Received 30 January 2023, accepted 17 February 2023, date of publication 27 February 2023, date of current version 8 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3249280

RESEARCH ARTICLE

A Theoretical Framework for End-to-End Learning of Deep Neural Networks With Applications to Robotics

SITAN LI, HUU-THIET NGUYEN, AND CHIEN CHERN CHEAH, (Senior Member, IEEE)
School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798

Corresponding author: Chien Chern Cheah (ecccheah@ntu.edu.sg)

This work was supported by the Ministry of Education (MOE) Singapore, Academic Research Fund (AcRF) Tier 1, under Grant RG65/22.

ABSTRACT Deep Learning (DL) systems are difficult to analyze and proving convergence of DL algorithms like backpropagation is an extremely challenging task as it is a highly non-convex and high-dimensional problem. When using DL algorithms in robotic systems, theoretical analysis of stability, convergence, and robustness is a vital procedure as robots need to operate in a predictable manner to ensure safety. This paper presents the first unified End-to-End (E2E) learning framework that can be applied to both classification problems and real-time kinematic robot control tasks. In the proposed forward simultaneous learning method, the weights of all layers of the fully connected neural networks are updated concurrently. The proposed E2E learning framework uses an adjustable ReLU activation function so that convergence of the output error to a bound, which is dependent on the neural network approximation error, can be ensured. In particular, it is shown that the error between ideal output and estimated output of the network can converge to and stay in a certain bound, and this bound reduces to zero when the approximation error is zero. Therefore, the robustness of the learning system can be ensured even in the presence of the approximation error. Two case studies are done on classification tasks using MNIST and CIFAR10 datasets by using the proposed learning method, and the results show that the E2E learning method achieves comparable test accuracy as the gradient descent method and the main advantage is that convergence can be ensured during training. The framework is also implemented on a UR5e robot with unknown kinematic model, which is the first result on real-time E2E learning of deep neural networks for kinematic control of robots with guaranteed convergence.

INDEX TERMS Deep neural networks, end-to-end, online learning.

I. INTRODUCTION

Deep learning networks have been employed for many successful applications, but the majority of them are based on empirical results. The learning of deep neural networks (DNNs), is generally based on backpropagation and gradient descent [1], [2] which are well-known to be a black-box approach. The theoretical understanding of why deep learning succeeds in many cases remains obscure. Applying these deep neural networks in robotics creates a unique problem, as robots usually work in a safety-critical scenario. Using deep learning models that lack theoretical understandings for robotic applications could pose potential risks. Convergence analysis, for instance, is an important issue but has yet to receive sufficient considerations from machine learning scientists. Applying an algorithm of which convergence is not guaranteed to a robot system can make the system unstable and cause damages or accidents. Moreover, ignorance of the convergence and other theoretical issues in deep learning would hinder its future development to become more reliable and more trustworthy. Therefore, developing a theoretical learning framework for deep neural network structures where the convergence of the algorithms can be analyzed is crucial for deep learning to be deployed safely and robustly in robotic systems.

For the past decades, the convergence analysis of robot and control systems has focused on using Shallow Neural Networks. Most works focused on training only the output

The associate editor coordinating the review of this manuscript and approving it for publication was Prakasam Periasamy.

weights of the shallow network with linear output activation functions [3], [4], [5], [6], [7], [8], [9], [10] and others provided methods of training two weights together, including input and output weights [11], [12], [13]. However, as DNNs enjoy better generalization property than shallow ones for more general tasks [14], employing DNNs in robotic and control systems has always been an important research topic to explore [15]. In [16], a real time DNN adaptive control structure was presented, but the inner layers were first trained offline by stochastic gradient descent and then fixed when the output weights are updated in real-time. In [17], a modular adaptive DNN control scheme which consists of an update law for weights of output layer and estimation laws for setting constraints for weights of inner layers was also proposed to ensure stability in tracking control of dynamic systems. In [18], the Lyapunov-based stability analysis of real-time weight adaptation laws for each layer of a feed-forward DNN was presented. These results in [16], [17], and [18] are focusing on real-time control of dynamic systems.

Besides the applications of DNNs in real-time control tasks, DNNs are more widely applied in data related problems such as classification and regression problems with large amount of data generated. Recently, there has been an increasing attention in explainable artificial intelligence (XAI) [19], which leads to the interests of developing theoretical frameworks for understanding the convergence problems of E2E learning of DNNs based on gradient descent (GD) [20], [21], [22]. The training loss is computed at the final output layer, and then the gradients are calculated at each layer to update the weights. Various attempts have been made to theoretically analyze DNNs with E2E learning. In [20], it was shown that stochastic gradient descent (SGD) could find a global minima on the training of over-parameterized DNNs. In [21], it was proved that gradient descent could achieve zero training loss when training deep over-parameterized neural networks (NNs). This analysis relied on the special structure of Gram Matrix. In [22], it was shown that, for a binary classification problem, under certain assumptions of the training data, GD and SGD could find the global minima in the training of an over-parameterized deep network with ReLU activation functions. However, these results [20], [21], [22] are achieved based on the strong assumption that the DNNs are over-parameterized. This means each inner layer of DNNs must have an infinite or huge width. Therefore these over-parameterized DNNs are not feasible for most practical implementations and hence approximation errors always exist. Moreover, the formulation is based on the optimization perspective, which is hard to integrate with existing real-time robot control methods.

As it is challenging to directly analyze the high-dimensional problems in DNNs, one way is dissecting the DNNs into various-layer NNs, which is called layer-wise learning. This method builds and trains the DNNs by adding the layers or modules sequentially, and training is based on the shallow networks. This method was first proposed as a pre-training method called greedy layer-wise pre-training [23], [24]. Since it was a pre-training method, fine-tuning of the whole network using backpropagation of global errors was still required. Recently, this idea has emerged again as an independent learning method to train deep neural networks [25], [26], [27], [28], [29], [30], [31], [32], [33]. Some theoretical analyses have been developed based on layer-wise learning. In [30], the convergence analysis of deep linear networks was given based on a layer-wise learning using block coordinate gradient descent. It gave the optimal learning rate for training and analyzed the effects of width, depth and initialization of deep linear networks. However, deep linear networks are rarely used in practical scenarios as they have poor approximation ability compared to nonlinear ones. In [31], an analytic layer-wise learning framework was proposed for learning deep fully connected neural networks. The learning algorithm was tested on both classification problems and real-time kinematic control. In [32], a forward progressive learning framework was proposed for training and analyzing the deep convolutional neural networks (CNNs). The proposed method was evaluated on benchmarking datasets on classification problems. In [33], an analytic layer-wise deep learning framework was proposed for robot control, and the convergence of the tracking error is guaranteed in real-time.

Since the learning of DNNs is an iterative process, the major issue with layer-wise learning is that the training of the subsequent layers or blocks needs to wait until the previous ones are fully trained. Online learning in layer-wise methods is not feasible for most real-time robot systems as the weights cannot be updated simultaneously as a full deep network in an E2E manner, which is essential for real-time learning operations. It is limited to repetitive tasks as only one layer can be updated in each operation. In addition, the weights of inner layers or blocks are fixed once completing training and hence cannot be adjusted according to changes in tasks.

This paper presents an E2E learning framework with convergence analysis that can be applied to both classification problems and real-time kinematic control. The proposed forward simultaneous E2E learning algorithm allows all layers of a deep fully connected network (FNN) to be updated simultaneously without having to wait for previous layers, which saves computation time compared to layer-wise learning. Each layer is updated by a virtual learning system with proposed update laws that ensure the convergence of the training process. The update laws of the virtual learning systems are based on Adjustable ReLU activation function and batch updating, while also taking into account the effects of approximation error. The proposed theoretical framework is general and therefore can also be used in other end-to end learning algorithms or update laws, so long as the convergence of each virtual learning system can be shown. The main contributions of this paper are listed as follows:

i, Development of a unified forward simultaneous E2E learning framework to ensure the convergence and robustness of the learning systems in the presence of approximation errors. This framework ensures the error converges to a certain bound considering the existence of the approximation error of the network. In most implementations, there exists a finite error in learning or training in neural network approximation. We show that the output error converges to certain bound which is related to the size of the approximation error. This is the first unified theoretical framework for E2E learning of deep networks, that can be used for both classification tasks and real-time kinematic control tasks while ensuring convergence. The case studies show that the proposed algorithm can achieve similar performance compared to SGD and the main advantage is that the convergence can also be ensured.

ii, Development of a real-time task space kinematic control method for robots with unknown kinematic model, based on the proposed E2E learning framework. This is the first result on real-time E2E deep learning kinematic control of robots with guaranteed convergence.

iii, The concept of adjustable ReLU activation function is introduced in the update laws so that the convergence can be achieved without the process of pre-training, which was required in [31] and [32]. The proposed framework also does not require the assumptions of over-parameterized NNs as in [20], [21], and [22]. Unlike the existing results on E2E deep learning, the effects of approximation errors are taken into consideration in the theoretical analysis to ensure robustness.

The proposed framework is applied to two classification tasks using the MNIST [34] dataset on a deep FNN and CIFAR10 [35] dataset on a deep CNN using transfer learning. An online kinematic robot control task was also implemented using a UR5e manipulator with the proposed learning framework. It is shown that the proposed method can achieve comparable results to SGD for classification tasks while ensuring convergence. The online kinematic control of the industrial robot is the first result which demonstrates the feasibility of E2E deep learning control of robots in real-time. The experimental results show that the convergence of the tracking errors can be ensured.

II. PROBLEM STATEMENT

The aim of End-to-End learning is to train each layer's weights of a multilayer fully connected network concurrently. The output of the multilayer NN can be formulated as:

$$y_{NN}(k) = \boldsymbol{\sigma}\left(\mathbf{W}_n\boldsymbol{\phi}_{n-1}(\cdots \mathbf{W}_2\boldsymbol{\phi}_1(\mathbf{W}_1\boldsymbol{x}_1(k)))\right) \tag{1}$$

where $\boldsymbol{x}_1(k)$ is the kth original input data, the ideal weight matrices of all layers are represented as $\mathbf{W}_1, \ldots, \mathbf{W}_n$, the activation function of jth layer is represented as $\boldsymbol{\phi}_j, j = 1, 2 \ldots n-1$, the activation function of the final layer is represented as $\boldsymbol{\sigma}$.

One of the most commonly used activation function in DNNs is the Leaky ReLU activation function. It was found to enable better training of deeper networks [36] compared to previous widely used activation functions such as sigmoid and tanh, and has become the most popular activation function for deep neural networks [37].

Consider an input column vector $\boldsymbol{x}$ with a weight matrix $\mathbf{W}$ and an output activation function $\boldsymbol{\phi}$, then we have:

$$\boldsymbol{\phi}(\mathbf{W}\boldsymbol{x}) = \boldsymbol{\phi}\left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}\boldsymbol{x}\right) \tag{2}$$

Let $\boldsymbol{w}_j = [w_{j1} \quad w_{j2} \quad \cdots \quad w_{jn}]$ denote the jth row vector of the weight matrix, and ϕ_j denote the activation function corresponding to each $\boldsymbol{w}_j\boldsymbol{x}$, then equation (2) is written as:

$$\begin{aligned}\boldsymbol{\phi}(\mathbf{W}\boldsymbol{x}) &= \boldsymbol{\phi}\left(\begin{bmatrix}\boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_m\end{bmatrix}^T \boldsymbol{x}\right) \\ &= \begin{bmatrix}\phi_1(\boldsymbol{w}_1\boldsymbol{x}) \\ \phi_2(\boldsymbol{w}_2\boldsymbol{x}) \\ \vdots \\ \phi_m(\boldsymbol{w}_m\boldsymbol{x})\end{bmatrix}\end{aligned} \tag{3}$$

When $\boldsymbol{\phi}$ is the Leaky ReLU activation function, for each element $\phi_i(\boldsymbol{w}_i\boldsymbol{x}), i = 1, \ldots, m$, in $\boldsymbol{\phi}(\mathbf{W}\boldsymbol{x})$:

$$\phi_i(\boldsymbol{w}_i\boldsymbol{x}) = r_i\boldsymbol{w}_i\boldsymbol{x}, \; r_i = \begin{cases} 1, & \textit{for } \boldsymbol{w}_i\boldsymbol{x} \geq 0 \\ s, & \textit{for } \boldsymbol{w}_i\boldsymbol{x} < 0 \end{cases} \tag{4}$$

where s is a small postive constant and is usually set as 0.01 for a standard Leaky ReLU.

Using equation (4), Equation (3) can be written as:

$$\begin{aligned}\boldsymbol{\phi}(\mathbf{W}\boldsymbol{x}) &= \begin{bmatrix} r_1\boldsymbol{w}_1\boldsymbol{x} \\ r_2\boldsymbol{w}_2\boldsymbol{x} \\ \vdots \\ r_m\boldsymbol{w}_m\boldsymbol{x}\end{bmatrix} \\ &= \begin{bmatrix} r_1 & & & \\ & r_2 & & \\ & & \ddots & \\ & & & r_m \end{bmatrix}\mathbf{W}\boldsymbol{x}\end{aligned} \tag{5}$$

***Property* 1**:
The activation function $\boldsymbol{\phi}$ *of Leaky ReLU can be represented as a diagonal matrix* $\mathbf{R}$ *so that*:

$$\boldsymbol{\phi}(\mathbf{W}\boldsymbol{x}) = \mathbf{R}\mathbf{W}\boldsymbol{x} \tag{6}$$

where

$$\mathbf{R} = \begin{bmatrix} r_1 & & & \\ & r_2 & & \\ & & \ddots & \\ & & & r_m \end{bmatrix}, \; r_i = \begin{cases} 1, & \textit{for } \boldsymbol{w}_i\boldsymbol{x} \geq 0 \\ s, & \textit{for } \boldsymbol{w}_i\boldsymbol{x} < 0 \end{cases} \tag{7}$$

III. FORWARD SIMULTANEOUS END-TO-END LEARNING FOR FNN BASED ON BATCHES

In this section, we introduce the forward simultaneous End-to-End learning based on batches. The learning procedure is then summarized in Algorithm 1 of section III-C.

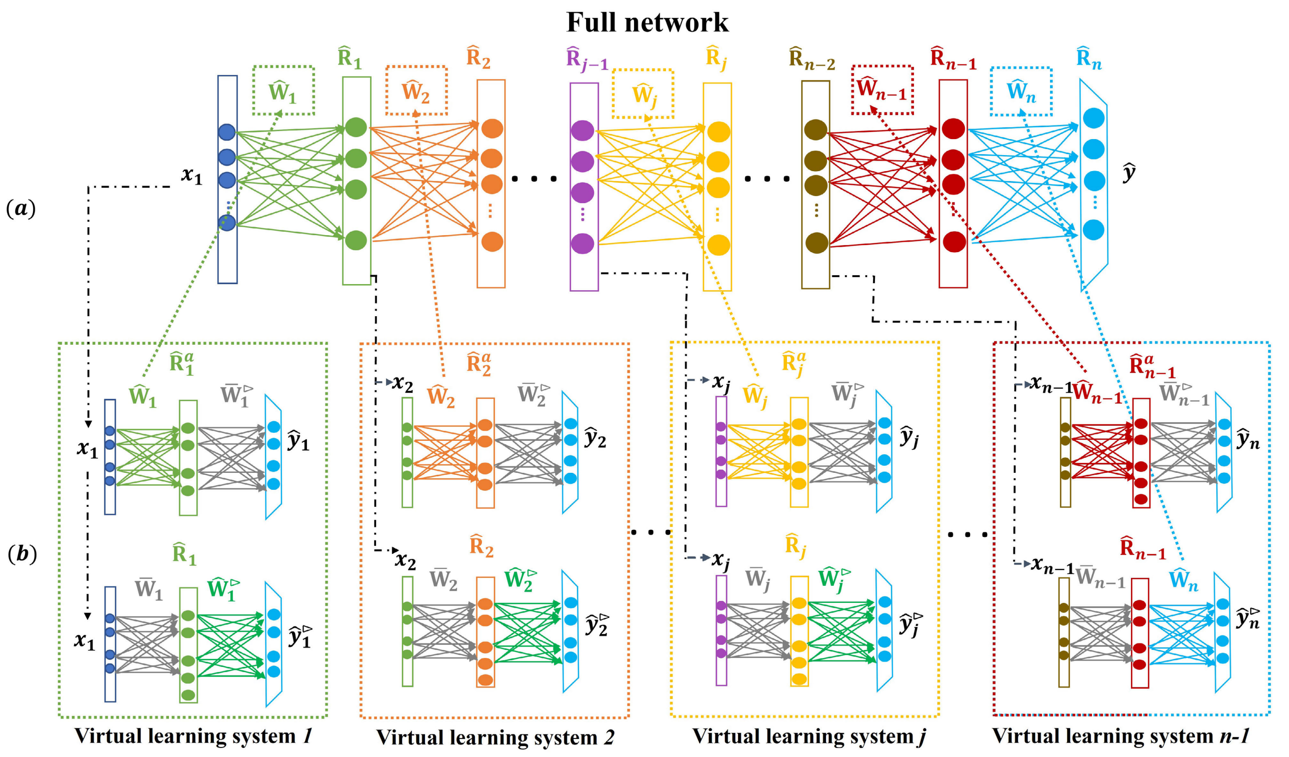
Full network
(a)
(b)
x_1
$\hat{y}$
Virtual learning system 1
Virtual learning system 2
Virtual learning system j
Virtual learning system n-1

FIGURE 1. A forward simultaneous End-to-End learning method, where an n layer multi-layer fully connected network is updated concurrently with $n-1$ virtual learning systems. (a) Full network– a deep fully connected network with $n-1$ hidden layers. (b) $n-1$ virtual learning systems. The input of each learning system x_j can be calculated by passing the original input through previous layers' weights. For each virtual learning system, only the input weights $\hat{\mathbf{W}}_j$ are kept for the full network.

A. FORWARD SIMULTANEOUS END-TO-END LEARNING PROCESS

Based on property 1, the target $\boldsymbol{y}(k)$ of an $n-1$ hidden layer FNN with the kth original input $\boldsymbol{x}_1(k)$ is formulated as:

$$\boldsymbol{y}(k) = \boldsymbol{\sigma}\left(\mathbf{W}_n\mathbf{R}_{n-1}(k)\cdots\mathbf{W}_2\mathbf{R}_1(k)\mathbf{W}_1\boldsymbol{x}_1(k)\right) + \boldsymbol{\epsilon}(k) \quad (8)$$

where $\boldsymbol{\epsilon}(k)$ is the approximation error, the ideal weight matrix of jth layer is represented as $\mathbf{W}_j$, and the ideal activation function matrix of jth layer is represented as $\mathbf{R}_j(k)$.

The complete structure of a deep fully connected network with $n-1$ hidden layers updated by E2E learning is shown in Fig 1. During training, when the kth input $\boldsymbol{x}_1(k)$ of a certain batch is fed into the network, all layer's weight matrices are updated as estimated weight matrices $\hat{\mathbf{W}}_1, \hat{\mathbf{W}}_2, \ldots, \hat{\mathbf{W}}_n$ respectively as shown in Fig 1(a), and therefore the estimated output of the multilayer NN is formulated as:

$$\hat{y}(k) = \boldsymbol{\sigma}\left(\hat{\mathbf{W}}_n(k)\hat{\mathbf{R}}_{n-1}(k)\cdots\hat{\mathbf{W}}_2(k)\hat{\mathbf{R}}_1(k)\hat{\mathbf{W}}_1(k)\boldsymbol{x}_1(k)\right) \quad (9)$$

For ease of presentation, we represent the input of jth layer during training as $\boldsymbol{x}_j(k)$,

$$\boldsymbol{x}_j(k) = \hat{\mathbf{R}}_{j-1}(k)\hat{\mathbf{W}}_{j-1}(k)\ldots\hat{\mathbf{W}}_2(k)\hat{\mathbf{R}}_1(k)\hat{\mathbf{W}}_1(k)\boldsymbol{x}_1(k) \quad (10)$$

where $\boldsymbol{x}_j(k)$ is a column vector with size of $(n_j \times 1)$, n_j is the number of neurons of jth layer, $\hat{\mathbf{W}}_t(k), t = 1, \ldots, j-1$ are estimated weights and $\hat{\mathbf{R}}_t(k)$ denotes the estimated activation function matrix which corresponds to the inputs $\hat{\mathbf{W}}_t(k)\boldsymbol{x}_t(k)$.

During the training process, $n-1$ virtual learning systems are designed in a forward manner as shown in Fig 1(b), meaning the input for each system is constructed by passing the original input through previous layers' weights. The systems are developed to train the corresponding layer's weights concurrently. Each virtual learning system is composed of two two-layer FNNs, one updates estimated input weights $\hat{\mathbf{W}}_j(k)$ and one updates estimated pseudo output weights $\hat{\mathbf{W}}_j^{\triangleright}(k), j = 1, 2, \ldots n-1$; and only the input weights $\hat{\mathbf{W}}_j(k)$ are kept after learning except for the last 2 layers represented by the $n-1$th learning system, where the output weights $\hat{\mathbf{W}}_{n-1}^{\triangleright}(k)$ are actually $\hat{\mathbf{W}}_n(k)$. Every virtual learning system updates the corresponding weight matrices of the full network simultaneously using the proposed update laws.

Before training, all the weights are randomly initialized as $\bar{\mathbf{W}}_j^{[0]}$ and $\bar{\mathbf{W}}_j^{\triangleright[0]}, j = 1, 2, \ldots n-1$ as illustrated in Fig 2(a). The training data is divided into multiple batches with batch size of p. When the 1st batch of data $x_j(k), k = 1, 2, \ldots p$ comes in, weights matrices $\hat{\mathbf{W}}_j(k)$ and $\hat{\mathbf{W}}_j^{\triangleright}(k)$ are updated at the same time. When a data in this batch comes in, the first virtual learning system learns the weights of the first layer $\hat{\mathbf{W}}_1(k)$ based on fixed $\bar{\mathbf{W}}_1^{\triangleright[0]}$. At the same time, the second virtual learning system learns the weight matrix $\hat{\mathbf{W}}_2(k)$ based on fixed $\bar{\mathbf{W}}_2^{\triangleright[0]}$, and the jth virtual learning system's input weight $\hat{\mathbf{W}}_j(k)$ is updated based on the fixed pseudo output weight $\bar{\mathbf{W}}_j^{\triangleright[0]}$. Also, the pseudo output weights

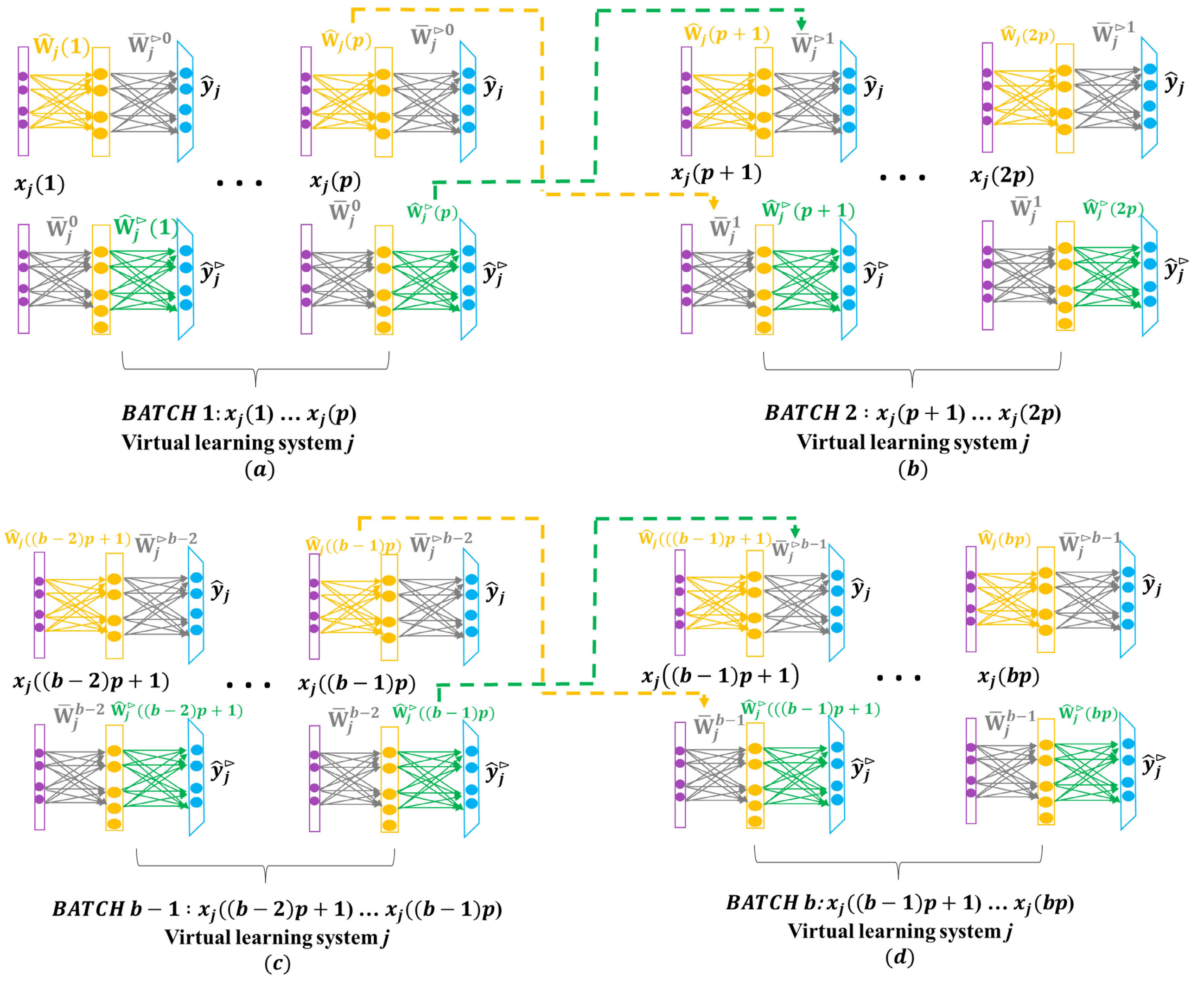
$\hat{W}_j(1)$ $\bar{W}_j^{\triangleright 0}$ $\hat{y}_j$
$x_j(1)$
$\hat{W}_j(p)$ $\bar{W}_j^{\triangleright 0}$ $\hat{y}_j$
$x_j(p)$
$\bar{W}_j^{0}$ $\hat{W}_j^{\triangleright}(1)$ $\hat{y}_j^{\triangleright}$
$\bar{W}_j^{0}$ $\hat{W}_j^{\triangleright}(p)$ $\hat{y}_j^{\triangleright}$
$\hat{W}_j(p+1)$ $\bar{W}_j^{\triangleright 1}$ $\hat{y}_j$
$x_j(p+1)$
$\hat{W}_j(2p)$ $\bar{W}_j^{\triangleright 1}$ $\hat{y}_j$
$x_j(2p)$
$\bar{W}_j^{1}$ $\hat{W}_j^{\triangleright}(p+1)$ $\hat{y}_j^{\triangleright}$
$\bar{W}_j^{1}$ $\hat{W}_j^{\triangleright}(2p)$ $\hat{y}_j^{\triangleright}$
$BATCH\ 1$: $x_j(1) \dots x_j(p)$
Virtual learning system j
(a)
$BATCH\ 2$: $x_j(p+1) \dots x_j(2p)$
Virtual learning system j
(b)
$\hat{W}_j((b-2)p+1)$ $\bar{W}_j^{\triangleright b-2}$ $\hat{y}_j$
$x_j((b-2)p+1)$
$\hat{W}_j((b-1)p)$ $\bar{W}_j^{\triangleright b-2}$ $\hat{y}_j$
$x_j((b-1)p)$
$\bar{W}_j^{b-2}$ $\hat{W}_j^{\triangleright}((b-2)p+1)$ $\hat{y}_j^{\triangleright}$
$\bar{W}_j^{b-2}$ $\hat{W}_j^{\triangleright}((b-1)p)$ $\hat{y}_j^{\triangleright}$
$\hat{W}_j(((b-1)p+1)$ $\bar{W}_j^{\triangleright b-1}$ $\hat{y}_j$
$x_j((b-1)p+1)$
$\hat{W}_j(bp)$ $\bar{W}_j^{\triangleright b-1}$ $\hat{y}_j$
$x_j(bp)$
$\bar{W}_j^{b-1}$ $\hat{W}_j^{\triangleright}(((b-1)p+1)$ $\hat{y}_j^{\triangleright}$
$\bar{W}_j^{b-1}$ $\hat{W}_j^{\triangleright}(bp)$ $\hat{y}_j^{\triangleright}$
$BATCH\ b-1$: $x_j((b-2)p+1) \dots x_j((b-1)p)$
Virtual learning system j
(c)
$BATCH\ b$: $x_j((b-1)p+1) \dots x_j(bp)$
Virtual learning system j
(d)

FIGURE 2. **Concurrent batch updating process of the forward simultaneous End-to-End learning method for the *j*th virtual learning system, with each batch contains *p* data. (a) updating of the *j*th virtual learning system using the 1st batch of data based on initialized weight $\bar{\mathbf{W}}_j^{[0]}$ and $\bar{\mathbf{W}}_j^{\triangleright[0]}$, $j = 1, 2, \dots n-1$. (b) Update the *j*th virtual learning system using the 2nd batch of data. Weight $\hat{\mathbf{W}}_j(p)$ and $\hat{\mathbf{W}}_j^{\triangleright}(p)$ from 1st batch will be used for 2nd batch updating, which means $\bar{\mathbf{W}}_j^{[1]} = \hat{\mathbf{W}}_j(p)$ and $\bar{\mathbf{W}}_j^{\triangleright[1]} = \hat{\mathbf{W}}_j^{\triangleright}(p)$. (c) Update the *j*th virtual learning system using the $b-1$th batch of data. (d) Update the *j*th virtual learning system using the *b*th batch of data. $\bar{\mathbf{W}}_j^{[b-1]} = \hat{\mathbf{W}}_j((b-1)p)$ and $\bar{\mathbf{W}}_j^{\triangleright[b-1]} = \hat{\mathbf{W}}_j^{\triangleright}((b-1)p)$.**

$\hat{\mathbf{W}}_1^{\triangleright}(k), \hat{\mathbf{W}}_2^{\triangleright}(k), \dots, \hat{\mathbf{W}}_j^{\triangleright}(k) \dots,$ are updated concurrently with the same data in the same batch when input weight matrices are fixed as $\bar{\mathbf{W}}_1^{[0]}, \bar{\mathbf{W}}_2^{[0]}, \dots, \bar{\mathbf{W}}_j^{[0]}, \dots$.

In this way, after training with the 1st batch of data, as shown in Fig 2(b), final weights of $\hat{\mathbf{W}}_1(p), \hat{\mathbf{W}}_2(p), \dots, \hat{\mathbf{W}}_{n-1}(p)$ are obtained as $\bar{\mathbf{W}}_1^{[1]}, \bar{\mathbf{W}}_2^{[1]}, \dots, \bar{\mathbf{W}}_{n-1}^{[1]}$. The final output weights after training are obtained as $\mathbf{W}^{\triangleright[1]}{}_1, \bar{\mathbf{W}}_2^{\triangleright[1]}, \dots, \bar{\mathbf{W}}_{n-1}^{\triangleright[1]}$. For the *j*th learning system, the input and output weights are obtained as: $\bar{\mathbf{W}}_j^{[1]} = \hat{\mathbf{W}}_j(p)$ and $\bar{\mathbf{W}}_j^{\triangleright[1]} = \hat{\mathbf{W}}_j^{\triangleright}(p)$. These weights are used for updating of the virtual learning systems for the next batch of data as illustrated in Fig 2(b).

After 1st batch is trained, the second batch's data comes in and all the weights are updated by repeating the previous procedure. Fig 2(d) shows the training procedure when *b*th batch of data $x_j(k), k = (b-1)p+1, (b-1)p+2, \dots bp$ comes in. The fixed weights of *b*th batch are obtained from $b-1$th batch, as shown in 2(c), as $\bar{\mathbf{W}}_j^{[b-1]} = \hat{\mathbf{W}}_j((b-1)p)$ and $\bar{\mathbf{W}}_j^{\triangleright[b-1]} = \hat{\mathbf{W}}_j^{\triangleright}((b-1)p)$. After all the data have been fed into the network, one epoch of training is completed and the training is repeated for multiple epochs. In the next subsection, update laws are developed to update all weights concurrently.

In this proposed learning framework, the weights of the full network are updated concurrently by the corresponding virtual learning systems. In a certain batch, all the systems work in an independent way and therefore are able to update the weights simultaneously, which reduces the computation time as compared to layerwise learning.

B. BATCH UPDATING LAWS FOR FORWARD SIMULTANEOUS END-TO-END LEARNING

As shown in Fig 1, each virtual learning system is a two-layer network that includes the input weight matrix $\hat{\mathbf{W}}_j$ and output matrix $\hat{\mathbf{W}}_j^{\triangleright}$. According to the training process briefly introduced in section III-A, virtual learning systems are designed to update estimated input and output weight matrices $\hat{\mathbf{W}}_j$ and $\hat{\mathbf{W}}_j^{\triangleright}$ concurrently.

Firstly, consider the data $\boldsymbol{x}_j(k)$ in the bth batch is fed into the jth virtual learning system during training, input weight matrix is updated as estimated input weight matrix $\hat{\mathbf{W}}_j(k)$ when output pseudo weight matrix $\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}$ is fixed, where $\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}$ is acquired from training of previous $b-1$th batch data. For the 1st batch, $\bar{\mathbf{W}}_j^{\triangleright^{[0]}}$ is obtained by random initialization. Hence, the estimated output of jth layer $\hat{\boldsymbol{y}}_j(k)$ is formulated as:

$$\hat{\boldsymbol{y}}_j(k) = \boldsymbol{\sigma}\left(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^a(k)\hat{\mathbf{W}}_j(k)\boldsymbol{x}_j(k)\right) \tag{11}$$

where $\boldsymbol{x}_j(k)$ is defined in (10) and $\hat{\mathbf{R}}_j^a(k)$ denotes an estimated Adjustable ReLU (AdjReLU) activation function matrix which corresponds to the inputs $\hat{\mathbf{W}}_j(k)\boldsymbol{x}_j(k)$, which is proposed as follows:

$$\hat{\mathbf{R}}_j^a(k) = \begin{bmatrix} \hat{r}_1(k) & & & \\ & \hat{r}_2(k) & & \\ & & \ddots & \\ & & & \hat{r}_m(k) \end{bmatrix},$$
$$\hat{r}_i(k) = \begin{cases} 1, & for\ \hat{\boldsymbol{w}}_i(k)\boldsymbol{x}(k) \geq 0 \\ a, & for\ \hat{\boldsymbol{w}}_i(k)\boldsymbol{x}(k) < 0 \end{cases} \tag{12}$$

Different from the leaky ReLU matrix $\mathbf{R}$ whose elements are fixed, the value of variable $a \in [0, 1]$ in AdjReLU matrix can be automatically adjusted during training for the purpose of ensuring the convergence.

There exists an ideal input weight matrix $\mathbf{W}_j^{[b]}$ corresponding to the fixed output weight matrix $\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}$, an ideal activation function matrix $\mathbf{R}_j^a(k)$ which corresponds to the inputs $\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k)$ and an approximation error $\boldsymbol{\epsilon}_j(k)$ such that the target of the network $\boldsymbol{y}(k)$ is:

$$\boldsymbol{y}(k) = \boldsymbol{\sigma}\left(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{R}_j^a(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k)\right) + \boldsymbol{\epsilon}_j(k) \tag{13}$$

Secondly, for the same virtual learning system, when kth input $\boldsymbol{x}_j(k)$ of the same batch of data is fed into the network during training, output weight matrix is updated as estimated weight matrix $\hat{\mathbf{W}}_j^{\triangleright}(k)$ when $\bar{\mathbf{W}}_j^{[b-1]}$ is fixed, where $\bar{\mathbf{W}}_j^{[b-1]}$ is acquired from training of previous $b-1$th batch data.. Hence, the estimated output of jth layer $\hat{\boldsymbol{y}}_j^{\triangleright}(k)$ is formulated as:

$$\hat{\boldsymbol{y}}_j^{\triangleright}(k) = \boldsymbol{\sigma}\left(\hat{\mathbf{W}}_j^{\triangleright}(k)\hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}\boldsymbol{x}_j(k)\right) \tag{14}$$

where $\hat{\mathbf{R}}_j(k)$ is an estimated activation function matrix which corresponds to the inputs $\bar{\mathbf{W}}_j^{[b-1]}\boldsymbol{x}_j(k)$.

There exists an ideal weight matrix $\mathbf{W}_j^{\triangleright^{[b]}}$, and an approximation error $\boldsymbol{\epsilon}_j^{\triangleright}(k)$ such that the target of the network $\boldsymbol{y}(k)$ is:

$$\boldsymbol{y}(k) = \boldsymbol{\sigma}\left(\mathbf{W}_j^{\triangleright^{[b]}}\hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}\boldsymbol{x}_j(k)\right) + \boldsymbol{\epsilon}_j^{\triangleright}(k) \tag{15}$$

After $\hat{\mathbf{W}}_j^{\triangleright}$ have been updated using this batch of data, it is denoted as $\bar{\mathbf{W}}_j^{\triangleright^{[b]}}$, which can be used in training of next batch of data.

From (11) and (13), the error $\boldsymbol{e}_j(k)$ between target $\boldsymbol{y}(k)$ and estimated output $\hat{\boldsymbol{y}}_j(k)$ can be formulated as:

$$\begin{aligned} \boldsymbol{e}_j(k) &= \boldsymbol{y}(k) - \hat{\boldsymbol{y}}_j(k) \\ &= \boldsymbol{\sigma}\Big(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^a(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k) + \bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k)\Big) \\ &\quad - \boldsymbol{\sigma}\left(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^a(k)\hat{\mathbf{W}}_j(k)\boldsymbol{x}_j(k)\right) + \boldsymbol{\epsilon}_j(k) \end{aligned} \tag{16}$$

where $\mathbf{E}_{R_j}(k) = \mathbf{R}_j^a(k) - \hat{\mathbf{R}}_j^a(k)$. Let

$$\begin{aligned} &\boldsymbol{\delta}_j(k) \\ &= \bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^a(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k) \\ &\quad - \bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^a(k)\hat{\mathbf{W}}_j(k)\boldsymbol{x}_j(k) + \boldsymbol{\epsilon}_j^E(k) \end{aligned} \tag{17}$$

where

$$\boldsymbol{\epsilon}_j^E(k) = \boldsymbol{\epsilon}_j(k) + \bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k) \tag{18}$$

Equation (17) can also be expressed as:

$$\boldsymbol{\delta}_j(k) = \bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^a(k)\Delta\mathbf{W}_j(k)\boldsymbol{x}_j(k) + \boldsymbol{\epsilon}_j^E(k) \tag{19}$$

where $\Delta\mathbf{W}_j(k) = \mathbf{W}_j^{[b]} - \hat{\mathbf{W}}_j(k)$. Using (14) and (15), the error $\boldsymbol{e}_j^{\triangleright}(k)$ between the target $\boldsymbol{y}(k)$ and estimated output $\hat{\boldsymbol{y}}_j^{\triangleright}(k)$ can be formulated as:

$$\begin{aligned} \boldsymbol{e}_j^{\triangleright}(k) &= \boldsymbol{y}(k) - \hat{\boldsymbol{y}}_j^{\triangleright}(k) \\ &= \boldsymbol{\sigma}\left(\mathbf{W}_j^{\triangleright^{[b]}}\hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}\boldsymbol{x}_j(k)\right) \\ &\quad - \boldsymbol{\sigma}\left(\hat{\mathbf{W}}_j^{\triangleright}(k)\hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}\boldsymbol{x}_j(k)\right) + \boldsymbol{\epsilon}_j^{\triangleright}(k) \end{aligned} \tag{20}$$

Let

$$\boldsymbol{\delta}_j^{\triangleright}(k) = \Delta\mathbf{W}_j^{\triangleright}(k)\hat{\mathbf{R}}_j(k)\,\bar{\mathbf{W}}_j^{[b-1]}\boldsymbol{x}_j(k) + \boldsymbol{\epsilon}_j^{\triangleright}(k) \tag{21}$$

where $\Delta\mathbf{W}_j^{\triangleright}(k) = \mathbf{W}_j^{\triangleright^{[b]}} - \hat{\mathbf{W}}_j^{\triangleright}(k)$.

The input weight matrix $\hat{\mathbf{W}}_j(k)$ is updated based on fixed weight $\bar{\mathbf{W}}q_j^{\triangleright^{[b-1]}}$ and error $\boldsymbol{e}_j(k)$, and the output weight matrix $\hat{\mathbf{W}}_j^{\triangleright}(k)$ is updated based on fixed weight $\bar{\mathbf{W}}_j^{[b-1]}$ and error $\boldsymbol{e}_j^{\triangleright}(k)$, the two weights are updated separately and concurrently based on the following two update laws:

$$\hat{\mathbf{W}}_j(k+1) = \hat{\mathbf{W}}_j(k) + \alpha_j\hat{\mathbf{R}}_j^a(k)(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\boldsymbol{x}_j^T(k) \tag{22}$$

$$\hat{\mathbf{W}}_j^{\triangleright}(k+1) = \hat{\mathbf{W}}_j^{\triangleright}(k) + \alpha_j^{\triangleright}\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k)(\hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}\boldsymbol{x}_j(k))^T \tag{23}$$

where α_j and $\alpha_j^{\triangleright}$ are positive scalars, $\mathbf{L}_j(k)$ and $\mathbf{L}_j^{\triangleright}(k)$ are positive diagonal matrices. Let $\boldsymbol{x}_{j+1}(k) = \hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}\boldsymbol{x}_j(k)$, then, $\hat{\mathbf{W}}_j(k)$ and $\hat{\mathbf{W}}_j^{\triangleright}(k)$ can be updated with the following update laws in vector form:

$$\hat{\boldsymbol{w}}_{j,i}(k+1) = \hat{\boldsymbol{w}}_{j,i}(k) + \alpha_j\boldsymbol{x}_{j,i}(k)\hat{\mathbf{R}}_j^a(k)(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k) \tag{24}$$

$$\hat{\boldsymbol{w}}_{j,i}^{\triangleright}(k+1) = \hat{\boldsymbol{w}}_{j,i}^{\triangleright}(k) + \alpha_j^{\triangleright}\boldsymbol{x}_{j+1,i}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k) \tag{25}$$

where $x_{j,i}(k)$ and $x_{(j+1),i}(k)$ are the ith element of $\boldsymbol{x}_j(k)$ and $\boldsymbol{x}_{j+1}(k)$, $\hat{\boldsymbol{w}}_{j,i}(k)$ and $\hat{\boldsymbol{w}}_{j,i}^{\triangleright}(k)$ denotes the ith column of the estimated weight matrices $\hat{\mathbf{W}}_j(k)$ and $\hat{\mathbf{W}}_j^{\triangleright}(k)$.

Let $\boldsymbol{\Delta w}_{j,i}(k) = \boldsymbol{w}_{j,i} - \hat{\boldsymbol{w}}_{j,i}(k)$ and $\boldsymbol{\Delta w}_{j,i}^{\triangleright}(k) = \boldsymbol{w}_{j,i}^{\triangleright} - \hat{\boldsymbol{w}}_{j,i}^{\triangleright}(k)$, $\boldsymbol{w}_{j,i}$ and $\boldsymbol{w}_{j,i}^{\triangleright}$ denotes the ith column of the ideal weight matrix $\mathbf{W}_j$ and $\mathbf{W}_j^{\triangleright}$, the update law (24) and (25) can be written as:

$$\begin{aligned}\boldsymbol{\Delta w}_{j,i}(k+1) &= \boldsymbol{\Delta w}_{j,i}(k) - \alpha_j x_{j,i}(k)\hat{\mathbf{R}}_j^a(k)(\bar{\mathbf{W}}_j^{\triangleright}[b-1])^T \\ &\quad\times \mathbf{L}_j(k)\boldsymbol{e}_j(k) \qquad (26)\\ \boldsymbol{\Delta w}_{j,i}^{\triangleright}(k+1) &= \boldsymbol{\Delta w}_{j,i}^{\triangleright}(k) - \alpha_j^{\triangleright}x_{j+1,i}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k) \qquad (27)\end{aligned}$$

The objective function $V(k)$ at kth step is proposed as:

$$\begin{aligned}V(k) &= \sum_{j=1}^{n-1}\frac{1}{\alpha_j}\sum_{i=1}^{n_j}\boldsymbol{\Delta w}_{j,i}^T(k)\boldsymbol{\Delta w}_{j,i}(k)\\ &\quad+\sum_{j=1}^{n-1}\frac{1}{\alpha_j^{\triangleright}}\sum_{i=1}^{n_j^{\triangleright}}\boldsymbol{\Delta w}_{j,i}^{\triangleright T}(k)\boldsymbol{w}_{j,i}^{\triangleright}(k) \qquad (28)\end{aligned}$$

where n_j and $n_j^{\triangleright}$ are the number of neurons of jth input layer and output layer. Therefore, the objective function at $k+1$th step is:

$$\begin{aligned}V(k+1) &= \sum_{j=1}^{n-1}\frac{1}{\alpha_j}\sum_{i=1}^{n_j}\boldsymbol{\Delta w}_{j,i}^T(k+1)\boldsymbol{\Delta w}_{j,i}(k+1)\\ &\quad+\sum_{j=1}^{n-1}\frac{1}{\alpha_j^{\triangleright}}\sum_{i=1}^{n_j^{\triangleright}}\boldsymbol{\Delta w}_{j,i}^{\triangleright T}(k+1)\boldsymbol{\Delta w}_{j,i}^{\triangleright}(k+1)\end{aligned} \tag{29}$$

Using equations (26) and (27), we have:

$$\begin{aligned}&V(k+1)\\ &= \sum_{j=1}^{n-1}\Bigg(\frac{1}{\alpha_j}\sum_{i=1}^{n_j}(\boldsymbol{\Delta w}_{j,i}(k) - \alpha_j x_{j,i}(k)\hat{\mathbf{R}}_j^a(k)(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k))^T\\ &\quad(\boldsymbol{\Delta w}_{j,i}(k) - \alpha_j x_{j,i}(k)\hat{\mathbf{R}}_j^a(k)(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k))\Bigg)\\ &\quad+\sum_{j=1}^{n-1}\Bigg(\frac{1}{\alpha_j^{\triangleright}}\sum_{i=1}^{n_j^{\triangleright}}(\boldsymbol{\Delta w}_{j,i}^{\triangleright}(k) - \alpha_j^{\triangleright}x_{j+1,i}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k))^T\\ &\quad(\boldsymbol{\Delta w}_{j,i}^{\triangleright}(k) - \alpha_j^{\triangleright}x_{j+1,i}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k))\Bigg)\end{aligned} \tag{30}$$

From kth step to $k+1$th step, the change in objective function can be expressed as $\Delta V(k)$.

$$\begin{aligned}&\Delta V(k)\\ &= V(k+1) - V(k)\\ &= \sum_{j=1}^{n-1}\Bigg(\sum_{i=1}^{n_j}-x_{j,i}(k)\Delta\boldsymbol{w}_{j,i}^T(k)\hat{\mathbf{R}}_j^a(k)(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\\ &\quad-\boldsymbol{e}_j^T(k)\mathbf{L}_j^T(k)\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^a(k)\boldsymbol{\Delta w}_{j,i}(k)x_{j,i}(k)\\ &\quad+\alpha_j\boldsymbol{e}_j^T(k)\mathbf{L}_j^T(k)\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^a(k)x_{j,i}^2(k)\hat{\mathbf{R}}_j^a(k)\\ &\quad\times(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\Bigg)\\ &\quad-\sum_{j=1}^{n-1}\Bigg(\sum_{i=1}^{n_j^{\triangleright}}\boldsymbol{\Delta w}_{j,i}^{\triangleright T}(k)x_{j+1,i}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k)\\ &\quad-\boldsymbol{e}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright T}(k)x_{j+1,i}(k)\boldsymbol{\Delta w}_{j,i}^{\triangleright}(k)\\ &\quad+\alpha_j^{\triangleright}\boldsymbol{e}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright T}(k)x_{j+1,i}^2(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k))\Bigg)\end{aligned} \tag{31}$$

Using equation (19) and (21), $\Delta V(k)$ is then formulated as:

$$\begin{aligned}&\Delta V(k)\\ &\quad= -\sum_{j=1}^{n-1}\Bigg((\boldsymbol{\delta}_j(k) - \boldsymbol{\epsilon}_j^E(k))^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\\ &\qquad+\boldsymbol{e}_j^T(k)\mathbf{L}_j^T(k)(\boldsymbol{\delta}_j(k) - \boldsymbol{\epsilon}_j^E(k))\\ &\qquad-\boldsymbol{e}_j^T(k)\alpha_j\mu_j(k)\mathbf{L}_j^T(k)\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^{a2}(k)\\ &\qquad\times(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\\ &\qquad-(\boldsymbol{\delta}_j^{\triangleright}(k) - \boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k)\\ &\qquad+\boldsymbol{e}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright T}(k)(\boldsymbol{\delta}_j^{\triangleright}(k) - \boldsymbol{\epsilon}_j^{\triangleright}(k))\\ &\qquad-\boldsymbol{e}_j^{\triangleright T}(k)\alpha_j^{\triangleright}\rho_j(k)\mathbf{L}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k)\Bigg)\end{aligned} \tag{32}$$

where $\mu_j(k) = \sum_{i=1}^{n_j}\hat{x}_{j,i}^2(k)$ and $\rho_j(k) = \sum_{\iota=1}^{n_j^{\triangleright}}x_{j+1,\iota}^2(k)$.

Consider the term $\boldsymbol{\epsilon}_j^{E^T}(k)\mathbf{L}_j(k)\boldsymbol{e}_j(k)$ in (32), where $\boldsymbol{\epsilon}_j^E(k)$ is consisted of approximation error $\boldsymbol{\epsilon}_j(k)$ and a term $\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k)$ as described in (18). Note that

$$\begin{aligned}\boldsymbol{\epsilon}_j^{E^T}(k)\mathbf{L}_j(k)\boldsymbol{e}_j(k) &= \boldsymbol{\epsilon}_j{}^T(k)\mathbf{L}_j(k)\boldsymbol{e}_j(k)\\ &\quad+(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k))^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\end{aligned} \tag{33}$$

Since $\mathbf{E}_{R_j}(k) = \mathbf{R}_j^a(k) - \hat{\mathbf{R}}_j^a(k)$, we have:

$$\mathbf{E}_{R_j}(k) = \begin{bmatrix} r_1(k) - \hat{r}_1(k) & & \\ & \ddots & \\ & & r_{n_j}(k) - \hat{r}_{n_j}(k)\end{bmatrix} \tag{34}$$

where $r_i(k)$ correspond to $\boldsymbol{w}_i^b\boldsymbol{x}_j(k)$ and $\hat{r}_i(k)$ correspond to $\hat{\boldsymbol{w}}_i\boldsymbol{x}_j(k)$.

During training, the variable a of AdjReLU activation function $\mathbf{R}_j^a(k)$ defined in (12) is adjusted such that the term

$(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k))^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)$ in (33) is close to zero.

Each element in $\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k)$ is represented as:

$$\big(r_i(k)-\hat{r}_i(k)\big)\boldsymbol{w}_i^b\boldsymbol{x}(k) = \begin{cases} 0, & \textit{if } \boldsymbol{w}_i^b\boldsymbol{x}_j(k), \hat{\boldsymbol{w}}_i(k)\boldsymbol{x}_j(k) \textit{ are of same sign} \\ (1-a)\boldsymbol{w}_i^b\boldsymbol{x}(k), & \textit{if } \boldsymbol{w}_i^b\boldsymbol{x}(k), \hat{\boldsymbol{w}}_i(k)\boldsymbol{x}_j(k) \textit{ are of opposite sign} \end{cases} \quad (35)$$

At the beginning phase, the variable a is selected as 1 so that the ideal AdjReLU activation function matrix $\mathbf{R}_j^a(k)$ and estimated AdjReLU $\hat{\mathbf{R}}_j^a(k)$ are both identity matrices. Therefore, each element $|r_i(k)-\hat{r}_i(k)|$ in the error $\mathbf{E}_{R_j}(k)$ equals to zero and that $(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k))^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)=0$

If errors $\boldsymbol{e}_j(k)$ are small after training with $a=1$, the estimated term $\hat{\boldsymbol{w}}_i(k)\boldsymbol{x}(k)$ is close to the ideal term $\boldsymbol{w}_i^b\boldsymbol{x}(k)$ and if they have the same sign, then, according to (35), $\big(r_i(k)-\hat{r}_i(k)\big)\boldsymbol{w}_i^b\boldsymbol{x}(k)=0$. Since the change of sign occurs at 0, when $\boldsymbol{w}_i^b\boldsymbol{x}(k)$ and $\hat{\boldsymbol{w}}_i\boldsymbol{x}(k)$ are of opposite sign, $\boldsymbol{w}_i^b\boldsymbol{x}(k)$ should be close to zero. Therefore, $\big(r_i(k)-\hat{r}_i(k)\big)\boldsymbol{w}_i^b\boldsymbol{x}(k)=(1-a)\boldsymbol{w}_i^b\boldsymbol{x}(k)$ are also small. Therefore, variable a can be adjusted according to the error $\boldsymbol{e}_j(k)$, so that the term $(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k))^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\approx 0$.

Hence, at the beginning phase, the value of a can be set as 1 ensuring that $(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k))^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)=0$. At the ending phase, the value of a can be adjusted from 1 towards a small positive scalar to introduce nonlinearity in the activation function while ensuring $(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\mathbf{E}_{R_j}(k)\mathbf{W}_j^{[b]}\boldsymbol{x}_j(k))^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\approx 0$. In that case, equation (33) has become:

$$\boldsymbol{\epsilon}_j^{E^T}(k)\mathbf{L}_j(k)\boldsymbol{e}_j(k)\approx\boldsymbol{\epsilon}_j^T(k)\mathbf{L}_j(k)\boldsymbol{e}_j(k) \quad (36)$$

Using (36), $\Delta V(k)$ is then expressed as:

$$\begin{aligned}\Delta V(k) = -\sum_{j=1}^{n-1}\Big(&(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\\ &+\boldsymbol{e}_j^T(k)\mathbf{L}_j^T(k)(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))\\ &-\boldsymbol{e}_j^T(k)\alpha_j\mu_j(k)\mathbf{L}_j^T(k)\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^{a^2}(k)\\ &\times(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\\ &+(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k)\\ &+\boldsymbol{e}_j^{\triangleright^T}(k)\mathbf{L}_j^{\triangleright^T}(k)(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))\\ &-\boldsymbol{e}_j^{\triangleright^T}(k)\alpha_j^{\triangleright}\rho_j(k)\mathbf{L}_j^{\triangleright^T}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k)\Big)\end{aligned} \quad (37)$$

Let d_{jM}, $d_{jM}^{\triangleright}$ be positive constants such that:

$$\boldsymbol{e}_j^T(k)\alpha_j\mu_j(k)\mathbf{L}_j^T(k)\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}\hat{\mathbf{R}}_j^{a^2}(k)(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k) \le d_{jM}L_{jM}^2\|\boldsymbol{e}_j(k)\|^2 \quad (38)$$

$$\boldsymbol{e}_j^{\triangleright^T}(k)\alpha_j^{\triangleright}\rho_j(k)\mathbf{L}_j^{\triangleright^T}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k) \le d_{jM}^{\triangleright}L_{jM}^{\triangleright^2}\|\boldsymbol{e}_j^{\triangleright}(k)\|^2 \quad (39)$$

Here L_{jM} and $L_{jM}^{\triangleright}$ denotes maximum eigenvalues of the matrix $\boldsymbol{L}_j(k)$ and $\boldsymbol{L}_j^{\triangleright}(k)$ for any k. Therefore, $\Delta V(k)$ is written as:

$$\begin{aligned}\Delta V(k) \le \sum_{j=1}^{n-1}\Big(&-(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))^T\mathbf{L}_j(k)(\boldsymbol{e}_j(k)-\boldsymbol{\epsilon}_j(k))\\ &-(\boldsymbol{e}_j(k)-\boldsymbol{\epsilon}_j(k))^T\mathbf{L}_j^T(k)(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))\\ &-(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))^T\mathbf{L}_j(k)\boldsymbol{\epsilon}_j(k)\\ &-\boldsymbol{\epsilon}_j(k)^T\mathbf{L}_j^T(k)(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))\\ &+d_{jM}L_{jM}^2\|\boldsymbol{e}_j(k)\|^2\Big)\\ &+\Big(-(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright}(k)(\boldsymbol{e}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))\\ &-(\boldsymbol{e}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright^T}(k)(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))\\ &-(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright}(k)\boldsymbol{\epsilon}_j^{\triangleright}(k)\\ &-\boldsymbol{\epsilon}_j^{\triangleright}(k)^T\mathbf{L}_j^{\triangleright^T}(k)(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))\\ &+d_{jM}^{\triangleright}L_{jM}^{\triangleright^2}\|\boldsymbol{e}_j^{\triangleright}(k)\|^2\Big)\end{aligned} \quad (40)$$

The activation functions of output layer $\boldsymbol{\sigma}$ are chosen as monotonically increasing and with bounded derivative whose upper bound is $f_{j\sigma M}$ and lower bound is $f_{j\sigma m}$. Comparing between $\boldsymbol{e}_j(k)-\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{e}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)$ in (16),(20) and $\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)$ in (19),(21), then:

i, the ith element of $\boldsymbol{e}_j(k)-\boldsymbol{\epsilon}_j(k)$ and $\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{e}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)$ and $\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)$ are of the same sign, i.e.

$$\begin{aligned}&(e_{j_i}(k)-\epsilon_{j_i}(k))(\delta_{j_i}(k)-\epsilon_{j_i}(k))\ge 0,\\ &\quad(e_{j_i}^{\triangleright}(k)-\epsilon_{j_i}^{\triangleright}(k))(\delta_{j_i}^{\triangleright}(k)-\epsilon_{j_i}^{\triangleright}(k))\ge 0,\ \forall i=1..p\end{aligned} \quad (41)$$

ii, the absolute value of the ith element of $\boldsymbol{e}_j(k)-\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{e}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)$ are no more than $f_{j\sigma M}$ times the ith elements of $\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)$ i.e.

$$\begin{aligned}&|e_{j_i}(k)-\epsilon_{j_i}(k)|\le f_{j\sigma M}|\delta_{j_i}(k)-\epsilon_{j_i}(k)|,\\ &\qquad|e_{j_i}^{\triangleright}(k)-\epsilon_{j_i}^{\triangleright}(k)|\\ &\qquad\le f_{j\sigma M}|\delta_{j_i}^{\triangleright}(k)-\epsilon_{j_i}^{\triangleright}(k)|,\ \forall i=1..p\end{aligned} \quad (42)$$

iii, the absolute value of the ith element of $\boldsymbol{e}_j(k)-\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{e}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)$ are no less than $f_{j\sigma m}$ times the corresponding elements of $\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)$, i.e.

$$\begin{aligned}&f_{j\sigma m}|\delta_{j_i}(k)-\epsilon_{j_i}(k)|\le|e_{j_i}(k)-\epsilon_{j_i}(k)|,\\ &\qquad f_{j\sigma m}|\delta_{j_i}^{\triangleright}(k)-\epsilon_{j_i}^{\triangleright}(k)|\\ &\qquad\le|e_{j_i}^{\triangleright}(k)-\epsilon_{j_i}^{\triangleright}(k)|,\ \forall i=1..p\end{aligned} \quad (43)$$

From the properties stated in (41), (42), the following inequalities can be assured for:

$$\begin{aligned}
&-(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))^T\mathbf{L}_j(k)(\boldsymbol{e}_j(k)-\boldsymbol{\epsilon}_j(k))\\
&\quad-(\boldsymbol{e}_j(k)-\boldsymbol{\epsilon}_j(k))^T\mathbf{L}_j^T(k)(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))\\
&\leq\frac{-2L_{jm}}{f_{j\sigma M}}(\|\boldsymbol{e}_j(k)\|^2+\|\boldsymbol{\epsilon}_j(k)\|^2-2\|\boldsymbol{e}_j(k)\|\|\boldsymbol{\epsilon}_j(k)\|) \quad (44)\\
&-(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright}(k)(\boldsymbol{e}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))\\
&\quad-(\boldsymbol{e}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright T}(k)(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))\\
&\leq\frac{-2L_{jm}^{\triangleright}}{f_{j\sigma M}}(\|\boldsymbol{e}_j^{\triangleright}(k)\|^2+\|\boldsymbol{\epsilon}_j^{\triangleright}(k)\|^2-2\|\boldsymbol{e}_j^{\triangleright}(k)\|\|\boldsymbol{\epsilon}_j^{\triangleright}(k)\|)
\end{aligned} \tag{45}$$

where L_{jm}, $L_{jm}^{\triangleright}$ is the minimum eigenvalues of the matrix $\boldsymbol{L}_j(k), \boldsymbol{L}_j^{\triangleright}(k)$ for any k.

The terms $(-(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))^T\mathbf{L}_j(k)\boldsymbol{\epsilon}_j(k)-\boldsymbol{\epsilon}_j(k)^T\mathbf{L}_j^T(k)$ $(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k)))$ in (40) is smaller than zero if $(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))^T$ and $\boldsymbol{\epsilon}_j(k)$ are of the same signs. However, since $(\boldsymbol{\delta}_j(k)-\boldsymbol{\epsilon}_j(k))^T$ and $\boldsymbol{\epsilon}_j(k)$ can be different signs, from the properties stated in (41), (43), the following inequality can be used:

$$\begin{aligned}
&(-\boldsymbol{\delta}_j(k)+\boldsymbol{\epsilon}_j(k))^T\mathbf{L}_j(k)\boldsymbol{\epsilon}_j(k)\\
&\quad+\boldsymbol{\epsilon}_j(k)^T\mathbf{L}_j^T(k)(-\boldsymbol{\delta}_j(k)+\boldsymbol{\epsilon}_j(k))\\
&\leq\frac{2L_{jM}}{f_{j\sigma m}}\|\boldsymbol{e}_j(k)\|\|\boldsymbol{\epsilon}_j(k)\|+\frac{2L_{jM}}{f_{j\sigma m}}\|\boldsymbol{\epsilon}_j(k)\|^2
\end{aligned} \tag{46}$$

Same for the term $(-(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright}(k)\boldsymbol{\epsilon}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)^T\mathbf{L}_j^{\triangleright T}(k)(\boldsymbol{\delta}_j^{\triangleright}(k)-\boldsymbol{\epsilon}_j^{\triangleright}(k)))$ in (40), from the properties stated in (41), (43), we can get the following inequality:

$$\begin{aligned}
&(-\boldsymbol{\delta}_j^{\triangleright}(k)+\boldsymbol{\epsilon}_j^{\triangleright}(k))^T\mathbf{L}_j^{\triangleright}(k)\boldsymbol{\epsilon}_j^{\triangleright}(k)\\
&\quad+\boldsymbol{\epsilon}_j^{\triangleright}(k)^T\mathbf{L}_j^{\triangleright T}(k)(-\boldsymbol{\delta}_j^{\triangleright}(k)+\boldsymbol{\epsilon}_j^{\triangleright}(k))\\
&\leq\frac{2L_{jM}^{\triangleright}}{f_{j\sigma m}}\|\boldsymbol{e}_j^{\triangleright}(k)\|\|\boldsymbol{\epsilon}_j^{\triangleright}(k)\|+\frac{2L_{jM}^{\triangleright}}{f_{j\sigma m}}\|\boldsymbol{\epsilon}_j^{\triangleright}(k)\|^2
\end{aligned} \tag{47}$$

Substituting inequalities (44),(45) and (46),(47) into (40), we have:

$$\begin{aligned}
\Delta V(k)\leq\sum_{j=1}^{n-1}\Bigg(&\frac{-2L_{jm}}{f_{j\sigma M}}(\|\boldsymbol{e}_j(k)\|^2+b_\epsilon^2-2\|\boldsymbol{e}_j(k)\|b_\epsilon)\\
&+\frac{-2L_{jm}^{\triangleright}}{f_{j\sigma M}}(\|\boldsymbol{e}_j^{\triangleright}(k)\|^2+b_\epsilon^{\triangleright 2}-2\|\boldsymbol{e}_j^{\triangleright}(k)\|b_\epsilon^{\triangleright})\\
&+\frac{2L_{jM}}{f_{j\sigma M}}\|\boldsymbol{e}_j(k)\|b_\epsilon+\frac{2L_{jM}}{f_{j\sigma M}}b_\epsilon^2\\
&+\frac{2L_{jM}^{\triangleright}}{f_{j\sigma M}}\|\boldsymbol{e}_j^{\triangleright}(k)\|b_\epsilon^{\triangleright}+\frac{2L_{jM}^{\triangleright}}{f_{j\sigma M}}b_\epsilon^{\triangleright 2}\\
&+d_{jM}L_{jM}^2\|\boldsymbol{e}_j(k)\|^2+d_{jM}^{\triangleright}L_{jM}^{\triangleright 2}\|\boldsymbol{e}_j^{\triangleright}(k)\|^2\Bigg)
\end{aligned} \tag{48}$$

Here b_ϵ, $b_\epsilon^{\triangleright}$ are upper bounds for approximation error $\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{\epsilon}_j^{\triangleright}(k)$ of the deep neural network for any k. Rearranging inequality (48) we can get:

$$\begin{aligned}
\Delta V(k)\leq\sum_{j=1}^{n-1}\Bigg(&-(\frac{2L_{jm}}{f_{j\sigma M}}-d_{jM}L_{jM}^2)\|\boldsymbol{e}_j(k)\|^2\\
&+(\frac{2L_{jM}}{f_{j\sigma m}}-\frac{2L_{jm}}{f_{j\sigma M}})b_\epsilon^2+(\frac{2L_{jM}}{f_{j\sigma m}}+\frac{4L_{jm}}{f_{j\sigma M}})\|\boldsymbol{e}_j(k)\|b_\epsilon\Bigg)\\
&+\Bigg(-(\frac{2L_{jm}^{\triangleright}}{f_{j\sigma M}}-d_{jM}^{\triangleright}L_{jM}^{\triangleright 2})\|\boldsymbol{e}_j^{\triangleright}(k)\|^2\\
&+(\frac{2L_{jM}^{\triangleright}}{f_{j\sigma m}}-\frac{2L_{jm}^{\triangleright}}{f_{j\sigma M}})b_\epsilon^{\triangleright 2}+(\frac{2L_{jM}^{\triangleright}}{f_{j\sigma m}}\\
&+\frac{4L_{jm}^{\triangleright}}{f_{j\sigma M}})\|\boldsymbol{e}_j^{\triangleright}(k)\|b_\epsilon^{\triangleright}\Bigg)
\end{aligned} \tag{49}$$

It can be shown that if

$$\begin{aligned}
&\|\boldsymbol{e}_j(k)\|\\
&\geq\frac{b_\epsilon}{2\left(\frac{2L_{jm}}{f_{j\sigma M}}-d_{jM}L_{jM}^2\right)}\Bigg[\frac{4L_{jm}}{f_{j\sigma M}}+\frac{2L_{jM}}{f_{j\sigma m}}\\
&+\sqrt{\frac{8L_{jm}}{f_{j\sigma M}}d_{jM}L_{jM}^2+\frac{8L_{jM}}{f_{j\sigma m}}\left(\frac{4L_{jm}}{f_{j\sigma M}}-d_{jM}L_{jM}^2\right)+\left(\frac{2L_{jM}}{f_{j\sigma m}}\right)^2}\,\Bigg]
\end{aligned} \tag{50}$$

$$\begin{aligned}
&\|\boldsymbol{e}_j^{\triangleright}(k)\|\\
&\geq\frac{b_\epsilon^{\triangleright}}{2\left(\frac{2L_{jm}^{\triangleright}}{f_{j\sigma M}}-d_{jM}^{\triangleright}L_{jM}^{\triangleright 2}\right)}\Bigg[\frac{4L_{jm}^{\triangleright}}{f_{j\sigma M}}+\frac{2L_{jM}^{\triangleright}}{f_{j\sigma m}}\\
&+\sqrt{\frac{8L_{jm}^{\triangleright}}{f_{j\sigma M}}d_{jM}^{\triangleright}L_{jM}^{\triangleright 2}+\frac{8L_{jM}^{\triangleright}}{f_{j\sigma m}}\left(\frac{4L_{jm}^{\triangleright}}{f_{j\sigma M}}-d_{jM}^{\triangleright}L_{jM}^{\triangleright 2}\right)+\left(\frac{2L_{jM}^{\triangleright}}{f_{j\sigma m}}\right)^2}\,\Bigg]
\end{aligned} \tag{51}$$

$$\text{and}\quad\frac{2L_{jm}}{f_{j\sigma M}}-d_{jM}L_{jM}^2>0,\ \frac{2L_{jm}^{\triangleright}}{f_{j\sigma M}}-d_{jM}^{\triangleright}L_{jM}^{\triangleright 2}>0 \tag{52}$$

then $\Delta V(k)\ \leq\ 0$ can be assured. This means that when the error is large, then $\Delta V(k)\ \leq\ 0$ and the error is therefore reduced until it reaches this the bound described by equation (50), (51), and finally stays inside it, ensuring convergence. So for a certain batch of data, the estimated weights of the multi layer NN are converging towards ideal weights for this batch. This process keeps repeating in batches until the error settles in a reasonable bound. Therefore, the error would stay within an ultimate bound after reaching it.

In the ideal case where, if $\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{\epsilon}_j^{\triangleright}(k)$ are zero for any k, from (50), (51) the bound reduces zero. If the approximation error can be considered as zero, then equation (37) can be rewritten as:

$$\begin{aligned}
\Delta V(k)=&-\sum_{j=1}^{n-1}\boldsymbol{\delta}_j^T(k)\mathbf{L}_j(k)\boldsymbol{e}_j(k)-\sum_{j=1}^{n-1}\boldsymbol{e}_j^T(k)\mathbf{L}_j^T(k)\boldsymbol{\delta}_j(k)\\
&-\sum_{j=1}^{n-1}\boldsymbol{\delta}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k)
\end{aligned}$$

$$
\begin{aligned}
&-\sum_{j=1}^{n-1} \boldsymbol{e}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright T}(k)\boldsymbol{\delta}_j^{\triangleright}(k)\\
&+\sum_{j=1}^{n-1} \boldsymbol{e}_j^{T}(k)\alpha_j\mu_j(k)\mathbf{L}_j^{T}(k)\\
&\times\bar{\mathbf{W}}_j^{\triangleright[b-1]}\hat{\mathbf{R}}_j^{a2}(k)(\bar{\mathbf{W}}_j^{\triangleright[b-1]})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\\
&+\sum_{j=1}^{n-1} \boldsymbol{e}_j^{\triangleright T}(k)\alpha_j^{\triangleright}\rho_j(k)\mathbf{L}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k) \quad (53)
\end{aligned}
$$

Removing the $\boldsymbol{\epsilon}_j(k)$, $\boldsymbol{\epsilon}_j^{\triangleright}(k)$ in property (41), (42), (43), the conditions still be assured. Therefore, the following inequality can be ensured:

$$
\begin{aligned}
&\Delta V(k)\\
&\leq -\sum_{j=1}^{n-1} \frac{2}{f_{\sigma m}}\boldsymbol{e}_j^{T}(k)\mathbf{L}_j(k)\boldsymbol{e}_j(k) - \sum_{j=1}^{n-1} \frac{2}{f_{\sigma m}}\boldsymbol{e}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright T}(k)\boldsymbol{e}_j^{\triangleright}(k)\\
&\quad+\sum_{j=1}^{n-1} \boldsymbol{e}_j^{T}(k)\alpha_j\mu_j(k)\mathbf{L}_j^{T}(k)\bar{\mathbf{W}}_j^{\triangleright[b-1]}\hat{\mathbf{R}}_j^{a2}(k)\\
&\quad\times(\bar{\mathbf{W}}_j^{\triangleright[b-1]})^T\mathbf{L}_j(k)\boldsymbol{e}_j(k)\\
&\quad+\sum_{j=1}^{n-1} \boldsymbol{e}_j^{\triangleright T}(k)\alpha_j^{\triangleright}\rho_j(k)\mathbf{L}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright}(k)\boldsymbol{e}_j^{\triangleright}(k) \quad (54)
\end{aligned}
$$

When $\mathbf{L}_j(k)$ and $\mathbf{L}_j^{\triangleright}(k)$ are chosen such that

$$
\frac{2}{f_\sigma}\mathbf{L}_j(k) - \left(\alpha_j\mu_j(k)\mathbf{L}_j^T(k)\bar{\mathbf{W}}_j^{\triangleright[b-1]}\hat{\mathbf{R}}_j^{a2}(k)(\bar{\mathbf{W}}_j^{\triangleright[b-1]})^T\mathbf{L}_j(k)\right) > 0 \quad (55)
$$

$$
\frac{2}{f_\sigma}\mathbf{L}_j^{\triangleright}(k) - \left(\alpha_j^{\triangleright}\rho_j(k)\mathbf{L}_j^{\triangleright T}(k)\mathbf{L}_j^{\triangleright}(k)\right) > 0 \quad (56)
$$

The above conditions ensure that $\Delta V(k) < 0$. Since the objective function of the n layer MLFN is non-positive, moreover, $V(k)$ is also bounded from below, $\Delta V(k)$ is converging as k increases, which also suggests that all errors are converging. α_j and $\alpha_j^{\triangleright}$ are selected small enough to ensure convergence. Also, theoretically, in each update the gain matrix $\mathbf{L}_j(k)$, $\mathbf{L}_j^{\triangleright}(k)$ can be automatically adjusted by (55) (56).

Remark: Note that the proposed framework is general as the convergence analysis is based on summation of multiple virtual learning systems. Therefore, other updating algorithms which the convergence of the each virtual system can be analyzed can also be similarly designed for End-to-End learning of deep neural networks by using the proposed framework.

C. SUMMARY OF END-TO-END LEARNING ALGORITHM

For a multi-layer fully connected network, formulated in (9), our algorithm updates each layer's weights $\hat{\mathbf{W}}_j, j = 1, \ldots, n$ of the MLFN concurrently. The learning algorithm of the n layer MLFN is presented in Algorithm 1, the details are described as follows:

(a) For each input data$(\boldsymbol{x}_1(k), \boldsymbol{y}(k))$ in the bth batch of training dataset:
   1. Formulate each virtual learning systems' input $\boldsymbol{x}_j(k)$ using (10) and output $\hat{\boldsymbol{y}}_j(k)$ $\hat{\boldsymbol{y}}_j^{\triangleright}(k)$ using (11) and (14).
   2. Calculate each virtual learning systems' error $\boldsymbol{e}_j(k) = \boldsymbol{y}(k) - \hat{\boldsymbol{y}}_j(k)$, $\boldsymbol{e}_j^{\triangleright}(k) = \boldsymbol{y}(k) - \hat{\boldsymbol{y}}_j^{\triangleright}(k)$.
   3. Concurrently update all $n-1$ input weight matrices $\hat{\mathbf{W}}_j(k)$ with fixed output weight matrix $\bar{\mathbf{W}}_j^{\triangleright[b-1]}$ and update all $n - 1$ output weight matrices $\hat{\mathbf{W}}_j^{\triangleright}(k)$ with fixed input weight matrix $\bar{\mathbf{W}}_j^{[b-1]}$ using (22) and (23). Fixed input and output weights for bth batch are obtained from $b-1$th batch: $\bar{\mathbf{W}}_j^{\triangleright[b-1]} = \hat{\mathbf{W}}_j(k)$, $\bar{\mathbf{W}}_j^{\triangleright[b-1]} = \hat{\mathbf{W}}_j^{\triangleright}(k)$, $k = (b-1)p$.
   4. When all the data in the bth batch have been trained, current weights $\hat{\mathbf{W}}_j(k)$ and $\hat{\mathbf{W}}_j^{\triangleright}(k)$ will be kept for $b+1$th batch's updating, which means $\bar{\mathbf{W}}_j^{\triangleright[b]} = \hat{\mathbf{W}}_j(k)$, $\bar{\mathbf{W}}_j^{\triangleright[b]} = \hat{\mathbf{W}}_j^{\triangleright}(k)$, $k = bp$.

(b) Continue with the next batch of training sample and repeat the steps described in (a) until all the input data in the training dataset have been fed into the deep FNN.

(c) After all the input data in the training dataset has been fed into the FNN, an epoch is completed. The learning is repeated for multiple epochs.

IV. CASE STUDIES

In this section, we present the testing results of the proposed forward simultaneous E2E learning algorithm on both classification tasks and online kinematic control task. The classification task is implemented on a fully connected network and classifier of a convolutional neural network using two datasets. The online control task is implemented on a UR5E industrial robot arm.

A. MNIST DATASET

MNIST dataset, which is a commonly used image dataset, is used to test our algorithm on classification problems using fully connected networks. The deep neural network is constructed as a 5-hidden layer network with the structure of 784-300-LReLU-200-LReLU-150-LReLU-100-LReLU-100-LReLU-10-sigmoid. The full network structure used standard Leaky ReLU activation functions with gradient of 0.01 when input is smaller than zero at hidden layers and sigmoid activation functions for the output layer.

The full network was trained based on five virtual learning systems which were running concurrently: 784-300-AdjReLU-10-sigmoid (virtual learning system I), 300-200-AdjReLU-10-sigmoid (virtual learning system II), 200-150-AdjReLU-10-sigmoid (virtual learning system III), 150-100-AdjReLU-10-sigmoid (virtual learning system IV), and 100-100-AdjReLU-10-sigmoid (virtual learning system V). Each virtual learning system updated the weights following update laws in (22) using AdjReLU and (23) using standard Leaky ReLU at hidden layers. The activation

Algorithm 1 Forward Simultaneous End-to-End Learning for Updating Each Layer's Weight $\hat{\mathbf{W}}_j$ of n Layer MLFN Based on Batches

```
1:  Initialization:
    $epoch = 0$,
    $b = 0$,
    $k = 0$,
    The weights of $n-1$ virtual learning systems are randomly initialized as $\bar{\mathbf{W}}_j^{[0]}$ and $\bar{\mathbf{W}}_j^{\triangleright[0]}$, $j = 1, 2, \ldots n-1$
2:  **repeat**
3:      epoch ← epoch+1
4:      **repeat**
5:          $b \leftarrow b+1$, where $b$ is the $b$th batch of input data
6:          **repeat**
7:              $k \leftarrow k+1$, where $k$ is the $k$th input data
8:              Update each layer's input weight $\hat{\mathbf{W}}_j(k)$ concurrently with fixed output weight matrix $\bar{\mathbf{W}}_j^{\triangleright[b-1]}$ based on Equation (22)
9:              Update each layer's output weight $\hat{\mathbf{W}}_j^{\triangleright}(k)$ concurrently with fixed input weight matrix $\bar{\mathbf{W}}_j^{\triangleright[b-1]}$ based on Equation (23)
10:         **until** $k = l * p, l = 1, 2, \ldots$, where $p$ is the batch size
11:         $\bar{\mathbf{W}}_j^{\triangleright[b]} = \hat{\mathbf{W}}_j^{\triangleright}(lp)$, $\bar{\mathbf{W}}_j^{[b]} = \hat{\mathbf{W}}_j(lp)$
12:     **until** all batches of data have been trained
13: **until** required epochs have been trained
**Output:** Estimated weight matrices $\hat{\mathbf{W}}_j,\ j = 1, 2, \ldots n$
```

function for the output layer are sigmoid activation functions, same as the full network.

During training, the value of a for AdjReLU in (12) was selected according to the value of error in (16) following the strategies in section III-B (paragraph before (36)). In this case study, the variable a is initially set as 1 and then gradually adjusted to 0.01 to introduce nonlinearity in the activation function. The value of a stops at 0.01, which is the default value for standard leaky ReLU activation function, so that the activation function keeps some information when input is negative to avoid dying ReLU problem. During the adjustment process, the value of a is calculated according to $a = \frac{\|e\|-\epsilon}{\|e\|}$, where ϵ is a small constant value set as 0.01. When the error is large at the beginning phase of training, a is close to 1 and when the error reduces, the value a also reduces with a stopping rule set so that it stops when a reaches 0.01 finally.

The deep neural network was trained with both SGD and proposed method. First, the initial learning rate for both methods were set as 0.5 and the same initial weights were used for both methods. For the proposed method, the learning gains were then automatically adjusted according to the conditions in (52) and (53). It can be seen from Fig 3 that SGD diverges after 6 epochs while the proposed method converges.

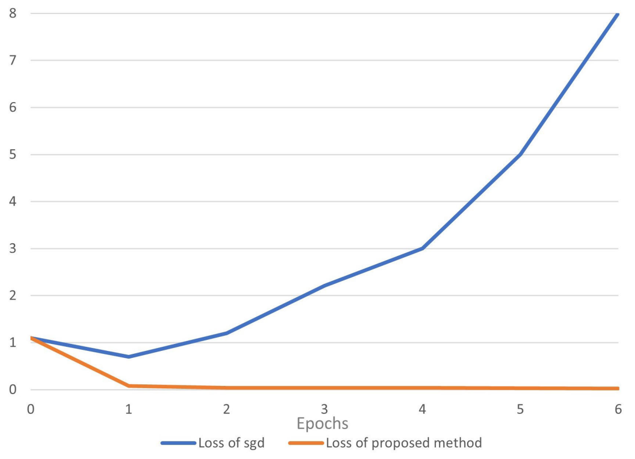

FIGURE 3. Loss of SGD and proposed method in first 6 epochs.

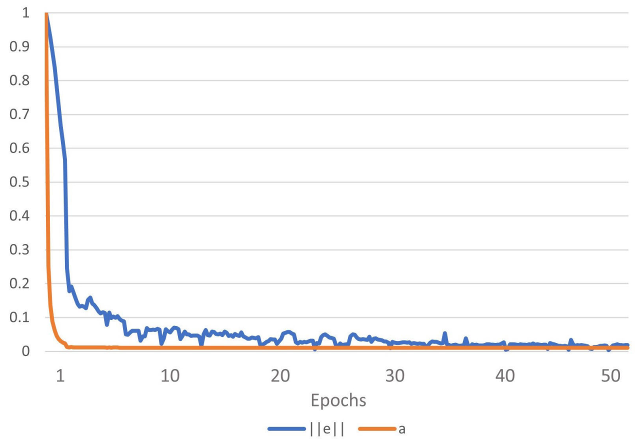

FIGURE 4. Norm of training error $\|e\|$ in (16) of virtual learning system V for MNIST dataset with AdjReLU with gradient of a.

To ensure the covergence of SGD for further comparison, the learning rate selected for SGD was later manually adjusted to 0.05 and was decreased by half after 50 epochs, other hyperparameters were also tuned for the test results of the full network to achieve best performance. The training and testing accuracies of both SGD and proposed method are presented in Table 1. The final test result of SGD is 98.48%. For the proposed method, the learning gains were also set as 0.05 and were then automatically adjusted according to the conditions in (52) and (53). The adjustment of gradident a for adjustable relu is shown in Fig 4 and the batch size was selected as 32 for all virtual networks.

It can be seen from the results in Table 1 that the accuracies of the proposed method is comparable to standard SGD while ensuring convergence during learning.

B. CIFAR10 DATASET

The CIFAR10 dataset contains 10 classes of color images with the size of 32 by 32 pixels. For this dataset, we used a pretrained CNN called VGG11 based on ImageNet [38] dataset and trained its classifier for CIFAR10. VGG11, a deep

TABLE 1. Training & test accuracies (%) by SGD and proposed method for MNIST dataset.

SGD		Forward Simultaneous E2E Learning	
Training accuracy/%	Testing accuracy/%	Training accuracy/%	Testing accuracy/%
99.9	98.48	99.87	**98.64**

Note: The value of a is for AdjReLU defined in (12) of virtual learning system V.

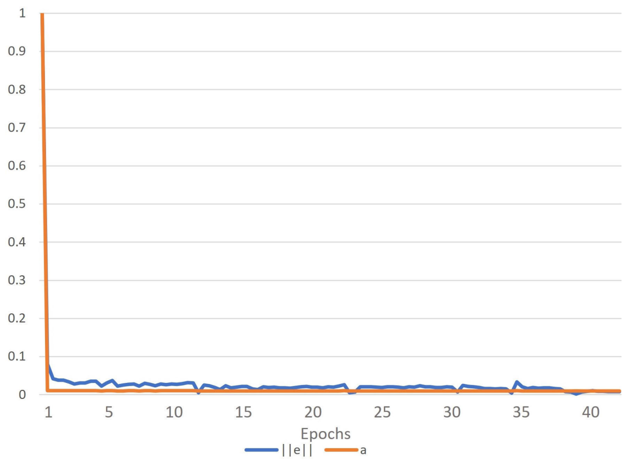

FIGURE 5. Norm of training error $\|e\|$ in (16) of virtual learning system II for CIFAR10 dataset with gradient of a.

CNN, contains eight convolutional layers with kernel size of three and a classifier with three fully connected layers. The full classifier has the structure of two hidden layers each has 512 neurons and a output layer as 512-512-LReLU-512-LReLU-10-sigmoid. The full classifier used standard Leaky ReLU activation functions with slope of 0.01 when input is smaller than zero at hidden layers and sigmoid activation functions for the output layer.

The full classifier was trained with two virtual learning systems which were running simultaneously: 512-512-AjReLU-10-sigmoid (virtual learning system I) and 512-512-AjReLU-10-sigmoid (virtual learning system II). Each virtual learning system updated the weights following update laws in (22) using AdjReLU and (23) using standard Leaky ReLU at hidden layers. The activation function for the output layer were sigmoid activation functions, same as the full classifier. The value of a was adjusted following the same strategies as in the previous case study as shown in Fig 5, and the test result for the full classifier of SGD was tuned to the best accuracy. The learning rate selected for SGD was 0.001 and it finally achieved a test accuracy of 88.4%. For the proposed method, the initial learning gains were set as: $\boldsymbol{L_1} = \boldsymbol{L_2} = \boldsymbol{diag}\ [0.001, \ldots 0.001]$ and were adjusted according to the conditions in (52) and (53) during training. The batch size was selected as 32 for all virtual learning systems.

TABLE 2. Training & test accuracies (%) by SGD and proposed method for CIFAR10 dataset.

SGD		Forward Simultaneous E2E Learning	
Training accuracy/%	Testing accuracy/%	Training accuracy/%	Testing accuracy/%
94.9	88.4	95	**88.55**

Note: The value of a is for AdjReLU defined in (12) of virtual learning system V.

It can be seen from results in Table 2 that, for the classifier of a CNN, the final results of the proposed method achieve similar results compared to standard SGD while ensuring convergence during learning.

C. ONLINE KINEMATIC CONTROL EXPERIMENT ON UR5E

The Jacobian matrix is essential to perform task space control of a robot [39]. When the Jacobian matrix is unknown, it is difficult to perform real-time control in task space [40] as the Jacobian matrix provides directional feedback information in task-space. In this subsection, we show that DNNs can be trained to approximate the unknown Jacobian matrix using our proposed learning algorithm.

Consider the system at the kth sampling time, the velocities of the end effector in task space $\dot{\boldsymbol{x}}$ is related to the rate of joint velocities $\dot{\boldsymbol{q}}$ as:

$$\dot{\boldsymbol{x}}(k) = \mathbf{J}(\boldsymbol{q}(k))\dot{\boldsymbol{q}}(k) \tag{57}$$

where $\mathbf{J}(\boldsymbol{q}(k))$ is the overall Jacobian matrix from joint space to sensory task space.

For a DNN, apply the idea of weight designation in [33], equation (57) can be written as following (13) and (15):

$$\dot{\boldsymbol{x}}(k) = \sum_{h=1}^{p} \sigma_h\big(\bar{\mathbf{W}}_j^{\triangleright[b-1]}(k)\mathbf{R}_j^a(k)\mathbf{W}_j^{[b]}(k)\boldsymbol{q}_j(k)\big)\dot{q}_h(k) + \boldsymbol{\epsilon}_j^{\triangleright}(k) \tag{58}$$

$$\dot{\boldsymbol{x}}(k) = \sum_{h=1}^{p} \sigma_h\big(\mathbf{W}_j^{\triangleright[b]}(k)\mathbf{R}_j(k)\bar{\mathbf{W}}_j^{[b-1]}(k)\boldsymbol{q}_j(k)\big)\dot{q}_h(k) + \boldsymbol{\epsilon}_j(k) \tag{59}$$

where $j = 1, \ldots n-1$ denote the jth learning systems, p denotes the number of joints, $\dot{q}_h(k)$ is the hth element in $\dot{\boldsymbol{q}}(k)$ and the activation function $\boldsymbol{\sigma}$ is selected as identity activation function for control task and $\boldsymbol{q}_j$ is the input for the jth learning system. The estimated Jacobian matrix $\hat{\mathbf{J}}_j$ can be obtained from the output of the network following (11) and (14):

$$\begin{aligned}\hat{\dot{\boldsymbol{x}}}_j^{\triangleright}(k) &= \hat{\mathbf{J}}_j^{\triangleright}(\boldsymbol{q}(k), \bar{\mathbf{W}}_j^{\triangleright[b-1]}, \hat{\mathbf{W}}_j)\dot{\boldsymbol{q}}(k) \\ &= \sum_{h=1}^{p} \sigma_h\big(\bar{\mathbf{W}}_j^{[b-1]}(k)\hat{\mathbf{R}}_j^a(k)\hat{\mathbf{W}}_j(k)\boldsymbol{q}_j(k)\big)\dot{q}_h(k) \qquad (60)\\ \hat{\dot{\boldsymbol{x}}}_j(k) &= \hat{\mathbf{J}}_j(\boldsymbol{q}(k), \hat{\mathbf{W}}_j^{\triangleright}, \bar{\mathbf{W}}_j^{[b-1]})\dot{\boldsymbol{q}}(k)\end{aligned}$$

$$= \sum_{h=1}^{p} \sigma_h\big(\hat{\mathbf{W}}_j^{\triangleright}(k)\hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}(k)\boldsymbol{q}_j(k)\big)\dot{q}_h(k) \quad (61)$$

For learning systems $j = 1, \ldots n-1$, the weights are updated based on output error $\boldsymbol{e}_j^{\triangleright}(k)$ formulated using (58) (60) and $\boldsymbol{e}_j(k)$ formulated using (59) (61):

$$\begin{aligned}
\boldsymbol{e}_j^{\triangleright}(k) &= \dot{\boldsymbol{x}}(k) - \hat{\dot{\boldsymbol{x}}}_j^{\triangleright}(k)\\
&= \sum_{h=1}^{p} \sigma_h\big(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}(k)\mathbf{R}_j^{a}(k)\mathbf{W}_j^{[b]}(k)\boldsymbol{q}_j(k)\big)\dot{q}_h(k)\\
&\quad + \boldsymbol{\epsilon}_j^{\triangleright}(k)\\
&\quad - \sum_{h=1}^{p} \sigma_h\big(\bar{\mathbf{W}}_j^{\triangleright^{[b-1]}}(k)\hat{\mathbf{R}}_j^{a}(k)\hat{\mathbf{W}}_j(k)\boldsymbol{q}_j(k)\big)\dot{q}_h(k) \quad (62)\\
\boldsymbol{e}_j(k) &= \dot{\boldsymbol{x}}(k) - \hat{\dot{\boldsymbol{x}}}_j(k)\\
&= \sum_{h=1}^{p} \sigma_h\big(\mathbf{W}_j^{\triangleright^{[b]}}(k)\hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}(k)\boldsymbol{q}_j(k)\big)\dot{q}_h(k)\\
&\quad + \boldsymbol{\epsilon}_j(k)\\
&\quad - \sum_{h=1}^{p} \sigma\big(\hat{\mathbf{W}}_j^{\triangleright}(k)\hat{\mathbf{R}}_j(k)\bar{\mathbf{W}}_j^{[b-1]}(k)\boldsymbol{q}_j(k)\big)\dot{q}_h(k) \quad (63)
\end{aligned}$$

Therefore the error $\boldsymbol{e}_j^{\triangleright}(k)$ and $\boldsymbol{e}_j(k)$ is the same as the output error in (16) and (20). The errors $\boldsymbol{e}_j^{\triangleright}(k), j = 1, \ldots n-1$ and $\boldsymbol{e}_j(k), j = 1, \ldots n-2$ are then applied in the update laws (24) and (25) to ensure convergence during training the network.

Note that $\mathbf{J}_j^{\triangleright}, j = 1, \ldots n-1$, are formulated for the update law and are not used during kinematic control. Only the jocabian matrix $\hat{\mathbf{J}}_{n-1}(\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})$ of the $n-1$th learning system is used for control task and rest are only for updating process. The control task is implemented on the $n-1$th vritual system. The reference joint velocity $\dot{\boldsymbol{q}}$ is proposed as:

$$\begin{aligned}
\dot{\boldsymbol{q}}(k) &= \hat{\mathbf{J}}_{n-1}^{\dagger}(\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})(\dot{\boldsymbol{x}}_d(k) - \alpha(\boldsymbol{x}(k) - \boldsymbol{x}_d(k)))\\
&= \hat{\mathbf{J}}_{n-1}^{\dagger}(\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})(\dot{\boldsymbol{x}}_d(k) - \alpha\Delta\boldsymbol{x}(k)) \quad (64)
\end{aligned}$$

where $\hat{\mathbf{J}}_{n-1}^{\dagger}(\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})$ denotes the pseudoinverse matrix of the estimated Jacobian $\hat{\mathbf{J}}_{n-1}(\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})$; α is a positive scalar; $\boldsymbol{x}_d(k)$ denotes the desired position and $\dot{\boldsymbol{x}}_d(k)$ represents the desired velocity of the end effector in the sensory task space. Multiply (64) with $\hat{\mathbf{J}}_{n-1}(\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})$ on the left hand side gives

$$\hat{\mathbf{J}}_{n-1}((\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})\dot{\boldsymbol{q}}(k) = \dot{\boldsymbol{x}}_d(k) - \alpha\Delta\boldsymbol{x}(k) \quad (65)$$

Subtracting (57) and (65) gives

$$\begin{aligned}
&\mathbf{J}(\boldsymbol{q}(k))\dot{\boldsymbol{q}}(k) - \hat{\mathbf{J}}_{n-1}(\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})\dot{\boldsymbol{q}}(k)\\
&\quad = \dot{\boldsymbol{x}}(k) - \dot{\boldsymbol{x}}_d(k) + \alpha\Delta\boldsymbol{x}(k) \quad (66)\\
&\quad = \Delta\dot{\boldsymbol{x}}(k) + \alpha\Delta\boldsymbol{x}(k)
\end{aligned}$$

Let $\boldsymbol{\varepsilon}(k) = \Delta\dot{\boldsymbol{x}}(k) + \alpha\Delta\boldsymbol{x}(k)$ be the online feedback error in online learning, from (57)(59)(61) and (63) (63) we have:

$$\begin{aligned}
\boldsymbol{\varepsilon}(k) &= \mathbf{J}(\boldsymbol{q}(k))\dot{\boldsymbol{q}}(k) - \hat{\mathbf{J}}_{n-1}(\boldsymbol{q}(k), \hat{\mathbf{W}}_n, \bar{\mathbf{W}}_{n-1}^{[b-1]})\dot{\boldsymbol{q}}(k)\\
&= \sum_{h=1}^{p} \sigma_h\big(\mathbf{W}_n(k)\hat{\mathbf{R}}_{n-1}(k)\bar{\mathbf{W}}_{n-1}^{[b-1]}(k)\boldsymbol{q}_{n-1}(k)\big)\dot{q}_h(k)\\
&\quad + \boldsymbol{\epsilon}_{n-1}(k)\\
&\quad - \sum_{h=1}^{p} \sigma_h\big(\hat{\mathbf{W}}_n(k)\hat{\mathbf{R}}_{n-1}(k)\bar{\mathbf{W}}_{n-1}^{[b-1]}(k)\\
&\quad \times \boldsymbol{q}_{n-1}(k)\big)\dot{q}_h(k) \quad (67)
\end{aligned}$$

Hence, we can use the online feedback error $\boldsymbol{\varepsilon}(k)$ instead of $\boldsymbol{e}_{n-1}(k)$ in (63) in the $n-1$th learning system. The online feedback error $\boldsymbol{\varepsilon}(k)$ is used in update law (25) and the convergence of $\boldsymbol{\varepsilon}(k)$ is therefore guaranteed.

In the online task, the robot performed an online trajectory tracking control task in 3D space. The input $[\boldsymbol{q}, \dot{\boldsymbol{q}}]$ and output $\dot{\boldsymbol{x}}$ can be collected from the robot communication with sampling time of 0.01s.

The full network used for approximating the jacobian matrix is a three-hidden layer FNN with the structure of 3-12-LReLU-12-LReLU-12-LReLU-3-identity. The input and output size of the network were both 3 and each hidden layer had 12 neurons.The activation functions used between hidden layers were standard Leaky ReLU activation functions, and the output activation functions were identity activation functions. The full network was trained based on two virtual learning systems which were running concurrently: 3-12-AdjReLU-3-identity (virtual learning system I), 12-12-AdjReLU-3-identity (virtual learning system II), 12-12-AdjReLU-3-identity (virtual learning system III). Each virtual learning system updated the weights following update laws in (22) using AdjReLU and (23) using standard Leaky ReLU at hidden layers. The activation function for the output layer are identity activation functions, same as the full network.

The trajectory tracking task were done on two separate circles $C1$ and $C2$. $C1$ is a circle centered at $[-0.4, -0.35, 0.5]$ with a radius of $0.15m$ and $C2$ is a circle centered at $[0.08, -0.4, 0.5]$ with a radius of $0.1m$. The specified trajectory is: $x_{C1_1} = -0.4 + 0.28\cos(\omega t) - 0.9\sin(\omega t)$; $x_{C1_2} = -0.35 + 0.85\cos(\omega t) + 0.4\sin(\omega t)$; $x_{C1_3} = 0.5 + 0.45\cos(\omega t) + 0.2\sin(\omega t)$; $x_{C2_1} = 0.08 + 0.1\cos(\omega t) - 0.9\sin(\omega t)$; $x_{C2_2} = -0.4 + 0.9\cos(\omega t) + 0.4\sin(\omega t)$; $x_{C2_3} = 0.5 + 0.45\cos(\omega t) + 0.2\sin(\omega t)$; At first 20 seconds, the desired angular frequency was 0 rad/s to do a set point control so that the robot can move to the initial position. The desired angular speed at full speed was $2\pi/T$ s, after accelerating from 0 rad/s in the T s. Then the robot would decelerate from full speed to 0 rad/s in another T s and finally be at rest for the last 20 s.

The two circles were placed on different planes and were far away from each other. The tracking task started at $C1$ first. As the method is model free, to provide a good initialization, data were manually collected near the plane of $C1$ and two

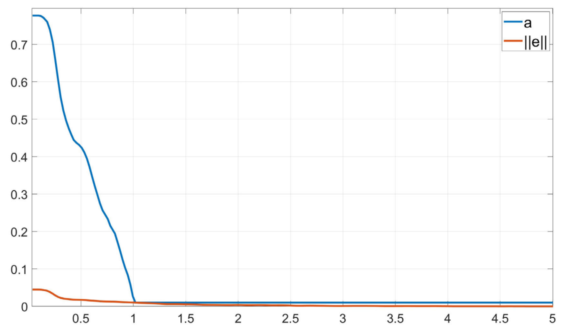

FIGURE 6. The value of a and norm of output error e in (62) of $C1$ of virtual learning system III.

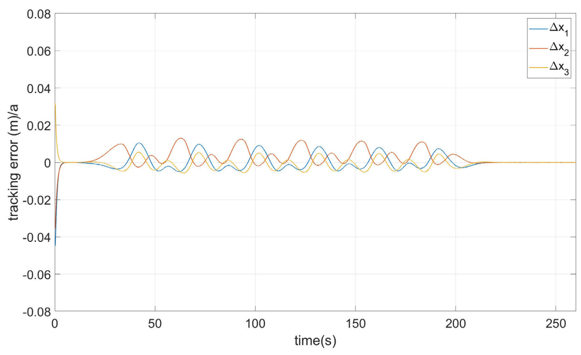

FIGURE 7. The tracking errors in $C1$.

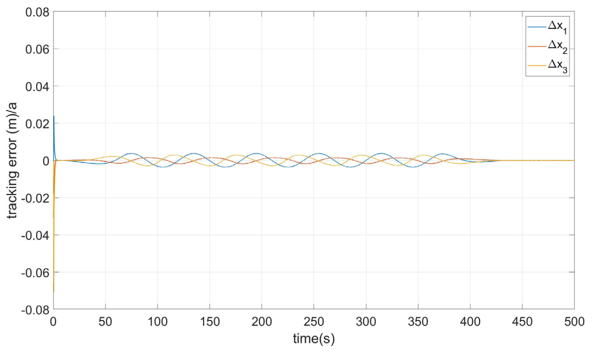

FIGURE 8. The tracking errors in $C2$.

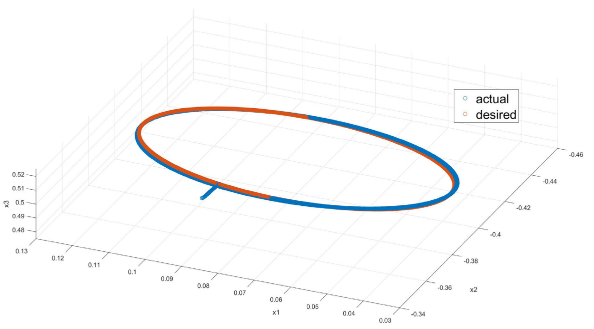

FIGURE 9. Desired and actual trajectories of the robot end-effector in sensory space for training circle $C2$.

virtual learning systems were pretrained using the proposed method for 200 epochs with initial learning rate of 0.01. This area is used as a home or starting position for all control tasks of the robot. Then the trained weights were used for the initialization weight for online tracking of $C1$. As shown in Fig. 6, the value of a for AdjReLU was selected to according to the value of error in (16) and (62) following the strategies in case studies. In this online experiment, the value of a was adjusted continuously. As shown in Fig 7, the online learning guaranteed the convergence of tracking errors for circle $C1$.

After online learning of $C1$, the robot was moved from $C1$ to the initial position near $C2$ by following a straight line path using the proposed controller. From there, the online learning of tracking $C2$ started directly without any offline training. As shown in Fig 8 and Fig 9, during online learning, the tracking errors converged to small value and the actual trajectory was close to the desired one.

From the online trajectory tracking task, our proposed E2E framework guarantees the convergence of the tracking errors during online training. The online learning allows the network to adapt to a new circle $C2$ which is far away from the original task space near $C1$.

V. CONCLUSION

In this paper, we have presented a forward simultaneous E2E framework to train deep fully connected neural networks so that convergence to a bound can be ensured. The classification tasks including MNIST and CIFAR10 datasets implemented on deep FNNs have shown that our E2E learning framework can achieve similar test accuracy as SGD method with guaranteed convergence. The online kinematic control task has also shown that a robot with unknown kinematics can learn with guaranteed convergence in a safe way.

REFERENCES

[1] Y. Bengio, I. Goodfellow, and A. Courville, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2016.
[2] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.
[3] R. M. Sanner and J.-J.-E. Slotine, "Gaussian networks for direct adaptive control," *IEEE Trans. Neural Netw.*, vol. 3, no. 6, pp. 837–863, 1992.
[4] F. L. Lewis, K. Liu, and A. Yesildirek, "Neural net robot controller with guaranteed tracking performance," *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 703–715, May 1995.
[5] X. Li and C. C. Cheah, "Adaptive neural network control of robot based on a unified objective bound," *IEEE Trans. Control Syst. Technol.*, vol. 22, no. 3, pp. 1032–1043, May 2014.
[6] C. Yang, X. Wang, L. Cheng, and H. Ma, "Neural-learning-based telerobot control with guaranteed performance," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3148–3159, Oct. 2017.
[7] S. Lyu and C. C. Cheah, "Data-driven learning for robot control with unknown Jacobian," *Automatica*, vol. 120, Oct. 2020, Art. no. 109120.
[8] C. Liu, G. Wen, Z. Zhao, and R. Sedaghati, "Neural-network-based sliding-mode control of an uncertain robot using dynamic model approximated switching gain," *IEEE Trans. Cybern.*, vol. 51, no. 5, pp. 2339–2346, May 2021.
[9] Y. Liu, D. Yao, H. Li, and R. Lu, "Distributed cooperative compound tracking control for a platoon of vehicles with adaptive NN," *IEEE Trans. Cybern.*, vol. 52, no. 7, pp. 7039–7048, Jul. 2022.
[10] Y. Hu, H. Yan, H. Zhang, M. Wang, and L. Zeng, "Robust adaptive fixed-time sliding-mode control for uncertain robotic systems with input saturation," *IEEE Trans. Cybern.*, early access, Apr. 20, 2022, doi: 10.1109/TCYB.2022.3164739.

[11] F. L. Lewis, A. Yesildirek, and K. Liu, "Multilayer neural-net robot controller with guaranteed tracking performance," *IEEE Trans. Neural Netw.*, vol. 7, no. 2, pp. 388–399, Mar. 1996.
[12] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks," *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 589–600, Jul. 1998.
[13] L. Cheng, Z.-G. Hou, and M. Tan, "Adaptive neural network tracking control for manipulators with uncertain kinematics, dynamics and actuator model," *Automatica*, vol. 45, no. 10, pp. 2312–2318, 2009.
[14] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015, pp. 1–9.
[15] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke, "The limits and potentials of deep learning for robotics," *Int. J. Robot. Res.*, vol. 37, nos. 4–5, pp. 405–420, 2018.
[16] R. Sun, M. L. Greene, D. M. Le, Z. I. Bell, G. Chowdhary, and W. E. Dixon, "Lyapunov-based real-time and iterative adjustment of deep neural networks," *IEEE Control Syst. Lett.*, vol. 6, pp. 193–198, 2022.
[17] D. M. Le, M. L. Greene, W. A. Makumi, and W. E. Dixon, "Real-time modular deep neural network-based adaptive control of nonlinear systems," *IEEE Control Syst. Lett.*, vol. 6, pp. 476–481, 2022.
[18] O. S. Patil, D. M. Le, M. L. Greene, and W. E. Dixon, "Lyapunov-derived control and adaptive update laws for inner and outer layer weights of a deep neural network," *IEEE Control Syst. Lett.*, vol. 6, pp. 1855–1860, 2022.
[19] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G.-Z. Yang, "XAI—Explainable artificial intelligence," *Sci. Robot.*, vol. 4, no. 37, 2019, Art. no. eaay7120.
[20] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 242–252.
[21] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1675–1685.
[22] D. Zou, Y. Cao, D. Zhou, and Q. Gu, "Gradient descent optimizes over-parameterized deep ReLU networks," *Mach. Learn.*, vol. 109, pp. 467–492, Oct. 2020.
[23] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 19, 2006, pp. 1–8.
[24] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why does unsupervised pre-training help deep learning?" in *Proc. Int. Conf. Artif. Intell. Statist.*, 2010, pp. 201–208.
[25] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Greedy layerwise learning can scale to ImageNet," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 583–593.
[26] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1–12.
[27] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers Neurosci.*, vol. 12, p. 608, Aug. 2018.
[28] Y. Wang, Z. Ni, S. Song, L. Yang, and G. Huang, "Revisiting locally supervised learning: An alternative to end-to-end training," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–21.
[29] M. Zeng, Y. Liao, R. Li, and A. Sudjianto, "Local linear approximation algorithm for neural network," *Mathematics*, vol. 10, no. 3, p. 494, Feb. 2022.
[30] Y. Shin, "Effects of depth, width, and initialization: A convergence analysis of layer-wise training for deep linear neural networks," *Anal. Appl.*, vol. 20, no. 1, pp. 73–119, Jan. 2022.
[31] H.-T. Nguyen, C. C. Cheah, and K.-A. Toh, "An analytic layer-wise deep learning framework with applications to robotics," *Automatica*, vol. 135, Jan. 2022, Art. no. 110007.
[32] H.-T. Nguyen, S. Li, and C. C. Cheah, "A layer-wise theoretical framework for deep learning of convolutional neural networks," *IEEE Access*, vol. 10, pp. 14270–14287, 2022.
[33] H.-T. Nguyen and C. C. Cheah, "Analytic deep neural network-based robot control," *IEEE/ASME Trans. Mechatronics*, vol. 27, no. 4, pp. 2176–2184, Aug. 2022.
[34] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.
[35] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," Toronto, ON, Canada, Tech. Rep., 2009.
[36] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2011, pp. 315–323.
[37] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–13.
[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5206848/
[39] C. C. Cheah and X. Li, *Task-Space Sensory Feedback Control of Robot Manipulators*. Cham, Switzerland: Springer, 2015.
[40] C. C. Cheah, C. Liu, and J. J. E. Slotine, "Adaptive tracking control for robots with unknown kinematic and dynamic properties," *Int. J. Robot. Res.*, vol. 25, pp. 283–296, Mar. 2006.

SITAN LI received the B.S. degree in automation from Xi'an Jiaotong University and the M.Sc. degree in computer control and automation from Nanyang Technological University, where she is currently pursuing the Ph.D. degree with the School of Electrical and Electronic Engineering. Her research interests include deep learning and robotic control.

HUU-THIET NGUYEN received the engineering degree in control and automation engineering from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2015, and the Ph.D. degree in electrical and electronic engineering from Nanyang Technological University, Singapore, in 2022. His research interests include robot control, robot learning, and machine learning in robotics and physical systems.

CHIEN CHERN CHEAH (Senior Member, IEEE) was born in Singapore. He received the B.Eng. degree in electrical engineering from the National University of Singapore, in 1990, and the M.Eng. and Ph.D. degrees in electrical engineering from Nanyang Technological University, Singapore, in 1993 and 1996, respectively. From 1990 to 1991, he was a Design Engineer with Chartered Electronics Industries, Singapore. He was a Research Fellow with the Department of Robotics, Ritsumeikan University, Japan, from 1996 to 1998. He is currently an Associate Professor with Nanyang Technological University. He served as an Associate Editor for IEEE Transactions on Robotics, from 2010 to 2013, the Program Co-Chair for the IEEE International Conference on Robotics and Automation, in 2017, the Award Chair for the IEEE/RSJ International Conference on Intelligent Robots and Systems, in 2019, and the Lead Guest Editor for IEEE Transactions on Mechatronics, in 2021. He serves as an Associate Editor for *Automatica*.