



Deep Multi-Layer Neural Network with Variable-Depth Output

Shiueng-Bien Yang * and Ting-Wen Liang †

Department of Digital Content of Application and Management

Wenzao Ursuline University of Languages

900 Mintsu 1st Road, Kaohsiung 807, Taiwan

**98010@mail.wzu.edu.tw; wenzao98010@gmail.com*

†devin@mail.wzu.edu.tw

Received 22 June 2019

Revised 9 August 2021

Accepted 19 September 2023

Published 20 December 2023

In this study, a deep multi-layer neural network (DMLNN) with variable-depth output (VDO), called VDO-DMLNN, is proposed for classification. Unlike the traditional DMLNN, for which a user must define the network architecture in advance, VDO-DMLNN is produced from the top-down, layer by layer, until the classification error rate of VDO-DMLNN no longer decreases. The user thus does not need to define the depth of VDO-DMLNN in advance. The combination of the genetic algorithm (GA) and the self-organizing feature map (SOFM), called GA-SOFM, is proposed to automatically generate the weights and proper number of nodes for each layer in VDO-DMLNN. In addition, the output nodes can be at different levels in VDO-DMLNN rather than all being at the last layer, as in the traditional DMLNN. Thus, the average of computing time required for the recognition of an input sample in VDO-DMLNN is less than that in traditional DMLNN when they have the same classification error rate. Finally, VDO-DMLNN is compared with some state-of-the-art neural networks in the experiments.

Keywords: Multi-layer neural network; deep neural network; genetic algorithm.

1. Introduction

The deep multi-layer neural network (DMLNN) is a known and effective neural network architecture.^{10,16,20,22,25,32,34,37,40,41} In order to enable DMLNN to solve more complex classification problems, it is designed to become very deep by increasing both the number of hidden layers and the number of hidden nodes in each hidden layer. If DMLNN is sufficiently deep, then it can efficiently solve more complex classification problems. The principle of the deep DMLNN involves using the current output layer as the input for the next layer.^{13,17} The concept of DMLNN is to use a

*Corresponding author.

large number of layers to strengthen the classification capability. Recently, DMLNNs have been applied to solve speech processing^{19,39} and pattern recognition.^{8,29} In 2017, a DMLNN including 100 hidden neurons was proposed for recognizing S1 and S2 heart sounds.⁷ In recent years, the new DMLNN, convolutional neural networks (CNNs), has been successfully applied in various fields. In Ref. 15, an innovative modeling approach based on a deep CNN method was presented for the rapid prediction of fluvial flood inundation. In Ref. 24, a distracted driving recognition method based on the deep CNN was proposed for driving image data captured by an in-vehicle camera. The CNN is also applied to the handwriting recognition field. Offline Arabic Handwriting Recognition (OAHR) has recently become instrumental in the areas of pattern recognition and image processing due to its application in several fields, such as office automation and document processing.^{1,3,23} In Ref. 42, DMLNN, designed as a multi-layer fully connected network, was used with higher accuracy than other methods to predict the liquefaction potential. In Ref. 5, a DMLNN architecture with a gated-attention mechanism was proposed for automated diagnosis of diabetic retinopathy. Also, other state-of-the-art DMLNNs and their variants, such as the VGG,³⁰ GoogLeNet,³⁵ ResNet,¹² PreLU-Net¹¹ and BN-Inception¹⁴ networks, have been applied to solve more complex problems. In the experiments, our proposed method is compared with these state-of-the-art models.

Although increasing the depth of DMLNNs is able to solve more complex classification problem, it will cause two general problems. First, the computing complexity is increased when the depth of the DMLNNs is very deep. That is, the design of traditional DMLNNs only considers how to enhance the recognition rate by increasing the depth, but usually ignores the computing time when they are designed. Therefore, in the design of a DMLNN, users usually make the depth of DMLNN very deep in order to achieve a small improvement in recognition rate. This is not helpful for reducing the overall computing time of DMLNN. An optimal DMLNN should consider how to increase the recognition rate and decrease the computing time. In this study, both the classification error rate and computing time are considered to design our proposed deep multi-layer neural network with variable-depth output (VDO), named as VDO-DMLNN. The second problem is that if an input sample is classified in a DMLNN, it travels from the first layer (input layer) to the last layer (output layer). Figure 1 shows an example to illustrate the second problem of DMLNNs. Figure 1(a) shows the training dataset, and Fig. 1(b) shows the corresponding DMLNN (depth = 3) designed to classify these samples in Fig. 1(a). Clearly, each input sample takes the same time to be recognized in Fig. 1(b). However, in many cases, most of the samples are easily identified and only a few samples are confusing and not easily distinguishable. In Fig. 1(c), all training samples in Fig. 1(a) are classified into two clusters, *A* and *B*. Cluster *A* is a pure class, and cluster *B* contains more complex classes. The samples contained in cluster *A* are easily distinguished from the samples in cluster *B* using a simple linear classification. Thus, the input sample belonging to cluster *A* should be recognized earlier than the input sample belonging to cluster *B*. That is, the computing

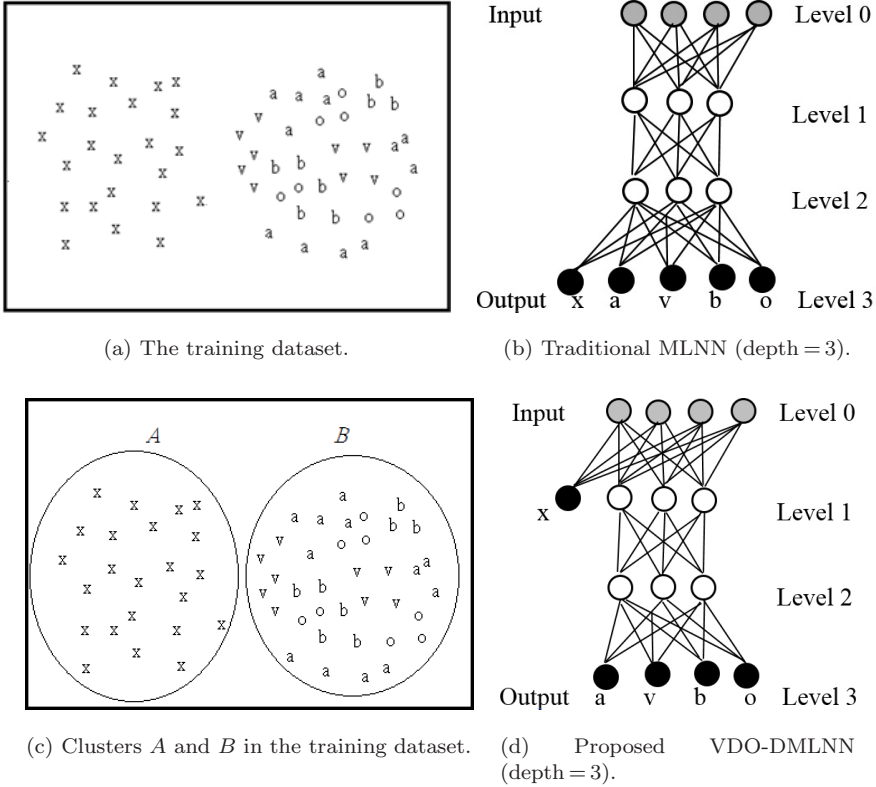


Fig. 1. Comparison of VDO-DMLNN and traditional DMLNNs.

Note: \odot : Input node, \circ : hidden node and \bullet : output node.

time of the input samples in cluster *A* should be smaller than that for the input samples in cluster *B*. Figure 1(d) shows our proposed VDO-DMLNN where the output node for the class *x* can be at different level (level 1) rather than being at the last layer (level 3) as in other methods. That is, the input sample belonging to class *x* can be recognized early than a sample belonging to other classes in our proposed VDO-DMLNN.

Self-organizing feature map (SOFM) is an unsupervised neural network approach that can be used to handle structured data. The principal goal of SOFM is to transform an incoming pattern of arbitrary dimension into a one- or two-dimensional discrete map, and to perform this transformation adaptively in a topologically ordered fashion.^{18,31} However, there are two problems in SOFM. First, the user finds it hard to determine the proper number of output nodes in SOFM. If the number of output nodes is too large, the redundant computing time is increased in SOFM. Otherwise, the classification error rate is increased when the small number of output nodes is given in SOFM. In Ref. 38, when the number of SOFM units is large, to facilitate quantitative analysis of the map and the data, similar units need

to be grouped by the k -means algorithm. However, the users usually have no idea about how to determine the number of output nodes in SOFM, and thus the value of k in the k -means algorithm is hard to be determined by the users. Next, the second problem of SOFM is how to determine these initial weights in it. Usually, SOFM can be initialized using random values for the weight vectors. In Ref. 26, an improved SOFM training algorithm using k -means initialization was proposed. In Ref. 26, the k -means algorithm is used as an initialization step for SOFM. However, the k -means algorithm may fail to converge to a local minimum under certain conditions.²⁸

Furthermore, the genetic algorithm (GA)^{6,21,27,36} has been a popular approach to learn the parameters of NNs. The benefit of GA is that it is able to have a global search in the solution space. However, if the solution space is too large, the GA usually finds a near-optimal solution. Therefore, the combination of GA and SOFM, called GA-SOFM, is proposed in this study. That is, the GA is proposed to automatically generate the proper number of nodes and the initial weights for each layer of VDO-DMLNN, and then these initial weights can be corrected by using the training rule of SOFM. In this study, the GA-SOFM is proposed to design each layer in VDO-DMLNN according to both the computing time and classification error rate of VDO-DMLNN.

Table 1 summarizes the differences between our proposed model and other methods. The contributions of our model are described as follows:

- (1) The VDO-DMLNN is proposed for classification. The difference between VDO-DMLNN and traditional DMLNN is that the output nodes can be at any level (or depth) in VDO-DMLNN, while the output nodes are at the last layer in the traditional MLNN. Thus, VDO-DMLNN has the capacity for early recognition of the input samples, thus reducing the overall recognition time of VDO-DMLNN.

Table 1. Summary of the differences between our proposed model and other methods.

Item	Our Proposed VDO-DMLNN	Other DMLNN Models
Where are the output nodes in the model?	The output nodes can be at any layer in VDO-DMLNN. The advantage is that VDO-DMLNN has the capacity for early recognition of the input samples, thus reducing the overall recognition time of it	The output nodes are at the last layer in DMLNN models, e.g. VGG, GoogLeNet and ResNet. Each input sample should spend the same large computing time while the model is deep
Do the number of layers and the number of nodes per layer need to be determined by the user?	GA-SOFM can automatically determine the number of layers and the number of nodes for each layer in VDO-DMLNN; the designed VDO-DMLNN then tends to become optimal	The users should predefine the number of layers and the number of nodes in each layer in DMLNN by trial and error; the designed DMLNN is then not optimal

- (2) Before designing the traditional MLNN, a user must define the structure of MLNN, including the number of layers and the number of nodes for each layer. However, it is difficult to do this in advance. The design method of VDO-DMLNN is from the top-down, layer by layer; thus, the user does not need to define the depth of VDO-DMLNN in advance. The VDO-DMLNN is designed until the classification error rate no longer decreases.
- (3) The GA-SOFM is proposed to design each layer of VDO-DMLNN. In GA-SOFM, the GA is able to generate the proper number of nodes, and the SOFM is then used to generate the weight for each node according to both the computing time and classification error rate of VDO-DMLNN. Thus, the VDO-DMLNN tends to be optimal.

The concept of VDO-DMLNN is described in Sec. 2. Section 3 shows the design of VDO-DMLNN. Section 4 describes the experimental results. Finally, conclusions are offered in Sec. 5.

2. VDO-DMLNN Concept

Figure 2 shows an example of VDO-DMLNN. In Fig. 2, the nodes O_1, O_2, O_3, O_4, O_5 and O_6 (black circles) denote the output nodes that are used to address the output class, and the other nodes are regarded as the hidden nodes (white circles). Each layer contains different numbers of output and hidden nodes. In VDO-DMLNN, if the classification error rate of the node is less than the threshold given by the user, the node is regarded as an output node. Otherwise, the node with a higher classification error rate than the threshold can be regarded as a hidden node.

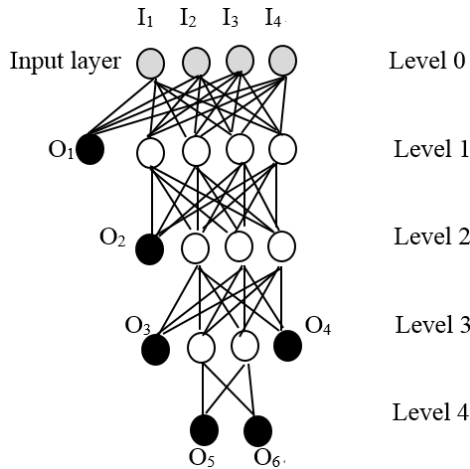


Fig. 2. An example of VDO-DMLNN.

Note: $\textcircled{\bullet}$: Input node, $\textcircled{\circ}$: hidden node and \bullet : output node.

Furthermore, in Fig. 2, only the hidden nodes in the lower level (level $k - 1$) need to be connected to each node in the higher level (level k).

In the recognition phase of VDO-DMLNN, if the input sample arrives at the output node O_1 at level 1 of VDO-DMLNN shown in Fig. 2, it means that the input sample is easily recognized because it can be early recognized using the single-layer NN. Similarly, if the input sample arrives at output nodes O_3 and O_4 at level 3, it indicates that the input sample is hard to be recognized using the three layers. With the real data, most of the data can be identified at an early level in VDO-DMLNN, and only a small part of the data needs to be recognized in the deeper level of the VDO-DMLNN layer. Thus, VDO-DMLNN takes less average computing time than the traditional DMLNNs with fixed depth output.

Let the input layer (level 0) of VDO-DMLNN consist of n_0 input nodes, and let $X = (x_1, x_2, \dots, x_{n_0})$ be the input sample that contains n parameters. The input sample travels from the input layer until it reaches the output node in VDO-DMLNN. The following describes how to recognize the input sample, X , in VDO-DMLNN:

Level 1. Let there be n_1 nodes, $\text{node}_1^1, \text{node}_2^1, \dots, \text{node}_{n_1}^1$, at level 1. Let $w_i^1 = (w_{i,1}^1, w_{i,2}^1, \dots, w_{i,n_0}^1)$ be the weights for node_i^1 . The values of Net_i^1 for node_i^1 , $1 \leq j \leq n_1$, are calculated as

$$\text{Net}_i^1 = \left(\sum_{j=1}^n w_{i,j}^1 x_j \right) - \theta_i^1, \tag{1}$$

where the value of θ_i^1 denotes the activation threshold of node_i^1 at level 1. The Rectified Linear Unit (ReLU), F , denotes the activation function that is defined as

$$F(x) = \max(0, x). \tag{2}$$

Let $\text{node}_{c_1}^1$ have the largest activation value for X at level 1. Then,

$$c_1 = \arg \max \{ F(\text{Net}_1^1), F(\text{Net}_2^1), \dots, F(\text{Net}_{n_1}^1) \}. \tag{3}$$

Then, X is regarded to be classified to $\text{node}_{c_1}^1$ when $\text{node}_{c_1}^1$ has the largest activation value for X . If $\text{node}_{c_1}^1$ is an output node at level 1, the input sample X is recognized as the class represented by $\text{node}_{c_1}^1$. Otherwise, $\text{node}_{c_1}^1$ represents a hidden node at level 1, and then the input sample X should be regarded to be classified to the nodes at level 2.

Level k . Let the k th layer consist of n_k nodes, $\text{node}_1^k, \text{node}_2^k, \dots, \text{node}_{n_k}^k$, and let there be \tilde{n}_{k-1} hidden nodes at the level $k-1$. Let $w_i^k = (w_{i,1}^k, w_{i,2}^k, \dots, w_{i,\tilde{n}_{k-1}}^k)$ be the weight vector for node_i^k at the level k . Notably, only the hidden nodes at the level $k-1$ should be connected with the nodes at the level k . Calculate the values of Net_i^k , $1 \leq i \leq n_k$, at the level k as

$$\text{Net}_i^k = \left[\sum_{j=1}^{\tilde{n}_{k-1}} w_{i,j}^k F(\text{Net}_j^{k-1}) \right] - \theta_i^k \quad \text{for } 1 \leq i \leq n_k. \tag{4}$$

Let $\text{node}_{c_k}^k$ have the largest activation value for X at the level k . Then,

$$c_k = \arg \max \{F(\text{Net}_1^k), F(\text{Net}_2^k), \dots, F(\text{Net}_{n_k}^k)\}. \quad (5)$$

Then, X is regarded to be classified to $\text{node}_{c_k}^k$ when $\text{node}_{c_k}^k$ has the largest activation value for X . If $\text{node}_{c_k}^k$ is an output node at level k , the input sample X is recognized as the class represented by $\text{node}_{c_k}^k$. Otherwise, $\text{node}_{c_k}^k$ represents a hidden node at level k , and then the input sample X continues to travel across the nodes at level $k + 1$ until it reaches the output nodes in VDO-DMLNN.

3. VDO-DMLNN

3.1. Basic definition of VDO-DMLNN

Our proposed design method should consider both the classification error rate and computing complexity of VDO-DMLNN. Then only the designed VDO-DMLNN tends to be optimal. Before the design method of VDO-DMLNN is described, both the classification error rate and computing complexity of VDO-DMLNN should be defined as follows.

The computing complexity of VDO-DMLNN is defined as follows. Let there be M training samples, X_1, X_2, \dots, X_M . Let the VDO-DMLNN, λ^r , consist of r layers and let there be n_k nodes, $\text{node}_1^k, \text{node}_2^k, \dots, \text{node}_{n_k}^k$, at level k . Notably, the value of n_0 denotes the number of nodes at the input layer (level 0). Thus, if an input sample arrives at the node node_i^k from the input layer to level k , the total number of weights required to be calculated for the node node_i^k , $\text{Cal}(\text{node}_i^k)$, is defined as

$$\text{Cal}(\text{node}_i^k) = \sum_{i=1}^k n_{i-1} n_i, \quad (6)$$

in the worst case. Let there be $N(\text{node}_i^k)$ training samples that arrived at the node node_i^k . Then, the probability of the training samples that arrived to the node node_i^k , $P(\text{node}_i^k)$, is defined as

$$P(\text{node}_i^k) = \frac{N(\text{node}_i^k)}{M}. \quad (7)$$

Therefore, the computing complexity of λ , $\text{CC}(\lambda)$, is defined as

$$\text{CC}(\lambda^r) = \sum_{k=1}^r \sum_{i=1}^{n_k} P(\text{node}_i^k) \text{Cal}(\text{node}_i^k). \quad (8)$$

The classification error rate of λ^r is described as follows. Let $S(\text{node}_i^k) = \{X_z | 1 \leq z \leq N(\text{node}_i^k)\}$ be the set that collects the training samples that arrived at the node node_i^k , and let $\text{class}(X_z)$ denote the class of the sample X_z . The represented class of the node node_i^k , $\text{class}(\text{node}_i^k)$, is set to be the same as the largest number of input samples of the same class in the node node_i^k . Then, the classification

error rate of the node node_i^k , $\text{Cer}(\text{node}_i^k)$, is defined as

$$\text{Cer}(\text{node}_i^k) = \frac{\sum_{X_z \in S(\text{node}_i^k)} \rho_z}{N(\text{node}_i^k)}, \quad (9)$$

where

$$\rho_z = \begin{cases} 1 & \text{if class}(X_z) \neq \text{class}(\text{node}_i^k), \\ 0 & \text{if class}(X_z) = \text{class}(\text{node}_i^k). \end{cases} \quad (10)$$

Finally, the classification error rate of λ^r , $\text{ER}(\lambda^r)$, is defined as

$$\text{ER}(\lambda^r) = \sum_{k=1}^r \sum_{i=1}^{n_k} P(\text{node}_i^k) \text{Cer}(\text{node}_i^k). \quad (11)$$

The design method of VDO-DMLNN, λ^r , is based on both the computing complexity, $\text{CC}(\lambda^r)$, and classification error rate, $\text{ER}(\lambda^r)$.

The following describes how to design the VDO-DMLNN with depth r , λ^r , after that with a depth $r-1$, λ^{r-1} , has been designed. Although λ^r has lower classification error rate than λ^{r-1} by increasing the depth, the computing complexity of λ^r is larger than that of λ^{r-1} . The slope of the classification error rate and computing complexity between λ^r and λ^{r-1} is defined as

$$\delta(\lambda^r, \lambda^{r-1}) = \frac{\Delta \text{ER}}{\Delta \text{CC}} = \frac{\text{ER}(\lambda^{r-1}) - \text{ER}(\lambda^r)}{\text{CC}(\lambda^r) - \text{CC}(\lambda^{r-1})}. \quad (12)$$

In designing λ^r , the value of ΔER is emphasized to be as large as possible and the value of ΔCC must be as small as possible in Eq. (12). Therefore, the value of $\delta(\lambda^r, \lambda^{r-1})$ defined in Eq. (12) is emphasized to be as large as possible when λ^r is designed. Then only the designed λ^r tends to be optimal.

3.2. SOFM

Let the SOFM consist of n neurons, $\text{node}_1, \text{node}_2, \dots, \text{node}_n$, and let $w_i = (w_{i,1}, w_{i,2}, \dots, w_{i,d})$ be the initial weight vector for node_i . The iterative SOFM training algorithm is described in Algorithm 1.

These n new weight vectors, $\hat{w}_i = (\hat{w}_{i,1}, \hat{w}_{i,2}, \dots, \hat{w}_{i,d})$, $1 \leq i \leq n$, generated by the SOFM algorithm, can be regarded as the new weight vectors for node_i , $1 \leq i \leq n$. The goal of SOFM is to update these initial weight vectors generated from GA-SOFM to the new weight vectors by classifying the samples in the training dataset. The GA-SOFM is described in Sec. 3.3.

3.3. Design of GA-SOFM

The VDO-DMLNN is built by GA-SOFM in a layer-by-layer manner. The GA-SOFM is designed based on both the GA and SOFM. The GA-SOFM automatically generates the proper number of nodes, the active threshold and the initial weights

Algorithm 1. SOFM

Input: The training dataset. The n neurons, $\text{node}_1, \text{node}_2, \dots, \text{node}_n$, and n initial weight vectors, $w_i = (w_{i,1}, w_{i,2}, \dots, w_{i,d}), 1 \leq i \leq n$.

Output: The n new weight vectors, $\hat{w}_i = (\hat{w}_{i,1}, \hat{w}_{i,2}, \dots, \hat{w}_{i,d}), 1 \leq i \leq n$, for these n neurons.

Step 1. Set iteration $t = 1$.

Step 2. Select a training sample, $X(t)$, from the training dataset. Compute the distances between $X(t)$ and all these n weights in the SOFM. The winning neuron, node_c , is defined as

$$c = \operatorname{argmin} \|X(t) - w_i\| \quad \text{for } 1 \leq i \leq n. \quad (13)$$

Step 3. Update the winner neuron and its neighbor neurons to move its weights toward the training sample, $X(t)$. The weight updating rule is given as

$$w_i(t+1) = w_i(t) + \eta(t)h(t)(X(t) - w_i(t)), \quad (14)$$

where $\eta(t)$ is the learning rate which decreases monotonically with iteration t . Also, $h(t)$ is defined as

$$\eta(t) = \eta_0 \exp\left(-\frac{t}{\tau}\right), \quad (15)$$

where τ means a constant time which is set equal to the maximum number of iterations. The neighborhood function, $h(t)$, defines the closeness of a neighboring neuron to the winning neuron. The neighborhood function, $h(t)$, also decreases gradually during the learning process, and is defined as

$$h(t) = \exp\left(\frac{-D^2}{2\delta^2(t)}\right), \quad (16)$$

where D is the distance between input and the winning neuron, and $\delta(t)$ denotes the radius of the neighborhood at time t . Also, $\delta(t)$ decreases with the iterations as

$$\delta(t) = \delta_0 \exp\left(-\frac{t}{\tau} \log(\delta_0)\right), \quad (17)$$

where δ_0 is the initial width.

Step 4. If the maximum iterations are reached, these n weight vectors can be renamed as $\hat{w}_i = (\hat{w}_{i,1}, \hat{w}_{i,2}, \dots, \hat{w}_{i,d}), 1 \leq i \leq n$, that can be set as the output of SOFM. Otherwise, set $t = t + 1$ and go to step 2.

for each node in VDO-DMLNN, and then these initial weights can be further corrected by using the training rule of SOFM. Figure 3 shows the flowchart of GA-SOFM. In Fig. 3, the GA-SOFM consists of many iterations, and each iteration consists of three phases, reproduction, crossover and mutation. The GA-SOFM

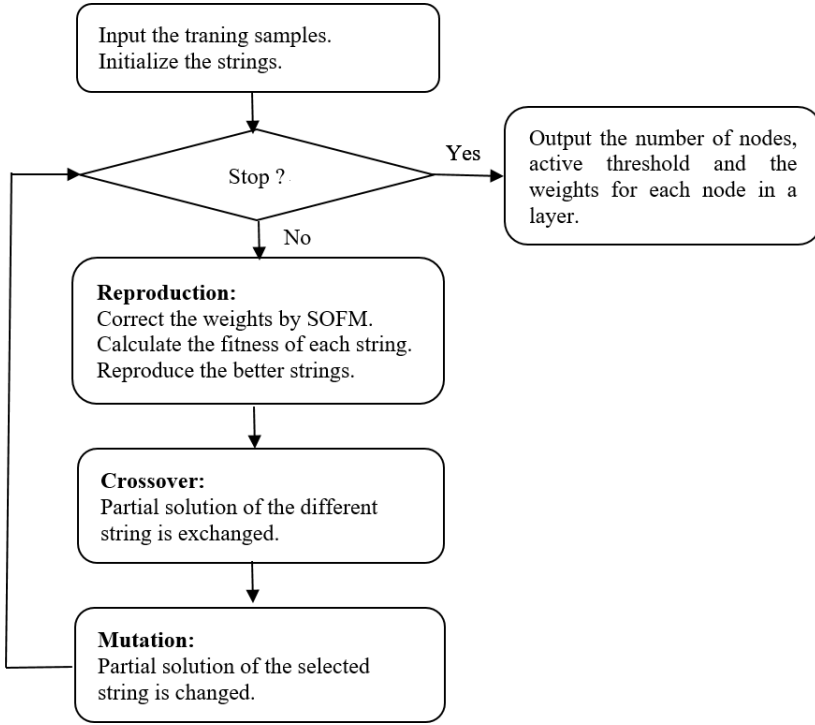


Fig. 3. The flowchart of GA-SOFM.

designs VDO-DMLNN based on both its classification error rate and computing complexity, thus the value of $\delta()$ Eq. (12) must be as large as possible during the design of each layer in VDO-DMLNN. Then only, the designed VDO-DMLNN tends to be optimal.

Assume that a VDO-DMLNN, λ^{k-1} , with depth $k - 1$ has been designed. Let there be \tilde{n}_{k-1} hidden nodes at level $k - 1$ in λ^{k-1} . The following describes how to design λ^k with depth k in GA-SOFM. The GA-SOFM consists of initialization step and three phases, reproduction, crossover and mutation, which are described as follows.

In the initialization step of GA-SOFM, a population of σ strings, R_z , $1 \leq z \leq \sigma$, is randomly generated. Let there be TOTAL training samples that are classified to these \tilde{n}_{k-1} hidden nodes at the level $k - 1$. The value of TOTAL is defined as

$$\text{TOTAL} = \sum_{i=1}^{\tilde{n}_{k-1}} N(\text{node}_i^{k-1}). \quad (18)$$

Then, the length of each string is randomly generated to be smaller than or equal to the value of TOTAL because the value of TOTAL is the maximal number of nodes generated by GA-SOFM at level k . Let the string R_z represent r_z nodes

($r_z \leq \text{TOTAL}$) at level k . Then, the string, R_z , is represented as follows:

$$R_z = (\text{node}_z(1), \text{node}_z(2), \dots, \text{node}_z(r_z)),$$

where $\text{node}_z(i) = (\theta_i, w_i^k) = (\theta_i, w_{i,1}^k, w_{i,2}^k, \dots, w_{i,\tilde{n}_{k-1}}^k)$, $1 \leq i \leq r_z$. The value of θ_i denotes the activation threshold, and w_i^k denotes the initial weight vector for the node $\text{node}_z(i)$. For example, let there be three hidden nodes ($\tilde{n}_{k-1} = 3$) at the level $k - 1$, and let the length of string R_x be 4 ($r_x = 4$). Then, the string R_x is represented to generate four nodes ($r_x = 4$) at level k as follows:

$$\begin{aligned} R_x &= (\text{node}_x(1), \text{node}_x(2), \text{node}_x(3), \text{node}_x(4)) \\ &= ((\theta_1, w_1^k), (\theta_2, w_2^k), (\theta_3, w_3^k), (\theta_4, w_4^k)) \\ &= ((\theta_1, w_{1,1}^k, w_{1,2}^k, w_{1,3}^k), (\theta_2, w_{2,1}^k, w_{2,2}^k, w_{2,3}^k), (\theta_3, w_{3,1}^k, w_{3,2}^k, w_{3,3}^k), \\ &\quad (\theta_4, w_{4,1}^k, w_{4,2}^k, w_{4,3}^k)). \end{aligned}$$

In the reproduction phase, the SOFM is used to update these initial weight vectors, w_i^k , $1 \leq i \leq r_z$, generated by the string R_z to the new weight vectors, \hat{w}_i^k , $1 \leq i \leq r_z$, by classifying these TOTAL training samples to these r_z nodes. That is, each training sample is selected to find the winner node as shown in Eq. (13), and then the weight vectors of the winner node and its neighbors can be updated as shown in Eq. (14). Furthermore, the winner node of the training sample can be regarded as the node to which the training sample is classified. Thus, all of the training samples can be classified to these r_z nodes generated by the string R_z after the SOFM algorithm is finished. Calculate $\text{CC}(\lambda^k)$ and $\text{ER}(\lambda^k)$ as shown in Eqs. (8) and (11), respectively. Then, the fitness for string R_z can be defined as

$$\text{Fitness}(R_z) = \delta(\lambda^k, \lambda^{k-1}), \quad (19)$$

where $\delta(\lambda^k, \lambda^{k-1})$ is defined in Eq. (12). The string with high fitness provides a better solution in GA-SOFM. Thus, after the fitness of each string in the population is calculated, the string with high fitness has a large probability to be chosen as the population in the next generation using a roulette wheel with slots whose sizes are the determined fitnesses. However, the total number of strings, σ , is fixed in the next generation.

In the crossover phase, many pairs of strings are chosen from the population with a probability to undergo a crossover operator. Then, two pieces of a pair of strings are randomly selected to be interchanged. If the number of nodes contained in the string is larger than the value of m after the crossover operation is finished, two pieces of strings should be randomly generated again. After the crossover operator is applied to the selected pairs of strings, R_x and R_y , two new strings, \hat{R}_x and \hat{R}_y , replace the strings, R_x and R_y , in the population. The significance of the crossover phase is that it exchanges the nodes including the active threshold of each node and the connected weights between different strings, to yield various neural networks. For example, let

$$R_x = (\text{node}_x(1), \text{node}_x(2), \text{node}_x(3))$$

and

$$R_y = (\text{node}_y(1), \text{node}_y(2), \text{node}_y(3), \text{node}_y(4), \text{node}_y(5))$$

be two strings that are chosen to perform crossover. Let the two values, $a = 2$ and $b = 3$, be randomly generated for the strings, R_x and R_y , respectively. After the crossover phase is finished, two new strings,

$$\acute{R}_x = (\text{node}_y(1), \text{node}_y(2), \text{node}_y(3), \text{node}_x(3))$$

and

$$\acute{R}_y = (\text{node}_x(1), \text{node}_x(2), \text{node}_y(4), \text{node}_y(5)),$$

are generated to replace the original two strings, R_x and R_y , in the population.

In the mutation phase, the active thresholds and weights of the strings are randomly chosen with a probability. Each chosen weight, $\hat{w}_{i,j}$, and active threshold, θ_i , are assigned a random value, respectively. Let φ_1 and φ_2 be two random values, $0 < \varphi_1 < 1$ and $0 < \varphi_2 < 1$. Then,

$$\hat{w}_{i,j}^k = \hat{w}_{i,j}^k \pm \varphi_1 \hat{w}_{i,j}^k, \quad (20)$$

$$\theta_i = \theta_i \pm \varphi_2 \theta_i. \quad (21)$$

After the mutation phase, the new string can be obtained, and replace the original string.

The user may specify the number of generations for running the GA-SOFM; the best solution can then be reserved to design the nodes at level k ; that is, the best answers must be retained in each generation and the best answers will be updated until the end of the program. The setting of the generation number must be sufficient for the GA-SOFM to converge and produce the best answer.

3.4. Design of VDO-DMLNN

The Design_VDO-DMLNN algorithm is proposed to design the VDO-DMLNN with classification error rate constraint. That is, the GA-SOFM is applied to design VDO-DMLNN until the classification error rate of VDO-DMLNN is lower than a constraint given by the user. The Design_VDO-DMLNN algorithm is described in Algorithm 2.

From step 2.3 of Algorithm 2, the Design_VDO-DMLNN algorithm only stops when all the nodes are designed as the output nodes and the classification error rate of each output node is smaller than the constraint, ε .

4. Experiments

4.1. Performance of VDO-DMLNN

In the experiments, eight datasets, including seven datasets extracted from the UCI Machine Learning Repository⁴ and one speech dataset extracted from the ISOLET

Algorithm 2. Design_VDO-DMLNN

Input: The classification error rate constraint ε and M training samples, $X_i \in R^n$, $i = 1, 2, \dots, M$.

Output: The VDO-DMLNN, λ .

Step 1. Let L be the level number. Generate n input nodes as the input layer ($L = 0$) in λ , since each training sample is an n -dimensional vector. Let the set, S , consist of M training samples.

Step 2. While the set S is not empty ($S \neq \phi$):

Step 2.1. The GA-SOFM is applied to the training samples in S . Let the best string in GA-SOFM generate u nodes, $\text{node}_1, \text{node}_2, \dots, \text{node}_u$ at level $L + 1$, the active threshold of each node at level $L + 1$ and the link weights between the nodes in levels L and $L + 1$.

Step 2.2. Classify all the training samples in S to these u nodes, $\text{node}_1, \text{node}_2, \dots, \text{node}_u$ at the level $L + 1$ as in Eq. (5).

Step 2.3. For each node, node_i , at level $L + 1$, perform the following:

Step 2.2.1. Calculate the value of $\text{Cer}(\text{node}_i)$ as shown in Eq. (8).

Step 2.2.2. If $\text{Cer}(\text{node}_i) \leq \varepsilon$, then node_i is designed as the output node in λ .

Step 2.2.3. If $\text{Cer}(\text{node}_i) > \varepsilon$, then node_i is designed as a hidden node in λ .

Step 2.4. Let the set, S , collect all the training samples contained in the hidden nodes at level $L + 1$. Notably, if all nodes are determined as the output nodes at level $L + 1$, the set S is set to be empty ($S = \phi$).

Step 2.5. Set $L = L + 1$.

Step 3. Output the VDO-DMLNN, λ , with depth = L .

database, are used to test the performance of our proposed VDO-DMLNN. Table 2 lists the features of these eight datasets. In the speech dataset, the ISOLET database using the 26 letters of the English alphabet was used in the isolated word recognition test. The speech dataset consisted of 6240 utterances from 120 speakers. Each utterance was sampled at 16 kHz with a 16-bit resolution. A Hamming window of 20 ms with 50% overlap was used to process each utterance further by Fast Fourier Transform (FFT). Each utterance was divided into 15 Hamming windows, each represented by 32 FFT coefficients. That is, each utterance consisted of 480 features.

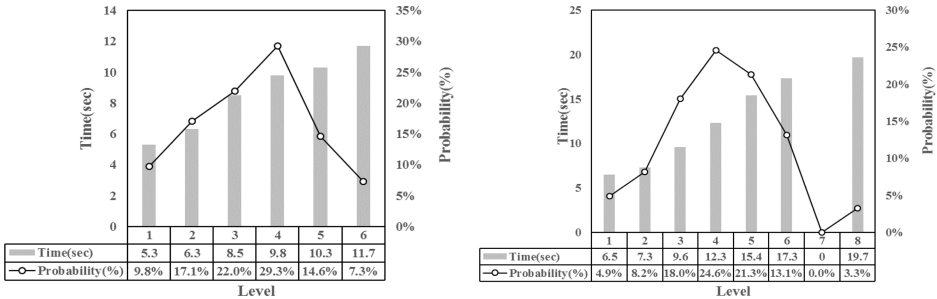
In the design of VDO-DMLNN, the proposed GA-SOFM is used to design the nodes at each layer, and then it is determined whether each node is an output node

Table 2. Description of eight datasets.

Dataset	No. of Features	Instances	No. of Classes
Abalone	8	4177	29
Pendigits	16	10 992	10
Letter	16	20 000	26
Hepatitis	19	155	2
Pima	8	768	2
Glass	9	214	6
Wine	13	178	3
Speech	480	6240	26

or a hidden node according to the classification error rate of the node. The main goal of the output nodes at low levels is to be able to recognize the input samples early in VDO-DMLNN, so that the overall average computing time of VDO-DMLNN can be reduced. In GA-SOFM, the population size is set to 200, the length of each string is set to the number of training samples at level k and the crossover and mutation probabilities are set to 80% and 5%, respectively. The GA-SOFM is run over 500 generations, and the best solution obtained from these 500 generations is retained. In Algorithm 2, the classification error rate constraint, $\varepsilon = 5\%$, is applied to four datasets in designing the VDO-DMLNN.

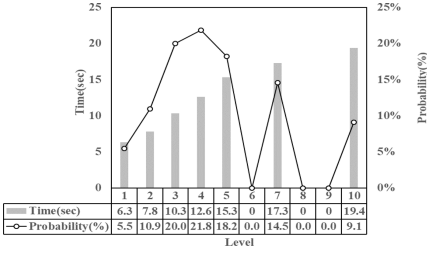
Figure 4 shows the values of average probability and average computing time of the testing sample falling at the output node of each level of VDO-DMLNN on the eight datasets. From Fig. 4, most of the testing samples are identified at the output nodes that are less than one-half the depth of VDO-DMLNN, and only a small number of testing samples must be output at high level in VDO-DMLNN. Notably, not every level of VDO-DMLNN contains output nodes. For example, in Fig. 4(c), the sixth, eighth and ninth levels have no output node in VDO-DMLNN on the Letter dataset, and thus neither the probability nor the computing time



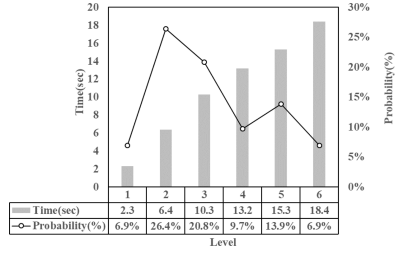
(a) Abalone dataset.

(b) Pendigits dataset.

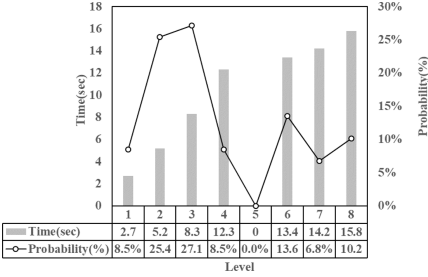
Fig. 4. The average probabilities and average computing times of the testing sample falling at the output node of each level of VDO-DMLNN on the eight datasets.



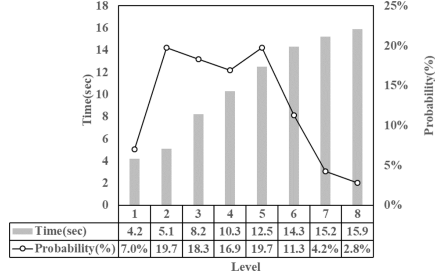
(c) Letter dataset.



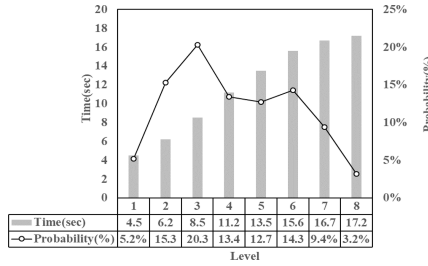
(d) Hepatitis dataset.



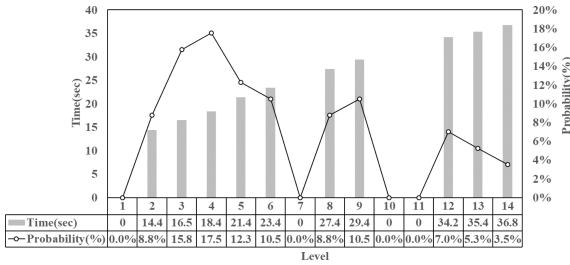
(e) Pima dataset.



(f) Glass dataset.



(g) Wine dataset.



(h) Speech dataset.

Fig. 4. (Continued).

of the output nodes can be calculated. Figure 4 is summarized in Table 3. From Table 3, for example, the depth of VDO-DMLNN is 10 on the Letter dataset,

Table 3. The performance of VDO-DMLNN for eight datasets.

Dataset	Depth	Classification Error Rate	Low-Level Range		Medium-Level Range		High-Level Range	
			Ave. Probability (Level Range)	Ave. Computing Time (s) (Level Range)	Ave. Probability (Level Range)	Ave. Computing Time (s) (Level Range)	Ave. Probability (Level Range)	Ave. Computing Time (s) (Level Range)
Abalone	6	2.8%	26.8% (levels 1,2)	5.8 (levels 1,2)	51.2% (levels 3,4)	8.3 (levels 3,4)	22% (levels 5,6)	11 (levels 5,6)
	8	3.5%	31.1% (levels 1-3)	7.8 (levels 1-3)	52.5% (levels 4,5)	12.3 (levels 4,5)	16.4% (levels 6-8)	18.5 (levels 6-8)
	10	4.1%	36.4% (levels 1-3)	8.1 (levels 1-3)	54.5% (levels 4-7)	13.1 (levels 4-7)	9.1% (levels 8-10)	19.4 (levels 8-10)
Hepatitis	6	4.8%	33.30% (levels 1,2)	4.4 (levels 1,2)	45.90% (levels 3,4)	10.8 (levels 3,4)	20.80% (levels 5,6)	16.8 (levels 5,6)
	8	4.7%	33.90% (levels 1-3)	5.4 (levels 1-3)	52.50% (levels 4,5)	9.1 (levels 4,5)	13.60% (levels 6-8)	14.1 (levels 6-8)
	8	4.5%	45.10% (levels 1-3)	5.8 (levels 1-3)	36.60% (levels 4,5)	10.3 (levels 4,5)	18.30% (levels 6-8)	15.13 (levels 6-8)
Wine	9	4.3%	40.80% (levels 1-3)	6.4 (levels 1-3)	32.30% (levels 4,5)	12.2 (levels 4,5)	26.90% (levels 6-8)	16.5 (levels 6-8)
Speech	14	4.8%	42.1% (levels 1-4)	16.4 (levels 1-4)	42.10% (levels 5-10)	25.2 (levels 5-10)	15.8% (levels 11-14)	35.5 (levels 11-14)

and the probabilities of the testing sample falling within the low-level range (levels 1–3) and high-level range (levels 8–10) are 36.4% and 9.1%, respectively. This means that 36.4% of the testing samples can be recognized early by the VDO-DMLNN, and they only take an average of 8.1s to be recognized. Also, only a small percentage, 9.1%, of the testing samples at the high-level range is recognized in an average of 19.4s in the VDO-DMLNN. Therefore, the computing time of each sample is variant, and many cases can be rapidly recognized in the VDO-DMLNN.

To further verify the performance of VDO-DMLNN, the multi-layer neural network, MLNN(BP), is compared with the proposed VDO-DMLNN. In the MLNN(BP), the backpropagation (BP) method is used to design the link weights. To make a fair comparison, the VDO-DMLNN is designed first because the GA-SOFM can automatically search for the proper number of nodes in each layer, and then the MLNN(BP) is designed with the same depth and number of hidden nodes in each layer as VDO-DMLNN. Notably, in the MLNN(BP), all the output nodes are at the final layer. In the experiments, four classification error rate constraints, $\lambda_{\varepsilon_1} = 3\%$, $\lambda_{\varepsilon_2} = 7\%$, $\lambda_{\varepsilon_3} = 11\%$ and $\lambda_{\varepsilon_4} = 15\%$, are used to design the VDO-DMLNN, respectively. Figure 5 shows the comparison of both VDO-DMLNN and MLNN(BP) on the eight datasets. From Fig. 5, the overall classification error rates of both VDO-DMLNN and MLNN(BP) are similar, but the average computing time of VDO-DMLNN is less than that of MLNN(BP) when they have the same classification error rate. The reason for this is that most input samples can be identified early by the output nodes at low levels in VDO-DMLNN, rather than at the last layer, as in MLNN(BP). Also, another benefit of VDO-DMLNN relative to MLNN(BP) is that BP usually searches for a local solution for designing the weights instead of searching for the global solutions like GA-SOFM.

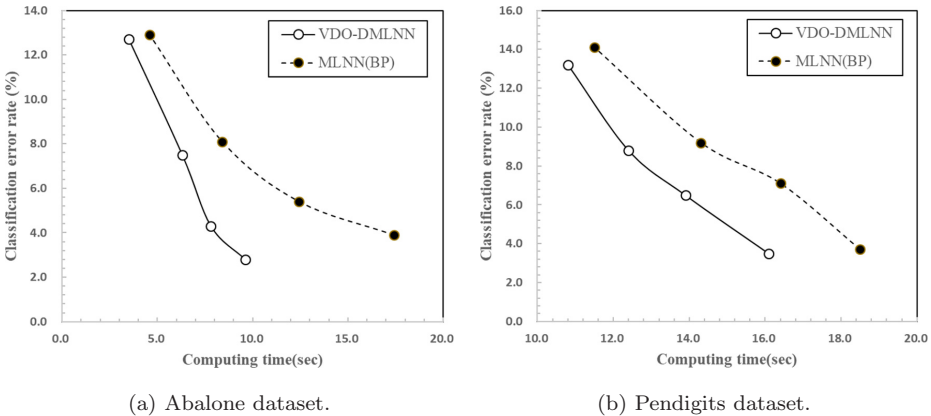
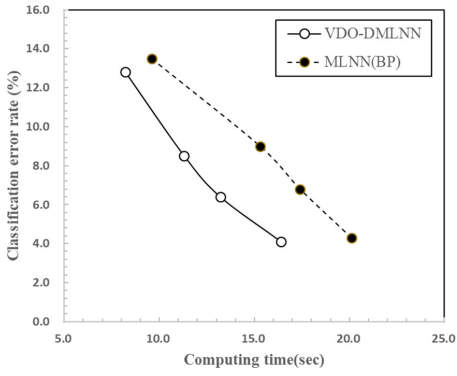
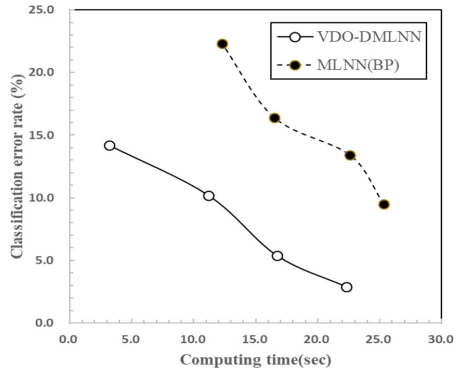


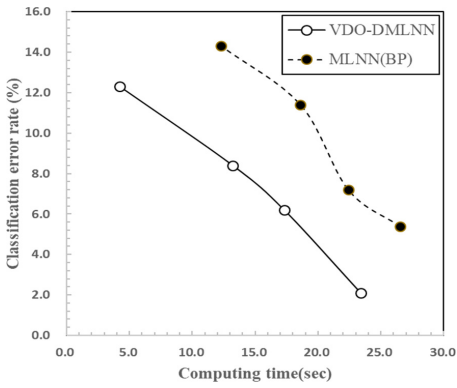
Fig. 5. Comparison of VDO-DMLNN and MLNN(BP) on the eight datasets.



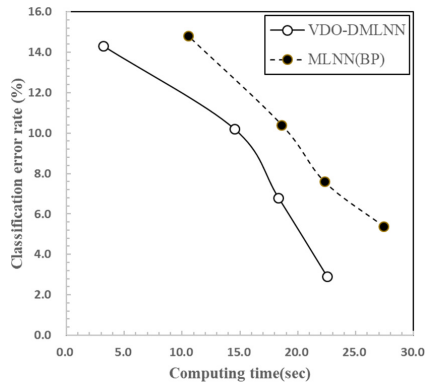
(c) Letter dataset.



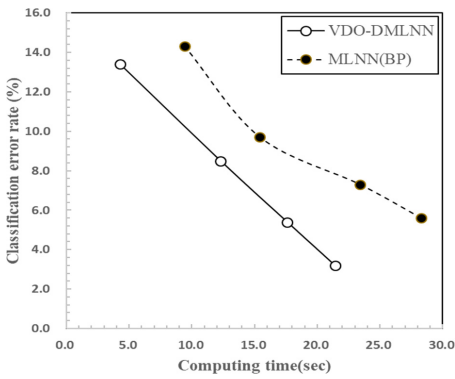
(d) Hepatitis dataset.



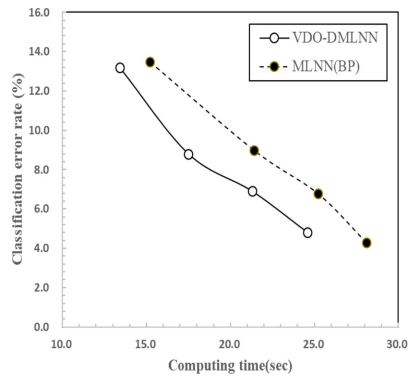
(e) Pima dataset.



(f) Glass dataset.



(g) Wine dataset.



(h) Speech dataset.

Fig. 5. (Continued).

4.2. Comparison of VDO-DMLNN and the state-of-the-art methods

First, the VDO-DMLNN is compared with the nontraditional NN model, morphological perceptron with dendritic processing (MPDP).³³ In Ref. 33, the differential evolutionary method (DEM)² was used to evaluate the weights of MPDP, and then MPDP using DEM obtained a lower classification error rate than those obtained by using other training algorithms. To compare the performance of VDO-DMLNN and MPDP, the same database, UCI Machine Learning Repository, is used to test the VDO-DMLNN, and then the performance of VDO-DMLNN is compared with that of MPDP using DEM published in Ref. 33. In Table 4, VDO-DMLNN outperforms MPDP using DEM because the former can be automatically expanded or, in other words, the depth and width widen during the training phase until the classification error rate of VDO-DMLNN falls below a threshold ($\lambda = 5\%$).

Next, the VDO-DMLNN is compared with other state-of-the-art deep neural networks (DNNs), VGG,³⁰ GoogLeNet,³⁵ ResNet,¹² PReLU-Net¹¹ and BN-Inception,¹⁴ by using the ImageNet dataset.⁹ The ImageNet dataset includes images of 1000 classes, and is split into three sets: training (1.3 M images), validation (50k images) and testing (100k images with held-out class labels). This study evaluated both top-1 and top-5 error rates, and compared them with the results reported in Ref. 1. From Table 5, the VDO-DMLNN outperforms the 50-layer ResNet, but performs worse than 101-layer and 152-layer ResNets. Although it is difficult to compare the effectiveness of VDO-DMLNN with other DNNs using different image features, the VDO-DMLNN still outperforms many previous models in Table 5. Furthermore, all of the DNNs emphasize reducing the classification error rate as much as possible, so their computational time complexity is increased with greater depth. However, the advantage of VDO-DMLNN is that it is possible to recognize the results earlier than the layer-by-layer DNNs in many cases. Also, the design of VDO-DMLNN considers both the classification error rate and computing time while other DNNs only consider how to reduce the classification error rate.

Table 4. Comparisons of VDO-DMLNN and MPDP using DEM.

Dataset	MPDP Using DEM		VDO-DMLNN	
	Classification Error Rate (Training Dataset) (%)	Classification Error Rate (Testing Dataset) (%)	Classification Error Rate (Training Dataset) (%)	Classification Error Rate (Testing Dataset) (%)
Abalone	77.1	78.2	2.1	2.8
Hepatitis	9.4	33.3	3.4	4.8
Pima	23.8	23.5	3.6	4.7
Glass	4.7	13.6	3.3	4.5
Wine	42.1	40	3.8	4.3


Table 5. The error rates of methods on ImageNet dataset (%).

Method	Top-1 Error	Top-5 Error
VGG ²⁴ (ILSVRC'14)	—	8.43
GoogLeNet ¹ (ILSVRC'14)	—	7.89
PReLU-net ²³	21.59	5.71
BN-inception ⁴²	21.99	5.81
ResNet-50 ³	20.74	5.25
ResNet-101 ³	19.87	4.60
ResNet-152 ³	19.38	4.49
VDO-DMLNN (depth = 26)	20.29	5.18

5. Conclusion

In this study, the VDO-DMLNN based on GA is proposed. The VDO-DMLNN is similar to a multi-layer NN, with the exception that the output nodes can be at different levels rather than at the last layer. Thus, the input sample can be recognized earlier, while it is classified to the output nodes at low level of VDO-DMLNN. The proposed GA automatically searches for the proper number of nodes for each layer and the link weights between adjacent layers, based on both the classification error rate and computing time of the VDO-DMLNN. Thus, the VDO-DMLNN tends to be optimal. In addition, the user does not need to predefine the depth of VDO-DMLNN before designing it. The VDO-DMLNN is designed from the top-down, layer by layer, until the classification error rate of each output node is less than a constraint given by the user. A method for determining the best VDO-DMLNN within the classification error rate range is also proposed. In the experiments conducted, the VDO-DMLNN outperformed most of the compared state-of-the-art DNNs.

ORCID

Shiueng-Bien Yang  <https://orcid.org/0009-0000-7622-6127>

Ting-Wen Liang  <https://orcid.org/0000-0003-2094-4618>

References

1. R. Ahmed *et al.*, Deep neural network-based contextual recognition of Arabic handwritten scripts, *Entropy* **23** (2021) 340.
2. F. Arce *et al.*, Dendrite morphological neural networks trained by differential evolution, in *Proc. 2016 IEEE Symp. Series on Computational Intelligence (SSCI)*, Vol. 1 (IEEE, 2016), pp. 1–8.
3. A. Ashiquzzaman *et al.*, An efficient recognition method for handwritten Arabic numerals using CNN with data augmentation and dropout, in *Data Management, Analytics and Innovation, Advances in Intelligent Systems and Computing*, Vol. 808 (Springer, Singapore, 2019), pp. 299–309.
4. C. Blake and C. J. Merz, UCI Machine Learning Repository (1998), Department of Information and Computer Science, University of California, <http://www.ics.uci.edu/mllearn/MLRepository.html>.

5. J. D. Bodapati *et al.*, Composite deep neural network with gated-attention mechanism for diabetic retinopathy severity classification, *J. Ambient Intell. Human Comput.* **12** (2021) 9825–9839, doi:10.1007/s12652-020-02727-z.
6. S. Chatterjee, S. Bandopadhyay and D. Machuca, Ore grade prediction using a genetic algorithm and clustering based ensemble neural network model, *Math. Geosci.* **42**(3) (2010) 309–326.
7. T. E. Chen *et al.*, S1 and S2 heart sound recognition using deep neural networks, *IEEE Trans. Biomed. Eng.* **64**(2) (2017) 372–380.
8. D. Ciresan *et al.*, Multi-column deep neural networks for image classification, in *Proc. 2012 IEEE Conf. Computer Vision and Pattern Recognition* (Institute of Electrical and Electronics Engineers, 2012), pp. 3642–3649.
9. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A large-scale hierarchical image database, in *Proc. 2009 IEEE Conf. Computer Vision and Pattern Recognition* (Curran Associates, Inc., 2009).
10. C. Dong, C. C. Loy, K. He and X. Tang, Image super-resolution using deep convolutional networks, *IEEE Trans. Pattern Anal. Mach. Intell.* **38**(2) (2016) 295–307.
11. K. He, X. Zhang, S. Ren and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, preprint, arXiv:1502.01852 [cs.CV].
12. K. He, X. Zhang, S. Ren and J. Sun, Deep residual learning for image recognition, preprint, arXiv:1512.03386 [cs.CV].
13. G. Hinton *et al.*, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Process. Mag.* **29** (2012) 82–97.
14. S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, preprint, arXiv:1502.03167 [cs.LG].
15. S. Kabir, S. Patidar, X. Xia, Q. Liang, J. Neal and G. Pender, A deep convolutional neural network model for rapid prediction of fluvial flood inundation, *J. Hydrol.* **590** (2020) 125481, doi:10.1016/j.jhydrol.2020.125481.
16. M. Khayyat, L. Lam and C. Y. Suen, Learning-based word spotting system for Arabic handwritten documents, *Pattern Recognit.* **47**(3) (2014) 1021–1030.
17. H. Larochelle *et al.*, Exploring strategies for training deep neural networks, *J. Mach. Learn. Res.* **10** (2009) 1–40.
18. W. P. Lee, S. Hasan, S. M. Shamsuddin and N. Lopes, GPUMLib: Deep learning SOM library for surface reconstruction, *Int. J. Adv. Soft Comput. Appl.* **9**(2) (2017) 1–16.
19. X. Lu *et al.*, Speech enhancement based on deep denoising autoencoder, in *Proc. INTERSPEECH 2013* (International Speech Communication Association, 2013), pp. 436–440.
20. B. Luca, J. H. Joao, V. Jack, T. Philip and V. Andrea, Learning feed-forward one-shot learners, in *Advances in Neural Information Processing Systems*, Vol. 29 (Curran Associates, Inc., 2016), pp. 523–531.
21. H. Mahmoudabadi, M. Izadi and M. B. Menhaj, A hybrid method for grade estimation using genetic algorithm and neural networks, *Comput. Geosci.* **13** (2009) 91–101.
22. C. Micheloni, A. Rani, S. Kumarb and G. L. Foresti, A balanced neural tree for pattern classification, *Neural Netw.* **27** (2012) 81–90.
23. M. E. Mustafa *et al.*, A deep learning approach for handwritten Arabic names recognition, *Int. J. Adv. Comput. Sci. Appl.* **11** (2020) 678–682.
24. X. Rao *et al.*, Distracted driving recognition method based on deep convolutional neural network, *J. Ambient Intell. Humaniz. Comput.* **12** (2021) 193–200.
25. T. Reza and K. Mohammadreza, A method for handwritten word spotting based on particle swarm optimization and multi-layer perceptron, *IET Softw.* **12**(2) (2018) 152–159.

26. A.-B. M. Salem, M. M. Syiam and A. F. Ayad, Improving self-organizing feature map (SOFM) training algorithm using K-means initialization, in *Proc. 5th Int. Conf. Enterprise Information Systems*, Vol. 2 (International Speech Communication Association, 2003), pp. 399–405.
 27. B. Samanta, S. Bandopadhyay and R. Ganguli, Data segmentation and genetic algorithms for sparse data division in Nome placer gold grade estimation using neural network and geostatistics, *Explor. Mining Geol.* **11**(1) (2004) 69–76.
 28. S. Z. Selim and M. A. Ismail, K-means-type algorithm: Generalized convergence theorem and characterization of local optimality, *IEEE Trans. Pattern Anal. Mach. Intell.* **6** (1984) 81–87.
 29. P. Y. Simard *et al.*, Best practices for convolutional neural networks applied to visual document analysis, in *Proc. Seventh Int. Conf. Document Analysis and Recognition* (IEEE Computer Society, 2003), pp. 958–963.
 30. K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, preprint, arXiv:1409.1556 [cs.CV].
 31. G. Singh and A. Kaur, Comparative analysis of K-means and Kohonen-SOM data mining algorithms based on student behaviors in sharing information on Facebook, *Int. J. Eng. Comput. Sci.* **6**(4) (2017) 20990–20993.
 32. S. M. Siniscalchi, An artificial neural network approach to automatic speech processing, *Neurocomputing* **140** (2014) 326–338.
 33. H. Sossa *et al.*, Morphological neural networks with dendritic processing for pattern classification, in *Advanced Topics on Computer Vision, Control and Robotics in Mechatronics*, eds. O. Vergara Villegas, M. Nandayapa and I. Soto (Springer, Cham, 2018), pp. 27–47, doi:10.1007/978-3-319-77770-2_2.
 34. S. Sudholt and G. A. Fink, A deep convolutional neural network for word spotting in handwritten documents, in *Proc. 15th Int. Conf. Frontiers in Handwriting Recognition (ICFHR)* (IEEE, 2016), pp. 277–282.
 35. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, Going deeper with convolutions, preprint, arXiv:1409.4842 [cs.CV].
 36. P. Tahmasebi and A. Hezarkhani, Application of optimized neural network by genetic algorithm, *IAMG09*, Vol. 2 (Stanford University, California, 2009), pp. 15–23.
 37. A. H. Toselli, J. Puigcerver and E. Vidal, Two methods to improve confidence scores for Lexicon-free word spotting in handwritten text, in *Proc. 15th Int. Conf. Frontiers in Handwriting Recognition (ICFHR)* (IEEE Computer Society, 2016), pp. 349–354.
 38. J. Vesanto and E. Alhoniemi, Clustering of the self-organizing map, *IEEE Trans. Neural Netw.* **11**(3) (2000) 586–600.
 39. Y. Xu *et al.*, A regression approach to speech enhancement based on deep neural networks, *IEEE Trans. Audio Speech Lang. Process.* **23**(1) (2015) 7–19.
 40. K. Zagoris, I. Pratikakis and B. Gatos, Unsupervised word spotting in historical handwritten document images using document-oriented local features, *IEEE Trans. Image Process.* **26**(8) (2017) 4032–4041.
 41. M. Zhang and J. Fulcher, Face recognition using artificial neural networks group-based adaptive tolerance (GAT) trees, *IEEE Trans. Neural Netw.* **7** (1996) 555–567.
 42. Y. Zhang, Y. Xie, Y. Zhang, J. Qiu and S. Wu, The adoption of deep neural network (DNN) to the prediction of soil liquefaction based on shear wave velocity, *Bull. Eng. Geol. Environ.* **80** (2021) 5053–5060, doi:10.1007/s10064-021-02250-1.
-



Shiueng-Bien Yang received his B.S. degree in 1993 and Ph.D. degree in 1999 both from the Department of Applied Mathematics, National Chung Hsing University, Taichung, Taiwan. He is currently a Professor at the Department of Digital

Content of Application and Management, Wenzao Ursuline University of Languages, Kaohsiung, Taiwan. His research interests include pattern recognition, speech coding, image processing, and neural network.



Ting-Wen Liang received his Bachelor's degree in 1991 from the National Taiwan Normal University, followed by the Master's degree in 1993 from the National Tsing-Hua University in Hsinchu, Taiwan. He is currently serving as a Lecturer in

the Department of Digital Content Application and Management at Wenzao Ursuline University of Languages, Kaohsiung, Taiwan. Additionally, he is pursuing his Ph.D. at Swinburne University of Technology in Sarawak, Malaysia. His research interests are centered around machine learning, deep learning, and artificial intelligence, to which he actively contributes through his academic endeavors.