A PID Controller Approach for Stochastic Optimization of Deep Networks

Wangpeng An[1,2], Haoqian Wang[1,3], Qingyun Sun[4], Jun Xu[2], Qionghai Dai[1,3], and Lei Zhang *[2]

[1]Graduate School at Shenzhen, Tsinghua University, Shenzhen, China
[2]Dept. of Computing, The Hong Kong Polytechnic University, Hong Kong, China.
[3]Shenzhen Institute of Future Media Technology, Shenzhen, China
[4]Stanford University, CA, USA
[1]anwangpeng@gmail.com, [1]wanghaoqian@tsinghua.edu.cn, [2]cslzhang@comp.polyu.edu.hk

Abstract

Deep neural networks have demonstrated their power in many computer vision applications. State-of-the-art deep architectures such as VGG, ResNet, and DenseNet are mostly optimized by the SGD-Momentum algorithm, which updates the weights by considering their past and current gradients. Nonetheless, SGD-Momentum suffers from the overshoot problem, which hinders the convergence of network training. Inspired by the prominent success of proportional-integral-derivative (PID) controller in automatic control, we propose a PID approach for accelerating deep network optimization. We first reveal the intrinsic connections between SGD-Momentum and PID based controller, then present the optimization algorithm which exploits the past, current, and change of gradients to update the network parameters. The proposed PID method reduces much the overshoot phenomena of SGD-Momentum, and it achieves up to 50% acceleration on popular deep network architectures with competitive accuracy, as verified by our experiments on the benchmark datasets including CIFAR10, CIFAR100, and Tiny-ImageNet.

1. Introduction

Benefitting from the availability of large-scale visual datasets such as ImageNet [1], deep neural networks (DNN), especially deep convolutional neural networks (CNNs), have significantly improved the system accuracy in many computer vision problems, such as image classification [2], object detection [3], and face recognition [4], etc. Despite the great successes of deep learning, the training of deep networks on large-scale datasets is usually computationally expensive, costing several days or even weeks using GPU equipped high-end PCs. It is substantially important to investigate how to accelerate the training speed of deep models without sacrificing the accuracy, which can save the time and memory cost, particularly for resource limited applications.

The key component of DNN training is the optimizer, which defines how the millions or even billions of parameters of a deep model are updated. The learning rate is one of the most important hyper-parameters to train a DNN [5]. Based on how the learning rate is set, deep learning optimizers can be categorized into two groups, hand-tuned learning rate optimizers such as stochastic gradient descent (SGD) [6], SGD Momentum [7] and Nesterov′s Momentum [7], and auto learning rate optimizers such as AdaGrad [8], RMSProp [9] and Adam [10], etc. Auto learning rate optimizers adaptively tune an individual learning rate for each parameter. Such a goal of fine adaptation is attractive and it is expected to yield better deep model learning results. However, the recent findings by Wilson et al. [11] show that hand-tuned SGD-Momentum achieves better result at the same speed or even faster speed. The hypothesis put forth here is that adaptive methods may converge to different local minima [12]. It is also noted that most of the best-performance deep models such as ResNet [13] and DenseNet [14] are usually trained by SGD-Momentum.

The strategy of SGD-Momentum is to consider both the past and present gradients to update the network parameters. However, SGD-Momentum suffers from the overshoot problem [15], which refers to the phenomena that a weight′s value exceeds much its target value and does not change its update direction. Such an overshoot problem hinders the

*Corresponding author. This work is supported by HK RGC GRF grant (PolyU 152135/16E), the NSFC fund (61571259, 61531014), Shenzhen Science and Technology Project under Grant (GGFW2017040714161462, JCYJ20170307153051701) and the National High-tech *R&D* Program of China (863Program, 2015*AA*015901).

convergence of SGD-Momentum, and costs more training time and resources. It is of significant importance to investigate whether we can design a new DNN optimizer which is free of overshoot problem and has faster convergence speed while maintaining good accuracy.

It has been found that many optimization algorithms popularly employed in machine learning studies share certain similarity to those classic control methods studied since 1950s [16]. In literature of automatic control, the feedback control system plays a key role, while the proportional-integral-derivative (PID) controller is the most commonly used feedback control mechanism due to its simplicity, functionality, and broad applicability [17]. More than 90% of industrial controllers are implemented based on PID [18], including self-driving car [19], unmanned flying vehicles [20], robotics [21], etc. The basic idea of PID control is that the control action should be proportional to the current error (the difference between system output and desired output), the integral of the past error over time, and the derivative of the error, which represents future trend.

Though PID controller has gained massive successes in different industries of control and automation, little study has been done on its connections with stochastic optimization, as well as its potential applications to DNN training. In this paper, we make the first attempt along this line. We first bridge the gap between PID controller and stochastic optimization methods such as SGD, SGD-Momentum and Nesterov′s Momentum, and consequently develop a PID approach for DNN optimization. Compared with SGD-Momentum which utilizes the past and current gradients, the proposed PID optimization approach also utilizes the gradient changes to update the network. We further introduce the Laplace Transform [22] to initialize the hyper-parameter introduced in our method, resulting in a simple yet effective stochastic DNN optimization algorithm. The major contributions of this work are summarized as follows.

- By linking the calculation of errors in feedback control system and the calculation of gradient in network updating, we reveal the intrinsic connections between deep network optimization and feedback system control, and show that SGD-Momentum is a special case of PID controller with only proportional (P) and integral (I) components.
- We then propose a PID approach to optimize DNN by utilizing the present, past and changing information of the gradient. The classical Laplace Transform is introduced to understand and initialize the hyper-parameter in our algorithm.
- We systematically evaluate the proposed approach, and the extensive experiments on CIFAR10, CIFAR100 and Tiny-Imagenet datasets demonstrate the efficiency and effectiveness of our PID approach.

The rest of this paper is organized as follows. Section 2 briefly reviews related work. Section 3 connects PID controller with DNN optimization. Section 4 introduces the proposed PID approach for DNN optimization. Experimental results and detailed analysis are reported in Section 5. Section 6 concludes this paper.

2. Related Work

2.1. Deep Learning Optimization

The learning rate is the most important hyper-parameter to train deep neural networks [9]. Based on how the learning rate is set, two classes of deep learning optimization methods can be categorized. The first class indicates fixed learning rate methods such as SGD [6], SGD Momentum [7], and Nesterov′s Momentum [7], etc., and the second class includes auto learning rate methods, such as AdaGrad [8], RMSProp [9], and Adam [10], etc. Our work is based on fixed learning rate methods considering that the current state-of-the-art results on CIFAR10, CIFAR100, ImageNet, PASCAL VOC and MS COCO datasets were mostly obtained by Residual Neural Networks [13, 14, 23, 24] trained by use of SGD Momentum.

Stochastic Gradient Descent (SGD) [6] is a widely used optimization algorithm for machine learning in general, especially for deep learning. SGD usually uses a fixed learning rate. This is because the SGD gradient estimator introduces a source of noise (the random sampling of m training examples), and that noise does not vanish even when the loss arrives at a minimum.

SGD Momentum [7] is designed to accelerate learning, especially in the case of small and consistent gradients. The momentum algorithm accumulates an exponentially decayed moving average of past gradients and continues to move in the consistent direction. The name momentum derives from a physical analogy, in which the negative gradient is a force moving a particle through parameter space. A hyper-parameter $\alpha \in (0, 1)$ determines how much the past gradients to the current update of the weights.

Nesterov′s Momentum [7] is a variant of the momentum algorithm that was motivated by Nesterov′s accelerated gradient method [25]. The difference between Nesterov momentum and regular momentum lies on where the gradient is evaluated. With Nesterov′s momentum, the gradient is estimated after the current velocity is applied. Thus one can interpret Nesterov′s momentum as attempting to add a correction factor to the standard method of momentum. Recently, Nesterov′s Momentum method has been characterized as a second order ordinary differential equation in the small step limit [26].

2.2. PID Controller

The PID controller exploits the present, past and future information of prediction error to control a feedback system [18]. PID based controller originates in the 19th century for speed control. The theoretical foundation for the operation of PID was first described by Maxwell in 1868 in his seminal paper "On Governors" [27]. Minorsky [28] then gave this a mathematical formulation. Over the years, many advanced control algorithms have also been proposed. However, most industrial controllers are implemented with a PID algorithm because it is simple, robust and easy to use [29]. A PID controller continuously calculates an error $e(t)$, which is the difference between the desired optimal output and a measured system output, and applies a correction $u(t)$ to the system based on the proportional (P), integral (I), and derivative (D) terms of $e(t)$. Mathematically, there is:

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

where K_p, K_i and K_d are the gain coefficients on the P, I and D terms, respectively.

One can see that the error $e(t)$, defined as the difference between the desired value and the actual output, has the same spirit as the gradient used in deep learning optimization. The coefficients K_p, K_i and K_d determine the contributions of present, past and future errors to the current correction. Such analyses inspire us to adapt the PID control techniques to the field of deep network optimization. To the best of our knowledge, we are the first to introduce the idea of PID into the field of deep learning as a new optimizer. As we will see later in this paper, the proposed optimizer inherits fantastic advantages of PID controller and stays simple and efficient.

3. PID and Deep Network Optimization

In this section, we disclose the connections between PID control and SGD based deep optimization. Such connections motivate us to propose a new optimization method to accelerate the training of DNNs. Updating the weights in a deep network can be viewed as deploying many PID controllers to drive the system to reach an equilibrium.

3.1. General Connections

In Figure 1, we show the flowchart of a PID controller based feedback control system, and the flowchart of SGD-Momentum based DNN optimization. The goal of a control system is to measure the output system status consecutively and update it to the desired status by using a control unit. In feedback control, the output will affect the input quantity, and the controller will make appropriate updates of the system status based on the error $e(t)$ between the measured system status and the desired status. To reach this goal, the PID controller computes a control variable $u(t)$ based on the current, past and future (i.e., derivative) of the error $e(t)$, as shown in Eq. (1).

Deep learning aims to learn an approximation function or mapping function f with parameters θ to map the input x to the desired output y, i.e., $y = f(x,\theta)$, assuming that there are (complex) relationships or causality between x and y. With enough training data, deep learning can train a network with millions of parameters (weights w) to fit those complex relationships which cannot be formulated using analytical functions. Usually, a loss function L will be defined based on the desired output y and the predicted output $f(x,\theta)$ to measure whether the goal is reached. The loss affects the weights by performing "backward propagation of errors" [30]. That is, it distributes the error to each node by calculating the gradients of weights. If the loss L is not small enough, the network will update its weights θ based on the gradients $\partial L/\partial\theta$. Therefore, it is reasonable to associate the "error" in PID control with the "gradient" in DL. This procedure is iterated till L converges or is small enough. Many optimizers have been proposed to minimize the loss L by updating θ using the gradients $\partial L/\partial\theta$, including SGD, SGD-Momentum, Adam, etc.

From the above discussions, we can see that deep network optimization shares high similarity to PID based control. Both of them update the system/network based on the difference/loss between actual output and desired output. The feedback in PID control corresponds to the back-propagation in network optimization. The major difference is that the PID controller computes the update using system error $e(t)$, while deep network optimizers determines the updates based on gradient $\partial L/\partial\theta$. If we view gradient $\partial L/\partial\theta$ as the incarnation of error $e(t)$, PID controller can be fully connected with DNN optimization. In the following, we will see that SGD, SGD-Momentum and Nesterov's Momentum all can be explained as a kind of PID controller.

3.2. SGD is a P Controller

SGD and its variants are probably the most widely used optimization algorithms for DNN optimization. The parameter update rule of SGD from time (i.e., iteration) t to time $t+1$ is given by:

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

where r is the learning rate. By viewing the gradient $\partial L_t/\partial\theta_t$ as error $e(t)$, and comparing Eq. (2) to PID controller in Eq. (1), one can see that SGD only uses the present gradient to update the weights. It is a type of P controller with $K_p = r$.

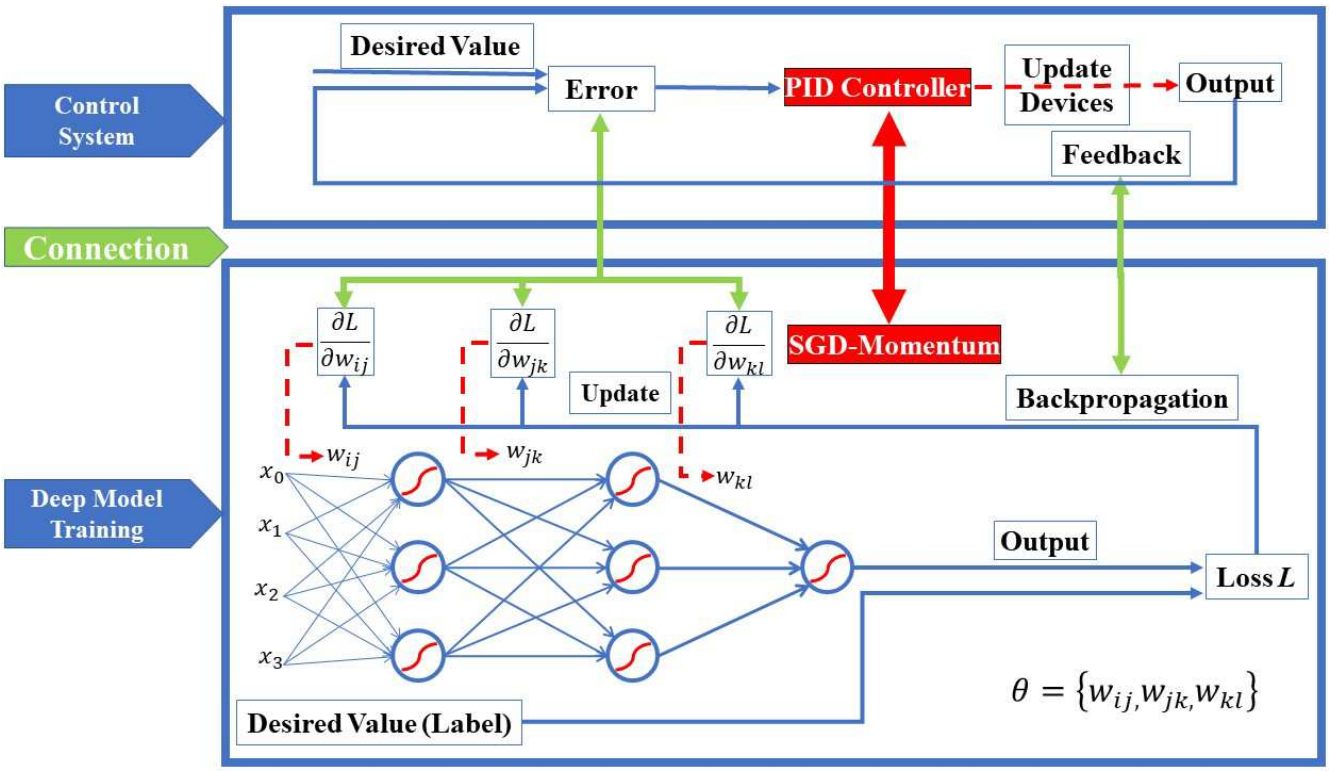


Figure 1. The connection between control system and deep model training, and the connection between PID controller and SGD-Momentum.

3.3. Momentum Optimization is a PI Controller

SGD-Momentum is able to reach the objective more quickly than SGD along the small but consistent directions, resulting in a faster convergence speed. Its parameter update rule 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 the accumulation of history gradient, and $\alpha \in (0,1)$ is the rate of moving average decay.

With some mathematical tricks (Sum Formula for a Sequence of Numbers [31]), we can remove V_t from Eq. (3), and rewrite the update rule as:

$$\theta_{t+1} = \theta_t - r\partial L_t/\partial \theta_t - r(\sum_{i=0}^{t-1}(\partial L_i/\partial \theta_i \alpha^{t-i})). \tag{4}$$

One can see that the update of parameters relies on both the present gradient ($r\partial L_t/\partial \theta_t$) and the integral of past gradients $r\sum_{i=0}^{t-1}(\partial L_i/\partial \theta_i \alpha^{t-i})$. The only difference is that there is a decay term α in the I term. This difference is because deep learning algorithms use a mini-batch of training examples to compute the gradient, and thus the gradients are stochastic. The introduction of decay term α is to forget the gradients far away from present to reduce noise. Overall, SGD-Momentum can be viewed as a PI controller.

3.4. Nesterov's Momentum Optimization is a PI Controller with larger P

The Nesterov′s Momentum update rule is given by:

$$\begin{cases} V_{t+1} = \alpha V_t - r\partial L_t/\partial(\theta_t + \alpha V_t) \\ \theta_{t+1} = \theta_t + V_{t+1}, \end{cases} \tag{5}$$

By using a variable transform $\hat{\theta}_t = \theta_t + \alpha V_t$, and expressing the update rule in terms of $\hat{\theta}$, we have:

$$\begin{cases} V_{t+1} = \alpha V_t - r\partial L_t/\partial \hat{\theta}_t \\ \hat{\theta}_{t+1} = \hat{\theta}_t + (1+\alpha)V_{t+1} - \alpha V_t. \end{cases} \tag{6}$$

Again, by using the Sum Formula for a Sequence of Numbers [31], we can have (the detailed derivation can be found in the supplementary file):

$$\begin{aligned} \hat{\theta}_{t+1} = \hat{\theta}_t &- r(1+\alpha)\partial L_t/\partial \hat{\theta}_t \\ &- r\alpha(\alpha^{t-i}\sum_{i=1}^{t-1}(\partial L_i/\partial \hat{\theta}_i)). \end{aligned} \tag{7}$$

One can see that like SGD-Momentum, the Nesterov′s Momentum also uses the present gradient and integral of past gradients to update the parameters, while the gain coefficient K_p is larger than that in SGD-Momentum.

4. PID based Deep Optimization

4.1. The Overshoot Problem of SGD-Momentum

From Eq. (4) and Eq. (7), one can see that the Momentum will accumulate history gradients. However, if the weights should change their descending direction, the history gradients will lag the update of weights. Such a phenomenon caused by history gradient is called overshoot, which is defined in discrete-time control systems [15] as "the maximum peak value of the response curve measured from the desired response of the system". Mathematically, it is defined as:

$$\textbf{Overshoot} = \frac{\theta_{max} - \theta^*}{\theta^*}, \tag{8}$$

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

One commonly used test benchmark of overshoot is the first function of De Jong′s [32] because it is smooth, unimodal, and symmetric. The function can be defined as $f(x) = 0.1x_1^2 + 2x_2^2$, whose search domain is $-10 \leq x_i \leq 10, i = 1,2$. There is no local minimum but a global minimum of this function: $x^* = (0,0), f(x^*) = 0$.

We add a derivative (change of gradient) term to SGD-Momentum to build a simple PID optimizer:

$$PID = Momentum + K_d(\partial f(x)/\partial x_c - \partial f(x)/\partial x_{c-1}), \tag{9}$$

where c is the current iteration number for x. The simulation results by setting different values of K_d in Eq. (9) are illustrated in Figure 2. The background is the loss-contour map; the redder, the bigger the loss value is, and the bluer, the smaller the loss 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. The yellow line shows the optimization route of SGD-Momentum, and the red line shows the route of PID optimizer. One can see that SGD-Momentum has obvious overshoot problem. With the K_d set to 0.1, 0.5 and 0.93, respectively, the PID optimizer exploits more "future" error (the change of gradients), and largely reduces the overshoot problem.

4.2. PID Optimizer for DNN

The toy example in Section 4.1 motivates us to propose a PID optimizer to accelerate the training of DNN. As we show in Eq. (4), SGD-Momentum is actually a PI controller which uses present and past gradient information. By adding a derivative term to introduce the future information, a PID controller can effectively reduce the overshoot problem, as shown in Figure 2. Considering that the training of deep models is usually in a mini-batch based manner, which may introduce noise in the computing of gradients, we also compute the moving average of the derivative part. The proposed PID optimizer updates parameter θ at iteration $(t+1)$ by:

$$\begin{cases} V_{t+1} = \alpha V_t - r\partial L_t/\partial \theta_t \\ D_{t+1} = \alpha D_t + (1-\alpha)(\partial L_t/\partial \theta_t - \partial L_{t-1}/\partial \theta_{t-1}) \\ \theta_{t+1} = \theta_t + V_{t+1} + K_d D_{t+1}. \end{cases} \tag{10}$$

As can be seen from Eq. (10), however, our optimizer introduces a hyperparameter K_d compared with SGD-Momentum. Fortunately, this hyper-parameter K_d can be well initialized by employing the theory of Laplace Transform [22] with Ziegler-Nichols [33] tuning method, as we describe in the following section.

4.3. Initialization of Hyper-parameter K_d

The Laplace Transform converts the function of real variable t (time) to a function of complex variable s (frequency). Denote by $F(s)$ the Laplace transform of $f(t)$. There is

$$F(s) = \int_0^\infty e^{-st} f(t)\, dt, \quad \text{for } s > 0. \tag{11}$$

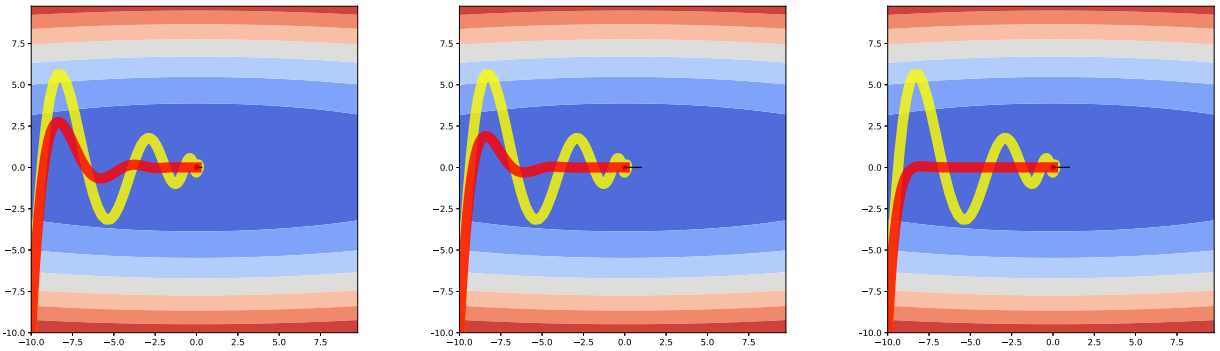
Usually $F(s)$ is easier to solve than $f(t)$, and $f(t)$ can be recovered from $F(s)$ by the Inverse Laplace transform:

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

where γ is a real number and i is the unit of imagery part. In practice, we could decompose a Laplace transform into known transforms of functions in the Laplace table [34], which includes most of the commonly used Laplace transforms, and then construct the inverse transform. With Laplace Transform, we can convert the PID optimizer into its Laplace transformed functions of s, and then simplify the algebra. Once we find the transformed solution of $F(s)$, we can inverse the transform to obtain the required solution f as a function of t.

A weight of a deep model node is initialized as a scalar θ_0, and it is updated iteratively to reach its optimal value denoted by θ^*. Then the optimization of each weight in DNN can be simplified as a step response (from θ_0 to θ^*) in control theory. We can use the Laplace Transform as a guide to set K_d. Denote by $\theta(t)$ the time domain change of weight θ. After some mathematical derivation (please refer to our supplementary file for the detailed derivation process), we have:

$$\theta(t) = \theta^* - \frac{(\theta^* - \theta_0)\sin(\omega_n\sqrt{1-\zeta^2}t + \arccos(\zeta))}{e^{\zeta\omega_n t}\sqrt{1-\zeta^2}}, \tag{12}$$



Small K_d Moderate K_d Big K_d

Figure 2. The overshoot problem of momentum. The red and yellow lines are the results obtained by PID and SGD-Momentum, respectively.

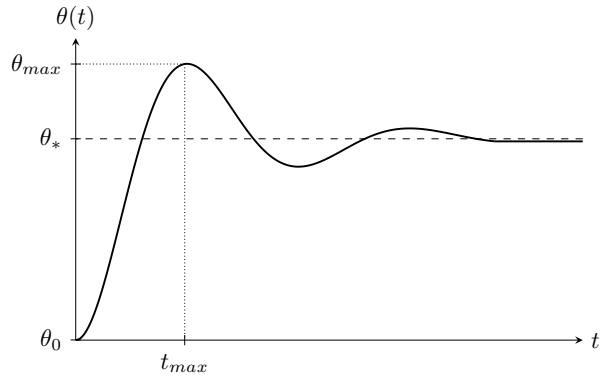


Figure 3. The evolution of the weight by PID optimizer

and

$$\begin{cases} (K_p+1)/K_d = 2\zeta\omega_n \\ K_i/K_d = \omega_n^2 \end{cases}, \tag{13}$$

where ζ and ω_n are damping ratio and natural frequency of the system, respectively. In Figure 3, we show the evolution process of a weight as an example of $\theta(t)$. From Eq. (13), we have $K_i = \frac{(K_p+1)^2}{4K_d\zeta}$. One can see that K_i is a monotonically decreasing function of ζ. Refer to the definition of overshoot in Eq. (8), one can see that ζ is monotonically decreasing with overshoot. Then K_i is a monotonically increasing function of overshoot. So more history error (Integral part), more overshoot the system will have. That is the reason why SGD-Momentum which accumulates past gradients will overshoot its target and spend more time during training.

As can be observed from Eq. (12), 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. One should note the value of hyper-parameter K_d in calculating the derivate $e^{-\zeta\omega_n} = e^{-\frac{K_p+1}{2K_d}}$. It is easy to observe that the larger the derivate, the earlier the training convergence we will reach. However, when K_d gets too large, the system will be fragile. In practice, we set the hyper-parameter K_d based on the Ziegler-Nichols optimum setting rule [33], which is widely used by engineers in PID feedback control since its origin in 1940s.

According to Ziegler-Nichols' rule, the ideal setup of K_d should be one third of the oscillation period, which means $K_d = \frac{1}{3}T$, where T is the period of oscillation. From Eq. (12), 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. (13), K_d will have a closed form solution:

$$K_d = 0.25r + 0.5 + (1 + \frac{16}{9}\pi^2)/r \tag{14}$$

In practice, we can start with this ideal setting of K_d and change it slightly when use different network models to train on different datasets.

5. Experimental Results

In this section, we first trained an MLP on the MNIST handwritten digit dataset in Section 5.2 to show the advantage of PID optimizer, and then trained CNNs on the CIFAR datasets in Section 5.3 to demonstrate that PID optimizer is competitive with SGD-Momentum in accuracy but with much faster training speed. To further validate our PID optimizer on a larger dataset, in Section 5.4 we performed experiments on the Tiny-Imagenet dataset [36]. The results showed that our PID optimizer can generalize to modern networks and datasets. It should be noted that except for the additional hyper-parameter K_d which is set by Eq.(14), all the other hyper-parameters in our PID optimizer are set as the same as SGD-Momentum. The learning rate starts from 0.01 and is divided by 10 when the error plateaus. The source code of our PID optimizer can be found at https://github.com/tensorboy/PIDOptimizer.

Table 1. Test errors and training epochs of PID and SGD-Momentum 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 [13]	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 [23]	164	1.7	5	**5.23**/5.46	**230**/271	**24.17**/24.33	**241**/282
ResNeXt29 [35]	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 [24]	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 [14]	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

5.1. Dataset

MNIST dataset: The MNIST dataset [37] contains 60,000 training samples and 10,000 test samples of the handwritten digits from 0 to 9. The images are of 28×28 pixels and in grey level format.

CIFAR Dataset: The CIFAR10 and CIFAR100 datasets [38] consist of 60,000 RGB color images of resolution 32×32, drawn from 10 and 100 classes, respectively, and both split into 50,000 training and 10,000 test images. For data augmentation, we performed horizontal flips and random crops on the original image padded by 4 pixels on each side.

Tiny ImageNet Dataset: The Tiny-ImageNet [36] dataset has 200 classes. Each class has 500 training images, 50 validation images, and 50 test images. The Tiny-ImageNet is more difficult than the CIFAR datasets because more classes are involved, and the relevant objects to be classified often cover only a tiny subspace of the image.

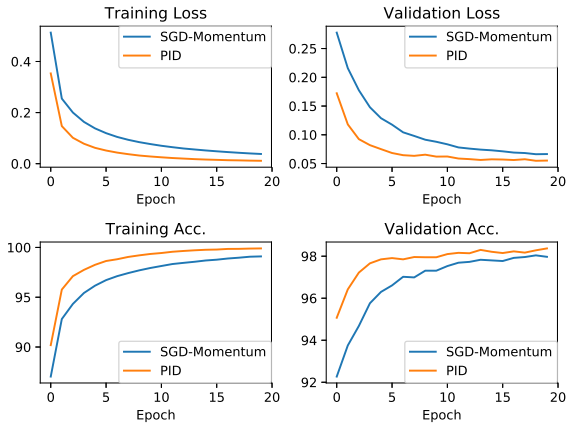


Figure 4. PID *vs.* SGD-Momentum on the MNIST dataset for 20 epochs. Top row: the curves of training loss and validation loss. Bottom row: the curves of training accuracy and validation accuracy.

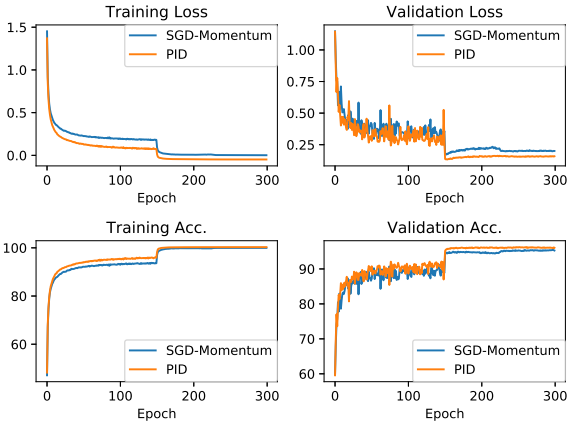


Figure 5. PID *vs.* SGD-Momentum on the CIFAR10 dataset by using DenseNet 190-40. Top row: the curves of training loss and validation loss. Bottom row: the curves of training accuracy and validation accuracy.

5.2. Results of MLP on MNIST dataset

We first trained a simple MLP network on the MNIST handwritten digit classification dataset using the proposed PID optimizer and compare it with SGD-Momentum [7]. The MLP network is with ReLU nonlinearity and 1,000 hidden nodes in the hidden layer, followed by the softmax output layer on top. The training was on mini-batches with 128 images per batch for 20 epochs through the training set. We run the experiments for 10 times and reported the average results. The detailed training statistics by the two methods are illustrated in Figure 4, from which we can see that PID optimizer not only converges more quickly than SGD-Momentum with lower loss and higher accuracy, but also has higher generalization ability on the validation dataset. On the test dataset, PID optimizer achieves 98% accuracy and SGD-Momentum achieves an accuracy of 97.5%.

5.3. Results on CIFAR datasets

We then compared PID and SGD-Momentum optimizers on CIFAR10 and CIFAR100 by using five state-of-the-art CNN models, including ResNet [13], PreActResNet [23], ResNeXt29 [35], WRN [24], and DenseNet [14]. The results are summarized in Table 1. The second column lists the number of depth of those networks, while the third column lists the number of parameters for each network model. The fourth column indicates the number of runs to calculate the average test error. In the fifth and sixth columns of Table 1, we presented the average test errors on CIFAR10 and showed the numbers of Epochs by PID and SGD-Momentum when they achieve the reported test errors for the first time (i.e., the least number of Epochs to reach the best accuracy). The last two columns of Table 1 present such comparisons on CIFAR100.

From Table 1, we can have the following observations. First, our proposed PID optimizer achieves lower test errors than SGD-Momentum for all the used CNN architectures on both the two CIFAR datasets, except for ResNet with depth 1202. Second, PID optimizer converges faster (with less Epochs) than SGD-Momentum to reach the best results. In particular, our PID optimizer has on average 35% and up to 50% acceleration compared with SGD-Momentum. This demonstrates the importance of the change of gradient, which can be exploited to reduce the overshoot problem and speed up the learning process of DNNs. Figure 5 shows the detailed training statistics by the two methods on CIFAR10 with DenseNet 190-40 (190 layers with growth rate of 40) [14]. One can see that PID optimizer converges faster than SGD-Momentum with lower loss and higher accuracy.

5.4. Experiments on Tiny-ImageNet

To further demonstrate the effectiveness of our PID optimizer, we employed the DenseNet190-40 architecture to perform experiments on the Tiny-ImageNet dataset. Figure 6 shows the curves of training loss and accuracy over Epochs, as well as the validation loss and accuracy by the PID and SGD-Momentum optimizers. The learning rate of SGD-Momentum and PID was fixed to 0.01. Training was conducted 150 epochs using batch size 64. The results are averaged over 5 runs.

Similar conclusions to those on CIFAR datasets can be made. In both training and validation, PID converges faster than SGD-Momentum, has lower loss and achieves higher accuracy. Such results confirm the generalization capability of PID based DNN optimizer to large-scale datasets.

6. Conclusion

Inspired by the prominent success of PID controller in the field of automatic control, we investigated its connections with stochastic optimizers such as SGD and its variants, and presented a novel PID controller approach to deep network optimization. The proposed PID optimizer exploits the present, past and change information of gradients to update the network parameters, reducing greatly the overshoot problem of SGD-momentum and accelerating the learning process of DNNs. Our experiments on MINIST, CIFAR and Tiny-ImageNet datasets validated that the proposed PID optimizer is 30% $\sim$ 50% faster than SGD-Momentum, whiling resulting in lower error rate. In future work, we will investigate how to adapt our PID optimizer to other network architectures such as LSTM and RNN, and how to associate PID optimizer with an adaptive learning rate for DNN optimization.

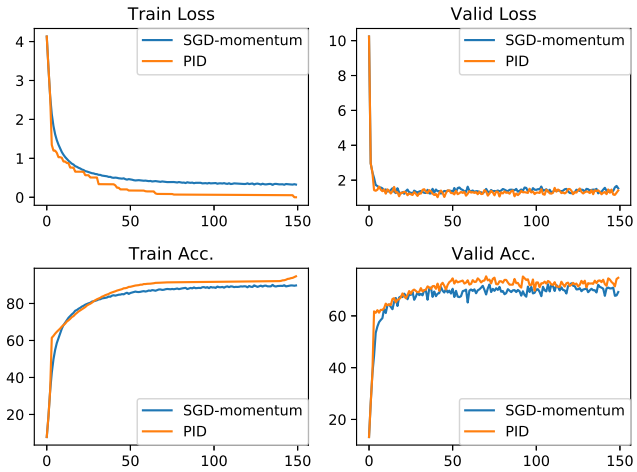


Figure 6. PID *vs.* SGD-Momentum on the Tiny-imagenet dataset by using DenseNet 190-40. Top row: the curves of training loss and validation loss. Bottom row: the curves of training accuracy and validation accuracy.

References

[1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei fei Li. Imagenet large scale visual recognition challenge. *IEEE International Journal of Computer Vision (IJCV)*, 2015. 1

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)*, 2012. 1

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems (NIPS)*, 2015. 1

[4] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clus-

tering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. 1

[6] Léon Bottou. Online learning in neural networks. chapter Online Learning and Stochastic Approximations, pages 9–42. Cambridge University Press, 1998. 1, 2

[7] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, 2013. 1, 2, 7

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 1, 2

[9] Geoffrey Hinton, N Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent. 1, 2

[10] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*, 2014. 1, 2

[11] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. 1

[12] Daniel Jiwoong Im, Michael Tao, and Kristin Branson. An empirical analysis of the optimization of deep network loss surfaces. In *International Conference for Learning Representations (ICLR)*, 2017. 1

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1, 2, 7, 8

[14] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 2, 7, 8

[15] Katsuhiko Ogata. *Discrete-time control systems*, volume 2. Prentice Hall Englewood Cliffs, NJ, 1995. 1, 5

[16] Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016. 2

[17] 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. 2

[18] Kiam Heong Ang, G Chong, and Yun Li. Pid control system analysis, design, and technology. 13:559 – 576, 08 2005. 2, 3

[19] Pan Zhao, Jiajia Chen, Yan Song, Xiang Tao, Tiejuan Xu, and Tao Mei. Design of a control system for an autonomous vehicle based on adaptive-pid. *International Journal of Advanced Robotic Systems*, 9(2):44, 2012. 2

[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)*, volume 1, pages 1–5, May 2010. 2

[21] Paolo Rocco. Stability of pid control for industrial robot arms. *IEEE transactions on robotics and automation*, 1996. 2

[22] Pierre Simon de Laplace. *Théorie analytique des probabilités*, volume 7. Courcier, 1820. 2, 5

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *IEEE European Conference on Computer Vision (ECCV)*, 2016. 2, 7, 8

[24] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016. 2, 7, 8

[25] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, 1983. 2

[26] Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov′s accelerated gradient method: Theory and insights. In *Advances in Neural Information Processing Systems (NIPS)*, 2014. 2

[27] J Clerk Maxwell. On governors. *Proceedings of the Royal Society of London*, 16:270–283, 1867. 3

[28] Nicolas Minorsky. Directional stability of automatically steered bodies. *Journal of ASNE*, 1922. 3

[29] Emre Sariyildiz, Haoyong Yu, and Kouhei Ohnishi. A practical tuning method for the robust pid controller with velocity feed-back. *Machines*, 3(3):208–222, 2015. 3

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 1986. 3

[31] Murray R Spiegel. *Advanced mathematics*. McGraw-Hill, Incorporated, 1991. 4

[32] KA DE JONG. An analysis of the behavior of a class of genetic adaptive systems. *Doctoral Dissertation, University of Michigan*, 1975. 5

[33] John G Ziegler and Nathaniel B Nichols. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942. 5, 6

[34] George E Robert and Hyman Kaufman. *Table of Laplace transforms*. Saunders, 1966. 5

[35] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 7, 8

[36] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. 2015. 6, 7

[37] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 7

[38] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 7