$\frac{\|\mathbf{x}\|^2}{2}-\sum_i\left[\frac{(\mathbf{U}_i^T\mathbf{x})^2}{2}+\mathbf{U}_i^T\mathbf{x}\log\left(\frac{1}{e^{\mathbf{U}_i^T\mathbf{x}}}+1\right)-\text{polylog}\left(2,-\frac{1}{e^{\mathbf{U}_i^T\mathbf{x}}}\right)\right]$

Appendix B Conjecture: Faster Optimization Algorithm May Imply Better Network

Given a dataset $\{\{\mathbf{z}_0^i, l_i\} : i = 1, \cdots, m\}$, where $\mathbf{z}_0^i$ is the i-th data and l_i is its label, we assume that $\mathbf{z}_0^i$ and $\mathbf{f}_i$ have the same dimension. In this section we use $\{\mathbf{z}_0, \mathbf{f}\}$ instead of $\{\mathbf{z}_0^i, \mathbf{f}_i\}$ for simplicity.

Appendix B.1 Same Linear Transformation in Different Layers

We first assume that the simplified neural network model has the same linear transformation $\mathbf{Wz}$ in different layers. Actually, this corresponds to the recurrent neural networks [73]. As discussed in Section 4, the propagation in a deep neural network can be deemed as an optimization process that minimizes a cost function $F(\mathbf{z})$ by the gradient descent algorithm.

When we use the GD algorithm to minimize $F(\mathbf{z})$ with initializer $\mathbf{z}_0$, the iterative procedure is equivalent to (7). Let $\hat{\mathbf{f}}$ be the output feature of this feedforward neural network. It is known that the GD algorithm takes $O\left(\frac{L}{\mu}\log\frac{1}{\epsilon}\right)$ iterations to attain 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 turn to a better optimization algorithm, e.g., the HB algorithm and the AGD algorithm, their iterative procedures are equivalent to (10) and (12), respectively. By doing so, these algorithms take $O\left(\sqrt{\frac{L}{\mu}}\log\frac{1}{\epsilon}\right)$ iterations to satisfy $\|\hat{\mathbf{f}}-\mathbf{f}\|\leqslant\epsilon$. Therefore,

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

Table B1 MSE comparisons of different optimization algorithm-inspired neural architectures.

depth	ADMM (13)	GD (7)	HB (10)	AGD (11)	AGD2 (12)
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

the network architectures corresponding to faster algorithms (also characterized by the same $\mathbf{U}$ but has different architectures) require fewer layers than the feedforward neural network discussed above.

Generally, the network architecture with fewer layers to reach the same approximation accuracy can be regarded as the better one for architecture selection.

Appendix B.2 Different Linear Transformation in Different Layers

Requiring the linear transformation is the same for different layers is not practical and is only introduced for theoretical analyses. Now we consider the case where each layer has a different transformation.

For a network with finite layers, we have $\|\mathbf{f} - \mathrm{Net}_{\mathbf{U}}(\mathbf{z}_0)\| \leqslant \epsilon$, where $\mathrm{Net}_{\mathbf{U}}(\mathbf{z}_0)$ denotes the final output. Relaxing the parameter $\mathbf{U}$ to be different in different layers, we can solve the following optimization problem to learn $\mathbf{U}_1, \cdots, \mathbf{U}_n$ with a fixed neural architecture:

$$\min_{\mathbf{U}_1,\cdots,\mathbf{U}_n} \|\mathbf{f} - \mathrm{Net}_{\mathbf{U}_1,\cdots,\mathbf{U}_n}(\mathbf{z}_0)\|^2. \tag{B1}$$

We then have that $\|\mathbf{f} - \mathrm{Net}_{\mathbf{U}_1^*,\cdots,\mathbf{U}_n^*}(\mathbf{z}_0)\| \leqslant \|\mathbf{f} - \mathrm{Net}_{\mathbf{U}}(\mathbf{z}_0)\|$, where $\mathbf{U}_1^*, \cdots, \mathbf{U}_n^*$ refer to the optimal parameters.

Appendix B.3 Simulation Experiment

We conduct toy experiments to verify our proposed conjecture that the neural architecture inspired by the faster optimization algorithm is the better one.

We compare the five neural architectures inspired by the GD, HB, two variants of AGD, and ADMM algorithms that correspond to the operations in (7), (10), (11), (12), and (13). We use sigmoid as the non-linear activation Φ and perform $\mathbf{Wz}$ as a fully-connected layer. We set β as 0.3 for (10). Other parameters in (11) and (12) are exactly the same as their corresponding optimization algorithms. We solve the problem (B1) to optimize the parameters of each layer. 10,000 pairs of $\{\mathbf{z}_0^i, \mathbf{f}_i\}$ are randomly generated from the Gaussian distribution $N(\mathbf{0}, \mathbf{I})$ as the training data. The dimension of both $\mathbf{z}_0^i$ and $\mathbf{f}_i$ is 100. We train the five models with different depths for 1,000 epoches and then report the Mean Squared Error (MSE)[1)] loss values.

As shown Table B1, the architectures inspired by the HB, AGD, and AGD2 algorithms perform better than the architecture inspired by the GD algorithm. This is in line with the fact that the theoretical convergence rates of both HB and AGD algorithms are better than that of GD algorithm. The architecture inspired by ADMM performs the worst. Actually, although ADMM has been popular to solve a wide range of optimization problems, it does not have a faster theoretical convergence rate than GD. It is also observed that as the depth increases the MSEs of GD, HB, and AGD-inspired architectures do not always decrease, which means that the deeper GD, HB, and AGD-inspired architectures are harder to train. As a comparison, the AGD2-inspired architecture does suffer from this dilemma. The reason may be that the AGD2 optimization algorithm has a better numerical stability, despite the fact that AGD2 is theoretically equivalent to AGD without considering numerical error.

Appendix C Engineering Implementation

In this section, we introduce the neural architectures inspired by optimization algorithms (7), (10), (11), (12), and (13) for practical implementations.

Relax Φ and $\mathbf{W}$. We relax $\mathbf{Wz}$ from the fully connected layer to convolution, which is also a linear transformation. For practical implementations, We allow $\mathbf{W}$ in different layers to be different parameters and may not be square matrices, so the input and output dimensions of any layer can be different. We further allow the non-linear activation Φ to be relaxed to pooling and batch normalization (BN) operations. So $\Phi(\cdot)$ can be a composite function of nonlinear activation, pooling, BN, convolution, or fully-connected layer. With different combinations of these operations, we see that the neural architecture (7) actually covers many popular CNN architectures, such as LeNet [74] and VGG [6]. The non-linear activation can also be learnable, as adopted in [75].

We replace $\Phi(\mathbf{Wz})$ with the notation $T(\mathbf{z})$ for simplicity.

Adaptive Coefficients. The coefficients in optimization algorithms are usually fixed. In practical implementations, we allow these coefficients for neural architectures to be learnable. Specifically, we rewrite (10) as:

$$\mathbf{z}_{k+1} = T(\mathbf{z}_k) + \beta_1 \mathbf{z}_k + \beta_2 \mathbf{z}_{k-1}, \tag{C1}$$

where β_1 and β_2 can be any constants or jointly optimized with the network parameters. When they equal to 0, it is equivalent to dropping the corresponding paths.

The architecture of (C1) is shown in Figure C1(a), from which we can see that $\mathbf{z}_{k+1}$ is a combination of $T(\mathbf{z}_k)$, $\mathbf{z}_k$, and $\mathbf{z}_{k-1}$.

Now we rewrite (11) and (12) as:

$$\mathbf{z}_{k+1} = T(\beta_1 \mathbf{z}_k + \beta_2 \mathbf{z}_{k-1}), \tag{C2}$$

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

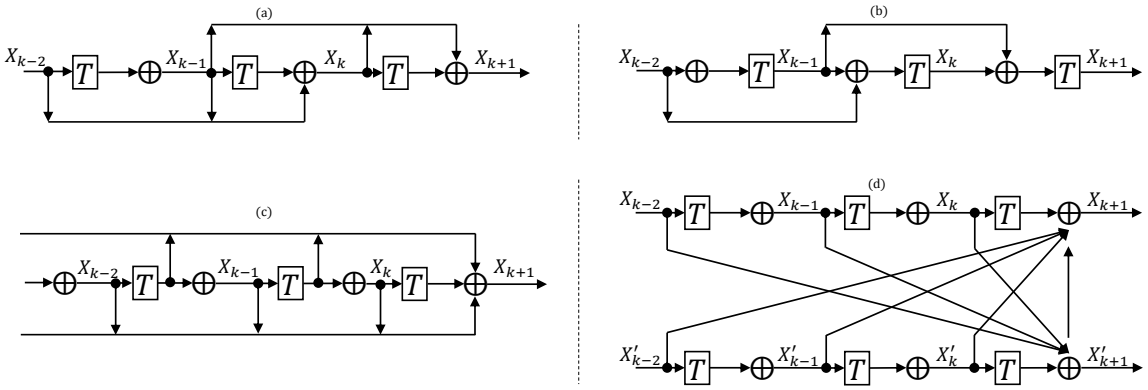

Figure C1 (Figure 1 in [67]) Illustrations of neural architectures (C1), (C2), (C3) and (C5) in (a), (b), (c) and (d), respectively.

Table C1 Optimization algorithms and their inspired neural architectures.

Algorithm	Neural Architecture	Transforming Setting
GD (1)	CNN	$\mathbf{Wz}$ →convolution
HB (2)	ResNet [9]	$\beta_2 = 0$ in (C1)
AGD (4)	DenseNet [10]	$\beta = 0$, $\alpha = 1$ in (C3)
ADMM (6)	DMRNet [76]	$\alpha_k = \beta_k = \frac{1}{2}$ in (C5)

and

$$\mathbf{z}_{k+1} = \sum_{j=0}^{k} \alpha_{k+1}^{j} T(\mathbf{z}_j) + \sum_{j=0}^{k} \beta_{k+1}^{j} \mathbf{z}_j. \tag{C3}$$

All the coefficients α and β can be any constants or jointly optimized with the network parameters.

We show the architectures of (C2) and (C3) in Figure C1(b) and C1(c). As shown in Figure C1(b), it first makes a combination of $\mathbf{z}_k$ and $\mathbf{z}_{k-1}$, and then the operator T follows. In Figure C1(c), $\mathbf{z}_{k+1}$ is the summation of all of $T(\mathbf{z}_1), \cdots, T(\mathbf{z}_k)$ and $\mathbf{z}_1, \cdots, \mathbf{z}_k$.

The popular neural architecture, residual network (ResNet), computes each layer by adding a skip connection on a composite function:

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

Actually we can recover the ResNet architecture from (C1) by setting $\beta_2 = 0$ and $\beta_1 = 1$.

Densely connected network (DenseNet) is another popular architecture that connects different layers by concatenation. It takes the following form:

$$\mathbf{z}_{k+1} = T([\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_k]), \tag{C4}$$

where [ ] denotes the concatenation. We can recover the DenseNet architecture from (C3) by setting $\beta_k^j = 0, \forall k, j$.

We further rewrite (13) as:

$$\begin{aligned} \mathbf{z}'_{k+1} =& T(\mathbf{z}'_k) + \sum_{t=1}^{k} \alpha_t \mathbf{z}'_t + \sum_{t=1}^{k} \beta_t \mathbf{z}_t, \\ \mathbf{z}_{k+1} =& T(\mathbf{z}_k) + \sum_{t=1}^{k} \alpha_t \mathbf{z}'_t + \sum_{t=1}^{k} \beta_t \mathbf{z}_t. \end{aligned} \tag{C5}$$

The architecture of (C5) is illustrated in Figure C1(d), from which we can see that the two paths interact with each other. It is shown that the ADMM-inspired architecture has two parallel paths. Note that we can even recover the two-path network architecture, DMRNet [76], which has the following form:

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

We summarize the optimization algorithms and their corresponding existing neural architectures in Table C1.

Block Based Architecture. To enable features with different resolutions in a network, we decompose the whole architecture into multiple blocks. Neighboring blocks are connected by down-sampling to reduce the resolution. The formulations of (C1), (C2), and (C3) are only valid in each block.

Appendix C.1 HB-Inspired Architecture (HBNet)

We introduce the HB-inspired architecture (HBNet) by directly setting $\beta_1 = 1$ and $\beta_2 = -1$ in (C1):

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

where T is a two-layer residual branch. The two layers in T are the same as the ResNet design.

Appendix C.2 AGD-Inspired Architecture (AGDNet)

In line with our analyses, we then introduce the AGD-inspired architecture (AGDNet) from (C3):,

$$\mathbf{z}_{k+1} = \sum_{j=0}^{k} \alpha_{k+1}^{j} T(\mathbf{z}_j) + \beta \left(\mathbf{z}_k - \sum_{j=0}^{k} h_{k+1}^{j} \mathbf{z}_j \right), \tag{C6}$$

where T is the same composite function as that in DenseNet. The coefficients α_{k+1}^{j} are joint optimized with the network parameters. The coefficients h_{k+1}^{j} are produced by (5). We set β as 0.1 in our experiments.

Appendix D 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 weight initialization method in [81], and use the Xavier initialization [80] 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.