SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

May 2021, Vol. 64 150102:1–150102:15
https://doi.org/10.1007/s11432-020-3163-0

Special Focus on Constraints and Optimization in Artificial Intelligence

Learning dynamics of gradient descent optimization in deep neural networks

Wei WU[1*], Xiaoyuan JING[1*], Wencai DU[2] & Guoliang CHEN[3]

[1]*School of Computer Science, Wuhan University, Wuhan* 430072*, China;*
[2]*Institute of Data Science, City University of Macau, Macau* 999078*, China;*
[3]*College of Computer Science and Software Engineering, Shenzhen University, Shenzhen* 518060*, China*

Received 26 April 2020/Revised 22 August 2020/Accepted 19 November 2020/Published online 8 April 2021

Abstract Stochastic gradient descent (SGD)-based optimizers play a key role in most deep learning models, yet the learning dynamics of the complex model remain obscure. SGD is the basic tool to optimize model parameters, and is improved in many derived forms including SGD momentum and Nesterov accelerated gradient (NAG). However, the learning dynamics of optimizer parameters have seldom been studied. We propose to understand the model dynamics from the perspective of control theory. We use the status transfer function to approximate parameter dynamics for different optimizers as the first- or second-order control system, thus explaining how the parameters theoretically affect the stability and convergence time of deep learning models, and verify our findings by numerical experiments.

Keywords learning dynamics, deep neural networks, gradient descent, control model, transfer function

Citation Wu W, Jing X Y, Du W C, et al. Learning dynamics of gradient descent optimization in deep neural networks. Sci China Inf Sci, 2021, 64(5): 150102, https://doi.org/10.1007/s11432-020-3163-0

1 Introduction

Deep neural networks (DNNs) are well applied to solve recognition problems of complex data including image, voice, text, and video, due to their high-dimensional computing capability. Generally, applying a DNN model to an engineering issue has three steps: analyzing historical data and initializing a proper model, training parameters for the model, and applying input data to compare network output to real values. A serious defect is that the status of a deep learning model with a predefined structure and limited training may not be dynamically stable to new inputs. The system stability cannot be guaranteed. It may produce an abnormal output with unexpected errors, which can cause catastrophic damage in application.

The stochastic gradient descent (SGD) method and many SGD improvements aim to quickly and accurately solve parameter optimization, so as to reduce the error between a model output and a desired value, i.e., error backpropagation processing. The incomparable ability for spatial feature representation benefits from the parameters of massive neuron nodes and interconnected weights inside. To obtain an excellent model parameter distribution is the priority of many optimization methods, while they ignore the dynamics of how parameters converge.

To avert potential threats to DNNs, we explain the mathematical principles from the control perspective. Benefiting by introducing the transfer function of the control model, we analyze how to keep a multiple-parameter-based optimizer working in an effective way.

2 Related work

Gradient descent optimization algorithms are often used in DNN parameter learning. It is an error-based updating method for all neurons and weights. To handle different problems, DNNs have a variety of structures regarding connections and weights. Multilayer perception (MLP) is a common feedforward

*Corresponding author (email: wuweiux@163.com, jingxy_2000@126.com) form. The convolutional neural network (CNN) uses a weight-sharing mechanism to process matrix data. The recurrent neural network (RNN) and long short-term memory (LSTM) indicate good effects in time series data analysis. The gradient descent learning method is generally the optimization foundation throughout the training stage.

The gradient descent strategy is used as a black-box optimizer [1], which requires sophisticated training parameter settings, and the fine-tuning process is long and costly when tackling complex deep networks.

Much work focuses on improving SGD-based algorithms by adding more control variables, while many depend on expert initialization, which hampers the application of DNNs. Batch gradient descent is proposed to obtain a global optimum using all training data once for all, but it is usually computationally expensive, even using high-performance computers equipped with GPUs [2]. A depth-controllable network and training principles [3] reflect the thinking of balancing system control and computational burden. To accelerate the training speed, mini-batch gradient descent uses a subset from training instances [4], benefiting from the separation of training data into small batches; this method often reaches a good result in a few epochs. To avoid the slow step of SGD on the flat loss surface, the momentum factor [5], by making a small movement based on the error trend, is combined with the improved gradient to form a more effective vector to enhance the convergence speed, but requires a preset learning rate number properly. To weaken the influence of initialization, the Adagrad method [6] can adaptively adjust the learning rate and can globally affect the gradient descent. However, this factor amplifies the gradient in the early phase of training, subjecting the system to more vibration or instability, or reducing the gradient to a very small value in the later phase, leading the training to terminate abnormally. Adadelta [7] cancels the learning rate factor in Adagrad and improves training in the middle stage when the gradient is not too big or small, but is vulnerable to local minima. Similar to Adadelta, RMSprop [8], introduced by Hinton, is another method to accelerate convergence by dividing the learning rate as an exponentially decaying average of squared gradients. To reduce the drawbacks of big second-order moment in the training stage, Adam [9] controls the first- and second-order of momentum simultaneously on the basis of RMSProp, and AMSGrad [10] guarantees the learning rate positive in Adam. In certain cases, the big learning rate in Adam benefits fast training but may lead weak generalization capability, on the contrast, a fixed small learning rate in SGD training is time-costly but robust. As a result, the AdaBound [11] links the learning rate from adaptive method and manual SGD to achieve a balance between generalization capability and training speed. The animation figure provides an intuitive understanding of the optimization behavior of some popular algorithms[1)].

However, the transfer function of a DNN is not easily expressed due to the deep layered structure and operation in matrix form, which may bring the zero points and pole points of the system into high-dimensional orders, preventing the status space expression inferring from input data, system parameter and output from an analytical transfer function matrix. Ref. [12] presented a mathematical description of an integral form to bridge the gap between the theory and practice of deep learning of all layers. The classical methods are carried out by transfer function in the time and frequency domains, in which introducing the Lyapunov stability theory in cybernetics to study status changing pattern remains popular. For some neural network models with time-varying delay stages, specified Lyapunov functionals are proposed, and are derived in the form of linear matrix inequalities in [13]. For neutral-type neural networks including constant delay parameters, a properly modified Lyapunov functional employing Lipschitz activation functions is derived in [14].

Transforming coordinates from a standard parameter space to a high-dimensional space enables the Taylor expansion to be meaningful in singularities, providing the local influence derived from model parameters. Series work focused on the singularity area, calculates the partial derivative for each new variables, shows the dynamic variation affected by weights in the hidden layers and output layer [15]. The connection between non-convex optimization for training DNNs and nonlinear partial differential equations has been established to enhance the robustness of SGD [16].

Dynamics of how model parameters of DNN converged and how system output stabilized is a control model theoretically. A proportional, integral and derivative (PID) controller approach for stochastic optimization was proposed to simulate a DNN [2, 17]. By linking the calculation of errors in a feedback control system and the calculation of gradients in network updating, it revealed the strong connection between error-based PID control and the gradient decent method, and provided the analytical control form of a typical SGD. Transforming coordinates from a standard parameter space to a new space, to

1) http://cs231n.github.io/neural-networks-3/.

understand the influence from input data and model variables, some studies focused on the singularity area [18, 19]; the partial derivative for each new variable shows the dynamic variation near singularities affected by weights between the hidden layer and output layer. In [20], by studying the geometry induced by the kernel mapping, a multilayer kernel based on CNNs characterized the corresponding reproducing kernel Hilbert space (RKHS).

To reveal optimization methods from the perspective of control is a novel way to understand DNN characteristics, facilitating the analysis of system stability, convergence speed, and track. Thus, we can design a more compact DNN structure with fewer neurons and connections, which can reduce computing complexity. The predefined network learning rate and other factors can be initialized in a specified domain.

Our work contributes to the dynamics learning processing of DNN theoretically, from three aspects:

(1) We propose first- and second-order control models for parameter optimizers for SGD, SGD momentum, and Nesterov accelerated gradient (NAG). The proposed transfer functions benefit the understanding of deep learning optimization models from a more systemic perspective view.

(2) We analyze how the learning rate, momentum factor, and gain coefficient affect the optimizer dynamics by using the time response root locus of each control equation, reveal how the model order and element affect the system characteristics, respectively.

(3) We compare the learning dynamics of optimizers with varying parameters on some popular datasets, which validate how the system performance is affected by proper parameter combination settings.

3 Gradient descent optimization dynamics

The key to gradient descent is to approach the solution space of deep learning models. Define the mapping function $f(\cdot)$ from the input x to the desired output y^* with the model parameter set θ^*. The global computing model M^* is obtained as

$$M^* : y^* = f(x; \theta^*). \tag{1}$$

Note that x can be a scalar, vector, or matrix, and y^* is a scalar or vector, depending on the application. θ^* is a parameter set of all neurons, weights, and bias. The learning system $M : y = f(x; \theta)$ for the real system M^* has a small error δ, i.e., the distance between the algorithm result y and the real y^* is computed by

$$\mathrm{dist}\{y^*, y\} \leqslant \delta. \tag{2}$$

In different applications, we use the corresponding forms to measure δ, such as mean square error (MSE), mean absolute error (MAE), Huber loss, and log-likelihood loss.

Our goal is to train M to replace the unknown real system in limited steps in certain instances. Figure 1 shows the learning from a random initial state M_0 to M. The optimized parameter hyperspace is a subset in the high-dimensional coordinate system formed by nodes, weights, bias, and other variables, where M and M^* are approximated by the status space and real status space, respectively. The distance metric between these two spaces is δ. We show four examples of routes in Figure 1. r_1 and r_2 converge to M, and r_3 moves to M^* in diverse initializations and paths, but the final model status of r_1, r_2, and r_3 is stable. However, r_4 is going in the opposite direction, with an eventual unstable output.

SGD and its improvements are the most popular optimization algorithms for DNN parameter optimization to realize M. We discuss three main branches from the perspective of control system with transfer functions theories progressively.

Lemma 1. The transfer function from an input signal x to the desired output y is described as a differential equation (DE):

$$a_n \frac{\mathrm{d}^n y(t)}{\mathrm{d}t^n} + a_{n-1} \frac{\mathrm{d}^{n-1} y(t)}{\mathrm{d}t^{n-1}} + \cdots + a_1 \frac{\mathrm{d}y(t)}{\mathrm{d}t} + a_0 y(t) = b_m \frac{\mathrm{d}^m x(t)}{\mathrm{d}t^m} + \cdots + b_1 \frac{\mathrm{d}x(t)}{\mathrm{d}t} + b_0 x(t). \tag{3}$$

By using the Laplace transform of x and y,

$$X(s) = \int_0^\infty x(t)\mathrm{e}^{-st}\mathrm{d}t, \quad Y(s) = \int_0^\infty y(t)\mathrm{e}^{-st}\mathrm{d}t. \tag{4}$$

Eq. (3) is transformed into

$$(a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0) Y(s) = (b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0) X(s). \tag{5}$$

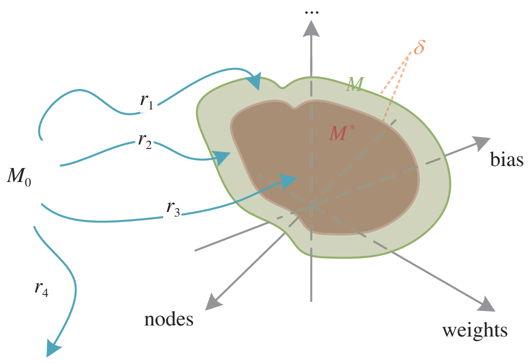

Figure 1 (Color online) Learning from random states to the acceptable parameter space M or M^*. The coordinate system is formed by a deep learning parameter set including nodes and weights, and r_1, r_2, r_3, r_4 are different learning routes.

As a result, the transfer function is obtained as

$$G(s) = \frac{Y(s)}{X(s)} = \frac{b_m s^m + b_{m-1}s^{m-1} + \cdots + b_1 s + b_0}{a_n s^n + a_{n-1}s^{n-1} + \cdots + a_1 s + a_0}. \tag{6}$$

The Laplace transform operation is a linear integral operation, which depends entirely on the structure and parameters of the system. The transfer function provides an analytical tool for dynamics processing, including stability, convergence and error control for optimizers.

3.1 SGD optimizer

The iteration of SGD from one step to the next is given by

$$\theta_{k+1} = \theta_k - r\frac{\partial L_k}{\partial \theta_k}, \tag{7}$$

where r is the learning rate, and L is the loss function from the output y to the desired y^*. The expansion of formula (7) is

$$\begin{cases} \theta_2 = \theta_1 - r\partial L_1/\partial\theta_1, \\ \theta_3 = \theta_2 - r\partial L_2/\partial\theta_2, \\ \quad\vdots \\ \theta_n = \theta_{n-1} - r\partial L_{n-1}/\partial\theta_{n-1}. \end{cases} \tag{8}$$

We obtain the summation form as

$$\theta_n = \theta_1 - r\sum_{i=1}^{n-1}\partial L_i/\partial\theta_i. \tag{9}$$

Taking the training step n_t satisfies

$$\operatorname{dist}\left(f(x,\theta_{t+1}) - y\right) \leqslant \delta. \tag{10}$$

Initializing the primitive status as $\theta_1 = 0$, we have the following approximating form from (7):

$$f(x;\theta_t) = f\left(x; -r\sum_{i=1}^{t-1}\partial L_i/\partial\theta_i\right). \tag{11}$$

The system output converges to y_t, and L reflects the qualitative impact of all historical errors on θ [17]. We initialize an amplifier K_0 as the actuator to transfer the input signal to the application, then we can obtain the transfer function of SGD.

Theorem 1. The gradient descent optimizer is a first-order control system affected by the learning rate r and proportional coefficient K_0, with the open-loop transfer function

$$G_{\theta_{\text{sgd}}}(s) = K_0 \cdot \frac{r}{s}, \quad K_0 > 0,\ r > 0. \tag{12}$$

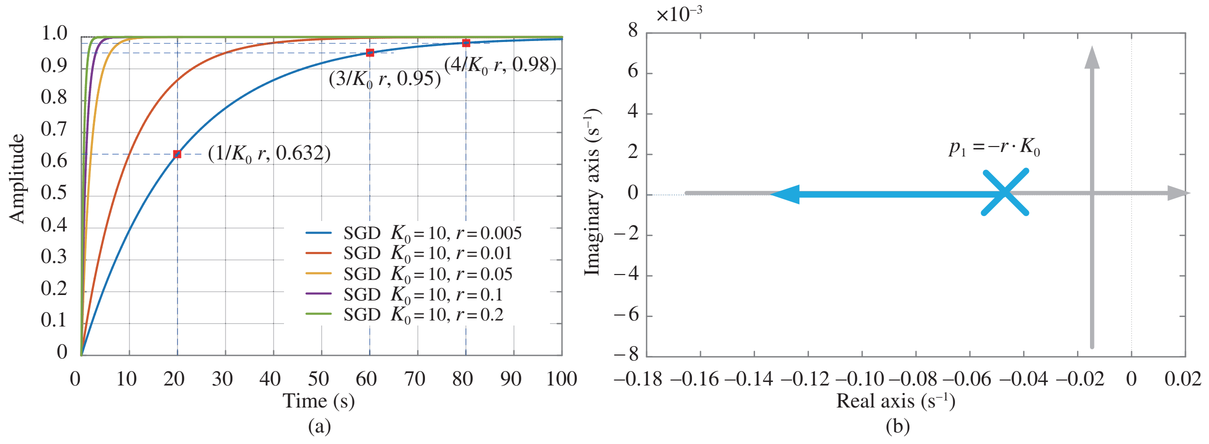

Figure 2 (Color online) (a) Step response and (b) root locus of SGD ($K_0 > 0, r > 0$).

Remark 1. The transfer function of SGD is derived from formula (9), where the amplifier K_0 is often acts as the normalization tool to shift intermediate values with limited constraint boundaries, including the RGB image channel value 0-1 normalization, output classification probability 0-1 normalization. The learning rate is a fixed value initialized by designers, the s parameter equals $\sum_{i=1}^{n-1} \partial L_i/\partial\theta_i$, viewed as integration of error factor $e(t)$ [2].

The closed-loop control system of the gradient descent (GD) model is

$$\varphi(s) = \frac{Y(s)}{X(s)} = \frac{K_0 \cdot \frac{r}{s}}{1 + K_0 \cdot \frac{r}{s}} = \frac{K_0 r}{s + K_0 r}. \tag{13}$$

We rewrite $\varphi(s)$ in the time-constant form,

$$\varphi(s) = \frac{1}{Ts+1}, \quad T = 1/K_0 r. \tag{14}$$

Under the excitation of an unit step signal, the output of the deep learning model is

$$Y(s) = \varphi(s)X(s) = \frac{1}{Ts+1} \cdot \frac{1}{s} = \frac{1}{s} - \frac{1}{s + \frac{1}{T}}. \tag{15}$$

Solving it in the time domain with a Laplace transform, we have

$$y(t) = \ell^{-1}[Y(s)] = \left[1 - \mathrm{e}^{-t/T}\right] \cdot 1(t). \tag{16}$$

This is a stable system without overshoot. The convergence time is determined by the learning rate r and proportional element K_0. The stabilized output is

$$\lim_{t\to\infty} y(t) = \left[1 - \mathrm{e}^{-t/T}\right] \cdot 1(t) = 1(t). \tag{17}$$

This first-order system gradually approaches 1. Generally, $t_s = \frac{3}{K_0 r}$ is used as the settling time corresponding to the output with $\delta \leqslant 5\%$.

We show an example of the time response and root locus diagram in Figure 2. The input x is simplified as the step signal, and y is a scalar. In Figure 2(a), the response experiences a long settling time from initialization to the final stage. Different learning rates change the speed intuitively. A smaller r needs more time to achieve a stable y. However, if we choose a greater learning rate, then the system may be fragile because of the big step to the latent optimization zone.

The root locus diagram in Figure 2(b) implies that the system is globally stable under the condition of the specified zone $(-\infty, -K_0 r)$ on the real axis, with zero in each model located at the left panel of the imaginary axis, which guarantees system convergence.

3.2 SGD momentum

Momentum is proposed to speed up the training of SGD, with a classical form [21],

$$\begin{cases} v_{k+1} = \mu v_k - r\dfrac{\partial L_k}{\partial \theta_k}, \\ \theta_{k+1} = \theta_k + v_{k+1}. \end{cases} \tag{18}$$

Dividing the first formula of (18) by μ^{k+1} [17], we have

$$\begin{cases} \dfrac{v_1}{\mu^1} = \dfrac{v_0}{\mu^0} - \dfrac{r}{\mu^1}\dfrac{\partial L_0}{\partial \theta_0}, \\ \dfrac{v_2}{\mu^2} = \dfrac{v_1}{\mu^1} - \dfrac{r}{\mu^2}\dfrac{\partial L_1}{\partial \theta_1}, \\ \quad\vdots \\ \dfrac{v_{k+1}}{\mu^{k+1}} = \dfrac{v_k}{\mu^k} - \dfrac{r}{\mu^{k+1}}\dfrac{\partial L_k}{\partial \theta_k}. \end{cases} \tag{19}$$

Summarizing (19), a simplified expression is obtained as

$$v_{k+1} = -r\sum_{i=1}^{k} \mu^{k-i}\frac{\partial L_i}{\partial \theta_i}. \tag{20}$$

Similarly, we add up each part of the second equation of (19) to obtain

$$\theta_{k+1} = \theta_1 + (v_2 + v_3 + \cdots + v_{k+1}), \tag{21}$$

where each v_i is expanded as

$$\begin{cases} v_2 = -r \cdot \left(\mu^0 \dfrac{\partial L_1}{\partial \theta_1}\right), \\ v_3 = -r \cdot \left(\mu^1 \dfrac{\partial L_1}{\partial \theta_1} + \mu^2 \dfrac{\partial L_2}{\partial \theta_2}\right), \\ v_{k+1} = -r \cdot \left(\mu^{k-1} \dfrac{\partial L_1}{\partial \theta_1} + \cdots + \dfrac{\partial L_k}{\partial \theta_k}\right). \end{cases} \tag{22}$$

We rewrite the final θ_{k+1} as

$$\theta_{k+1} - \theta_1 = -r \cdot \left(\frac{\partial L_1}{\partial \theta_1}, \frac{\partial L_2}{\partial \theta_2}, \ldots, \frac{\partial L_k}{\partial \theta_k}\right) \begin{pmatrix} \mu^0 + \mu^1 + \cdots + \mu^{k-2} + \mu^{k-1} \\ \mu^0 + \mu^1 + \cdots + \mu^{k-2} \\ \vdots \\ \mu^0 \end{pmatrix}. \tag{23}$$

In many DNN applications, the momentum optimizer keeps the training tracks more resistant, with the explanation of it considered the historical gradient impact on moving direction empirically.

Benefiting from the accumulated gradient, a directional variable reflects the partial derivative processing [21] of the gradient itself, and the momentum plays the role of an inertial element to adjust the system.

Theorem 2. The SGD momentum optimizer is a second-order control system, with the transfer function $G_{\theta_{\mathrm{sgdm}}}(s) = \frac{K_0 r}{g(\mu)s^2+s}$, formed by the learning rate r, momentum factor μ, and proportional coefficient K_0. The dynamics is determined by the oscillation frequency $\omega_n = \sqrt{\frac{K_0 r}{g(\mu)}}$ and damping coefficient $\xi = \frac{1}{\sqrt{4K_0 r g(\mu)}}$, where $g(\mu) > 0$.

Remark 2. The SGD momentum optimizer is realized by a series connection from SGD to momentum. Given the inertial element of momentum factor $G_m = \frac{1}{Ts+1}$, hence, the transfer function from SGD can be improved as

$$G_{\theta_{\rm sgdm}}(s) = G_{\theta_{\rm sgd}}(s) \cdot G_m = K_0 \cdot \frac{r}{s} \cdot \frac{1}{g(\mu)s+1} = \frac{K_0 r}{g(\mu)s^2+s}, \tag{24}$$

where $g(\mu)$ is a function of μ.

Consequently, the closed-loop transfer function of θ is obtained as

$$\varphi(s) = \frac{K_0 r}{g(\mu)s^2+s+K_0 r} = \frac{\frac{K_0 r}{g(\mu)}}{s^2+\frac{1}{g(\mu)}s+\frac{K_0 r}{g(\mu)}}. \tag{25}$$

Therefore, we have the following theorem for the momentum optimizer, by solving

$$\begin{cases} 2\xi\omega_n = \dfrac{1}{g(\mu)}, \\ \omega_n^2 = \dfrac{K_0 r}{g(\mu)}. \end{cases} \tag{26}$$

We have

$$\omega_n = \sqrt{\frac{K_0 r}{g(\mu)}}, \quad \xi = \frac{1}{\sqrt{4K_0 r g(\mu)}}, \quad T = \frac{1}{\omega_n}. \tag{27}$$

As a result, the characteristic equation is described as

$$s^2 + \frac{1}{g(\mu)}s + \frac{K_0 r}{g(\mu)} = 0. \tag{28}$$

Case 1. Let $0 < \xi < 1$, i.e., $K_0 r g(\mu) > 0.25$. Then the system has a pair of conjugate solutions,

$$-s_{1,2} = -\frac{1}{2g(\mu)} \pm {\rm j}\frac{\sqrt{4K_0 r g(\mu)-1}}{2g(\mu)}. \tag{29}$$

The system is a damped concussion with output:

$$\begin{aligned} y(t) &= 1 - \frac{1}{\sqrt{1-\xi^2}}{\rm e}^{-\xi t/T}\sin\left(\sqrt{1-\xi^2}\frac{t}{T} + \arctan\frac{\sqrt{1-\xi^2}}{\xi}\right) \\ &= 1 - \frac{2\sqrt{K_0 r g(\mu)}}{\sqrt{4K_0 r g(\mu)-1}}{\rm e}^{-\frac{1}{2g(\mu)}t}\sin\left(\frac{\sqrt{4K_0 r g(\mu)-1}}{2K_0 r}t + \arctan\sqrt{4K_0 r g(\mu)-1}\right). \end{aligned} \tag{30}$$

The dynamics described by overshoot and settling time are

$$\begin{cases} \sigma\% = \exp\left(-\dfrac{\pi}{\sqrt{4K_0 r g(\mu)-1}}\right) \times 100\%, \\ t_s \approx 6g(\mu). \end{cases} \tag{31}$$

Case 2. Let $\xi = 1$, i.e., $K_0 r g(\mu) = 0.25$. Then, the eigenroots of the characteristic equation are

$$-s_{1,2} = -\frac{1}{2g(\mu)}. \tag{32}$$

The system is a monotonic attenuation process with output:

$$y(t) = 1 - \left(1+\frac{t}{T}\right){\rm e}^{-t/T} = 1 - (1+2g(\mu)t)\,{\rm e}^{-2g(\mu)t}. \tag{33}$$

Case 3. Let $\xi > 1$, i.e., $K_0 r g(\mu) < 0.25$. The eigenroots are distributed on the negative real axis as

$$-s_{1,2} = -\frac{1}{2g(\mu)} \pm \frac{\sqrt{1-4K_0 r g(\mu)}}{2g(\mu)}. \tag{34}$$

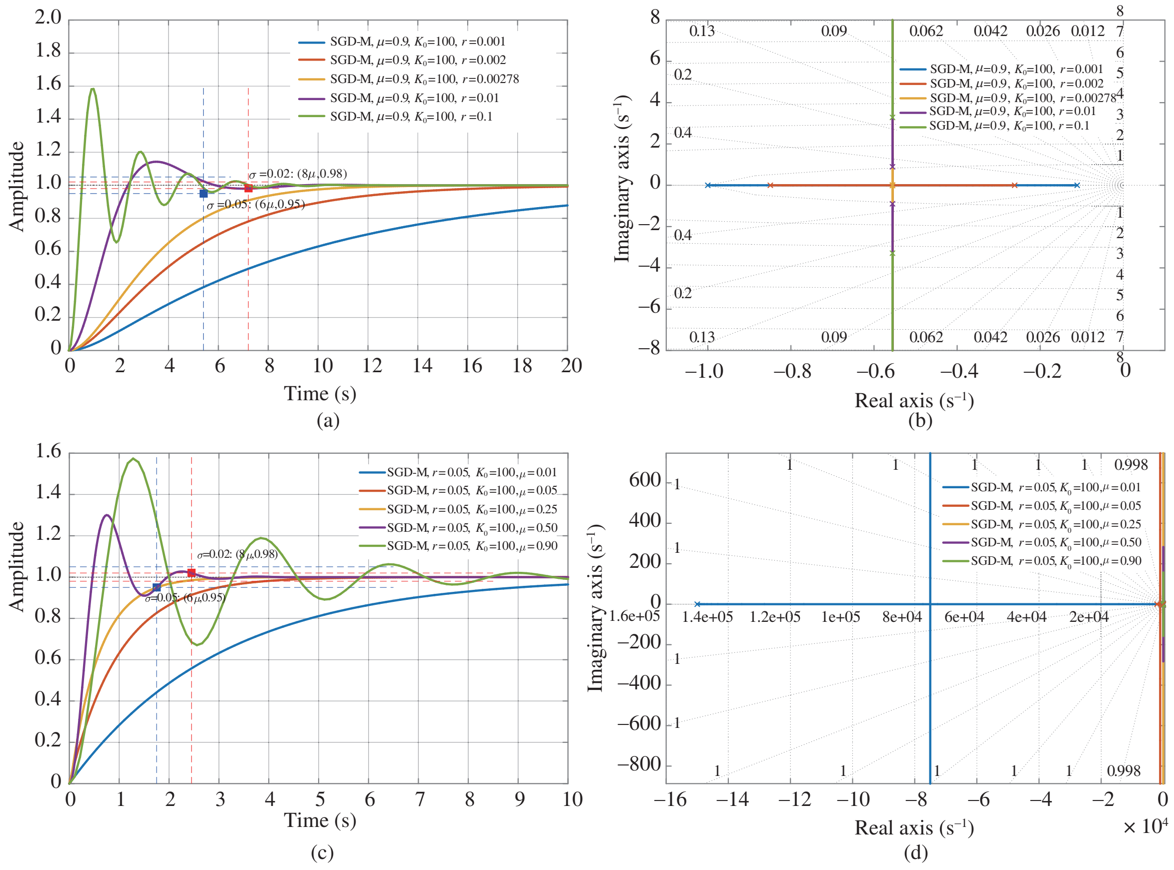

Figure 3 (Color online) (a), (c) Step response and (b), (d) root locus of SGD momentum.

It is also a monotonic attenuation system with long time-costly adjusting period, whose output is given by

$$\begin{aligned} y(t) =& 1 - \frac{-\xi+\sqrt{\xi^2-1}}{2\sqrt{\xi^2-1}} \exp\left(-(\xi+\sqrt{\xi^2-1})t/T\right) - \frac{-\xi-\sqrt{\xi^2-1}}{2\sqrt{\xi^2-1}} \exp\left(-(\xi-\sqrt{\xi^2-1})t/T\right) \\ =& 1 - \frac{\sqrt{1-4K_0rg(\mu)}-1}{2\sqrt{1-4K_0rg(\mu)}} \exp\left(-\frac{1}{2}(1+\sqrt{1-4K_0rg(\mu)})t\right) \\ &+ \frac{\sqrt{1-4K_0rg(\mu)}+1}{2\sqrt{1-4K_0rg(\mu)}} \exp\left(-\frac{1}{2}(1-\sqrt{1-4K_0rg(\mu)})t\right). \end{aligned} \tag{35}$$

We show the step response and root locus in Figure 3. Here, we use a simple form of $g(\mu) = \mu$. We draw the settling time line at $\sigma = 0.05$, where $t_s = 6\mu$ in blue color, and $\sigma = 0.02$, and $t_s = 8\mu$ in red color.

In Figure 3(a), by fixing μ and K_0, the optimization dynamics are determined by r only. The critical line is determined by $K_0rg(\mu) = 0.25$, as drawn in yellow, where $K_0 = 100$, $r = 0.00278$, $g(\mu) = 0.9$. Considering the limitation of $\sigma = 0.05$ and $\sigma = 0.02$, only with $r = 0.01$ and $r = 0.1$ is the system output stabilized after the settling time line, while the system with smaller r needs more time to produce an acceptable output. The difference between the first- and second-order optimizer is that the first-order system requires more settling time but has no overshoot, while the second-order system converges faster and is accompanied by some overshoot. Another view of $r = 0.01$ and $r = 0.1$ is that a bigger learning rate may cause the system to experience larger vibrations, which may also lead the system to be applied improperly.

The root locus in Figure 3(b) shows the stability of all cases in Figure 4(a), with different zones decided by eigenroots. The distribution of $-s_{1,2}$ at the real axis implies that the dynamics of the optimizers with $r = 0.001$ and $r = 0.002$ are cases of monotonic attenuation, while the rest of the models have overshoot to various degrees. The roots located far away on the imaginary axis have large vibrations.

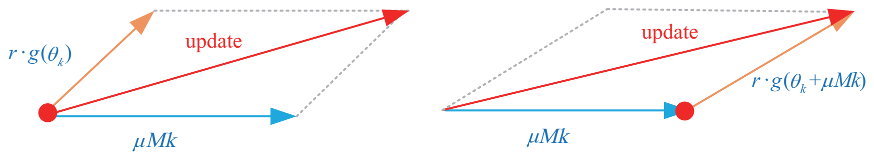

Figure 4 (Color online) Updating tracks of classical momentum (left) and Nesterov accelerated gradient (right).

In Figure 3(c), we simulate the momentum optimizer varying from 0.01 to 0.9, with fixed r and K_0. The system vibrates at different magnitudes determined by μ. The overshoot increases dramatically if the momentum is too big, accompanied by a delayed settling time (see the green curve where $\mu = 0.9$). However, a small μ may change the model without overshoot, with a reduced convergence speed. All eigenroot tracks in Figure 3(d) are in the left area of the imaginary axis, indicating that all systems are stable.

3.3 Nesterov accelerated gradient

The NAG [21] is a way to use the momentum term before the current step, with the form:

$$\begin{cases} v_{k+1} = \mu v_k - r\dfrac{\partial L_k}{\partial(\theta_k + \mu v_k)}, \\ \theta_{k+1} = \theta_k + v_{k+1}. \end{cases} \tag{36}$$

Let $\hat{\theta}_k = \theta_k + \mu v_k$, and rewrite v_{k+1} in the same way as SGD momentum,

$$v_{k+1} = -r\sum_{i=1}^{k} \mu^{k-i}\frac{\partial L_i}{\partial \hat{\theta}_i}. \tag{37}$$

Thus, Eq. (33) is transformed to

$$\begin{aligned} \theta_{k+1} - \theta_1 = (v_2 + v_3 + \cdots + v_{k+1}) &= -r\begin{pmatrix} \mu^0\dfrac{\partial L_1}{\partial\hat{\theta}_1} \\ \mu^1\dfrac{\partial L_1}{\partial\hat{\theta}_1} + \mu^2\dfrac{\partial L_2}{\partial\hat{\theta}_2} \\ \vdots \\ \mu^{k-1}\dfrac{\partial L_1}{\partial\hat{\theta}_1} + \cdots + \dfrac{\partial L_k}{\partial\hat{\theta}_k} \end{pmatrix} \\ &= -r \cdot \left(\frac{\partial L_1}{\partial\hat{\theta}_1}, \frac{\partial L_2}{\partial\hat{\theta}_2}, \ldots, \frac{\partial L_k}{\partial\hat{\theta}_k}\right)\begin{pmatrix} \mu^0 + \mu^1 + \cdots + \mu^{k-2} + \mu^{k-1} \\ \mu^0 + \mu^1 + \cdots + \mu^{k-2} \\ \vdots \\ \mu^0 \end{pmatrix}. \end{aligned} \tag{38}$$

One if the major difference between momentum and NAG is the updating logic from the current position (see Figure 4). Momentum uses both the gradient information and momentum of the current step, while in NAG, the gradient of momentum is introduced as an extra control factor to rectify the moving tracks from a previous step based on the second derivative of the gradient, or the first derivative of the momentum.

Theorem 3. The Nesterov accelerated gradient optimizer is a second-order control system with the transfer function $G_{\theta_{\text{nag}}}(s) = \frac{K_0 r g(\mu)s + K_0\alpha r}{g(\mu)s^2 + s}$, formed by learning rate r, momentum factor μ, proportional coefficient K_0, and advanced control factor α.

Remark 3. According to the series connection from SGD to NAG, the anticipatory control factor is a classical element described by $G_n = \alpha\frac{Ts+1}{\alpha Ts+1}, \alpha < 1$, thus, the transfer function of NAG is achieve as

$$G_{\theta_{\text{nag}}}(s) = G_{\theta_{\text{sgd}}}(s) \cdot G_n = K_0 \cdot \frac{r}{s} \cdot \frac{\alpha Ts + \alpha}{\alpha Ts + 1}. \tag{39}$$

To reveal the difference from momentum, rewrite $G_{\theta_{\rm nag}}(s)$ as

$$G_{\theta_{\rm nag}}(s) = K_0 \cdot \frac{r}{s} \cdot \frac{1}{\alpha Ts+1} \cdot (\alpha Ts + \alpha). \tag{40}$$

Let

$$G_{\theta_{\rm nag}}(s) = K_1 \cdot \frac{r}{s} \cdot \frac{1}{g(\mu)s+1} \cdot \left(\frac{g(\mu)}{\alpha}s+1\right), \tag{41}$$

where $g(\mu) = \alpha T$, $K_1 = K_0\alpha$, $\alpha < 1$, then

$$G_{\theta_{\rm nag}}(s) = \frac{K_0 r g(\mu) s + K_0 \alpha r}{g(\mu)s^2 + s}. \tag{42}$$

The closed-loop transfer function is

$$\varphi(s) = \frac{\frac{g(\mu)}{\alpha}s+1}{\frac{g(\mu)}{K_1 r}s^2 + \frac{(K_1 r \frac{g(\mu)}{\alpha}+1)}{K_1 r}s + 1}. \tag{43}$$

We can simplify the function $g(\mu) = \mu$ and rewrite it in the common form,

$$\phi(s) = \phi_1(s) + \phi_2(s),$$
$$\begin{cases} \phi_1(s) = \dfrac{1}{\frac{\alpha}{K_0 r}s^2 + \frac{K_0 r^2\alpha+\alpha}{K_0 r\mu}s + \frac{\alpha}{\mu}}, \\ \phi_2(s) = \dfrac{s}{\frac{\alpha}{K_1 r}s^2 + \frac{(K_1 r\mu+\alpha)}{K_1 r\mu}s + \frac{\alpha}{\mu}}. \end{cases} \tag{44}$$

The step response of the system is

$$y(s) = \varphi_1(s)s + \varphi_2(s)s = y_1(s) + y_2(s). \tag{45}$$

The output is

$$y(t) = y_1(t) + y_2(t),$$
$$\begin{cases} y_1(t) = 1 - \dfrac{1}{\sqrt{1-\xi^2}}{\rm e}^{-\xi\omega_n t}\sin\left(\sqrt{1-\xi^2}\omega_n t + \arctan\dfrac{\sqrt{1-\xi^2}}{\xi}\right), \\ y_2(t) = \dfrac{\tau\omega_n}{\sqrt{1-\xi^2}}{\rm e}^{-\xi\omega_n t}\sin\left(\sqrt{1-\xi^2}\omega_n t\right). \end{cases} \tag{46}$$

According to Laplace transform theory, we have

$$y_2(t) = r\ell^{-1}[sy_1(s)] = r\frac{{\rm d}y_1(t)}{{\rm d}t} + r\ell^{-1}[y_1(0)]. \tag{47}$$

Since $y_1(0) = 0$, we have

$$y_2(t) = r\frac{{\rm d}y_1(t)}{{\rm d}t}. \tag{48}$$

We demonstrate the step response of this system in Figure 5. Note that we use l to represent the distance between the zero and pole.

NAG generally improves the SGD momentum in overshoot and time. In Figure 5(a), the vibration magnitude is reduced to a relatively small zone compared to Figure 3(a), under the condition that all parameters remain the same. For example, given $\mu = 0.9$, $K_0 = 100$, and $r = 0.1$, the peak value of the amplitude reaches 1.6 in the momentum method; note that this overshoot is about 60% of the step signal. In NAG, the peak is reduced to 1.35. Furthermore, settling into momentum takes about 10 s, but takes less than 3 s in NAG.

In Figure 5(b), all eigenroots are at the left part of the imaginary axis. Compared to the eigenroots in Figure 3(b), the eigenroots move to the real axis after hundreds of seconds from 100 to 2000, and then

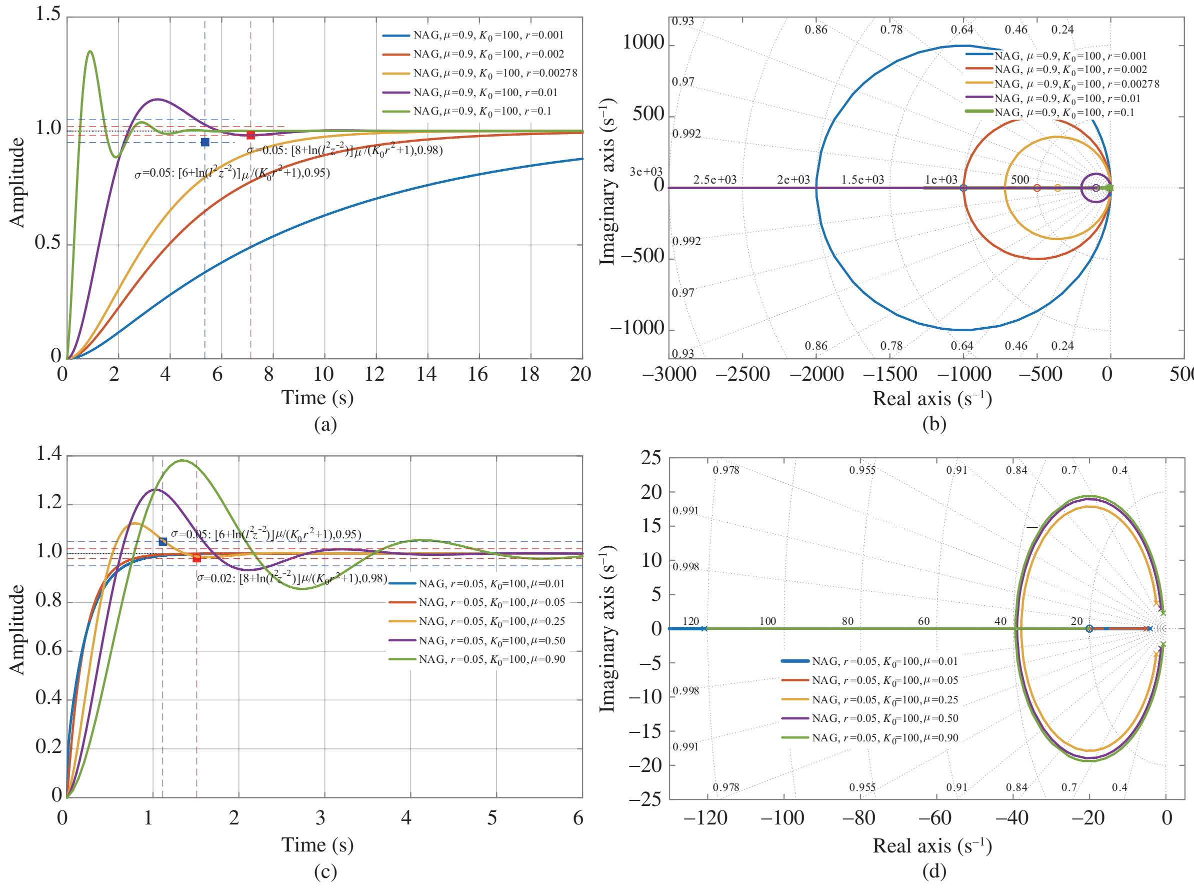

Figure 5 (Color online) (a), (c) Step response and (b), (d) root locus of NAG.

Table 1 Step-signal response of optimization models SGD, SGD momentum, and NAG

Optimization model	Parameters	Transfer function	Order	Overshoot	Settling time
SGD	K_0, r	$G_{\theta_{\rm sgd}}(s) = K_0 \cdot \frac{r}{s}$	1	No	Long
SGD momentum	K_0, r, μ	$G_{\theta_{\rm sgdm}}(s) = \frac{K_0 r}{\mu s^2 + s}$	2	Depend on $\xi = \frac{1}{\sqrt{4K_0 r\mu - 1}}$	Middle-long
NAG	K_0, r, μ, α	$G_{\theta_{\rm nag}}(s) = \frac{K_0 r\mu s + K_0 r\alpha}{\mu s^2 + s}$	2	Depend on $\xi = \frac{\mu K_0 r\alpha + 1}{2\sqrt{\mu K_0 r\alpha}}$	Middle-short

enter the monotone period. In Figure 4(b), most eigenroots have the imaginary parts, which means the system needs more settling time.

In Figure 5(c), the learning rate r and proportional coefficient K_0 are fixed, and the system converges faster than in Figure 3(c). The main reason is that NAG benefits the moving directions of the system in advance. The increasing overshoots reflect the affections caused by historical NAG. A bigger NAG reaches the peak in less than 1.5 s (with $\mu = 0.9$, $K_0 = 100$, and $r = 0.05$), which implies that the system may not smooth if the approximation target has too many saddle points. Figure 5(d) describes the vibration period, which decreases more quickly than in Figure 3(d). All eigenroots enter the negative real axis after 40 s, making the system track more quickly into a diminishing period.

3.4 Signal response of the optimizers

We summarize the system dynamics of SGD, SGD momentum, and NAG in Table 1. Each model is determined by some predefined parameters. The open transfer functions provide an approximation of each model in a directive way. SGD is a first-order control system without overshoot, but the settling time is longer. SGD momentum improves SGD by adding an inertial adjuster, thus changing the system to a classical second-order system, and the performance depends on the fine-tuning of the parameters. NAG uses a differentiation element to enhance the momentum method, with a zero pole added to make an emendation in advance, and the settling time and overshoot both diminish quickly. The improved

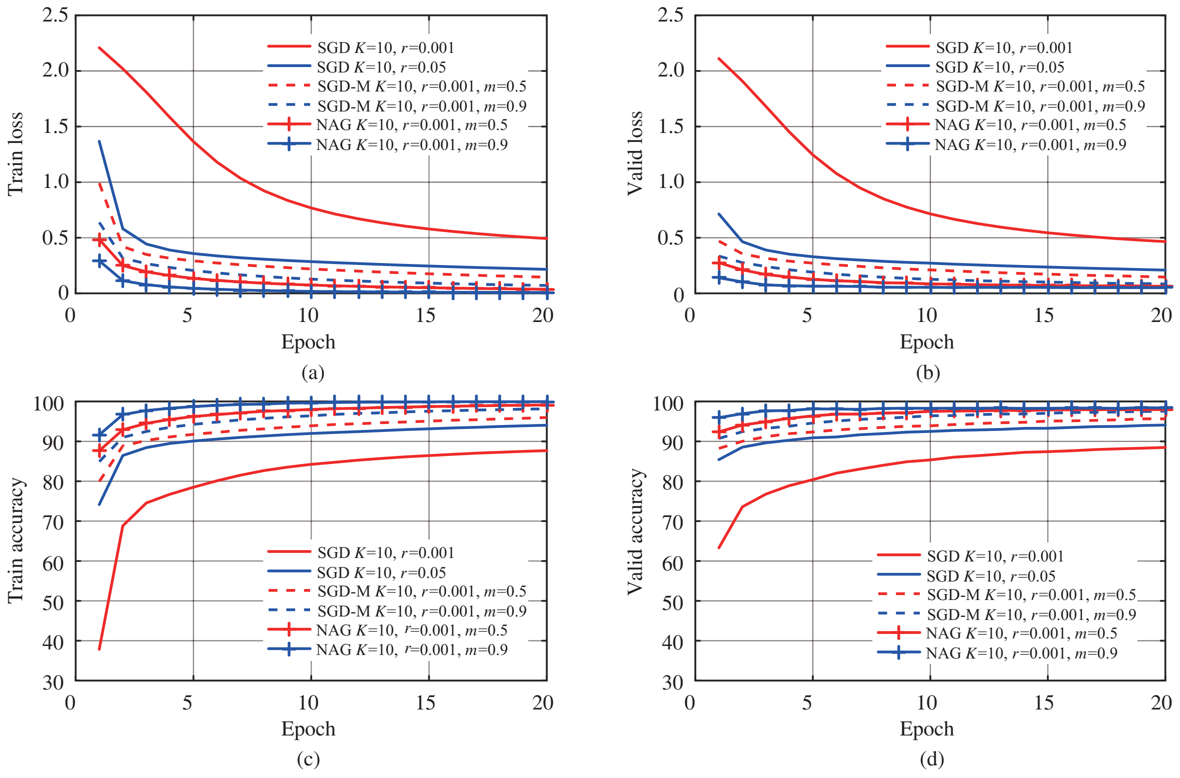

Figure 6 (Color online) Identification results on the MNIST dataset of different optimizers. (a) The training loss, (b) valid loss, (c) training accuracy, and (d) valid accuracy of SGD, momentum and NAG, respectively.

methods benefit the original SGD, but require sophisticated parameter design techniques.

Remark 4. The stability of a deep learning model is affected by network structure, optimization strategy and application environment. Generally, a DNN model is formed by many layered neurons/kernels, the risk of gradient decay and explosion is increasing with intensified layers. In extremes of abnormal inputs, an improper learning rate will cause system vibration or even unstable.

4 Experiments

We use the above optimization models to verify how the parameters in each model affect the system performance on two popular datasets. The MNIST dataset of handwritten digits has a training set of 60000 examples, and a testing set of 10000 examples [22]2). The CIFAR-10 dataset contains 60000 32×32 RGB images, which are divided into 10 classes, respectively3).

We compare the recognition results on the MNIST dataset in Figure 6, and report the numerical results of some key levels in Table 2.

The SGD optimization with a small learning rate ($r = 0.001$) in the training stage greatly delays the system response because the settling time t_s is affected by $\frac{1}{K_0 r}$. Note that we use the batch gradient descent method to accelerate the training, thus we can learn the relationship between t_s and $K_0 r$, rather than obtaining a precise definition. The training is improved with the r increased of the rest settings. In SGD momentum case, the validation loss reduces faster than SGD during training, and the best validation accuracy achieved finally is 98.22%.

The importance of the momentum factor is implied by comparing SGD momentum and NAG, by fixing r and K_0, the training loss is downsized when the momentum factor is enhanced. The model has a training accuracy of 94.99% after four epochs, which reflects the adjusting function of the first-order inertial element. The validation result is also improved from Figure 6(b) and (d), as it is more accurate with less time cost. The dynamics of NAG with all parameter settings remaining in momentum. The

2) https://www.graviti.cn/open-datasets/MNIST.

3) http://www.cs.toronto.edu/~kriz/cifar.html.

Table 2 Key performance index of SGD, SGD momentum and NAG on the MNIST dataset at given accuracy levels

Train accuracy	Index	SGD	SGD momentum	NAG
		$K_0 = 10$, $r = 0.05$	$K_0 = 10$, $r = 0.05$, $\mu = 0.9$	$K_0 = 10$, $r = 0.05$, $\mu = 0.9$
$\geqslant$ 95%	Epoch	8	5	3
	Time (s)	127.321	51.554	21.339
	Train loss	0.164	0.162	0.115
	Valid loss	0.157	0.145	0.103
	Train accuracy (%)	95.39	95.43	96.69
	Valid accuracy (%)	95.61	95.75	96.94
$\geqslant$ 98%	Epoch	19	11	5
	Time (s)	278.489	111.552	36.887
	Train loss	0.072	0.075	0.058
	Valid loss	0.085	0.087	0.069
	Train accuracy (%)	98.09	98.01	98.29
	Valid accuracy (%)	97.46	97.41	97.74

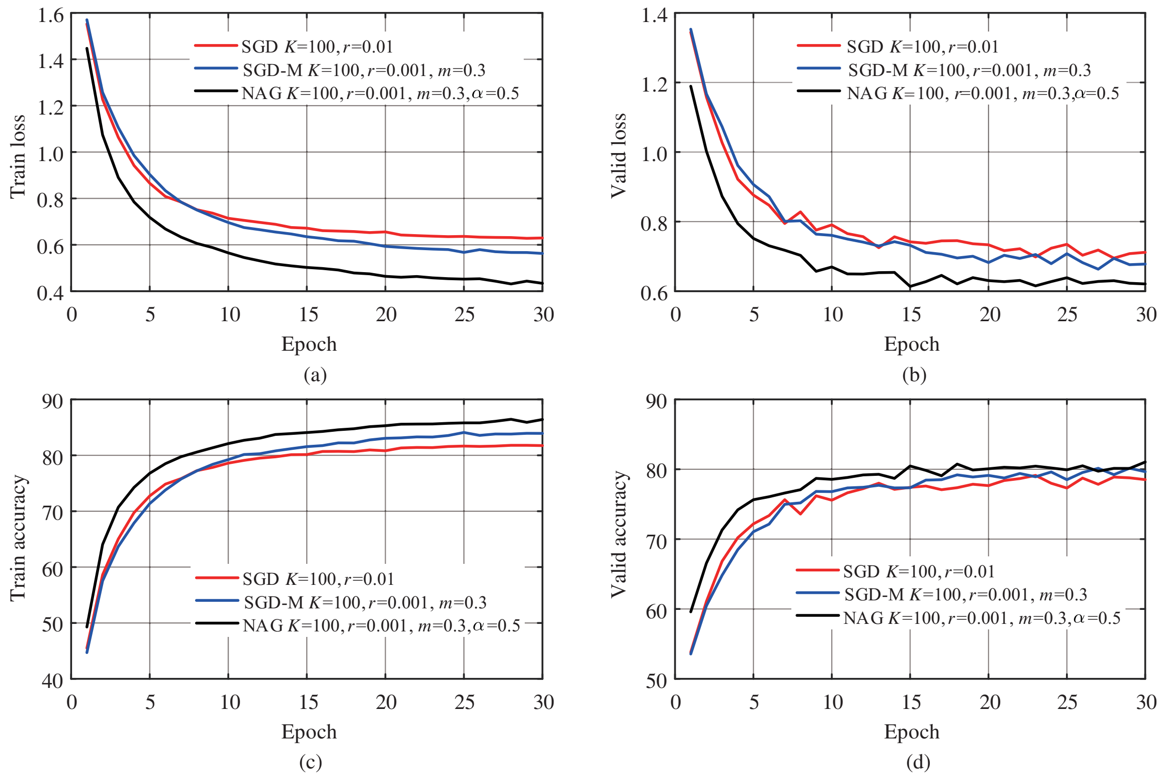

Figure 7 (Color online) Identification results on the CIFAR-10 dataset of different optimizers. (a) The training loss, (b) valid loss, (c) training accuracy, and (d) valid accuracy of SGD, SGD momentum and NAG, respectively.

training loss is reduced by 29.8% than SGD, and 29.01% in Momentum. Taking the case of $\mu = 0.9$ as an example, the training accuracy approaches 100% (99.96%, actually) after 20 epochs, with the first landmark, 96.69% at the second epoch. For the MNIST dataset, the learning speed is increased with bigger learning rate and momentum.

The performance on the CIFAR dataset is shown in Figure 7, accompanied by two comparative training stages listed in Table 3. To reduce the negative effect from manual interventions, we introduce the Hyperband [23] as hyper-parameter optimization tool for our idea, based on the grid searching suggestions, we select $r = 0.01$ as the representative case in SGD optimizers, $r = 0.001$, $m = 0.3$ in SGD-M and $r = 0.001$, $m = 0.3$ in NAG.

Note that the CIFAR dataset has more color channels and feathers, the training requires more time for computation. The SGD method with bigger learning rate ($r = 0.01$) indicates better accuracy than ($r = 0.001$). The momentum factor ($m = 0.3$) improves the learning speed a lot under the same hyper-

Table 3 Key performance index of SGD, SGD momentum and NAG on the CIFAR dataset at given accuracy levels

Train accuracy	Index	SGD	SGD momentum	NAG
		$K_0 = 100$, $r = 0.001$	$K_0 = 100$, $r = 0.001$, $\mu = 0.5$	$K_0 = 100$, $r = 0.001$, $\mu = 0.5$, $\alpha = 0.5$
$\geqslant 70\%$	Epoch	38	9	8
	Time (s)	3713.554	1949.627	2274.326
	Train loss	1.199	1.079	1.286
	Valid loss	1.258	1.574	1.547
	Train accuracy (%)	70.28	70.71	70.73
	Valid accuracy (%)	69.01	65.01	54.76
$\geqslant 80\%$	Epoch	78	35	16
	Time (s)	6578.489	7784.554	4479.583
	Train loss	0.931	1.219	1.194
	Valid loss	0.924	1.034	1.364
	Train accuracy (%)	80.04	80.03	80.93
	Valid accuracy (%)	80.74	79.55	60.38

parameter settings in SGD ($K = 100$, $r = 0.01$), and has been further improved in NAG under the same conditions.

An interesting phenomenon is, by fixing the hyper-parameters ($K = 100$, $r = 0.006$, $m = 0.5$), the momentum way experiences some fluctuations in training and validation. This is caused by the production $Kr\mu = 0.30 > 0.25$, acts as a second-order model. Moreover, the unstable vibrations is downsized in NAG because of system coordinating by an extra zero introduced.

Table 2 compares the dynamics of three optimizers on the MNIST dataset. Often, the training accuracy at the 95% and 98% levels is regarded as the key points. The settling time of SGD is much longer than those of SGD momentum and NAG. SGD obtains 95.39% training accuracy in eight epochs, while SGD momentum and NAG reach this level in five and three epochs, respectively. In summary, NAG is more adaptive than SGD momentum and SGD. Similar trends are easily to be read in Table 3, consider the training requires more effort, we select the accuracy level by 70% and 80%, though some fluctuations exist, the NAG and Momentum method performs better than SGD under the same conditions.

5 Conclusion

DNNs obtain promising results from the given input to the desired target, and benefit from the fine-tuned parameters. In this paper, the transfer function is applied to approximate three SGD-based optimizers. Basically, the SGD optimizer is a traditional first-order system, and it needs more time than a momentum optimizer or NAG optimizer to reach stability. The momentum optimizer changes SGD by adding one inertial element, transforming the system to a second-order system. Furthermore, the NAG optimizer puts one more zero on the second-order system of Momentum way. Both revisions move the eigenroots of the optimizers moving toward the left panel of root locus coordinates. As a result, while applying these optimizers into real problems, trying to examine it as a control model provides us with a systemic scenery. How to simulate and simplify those optimizers is our next move.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant Nos. 61933013, U1736211), Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDA22030301), Natural Science Foundation of Guangdong Province (Grant No. 2019A1515011076), and Key Project of Natural Science Foundation of Hubei Province (Grant No. 2018CFA024).

References

1 Ruder S. An overview of gradient descent optimization algorithms. 2016. ArXiv:1609.04747
2 An W P, Wang H Q, Sun Q Y, et al. A PID controller approach for stochastic optimization of deep networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2018. 8522–8531
3 Kim D, Kim J, Kwon J, et al. Depth-controllable very deep super-resolution network. In: Proceedings of International Joint Conference on Neural Networks, 2019. 1–8
4 Hinton G, Srivastava N, Swersky K. Overview of mini-batch gradient descent. 2012. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
5 Qian N. On the momentum term in gradient descent learning algorithms. Neural Netw, 1999, 12: 145–151
6 Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. J Mach Learn Res, 2011, 12: 2121–2159
7 Zeiler M D. Adadelta: an adaptive learning rate method. 2012. ArXiv:1212.5701

8 Dauphin Y N, de Vries H, Bengio Y. Equilibrated adaptive learning rates for nonconvex optimization. In: Proceedings of Conference and Workshop on Neural Information Processing Systems, 2015
9 Kingma D, Ba J. Adam: a method for stochastic optimization. In: Proceedings of International Conference on Learning Representations, 2015. 1–15
10 Reddi S J, Kale S, Kumar S. On the convergence of ADAM and beyond. In: Proceedings of International Conference on Learning Representations, 2018. 1–23
11 Luo L C, Xiong Y H, Liu Y, et al. Adaptive gradient methods with dynamic bound of learning rate. In: Proceedings of International Conference on Learning Representations, 2019. 1–19
12 Saxe A M, McClelland J L, Ganguli S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. 2013. ArXiv:1312.6120
13 Lee T H, Trinh H M, Park J H. Stability analysis of neural networks with time-varying delay by constructing novel Lyapunov functionals. IEEE Trans Neural Netw Learn Syst, 2018, 29: 4238–4247
14 Faydasicok O, Arik S. A novel criterion for global asymptotic stability of neutral type neural networks with discrete time delays. In: Proceedings of International Conference on Neural Information Processing, 2018. 353–360
15 Vidal R, Bruna J, Giryes R, et al. Mathematics of deep learning. 2017. ArXiv:1712.04741
16 Chaudhari P, Oberman A, Osher S, et al. Deep relaxation: partial differential equations for optimizing deep neural networks. Res Math Sci, 2018, 5: 30
17 Wang H Q, Luo Y, An W P, et al. PID controller-based stochastic optimization acceleration for deep neural networks. IEEE Trans Neural Netw Learn Syst, 2020, 31: 5079–5091
18 Cousseau F, Ozeki T, Amari S. Dynamics of learning in multilayer perceptrons near singularities. IEEE Trans Neural Netw, 2008, 19: 1313–1328
19 Amari S, Park H, Ozeki T. Singularities affect dynamics of learning in neuromanifolds. Neural Comput, 2006, 18: 1007–1065
20 Bietti A, Mairal J. Group invariance, stability to deformations, and complexity of deep convolutional representations. J Mach Learn Res, 2019, 20: 876–924
21 Sutskever I, Martens J, Dahl G, et al. On the importance of initialization and momentum in deep learning. In: Proceedings of International Conference on Machine Learning, 2013. 1139–1147
22 Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. Proc IEEE, 1998, 86: 2278–2324
23 Li L S, Jamieson K, DeSalvo G, et al. Hyperband: a novel bandit-based approach to hyperparameter optimization. J Mach Learn Res, 2018, 18: 1–52