SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

November 2021, Vol. 64 212103:1–212103:15
https://doi.org/10.1007/s11432-020-3022-3

Learning dynamics of kernel-based deep neural networks in manifolds

Wei WU[1,3], Xiaoyuan JING[1,2*], Wencai DU[4] & Guoliang CHEN[5]

[1]*School of Computer Science, Wuhan University, Wuhan* 430072*, China;*
[2]*School of Computer, Guangdong University of Petrochemical Technology, Maoming* 525000*, China;*
[3]*Institute of Deep-sea Science and Engineering, Chinese Academy of Sciences, Sanya* 572000*, China;*
[4]*Institute of Data Science, City University of Macau, Macau* 999078*, China;*
[5]*College of Computer Science and Software Engineering, Shenzhen University, Shenzhen* 518060*, China*

Received 31 January 2020/Revised 26 April 2020/Accepted 4 June 2020/Published online 12 October 2021

Abstract Convolutional neural networks (CNNs) obtain promising results via layered kernel convolution and pooling operations, yet the learning dynamics of the kernel remain obscure. We propose a continuous form to describe kernel-based convolutions through integration in neural manifolds. The status of spatial expression is proposed to analyze the stability of kernel-based CNNs. We divide CNN dynamics into the three stages of unstable vibration, collaborative adjusting, and stabilized fluctuation. According to the system control matrix of the kernel, the kernel-based CNN training proceeds via the unstable and stable status and is verified by numerical experiments.

Keywords learning dynamics, kernel-based convolution, manifolds, control model, network stability

Citation Wu W, Jing X Y, Du W C, et al. Learning dynamics of kernel-based deep neural networks in manifolds. Sci China Inf Sci, 2021, 64(11): 212103, https://doi.org/10.1007/s11432-020-3022-3

1 Introduction

The convolutional neural network (CNN) is well applied in image recognition, voice identification, and many industrial problems owing to its strong feature representation and high-dimensional spatial computing capability. However, the dynamics of neural networks have not been extensively studied, and could result in disaster under certain conditions. Mathematical iteration can be regarded as a control-model-driven application, which may or may not be dynamically stable. The serious defect here is that the status of an untrained system is defined by a predefined model structure and learning strategy, and affected by artificial input-output transformation.

CNN-related studies are mainly result-oriented rather than theory-driven. With the advantages of multiple parameter optimization and abstract feature representation, a CNN model combines two basic layered structures, that is, convolution and pooling, to efficiently extract features from a large number of high-dimensional data samples and reduce computational complexity while the dynamic mechanism remains unrevealed.

The application of a CNN model to an engineering issue has three steps: analyzing historical data and selecting a proper model, training parameters for the model, and applying input data to verify network output against real values. If a model is trained with limited iterations, its stability cannot be guaranteed, and it may not produce reasonable output free of unexpected error (or being out of bounds), which may cause catastrophic damage in the application.

To avert potential threats to CNNs, we explain the mathematical principles for convolution in manifolds from the control perspective.

* Corresponding author (email: jingxy_2000@126.com)

2 Related work

Convolution operations have been studied to improve system performance, visualize kernel values, and analyze parameter distributions. The stability analysis of neural networks in nonlinear function approximation has been studied to infer system output characteristics.

The kernel size varies in many deep learning models, with the goal to reduce the error between the system output and real values. Google Inception uses an adaptive method to introduce kernels of various dimensions [1–3]. Visualizing work [4] indicated that regularization preprocessing improved algorithm efficiency, and a convolution kernel obtained abstract features hierarchically in different layers, consistent with human recognition. A better understanding of how kernel design affects convergence speed and output accuracy, with kernel element optimization tricks based on error backpropagation and model structure adjustment, is restricted by limited data dimension. An effort was made [5] to understand the flattening of such manifold-shaped data, and determine what amount of flattening a deep neural network can achieve, by a few quantities for measuring manifold entanglement. The manifold perspective also reveals the usefulness of a fluent structure in deep models.

The use of geometric manifolds to learn model parameters from their original versions to another space has been studied in many neural networking models, including radial basis function neural networks (RBFNNs) and multilayer perceptrons (MLPs). Studies focused on the singularity areas caused by algorithm identification parameters, and revealed that weights of a single neuron and linear summation between two neuron units affect system stability [6–8]. To transform coordinates from a standard parameter space to a high-dimensional space enables the Taylor expansion to be meaningful in singularities, and provides local influence derived from model parameters. The connection between non-convex optimization for training deep neural networks and nonlinear partial differential equations is established to enhance the robustness of stochastic gradient descent [9].

The spatial distribution of input data induces system instability in certain ways. In all CNN layers, the learning from input data to a supervised target requires adjustment through all kernel settings, including size, learning rate, weight, and bias. The route varies for linearly and nonlinearly distributed input data [10]. Considering the input data dimension, a genetic algorithm model may produce good simulation results with one- and two-input pole balancing problems, but may not be stable when input shifts to an unexpected status out of the regular track or angle of a system [11]. Moreover, the accuracy of system estimation depends on the interactions between observational data intervals and density, and it is necessary to establish conditions to ensure asymptotic stability [12]. Arbitrary Markovian control policies for Gaussian processing in both average and full distributions have been analyzed [13]. A closed-loop control system has been proved to converge to a desired range in finite time.

The dynamics of the convergence of the model parameters of a neural network and how the system output is stabilized are theoretically a control model. An analytical description of an integral form bridges the gap between the theory and practice of deep learning of all layers [14, 15]. Classical methods are implemented by transfer functions in the time and frequency domains, where it remains popular to introduce Lyapunov stability theory in cybernetics to study the status changing pattern. For some neural network models with time-varying delay stages, specified Lyapunov functionals are derived in the form of linear matrix inequalities [16]. For neutral-type neural networks including constant delay parameters, a properly modified Lyapunov functional employing Lipschitz activation functions was derived [17]. However, the transfer function of convolution is not easily expressed owing to the deep layered structure and kernel operation in matrix form, which brings the zero points and pole points of the system into high-dimensional orders, preventing the status space expression from inferring from input data, system parameters, and output from an analytical transfer function matrix.

Model parameters apparently affect system output stability. Transforming coordinates from standard parameter space to a new space, to understand the influence from input data and model variables, the series work focused on singularities area [7,18], calculating the partial derivative for each new variable, and showing the dynamic variation near singularities affected by weights between hidden layers and output layers.

The dynamics of model parameter convergence and system output stability are studied in the time domain by introducing the Lyapunov stability method. For neural networks with a time-varying delay, specified Lyapunov functionals are proposed in the form of linear matrix inequalities in [16]. For neutral-type neural networks including constant delay parameters, a properly modified Lyapunov functional employing Lipschitz activation functions was derived [17].

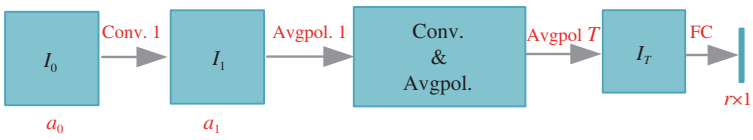

Figure 1 (Color online) A CNN example: I_0 is the input data of dimension $a_0 \times a_0$; Conv.i and Avgpol.i are basic operations of convolution and average pooling, respectively; and the final layer is given by fully connecting FC.

System stability analysis is complex owing to input variables, model structures, parameters, and training strategies. No dynamics for deep learning models based on RBFNNs in the frequency domain exists using control theory, especially regarding the dynamical progress of system accuracy and convergence speed.

Kernel-based convolutional computing in CNNs is generally complex owing to input variables, kernel structures, and learning strategies. The backward method of the element multiplication operation has few spacial properties. Our work contributes to the theory of convolution from two aspects.

(1) We propose an analytical expression of kernel-based convolution operations in CNNs, as a high-dimensional integration throughout the manifolds with embedded input data and kernel, approximating the discrete piecewise convolution operation in a continuous form in the manifold domain.

(2) We transform the kernel in the CNN as a status control variable, under the excitation of input data used for learning processing. We divide learning into three stages based on training iterations and discuss the convergence of the CNN.

The remainder of this study is organized as follows: Section 3 discusses preliminaries of the convolution operation and presents an approximation method for convolution from discrete space to continuous manifolds. We theoretically discuss the learning process in Section 4. The dynamics of CNN kernels are analyzed in Section 5, with examples based on experiments. We draw conclusion in Section 6.

3 Convolution operation in manifolds

3.1 Discrete kernel-based convolution

An example of CNN operation is shown in Figure 1. Through a series of convolution and pooling operations, we map its original form to a fully connected version that satisfies the output requirements. By introducing M kernels $\kappa_{(1,\cdot)}$ in the first layer, the output corresponding to the m-th kernel is produced via the convolution operation:

$$I_{(1,a_1)} = I_0 * \kappa_{(1,m)}, \tag{1}$$

where $\kappa_{(1,\cdot)} = \{\kappa_{(1,1)}, \kappa_{(1,2)}, \ldots, \kappa_{(1,M)}\}$, t_1 is the predefined kernel dimension, $\kappa_{(1,a_1)} \in R^{(t_1,t_1)}$, and $I_0 \in R^{(a_0,a_0)}$. Then we obtain the convolution result of the first layer by

$$I_{(1,\cdot)} = \Upsilon_{m=1}^{M} I_{(1,\cdot)} = \left[\!\left[I_{(1,1)}, I_{(1,2)}, \ldots, I_{(1,M)}\right]\!\right], \tag{2}$$

where Υ represents the combination operation, $[\![\cdots]\!]$ is the tensor from all matrices, and $I_{(1,\cdot)} \in R^{(a_0-t_1,a_0-t_1,M)}$.

To reduce the data dimension of the previous convolution result, for a small block of $I_{(1,\cdot)}$, we use its maximum element or average value to cut down the feature map. For example, an average pooling operator g_{E_1} for all small blocks in $I_{(1,\cdot)}$ is

$$I_{(2,\cdot)} = I_{(1,\cdot)} * g_{E_1}, \tag{3}$$

where $g_{E_1} = \frac{1}{4}E_1$, and $E_1 \in R^{(2,2,M)}$ is a tensor with all elements defined as 1.

As a result, the output of the last layer before fully connecting is

$$I_{(T,\cdot)} = \Upsilon_{m'=1}^{M'} I_{((T-1),\cdot)} * g_{E_T}. \tag{4}$$

The final output is obtained through several full connecting layers,

$$Y = B\phi(I_{(T,\cdot)}), \tag{5}$$

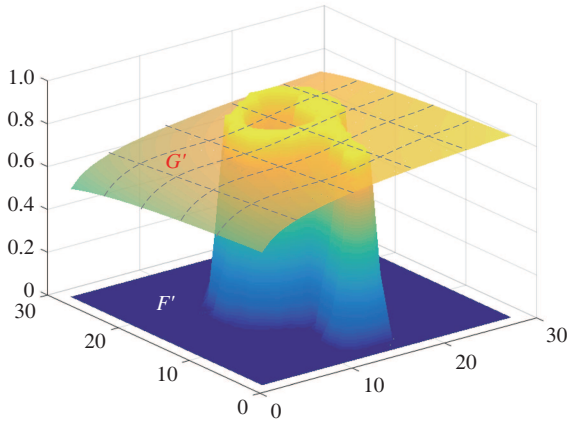

Figure 2 (Color online) A convolution processing example in $\Omega'(u,v,w)$: the original image F represents a handwritten number 9 in RGB mode, a regularized surface π'_I of F' is obtained in $\Omega'(u,v,w)$ with elements restricted to the range $[0,1]$, and G' is a kernel surface π'_κ.

where ϕ is the full connection operator, and B is the control matrix determined by the problem requirement of the workspace, $Y \in R^{(r,1)}$.

The convolution output among kernels from an input instance is a feature map represented by a tensor (R^3), a latent goal of element multiply way is to reduce computing complexity by sharing weights and pooling squares, and the layered structure is applied to reduce the computing complexity caused by high-dimensional kernel functions.

3.2 Spacial integration of kernel convolution

Assume an input tensor x_i in $(M+1)$-dimensional space Ω_{M+1} is represented as a surface,

$$\pi_I = f(x_i;\theta), \quad \theta = \{\theta_1, \theta_2, \ldots, \theta_{M+1}\}, \tag{6}$$

where $\{e_1, e_2, \ldots, e_{M+1}\}$ is one of the coordinate systems of Ω; note that the coordinate system is not unique in Ω. Similarly, let another space κ in Ω be represented by

$$\pi_\kappa = g(x_i;\theta), \quad \theta = \{\theta_1, \theta_2, \ldots, \theta_{M+1}\}. \tag{7}$$

The convolution between input data I and operator κ in Ω can be calculated by

$$C_\Omega(I, \kappa_j) = I * \kappa_j = \sum_{i=1}^{n} I_i \times \kappa_j \times h_{(I,\kappa)}, \tag{8}$$

where $h_{(I,\kappa)}$ is the distance metric between I and κ_j; in common convolution processing, we let $h_{(I,\kappa)} = 1$. n is the multiply operation times determined by the input data scale, kernel size, and convolution strategy.

Many improved CNNs apply regularization before elements multiplying from two matrix, to adjust the original data into a more impact space, with all elements in a more closed sphere,

$$\Omega' : \theta_1^2 + \theta_2^2 + \cdots + \theta_{M+1}^2 \leqslant 1. \tag{9}$$

We indicate the operation in Figure 2. As a result, I and κ are expanded into another bounded space Ω'. The transformed surface is given by

$$\begin{cases} \pi'_I : f_{I'} = F'(u,v,w), \\ \pi'_\kappa : g_{\kappa'} = G'(u,v,w). \end{cases} \tag{10}$$

We obtain another version of κ_Ω as

$$C_{\Omega'}(I', \kappa'_j) = I' * \kappa'_j = \sum_{i=1}^{n} I'_i \times \kappa'_j \times h'_{(I',\kappa')}. \tag{11}$$

Remark 1. The convolution of elements multiply in discrete R^2 holds. When we calculate a convolution between a kernel and a given data matrix in R^2, the surface is not smoothed, each point is a piece with respective increments in the x and y directions predefined as $\mathrm{d}x = 1 \cdot e_1$ and $\mathrm{d}y = 1 \cdot e_2$, and the surface is seen as a divisional function at each piece as $I_{\Omega_{xy}} = \{I(x, y), (x, y) \in R^{n \times n}\}$. In this case, the connection between two patches is not smoothed, that is, $I(x_1, y_1) \neq I(x_2, y_2)$, when $(x_1 - x_2)^2 + (y_1 - y_2)^2 \neq 0$. However, since the convolution operation is carried out at the edge, $\Delta x \Delta y$ is $\varepsilon_x \varepsilon_y$, while the kernel element value is limited between g_j and g_i. If let it be $g_{\Delta ij}$, then the convolution at the patch edge is $C_\varepsilon = g_{\Delta_{ij}} \cdot \varepsilon_x \varepsilon_y$. As a result, because $\varepsilon_x \varepsilon_y \to 0$, $\lim C_\varepsilon = 0$.

Solving the inverse function of $f_{I'}$ and $g_{\kappa'}$ into R^M, we have

$$\begin{cases} z_{I'} = f_{I'}^{-1}(\xi_1, \xi_2, \ldots, \xi_M), \\ z_{\kappa'} = g_{\kappa'}^{-1}(\xi_1, \xi_2, \ldots, \xi_M). \end{cases} \tag{12}$$

Calculate $C_{\Omega'}$ in R^M as

$$C_{\Omega'}(I', \kappa_j') = \sum_{i=1}^{n} z_{I'} \times z_{\kappa'} \times h'_{(I', \kappa')}. \tag{13}$$

The following form equals $z_{I'} \times z_{\kappa'}$ in square difference form:

$$\begin{aligned} C_{\Omega'}(I', \kappa_{j'}) &= C_{\mathrm{sq}} + C_{\mathrm{res}} \\ &= -\frac{1}{2} \sum_{i=1}^{n} (z_{\kappa'} - z_{I'})^2 \times h'_{(I', \kappa')} + \frac{1}{2} \sum_{i=1}^{n} (z_{I'}^2 + z_{\kappa'}^2) \times h'_{(I', \kappa')}. \end{aligned} \tag{14}$$

By dividing the mapping of $(z_{\kappa'} - z_{I'})$ into n' parts, we obtain

$$C_{\mathrm{sq}}(I', \kappa_{j'}) = -\frac{1}{2} \sum_{d\lambda}^{n'} \left[\left(g^{-1}(\lambda) - f^{-1}(\lambda) \right) \times \sqrt{h'} \right]^2. \tag{15}$$

Define $\lambda = \{\xi_1, \xi_2, \ldots, \xi_M\}$. If the condition of the segmentation is sufficient, we have

$$C_{\mathrm{sq}}(I', \kappa_{j'}) = -\frac{1}{2} \lim_{d\lambda \to 0} \sum_{\lambda} \left[(g^{-1}(\lambda) - f^{-1}(\lambda)) \times \sqrt{h'} \right]^2. \tag{16}$$

Eq. (16) satisfies the definition of a multiple integration in R^M. Finally, we obtain the computational result C_{ij} for the entire layer I, where each element is a convolution output approximated by $C_{\Omega'}$.

Now we expand the convolution operation from a discrete space Ω into continuous manifolds ψ_S.

Differential geometry allows us to take problems from a variety of fields, including statistics, information theory, and control theory. The idea is to find more meaningful information from their connections about data in a high-dimensional space.

We focus on the convolution operation between a kernel and an input data sample, both represented by matrices of different dimension (the dimension of a kernel is usually less than an input instance).

We first define the mapping from $\Omega_{(x,y,z)}$ to ψ_S.

Lemma 1. Let M and N be manifolds with respective coordinate systems $\psi_M : M_\zeta = [\zeta_1, \zeta_2, \ldots, \zeta_a]$ and $\psi_N : N_\xi = [\xi_1, \xi_2, \ldots, \xi_b]$. A mapping $\Theta : M \to N$ is said to be C^∞ (infinitely many times differentiable) or smooth if the mapping $\Theta^{-1} : N \to M$ is C^∞ from an open subset R^m to R^n. If a C^∞ mapping Θ is a bijection (i.e., one-to-one mapping) and the inverse Θ^{-1} is also C^∞, then Θ is called a C^∞ diffeomorphism from M to N.

Lemma 1 proposes the conditions of mapping from two manifolds [18]. Actually, applying a high-dimensional function to approximate a given dataset is a common method to extract latent features [13]. Other methods have been proposed to deal with different sizing data, such as transfer learning [12] and dictionary learning [9], based on a matrix to reshape diverse data into a sharing dimension, with excellent results.

Let f_Ω and κ_Ω in R^3 be smooth curved surfaces. The transform from C_Ω to C_{ψ_S} exists on the condition:

$$\lim P(f_\Omega, \kappa_\Omega) \subset P(f_{\psi_S}, \kappa_{\psi_S}), \tag{17}$$

where P is a defined operation, the slice on R^3 is infinitely small, U is an open set in C_{ψ_S}, $\Theta^{-1}(\cdot)$ is the inverse function to solve corresponding independent variables, and $\Theta^{-1}(U) \in C_\Omega$.

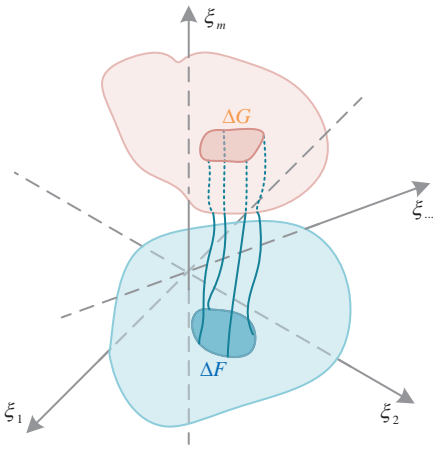

Figure 3 (Color online) Convolution in manifold $\psi_S = \{\xi_1, \xi_2, \ldots, \xi_m\}$. ΔF and ΔG are two small slices from F_{ψ_S} and G_{ψ_S}.

Theorem 1. ψ_S is an equivalent expansion manifold from $\Omega_I(x, y, z)$ in R^m satisfying three conditions.

(1) Each element in ψ_S is a bijection from coordinate system $S_\xi = \{\xi_1, \xi_2, \ldots, \xi_m\}$ to an exclusive open subset R^m.

(2) ψ_S is m-dimensional C^∞ differentiable.

(3) $\Omega_I(x, y, z)$ is a three-dimensional differentiable open set calculated from ψ_S.

We give an example in Figure 3, where the convolution operation of C_{ψ_S} in the hyperspace is defined as

$$C_{\psi_S} = \iint \cdots \int_{\Omega_S} (G_\xi - F_\xi) \mathrm{d}\xi_1 \cdots \mathrm{d}\xi_m. \tag{18}$$

The manifold F_ξ represents the original data I, and G_ξ represents the kernel κ. An original data record is a bounded set in R^3. The transfer from R^3 to R^m is a bounded mapping, ensuring the multiple integral operation in Ω_S is upper and lower bounded.

Theorem 1 guarantees the transformation in (17) is stable when reducing the dimension from $M + 1$ to M. Rewrite $C_{\Omega'}(I', \kappa_{j'})$ in (14) as

$$C_{\Omega'}(I', \kappa_{j'}) = -\frac{1}{2} C_{\psi_S}^2 + \frac{1}{2} \int \cdots \int_{\Omega_S} \left(G_\xi^2 + F_\xi^2\right) \mathrm{d}\xi_1 \cdots \mathrm{d}\xi_m. \tag{19}$$

Followed by average pooling and full connecting layer, we obtain the final output of the neural networking model.

4 Learning dynamics of convolution kernel

4.1 Spatial radial basis function kernel convolution

The RBFNN model is well designed for a nonlinear function approximating a continuous form. A basic form is shown in Figure 4.

The mixed structure is accepted to describe the relationship between influential factors owing to the interconnection among neurons, where X is sent to the hidden layer for nonlinear mapping, the activation functions adopted here have a Gaussian form, and the output of node l activates x_m by

$$u_{ml} = f(x_m, \mu_l, \sigma_l) = \exp\left(-\frac{\|x_m - \mu_l\|^2}{2\sigma_l^2}\right), \tag{20}$$

where $u \in R^{(M-m+1)}$, σ_j is the normalized constant of kernel k_l, $\mu_l \in R^{m \times m}$ is the spread center, and the feature map formed by u_{ml} is achieved from the hidden layer to the output layer.

The natural logarithm function $\ln(\cdot)$ is applied before the pooling stage; for the l-th feature map $u_{(\cdot,l)}$, we have

$$I_l = -\ln(u_{(\cdot,l)}). \tag{21}$$

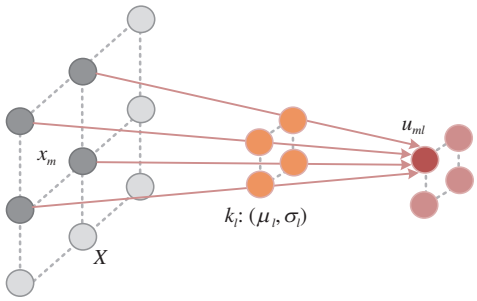

Figure 4 (Color online) Structure of RBF convolution: input $x_m \in R^{i\times i}$ accepts data from real systems ($X \in R^{M\times M}$); kernel k_l is determined by μ_l and σ_l; corresponding output is u_{ml}.

The natural logarithm function transfers u into the restricted area $[-\infty, 0)$ by $\ln(u_{(\cdot,l)})$; we use the reverse form to ensure the normal gradient backpropagation.

I_l is then used for common average pooling as in (4). The fully connected layer is given by

$$y_k = \sum_{i}^{M}\sum_{j}^{K} w_k I_l + b_k, \tag{22}$$

where w_k are adjusting weights from the hidden layer to the output layer, and b_k is the predefined bias. y_k is the response signal to the corresponding input; it transferred to workspace depending on the application requirement.

Remark 2. It can be seen that $u_{\max}(x, \mu, \sigma) = 1$. Assume there are L hidden neurons, $w \in [0, 1]$. Then the output of the system is

$$y_{\text{out}} = \sum w_j \exp\left(-\frac{(x-\mu_j)^2}{2\sigma_j^2}\right) \leqslant K \cdot w_{\max}. \tag{23}$$

Moreover, let $\hat{\mu}$ be the maximum weight; then we get

$$w_j \exp\left(-\frac{(x-\mu_j)^2}{2\sigma_j^2}\right) < 1, \quad x > \hat{\mu}. \tag{24}$$

For arbitrary x,

$$\lim_{x\to\pm\infty} w_j \exp\left(-\frac{(x-\mu_j)^2}{2\sigma_j^2}\right) = 0. \tag{25}$$

As a result, the system output is limited within the upper bound K, activated by a transfer function in the Gaussian form, and the system output is stable.

A system state expression can be obtained based on this method, and it is easy to analyze the stability of the system, but the actual operation system contains redundant information beyond extracted utilization, where its stability analysis is limited and cannot provide detailed parameters for application design or model settings.

4.2 Spatial transfer function

A control model with three transfer function parts and one feedback part is shown in Figure 5. x_m is the input instance; G_1, G_2, and G_3 are respectively the kernel, adjustment, and soft mapping parts; and H is the feedback from the system output.

Let the spatial transfer function activate input x_i by the kernel G_1 (size 2×2). Then the feature map is formed by

$$f_1 = \begin{pmatrix} G_{11} & G_{12} \\ G_{13} & G_{14} \end{pmatrix} x_i. \tag{26}$$

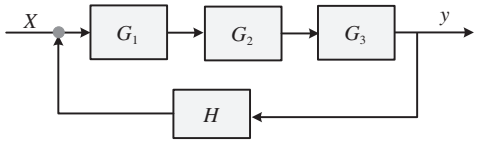

Figure 5 Computing flow from input X to output y, with transfer functions from G_1, G_2, G_3, and feedback H.

The natural logarithm function $\ln(\cdot)$ of G_2 regularizes the feature map from G_1. We assign G_2 the value -1 to control the output to be bounded positive. Then we get

$$f_2 = G_2(f_1) = -\ln\left(\exp\left(-\frac{\|x_m - \mu_l\|^2}{2\sigma_l^2}\right)\right) = \frac{\|x_m - \mu_l\|^2}{2\sigma_l^2}. \tag{27}$$

Using the average pooling operation in G_3, we have

$$y = G_3(f_2) = \frac{1}{4}\sum_{m=1}^{4}\frac{\|x_m - \mu_l\|^2}{2\sigma_l^2}. \tag{28}$$

Let x_m denote a small slice from input matrix X. G_1 is the l-th kernel, and a combination of G_3 forms output y as a fully connected layer, adjusted by weights θ_{kl}.

$$Y_k = \frac{1}{4}\sum_{k=1}^{K}\left(\sum_{l=1}^{L}\theta_{kl}\frac{\|x_k - \mu_l\|^2}{2\sigma_l^2}\right). \tag{29}$$

The integration from input to output is approximately equal to

$$\begin{aligned}\tilde{Y}_k &= \frac{1}{8}\sum_{k=1}^{K}\sum_{l=1}^{L}\theta_{kl}\left(\frac{x_k - \mu_l}{\sigma_l}\right)^2\\ &= \frac{1}{8}\sum_{k=1}^{K}\sum_{l=1}^{L}\theta_{kl}\left(\frac{x_k^2 + \mu_l^2 - 2x_k\mu_l}{\sigma_l^2}\right)\\ &= \frac{1}{8}\sum_{k=1}^{K}\sum_{l=1}^{L}\theta_{kl}\frac{x_k^2}{\sigma_l^2} + \frac{1}{8}\sum_{k=1}^{K}\sum_{l=1}^{L}\theta_{kl}\frac{\mu_l^2}{\sigma_l^2} - \frac{1}{4}\sum_{k=1}^{K}\sum_{l=1}^{L}\theta_{kl}\frac{x_k\mu_l}{\sigma_l^2}.\end{aligned} \tag{30}$$

We seek to embed the original input data X accompanied by target Y_0 into proper manifolds $F_\xi^k \in \psi_S$. Define the kernel set $g_\kappa(l)$ into $G_\xi^l \in \psi_S$. The goal is to minimize the distance by computing and optimizing the integral computation among large samples,

$$\begin{cases}\min\|\sum(Y_0 - Y_k)\|,\\ C_{\psi_S}(k,l) = \int\cdots\int_{\Omega_S}(G_\psi^l - F_\psi^k)\mathrm{d}\xi_1\cdots\mathrm{d}\xi_m,\end{cases} \tag{31}$$

where $C_{\psi_S}(k,l)$ is a scalar, θ_{kl} is the weight of its activation function, and k is the number of all instances in small batches.

Error-based parameter modification is effective in most networking models. Suppose the current iteration output is $y_k(t)$. By comparing the reference value y_0, the loss function based on an error $e(t)$ is produced by

$$l(t) = \frac{1}{2}\left(y_0 - y_k(t)\right)^2. \tag{32}$$

The gradient descent method to adjust an element in the kernel is iterated by

$$w(t+1) = w(t) - \eta\frac{l(t)|_w}{w(t)}. \tag{33}$$

The learning rate η is a predefined constant. In some improved methods [19, 20], it can be updated by solving Fisher information metric real-time. This kernel-based convolution proceeds in a scalar way, while the information about the kernel structure and direction may be neglected, including many deep learning structures considering the influence of the directional kernel [21, 22].

Theorem 2. Let $\psi_G : G_\xi = [\xi_1, \xi_2, \ldots, \xi_m]$. The partial differentiable function at ξ_i is

$$k(\xi_i) = \frac{\partial G_\xi}{\partial \xi_i} = \frac{\partial G(\xi_1, \xi_2, \ldots, \xi_m)}{\partial \xi_i}. \tag{34}$$

The tangent space T_p represents the directional weight at point $p \in G_\xi$, corresponding to the convolution kernel manifolds, by forming all partial differentiable functions:

$$T_p = \{k(\xi_1), k(\xi_2), \ldots, k(\xi_m)\}|_p. \tag{35}$$

The vector $k(\xi_i)$ is parallel to the i-th coordinate of ξ_i, and a small step between points p and p' can be described by $pp' = d\xi_i k(\xi_i)$, with a fixed constant of all the rest ξ_j $(j \neq i)$.

The gradient field V_ψ of G_ψ is

$$V_\psi = k(\xi_1) \cdot \xi_1 + k(\xi_2) \cdot \xi_2 + \cdots k(\xi_m) \cdot \xi_m. \tag{36}$$

Assume point p is located in a small slice ΔG. The weight of p at the t-th iteration is

$$w_p(t) = V_\psi|_p, \tag{37}$$

where $w_p(t)$ is a point in T_p, and the magnitude of $w_p(t)$ is

$$\left\| V_\phi|_p \right\| = \sqrt{k(\xi_1)|_p^2 + \cdots + k(\xi_m)|_p^2}. \tag{38}$$

Because ψ_S is C^∞ differentiable, T_p is a smooth curve in ψ'_S, and $w_p(t) \approx w_{\Delta G}(t)$ holds, we have

$$w_{\Delta G}(t) = \left[\left[p_1(t) \ \cdots \ p_m(t) \right]\right], \tag{39}$$

where $w_{\Delta G}(t)$ is a tensor based on all substrates ξ. The point p contains the directional information of $w_{\Delta G}(t)$. Element-to-element multiplication in a matrix is an operation in R^m, and w is adjusted in a scalar way.

4.3 Kernel learning in state spatial equation

To reveal the state of a kernel, we apply the spatial matrix to map the system status, data of both input and output. Theoretically, the dynamic process for a control model is represented by convergence speed, accuracy, and stability. Let the status of kernels be represented by κ, input data by u, and the full connected layer parameter by C. Then the state space expression can be described by the state space equation:

$$\begin{cases} \dot{\kappa} = A(t)\kappa + B(t)u, \\ y = C(t)\kappa, \end{cases} \tag{40}$$

where $\dot{\kappa}$ is the differential form of kernels, the learning dynamics are described by A and B, and the output y is determined by C and kernel κ. A group of independent state vectors $\kappa^{\mathrm{T}}(t) = [\kappa_1(t), \kappa_2(t), \ldots, \kappa_n(t)]$ constitutes a first-order differential equation of the system, A is the n-order system matrix, $u^{\mathrm{T}} = [u_1, u_2, \ldots, u_r]$ is an r-dimensional input vector, $y^{\mathrm{T}} = [y_1, y_2, \ldots, y_m]$ is the m-dimensional output vector, B is the input matrix $(n \times r)$, and C is the output matrix $(m \times n)$. The direct transmission matrix $D \in (m \times r)$ is not used because there are no direct connections from source to end.

Remark 3. The control model of kernels evolved from (42) indicates the dynamical processing of system convergence. To be specific, $\dot{\kappa}$ is expected to be weakened when the system is stabilized. In the stepwise case, we have another form for κ,

$$\frac{\kappa(t+1) - \kappa(t)}{\kappa(t)} = A(t)\kappa(t) + B(t)u. \tag{41}$$

Note that $A(t)$ indicates the system control matrix throughout all epochs, which is time-varying; hence, we need to quantify it in the time domain. Because this training strategy is ample, we consider it as a

Table 1 Network parameter settings of comparative deep neural network models

Description	AlexNet [24]	VGG-19 [25]	Resnet [26]	CNN	K-CNN
Layers	8	19	152	5	5
Convolution layer	5	16	151	2	2
Full-connected layer	3	3	1	1	1
Parameters (M)	64.2	172.5	23.6	1.7	3.3

Table 2 Identification comparison results on MNIST dataset

Result		AlexNet	VGG-19	Resnet	CNN	K-CNN
iter = 50	Train accuracy (%)	64.22	47.81	75.24	82.45	79.84
	Valid accuracy (%)	58.49	45.66	71.87	80.77	77.45
	Train loss	14.2785	48.2274	8.7784	2.2274	5.7782
	Valid loss	17.8847	50.2389	6.7947	2.3785	6.2845
iter = 100	Train accuracy (%)	77.55	65.28	81.47	87.34	82.23
	Valid accuracy (%)	73.41	64.59	80.25	85.75	81.54
	Train loss	10.1785	32.8954	4.7891	1.9887	3.7841
	Valid loss	12.5564	33.2786	4.5823	2.0149	3.6248
iter = 300	Train accuracy (%)	92.88	95.45	94.46	93.76	94.58
	Valid accuracy (%)	91.57	94.77	94.05	93.07	94.25
	Train loss	3.2247	2.8564	0.5641	0.3976	0.5875
	Valid loss	3.4896	2.8713	0.5826	0.4378	0.5924

full excitation. The direct connection from input to $\kappa(t)$ is zero, hence we let $B(t) = 0$. Owing to the RBFNN structure, the kernel operation error equals the network loss, and we have

$$\frac{F\left(x;\kappa(t+1)\right) - F\left(x;\kappa(t)\right)}{\kappa(t)} = A(t)\kappa(t). \tag{42}$$

The loss function is

$$L\left(x;\kappa(t+1)\right) = \kappa(t)A(t)\kappa(t). \tag{43}$$

As a result, we can obtain the kernel control matrix $A(t)$ in a stepwise way.

5 Experiment

We validated the dynamics of the proposed kernel-based convolution on the MNIST and CIFAR-10 datasets. The MNIST database of handwritten digits has a training set of 60000 28×28 examples and a testing set of 10000 examples [23], while the CIFAR-10 dataset contains 60000 32×32 RGB images, which are divided into 10 classes. Furthermore, we compared the performance of some popular deep learning models including AlexNet, VGG-19, Resnet, and our kernel-based CNNs, with hyperparameter learning rate $\eta = 0.001$ and batch size $bz = 64$; the parameter descriptions are listed in Table 1 [24–26]. The original CNN was designed with five layers, including two convolutional layers (the size of conv.1 is 5×5, the sizes of conv.3 are 3×3 for MNIST and 5×5 for CIFAR), reshaped by two average pooling layers (avg-pool.2 and avg-pool.4, both size 2×2), and two fully-connected layers of dimension 10×1 corresponding to the target categories.

The recognition results at given training iterations are listed in Tables 2 and 3. The MNIST dataset is relatively simple for recognition. Many deep CNNs obtain good accuracy on this dataset. From Table 2, though some complex deep networks eventually attain high accuracy, training on the large scale of parameters requires more time for convergence. However, for the image classification task on the CIFAR dataset, in Table 3, AlexNet, VGG-19, and Resnet all have accuracy above 70.00% at the 100th iteration, and above 80.00% at the 300th iteration, much better than CNN or K-CNN, whose simple CNN structures may work insufficiently well owing to limited inner parameters.

Remark 4. The difference between the three deeper models (AlexNet, VGG-19, and Resnet) and the simple CNN models is determined by the controller order and operation mode. For example, VGG-19 has 16 convolution layers; hence their integration operations outnumber those of K-CNN or basic CNN. Consequently, the controller has more poles to reduce those unstable statuses in training. Furthermore,

Table 3 Identification comparison results on CIFAR dataset

Result		AlexNet	VGG-19	Resnet	CNN	K-CNN
iter = 50	Train accuracy (%)	50.49	37.94	68.65	20.35	17.26
	Valid accuracy (%)	46.84	37.21	67.37	16.74	16.15
	Train loss	2287.64	78.4496	12.6679	7.5471	28.9975
	Valid loss	2409.83	76.2256	13.4286	7.6523	30.6547
iter = 100	Train accuracy (%)	72.39	56.45	76.49	37.35	35.47
	Valid accuracy (%)	70.68	55.82	70.22	36.49	34.82
	Train loss	1895.48	42.2641	9.7185	4.7864	13.1476
	Valid loss	2054.37	39.7745	10.1642	4.8147	14.8457
iter = 300	Train accuracy (%)	80.42	84.21	88.46	66.28	75.18
	Valid accuracy (%)	80.21	81.45	85.35	64.53	69.59
	Train loss	134.15	12.3864	6.5894	2.2538	3.7716
	Valid loss	146.57	13.7413	7.1157	2.3129	4.1728

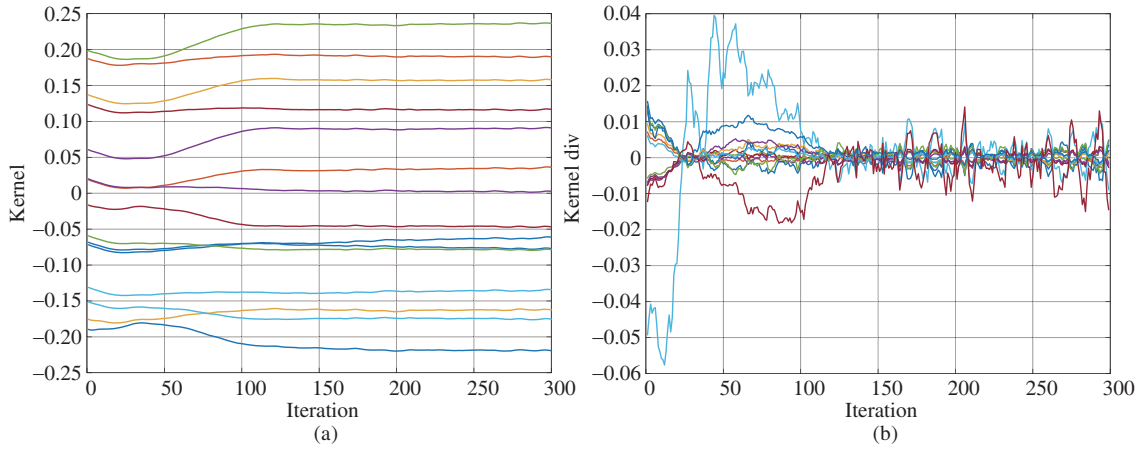

Figure 6 (Color online) Kernel learning dynamics of K-CNN at training MNIST dataset. (a) Adjusting status of 15 elements in the first layer; (b) first derivative values of 15 elements through 300 iterations.

the pooling operation smooths the intermediate convolution outputs like an inertial element that slows down training with more fluctuation.

The fundamental CNN with only five layers converged quickly with acceptable accuracy of about 93.07% after just 300 iterations, and its kernel-based method has better accuracy at the initial training stage than complex models such as AlexNet, VGG, and Resnet. It achieves a balance between accuracy and computational cost, with accuracy of 94.25% at the 300th iteration. The kernel element status of K-CNN is shown in Figure 6; this kernel is stabilized after about 130 iterations. The first derivative values indicate that the kernel has three stages during training, consisting of random gradient descent with vibrations, directional error reduction, and limited fluctuations.

To reveal the kernel dynamics of kernel-based CNN training on the MNIST dataset, we introduced the root locus and Nyquist diagram in the 50th, 100th, and 300th iterations, with results as shown in Figure 7, with some parameters listed in Tables 4–6. As seen in Figure 7(a), the kernel is not always stable because of random kernel initialization. The Nyquist diagram shows that point $(-1, j0)$ is not included in the trajectory circle, but there are two poles in Figure 7(b) in the right coordinates ($B \to C$, $A \to +\infty$), which confirm that the unstable training stage does exist with certain initializations, the stages of $D \to I$, $G \to F$ have limited stability, and $H \to -\infty$ is stable. In the second stage, from Figures 7(c) and (d), the system trajectory excludes the point $(-1, j0)$, and the training stability is much improved as the roots on the real axis are a limited zone ($A \to B$ and $C \to D$), while most roots are located at the left real axis, and stages $E \to F$, $G \to H \to I$ ensure system stability during batch training. From Figures 7(e) and (f), the system stability is enhanced by only a small circle for parameter adjustment. The root trajectory from $A \to B$ is reduced to a tiny zone, and the track from $C \to D$ converges to a small range.

The training process on the CIFAR-10 dataset has a similar trend to a kernel-based CNN. We illustrate the root locus and Nyquist diagram in Figure 8, and some critical status values are shown in Tables 7–9.

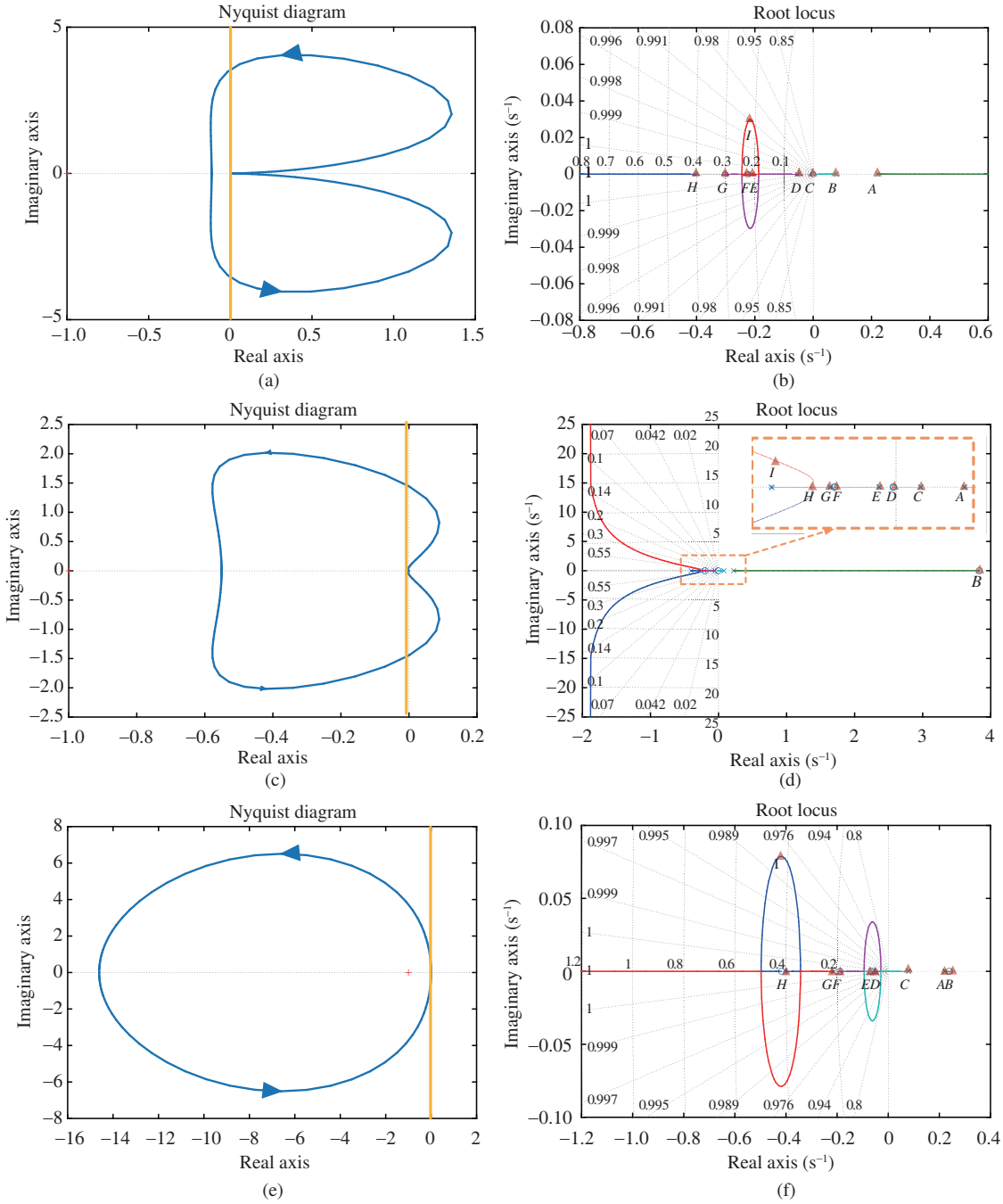

Figure 7 (Color online) Root locus and Nyquist diagram of control matrix A of CNN kernel at different iterations on MNIST dataset. (a) and (b) at 50th iteration; (c) and (d) at 100th iteration; (e) and (f) at 300th iteration. B is selected as the Jordan matrix of dimension 5×5; C is the softmax layer with each element not equal to zero.

Table 4 Kernel dynamics of K-CNN in training MNIST at 50th iteration

Parameter	A	B	C	D	E	F	G	H	I
Gain	0.052	0.0146	3.26	0.0159	0.286	3.4	18.1	0.128	1.28
Pole	0.225	0.0725	0.00148	−0.0575	−0.209	−0.237	−0.295	−0.41	−0.217 + 0.0297i
Damping	−1	−1	−1	1	1	1	1	1	0.991
Overshoot (%)	0	0	0	0	0	0	0	0	0
Frequency (rad·s^{-1})	0.225	0.0725	0.00148	0.0575	0.209	0.237	0.295	0.41	0.219

The Nyquist curve in Figure 8(a) is on the right panel with point $(-1, \mathrm{j}0)$ excluded, while there are poles A, C on the real axis in Figure 8(b), which lead the kernel is not stable, that is, the untrained learning

Table 5 Kernel dynamics of K-CNN in training MNIST at 100th iteration

Parameter	A	B	C	D	E	F	G	H	I
Gain	3.27e+04	0.00765	0	1.54	0.00838	1.68	0.11	0.517	11.8
Pole	3.62	0.223	0.0799	0.00155	-0.0544	-0.191	-0.215	-0.276	$-0.403 + 0.583$i
Damping	-1	-1	-1	-1	1	1	1	1	0.568
Overshoot (%)	0	0	0	0	0	0	0	0	11.4
Frequency (rad $\cdot$ s^{-1})	3.62	0.223	0.799	0.00155	0.0544	0.191	0.215	0.276	0.708

Table 6 Kernel dynamics of K-CNN in training MNIST at 300th iteration

Parameter	A	B	C	D	E	F	G	H	I
Gain	0.000884	2.19	0.00144	0.0248	0.799	2.14	0.0436	0.0899	0.469
Pole	0.221	0.238	0.0782	-0.0526	-0.0655	-0.185	-0.217	-0.398	$-0.42 + 0.0787$i
Damping	-1	-1	-1	1	1	1	1	1	0.983
Overshoot (%)	0	0	0	0	0	0	0	0	0
Frequency (rad $\cdot$ s^{-1})	0.221	0.238	0.0782	0.0526	0.0655	0.185	0.217	0.398	0.427

Table 7 Kernel dynamics of K-CNN in training CIFAR-10 at 50th iteration

Parameter	A	B	C	D	E	F	G	H	I
Gain	0	1.41	0.00647	1.24	0.0039	1.67	0.0038	0.0505	0.15
Pole	0.221	0.217	0.075	0.0074	-0.0563	-0.127	-0.215	-0.41	$-0.406 + 0.0324$i
Damping	-1	-1	-1	-1	1	1	1	1	0.997
Overshoot (%)	0	0	0	0	0	0	0	0	0
Frequency (rad $\cdot$ s^{-1})	0.221	0.217	0.075	0.0074	0.0563	0.127	0.215	0.41	0.408

Table 8 Kernel dynamics of K-CNN in training CIFAR-10 at 100th iteration

Parameter	A	B	C	D	E	F	G	H	I
Gain	0.0205	1.13	0.0033	0.0623	0.0084	0.392	0	0.249	0.302
Pole	0.233	0.235	0.0774	-0.0518	-0.0509	$-0.0543 + 0.0264$i	-0.212	-0.397	$-0.403 + 0.0149$i
Damping	-1	-1	-1	1	1	0.9	1	1	1
Overshoot (%)	0	0	0	0	0	0.155	0	0	11.4
Frequency (rad $\cdot$ s^{-1})	0.233	0.235	0.0774	0.0518	0.0509	0.0604	0.212	0.397	0.403

Table 9 Kernel dynamics of K-CNN in training CIFAR-10 at 300th iteration

Parameter	A	B	C	D	E	F	G	H	I
Gain	0.0177	1.52	0.0041	0.0213	7.82	0.0029	3.3	0.0359	0.408
Pole	0.223	0.238	0.0765	-0.0518	$-0.0866 + 0.0379$i	-0.213	0.398	-0.403	$-0.0603 + 0.0483$i
Damping	-1	-1	-1	1	0.916	1	1	1	0.78
Overshoot (%)	0	0	0	0	0.0769	0	0	0	0.14
Frequency (rad $\cdot$ s^{-1})	0.223	0.238	0.0765	0.0518	0.0946	0.213	0.398	0.403	0.0772

model requires sufficient training. In Figure 8(c), the trajectory circle surrounds $(-1, \mathrm{j}0)$ once, and in Figure 8(d), the zone $A \to B$ is reduced, and the path $C \to E$ goes through F not on the negative real axis, extending the learning period of the stable stage. Furthermore, the path in Figure 8(f) is extended to a much bigger processing; the learning is much reduced while the stabilized status is enhanced.

6 Conclusion

We have proposed a novel expression for manifolds of CNNs. The layered structure is preceded by continuous surface integration in a limited space, with weights adjusted, including the value and direction. The kernel-based CNN computing flow can be approximated by kernels in a high-dimensional manifold. Moreover, we analyzed the kernel control model in the frequency domain and proposed three stages of CNN training, consisting of vibrations, directional error reduction, and limited fluctuations. Generally, the unstable status in kernel-based CNN training drives the neural networking model to learn from supervised datasets. An experiment on the MNIST and CIFAR datasets revealed that the dynamics of

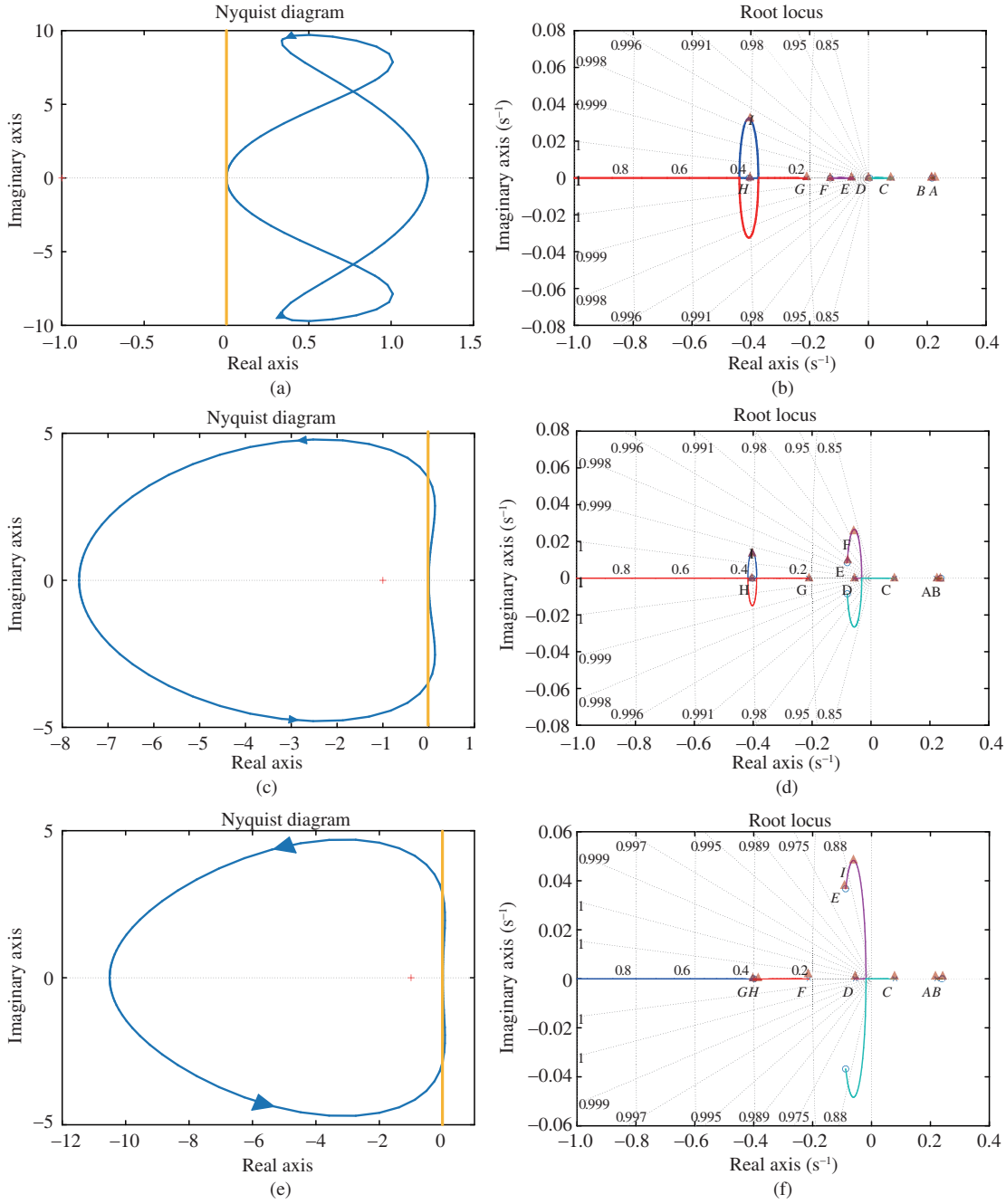

Figure 8 (Color online) Root locus and Nyquist diagram of control matrix A of CNN kernel at different iterations on CIFAR dataset. (a) and (b) at 50th iteration; (c) and (d) at 100th iteration; (e) and (f) at 300th iteration. B is selected as the Jordan matrix of dimension 5×5; C is the softmax layer with each element not equal to zero.

kernels in CNNs should be stable after proper training through root trajectories.

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

References

1 Chollet F. Xception: deep learning with depthwise separable convolutions. In: Proceedings of IEEE Conference on Computer Vision & Pattern Recognition, 2016. 1–8
2 Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the inception architecture for computer vision. In: Proceedings of IEEE Conference on Computer Vision & Pattern Recognition, 2016. 2818–2826

3 Zheng T Y, Chen G, Wang X Y, et al. Real-time intelligent big data processing: technology, platform, and applications. Sci China Inf Sci, 2019, 62: 082101
4 Yosinski J, Clune J, Nguyen A, et al. Understanding neural networks through deep visualization. In: Proceedings of the 31st International Conference on Machine Learning, 2015. 1–15
5 Brahma P P, Wu D, She Y. Why deep learning works: a manifold disentanglement perspective. IEEE Trans Neural Netw Learn Syst, 2016, 27: 1997–2008
6 Guo W L, Wei H K, Zhao J S, et al. Numerical analysis near singularities in RBF networks. J Mach Learn Res, 2018, 19: 1–39
7 Amari S, Park H, Ozeki T. Singularities affect dynamics of learning in neuromanifolds. Neural Computation, 2006, 18: 1007–1065
8 Sun H F, Peng L Y, Zhang Z N. Information geometry and its applications. Adv Math, 2011, 48: 75–102
9 Pratik C, Adam O, Stanley O, et al. Deep relaxation: partial differential equations for optimizing deep neural networks. Res Math Sci, 2018, 5: 1–22
10 Schilling R J, Carroll J J, Al-Ajlouni A F. Approximation of nonlinear systems with radial basis function neural networks. IEEE Trans Neural Netw, 2001, 12: 1–15
11 Wieland A P. Evolving neural network controllers for unstable systems. In: Proceedings of International Joint Conference on Neural Networks, 1991. 667–673
12 Scharf L L, Lytle D W. Stability of parameter estimates for a Gaussian process. IEEE Trans Aerosp Electron Syst, 1973, 9: 847–851
13 Vinogradska J, Bischoff B, Achterhold J, et al. Numerical quadrature for probabilistic policy search. IEEE Trans Pattern Anal Mach Intell, 2020, 42: 164–175
14 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
15 Buxhoeveden D P, Casanova M F. The minicolumn hypothesis in neuroscience. Brain, 2002, 125: 935–951
16 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
17 Faydasicok O, Arik S. A novel criterion for global asymptotic stabilityof neutral type neural networks with discrete time delays. In: Proceedings of International Conference on Neural Information Processing, 2018. 353–360
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. Natural gradient works efficiently in learning. Neural Comput, 1998, 10: 251–276
20 Wei H K, Zhang K J, Cousseau F, et al. Dynamics of learning near singularities in layered networks. Neural Comput, 2008, 20: 813–843
21 Sabour S, Frosst N, Hinton G E. Dynamic routing between capsules. In: Proceedings of the 31st Conference on Neural Information Processing Systems, 2017. 1–11
22 Li B. Parametric definition and production of directional convolution kernel. Chin J Comput, 1988, 11: 701–704
23 Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition. Proc IEEE, 1998, 86: 2278–2324
24 Alex K, Ilya S, Geoffrey E H. ImageNet classification with deep convolutional neural networks. In: Proceedings of the Conference and Workshop on Neural Information Processing Systems, 2012. 1–9
25 Karen S, Andrew Z. Very deep convolutional networks for large-scale image recognition. In: Proceedings of International Conference on Learning Representations, 2014. 1–14
26 He K M, Zhang X Y, Ren S Q. Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision & Pattern Recognition, 2015. 1–12