PID Controller based Stochastic Optimization Acceleration for Deep Neural Networks

Haoqian Wang, Yi Luo, Wangpeng An, Qingyun Sun,
Jun Xu, Yongbing Zhang, Yulun Zhang, and Lei Zhang, *Fellow, IEEE*

Abstract*—Deep neural networks (DNNs) are widely used and demonstrated their power in many applications, like computer vision and pattern recognition. However, the training of these networks can be time-consuming. Such a problem could be alleviated by using efficient optimizers. As one of the most commonly used optimizers, SGD-Momentum uses past and present gradients for parameter updates. However, in the process of network training, SGD-Momentum may encounter some drawbacks, such as the overshoot phenomenon. This problem would slow the training convergence. To alleviate this problem and accelerate the convergence of DNN optimization, we propose a proportional-integral-derivative (PID) approach. Specifically, we investigate the intrinsic relationships between PID based controller and SGD-Momentum firstly. We further proposed a PID based optimization algorithm to update the network parameters, where the past, current, and change of gradients are exploited. Consequently, our proposed PID based optimization alleviates the overshoot problem suffered by SGD-Momentum. When tested on popular DNN architectures, it also obtains up to 50% acceleration with competitive accuracy. Extensive experiments about computer vision and natural language processing demonstrate the effectiveness of our method on benchmark datasets, including CIFAR10, CIFAR100, Tiny-ImageNet, and PTB. We've released the code at https://github.com/tensorboy/PIDOptimizer.

***Index Terms*—Deep neural network, optimization, PID control, SGD-Momentum.**

I. Introduction

Benefitting from the availability of great number of data (e.g., ImageNet [1]) and the fast-growing power of GPUs, deep neural networks (DNNs) success in a wide range of applications, like computer vision and natural language processing. Despite the significant successes of DNNs, the training and inference of deep and wide DNNs are often computationally expensive, which may take several days or longer even with powerful GPUs. Many stochastic optimization algorithms are not only used in the field of machine learning [2], but also deep learning [3]. It is very important to explore how to boost the speed of training DNNs while maintaining performance. Furthermore, with a better optimization method, even a computation limited hardware (e.g., IoT device) can save lots of time and memory usage. The accelerating methods of the computational time for DNNs can be divided into two parts, the speed-up of training and that of test. The methods in [4]–[6] aiming to speed up test process of DNNs often focus on not only the decomposition of layers but also the optimization solutions to the decomposition. Besides, there has been other streams on improving testing performance of DNNs, such as the FFT-based algorithms [7] and reduced parameters in deep nets [8]. As for the methods to speed up the training speed of DNNs, the key factor is the way to update the millions of parameters of a DNN. This process mainly depends on optimizer and the choice of optimizer is also a key point of a model. Even with the same dataset and architecture, different optimizers could result in very different training effects, due to different directions of the gradient descent, different optimizers may reach completely different local minimum [9].

The learning rate is another principal hyper-parameter for DNN training [10]. Based on different strategies of choosing learning rates, DNN optimizers can be categorized into two groups: 1. Hand-tuned learning rate optimizers: stochastic gradient descent (SGD) [11], SGD Momentum [12], Nesterov′s Momentum [12], etc. 2. Auto learning rate optimizers such as AdaGrad [13], RMSProp [14] and Adam [15], etc.

The SGD-Momentum method puts past and current gradients into consideration and then updates the network parameters. Although SGD-Momentum performs well in most cases, it may encounter overshoot phenomenon [16], which indicates the case where the weight exceeds its target value too much and fails to correct its update direction. Such an overshoot problem costs more resource (e.g., time and GPUs) to train a DNN and also hampers the convergence of SGD-Momentum. So, a more efficient DNN optimizer is eagerly desired to alleviate the overshoot problem and achieve better convergence.

The similarity between optimization algorithms popularly employed in DNN training and classic control methods has been investigated in [17]. In automatic control systems, the feedback control is essential. Proportional-integral-derivative (PID) controller is the most widely used feedback control mechanism, due to its simplicity and functionality [18]. Most of industrial control system are based on PID [19], such as unmanned aerial vehicles [20], robotics [21], and autonomous

This work is partially supported by the NSFC fund (61571259, 61831014, 61531014), in part by the Shenzhen Science and Technology Project under Grant (GGFW2017040714161462, JCYJ20170307153051701). (Corresponding author: Y. Zhang, Email: zhang.yongbing@sz.tsinghua.edu.cn.)

H. Wang, Y. Luo, W. An, and Y. Zhang are with the Graduate School at Shenzhen, Tsinghua University, and also with Shenzhen Institute of Future Media Technology, Shenzhen 518055, China. E-mail: wanghaoqian@tsinghua.edu.cn, vast2stars@gmail.com, anwangpeng@gmail.com, zhang.yongbing@sz.tsinghua.edu.cn.

Q. Sun is with Department of Mathematics, Stanford University, Stanford, CA 94305. E-mail: qysun@stanford.edu.

J. Xu is with College of Computer Science, Nankai University, Tianjin 300071, China. E-mail: nankaimathxujun@gmail.com.

Y. Zhang is with Department of ECE, Northeastern University, Boston, MA 02115. E-mail: yulun100@gmail.com.

L. Zhang is with Department of Computing, the Hong Kong Polytechnic University, Hong Kong, and also with the Artificial Intelligence Center, Alibaba DAMO Academy. Email: cslzhang@comp.polyu.edu.hk.

vehicles [22]. PID control takes current error, change in error (differentiation of the error over time), and the past cumulative error (integral of the error over time) into account. So, the difference between current and expected outputs will be minimized.

On the other hand, few studies have been done on the connections between PID with DNN optimization. In this work, we investigate specific relationships analytically and mathematically towards this research line. We first clarify the intrinsic connection between PID controller and stochastic optimization methods, including SGD, SGD-Momentum, and Nesterov′s Momentum. Finally, we propose a PID based optimization method for DNN training. Similar to SGD-Momentum, our proposed PID optimizer also considers the past and current gradients for network update. The Laplace Transform [23] is further introduced for hyper-parameter initialization, which makes our method simple yet effective. Our major contributions of this work can be summarized in three folds:

- By combining the error calculation in the feedback control system with network parameters' update, we reveal a potential relationship between DNN optimization and feedback system control. We also find that some optimizers (e.g., SGD-Momentum) are special cases of PID control device.
- We propose a PID based DNN optimization approach by taking the past, current, and changing information of the gradient into consideration. The hyper-parameter in our PID optimizer is initialized by classical Laplace Transform.
- We systematically experiment with our proposed PID optimizer on CIFAR10, CIFAR100, Tiny-ImageNet, and PTB datasets. The results show that PID optimizer is faster than SGD-Montum in DNN training process.

A preliminary version of this work was presented as a conference version [24]. In the current work, we incorporate additional contents in significant ways:

- We evaluate the performance of our PID optimizer on the language modeling application by utilizing the character-level Penn Treebank (PTB-c) dataset with an LSTM network.
- The proposed PID optimizer is applied on GAN with MNIST dataset and show the digital images generated by them separately to illustrate that our method is also applicable in GAN.
- We update the conclusion that the proposed PID optimizer exceeds SGD-Momentum in GANs and RNNs.

We organize the rest of this paper as follows. Section II briefly surveys related works. Section III investigates the relationship between PID controller and DNN optimization algorithms. Section IV introduces the proposed PID approach for DNN optimization. Experimental results and detailed analysis are reported in Section V. Section VI concludes this paper.

II. Related Works

A. Classic Deep Neural Network Architectures

CNN. Convolutional neural networks (CNNs) [25] have recently achieved great successes in visual recognition tasks, including image classification [26], object detection [27]–[29], and scene parsing [30]. Recently, lots of deep CNN architectures, such as VGG, ResNet, and DenseNet, have been proposed to improve the performance of these tasks mentioned above. Network depth tends to improve network performance. However, the computational cost of these deep networks also increases significantly. Moreover, real-world systems may be affected by the high cost of these networks.

GAN. Goodfellow *et al.* firstly proposed generative adversarial network (GAN) [31], which consists of generative and adversarial networks. The generator tries to obtain very realistic outputs to foolish the discriminator, which would be optimized to distinguish between the real data and the generated outputs. GANs will be trained to generate synthetic data, mimicking genuine data distribution.

In machine learning, models can be classified into two categories: generative model and discriminative model. A discriminative network (denoted as D) can discriminate between two (or more) different classes of data, such as CNN trained for image classification. A generative network (denoted as G) can generate new data, which fit the distribution of the training data. For example, a trained Gaussian Mixture Model is able to generate new random data, which more-or-less fit the distribution of the training data.

GANs pose a challenging optimization problem due to the multiple loss functions, which must be optimized simultaneously. The optimization of GAN is conducted by two steps: 1) optimize discriminative network while fixing the generative one. 2) optimize the generative network while fixing the discriminative network. Here, fixing a network means only allowing the network to pass forward and not perform back-propagation. These two steps are seamlessly alternating updated and dependent on each other for efficient optimization. After enough training cycles, the optimization objective $V(D,G)$ introduced in [31] will reach the situation, where the probability distribution of the generator exactly matches the true probability distribution of the training data. Meanwhile, the discriminator has the capability to distinguish the realistic data from the virtual generated ones. However, the perfect cooperation between the generator and discriminator will fail occasionally. The whole system will reach the status of "model collapse", indicating that the discriminator and the generator tend to produce the same outputs.

LSTM. Hochreiter *et al.* firstly proposed the Long Short Term network, generally called LSTM, to obtain long-term dependency information from the network. As a type of recurrent neural network (RNN), LSTM has been widely used and obtained excellent success in many applications. LSTM is deliberately designed to avoid long-term dependency problems. Remember that long-term information is the default behavior of LSTM in practice, rather than the ability to acquire at great cost. All RNNs have a chained form of repeating network modules. In the standard RNN, this repeating module often has a simple structure (e.g., "tanh" layer). The outputs of all LSTM cells are utilized to construct a new feature, where multinomial logistic regression is introduced to form the LSTM model.

One widely used way to evaluate RNN models is the adding task [32], [33], which takes two sequences of length T as input. By sampling in the range $(0,1)$ uniformly, we form the first sequence. For another sequence, we set two entries as 1 and the rest as 0. The output is obtained by adding two entries in the first sequence. The positions of the entries are determined by the two entries of 1 from the second sequence.

B. Accelerating the Training/Test Process of DNNs

Training process acceleration. Since DNNs are mostly computationally intensive, Han *et al.* [34] proposed a deep compression method to reduce the storage requirement of DNNs by 35× to 49× without affecting the accuracy. Moreover, the compressed model has 3× to 4× layer-wise speedup and 3× to 7× better energy efficiency. Unimportant connections are pruned. Weight sharing and Huffman coding are applied to quantize the network. This work mainly attempts to reduce the number of parameters of neural networks. Liu *et al.* proposed the network slimming technique that can simultaneously reduce the model size, running-time memory, and computing operations [35]. Yang *et al.* proposed a new filter pruning strategy based on the geometric median to accelerate the training of deep CNNs [36]. Dai *et al.* proposed a synthesis tool to synthesize compact yet accurate DNNs [37]. Du *et al.* proposed a Continuous Growth and Pruning (CGaP) scheme to minimize the redundancy from the beginning [38]. Hubara *et al.* introduced a method to train Quantized Neural Networks that reduce memory size and accesses during forward pass [39]. In [40], Han *et al.* presented an intuitive and easier-to-tune version of ASGD (please refer to Section IV) and showed that ASGD leads to faster convergence significantly with a comparable accuracy than SGD, Heavy Ball, and Nesterov′s Momentum [12].

Test process acceleration. Denton *et al.* [4] proposed a method that compresses all convolutional layers. This is achieved by approximating proper low-rank and then updating the upper layers until the prediction result is enhanced. Based on singular value decompositions (SVD), this process consists of numerous tensor decomposition operations and filter clustering approaches to make use of similarities among learned features. Jaderberg *et al.* [5] introduced an easy-to-implement method that can significantly speed up pretrained CNNs with minimal modifications to existing frameworks. There can be a small associated loss in performance, but this is tunable to a desired accuracy level. Zhang *et al.* [6] first proposed a response reconstruction method, which introduces the nonlinear neurons and a low-rank constraint. Without the usage of SGD and based on generalized singular value decomposition (GSVD), a solution is developed for this nonlinear problem. Li *et al.* presented a method to prune filters with relatively low weight magnitudes to produce CNNs with reduced computation costs without introducing irregular sparsity [41].

C. Deep Learning Optimization

In the training of DNN [10], learning rate is an essential hyper-parameter. DNN optimizers can be categorized into two groups based on different strategies of setting the learning rate: 1. Hand-tuned learning rate optimizers: stochastic gradient descent (SGD) [11], SGD Momentum [12], Nesterov′s Momentum [12], etc. 2. Auto learning rate optimizers, such as AdaGrad [13], RMSProp [14], and Adam [15], etc. Good results have been achieved on CIFAR10, CIFAR100, ImageNet, PASCAL VOC, and MS COCO datasets. They were mostly obtained by residual neural networks [42]–[45] trained by using SGD-Momentum. This work focuses on the improvement of the fist category of optimizers. The introduction to these optimizers is as follows.

Classical Momentum [25] is the first ever variant of gradient descent involving the usage of a momentum parameter. In the objective across iterations, it accelerates gradient descent that collects a velocity vector in directions of continuous reduction.

Stochastic Gradient Descent (SGD) [11] is a widely used optimizer for DNN training. SGD is easy to apply, but the disadvantage of SGD is that it converges slowly and may oscillate at the saddle point. Moreover, how to choose the learning rate reasonably is a major difficulty of SGD.

SGD Momentum (SGD-M) [12] is an optimization method that considers momentum. Compared to the original gradient descent step, the SGD-M introduces variables related to the previous step. It means that the parameter update direction is decided not only by the present gradient, but also by the previously accumulated direction of the fall. This allows the parameters to change little in the direction where gradient change frequently. Contrary to this, SGD-M changes parameters a lot in the direction where gradient change slowly.

Nesterov′s Momentum [12] is another momentum optimization algorithm motivated by Nesterov′s accelerated gradient method [46]. Momentum is improved from the SGD algorithm, so that each parameter update direction depends not only on the gradient of the current position, but also on the direction of the last parameter update. In other words, Nesterov′s Momentum essentially uses the second-order information of the objective (loss function) so it can accelerate the convergence better.

D. PID Controller

Traditionally, the PID controller has been used to control a feedback system [19] by exploiting the present, past, and future information of prediction error. The theoretical basis of the PID controller was first proposed by Maxwell in 1868 in his seminal paper "On Governors" [47]. Mathematical formulation was given by Minorsky [48]. In recent years, several advanced control algorithms have been proposed.

We define the difference between the actual output and the desired output as error $e(t)$. The PID controller calculates the error $e(t)$ in every step t, and then applies a correction $u(t)$ to the system as a function of the proportional (P), integral

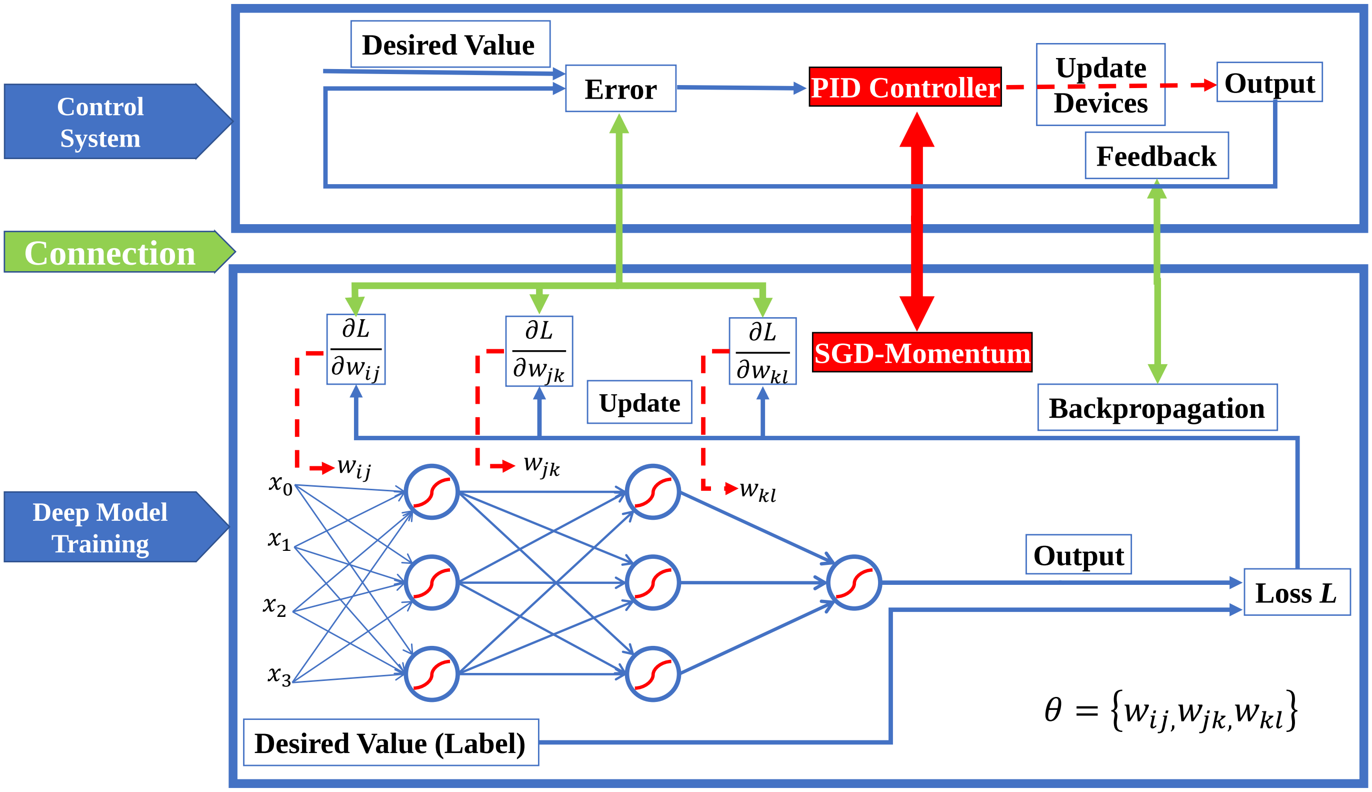


Fig. 1. Illustrations about the relationships between control system and deep model training. It also shows the connection between PID controller and SGD-Momentum.

(I), and derivative (D) terms of $e(t)$. Mathematically, the PID controller can be described as

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{d}{dt} e(t), \tag{1}$$

where K_p, K_i, and K_d correspond to the gain coefficients of the P, I, and D terms, respectively. The function of error $e(t)$ is the same as the gradient in optimization of deep learning. The coefficients K_p, K_i, and K_d reflect the contribution to the current correction to the current, past, and future errors respectively.

According to our analyses, we find that PID control techniques can be more useful for optimization of deep network. The study presented in this paper is one of the first investigations to apply PID as a new optimizer to deep learning field. Our studies have succeeded in demonstrating significant advantages of the proposed optimizer. With the inheritance of the advantages of PID controller, the proposed optimizer performs well despite its simplicity.

III. PID and Deep Neural Network Optimization

We reveal the intrinsic relation between PID control and DNNs optimization. The intrinsic relation inspires us to explore new DNNs optimization methods. The core idea of this section is to regard the parameter update in DNNs training process as using PID controller in the system to reach an equilibrium.

A. Overall Connections

At first, we summarize the training process of deep learning. Deep neural networks (DNNs) need to map the input x to the output y though parameters θ. To measure the gap between the DNN output and desired output, the loss function L is introduced. Given some training data, we can calculate the loss function $L(\theta, X_{train})$. In order to minimize the loss function L, we find the derivative of the loss function L with respect to the parameter θ and update θ with the gradient descent method in most cases. DNNs gradually learn the complex relationship between input x and output y by constantly updating the parameters θ, which called DNN's training. The updating of θ is driven by the gradient of loss function until it's converged.

Then, the purpose of an automated control system is to evaluate the system status and make it to the desired status through a controller. In feedback control system, the controller's action is affected by the system's output. The error $e(t)$ between the measured system status and desired status is taken into consideration, so that controller can make system get close to desired status.

More specifically, as shown in Eq. (1), PID controller estimates a control variable $u(t)$ by considering the current, past, and future (derivative) of the error $e(t)$.

From here we can see that the error in the PID control system is related to the gradient in the deep neural network training process. The update of parameters during deep neural network (DNNs) training can be analogized to the adjustment of the system by the PID controller.

As can be seen from the discussion above, there is high similarity between DNNs optimization and PID based control system. Fig. 1 shows their flowchart respectively and we can see the similarity more intuitively. Based on the difference between the output and target, both of them change the system/network. The negative feedback process in PID controller is similar as the back-propagation in DNNs optimization.

One key difference is that the PID controller computes the update utilizing system error $e(t)$. However, DNN optimizer decides the updates by considering gradient $\partial L/\partial\theta$. Let's regard the gradient $\partial L/\partial\theta$ as the incarnation of error $e(t)$. Then, PID controller could be fully related with DNN optimization. In the next, we prove that SGD, SGD-Momentum and Nesterov′s Momentum all are special cases of PID controller.

B. Stochastic Gradient Descent (SGD)

In DNN training, there are widely used optimizers, such as SGD and its variants. The parameter update rule of SGD from iteration t to $t+1$ is determined by

$$\theta_{t+1} = \theta_t - r\partial L_t/\partial\theta_t, \tag{2}$$

where r is the learning rate. We now regard the gradient $\partial L_t/\partial\theta_t$ as error $e(t)$ in PID control system. Comparing with PID controller in Eq. (1), we find that SGD can be viewed as one type of P controller with $K_p = r$.

C. SGD-Momentum

SGD-Momentum is faster than SGD to train a DNN, because it can use history gradient. The rule of SGD-M updating parameter is given by

$$\begin{cases} V_{t+1} = \alpha V_t - r\partial L_t/\partial\theta_t \\ \theta_{t+1} = \theta_t + V_{t+1} \end{cases}, \tag{3}$$

where V_t is a term that accumulates historical gradients. $\alpha \in (0,1)$ is the factor that balances the past and current gradients. It is usually set to 0.9 [49]. Dividing two sides of the 1st formula of Eq. (3) by α^{t+1}

$$\frac{V_{t+1}}{\alpha^{t+1}} = \frac{V_t}{\alpha^t} - r\frac{\partial L_t/\partial\theta_t}{\alpha^{t+1}}. \tag{4}$$

By applying Eq. (4) from time $t+1$ to 1, we have

$$\begin{cases} \frac{V_{t+1}}{\alpha^{t+1}} - \frac{V_t}{\alpha^t} = -r\frac{\partial L_t/\partial\theta_t}{\alpha^{t+1}} \\ \frac{V_t}{\alpha^t} - \frac{V_{t-1}}{\alpha^{t-1}} = -r\frac{\partial L_{t-1}/\partial\theta_{t-1}}{\alpha^t} \\ \quad\vdots \\ \frac{V_1}{\alpha^1} - \frac{V_0}{\alpha^0} = -r\frac{\partial L_0/\partial\theta_0}{\alpha^1}. \end{cases} \tag{5}$$

By adding the aforementioned equations together, we get

$$\frac{V_{t+1}}{\alpha^{t+1}} = \frac{V_0}{\alpha^0} - r\sum_{i=0}^{t}\frac{\partial L_i/\partial\theta_i}{\alpha^{i+1}}. \tag{6}$$

To make it more general, we set the initial condition $V_0 = 0$, and thus the above equation can be simplified as follows

$$V_{t+1} = -r\sum_{i=0}^{t}\alpha^{t-i}\partial L_i/\partial\theta_{t-1}. \tag{7}$$

Put V_{t+1} into the 2nd formula of Eq. (3), we have

$$\theta_{t+1} - \theta_t = -r\frac{\partial L_t}{\partial\theta_t} - r\sum_{i=0}^{t-1}\alpha^{t-i}\partial L_i/\partial\theta_i. \tag{8}$$

We could learn that parameter update process considers both the current gradient (P control) and the integral of past gradients (I control). If we assume $\alpha = 1$, we get following equation

$$\theta_{t+1} - \theta_t = -r\partial L_t/\partial\theta_t - r\sum_{i=0}^{t-1}\partial L_i/\partial\theta_i. \tag{9}$$

Comparing Eq. (9) with Eq. (1), we can see that SGD-Momentum is a PI controller with $K_p = r$ and $K_i = r$. By using some mathematical skill [50], we simplify Eq. (3) by removing V_t. Then, Eq. (9) can be rewritten as

$$\theta_{t+1} = \theta_t - r\partial L_t/\partial\theta_t - r\sum_{i=0}^{t-1}\partial L_i/\partial\theta_i\alpha^{t-i}. \tag{10}$$

We can see it clear that the network parameter update depends on both current gradient $r\partial L_t/\partial\theta_t$ and the integral of past gradients $r\sum_{i=0}^{t-1}\partial L_i/\partial\theta_i\alpha^{t-i}$. It should be noted that the I term includes a decay factor α. Due to the huge number of training data, it's better to calculate the gradient based on mini-batch of training data. So, the gradients behave in a stochastic manner. The purpose of the introduction of decay term α is to keep the gradients away from current value, so that it can alleviate noise. In all, based on the analyses, we can view SGD-Momentum as a PI controller.

D. Nesterov′s Momentum

Momentum is improved from the SGD algorithm and it considers the second-order information of the objective (loss function), so it can accelerate the convergence better. We set the update rule as

$$\begin{cases} V_{t+1} = \alpha V_t - r\partial L_t/\partial(\theta_t + \alpha V_t) \\ \theta_{t+1} = \theta_t + V_{t+1}. \end{cases} \tag{11}$$

By using a variable transform $\hat{\theta}_t = \theta_t + \alpha V_t$, and formulating the update rule with respect to $\hat{\theta}$, we have

$$\begin{cases} V_{t+1} = \alpha V_t - r\partial L_t/\partial\hat{\theta}_t \\ \hat{\theta}_{t+1} = \hat{\theta}_t + (1+\alpha)V_{t+1} - \alpha V_t. \end{cases} \tag{12}$$

Similar to the derivation process in Eq. (4)-(6) of SGD-Momentum, we have

$$V_{t+1} = -r(\sum_{i=1}^{t}(\alpha^{t-i}\partial L_i/\partial\hat{\theta}_i)). \tag{13}$$

With Eq. (13), Eq. (11) can be rewritten as

$$\hat{\theta}_{t+1} - \hat{\theta}_t = -r(1+\alpha)\partial L_t/\partial\hat{\theta}_t - \alpha r(\sum_{i=1}^{t-1}(\alpha^{t-i}\partial L_i/\partial\hat{\theta}_i)). \tag{14}$$

We could conclude that the network parameter update considers the current gradient (P control) and the integral of past gradients (I control). If we assume $\alpha = 1$, then

$$\hat{\theta}_{t+1} - \hat{\theta}_t = -2r(\partial L_t/\partial \hat{\theta}_t) - r(\sum_{i=0}^{t-1}(\partial L_i/\partial \hat{\theta}_i)). \quad (15)$$

Comparing Eq. (15) with Eq. (1), we can prove that Nesterov′s Momentum is a PI controller with $K_p = 2r$ and $K_i = r$. What's more, compared with SGD-Momentum, the Nesterov′s Momentum would utilize the current gradient and integral of past gradients to update the network parameters, but achieves larger gain coefficient K_p.

IV. PID Based DNN Optimization

A. The Overshoot Problem of SGD-Momentum

We can learn it from Eqs. (10) and (14) that the Momentum optimizer will accumulate history gradients to accelerate. But on the other hand, the updating of parameters may be in wrong path, if the history gradients lag the update of parameters. According to the definition "the maximum peak value of the response curve measured from the desired response of the system" in discrete-time control systems [16], this phenomenon is named as overshoot. Specifically, it can be written as

$$Overshoot = \frac{\theta_{max} - \theta^*}{\theta^*}, \quad (16)$$

where θ_{max} and θ^* are the maximum and optimum values of the weight, respectively.

The overshoot problem's test benchmark is the first function of De Jong′s [51] due to its smooth, unimodal, and symmetric characteristics. The function can be written as

$$f(x) = 0.1x_1^2 + 2x_2^2, \quad (17)$$

whose search domain is $-10 \leq x_i \leq 10, i = 1,2$. For this function $x^* = (0,0), f(x^*) = 0$, we can pursue a global minimum rather then a local one.

To build a simple PID optimizer, we introduce a derivative term of gradient based on SGD-Momentum

$$PID = Momentum + K_d(\partial f(x)/\partial x_c - \partial f(x)/\partial x_{c-1}), \quad (18)$$

where c is the present iteration index for x. With different choices of K_d in Eq. (18), we shows the results of simulation in Fig. 2, where the loss-contour map is represented as the background. The redder, the bigger the loss function value is. In contrast, the bluer, the smaller the loss function value is. The x-axis and y-axis denote x_1 and x_2, respectively. Both x_1 and x_2 are initialized to -10. We use red and yellow lines to show the path of PID and SGD-Momentum, respectively. It is obvious that SGD-Momentum optimizer suffers from overshoot problem. By increasing K_d gradually (0.1, 0.5, and 0.93, respectively), our PID optimizer uses more "future" error, so that it can largely alleviate the overshoot problem.

B. PID Optimizer for DNN

We are motivated by the simple example in Section IV-A and seek a PID optimizer to boost the convergence of DNN training. From Eq. (10), SGD-Momentum can be viewed as a PI controller, which takes current and past gradient information actually. Fig. 2 shows that PID controller introduces a derivative term of gradient to use the future information. Then, the overshoot problem can be alleviated obviously.

On the other hand, it is very easy to introduce noise when computing of gradients, because the training is often conducted in a mini-batch manner. We also try to estimate the average moving of the derivative part. Our proposed PID optimizer updates network parameter θ in iteration $(t+1)$ by

$$\begin{cases} V_{t+1} = \alpha V_t - r\partial L_t/\partial \theta_t \\ D_{t+1} = \alpha D_t + (1-\alpha)(\partial L_t/\partial \theta_t - \partial L_{t-1}/\partial \theta_{t-1}) \\ \theta_{t+1} = \theta_t + V_{t+1} + K_d D_{t+1}. \end{cases} \quad (19)$$

We could learn from Eq. (19) that a hyper-parameter K_d is introduced in the proposed PID optimizer. We initialize K_d by introducing Laplace Transform [23] theory and Ziegler-Nichols [52] tuning method.

C. Initialization of Hyper-parameter K_d

The Laplace Transform converts the function of real variable t to a function of complex variable s. The most common usage is to convert time to frequency. Denote the Laplace transformation of $f(t)$ as $F(s)$. There is

$$F(s) = \int_0^\infty e^{-st} f(t)\,dt, \quad \text{for } s > 0. \quad (20)$$

In general, it's easier to solve $F(s)$ than $f(t)$, which can be reconstructed from $F(s)$ with the Inverse Laplace transform

$$f(t) = \frac{1}{2\pi i} \lim_{T\to\infty} \int_{\gamma-iT}^{\gamma+iT} e^{st} F(s) ds, \quad (21)$$

where i is the unit of imagery part and γ is a real number.

By using Laplace Transform, we can first transform our PID optimizer into its Laplace transformed functions of s, and then simplify the algebra. After obtaining the transformation $F(s)$, we can achieve the desired solution $f(t)$ with the inverse transform.

We initialize a parameter of a node in DNN model as a scalar θ_0. After enough times of updates, the optimal value θ^* can be obtained. We simplify the parameter update in DNN optimization as one step response (from θ_0 to θ^*) in control system. We introduce the Laplace Transform to set K_d and denote the time domain change of weight θ as $\theta(t)$.

The Laplace Transform of θ^* is $\frac{\theta^*}{s}$ [53]. We denote by $\theta(t)$ the weight at iteration t. The Laplace Transform of $\theta(t)$ is denoted as $\theta(s)$, and that of error $e(t)$ as $E(s)$,

$$E(s) = \frac{\theta^*}{s} - \theta(s).$$

The Laplace transform of PID [53] is

$$U(s) = (K_p + K_i\frac{1}{s} + K_d s)E(s). \quad (22)$$

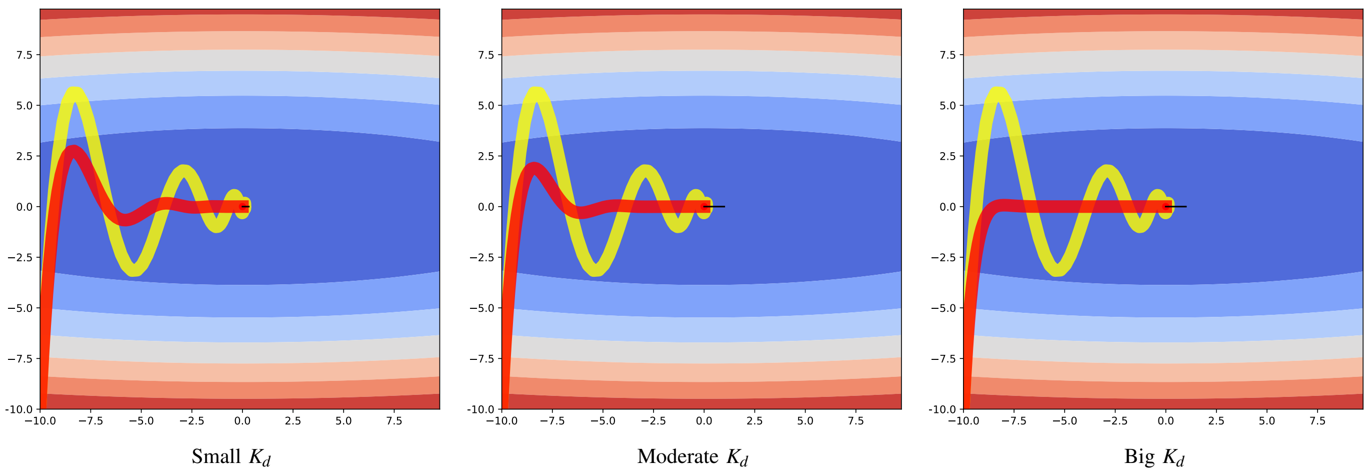


Fig. 2. The overshoot problem of momentum with different values of K_d. The red and yellow lines indicate the results obtained by PID and SGD-Momentum respectively.

In our case, the $u(t)$ corresponds to the update of $\theta(t)$. So we replace $U(s)$ with $\theta(s)$, and with $E(s) = \frac{\theta^*}{s} - \theta(s)$. Eq. (22) can be rewritten as

$$\theta(s) = (K_p + K_i \frac{1}{s} + K_d s)(\frac{\theta^*}{s} - \theta(s)). \tag{23}$$

With this form, it is easy to derive a standard closed loop transfer function [54] as

$$\frac{\theta^*}{s} - \theta(s) = \frac{1}{K_d} \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \tag{24}$$

where

$$\begin{cases} \frac{K_p+1}{K_d} & = 2\zeta\omega_n \\ \frac{K_i}{K_d} & = \omega_n^2. \end{cases} \tag{25}$$

Eq. (24) can be rewritten as

$$\frac{\theta^*}{s} - \theta(s) = \frac{(s+\zeta\omega_n) + \frac{\zeta}{\sqrt{1+\zeta^2}}\omega_n\sqrt{1-\zeta^2}}{(s+\zeta\omega_n)^2 + \omega_n^2(1-\zeta^2)}. \tag{26}$$

We can get the time (iteration) domain form of $\theta(s)$ by using the Inverse Laplace Transform table [53] and the initial condition of the θ (θ_0):

$$\theta(t) = \theta^* - \frac{(\theta^* - \theta_0)\sin(\omega_n\sqrt{1-\zeta^2}t + \arccos(\zeta))}{e^{\zeta\omega_n t}\sqrt{1-\zeta^2}} \tag{27}$$

and

$$\begin{cases} (K_p+1)/K_d = 2\zeta\omega_n \\ K_i/K_d = \omega_n^2, \end{cases} \tag{28}$$

where ζ is damping ratio and ω_n is natural frequency of the system. The evolution process of a weight as an example of $\theta(t)$ is shown in Fig. 3. From Eq. (28), we get

$$K_i = \frac{(K_p+1)^2}{4K_d\zeta}. \tag{29}$$

From Eq. (29) we know that K_i is a monotonically decreasing function of ζ. Based on the definition of overshoot in Eq. (16), it is obvious that ζ is monotonically decreasing with overshoot. Then, K_i is a monotonically increasing function of overshoot. In a word, the more history error (Integral part), the more overshoot problem in the system. This is a good explanation of why SGD-Momentum overshoots its target and need more training time.

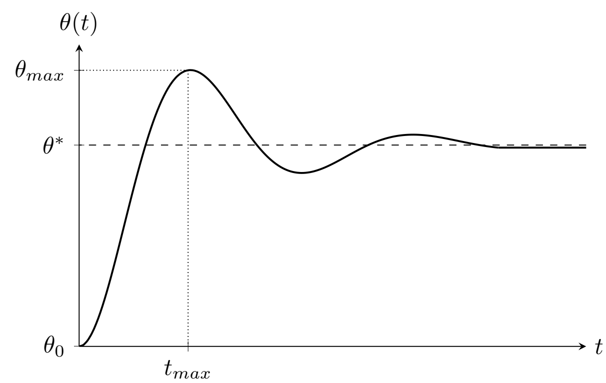


Fig. 3. The evolution process of the weight $\theta(t)$ for PID optimizer.

By differentiating $\theta(t)$ w.r.t. time t, and let

$$\frac{d\theta(t)}{dt} = 0.$$

We have the peak time of the weight as

$$t_{max} = \frac{\pi}{w_n\sqrt{1-\zeta^2}}. \tag{30}$$

Put t_{max} to Eq. (27), we have θ_{max}, and put θ_{max} to Eq. (16), we have

$$Overshoot = \frac{\theta(t_{max}) - \theta^*}{\theta^*} = e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}}. \tag{31}$$

We could learn from Eq. (27) that the term $\sin(\omega_n\sqrt{1-\zeta^2}t + \arccos(\zeta))$ brings periodically oscillation change to the weight, which is no more than 1. The term $e^{-\zeta\omega_n t}$ mainly controls the convergence rate. It should be noted that the value of hyper-parameter K_d in calculating the derivate

$$e^{-\zeta\omega_n} = e^{-\frac{K_p+1}{2K_d}}. \tag{32}$$

Based on the above analyses, we know that the training of DNN can be accelerated by using large derivate. But on the other hand, if K_d is too large, the system will be fragile. After some experiments, we set the K_d based on the Ziegler-Nichols optimum setting rule [52].

According to the Ziegler-Nichols′ rule, the ideal setup of K_d should be one third of the oscillation period (T), which means $K_d = \frac{1}{3}T$. From Eq. (27), we can get $T = \frac{2\pi}{\omega_n\sqrt{1-\zeta^2}}$. If we make a simplification that the α in Momentum is equal to 1, then $K_i = K_d = r$. Combined with Eq. (28), K_d will have a closed form solution

$$K_d = 0.25r + 0.5 + (1 + \frac{16}{9}\pi^2)/r. \tag{33}$$

For real-world cases, where different DNNs are applied train on different datasets, we would firstly start with this ideal setting of K_d and change it slightly latter.

V. Experimental Results

We introduce four commonly used datasets for the experiments. Then, we compare our proposed optimizer with other optimizers by using CNN and LSTM on four commonly used datasets. Specifically, we first train an multilayer perceptron (MLP) on the MNIST dataset to demonstrate the advantages of PID optimizer. We then train CNNs on the CIFAR datasets to show that our PID optimizer achieves competitive accuracy compared with other optimizers, but it has a faster training speed. Further studies are carried out to prove that our PID optimizer also performs well on a larger dataset. Based on the Tiny-ImageNet dataset [55], we carry out a series of experiments. The results indicate that it is applicable for our PID optimizer to be extended to modern networks. Our proposed PID optimizer is set to use all hyper-parameters that are detailed for SGD-Momentum. The initial learning rate and learning rate schedule vary with different experiments.

A. Datasets

MNIST Dataset. The MNIST dataset [56] of handwritten numbers from 0 to 9. Being a subset of a larger dataset NIST, MNIST consists of 60,000 training data and 10,000 test ones. The digits have been size-normalized and centered in a fixed-size image of 28×28 pixels. With the usage of anti-aliasing technique, the preprocessed images contain gray levels.

CIFAR Datasets. The CIFAR10 dataset [57] has 60,000 RGB color images, the shape of which is 32×32. There are 10 classes, each of which includes 6,000 images. 50,000 and 10,000 images are used for training and testing respectively. Similar as CIFAR10, CIFAR100 dataset [57] consists of 100 classes with 600 images for each class. 500 and 100 images are extracted from each class for training and testing respectively. The 100 classes in the CIFAR100 [57] are further arranged into 20 super classes. We performed random crops, horizontal flips, and padded 4 pixels around each side on the original image for data augmentation.

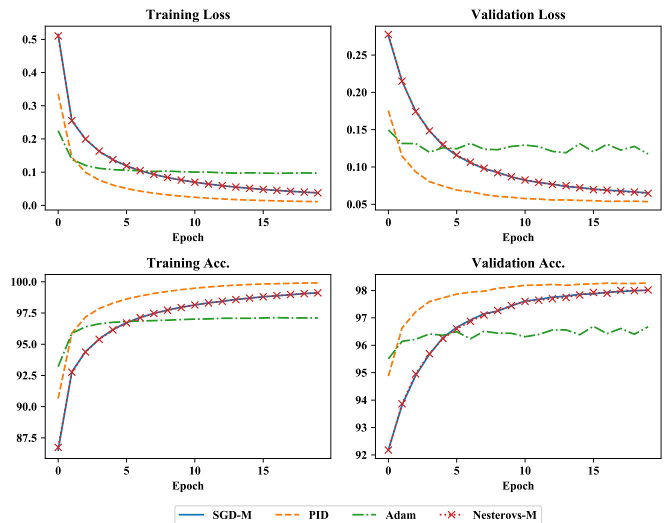


Fig. 4. Comparison between PID and other optimizers on the MNIST dataset for 20 epochs. Top row: the curves of training loss and validation loss. PID optimizer obtains lower losses and converges faster. Bottom row: the curves of training accuracy and validation accuracy. PID optimizer performs much better than others for both training and test accuracies.

Fig. 5. PID *vs.* other optimizers on the MNIST dataset for 20 epochs. Standard deviation of 10 runs. Top row: the curves of training loss and validation loss. Bottom row: the curves of training accuracy and validation accuracy.

Tiny ImageNet Dataset. There are 200 classes in Tiny-Imagenet [55] dataset. Each class contains 500, 50, and 50 images for training, validation, and testing respectively. The Tiny-ImageNet is harder to be classed correctly than the CIFAR datasets. It is not only because a larger number of classes, but also the relevant objects need to be classified usually occupy little pixels of the whole image.

PTB Dataset. Penn Treebank dataset, known as PTB dataset, is widely used in machine learning of NLP (Natural Language Processing) research. The PTB dataset has 2,499 stories which come from a three-year WSJ collection of 98,732 stories.

TABLE I
COMPARISONS BETWEEN PID AND SGD-MOMENTUM OPTIMIZERS IN TERMS OF TEST ERRORS AND TRAINING EPOCHS. WE REPORT THE RESULTS BASED ON CIFAR10 AND CIFAR100.

Model	Depth-k	Params (M)	Runs	CIAFR10	Epochs	CIFAR100	Epochs
-	-	-	-	PID/SGD-M	PID/SGD-M	PID/SGD-M	PID/SGD-M
Resnet [42]	110	1.7	5	**6.23**/6.43	**239**/281	**24.95**/25.16	**237**/293
	1202	10.2	5	**7.81**/7.93	**230**/293	27.93/**27.82**	**251**/296
PreActResNet [44]	164	1.7	5	**5.23**/5.46	**230**/271	**24.17**/24.33	**241**/282
ResNeXt29 [58]	8-64	34.43	10	3.65/**3.43**	**221**/294	**17.46**/17.77	**232**/291
	16-64	68.16	10	**3.42**/3.58	**209**/289	**17.11**/17.31	**229**/283
WRN [45]	16-8	11	10	**4.42**/4.81	**213**/290	**21.93**/22.07	**229**/283
	28-20	36.5	10	4.27/**4.17**	**208**/290	**20.21**/20.50	**221**/295
DenseNet [43]	100-12	0.8	10	**3.83**/4.30	**196**/291	**19.97**/20.20	**213**/294
	190-40	25.6	10	**3.11**/3.32	**194**/293	**16.95**/17.17	**208**/297

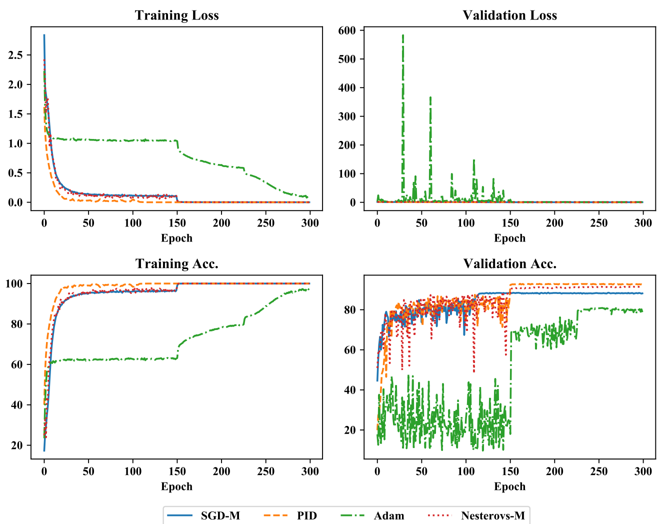


Fig. 6. Comparison among PID and other optimizers on the CIFAR10 dataset by using DenseNet 190-40. Top row: the curves of training and validation loss. PID optimizer obtains lower losses and behaves more stable. Bottom row: the curves of training accuracy and validation accuracy. PID optimizer performs slightly better than SGD-Momentum for both training and test accuracies.

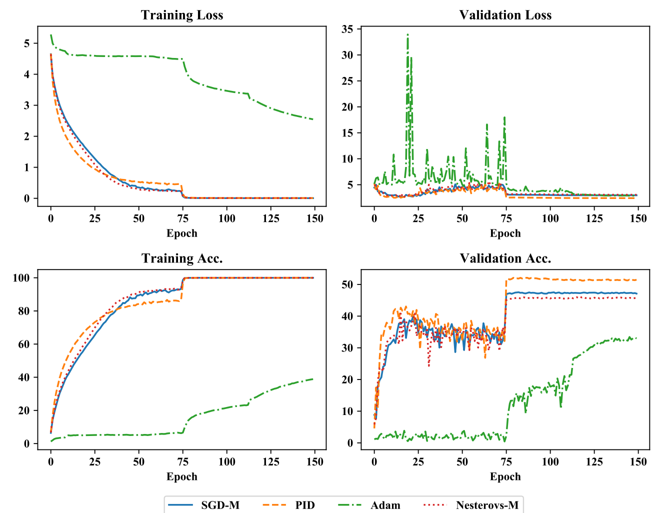


Fig. 7. Comparison among PID and other optimizers on the Tiny-imagenet dataset with DenseNet 100-12 backbone. Top row: the curves of training and validation loss. We can see PID optimizer obtains lower training and validation losses. Bottom row: the curves of training accuracy and validation accuracy. PID optimizer achieves best performance.

B. Results of CNNs

Results of MLP on MNIST dataset. To compare the proposed PID optimizer with SGD-Momentum [12], we first carry out a series of experiments. We use MNIST dataset to train a basic network, MLP. There are $1,000$ hidden nodes in the hidden layer. ReLU acts as nonlinearity layer in the MLP network. We place softmax layer on the top. The training batch size is 128 for 20 epochs. After running the experiments for 10 times, we obtain the average results.Fig. 4 shows comparisons among four methods in terms of training statistics. The Adam performs well in the early stages of training, but overall it could be very unstable and slower than PID optimizer. As Fig. 4 shows, the PID optimizer performs faster convergence than other optimizers. What's more, PID optimizer achieves lower loss and higher accuracy in both training and validation phases. Plus, it has stronger generalization ability on the test dataset. It can be seen from Fig. 5 that the standard deviation of the PID optimizer during training is minimal, which proves its training stability. The accuracy is 98% in PID optimizer and 97.5% in SGD-Momentum.

Results on CIFAR datasets. In order to fully test our proposed PID optimizer, we compare it with SGD-Momentum optimizer on recent leading DNN models (ResNet [42], PreActResNet [44], ResNeXt29 [58], WRN [45], and DenseNet [43]). The details are shown in Tab. I, where the second column lists the depth of networks and k. The k in ResNeXt29, WRN, and DenseNet represent cardinality, widening factor, and growth rate respectively. The third column lists the number of parameters. The fourth column shows the update numbers to calculate the mean test error. The next 4 columns show the average test error and the numbers of epoch, when they accomplish the test errors firstly (the minimum number of epoch to reach the best accuracy).

The following conclusion can be given from Tab. I. First, compared with SGD-Momentum, our PID optimizer obtains lower test errors for all architectures (except for ResNet with depth $1,202$) based on results from CIFAR10 and CIFAR100 datasets. Second, for the training epochs needed to reach the best results, PID optimizer needs less number of training than SGD-Momentum. Specifically, compared with SGD-Momentum, our proposed PID optimizer achieves 35% and up to 50% acceleration on average. This reveals that the gradient descent's direction acts a very important role, which can be utilized to alleviate the overshoot problem and contribute to faster convergence for training of DNNs. In Fig. 6, we further present more training statistics on CIFAR10 to compare PID and SGD-Momentum optimizers. For the backbone DenseNet 190-40 [43], we set its network depth as 190 and growth rate as 40. Based on the experiments, we can obviously conclude that our PID optimizer achieves faster converges than SGD-Momentum. More important, in both training and validation phases, PID optimizer obtains lower loss and higher accuracy.

Results on Tiny-ImageNet. We also apply our proposed PID optimizer on the Tint-ImageNet dataset with DenseNet 100-12 architecture to indicate its effectiveness. The initial learninig rate of four optimizers are 0.1. The decreasing schedule is set to 50% and 75% of training epochs. The batch size is 500. In Fig. 7, we show the curves of training loss and accuracy, as well as validation loss and accuracy over the change of epochs for the four optimizers. Similar to results tested on the CIFAR datasets, the proposed PID optimizer not only converges faster but also obtains better performance. These results prove the generalization ability of our proposed PID optimizer.

C. Results of GANs

During the training of generative adversarial networks (GANs), both G and D needs to be trained. We train them both in an alternating manner. Each of their objectives can be expressed as a loss function that we can optimize via gradient descent. So, we train G for a couple steps, then train D for a couple steps, then give G the chance to improve itself, and so on. The result is that the generator and the discriminator get better at their objectives in terms. So that the generator can fool the most sophisticated discriminator finally. In practice, this method ends up with generative neural nets that are good at producing new data.

In the experiments, we use a deep convolutional generative adversarial networks (DCGAN) to test our proposed PID optimizer. The discriminator of this DCGAN consists of 2 convolutional layers (with ReLU function and max pooling) and 2 fully-connected layers. The generator of this DCGAN consists of a fully connected layer (with batch normalization and ReLU function) and 3 convolutional layers. The binary cross entropy is used as a loss function. The learning rate is initialized to 0.0003 for all optimizers. The qualitative results of PID are illustrated in Fig. 8(b) and the SGD-Momentum results are demonstrated in Fig. 8(a). From Fig. 8, we could find that the generated images with PID optimizer are more realistic than these with SGD-Momentum optimizer.

D. Results of RNNs

In this experiment, we employ a simple LSTM that only has 1 layer with 100 hidden units. Mean squared error (MSE) is used as the objective function for the adding problem. The initial learning rate is set to 0.002 for SGD-Momentum and PID optimizer. The learning rate is reduced by a factor of 10 every $20,000$ training steps. We randomly generate all the training and testing data throughout the whole experiments. The results are shown in Fig. 9. The LSTM model with SGD-Momentum has troubles in convergence. However, our proposed PID optimizer can reach to a small error with very fast convergence. It indicates that our proposed PID optimizer could effectively train LSTM.

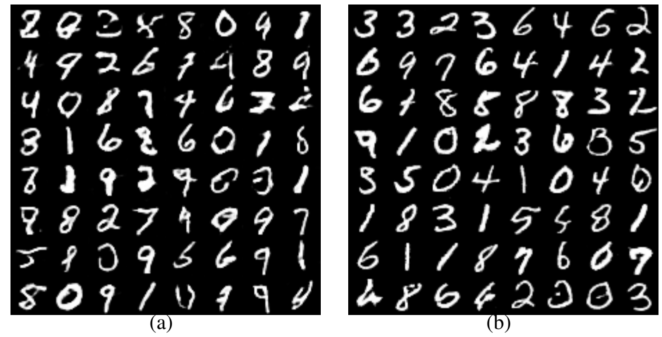


Fig. 8. PID *vs.* SGD-Momentum for generating images through GANs on MNIST dataset. (a) The generated images from SGD-Momentum. (b) The generated images from PID.

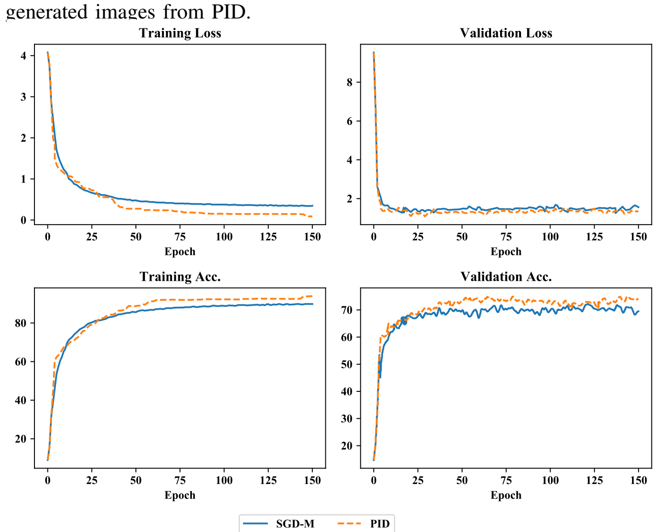


Fig. 9. Comparison between PID and SGD-Momentum for the Adding task of RNN. Top row: the curves of training and validation loss. PID optimizer achieves lower training and validation losses than SGD-Momentum. Bottom row: the curves of training and validation accuracy. Our PID optimizer performs better in both training and test performance.

Results on PTB dataset. In this subsection, we evaluate the character-level Penn Treebank (PTB-c) dataset to evaluate our proposed PID optimizer. We follow the similar experimental settings as in [59]. Specifically, we apply the frame-wise batch normalization [60] and set batch size as 128. The learning rate is initially set to 0.0002 and decreases by 10 times when the validation performance no longer improve. We also introduce dropout [61] by using dropping probability of 0.25 and 0.3. There is no overlapping in the sequences, whose length are set as $T = 50$ for both training and testing. Then we train networks with PID and SGD-Momentum optimizers. The results are shown in Fig. 10. Comparing with the SGD-Momentum, we can see that our proposed PID optimizer achieves better performance on the LSTM model.

E. Results of different K_i and K_d

We also perform an ablation study on the hyper-parameters of PID controller. The experiments are run on the CIFAR10 dataset with DenseNet 100-12. The initial learning rate is 0.1, and it is reduced by 10 in the 150 and 225 epochs.

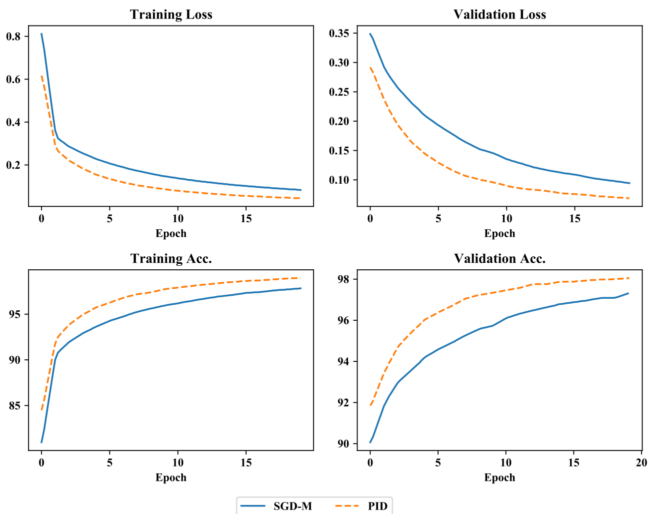


Fig. 10. Comparison between PID and SGD-Momentum to train LSTM on PTB dataset. Top row: the curves of training and validation loss. PID optimizer helps to achieve smaller training and validation losses. Bottom row: the curves of training and validation accuracy. PID optimizer helps to achieve higer training and validation performance.

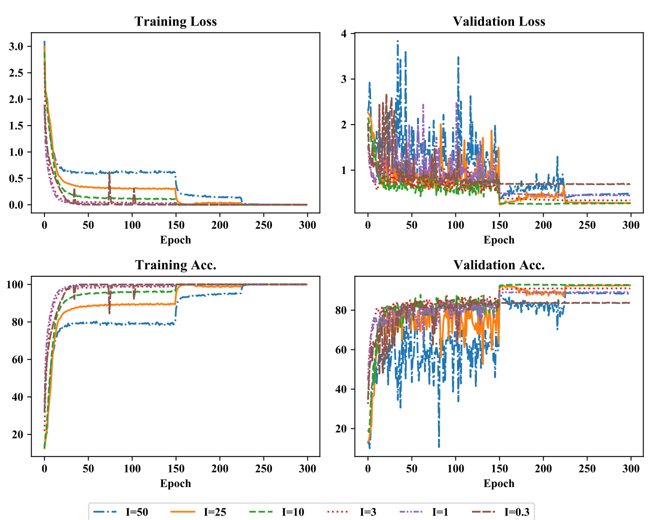


Fig. 11. Comparison among PID controllers with different K_i on the CIFAR10 dataset by using DenseNet 100-12. K_d is fixed to 10. Top row: the curves of training and validation loss. Bottom row: the curves of training and validation accuracy. Within a certain range, larger K_i achieves better validation accuracy.

The first group of experiments investigates the variation of training and verification statistics with K_i while K_d is fixed. Fig. 11 demonstrates six PID controllers whose K_d is 10. In the training, the performance of all controllers differ from each other at an early stage, but eventually they can reach the same level. In validation, controller with $K_i = 10$ achieves lowest loss and highest validation accuracy. We also repeat this experiment with $K_d = 10, 25, 50, and 100$ respectively, and results are highly similar to Fig. 11. One interesting phenomenon is that the larger the K_i, the more affected by the decreasing schedule.

Then we change the research object to K_d. The settings of the second group of experiments are kept the same as previous experiments, but the K_i is fixed. Fig. 12 shows that their performance is highly consistent. It is also shown that the

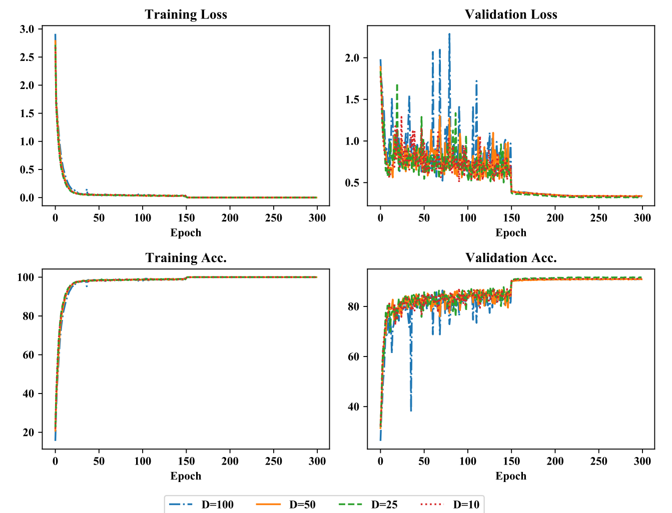


Fig. 12. Comparison among PID controllers with different K_d on the CIFAR10 dataset by using DenseNet 100-12. K_i is fixed to 3. Top row: the curves of training and validation loss. Bottom row: the curves of training and validation accuracy.

larger the K_d, the more unstable the validation performance. The reasons may be that large K_d leads to more change of optimization path.

As can be seen from these experiments, K_i is more important than K_d in this specific tasks (CIFAR10 with Densenet100-12). K_i not only affects the speed of convergence, but also affects the accuracy of verification.

VI. Conclusion and Future Work

Motivated by the outstanding performance of proportional-integral-derivative (PID) controller in the field of automatic control, we reveal the connections between PID controller and stochastic optimizers and its variants. Then we propose a new PID optimizer used in deep neural network training. The proposed PID optimizer reduces the overshoot phenomenon of SGD-momentum and accelerates the training process of DNNs by combining the present, the past and the change information of gradients to update parameters. Our experiments on both image recognition tasks with MNIST, CIFAR, and Tiny-ImageNet datasets and LSTM tasks with PTB dataset validates that the proposed PID optimizer is 30% to 50% faster than SGD-Momentum, while obtaining lower error rate. We will continue to study the relationship among optimal hyper-parameters(K_p, K_i, and K_d) in specific task. We will conduct more in-depth researches for more general cases in the future. And we will investigate how to associate PID optimizer with an adaptive learning rate for DNNs/RNNs optimization in future works.

Acknowledgment

This work is partially supported by the NSFC fund (61571259, 61831014, 61531014), in part by the Shenzhen Science and Technology Project under Grant (GGFW2017040714161462, JCYJ20170307153051701).

REFERENCES

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. fei Li, "Imagenet large scale visual recognition challenge," *IEEE International Journal of Computer Vision (IJCV)*, 2015.

[2] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[3] J. Zhang, "Gradient descent based optimization algorithms for deep learning models training," *arXiv preprint arXiv:1903.03614*, 2019.

[4] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in neural information processing systems*, 2014, pp. 1269–1277.

[5] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[6] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 1943–1955, 2016.

[7] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," *arXiv preprint arXiv:1412.7580*, 2014.

[8] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *International Conference on Learning Representations*, 2015.

[9] D. J. Im, M. Tao, and K. Branson, "An empirical analysis of the optimization of deep network loss surfaces," in *International Conference for Learning Representations (ICLR)*, 2017.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[11] L. Bottou, "Online learning in neural networks," D. Saad, Ed. Cambridge University Press, 1998, ch. Online Learning and Stochastic Approximations, pp. 9–42. [Online]. Available: http://dl.acm.org/citation.cfm?id=304710.304720

[12] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013.

[13] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[14] G. Hinton, N. Srivastava, and K. Swersky, "Lecture 6a overview of mini–batch gradient descent."

[15] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference for Learning Representations (ICLR)*, 2014.

[16] K. Ogata, *Discrete-time control systems*. Prentice Hall Englewood Cliffs, NJ, 1995, vol. 2.

[17] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM Journal on Optimization*, vol. 26, no. 1, pp. 57–95, 2016.

[18] L. Wang, T. J. D. Barnes, and W. R. Cluett, "New frequency-domain design method for pid controllers," *IEEE Control Theory and Applications*, Jul 1995.

[19] K. Heong Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," vol. 13, pp. 559 – 576, 08 2005.

[20] A. L. Salih, M. Moghavvemi, H. A. F. Mohamed, and K. S. Gaeid, "Modelling and pid controller design for a quadrotor unmanned air vehicle," in *IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, vol. 1, May 2010, pp. 1–5.

[21] P. Rocco, "Stability of pid control for industrial robot arms," *IEEE transactions on robotics and automation*, 1996.

[22] P. Zhao, J. Chen, Y. Song, X. Tao, T. Xu, and T. Mei, "Design of a control system for an autonomous vehicle based on adaptive-pid," *International Journal of Advanced Robotic Systems*, vol. 9, no. 2, p. 44, 2012. [Online]. Available: https://doi.org/10.5772/51314

[23] P. S. de Laplace, *Théorie analytique des probabilités*. Courcier, 1820, vol. 7.

[24] W. An, H. Wang, Q. Sun, J. Xu, Q. Dai, and L. Zhang, "A pid controller approach for stochastic optimization of deep networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[25] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[26] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan, "Hcp: A flexible cnn framework for multi-label image classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 9, pp. 1901–1907, Sept 2016.

[27] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, Jan 2016.

[28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, June 2017.

[29] W. Ouyang, X. Zeng, X. Wang, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, H. Li, K. Wang, J. Yan, C. C. Loy, and X. Tang, "Deepid-net: Object detection with deformable part based convolutional neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 7, pp. 1320–1334, July 2017.

[30] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[33] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 2016, pp. 1120–1128. [Online]. Available: http://jmlr.org/proceedings/papers/v48/arjovsky16.html

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *International Conference for Learning Representations (ICLR)*, 2015.

[35] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[36] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4340–4349.

[37] X. Dai, H. Yin, and N. Jha, "Nest: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Transactions on Computers*, 2019.

[38] X. Du, Z. Li, and Y. Cao, "Cgap: Continuous growth and pruning for efficient deep learning," *arXiv preprint arXiv:1905.11533*, 2019.

[39] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[40] R. Kidambi, P. Netrapalli, P. Jain, and S. M. Kakade, "On the insufficiency of existing momentum schemes for stochastic optimization," in *International Conference for Learning Representations (ICLR)*, 2018.

[41] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[43] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *IEEE European Conference on Computer Vision (ECCV)*, 2016.

[45] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *BMVC*, 2016.

[46] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Soviet Mathematics Doklady*, 1983.

[47] J. C. Maxwell, "On governors," *Proceedings of the Royal Society of London*, vol. 16, pp. 270–283, 1867.

[48] N. Minorsky, "Directional stability of automatically steered bodies," *Journal of ASNE*, 1922.

[49] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[50] M. R. Spiegel, *Advanced mathematics*. McGraw-Hill, Incorporated, 1991.

[51] K. DE JONG, "An analysis of the behavior of a class of genetic adaptive systems," *Doctoral Dissertation, University of Michigan*, 1975.

[52] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *trans. ASME*, vol. 64, no. 11, 1942.

[53] G. E. Robert and H. Kaufman, *Table of Laplace transforms*. Saunders, 1966.

[54] H. K. Khalil, *Noninear Systems*. Prentice-Hall, New Jersey, 1996.

[55] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," 2015.

[56] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[57] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

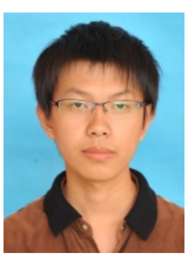
[58] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[59] T. Cooijmans, N. Ballas, C. Laurent, and A. C. Courville, "Recurrent batch normalization," *CoRR*, vol. abs/1603.09025, 2016. [Online]. Available: http://arxiv.org/abs/1603.09025

[60] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, 2016, pp. 2657–2661. [Online]. Available: https://doi.org/10.1109/ICASSP.2016.7472159

[61] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 1019–1027.

Haoqian Wang (M'13) received the B.S. and M.E. degrees from Heilongjiang University, Harbin, China, in 1999 and 2002, respectively, and the Ph.D. degree from the Harbin Institute of Technology, Harbin, in 2005.He was a Post-Doctoral Fellow with Tsinghua University, Beijing, China, from 2005 to 2007. He has been a Faculty Member with the Graduate School at Shenzhen, Tsinghua University, Shenzhen, China, since 2008, where he has also been an Associate Professor since 2011, and the director of Shenzhen Institute of Future Media Technology. His current research interests include generative adversarial networks, video communication and signal processing.



Yi Luo is received the B.E. degree in Xidian University, Xi'an, China, in 2019. He is pursuing a master degree in Tsinghua University. He is also working as a research assistance in Graduate School at Shenzhen, Tsinghua University, Shenzhen, China. His research interests include optimization in deep learning.

Wangpeng An received the B.E. degree from Kunming University of Science and Technology in 2012. He is pursuing a master degree in Tsinghua University, supervised by Professor Qionghai Dai. His research interests include face attribute recognition, generative adversarial networks and deep learning optimization.

Qingyun Sun is current working toward the Ph.D. degree in Department of Mathematics, Stanford University. He received B.S. from School of Mathematical Sciences, Peking University, Beijing, China, in 2014. His research interests include mathematical foundation for artificial intelligence, data science, machine learning, algorithmic game theory, multi-agent decision making, optimization, and high dimensional statistics.

Jun Xu is an Assistant Professor in College of Computer Science, Nankai University, Tianjin, China. Before that, he worked as a Research Scientist in Inception Institute of Artificial Intelligence. He received the B.Sc. degree in pure mathematics and the M.Sc. degree in Information and Probability both from the School of Mathematics Science, Nankai University, Tianjin, China, in 2011 and 2014, respectively. He received the Ph.D. degree in 2018 from the Department of Computing, The Hong Kong Polytechnic University, supervised by Prof. David Zhang and Prof. Lei Zhang.

Yongbing Zhang received the B.A. degree in English and the M.S. and Ph.D degrees in computer science from the Harbin Institute of Technology, Harbin, China, in 2004, 2006, and 2010, respectively. He joined Graduate School at Shenzhen, Tsinghua University, Shenzhen, China in 2010, where he is currently an associate professor. He was the receipt of the Best Student Paper Award at IEEE International Conference on Visual Communication and Image Processing in 2015. His current research interests include signal processing, computational imaging, and machine learning.

Yulun Zhang received B.E. degree from School of Electronic Engineering, Xidian University, China, in 2013 and M.E. degree from Department of Automation, Tsinghua University, China, in 2017. He is currently pursuing the Ph.D. degree with the Department of ECE, Northeastern University, USA. He was the receipt of the Best Student Paper Award at IEEE International Conference on Visual Communication and Image Processing(VCIP) in 2015. He also won the Best Paper Award at IEEE International Conference on Computer Vision (ICCV) RLQ Workshop in 2019. His research interests include image restoration and deep learning.

Lei Zhang (M'04-SM'14-F'18) received his B.Sc. degree in 1995 from Shenyang Institute of Aeronautical Engineering, Shenyang, P.R. China, and M.Sc. and Ph.D degrees in Control Theory and Engineering from Northwestern Polytechnical University, Xi'an, P.R. China, in 1998 and 2001, respectively. From 2001 to 2002, he was a research associate in the Department of Computing, The Hong Kong Polytechnic University. From January 2003 to January 2006 he worked as a Postdoctoral Fellow in the Department of Electrical and Computer Engineering, McMaster University, Canada. In 2006, he joined the Department of Computing, The Hong Kong Polytechnic University, as an Assistant Professor. Since July 2017, he has been a Chair Professor in the same department. His research interests include Computer Vision, Image and Video Analysis, Pattern Recognition, and Biometrics, etc. Prof. Zhang has published more than 200 papers in those areas. As of 2019, his publications have been cited more than 38,000 times in literature. Prof. Zhang is a Senior Associate Editor of IEEE Trans. on Image Processing, and an Associate Editor of SIAM Journal of Imaging Sciences and Image and Vision Computing, etc. He is a Clarivate Analytics Highly Cited Researcher from 2015 to 2018. More information can be found in his homepage http://www4.comp.polyu.edu.hk/cslzhang/.