Bi-PIL: Bidirectional Gradient-Free Learning Scheme for Multilayer Neural Networks

Ke Wang, Binghong Liu, Pandi Liu, Yungao Shi, Ping Guo, *Senior Member, IEEE*, Yafei Li, *Member, IEEE*, and Mingliang Xu

***Abstract*—Training deep neural networks typically relies on gradient descent learning schemes, which is usually time-consuming, and the design of complex network architectures is often intractable. In this article, we explore the building of multilayer neural networks based on an efficient gradient-free learning scheme offering a potential solution to the architectural design. The proposed learning scheme encompasses both forward and backward training (BT) processes. In the forward process, the pseudoinverse learning (PIL) algorithm is employed to train a multilayer neural network, in which the network is dynamically constructed leveraging a layer-by-layer greedy strategy, enabling the automatic determination of the architecture across different hierarchies in a data-driven manner. The network architecture and connection weights determined in the forward training (FT) process are shared with the backward process which also conducts gradient-free learning to update the connection weights. After the bidirectional learning, a neural network comprising two twin subnetworks is obtained, and the fused features of subnetworks are used as inputs for downstream tasks. Comprehensive experiments and detailed analyses demonstrate the effectiveness and superiority of the proposed learning scheme.**

***Index Terms*—Autoencoder (AE), deep learning, multilayer neural networks, nongradient descent learning, pseudoinverse learning (PIL).**

I. Introduction

THE process of training multilayer neural networks can be generally formulated as an optimization problem, in which a certain learning algorithm is employed to minimize a loss function defined according to the learning task, e.g., representation learning, classification, regression, data reconstruction. Currently, the most commonly used learning algorithms are variants of error backpropagation based on gradient descent processes, such as the batch gradient descent, stochastic gradient descent (SGD), adaptive gradient (AdaGrad) algorithm, root mean square propagation (RMSProp), adaptive moment (Adam) estimation, and Nesterov-accelerated Adam (Nadam) estimation [1]. Although these algorithms have been successfully applied, there are still some difficulties, especially when the networks are deeper. First, training complex neural networks using gradient descent learning algorithms is often computationally intensive. The gradient descent learning requires multiple iterations of the optimization process over a dataset (usually on a large scale) to determine the connection weights. The sophisticated architecture of deep neural networks, encompassing huge amounts of parameters, inevitably necessitates extensive computational resources and lengthy training durations. Second, the convergence and performance of these learning algorithms are critically contingent upon the tuning of hyperparameters, including both learning control hyperparameters (e.g., maximum epoch, learning rate, and momentum that are associated with the gradient descent process) and network architecture hyperparameters (e.g., network depth, width, and interlayer connection). Nevertheless, the setting of these hyperparameters is typically approached through a trial-and-error way, relying heavily on empirical tricks [2], [3], [4]. In addition, this trial-and-error way unfortunately exacerbates the training efficiency issue. Third, gradient descent learning algorithms are vulnerable to the issues of gradient vanishing or saturation, particularly when dealing with deep models. Although various techniques such as batch normalization, residual connections, and adopting proper activation functions have been employed to address these issues, gradient vanishing or saturation problems arise due to the inherent multiplicative effect of gradients during backpropagation through multilayer neural networks.

In this case, some gradient-free learning algorithms have begun to attract the attention of researchers as a way to address the inherent issues of gradient descent algorithms. The representative algorithms include random vector functional-link neural networks (RVFLNNs) [5], pseudoinverse learning (PIL) [6], extreme learning machine (ELM) [7], broad learning system (BLS) [8], analytic network (ANnet) learning [9], kernel-and-range space network (KARnet) [10], and correlation projection network (CPNet) [11]. These algorithms exhibit significant advantages in terms of efficiency and hyperparameter tuning, and they do not suffer from issues related to gradient vanishing or gradient saturation.

Received 23 April 2024; revised 17 January 2025; accepted 23 April 2025. Date of publication 15 May 2025; date of current version 4 September 2025. This work was supported in part by NSFC under Grant 62325602, Grant 62036010, and Grant 62372416; in part by China Postdoctoral Science Foundation under Grant 2020M682348; and in part by the Natural Science Foundation of Henan Province under Grant 242300421215. *(Corresponding authors: Mingliang Xu; Yafei Li.)*

Ke Wang, Binghong Liu, Pandi Liu, Yungao Shi, Yafei Li, and Mingliang Xu are with the School of Computer Science and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China (e-mail: iekwang@zzu.edu.cn; bhliu@gs.zzu.edu.cn; pdliu@gs.zzu.edu.cn; shiyungao0718@gs.zzu.edu.cn; ieyfli@zzu.edu.cn; iexumingliang@zzu.edu.cn).

Ping Guo is with the School of System Science, Beijing Normal University, Beijing 100875, China, and also with the International Academic Center of Complex Systems, Beijing Normal University, Zhuhai 519087, China (e-mail: pguo@bnu.edu.cn).

Digital Object Identifier 10.1109/TNNLS.2025.3564654

However, the existing gradient-free learning algorithms also present several common issues. First, constructing multilayer neural networks based on gradient-free learning algorithms typically involves a greedy layer-wise training scheme [12], [13]. This involves training multiple building blocks and then stacking them into a multilayer network. Unlike gradient descent learning, gradient-free learning algorithms lack effective global fine-tuning techniques for multilayer architecture. Furthermore, within the greedy layer-wise training scheme, building blocks are generally trained as autoencoders (AEs) in an unsupervised manner. Consequently, the resulting models tend to overfit to the reconstruction task, failing to fully utilize the labels in the dataset. Second, the architecture of multilayer neural networks is typically designed prior to training. This often leads to unnecessary complexity in the model, which can be supported by neural network compression techniques [14], [15].

In this article, we propose a gradient-free learning scheme for multilayer neural networks. The proposed learning scheme includes a bidirectional training process. In the forward training (FT) process, the PIL algorithm is employed to train a network with AEs as building blocks. Notably, the proposed approach can dynamically add building blocks in a greedy layer-wise manner, incorporating early stop criteria to terminate the FT process automatically. The backward training (BT) process shares the same network architecture as the FT and also employs a gradient-free learning algorithm to obtain closed-form solutions. Following this bidirectional training, a neural network consisting of two twin subnetworks is derived. The features extracted from subnetworks are then fused and utilized as inputs for subsequent tasks.

II. Related Works

A. Multilayer Perceptron

The multilayer perceptron (MLP) which is also known as the multilayer neural network is a classical type of feedforward neural network. It consists of multiple connected layers. Each layer performs a linear transformation followed by an activation function which is usually nonlinear and piecewise continuous. Recent research in the field of deep learning has shown that MLP can serve as the foundation for many advanced network structures such as Transformer [16], MLP-Mixer [17], and ResMLP [18].

Given a training set, $D = \{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^N$, which consists of N samples in total. $\mathbf{x}^i = (x_1, x_2, \ldots, x_d)^{\mathrm{T}} \in \mathbf{R}^d$ denotes the ith d-dimensional training sample and $\mathbf{y}^i = (y_1, y_2, \ldots, y_m)^{\mathrm{T}} \in \mathbf{R}^m$ is the corresponding label vector. The loss function of an MLP can be generally defined as follows:

$$J(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \left\| f(\mathbf{x}^i, \theta) - \mathbf{y}^i \right\|_2 \tag{1}$$

where $\theta = \{\mathbf{W}, \mathbf{b}\}$ represents the connection weights and the bias (without loss of generality, the bias could be omitted for concision) which need to be learned. A mapping function $f(\cdot)$ is used to model the MLP in (1). To improve the generalization, a regularization term could be added as follows:

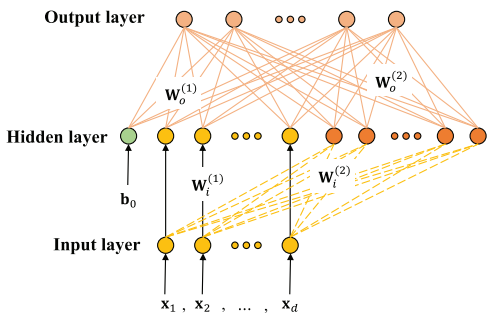


Fig. 1. Basic architecture of the second-order neural network.

$$J(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \left\| f(\mathbf{x}^i, \theta) - \mathbf{y}^i \right\|_2 + \frac{\lambda}{2} \sum_{j=1}^{M} w_j^2. \tag{2}$$

The second term in (2) is an l_2 regularization term (also called the weight decay term). The hyperparameter λ is the regularization coefficient. M is the count of weights and w_j denotes an arbitrary connection between two neurons lying in adjacent layers.

B. Autoencoder

AE is a particular type of MLP that is usually used as the building block for deep learning models within a greedy layer-wise manner [12]. It typically comprises an input layer, a hidden layer, and an output layer, with an encoder and a decoder as its components. It could be trained in an unsupervised manner with the aim of learning representation by reconstructing the original data based on the features in the hidden layer.

The encoder first maps the input data matrix $\mathbf{X}$ into a hidden space as follows:

$$\mathbf{Z} = \sigma(\mathbf{W}_e \mathbf{X}). \tag{3}$$

Conversely, the decoder reconstructs $\mathbf{X}$ from the hidden feature $\mathbf{Z}$ as follows:

$$\hat{\mathbf{X}} = \sigma'(\mathbf{W}_d \mathbf{Z}). \tag{4}$$

In aforementioned (3) and (4), $\mathbf{W}_e$ and $\mathbf{W}_d$ represent the weight matrices, while $\sigma(\cdot)$ and $\sigma'(\cdot)$ represent different activation functions. In common practice, a linear function or an identity function is usually used for $\sigma'(\cdot)$. Since AE can be viewed as a particular MLP, the weights matrices could be determined by minimizing a loss function derived from (1) as follows:

$$J = \frac{1}{2N} \sum_{i=1}^{N} \left\| \mathbf{W}_d(\sigma(\mathbf{W}_e \mathbf{x}^i)) - \mathbf{x}^i \right\|_2. \tag{5}$$

To reduce the count of parameters, the weight matrix $\mathbf{W}_d$ could be set as $\mathbf{W}_d = \mathbf{W}_e^{\mathrm{T}}$, which is known as tied weights [19].

An AE can be used to learn a compressed representation of the input data, particularly when the count of hidden neurons is deliberately configured to be fewer than the count of input neurons. This can also be seen as a dimension-reduction

TABLE I
DIFFERENT IMPLEMENTATIONS OF SOME TYPICAL NONGRADIENT DESCENT LEARNING ALGORITHMS

Methods	$\mathbf{W}_i^{(1)}$	$\mathbf{W}_i^{(2)}$	$\mathbf{H}$	$\mathbf{O}$
RVFLNN	Identity matrix $\mathbf{I}_{d\times d}$	Randomly initialized matrix	$\left[\mathbf{X}\middle\vert\sigma\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)\right]$	$\left[\mathbf{W}_o^{(1)}\middle\vert\mathbf{W}_o^{(2)}\right]\left[\mathbf{X}\middle\vert\sigma\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)\right]$
ELM	0	Randomly initialized matrix	$\sigma\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)$	$\mathbf{W}_o^{(2)}\sigma\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)$
BLS	A set of random matrices $\mathbf{W}_{i1}^{(1)}$, $\mathbf{W}_{i2}^{(1)}$, $\cdots$, $\mathbf{W}_{in}^{(1)}$	A set of random matrices $\mathbf{W}_{i1}^{(2)}$, $\mathbf{W}_{i2}^{(2)}$, $\cdots$, $\mathbf{W}_{in}^{(2)}$	$[\mathbf{Z}_1\cdots\vert\mathbf{Z}_n]\vert\zeta\left(\mathbf{W}_{i1}^{(2)}[\mathbf{Z}_1\cdots\vert\mathbf{Z}_n]\right)\cdots\vert\zeta\left(\mathbf{W}_{in}^{(2)}[\mathbf{Z}_1\cdots\vert\mathbf{Z}_n]\right)$, in which $\mathbf{Z}_j=\sigma\left(\mathbf{W}_{ij}^{(1)}\mathbf{X}\right)$	$\left[\mathbf{W}_o^{(1)}\middle\vert\mathbf{W}_o^{(2)}\right][\mathbf{Z}_1\cdots\vert\mathbf{Z}_n]\vert\zeta\left(\mathbf{W}_{i1}^{(2)}[\mathbf{Z}_1\cdots\vert\mathbf{Z}_n]\right)\cdots\vert\zeta\left(\mathbf{W}_{in}^{(2)}[\mathbf{Z}_1\cdots\vert\mathbf{Z}_n]\right)$
PIL	0	Pseudoinverse of the input matrix, $\mathbf{X}^+$	$\sigma\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)$	$\mathbf{W}_o^{(2)}\sigma\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)$
kernel PIL	0	Using a kernel function, e.g., $\mathbf{\Omega}(i,j)=\exp\left(-\frac{\Vert\mathbf{x}^i-\mathbf{x}^j\Vert^2}{2\sigma^2}\right)$ as a substitute for explicit weights	$\sigma(\mathbf{\Omega})$	$\mathbf{W}_o^{(2)}\sigma(\Omega)$
ANnet	0	$\mathbf{S}\mathbf{X}^T$, where $\mathbf{S}$ is a random scaling matrix	$\sigma\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)$	$\mathbf{W}_o^{(2)}\sigma\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)$
KARnet	0	Matrix derived from the label with Moore-Penrose inverse.	$\sigma_1\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)$	$\sigma_2\left(\mathbf{W}_o^{(2)}\left[1,\sigma_1\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)\right]\right)$
CPNet	0	$\mathbf{P}\mathbf{X}^T$, where $\mathbf{P}\sim N(0,1)$	$\sigma_1\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)$	$\sigma_2\left(\mathbf{W}_o^{(2)}\sigma_1\left(\mathbf{W}_i^{(2)}\mathbf{X}\right)\right)$

process. Conversely, if the number of hidden neurons exceeds the input ones, the AE will learn the representation of the data in a space with higher dimensionality.

C. Nongradient Descent Learning Schemes

Since the family of gradient descent learning algorithms inevitably suffers from a series of problems including the low learning efficiency, difficult tuning of hyperparameters, gradient vanish and saturation, nongradient descent learning algorithms have been intensively studied, such as RVFLNN, PIL, ELM, BLS, ANnet, KARnet, and CPNet. Although these nongradient descent learning algorithms are differentiated from each other by the detailed procedures and implementations, they all follow a basic paradigm. First, they all employ a fixed set of weight parameters to project the data into a hidden feature space and these parameters are frozen during training. However, these weight parameters are set in different ways. To be specific, ELM, ANnet, and CPNet use random input weights. RVFLNN and BLS employ a set of enhancement neurons connected to the input data or mapped feature nodes with random weights. PIL uses the pseudoinverse or its low-rank approximation of the input data as the weights [20], [21], [22]. CPNet uses matrices derived by a sequential label projection into the hidden layers as input weights. In the kernel variants of ELM [7], BLS [23], and PIL [24], a kernel, e.g., the radial basis function, is used to conduct the mapping. Second, all these learning algorithms use similar methods, i.e., the Moore–Penrose inverse, to obtain the output weights by analytically solving a linear problem. The difference among these methods may lie in the regularization term. Except for the commonly used weight decay regularization, there are Jacobian regularization [22], manifold regularization [25], and so on. For the sake of brevity, while maintaining generality, we utilize a second-order neural network comprising a single hidden layer as shown in Fig. 1 to formulate typical gradient-free learning methods with more details. The detailed results are summarized in Table I, in which $\mathbf{W}_i^{(1)}$ and $\mathbf{W}_i^{(2)}$ denote the input weights of the first and the second order term, respectively. $\mathbf{H}$ and $\mathbf{O}$ represent the activations in the hidden and output matrix of the network, respectively. "|" denotes the concatenating operation. $\sigma(\cdot)$ and $\zeta(\cdot)$ are nonlinear activation functions. $\mathbf{W}_o^{(1)}$ and $\mathbf{W}_o^{(2)}$ denote the output weights of the first and the second order term, respectively.

In addition to the aforementioned methods, there exists gradient-free heuristic algorithms for training neural networks.

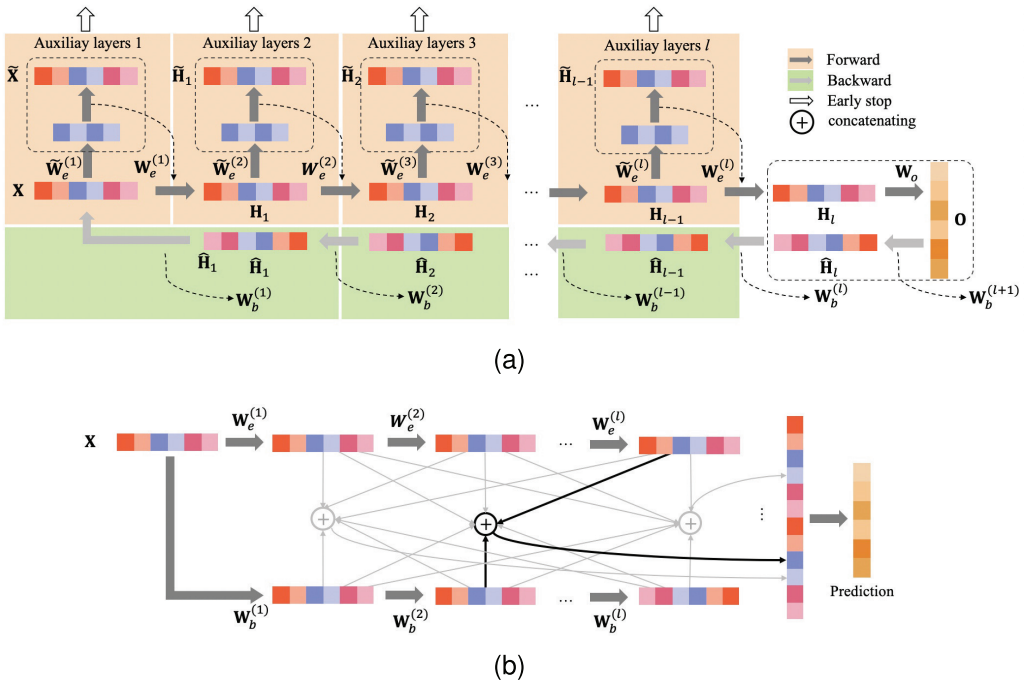


Fig. 2. Architecture and schematic of the proposed model. (a) Training: the training process consists of forward and backward processes, denoted by different colored backgrounds for clarity. During the forward process, a greedy layer-wise training strategy is employed, where multiple building blocks are stacked to form a multilayer network structure. Each building block consists of an encoder and a decoder, trained in an unsupervised manner on data reconstruction tasks. Within each building block, the layers enclosed in dashed-line boxes (decoder) serve as auxiliary layers. Furthermore, each building block incorporates an early stop-exit mechanism, which can dynamically determine the model's depth by halting further growth if no performance enhancement is observed. Upon completion of the forward subnetwork's construction and training, a twin backward subnetwork is concurrently derived. This backward subnetwork mirrors the architecture of the forward subnetwork, with its connection weights initialized from those of the forward subnetwork and subsequently updated during the BT process. (b) Inference: following the completion of the bidirectional training, features are extracted from both paths (subnetworks). These features from diverse paths are then fused to facilitate subsequent predictions.

Evolutionary computation-based methods are one of the representative approaches. These methods initially encode the parameters that need to be learned (i.e., weights and biases). Candidate solutions are then iteratively optimized through the crossover, mutation, and selection operations, with the fittest candidates being retained [26], [27], [28]. Furthermore, some evolutionary computation-based methods can simultaneously optimize both weights and network architectures [29], [30]. Ensemble and hybrid algorithms have also been widely used, such as the combination of evolutionary algorithms and gradient descent [31], [32]. Besides evolutionary computation-based ones, other meta-heuristic algorithms have also been employed for the training of neural networks, such as particle swarm optimization [33], gravitational search algorithm [34], and gray wolf optimizer [35]. Despite their advantage in avoiding trapping into local optima, meta-heuristic algorithms usually suffer from the curse of dimensionality and are inefficient in training complex neural networks [31].

III. Proposed Methodology

The architecture and schematic of the proposed model are shown in Fig. 2. It conducts a bidirectional training process, i.e., a forward process and a backward process successively. The layers marked with dashed line boxes are auxiliary layers. In addition, this model has multiple early stop exits which can dynamically determine the depth of the model by terminating the growth if there is no significant performance improvement. Once the training is completed, all auxiliary layers should be removed. In the inference phases that are shown in Fig. 2(b), features are extracted, respectively, in two paths (subnetworks) corresponding to the forward process and backward process. The features from different paths are then fused to conduct the subsequent prediction. As is shown in Fig. 2(a), the proposed model is constructed with different components. Specifically, the PIL-based AE is used as the base component to construct multilayer neural networks. An additional single hidden layer feedforward neural network (SHLFN) is employed to perform the classification or regression task taking the fused features from subnetworks as inputs.

A. Pseudoinverse Learning-Based AE

PIL was initially introduced for the supervised training of SHLFNs. Since AE is essentially a particular type of SHLFN, it is natural to train AEs with PIL. PIL was employed to train AEs for the first time in [36] which proposed the pseudoinverse learning-based AE (PILAE). Similar to original the autoencoder, PILAE tries to learn an approximation of an identity function with PIL to reconstruct inputs and capture their representations in the latent space associated with the hidden layer.

1) Encoder: For the encoder of PILAE, the weights denoted as $\mathbf{W}_e$ in (3) should be initialized to project the input data into a hidden space. To facilitate a clear description, we use $\tilde{\mathbf{W}}_e$ to represent the initial weights of the encoder. There

are different ways to set the weights depending on which variant algorithm is used. Specifically, the original PIL [6], [20] uses the pseudoinverse of the input matrix, $\mathbf{X}^+$, as the initial encoder weights. PIL0 [37] uses a random matrix. Gn-PIL [37] set the weights as the pseudoinverse of inputs added with a perturbation matrix

$$\tilde{\mathbf{W}}_e = \mathbf{X}^+ + \tilde{\mathbf{G}} \tag{6}$$

in which $\tilde{\mathbf{G}}$ is a perturbation matrix drawn from a Gaussian distribution. The low-rank constraint PIL [21], [22] adopts the low-rank approximation of $\mathbf{X}^+$ as the initial weights. To this end, the singular value decomposition of the input matrix is first conducted

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}. \tag{7}$$

Note that the rank of the input matrix can be concurrently determined by enumerating the nonzero singular values present within the diagonal matrix $\boldsymbol{\Sigma}$, which can provide a reference for the design of network architecture [21], [38]. According to (7), the pseudoinverse of matrix $\mathbf{X}$ could be calculated as follows:

$$\mathbf{X}^+ = \mathbf{V}\boldsymbol{\Sigma}^+\mathbf{U}^{\mathrm{T}} \tag{8}$$

where $\boldsymbol{\Sigma}^+$ is a diagonal matrix generated by transposing matrix $\boldsymbol{\Sigma}$ while replacing the nonzero singular values with their reciprocals. Then, the low-rank approximation of $\mathbf{X}^+$ is obtained and used as the initial encoder weights

$$\tilde{\mathbf{W}}_e = \hat{\mathbf{V}}\boldsymbol{\Sigma}^+\mathbf{U}^{\mathrm{T}}. \tag{9}$$

$\hat{\mathbf{V}}$ is a truncation of matrix $\mathbf{V}$, which is composed of the first p rows of $\mathbf{V}$. The hyperparameter p is the number of hidden neurons of PILAE, which can be manually set or determined with an automated data-driven way [21], [22], [38].

2) Decoder: For the decoder of PILAE, the weights denoted as $\mathbf{W}_d$ in (4) can be optimized by minimizing the loss function derived from (5) by adding a regularization term

$$J(\mathbf{W}_d) = \frac{1}{2}\|\mathbf{W}_d\mathbf{H} - \mathbf{X}\|_2 + \frac{\lambda}{2}\|\mathbf{W}_d\|_q \tag{10}$$

where $\mathbf{H} = \sigma(\mathbf{W}_e\mathbf{X})$ denotes the activations in the hidden layer.

If q is set to 2, the regularization term is known as the weight decay regularization. The optimization problem defined in (10) is also known as the ridge regression. To solve it, let the partial derivative of J with respect to $\mathbf{W}_d$ be equal to zero

$$\frac{\partial J}{\partial \mathbf{W}_d} = 0 \Rightarrow \mathbf{W}_d\left(\mathbf{H}\mathbf{H}^T + \lambda\mathbf{I}\right) = \mathbf{X}\mathbf{H}^T. \tag{11}$$

Since $\lambda \neq 0$, $(\mathbf{H}\mathbf{H}^{\mathrm{T}} + \lambda\mathbf{I})$ is always invertible, then the solution of (10) can be calculated as follows:

$$\mathbf{W}_d = \mathbf{X}\mathbf{H}^{\mathrm{T}}(\mathbf{H}\mathbf{H}^{\mathrm{T}} + \lambda\mathbf{I})^{-1}. \tag{12}$$

An alternative way to solve the problem defined in (10) is based on the generalized inverse theory [39], [40]. For $\mathbf{W}_d\mathbf{H} = \mathbf{X}$, where $\mathbf{H}$ is $p \times N$ and $p < N$ in general, it has many solutions, but

$$\mathbf{W}_d = \mathbf{X}\mathbf{H}^{\mathrm{T}}(\mathbf{H}\mathbf{H}^{\mathrm{T}})^{-1} = \mathbf{X}\mathbf{H}^+ \tag{13}$$

is the approximate solution (least-norm solution) which minimizes $\|\mathbf{W}_d\mathbf{H} - \mathbf{X}\|_2$ and the norm $\|\mathbf{W}_d\|_2$ as well. Since Moore–Penrose inverse also has a definition in limit form as follows:

$$\mathbf{H}^+ = \lim_{\lambda \to 0} \mathbf{H}\left(\mathbf{H}\mathbf{H}^{\mathrm{T}} + \lambda\mathbf{I}\right)^{-1} \tag{14}$$

the solution in (13) is actually identical with the one defined in (12) when the value of λ is sufficiently small.

To make our approach self-contained, we present the relative theorems. More details can be found in [40].

Theorem 1: Let $\mathbf{A}\mathbf{x} = \mathbf{b}$, and suppose $\mathbf{b} \in \mathrm{Sp}(\mathbf{A})$. Then,

$$\mathbf{x} = \mathbf{A}^+\mathbf{b} + (\mathbf{I} - \mathbf{A}^+\mathbf{A})\mathbf{z} \tag{15}$$

in which z denotes an arbitrary vector.

Proof: Let $\mathbf{x}_1 = \mathbf{A}^+\mathbf{b}$ and $\mathbf{b} \in \mathrm{Sp}(\mathbf{A})$. Then, $\mathbf{A}\mathbf{x}_1 = \mathbf{b}$. Meanwhile, since $\mathrm{Ker}(\mathbf{A}) = \mathrm{Sp}(\mathbf{I} - \mathbf{A}^+\mathbf{A})$, a solution of $\mathbf{A}\mathbf{x}_0 = \mathbf{0}$ can be derived by $\mathbf{x}_0 = (\mathbf{I} - \mathbf{A}^+\mathbf{A})\mathbf{z}$. Hence, (15) can be obtained by $\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_0$. So (15) satisfies $\mathbf{A}\mathbf{x} = \mathbf{b}$. ■

Theorem 2: The necessary and sufficient condition for $\mathbf{A}^-$ in $\mathbf{x} = \mathbf{A}^-\mathbf{b}$ to minimize $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$ and $\|\mathbf{x}\|_2$ is $\mathbf{A}^- = \mathbf{A}^+$.

Proof: (Sufficient) According to Theorem 1, we know the $\mathbf{x}$ minimizing $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2$ can be expressed as (15). From the aforementioned fact $\mathrm{Ker}(\mathbf{A}) = \mathrm{Sp}(\mathbf{I} - \mathbf{A}^+\mathbf{A})$, and $\mathbf{A}^+$ is also a minimum norm generalized inverse, the fact that $(\mathbf{I} - \mathbf{A}^+\mathbf{A})\mathbf{b}$ and $\mathbf{A}^+\mathbf{b}$ are mutually orthogonal can be derived, then we obtain

$$\|\mathbf{x}\|^2 = \|\mathbf{A}^+\mathbf{b}\|^2 + \|(\mathbf{I} - \mathbf{A}^+\mathbf{A})\mathbf{z}\|^2 \geq \|\mathbf{A}^+\mathbf{b}\|^2 \tag{16}$$

which indicates that $\|\mathbf{A}^+\mathbf{b}\|^2$ minimizes $\|\mathbf{x}\|^2$.

(Necessity) If $\mathbf{A}^+\mathbf{b}$ can minimize the norm $\|\mathbf{x}\|^2$, $\mathbf{A}^+$ that satisfies (16) also satisfies $(\mathbf{A}^+)^{\mathrm{T}}(\mathbf{I} - \mathbf{A}^+\mathbf{A}) = \mathbf{0}$, which is clearly equivalent to

$$(\mathbf{A}^+)^{\mathrm{T}}\mathbf{A}^+\mathbf{A} = (\mathbf{A}^+)^{\mathrm{T}}. \tag{17}$$

Then, by postmultiplying and premultiplying $\mathbf{A}^+$ and $(\mathbf{A}\mathbf{A}^+ - \mathbf{I})^{\mathrm{T}}$ on both sides of (18), respectively, we can obtain $(\mathbf{A}^+\mathbf{A}\mathbf{A}^+ - \mathbf{A}^+)^{\mathrm{T}}(\mathbf{A}^+\mathbf{A}\mathbf{A}^+ - \mathbf{A}^+) = \mathbf{0}$. On the other hand, by premultiplying (18) by $\mathbf{A}^{\mathrm{T}}$ on both sides, we have $(\mathbf{A}^+\mathbf{A})^{\mathrm{T}} = \mathbf{A}^+\mathbf{A}$. The remaining two conditions $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$ and $(\mathbf{A}^{\mathrm{T}}\mathbf{A}^+)^{\mathrm{T}} = \mathbf{A}\mathbf{A}^+$ can be deduced from the fact that $\mathbf{A}^+$ also serves as a least squares generalized inverse. ■

Based on Theorem 2, the following lemma could be easily obtained.

Lemma 1: The approximate solution defined in (13) is the optimal solution when the row rank of $\mathbf{H}$ is full.

Proof: According to Theorem 2, the general solution of $\mathbf{W}_d\mathbf{H} = \mathbf{X}$ associated with a decoder should be of the form

$$\mathbf{W}_d^* = \mathbf{X}\mathbf{H}^+ + \mathbf{Z}(\mathbf{I} - \mathbf{H}\mathbf{H}^+) \tag{18}$$

where $\mathbf{Z}$ is an arbitrary matrix, $\mathbf{H}^+ = \mathbf{H}^{\mathrm{T}}(\mathbf{H}\mathbf{H}^{\mathrm{T}})^{-1}$ denotes the Moore–Penrose inverse of $\mathbf{H}$. Thus, the error of the approximate solution is exclusively determined by $\mathbf{Z}(\mathbf{I} - \mathbf{H}\mathbf{H}^+)$. If $\mathbf{H}$ is full row rank, then $\mathbf{I} - \mathbf{H}\mathbf{H}^+ = 0$■

Therefore, the row rank of matrix $\mathbf{H}$ is expected to be high so as to make $\mathbf{H}\mathbf{H}^+$ close to an identity matrix. In fact, adding perturbation as (6) and generating the encoder weights based on the pseudoinverse as (9) both aim to achieve this. Under ideal settings, PILAE is capable of achieving exact learning, with the training error being dependent solely on the number

of training samples. To demonstrate this, we substitute (13) into (10) and obtain the the training error as follows:

$$E = \frac{1}{2}\|\mathbf{X}\mathbf{H}^{+}\mathbf{H} - \mathbf{X}\|_2 + \frac{\lambda}{2}\left\|\mathbf{X}\mathbf{H}^{+}\right\|_2. \tag{19}$$

If the number of hidden neurons of PILAE equals the training sample count, a sufficiently small coefficient λ can ensure the existence of the pseudoinverse matrix $\mathbf{H}^{+}$. Consequently, the first term of (19), i.e., the reconstruction error, will be zero. In particular, if the matrix $\mathbf{H}$ is an identity matrix (for instance, using the pseudoinverse of the training data as the input weights), then the training error can be calculated as follows:

$$E = \frac{\lambda}{2}\left\|\mathbf{X}\mathbf{H}^{+}\right\|_2 = \frac{\lambda}{2}\left\|\mathbf{X}(1+\lambda)^{-1}\mathbf{I}\right\|_2 = \frac{\lambda}{2(1+\lambda)}\|\mathbf{X}\|_2. \tag{20}$$

When the matrix $\mathbf{X}$ has been normalized in the data reconstruction task or the commonly used one-hot encoding is employed for the labels in classification tasks, $\|\mathbf{X}\|_2 \propto N$. Since λ is sufficiently small, the training error could exhibit a correspondingly small value as follows:

$$E \propto \frac{\lambda}{2(1+\lambda)}N. \tag{21}$$

If we set $q = 1$ in (10), the regularization term becomes the sparse regularization and the optimization problem is also known as the LASSO [41]. Different kinds of methods could be adopted to solve this optimization problem, such as the K-SVD algorithm [42], alternating direction method of multipliers [43], interior point methods [44], and iterative shrinkage-thresholding algorithm [45]. In this work, an efficient variant of the iterative shrinkage-thresholding algorithm [46] is employed to solve the optimal problem defined in (10). To be specific, the following procedures are performed iteratively:

$$\mathbf{W}_d^k = \mathrm{pL}(\mathbf{V}^k) \tag{22}$$

$$t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1+4t_k^2}\right) \tag{23}$$

$$\mathbf{V}^{k+1} = \mathbf{W}_d^k + \left(\frac{t_k - 1}{t}_{k+1}\right)\left(\mathbf{W}_d^k - \mathbf{W}_d^{k-1}\right) \tag{24}$$

where t_1 is initialized to 1. $\mathrm{pL}(\cdot)$ represents the iterative shrinkage operator that is defined as follows:

$$\mathrm{pL}(\mathbf{V}) = \arg\min_{\mathbf{W}}\left\{\|\mathbf{W}\|_1 + \frac{L}{2}\left\|\mathbf{W} - \mathbf{V} - \frac{1}{L}\nabla q(\mathbf{V})\right\|_2\right\} \tag{25}$$

in which L is the Lipschitz constant, $q(\mathbf{V}) = \|\mathbf{V}\mathbf{H} - \mathbf{X}\|_2$. $\nabla q = 2E_{\max}(\mathbf{H}\mathbf{H}^{\mathrm{T}})$, in which $E_{\max}(\cdot)$ calculates the maximum eigenvalue of the given matrix.

Once decoder weights, $\mathbf{W}_d$, are obtained, the final encoder weights, $\mathbf{W}_e$, can be set as $\mathbf{W}_e = \mathbf{W}_d^{\mathrm{T}}$ (known as tied weights) so as to reduce the freedom degree. After training, the decoder is removed while the encoder is used to embed the input data into the hidden feature space via (3).

B. Multilayer Neural Network Based on PILAE

In Section III-A, we present the implementation details of PILAE, i.e., the base component of the proposed model. Recall that our proposed model has a multilayer architecture, hence the PILAE is then used to construct an MLP. It is straightforward to extend the single hidden layer PILAE to an MLP with a greedy layer-wise strategy. Specifically, each PILAE is trained individually and the generated hidden features of the current PILAE are used as the inputs to the subsequent one. To facilitate a clear description, we use $\mathbf{W}_e^{(l)}$ and $\mathbf{H}_l$ to denote the encoder weights and the hidden features of the lth PILAE. The extracted features from an MLP with l hidden layers should be calculated as follows:

$$f(\mathbf{x}) = \sigma\left(\mathbf{W}_e^{(l)}\sigma\left(\mathbf{W}_e^{(l-1)}\cdots\sigma\left(\mathbf{W}_e^{(2)}\sigma\left(\mathbf{W}_e^{(1)}\mathbf{x}\right)\right)\cdots\right)\right). \tag{26}$$

According to (12), $\mathbf{W}_e^{(l)}$ could be obtained as follows:

$$\mathbf{W}_e^{(l)} = \left(\mathbf{H}_{l-1}(\mathbf{H}_l)^{\mathrm{T}}\left(\mathbf{H}_l(\mathbf{H}_l)^{\mathrm{T}} + \lambda\mathbf{I}\right)^{-1}\right)^{\mathrm{T}} \tag{27}$$

in which $\mathbf{H}_l$ is calculated as follows:

$$\mathbf{H}_l = \begin{cases}\sigma\left(\mathbf{W}_e^{(l)}\cdots\sigma\left(\mathbf{W}_e^{(2)}\sigma\left(\mathbf{W}_e^{(1)}\mathbf{X}\right)\right)\cdots\right), & l > 0\\ \mathbf{X}, & l = 0.\end{cases} \tag{28}$$

C. Bidirectional Learning Scheme

As is shown in Fig. 2(a), we perform the forward and BT sequentially in the training phase, resulting in a forward subnetwork and a backward subnetwork, respectively.

1) Forward Process: During the FT, we construct a multilayer network using PILAE as the base component, as described earlier. As for the hyperparameter l in (26)–(28), since it is usually intractable to determine the depth of the model in advance, it is natural to build the model dynamically and gradually. To this end, we could start with only one base component and add new ones gradually. This procedure is repeated until the performance on a validation set converges, which corresponds to the early stop exits shown in Fig. 2(a). Note that the proposed model consists of two subnetworks, so after completing the construction and training of the forward subnetwork, a twin backward subnetwork can be obtained simultaneously. This backward subnetwork shares the same network architecture as the forward subnetwork. In addition, the connection weights of the backward subnetwork are initialized with the ones of the forward subnetwork and will be updated in the subsequent BT process.

2) Backward Process: The FT is based on the unsupervised data reconstruction task to extract features. This inevitably leads the features to overfit to the data reconstruction task, which may be not beneficial for downstream learning tasks. In addition, the FT fails to fully utilize the information in the target labels which is clearly helpful to extract features relevant to specific learning tasks. This motivates us to propose the BT as a complement to the FT.

Recall that the backward subnetwork work has been obtained as a twin of the forward subnetwork after the FT. The BT is conducted from the last hidden layer to the input layer. According to (26), the final learning task (e.g., classification) is performed as follows:

$$\mathbf{O} = \mathbf{W}_o f(\mathbf{x}) \tag{29}$$

where $\mathbf{O}$ denotes the outputs, $\mathbf{W}_o$ is the weights connecting the last hidden layer with the output layer and can be calculated according to (12) as follows:

$$\mathbf{W}_o = \mathbf{Y} f(\mathbf{x})^{\mathrm{T}} \left(f(\mathbf{x}) f(\mathbf{x})^{\mathrm{T}} + \lambda \mathbf{I} \right)^{-1}. \tag{30}$$

In the supervised learning task, $\mathbf{O}$ is expected to be close to the target label matrix $\mathbf{Y}$. Hence, we can solve the following optimization problem:

$$\arg\min_{\hat{\mathbf{H}}_l} \|\mathbf{O} - \mathbf{Y}\|_2 = \arg\min_{\hat{\mathbf{H}}_l} \|\mathbf{W}_o \hat{\mathbf{H}}_l - \mathbf{Y}\|_2. \tag{31}$$

According to Theorem 2, the matrix $\hat{\mathbf{H}}_l$ that minimizes the error can be obtained as follows:

$$\hat{\mathbf{H}}_l = \mathbf{W}_o^{+} \mathbf{Y} = \left(\mathbf{W}_o^{+} \right)^{\mathrm{T}} \left(\mathbf{W}_o^{+} \left(\mathbf{W}_o^{+} \right)^{\mathrm{T}} + \lambda \mathbf{I} \right)^{-1} \mathbf{Y}. \tag{32}$$

Similarly, $\hat{\mathbf{H}}_{l-1}$ corresponding to the $(l-1)$th hidden layer can be calculated as follows:

$$\hat{\mathbf{H}}_{l-1} = \left(\mathbf{W}_e^{(l)} \right)^{+} \sigma^{-1}(\hat{\mathbf{H}}_l). \tag{33}$$

$\sigma^{-1}(\cdot)$ is the inverse of the activation function $\sigma(\cdot)$. The aforementioned procedure should be repeated from the lth hidden layer backward to the first hidden layer, which can be formulated in the following form:

$$\hat{\mathbf{H}}_1 = \left(\mathbf{W}_e^{(2)} \right)^{+} \times \sigma^{-1} \left(\left(\mathbf{W}_e^{(3)} \right)^{+} \sigma^{-1} \left(\cdots \left(\mathbf{W}_e^{(l)} \right)^{+} \sigma^{-1} \left(\mathbf{W}_o^{+} \mathbf{Y} \right) \cdots \right) \right). \tag{34}$$

This process could be regarded as the backpropagation of label information through the network. Note that the "backpropagation" mentioned here is also gradient-free, which is different from the gradient-based backpropagation algorithms. Once $\hat{\mathbf{H}}_1$ is obtained, the weights of the backward subnetwork can be updated from the input layer to the output layer. Let matrix $\mathbf{W}_b^{(1)}$ denote the weights connecting the input layer and the first hidden layer, we can update the weights by solving the following problem:

$$\arg\min_{\mathbf{W}_b^{(1)}} \|\mathbf{W}_b^{(1)} \mathbf{X} - \hat{\mathbf{H}}_1\|_2. \tag{35}$$

This problem is similar with the one defined in Theorem 2. The optimal approximate solution can be derived as follows:

$$\mathbf{W}_b^{(1)} = \hat{\mathbf{H}}_1 \mathbf{X}^{+}. \tag{36}$$

Without loss of generality, let $\hat{\mathbf{H}}_{l+1}$ denote $\mathbf{Y}$, the procedure described in (36) can be generalized as follows:

$$\begin{aligned} \mathbf{W}_b^{(2)} &= \hat{\mathbf{H}}_2 \left(\sigma \left(\mathbf{W}_b^{(1)} \mathbf{X} \right) \right)^{+} \\ &\cdots \\ \mathbf{W}_b^{(l+1)} &= \hat{\mathbf{H}}_{l+1} \left(\sigma \left(\mathbf{W}_b^{(l)} \sigma \left(\mathbf{W}_b^{(l-1)} \sigma \left(\cdots \sigma \left(\mathbf{W}_b^{(1)} \mathbf{X} \right) \cdots \right) \right) \right) \right)^{+}. \end{aligned} \tag{37}$$

This process updates the weights of the backward subnetwork layer by layer. Once the weights have been determined, the learning task is performed by the backward subnetwork as follows:

$$\mathbf{O} = \mathbf{W}_b^{(l+1)} \sigma \left(\mathbf{W}_b^{(l-1)} \sigma \left(\mathbf{W}_b^{(l-2)} \sigma \left(\cdots \sigma \left(\mathbf{W}_b^{(1)} \mathbf{X} \right) \cdots \right) \right) \right). \tag{38}$$

Once the bidirectional training mentioned above is completed, two subnetworks, i.e., the forward and the backward subnetworks are obtained. These two subnetworks share the same architecture but have different weights that are determined within different training processes. The subnetworks will be used as representation learning models rather than performing the final learning task directly.

3) Feature Fusion: The features from different subnetworks are derived with different optimization objectives, and hence the patterns they focus on are also different. In order to improve the generalization performance, the features from different hidden layers of different subnetworks are fused. As shown in Fig. 2(b), features from different layers are concatenated and then used to conduct the subsequent task. For two subnetworks with l hidden layers for each, there exists a total of $l \times l$ fusion ways. If we further fuse the results of different feature fusions, more fusion results can be obtained. Furthermore, the prediction results derived from different feature fusion ways can also be fused. From the view of ensemble learning, this fusion process is equivalent to constructing a series of ensemble models with different topologies which are embedded in the subnetworks.

D. Single Hidden Layer Feedforward Neural Network

The fused features are then input into a module to perform specific learning tasks, such as classification or regression. In the proposed model, an additional SHLFN is employed to perform this task. This network can also be trained using the PIL algorithm analogous to PILAE. First, we set the input weights in the same way as PILAE. Then, the output weights can be calculated by replacing $\mathbf{X}$ in (12) with label matrix $\mathbf{Y}$.

For this SHLFN, the number of neurons in the hidden layer is a hyperparameter that is crucial to the performance. To properly set this hyperparameter, we adopt an empirical formula to estimate it based on both the rank and the number of training samples of the dataset. This formula is defined as follows:

$$h_n = \begin{cases} P(N, r), & \text{if } P(N, r) > 0 \\ r + \lceil \alpha(d - r) \rceil, \alpha \in (0, 1], & \text{otherwise} \end{cases} \tag{39}$$

where

$$P(N, r) = \left\lceil \theta_0 + \theta_1 N + \theta_2 r + \theta_3 N^2 + \theta_4 r^2 \right\rceil \tag{40}$$

where r represents the rank, N denotes the count of training samples, and θ_i represents coefficients. To determine the coefficients, we first prepare several datasets with varying ranks and sample counts. Subsequently, a series of SHLFNs are trained with different h_n on each dataset to search a well-performed architecture, i.e., a proper h_n. By doing this, we can construct a dataset of datasets, $\mathcal{D} = \{\mathbf{x}^i, \mathbf{y}^i\}_{i=1}^{M}$, in which the sample $\mathbf{x}^i$ specifies the rank and sample count of the ith dataset. $\mathbf{y}^i = h_n^i$ is the proper h_n for the ith dataset. Then, a regression analysis

TABLE II
COMPARISON OF THE CLASSIFICATION PERFORMANCE BETWEEN THE PROPOSED METHOD WITH BASELINES

Data sets	AE	HELM	ELM-AE	BLS	stacked BLS	PILLS	PILAE	NF-RVFL	SLatDPL	RBM	Bi-PIL (l_1)	Bi-PIL (LR)	Bi-PIL (l_2)
Segment	91.59	92.32	93.01	92.86	93.13	89.28	95.39	93.30	85.85	93.33	95.65	**96.00**	94.20
Mfeat	98.99	96.00	95.13	99.33	97.99	98.49	98.75	83.37	96.88	98.49	99.19	**99.50**	98.79
Prior	94.78	94.81	95.67	93.37	92.17	91.51	89.32	88.78	83.80	90.14	**97.03**	96.80	94.66
Advertisement	95.71	96.94	93.77	91.76	96.69	96.73	95.42	72.60	82.88	96.93	**97.54**	97.24	97.18
Gina agnostic	90.54	87.17	88.60	84.62	88.81	88.22	84.70	87.34	75.54	85.14	92.67	**93.50**	88.07
Spambase	92.30	90.19	85.83	86.14	86.55	91.13	91.09	91.32	87.55	82.85	**92.59**	91.14	91.83
Isolet	95.24	96.22	96.49	96.67	96.09	94.80	92.69	91.71	87.04	95.37	**97.15**	94.35	96.87
Sylva	99.30	98.82	98.92	99.04	99.34	98.66	98.61	98.51	95.83	99.17	99.55	99.37	**99.63**
Har	96.10	83.88	96.68	82.76	96.63	**97.35**	95.95	96.20	91.40	93.48	96.74	96.61	96.99
Abalone	58.17	57.62	56.52	56.24	56.54	**59.84**	59.36	59.14	55.23	57.93	58.95	57.22	56.22
Amazon	69.08	77.07	71.87	80.80	75.17	70.85	55.61	**82.06**	-	52.95	79.20	77.39	78.92
Gisette	98.31	97.60	97.88	97.88	97.36	97.33	97.22	98.00	89.75	95.49	98.37	97.83	**98.40**
Kin8nm	88.13	79.77	77.53	81.65	82.28	85.82	80.44	72.62	57.38	81.15	89.67	88.93	**89.71**
Madelon	82.24	72.36	71.90	64.13	64.55	58.25	56.70	60.65	58.65	70.27	**83.94**	79.75	62.70
Occupancy	96.56	96.56	98.54	97.45	98.77	98.54	97.47	89.59	98.37	98.7	**98.94**	98.41	**98.94**
Semeion	92.99	95.26	95.14	94.93	95.16	94.51	94.94	86.88	92.87	85.99	**97.83**	97.07	84.56
Yeast	58.22	61.62	58.83	61.71	60.00	56.56	55.66	58.72	52.44	58.90	**62.60**	60.55	56.16
Average rank	6.29	7.35	7.24	7.53	6.47	7.44	8.88	8.76	11.75	8.21	1.82	4.12	5.00

The "average rank" for each method in the last row is determined by computing the mean of its accuracy rankings across all datasets.
Bi-PIL with l_1 regularization, Bi-PIL with low-rank approximation, and Bi-PIL with l_2 regularization are denoted as Bi-PIL (l_1), Bi-PIL (LR), and Bi-PIL(l_2), respectively.

is conducted on dataset $\mathcal{D}$ to learn the relationship between the rank, sample count, and hyperparameter h_n.

IV. EXPERIMENTS AND DISCUSSION

In this section, comparison experiments on classification tasks are conducted to verify the comprehensive performance of our proposed models including Bi-PIL with l_1 regularization denoted as Bi-PIL (l_1), Bi-PIL with low-rank approximation denoted as Bi-PIL (LR), and Bi-PIL with l_2 denoted as Bi-PIL (l_2), respectively.

A. Baselines and Setup

By referring to the literature proposing the representative competing methods, we choose several gradient-free methods as the baselines including HELM [47], ELM-AE [48] BLS [8], stacked BLS [49], PILLS [22], PILAE [38], KARnet [10], ANnet [9], CPNet [11], twin-incoherent self-expressive latent DPL (SLatDPL) [50], and neuro-fuzzy RVFL (NF-RVFL) [51]. We also compared with gradient-based methods, including AE [52], restricted Boltzmann machine (RBM) [53], vision transformer (ViT), as well as a newly proposed variant broad attention-based ViT (BViT) [54]. In addition, the baselines in the experiments are grouped into two categories. The baselines of the first category conduct the representation learning within the framework of AE that is based on the data reconstruction task. The second category is trained in the supervised way, which uses the label to obtain the weights layer by layer. The first group includes AE, RBM, HELM, ELM-AE, BLS, stacked BLS PILLS, PILAE, SLatDPL, and NF-RVFL. The second group includes KARnet, ANnet, and CPNet. Due to the page limit, the hyperparameters for different models involved in the experiments, including both learning control parameters and architecture hyperparameters, are summarized at https://github.com/B-berrypie/Bi-PIL/blob/main/Hyperparameter%20Settings.pdf

TABLE III
COMPARISON OF THE PERFORMANCE BETWEEN THE PROPOSED METHOD WITH KARNET, ANNET, AND CPNET

Data sets	KARnet	ANnet	CPNet	Bi-PIL (l_1)	Bi-PIL (LR)	Bi-PIL (l_2)
Letter	94.12	69.29	94.73	**95.16**	94.97	94.89
Optdigits	96.96	**99.09**	97.24	98.75	98.79	98.52
Pendigits	96.05	98.36	97.19	**98.44**	98.31	98.21
Balance	90.92	90.98	93.33	**99.02**	98.03	97.70
Waveform	83.27	82.17	86.16	**87.61**	87.14	86.77
Mushroom	95.71	95.62	99.67	**100.00**	**100.00**	**100.00**
Shuttle	95.52	96.85	97.20	99.02	98.98	**99.06**
Sonar	72.70	73.40	78.10	82.07	**85.26**	81.38
Average rank	5.63	4.50	4.25	1.50	2.00	2.75

The "average rank" for each method in the last row is determined by computing the mean of its accuracy rankings across all datasets.
Bi-PIL with l_1 regularization, Bi-PIL with low-rank approximation, and Bi-PIL with l_2 regularization are denoted as Bi-PIL (l_1), Bi-PIL (LR), and Bi-PIL(l_2), respectively.

B. Datasets

By reviewing the literature on competing methods, we use various benchmark datasets commonly used in related works, including MNIST, Fashion-MNIST, NORB, and 25 datasets from the UCI repository (http://archive.ics.uci.edu/ml) and the OpenML community (http://openml.org) to facilitate the comparison.

C. Generalization Comparison

Classification tasks are carried out on different datasets to compare the generalization performance between the proposed models and baselines. For a given dataset, since the hyperparameters and model architecture are crucial to the performance, we adopt the same setting reported in the related references if this dataset was used in their experiments.

1) Uci and OpenML Datasets: The comparison results of the proposed methods with the first and second group of

TABLE IV
COMPARISON OF THE PERFORMANCE ON MNIST, FASHION-MNIST, AND NORB

	MNIST	Fashion-MNIST	NORB	Average rank
AE	98.26±0.16	87.29±0.08	87.31±0.13	8.00
HELM	98.75±0.05	88.53±0.20	88.70±0.27	6.00
ELM-AE	98.78±0.03	88.60±0.11	89.14±0.15	4.67
BLS	98.60±0.01	89.60±0.15	86.79±0.50	7.00
stacked BLS	98.71±0.06	90.14±0.10	90.08±0.43	3.33
PILLS	97.48±0.21	87.29±0.15	90.46±0.17	7.00
PILAE	96.93±0.03	86.32±0.17	88.11±0.54	10.67
NF-RVFL	96.60±0.00	86.08±0.00	85.70±0.00	13.33
SLatDPL	94.22±0.00	83.59±0.00	76.49±0.00	15.00
ViT	98.21±0.16	89.66±0.36	85.08±1.02	8.33
RBM	97.33±0.04	85.53±0.15	87.04±0.39	11.67
BViT	98.71±0.18	88.77±0.23	86.06±0.84	7.67
Bi-PIL (l_1)	**98.78±0.01**	**90.19±0.06**	**91.05±0.05**	1.00
Bi-PIL (LR)	97.96±0.04	89.48±0.02	89.74±0.45	6.00
Bi-PIL (l_2)	97.07±0.08	88.94±0.14	87.12±0.35	9.00

The "average rank" for each method in the last column is determined by computing the mean of its accuracy rankings across all datasets.
Bi-PIL with l_1 regularization, Bi-PIL with low-rank approximation, and Bi-PIL with l_2 regularization are denoted as Bi-PIL (l_1), Bi-PIL (LR), and Bi-PIL(l_2), respectively.

baselines are summarized in Tables II and III, respectively. The performance is evaluated through the classification accuracy. The bottom row of the table presents the average rank of these methods in terms of classification accuracy across all datasets. It can be observed from Tables II and III that the three methods based on Bi-PIL outperform the baselines. Among the three Bi-PIL-based methods, Bi-PIL (l_1) achieves the best performance, followed by Bi-PIL (LR), while the average rank of Bi-PIL (l_2) is lower than the two aforementioned methods.

2) MNIST, Fashion-MNIST, and NORB: The performance comparison results of the proposed models with the competing baselines on MNIST, Fashion-MNIST, and NORB are summarized in Table IV. Consistent with the previous experimental setup, the performance is evaluated through classification accuracy. Bi-PIL (l_1) continues to achieve the best performance while Bi-PIL (LR) ranks fourth in this experiment, slightly trailing behind stacked BLS, and ELM-AE.

Since the proposed method aims to learn features, we conducted a visual analysis of the method's performance in feature learning tasks. Specifically, we utilized the Bi-PIL (l_1) model to extract and fuse features. We then applied t-SNE to these obtained features for visualization and compared the results with those obtained by directly applying t-SNE to the raw data. The results, as shown in Fig. 3, intuitively demonstrate that the features obtained by our method exhibit superior discriminability.

D. Efficiency Comparison

In addition to generalization performance, we also compare the training efficiency of the proposed methods with competing baselines, as training efficiency has always been a significant advantage and the primary motivation for gradient-free

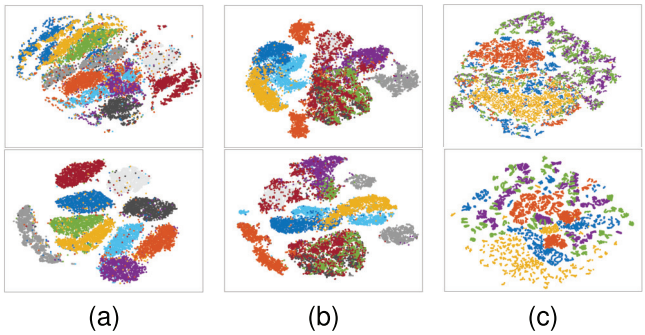


Fig. 3. Visualization of the representation obtained by using t-SNE. The top row presents the visualization results obtained by applying t-SNE directly to the raw data. The bottom row shows the visualization results obtained by applying t-SNE to the fused features. (a)–(c) Correspond to the MNIST, Fashion-MNIST, and NORB datasets, respectively.

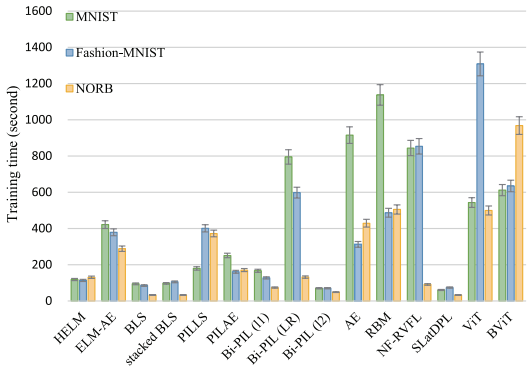


Fig. 4. Comparison of elapsed training time.

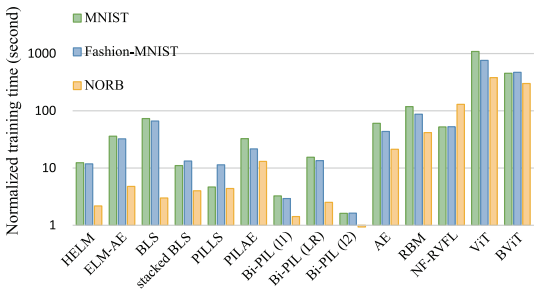


Fig. 5. Comparison of normalized training time calculated as dividing the original training time by model scales and then taking the logarithm with 10 as the base. The model scales are given in a million parameters.

algorithms. Fig. 4 illustrates the comparison of elapsed training time on different datasets. Compared to gradient descent-based methods such as AE, RBM, and transformer-based methods, our proposed methods, particularly Bi-PIL (l_1) and Bi-PIL (l_2), exhibit efficiency advantages consistent with most other gradient-free learning algorithms. Specifically, Bi-PIL (l_2) requires less training time than most other gradient-free learning algorithms and is comparable with SLatDPL. This is because Bi-PIL (l_2) is not only gradient-free but also does not require any iterative computations (such as the iterative shrinkage-thresholding algorithm or truncated SVD employed by other gradient-free methods).

It is worth noting that different methods utilize different network architectures. Therefore, apart from directly comparing the elapsed training time, we also compare the normalized training time that is calculated by dividing the elapsed training time by the model size (measured in millions of parameters) and then taking the logarithm with 10 as the base. Fig. 5 illustrates the comparison result of normalized training time (SLatDPL is excluded because it is not based on neural networks). The result reveals that Bi-PIL (l_2) remains the most efficient method. But when considering the model size, Bi-PIL (l_1) outperforms other methods and ranks second in efficiency. As for Bi-PIL (LR), since it requires truncated SVD, its training time is longer than both Bi-PIL (l_2) and Bi-PIL (l_1). Nevertheless, it still outperforms BLS, PILAE, and ELM-AE. It is noteworthy that PILLS also demonstrates relatively high efficiency when normalized training time is adopted as the metric. This is primarily due to the divide-and-conquer strategy employed by PILLS during training, which involves dividing the training set into a series of subsets and training submodels on each subset.

E. Ablation Study

Since our proposed methodology comprises multiple components, we conducted ablation experiments to analyze the effectiveness of each individual component. Specifically, we evaluated the following four components.

1) *Learnable Initialization of Weights for the BT (LIWBT):* We examined the effect of using learnable initialization of weights for the backward subnetwork, comparing it with random initialization.
2) *FT:* We analyzed the individual contribution of the forward subnetwork to the final performance by removing other components.
3) *BT:* We analyzed the individual contribution of the backward subnetwork to the final performance by removing other components.

The results of the ablation study are shown in Table V. It can be observed from the results that the LIWBT leads to improved classification performance on almost all datasets, with the exception of the Mushroom and Shuttle datasets, where the performance remains unchanged. In addition, the simultaneous employment of FT and BT produces superior results in 9 of the 14 datasets. This finding demonstrates the efficacy of feature fusion. In summary, the bidirectional learning framework introduced in this study surpasses the efficacy of conventional unidirectional learning approaches in most cases.

F. Performance With Respect to Hyperparameters

1) Performance With Respect to h_n: The objective of this experiment is to verify if the method described in (39) can find a good network architecture. To this end, we implement a leave-one-out cross-validation on the dataset of datasets, $\mathcal{D}$. Each sample is used as the validation set while the remaining ones are utilized to learn the relationship outlined in (40) via bivariate regression analysis. The learned relationship is then used to predict the proper h_n of the validation set.

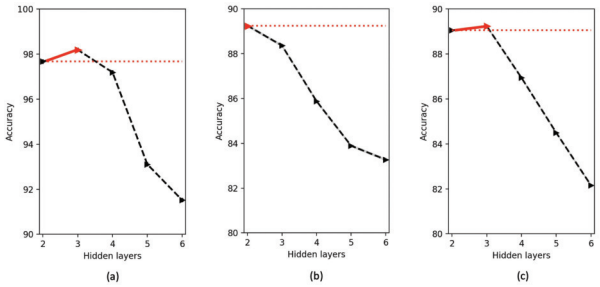


Fig. 6. Relationships between the accuracy and network depth (measured in hidden layer count) on (a) MNIST, (b) Fashin-MNIST, and (c) NORB.

Table VI summarizes the comparison of performance corresponding to the predicted h_n and the one acquired through search, which is denoted as $h_n{}^*$. The column "Test accuracy*" corresponds to the results obtained by using the PIL algorithm with the architecture specified by $h_n{}^*$. "Test accuracy" and "Test accuracy†," respectively, correspond to the results of the PIL algorithm and BP-based algorithm with the exact same architecture that is specified by h_n. The values marked with "(f)" are negative, indicating failure cases of (40). The column labeled as "Training time" demonstrates the training time of our proposed method, while the column "Training time†" represents the training time of the BP-based algorithm.

Table VI shows that the classification performance associated with the predicted h_n is comparable to the one with $h_n{}^*$, which indicates the effectiveness of the method described in (39). For the same network architecture specified by h_n, though the BP-based algorithm achieves slightly better results than the PIL algorithm, however, the elapsed training time is much more.

2) Performance With Respect to Model Depth: In this section, the relationships between the accuracy and network depth are analyzed in order to demonstrate the effectiveness of our strategy of dynamically adding hidden layers along with early stops for the FT process. The model we adopted in this experiment is Bi-PIL (l_1). The results on three datasets are shown in Fig. 6. The black dashed lines in the figures represent the relationship between the classification accuracy and the count of hidden layers. The red solid lines depict the process of dynamically adding hidden layers by using our proposed method and when this process terminates. The red dashed lines represent the baseline accuracy with the initial network depth, aiming to facilitate performance comparison and visually demonstrate the effectiveness of our proposed method.

For the MNIST and NORB datasets, the accuracy initially increases as the network goes deeper, but it begins to decrease after the network reaches a certain depth. The reason for this decrease in accuracy may be due to the overfitting caused by the increasing depth. For the Fashion-MNIST dataset, the optimal accuracy is achieved with the initial network structure, and further increasing the depth of the network does not enhance the model's performance. From the experimental results presented above, it can be observed that our proposed method of dynamically adding hidden layers combined with an early stop could determine the appropriate network depth

TABLE V
ABLATION STUDY RESULTS ON DIFFERENT DATASETS

Data sets	LIWBT	FT	BT	Accuracy	Data sets	LIWBT	FT	BT	Accuracy
Advertisement	✓	✓	✓	**97.54**	Waveform	✓	✓	✓	**87.61**
Advertisement		✓	✓	96.81	Waveform		✓	✓	85.57
Advertisement		✓		95.42	Waveform		✓		86.97
Advertisement			✓	95.40	Waveform			✓	81.72
Occupancy	✓	✓	✓	**98.94**	Mushroom	✓	✓	✓	**100.00**
Occupancy		✓	✓	98.64	Mushroom		✓	✓	**100.00**
Occupancy		✓		97.47	Mushroom		✓		99.88
Occupancy			✓	96.32	Mushroom			✓	**100.00**
Semeion	✓	✓	✓	**97.83**	Shuttle	✓	✓	✓	**99.02**
Semeion		✓	✓	93.88	Shuttle		✓	✓	**99.02**
Semeion		✓		94.94	Shuttle		✓		96.12
Semeion			✓	84.46	Shuttle			✓	87.42
Letter	✓	✓	✓	**95.16**	Sonar	✓	✓	✓	**82.07**
Letter		✓	✓	94.56	Sonar		✓	✓	80.69
Letter		✓		94.64	Sonar		✓		78.62
Letter			✓	75.46	Sonar			✓	73.10
Optdigits	✓	✓	✓	**98.75**	MNIST	✓	✓	✓	**98.78**
Optdigits		✓	✓	98.43	MNIST		✓	✓	98.14
Optdigits		✓		98.33	MNIST		✓		96.93
Optdigits			✓	94.36	MNIST			✓	89.50
Pendigits	✓	✓	✓	**98.44**	Fashion-MNIST	✓	✓	✓	**90.19**
Pendigits		✓	✓	98.25	Fashion-MNIST		✓	✓	89.53
Pendigits		✓		98.27	Fashion-MNIST		✓		86.32
Pendigits			✓	81.14	Fashion-MNIST			✓	83.08
Balance	✓	✓	✓	**99.02**	NORB	✓	✓	✓	**91.05**
Balance		✓	✓	97.39	NORB		✓	✓	87.62
Balance		✓		97.38	NORB		✓		88.11
Balance			✓	90.16	NORB			✓	71.94

TABLE VI
COMPARISON BETWEEN THE ACCURACY AND TRAINING EFFICIENCY OF THE PROPOSED METHOD WITH OTHER BASELINE METHODS

Data set	$[\theta_0, \theta_1, \theta_2, \theta_3, \theta_4]$	h_n	h_n^*	Test accuracy	Test accuracy†	Test accuracy*	Training time (sec)	Training time†(sec)
Advertisement	[-459.4272,0.6779,4.2053,-5.2552e-5,-0.0029]	2497	2556	95.42	**97.56**	95.42	**2.61**	171.79
Bupa	[-1.2982e3,1.0390,5.0603,-9.0308e-5,-0.0036]	-1064 (f)	161	-	-	81.16	-	-
Gina_agnostic	[-373.4045,0.7241,-1.9230,-4.7683e-5, 0.0063]	5339	2197	84.70	**86.29**	84.99	**5.19**	237.48
Har 1	[-515.3591,0.7571,5.5879,-6.714e-5,-0.0045]	2627	1557	97.67	97.33	**97.83**	**1.04**	146.90
Har 2	[-58.6140,0.4126,3.0542,-2.9070e-5,-0.0014]	2603	4657	98.03	98.16	**99.05**	**1.87**	289.51
Isolet	[-293.6718,0.6184,3.8698,-5.1742e-5,-0.0026]	2949	4062	92.69	92.27	**93.39**	**2.54**	210.99
Mfeat	[-377.4237,0.6002,4.7423,-4.5829e-05,-0.0033]	1921	1265	**98.75**	97.50	**98.75**	**0.65**	47.35
Prior	[-434.6510,0.6614,5.2149,-5.4856e-5,-0.0039]	2416	1979	89.32	**91.20**	90.19	**1.28**	59.33
Segment	[-926.2121,0.6169,5.6153,-4.1785e-5,-0.0037]	-52 (f)	1352	-	-	95.67	-	-
Spambase	[-712.0285,1.8134,0.7562,-0.0002,-0.0013]	3510	586	91.09	**92.80**	90.65	**1.75**	24.57
Sylva	[487.8985,-0.2173,3.04204,0.0001,-0.0014]	5893	1322	98.61	94.03	**98.85**	**9.46**	34.51

effectively and mitigate overfitting by halting the training process.

3) Performance With Respect to Model Width: In this section of the experiment, the relationships between the accuracy and network width are analyzed. The width is measured by the number of hidden neurons. Similar to the setup in [21], [22], and [38], we employ a hyperparameter β denoting the ratio of the hidden neuron count to the dimensionality of input to determine model width. For each value of β, we repeated the experiment ten times and calculated the average accuracy and standard deviation. The model we adopted in this experiment is Bi-PIL (l_1). The results on three datasets are shown in Fig. 7. For the MNIST and NORB datasets, the model performances generally increase with the model

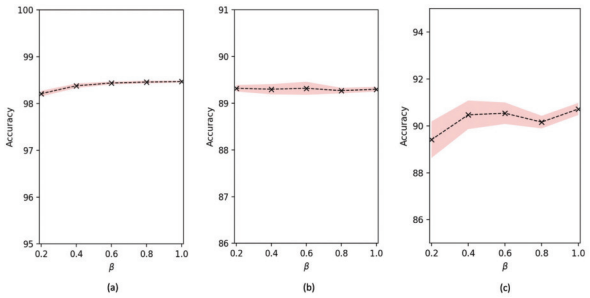


Fig. 7. Relationships between the accuracy and network width (controlled by β) on (a) MNIST, (b) Fashin-MNIST, and (c) NORB.

width. In addition, when the value of β exceeds a specific threshold (e.g., 0.6 in this experiment), the performances tend

to converge. For the Fashion-MNIST dataset, the performance is less sensitive to the model width. Furthermore, across all datasets in this experiment, increasing the model width generally reduces the standard deviation of accuracy. The experimental results indicate that the performance is not highly sensitive to the value of parameter β especially when it is within a range of $[0.6, 1]$. This fact reveals that the architecture tuning of Bi-PIL is easier than engaging in a comprehensive trial-and-error process.

G. Discussion

The results of the generalization comparison experiments demonstrate that the performance of the proposed model based on Bi-PIL is superior to representative nongradient learning algorithms, regardless of whether these algorithms conduct unsupervised representation learning within the AE framework or utilize labels to conduct supervised learning to determine the connection weights layer by layer. This result validates the effectiveness of the proposed bidirectional learning scheme. The reason for the superiority of the proposed learning scheme could be explained by the fact that Bi-PIL addresses the weight initialization issues and provides a fine-tuning approach, both of which are extensions and improvements to existing gradient-free learning algorithms.

The experimental results of efficiency comparison reveal that the proposed Bi-PIL algorithm is comparable in efficiency to typical gradient-free learning algorithms, and hence exhibits superiority in efficiency compared to iterative algorithms based on gradient descent. Furthermore, the comparison of normalized training time indicates that the proposed learning algorithms effectively mitigate unnecessary complexity in model architecture design. This is mainly because we adopted the strategy of dynamically constructing the model rather than determining the architecture prior to training.

From the analysis of performance with respect to model architecture hyperparameters, the following facts should be highlighted: first, for the SHLFN performing a specific learning task (i.e., the head), the design of its structure can be accomplished in a data-driven manner, where heuristic information is provided by the intrinsic properties of the data, such as the number of samples and the rank. Second, for the module performing feature learning (i.e., the backbone), the determination of both its depth and width can also be conducted in a data-driven way. Therefore, the model (including the head and backbone) can be constructed in a quasi-automated manner. Even though this approach may not directly generate the optimal network architecture, it can guide the design process, significantly reducing the search space of the network architecture. Moreover, this strategy is less computationally expensive compared to commonly used AutoML techniques that are based on evolutionary methods [55], Bayesian optimization [56], or reinforcement learning [57].

V. Conclusion

In this article, we propose a bidirectional gradient-free learning scheme for multilayer neural networks. The proposed learning scheme is based on PIL and includes both forward and backward processes. The FT is conducted on unsupervised reconstruction tasks and can automatically determine the network architecture across different hierarchical levels in a data-driven manner. Simultaneously, this FT provides initial weights and architectural design for the subsequent BT. The BT also employs an efficient gradient-free learning algorithm and can be considered a supervised fine-tuning process that embeds labels into the learned features, thereby preventing the network from overfitting to the data reconstruction tasks. The bidirectional training generates two twin networks, and the fusion of their features serves as input for downstream tasks. Our bidirectional gradient-free learning scheme offers a potential solution to the issues in current gradient-free training algorithms for multilayer neural networks, such as the lack of effective fine-tuning and the absence of robust model design methodologies, which often lead to unnecessary complexity.

Current studies of gradient-free learning schemes predominantly concentrate on tabular data (or image data represented in the form of 1-D feature vectors) and employ architectures that resemble MLP. However, the MLP-like network architectures always fall short of effectively extracting semantic information from certain types of multidimensional data, notably image data. In future work, we will investigate more diverse network architectures or integrate with other methodologies to make our method applicable to different types of data. Furthermore, while this study focuses on representation learning and pattern classification tasks, additional research is necessary for other learning tasks, e.g., regression, clustering, and content generation, to fully leverage the potential of the proposed learning schemes.

Acknowledgment

The authors would like to thank the anonymous reviewers for their invaluable insights and constructive comments.

References

[1] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

[2] J.-Y. Li, Z.-H. Zhan, J. Xu, S. Kwong, and J. Zhang, "Surrogate-assisted hybrid-model estimation of distribution algorithm for mixed-variable hyperparameters optimization in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 5, pp. 2338–2352, May 2023.

[3] L. Liao, H. Li, W. Shang, and L. Ma, "An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks," *ACM Trans. Softw. Eng. Methodology*, vol. 31, no. 3, pp. 1–40, Jul. 2022.

[4] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

[5] Y.-H. Pao and Y. Takefuji, "Functional-link net computing: Theory, system architecture, and functionalities," *Computer*, vol. 25, no. 5, pp. 76–79, May 1992.

[6] P. Guo, C. Chen, and Y. Sun, "An exact supervised learning for a three-layer supervised neural network," in *Proc. Int. Conf. Neural Inf. Process. (ICONIP'95)*, Beijing, China, 1995, pp. 1041–1044.

[7] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst., Man, Cybern., B. Cybern.*, vol. 42, no. 2, pp. 513–529, Apr. 2012.

[8] C. L. P. Chen and Z. Liu, "Broad learning system: An effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.

[9] K.-A. Toh, "Kernel and range approach to analytic network learning," *Int. J. Networked Distrib. Comput.*, vol. 7, no. 1, pp. 20–28, 2018.

[10] K.-A. Toh, "Learning from the kernel and the range space," in *Proc. IEEE/ACIS 17th Int. Conf. Comput. Inf. Sci. (ICIS)*, Singapore, Singapore, Jun. 2018, pp. 1–6.

[11] H. Zhuang, Z. Lin, and K.-A. Toh, "Correlation projection for analytic learning of a classification network," *Neural Process. Lett.*, vol. 53, no. 6, pp. 3893–3914, Dec. 2021.

[12] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[13] R. Salakhutdinov and G. Hinton, "Deep Boltzmann machines," in *Proc. 12th Int. Conf. Artif. Intell. Statist.*, Clearwater Beach, FL, USA, D. A. V. Dyk, and M. Welling, Eds., 2009, pp. 448–455.

[14] G. C. Marinó, A. Petrini, D. Malchiodi, and M. Frasca, "Deep neural networks compression: A comparative survey and choice recommendations," *Neurocomputing*, vol. 520, pp. 152–170, Feb. 2023.

[15] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Comput. Surveys*, vol. 55, no. 12, pp. 1–37, 2023.

[16] A. Dosovitskiy et al., "An image is worth 16×16 words: Transformers for image recognition at scale," in *Proc. 9th Int. Conf. Learn. Represent.*, Jan. 2020.

[17] I. Tolstikhin et al., "MLP-mixer: An all-MLP architecture for vision," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 24261–24272.

[18] H. Touvron et al., "ResMLP: Feedforward networks for image classification with data-efficient training," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 4, pp. 5314–5321, Apr. 2023.

[19] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, Helsinki, Finland, 2008, pp. 1096–1103.

[20] P. Guo and M. R. Lyu, "A pseudoinverse learning algorithm for feedforward neural networks with stacked generalization applications to software reliability growth data," *Neurocomputing*, vol. 56, pp. 101–121, Jan. 2004.

[21] K. Wang, P. Guo, X. Xin, and Z. Ye, "Autoencoder, low rank approximation and pseudoinverse learning algorithm," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Banff, AB, Canada, Oct. 2017, pp. 948–953.

[22] K. Wang and P. Guo, "A robust automated machine learning system with pseudoinverse learning," *Cogn. Comput.*, vol. 13, pp. 724–735, Mar. 2021.

[23] Z. Yu, K. Lan, Z. Liu, and G. Han, "Progressive ensemble kernel-based broad learning system for noisy data classification," *IEEE Trans. Cybern.*, vol. 52, no. 9, pp. 9656–9669, Sep. 2022.

[24] X. Deng, M. Mahmoud, Q. Yin, and P. Guo, "An efficient and effective deep convolutional kernel pseudoinverse learner with multi-filter," *Neurocomputing*, vol. 457, nos. 1–4, pp. 74–83, Oct. 2021.

[25] G. Huang, S. Song, J. N. D. Gupta, and C. Wu, "Semi-supervised and unsupervised extreme learning machines," *IEEE Trans. Cybern.*, vol. 44, no. 12, pp. 2405–2417, Dec. 2014.

[26] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. 11th Int. Joint Conf. Artif. Intell.*, Detroit, MI, USA, Aug. 1989, pp. 762–767.

[27] M. Kaveh and M. S. Mesgari, "Application of meta-heuristic algorithms for training neural networks and deep learning architectures: A comprehensive review," *Neural Process. Lett.*, vol. 55, no. 4, pp. 4519–4622, Aug. 2023.

[28] E. Galván and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *IEEE Trans. Artif. Intell.*, vol. 2, no. 6, pp. 476–493, Dec. 2021.

[29] L. Zhang, H. Li, and X.-G. Kong, "Evolving feedforward artificial neural networks using a two-stage approach," *Neurocomputing*, vol. 360, pp. 25–36, Sep. 2019.

[30] J. Liang, G. Chen, B. Qu, C. Yue, K. Yu, and K. Qiao, "Niche-based cooperative co-evolutionary ensemble neural network for classification," *Appl. Soft Comput.*, vol. 113, Dec. 2021, Art. no. 107951.

[31] S. Yang, Y. Tian, C. He, X. Zhang, K. C. Tan, and Y. Jin, "A gradient-guided evolutionary approach to training deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 9, pp. 4861–4875, Sep. 2022.

[32] Y. Xue, Y. Tong, and F. Neri, "An ensemble of differential evolution and Adam for training feed-forward neural networks," *Inf. Sci.*, vol. 608, pp. 453–471, Aug. 2022.

[33] H.-G. Han, W. Lu, Y. Hou, and J.-F. Qiao, "An adaptive-PSO-based self-organizing RBF neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 104–117, Jan. 2018.

[34] R. García-Ródenas, L. J. Linares, and J. A. López-Gómez, "Memetic algorithms for training feedforward neural networks: An approach based on gravitational search algorithm," *Neural Comput. Appl.*, vol. 33, no. 7, pp. 2561–2588, Apr. 2021.

[35] S. Mirjalili, "How effective is the grey wolf optimizer in training multi-layer perceptrons," *Appl. Intell.*, vol. 43, no. 1, pp. 150–161, Jul. 2015.

[36] K. Wang, P. Guo, Q. Yin, A.-L. Luo, and X. Xin, "A pseudoinverse incremental algorithm for fast training deep neural networks with application to spectra pattern recognition," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Vancouver, BC, Canada, Jul. 2016, pp. 3453–3460.

[37] P. Guo, D. Zhao, M. Han, and S. Feng, "Pseudoinverse learners: New trend and applications to big data," in *Proc. INNS Big Data Deep Learn. Conf.*, Genoa, Italy, 2019, pp. 158–168.

[38] P. Guo, K. Wang, and X. L. Zhou, "PILAE: A non-gradient descent learning scheme for deep feedforward neural networks," 2018, *arXiv:1811.01545*.

[39] K. B. Petersen and M. S. Pedersen. (2012). *The Matrix Cookbook*. [Online]. Available: http://matrixcookbook.com

[40] H. Yanai, K. Takeuchi, Y. Takane, H. Yanai, K. Takeuchi, and Y. Takane, "Generalized inverse matrices," in *Projection Matrices, Generalized Inverse Matrices, and Singular Value Decomposition*. New York, NY, USA: Springer, 2011, pp. 55–86.

[41] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Stat. Soc., Ser. B, Methodol.*, vol. 58, no. 1, pp. 267–288, Jan. 1996.

[42] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[43] S. Boyd et al., "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[44] A. Ben-Tal and A. Nemirovskii, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. Philadelphia, PA, USA: SIAM, 2001.

[45] J. M. Bioucas-Dias and M. A. T. Figueiredo, "A new TwIST: Two-step iterative shrinkage/thresholding algorithms for image restoration," *IEEE Trans. Image Process.*, vol. 16, no. 12, pp. 2992–3004, Dec. 2007.

[46] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, Jan. 2009.

[47] J. X. Tang, C. Deng, and G. B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 4, pp. 809–821, May 2016.

[48] L. L. C. Kasun, H. M. Zhou, G. B. Huang, and C. M. Vong, "Representational learning with ELMs for big data," *IEEE Intell. Syst.*, vol. 28, no. 6, pp. 31–34, Nov. 2013.

[49] Z. Liu, C. L. P. Chen, S. Feng, Q. Feng, and T. Zhang, "Stacked broad learning system: From incremental flatted structure to deep model," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 1, pp. 209–222, Jan. 2021.

[50] Z. Zhang et al., "Twin-incoherent self-expressive locality-adaptive latent dictionary pair learning for classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 3, pp. 947–961, Mar. 2021.

[51] M. Sajid, A. K. Malik, M. Tanveer, and P. N. Suganthan, "Neuro-fuzzy random vector functional link neural network for classification and regression problems," *IEEE Trans. Fuzzy Syst.*, vol. 32, no. 5, pp. 2738–2749, May 2024.

[52] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu, "Autoencoders and their applications in machine learning: A survey," *Artif. Intell. Rev.*, vol. 57, no. 2, p. 28, Feb. 2024.

[53] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. 19th Int. Conf. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, 2006, pp. 153–160.

[54] N. Li, Y. Chen, W. Li, Z. Ding, D. Zhao, and S. Nie, "BViT: Broad attention-based vision transformer," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 9, pp. 12772–12783, Sep. 2024.

[55] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 2, pp. 550–570, Feb. 2023.

[56] H. Zhou, M. Yang, J. Wang, and W. Pan, "Bayesnas: A Bayesian approach for neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, Long Beach, CA, USA, 2019, pp. 7603–7613.

[57] B. Lyu, S. Wen, K. Shi, and T. Huang, "Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing," *IEEE Trans. Cybern.*, vol. 53, no. 2, pp. 1158–1169, Feb. 2023.

Ke Wang received the B.S. degree in software engineering and the M.S. and Ph.D. degrees in computer science from Beijing Institute of Technology, Beijing, China, in 2007, 2009, and 2017, respectively.

From 2009 to 2013, he was a Research and Development Engineer at MediaTek (Beijing) Inc., Beijing. From 2017 to 2019, he was a Post-Doctoral Research Fellow at the Department of Electrical and Computer Engineering, University of Michigan–Dearborn, Dearborn, MI, USA. He is currently an Associate Professor with the School of Computer Science and Artificial Intelligence, Zhengzhou University, Zhengzhou, China. His research interests include computational intelligence, machine learning theories, algorithms, and applications.

Binghong Liu received the B.E. degree from Henan University, Kaifeng, China, in 2022. She is currently pursuing the M.S. degree with the School of Computer Science and Artificial Intelligence, Zhengzhou University, Zhengzhou, China.

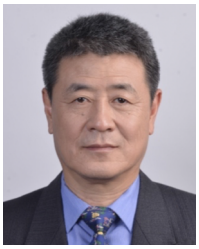
Her research interests include deep learning, gradient-free learning algorithms, and computer vision.

Pandi Liu received the B.E. degree in statistics from North China University of Water Resources and Electric Power, Zhengzhou, China, in 2022. She is currently pursuing the M.S. degree with the School of Computer Science and Artificial Intelligence, Zhengzhou University, Zhengzhou.

Her research interests include pseudoinverse learning, image retrieval, and deep learning.

Yungao Shi received the B.Eng. degree in information management and information systems from Hainan University, Haikou, China, in 2023. He is currently pursuing the M.S. degree in computer technology with the School of Computer Science and Artificial Intelligence, Zhengzhou University, Zhengzhou, China.

His current research interests include machine learning and content generation.



Ping Guo (Senior Member, IEEE) received the M.S. degree in optics from the Department of Physics, Peking University, Beijing, China, in 1983, and the Ph.D. degree in computer science from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2002.

He is a Professor (Emeritus) with the School of Systems Science, Beijing Normal University, Beijing, where he is the Founding Director of the Laboratory of Graphics, Image and Pattern Recognition. He has authored more than 360 articles and holds six patents. He is the author of two Chinese books titled *Computational Intelligence in Software Reliability Engineering* and *Image Semantic Analysis*. His current research interests include computational intelligence theory, as well as applications in pattern recognition, image processing, software reliability engineering, and astronomical data processing.

Dr. Guo is a CCF Senior Member. He received the 2012 Beijing Municipal Government Award of Science and Technology (Third Rank) titled *Regularization Method and Its Application*.

Yafei Li (Member, IEEE) received the Ph.D. degree in computer science from Hong Kong Baptist University, Hong Kong, in 2015.

He is currently a Professor with the School of Computer Science and Artificial Intelligence, Zhengzhou University, Zhengzhou, China. His research interests include crowd intelligence, mobile and spatial data management, location-based services, and urban computing. He has authored more than 30 journal and conference papers in these areas, including IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, PVLDB, IEEE ICDE, and WWW.

Mingliang Xu received the Ph.D. degree in computer science and technology from the State Key Laboratory of Computer Aided Design and Computer Graphics, Zhejiang University, Hangzhou, China, in 2012.

He is currently a Full Professor with the School of Computer Science and Artificial Intelligence, Zhengzhou University, Zhengzhou, China. His research interests include computer graphics, multimedia, and artificial intelligence. He has authored more than 100 journal articles and conference papers in the above areas, including *ACM Transactions on Graphics*, *ACM Transactions on Intelligent Systems and Technology*, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE TRANSACTIONS ON CYBERNETICS, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, ACM SIGGRAPH (Asia), ACM MM, and ICCV.