Minimization of Number of Neurons in Voronoi Diagram-Based Artificial Neural Networks

Chen-Yu Lin, Yung-Chih Chen, Chun-Yao Wang, *Member, IEEE*, Ching-Yi Huang, and Chiou-Ting Hsu, *Senior Member, IEEE*

Abstract—Artificial Neural Networks (ANNs) have been widely used to deal with various classification problems for decades. Different algorithms for synthesizing ANNs have been proposed as well. The number of neurons in an ANN usually controls the tradeoff between classification ability and computational efficiency. That is, more neurons tend to yield better results but are less efficient in either the training or recalling phase. Furthermore, if the neurons are implemented by physical devices, the implementation cost can be effectively reduced with fewer number of neurons in an ANN. In this paper, we propose a method to minimize the number of neurons used in an ANN that is built by using Voronoi diagrams without suffering any capability loss. We have conducted experiments on a set of benchmarks. The experimental results show that the resultant ANNs reduce the number of neurons by up to 94 percent.

Index Terms—Minimization, artificial neural networks, voronoi diagram

1 Introduction

Artificial Neural Network (ANN) is a model inspired by biological neural networks, in particular, the brain [8]. An ANN adapts a set of inputs, called training data, to form its functionality. During the learning (or training) process of ANN construction, different training algorithms are applied to adjust the weights of connections between neurons. By iteratively adjusting the weights, the functionality of ANNs is also changed until approaching the target problem appropriately. In general, ANNs are used to classify patterns in a multi-dimensional feature space for different applications, such as computer vision [19] and speech recognition [11], that are difficult to solve by using the ordinary rule-based programming [8].

An ideal ANN is able to learn from the training data and recall after the training process in order to classify new patterns. There are three types of learning paradigms in ANNs: supervised learning [8], unsupervised learning [17], and reinforcement learning [22]. Each learning paradigm is suitable for certain problems. For example, a task that falls within the paradigm of supervised learning is pattern recognition [17], whereas a task that falls within the paradigm of unsupervised learning is the estimation of statistical distributions [21].

Many training algorithms for ANNs have been proposed in the last decades [8]. Most of them employed some forms of gradient descent [14], such as backpropagation [8], [9], to minimize the loss function or error function. Although using this mathematical approach can theoretically achieve high accuracy, the training process is very time-consuming. Therefore, instead of using gradient descent, some other approaches were proposed to reduce the training efforts [4], [12], [23].

The previous work [4] exploited the Voronoi diagram (VoD) [16] to build ANNs and features 100 percent accuracy in classifying the training data. Fig. 1a is the VoD over a set of training data, which are represented as dots, belonging to the same class. The construction of VoD is to determine the hyperplanes, represented by the lines, on the VoD. The hyperplanes exist in between every pair of adjacent dots with an equal distance and is perpendicular to the invisible edge connecting the pair of dots. Each dot is within its separated area called Voronoi cell, formed by hyperplanes as shown in Fig. 1a. From the viewpoint of implementation, each hyperplane and Voronoi cell requires a specific neuron in the construction of the ANN. Thus, if a VoD can be constructed with fewer hyperplanes, or Voronoi cells, or both, the number of the corresponding neurons in an ANN will be minimized. As a result, the efficiency of calculation in an ANN will be elevated, or the hardware cost of ANNs will be reduced when ANNs are implemented by physical devices [2], e.g., CMOS solutions, capacituve implementations, conductance/current implementations, single electron tunneling (SET) solutions, and Resonant Tunneling Devices (RTDs). Note that, our goal is to minimize the number of neurons without compromising the capability. Fig. 1b is a minimized VoD of Fig. 1a, where the functionalities of the original and minimized ANNs are the same.

In this paper, we propose a method to minimize the number of neurons in VoD-based ANNs by simplifying its VoD. We conduct experiments on a set of benchmarks [25], [26]. The experimental results show that the resultant ANNs reduce the number of neurons by up to 94 percent.

• *C.-Y. Lin, C.-Y. Wang, C.-Y. Huang, and C.-T. Hsu are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30013, R.O.C. E-mail: pohy960@gmail.com, {wcyao, cthsu}@cs.nthu.edu.tw, s986516@m98.nthu.edu.tw.*
• *Y.-C. Chen is with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan 32003, R.O.C. E-mail: ycchen.cse@saturn.yzu.edu.tw.*

Manuscript received 6 Nov. 2015; revised 8 Mar. 2016; accepted 10 Apr. 2016. Date of publication 20 Apr. 2016; date of current version 19 Dec. 2016. Recommended for acceptance by H. Li.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMSCS.2016.2555303

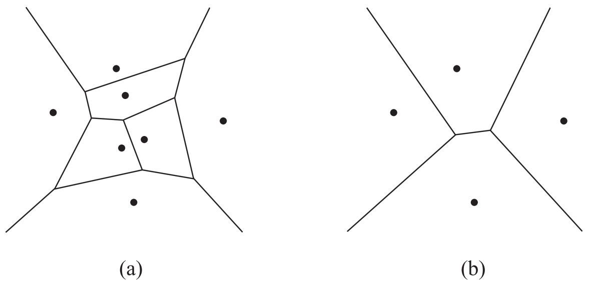


Fig. 1. (a) The VoD over a set of training data within the same class. (b) The VoD after minimization.

The main contributions of this work are two-fold:

1) To the best of our knowledge, this is the first work that minimizes the number of neurons in VoD-based ANNs.
2) The minimized ANNs obtained the same accuracy as the original ANNs.

The rest of this paper is organized as follows. Section 2 gives the background of ANN and VoD. Section 3 presents the proposed minimization method for VoD-based ANNs. Section 4 shows the experimental results. Section 5 presents the conclusion and future work.

2 Preliminaries

2.1 ANN

There are two kinds of input data for ANNs, training data set and test data set. The training data set is the input data used to train the ANNs. The test data set, on the other hand, is the input data used for classification after the ANNs are constructed. The process of classifying the test data is called recalling.

In the training phase, error functions are used to measure the difference between the expected output and the actual output of ANNs. An example of a common error function is the mean squared error [8]. An error function feedbacks information from the training data to adjust the weights of connections between neurons. Since the termination condition in the training phase usually depends on the magnitude of error, it is not surprising that a well-trained ANN can classify the training data with a very small error. In other words, a high-level of accuracy of an ANN for classifying the training data is always expected. However, the training phase might be time-consuming to converge to an acceptable error value. Although some heuristics for speeding up the training phase of ANNs were proposed [14], [15], the issue of error convergence has an inherent difficulty—determination of ANN topology.

When solving a classification problem by using ANNs, we first need to determine the topology of an ANN, such as the number of hidden layers, the number of neurons in the hidden layers, etc. The weights can be initialized randomly or heuristically [5], [7]. Different network topologies have been proposed for selection such as feedforward neural network [20], recurrent neural network [18], etc. Then designers conduct a learning algorithm to train the network. The major difficulty with this approach is that it is not trivial to determine the topology and the parameters of the ANN, due to lack of knowledge about the problems to be solved.

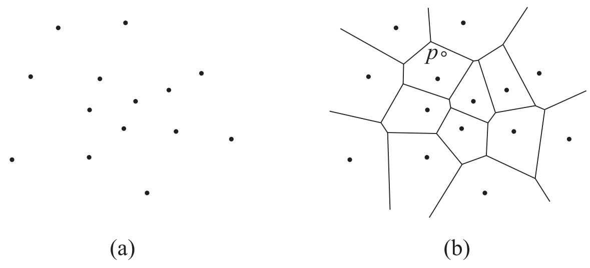


Fig. 2. (a) A set of seeds. (b) The VoD of the seeds in (a).

Moreover, because the convergence of error function is not always guaranteed, if the topology and the parameters of the selected ANN is not appropriate, the ANN may even fail to classify the training data correctly. Since determination of the topology and these parameters might be in a trial-and-error manner, it usually takes a long time to reach the local optimum of the error.

To solve the above-mentioned problems in the training phase of ANNs, some training algorithms without using gradient descent methods have been proposed [4], [12]. These algorithms used hyperplanes to partition the input space and then separated the regions of different classes by using some hyperplanes and additional AND and OR gates. [4] and [12] used the VoD, and a Linear Discriminant Analysis to construct the corresponding ANNs, respectively. As a result, they avoided the issue of topology and parameter determination and ensured the correctness of classifying the training data to be 100 percent. We will discuss the construction of VoD-based ANN in Section 2.3.

2.2 VoD Basics

A Voronoi diagram is a unique partitioning of a feature space into regions called Voronoi cells (also known as Voronoi regions). Let X be a metric space with the distance function d. Let $(P_k)_{k\in K}$ be a tuple (ordered collection) of nonempty subsets (the sites) in the space X, where K is a set of indices. The Voronoi cell, or Voronoi region, R_k, associated with the site P_k is the set of points in X whose distance to P_k is shorter than or equal to their distance to the other sites P_j, $j \neq k$ under the distance metric d [24]. In other words, if $d(x, A) = \inf\{d(x,a) | a \in A\}$, where $\inf$ represents the infimum of a subset, denotes the distance between the point x and the subset A, then

$$R_k = \{x \in X | d(x, P_k) \leq d(x, P_j) \text{ for all } j \neq k\}. \tag{1}$$

Specifically, given two points $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$ in the Euclidean n-space, the distance d between p and q is the Euclidean distance, defined by

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}. \tag{2}$$

In this Euclidean case, each site P_k contains only one single seed and the corresponding Voronoi cell R_k is a convex set.

For example, considering the set of seeds in Fig. 2a, its VoD can be constructed as shown in Fig. 2b. Any point p in a Voronoi cell is closer to the seed of its Voronoi cell than any other seeds.

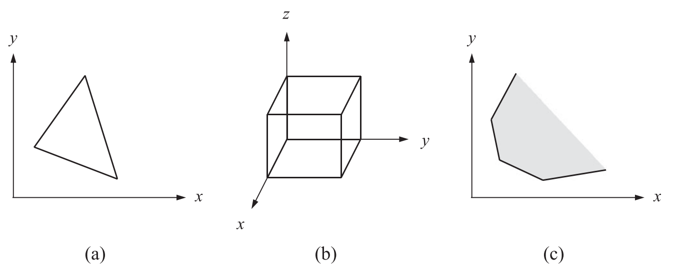


Fig. 3. (a) A polygon in a two-dimensional space. (b) A polyhedron in a three-dimensional space. (c) An unbounded 2-polytope.

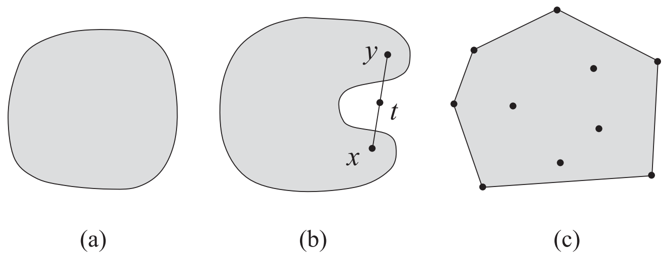


Fig. 4. (a) A convex set. (b) Not a convex set. (c) A convex hull of these dots.

In elementary geometry, a polytope is a geometric object with flat sides, and may exist in any number of dimensions n as an n-dimensional polytope or n-polytope. For example, a two-dimensional polygon is a 2-polytope and a three-dimensional polyhedron is a 3-polytope [6]. For example, the triangle in Fig. 3a is a 2-polytope and the cube in Fig. 3b is a 3-polytope. A traditional polytope is a closed region as shown in Figs. 3a and 3b. However, in our discussion, a polytope is allowed to be unbounded. As shown in Fig. 3c, the domain (the gray area) of this 2-polytope is unbounded.

In the Euclidean space, an object is a convex set if and only if every point on a straight line segment connecting two points within the object is also within the object. For example, Fig. 4a is a convex set. whereas Fig. 4b is not a convex set since some point *t* on the line segment connecting points *x* and *y* lies outside the set. Given a set X of points in the Euclidean space, the convex hull is the smallest convex set containing all the points in X. For example, Fig. 4c is the convex hull of the given set of points. Note that the convex hull of a finite set of points in the feature space is also a convex n-polytope.

2.3 VoD-Based ANN

Fig. 5a is a model of a neuron. A neuron in ANNs receives weighted inputs and sums them up to produce a value. This value is passed through a nonlinear function θ known as a transfer function to get the output value. The step function, as shown in Fig 5(b), is chosen as the transfer function for a VoD-based ANN, where T stands for the threshold. In a neuron, the output f is then evaluated as:

$$f = \theta(x_1, x_2, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq T \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < T. \end{cases} \tag{3}$$

That is, if the weighted sum is equal to or larger than the threshold T, the output is 1; otherwise, the output is 0.

The function of a neuron can be illustrated as a hyperplane in an n-dimensional space, i.e., $H : \Sigma_{i=1}^{n} x_i w_i - T = 0$. The half-spaces described by $\Sigma_{i=1}^{n} x_i w_i - T \geq 0$ and $\Sigma_{i=1}^{n} x_i w_i - T < 0$ are denoted as the positive half-space H^+ and the negative half-space H^-, respectively. When any point located in H^+ is applied to the corresponding neuron, the output equals 1; otherwise, the output equals 0 [4].

Fig. 5c is an example of threshold logic gate that models a neuron using a step function as the transfer function. The two 1s in Fig. 5c are the weights of the inputs x and y and the threshold value T is 2. The output f equals 1 if and only if $x \times 1 + y \times 1 \geq 2$. The function of this neuron can be illustrated in a two dimensional plane as shown in Fig. 5d. In Fig. 5d, when any point that is on or above the hyperplane L is applied to the neuron, the output f equals 1; and when any point below L is applied to the neuron, the output f equals 0. The arrow in Fig. 5d represents the half-space defined by the neuron. Since half-space can be defined by a single neuron, any convex polytope can be formed by a set of hyperplanes and one AND gate. For example, the convex 2-polytope in Fig. 5e can be formed by four neurons, H_1-H_4, and one AND gate as shown in Fig. 5f. The AND gate in Fig. 5f is used as a conjunction operation, and it can be implemented by a neuron as well [4]. Thus, the output f of the ANN in Fig. 5f equals 1 (0) when any point that is inside (outside) the convex 2-polytope in Fig. 5e is applied to the

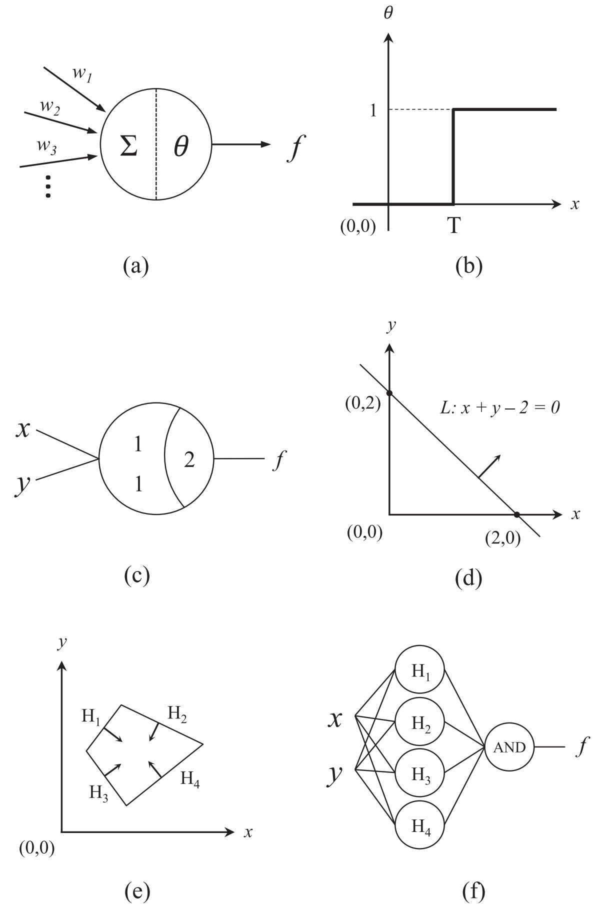


Fig. 5. (a) An artificial neuron model. (b) The step function with threshold T. (c) A two input threshold logic gate. (d) The hyperplane represented by (c). (e) A bounded convex 2-polytope. (f) The ANN representing the convex 2-polytope in (e).

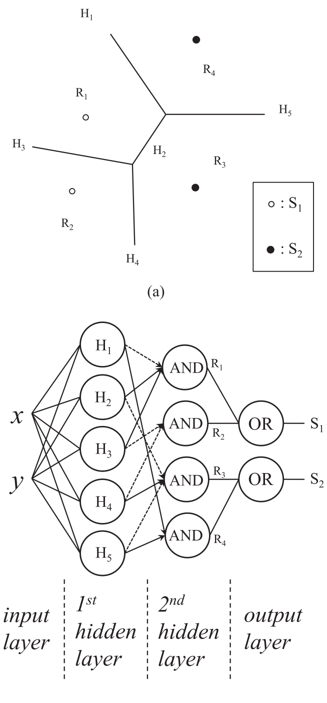


Fig. 6. (a) The VoD of a data set. (b) The ANN of (a).

network. Note that the construction method of ANN for unbounded convex n-polytope is the same.

In general, VoD-based ANNs consist of four layers including one input layer, two hidden layers, and one output layer. We use Fig. 6 to explain how to derive a VoD-based ANN. In Fig. 6a, assume that there are four seeds (training data) belonging to two classes, S_1 and S_2. Fig. 6b is the corresponding VoD-based ANN. The input layer is used to accept the inputs and it is connected to the neurons in the first hidden layer. The neurons in the first hidden layer serve as the hyperplanes H_1-H_5 in Fig. 6a for partitioning the feature space. Each neuron defines two half-space of one hyperplane. After deploying the neurons in the first hidden layer, AND gates in the second hidden layer are used to define every Voronoi cell (bounded or unbounded convex 2-polytope)[1] by collecting required half-space from the first hidden layer. For example, the unbounded convex 2-polytope R_4 is defined by H_1 and H_5. Therefore, H_1 and H_5 are connected to the fourth AND gate. Note that different half-spaces of a hyperplane are selected by assigning the weights of the connections between the first and second hidden layers to either 1 (solid lined arrow) or -1 (dashed lined arrow), as Fig. 6b shows [4]. Finally, the OR gates in the output layer are used to form the disjunction of the convex 2-polytopes of the same class. As a result, when any point in R_1 or R_2 of Fig. 6a is applied to the ANN of Fig. 6b, the output (S_1, S_2) will be (1, 0), which indicates that the point belongs to the class S_1. In contrast, when any point in R_3 or R_4 of Fig. 6a is applied to the ANN of Fig. 6b, the output (S_1, S_2) will be (0, 1).

The training process for constructing a VoD-based ANN is much faster than that of the gradient descent method-based ANN, because the searching process for a minimal error is no longer needed in a VoD-based ANN. As a result, the VoD-based ANN is able to be derived for classifying the training data with 100 percent accuracy. Furthermore, the ANN can be always derived for a classification problem as long as the corresponding VoD are built.

1. In this example, the Voronoi cells are all unbounded convex 2-polytope.

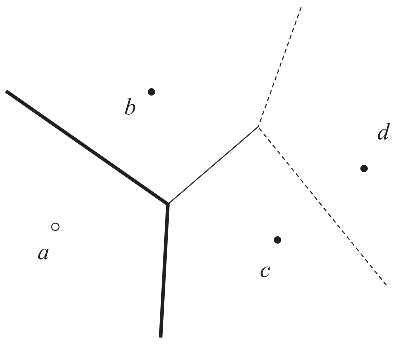


Fig. 7. The VoD over a set of seeds.

3 VoD-based ANN Minimization

In this section, we present our minimization algorithm for VoD-based ANNs. Although VoD-based ANN is superior on accuracy and efficiency, its main concern is the large number of neurons. This large amount of neurons are present due to many unnecessary hyperplanes in the feature space. To alleviate this problem, we propose a method to remove hyperplanes associated with certain redundant training data. As a result, the corresponding neurons are eliminated in the ANNs.

We first exploit Fig. 7 to introduce the terminologies used in this section. Fig. 7 is the VoD of a two dimension classification problem. Black and white dots indicate the seeds of two classes, respectively.

Let S_{all} be the set of all Voronoi cells. Given the VoD and S_{all} of a classification problem, we define the following terms:

Neighbor Cells: Let HS_i be the set of hyperplanes associated with the Voronoi cell R_i for the seed i. Given two Voronoi cells, R_i and R_j, they are neighbor cells if and only if $HS_i \cap HS_j \neq \emptyset$. The set of pairwise neighbor cells, denoted as S_{NC}, is defined as:

$$S_{NC} = \{(R_i, R_j) | R_i, R_j \in S_{all}, \\ HS_i \cap HS_j \neq \emptyset, i \neq j\}. \tag{4}$$

For example, in Fig. 7, R_a and R_b are neighbor cells to each other, whereas R_a and R_d are not neighbor cells to each other. $S_{NC} = \{(R_a, R_b), (R_a, R_c), (R_b, R_c), (R_b, R_d), (R_c, R_d)\}$.

Boundary Cell: Let C_i be the class that the Voronoi cell R_i belongs to. A cell R_i is called a boundary cell if and only if there exists a cell R_j, where $(R_i, R_j) \in S_{NC}$ and $C_i \neq C_j$.

The set of boundary cells, denoted as S_{BC}, is defined as:

$$S_{BC} = \{R_i | R_i \in S_{all}, \exists R_j, where (R_i, R_j) \in S_{NC}, C_i \neq C_j\}. \tag{5}$$

For example, in Fig. 7, $S_{BC} = \{R_a, R_b, R_c\}$.

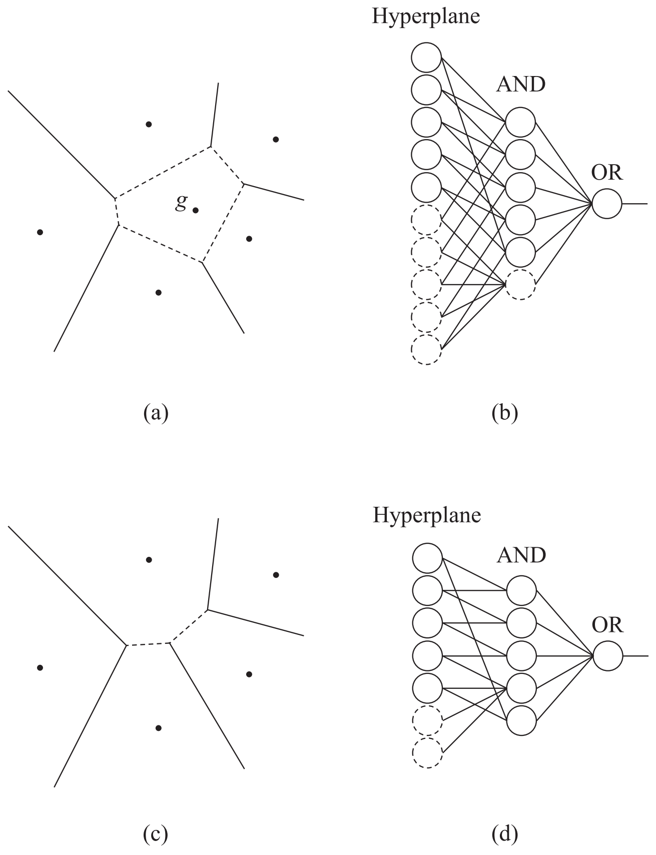


Fig. 8. (a) The VoD before minimization. (b) The ANN of (a). (c) The VoD after minimization. (d) The ANN of (c), which uses fewer neurons than (b).

Boundary Seed: A seed k is a boundary seed if and only if k is the seed of the cell R_k, where $R_k \in S_{BC}$.

For example, in Fig. 7, seeds a, b, and c are boundary seeds.

Isolated Cell: A Voronoi cell R_i is an isolated cell if and only if $R_i \in S_{IC} = S_{all} - S_{BC}$, where S_{IC} represents the set of isolated cells.

For example, in Fig. 7, the R_d is an isolated cell.

Isolated Seed: A seed k is an isolated seed if and only if k is the seed of the cell R_k, where $R_k \in S_{IC}$.

For example, in Fig. 7, seed d is an isolated seed.

Boundary Hyperplane: A hyperplane H is a boundary hyperplane if and only if $H = HS_i \cap HS_j \neq \emptyset$ for two boundary Voronoi cells $R_i, R_j \in S_{BC}$, and $C_i \neq C_j$.

For example, in Fig. 7, the bold lined hyperplanes are boundary hyperplanes.

Isolated Hyperplane: A hyperplane H is an isolated hyperplane if and only if $H \in HS_i$, where HS_i is the set of the hyperplanes that forms the Voronoi cell $R_i \in S_{IC}$.

For example, in Fig. 7, the dotted lined hyperplanes are isolated hyperplanes.

In a classification problem, we first use each training point as a seed to build a VoD and obtain its S_{all}. To minimize the number of neurons in an ANN, we then locate the set of boundary cells S_{BC} and the corresponding boundary seeds. Since we do not want to change the functionality of the ANN after the minimization, we keep all the boundary seeds while removing all isolated seeds from the training data. After that, we use the reduced training data to derive the new ANN.

Fig. 8 is a simple example to illustrate this minimization process. Fig. 8a depicts a set of Voronoi cells S_{all} of the same class, which consists of six Voronoi cells and the corresponding six seeds. Note that Fig. 8a is the VoD of a partial set of training data for brevity. There is one isolated seed g within the class, and it is separated from other cells by the isolated hyperplanes as shown in dotted lines. The rest of the Voronoi cells are $\in S_{BC}$. Fig. 8b shows the corresponding ANN of Fig. 8a. Note that the input layer is omitted in this example. Each neuron in the first hidden layer of Fig. 8b represents one hyperplane in Fig. 8a and the five dotted lined neurons in the first hidden layer represent the five isolated hyperplanes. The neurons in the second hidden layer of Fig. 8b are AND gates for representing the convex 2-polytopes, with respect to the Voronoi cells in Fig. 8a. Since none of the Voronoi cells in Fig. 8a can be combined with another one to form a larger convex 2-polytope, six AND gates are required to represent the cluster of six Voronoi cells. The dotted AND gate in the second hidden layer is used to represent S_{IC}. Finally, an OR gate is required in the output layer to represent this class.

Since the isolated hyperplanes (dotted lines) in Fig. 8a do not separate two seeds of different classes from each other, the isolated hyperplanes are unnecessary and can be removed. In order to remove the isolated hyperplanes, we delete the isolated seeds from the training data and re-construct the ANN with the reduced training data. Fig. 8c is the VoD with the reduced training data. In this simplified VoD, the five isolated hyperplanes (dotted lines) and one isolated cell in Fig. 8a are removed, and two new hyperplanes (dotted lines) in Fig. 8c are added. Fig. 8d is the resultant ANN, where the number of neurons is reduced by 23.5 percent.

Next, we present our proposed VoD-based minimization method, where a simple algorithm for identifying all the isolated seeds in the training data is shown in Algorithm 1. The input is a set of training data TD and the output is the set of isolated seeds IS. First, we initialize the status of every seed s as an isolated seed from line 1 to line 1. Then we construct the VoD based on the training data TD. After the VoD is constructed, each hyperplane is examined to see if the two seeds across the hyperplanes are within different classes. If the two seeds are within different classes, they are both identified as boundary seeds and not isolated seeds, as shown from line 1 to line 1. Fig. 9 is the flowchart of the proposed approach. The input is the training data set. First, we identify the isolated seeds in the training data set by Algorithm 1. If there exists any isolated seed, we remove it from the training data set. Then, we construct the VoD based on the trimmed training data set. Finally, we derive the ANN from the constructed VoD.

Note that the new ANN reserves the same functionality after this minimization process since none of the boundary hyperplanes are involved in this procedure. As another example in Fig. 10 shows, the regions of class S_1 and S_2 are the same before and after the minimization.

Next, we present the formal proof about the functionality preserving of ANNs after applying the proposed minimization process.

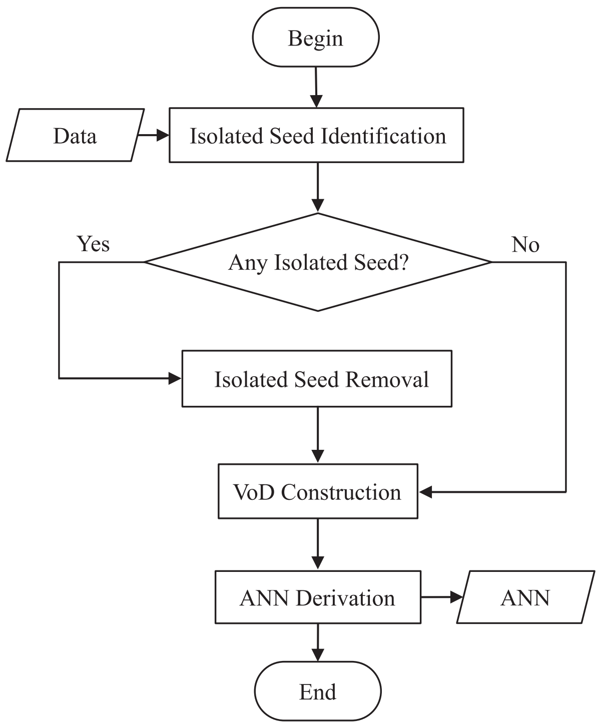


Fig. 9. The flowchart of the proposed approach.

Lemma 1. For an $R_i \in S_{IC}$, if there exists an R_j such that $(R_i, R_j) \in S_{NC}$, $C_i = C_j$.

Proof. According to the definition of $R_i \in S_{IC}$, then $R_i \notin S_{BC}$. That is, it is not the case that $\forall R_j$, $(R_i, R_j) \in S_{NC}$ and $C_i \neq C_j$. Hence, $C_i = C_j$. □

Theorem 1. *The functionality of the ANN keeps intact after applying the proposed VoD-based ANN minimization method.*

Proof. To prove Theorem 1, we need to prove that the regions of all cells within the same class do not change after the minimization. In other words, when an isolated seed is removed and the VoD is updated, the region of the isolated cell will be covered by other cells within the same class in the n-dimensional space.

When an isolated seed is removed from a VoD, its isolated cell will be removed as well on the updated VoD. According to Lemma 1, the isolated cell and its neighbor cells are within the same class. Thus, the region of the isolated cell on the updated VoD can be covered by its neighbor cells. Since the region of each class does not change, the functionality of the ANN is the same after the minimization. □

4 Experimental Results

The proposed algorithm for VoD-based ANN minimization was implemented in C++ language. We conducted experiments on a set of classification problem benchmarks [25], [26] on a Linux platform with an Intel Xeon E5530 2.40 GHz CentOS 4.6 platform and 64 GB memory. The VoD was constructed by the tool Qhull [1], [27]. In the Qhull, a VoD is obtained by converting the Delaunay triangulation into its dual. The Delaunay triangulation is achieved by computing a convex hull. Please see [27] for more information.

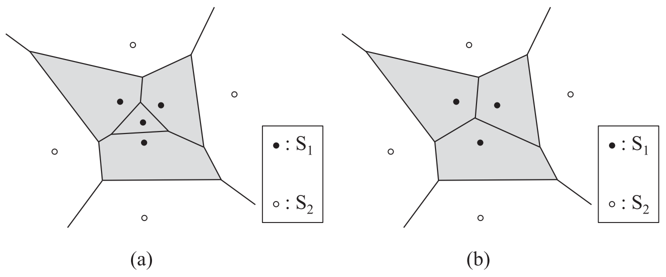


Fig. 10. (a) The VoD before minimization. (b) The simplified VoD of (a).

Algorithm 1. Isolated Seed Identification

```
Input: a set of training data TD
Output: a set of isolated seeds IS
 1: begin
 2:   for each seed s in TD do
 3:     $isolated[s] \leftarrow true$;   //initialization
 4:   end for
 5:   derive the VoD G of TD;
 6:   for each hyperplane h in G do
 7:     if the two seeds a, b across h are within different
        classes then
 8:       $isolated[a] \leftarrow false$;
 9:       $isolated[b] \leftarrow false$;
10:     end if
11:   end for
12:   for each seed s in TD do
13:     if $isolated[s]$ == true then
14:       add s into IS;
15:     end if
16:   end for
17:   return IS
18: end
```

To test if the constructed ANN is over-trained or not from a given training data set, designers usually apply another data set, called test data set, to the ANN. If the test data set is not available from the benchmark suite, the original training data set is usually divided into two sets, one is training data set, the other is test data set. However, in our approach, we focus on 100 percent accuracy on training data and prefer not to divide the training data set into two sets. The reason is as follows.

The VoD-based ANN is strongly related to the training data set and guarantees to have 100 percent accuracy. However, if some training data is considered as new test data arbitrarily, the structure of VoD will be changed and the corresponding functionality of derived ANN might be changed as Fig. 11 shows. Fig. 11a is an example of an original VoD, and it contains two classes. If an isolated seed is chosen to be the hidden training data in the experiments, like the target seeds in Fig. 11b, the functionality will not be changed even the ANN is derived with less training data, as shown in Fig. 11c. However, if a boundary seed is chosen to be the hidden training data in the experiments, like the target seeds in Fig. 11d, the functionality will be changed as shown in Fig. 11e. As a result, dividing the original training data into two sets is not appropriate in our work.

In our experiments, we compare the number of neurons in the ANNs before and after the minimization algorithm.

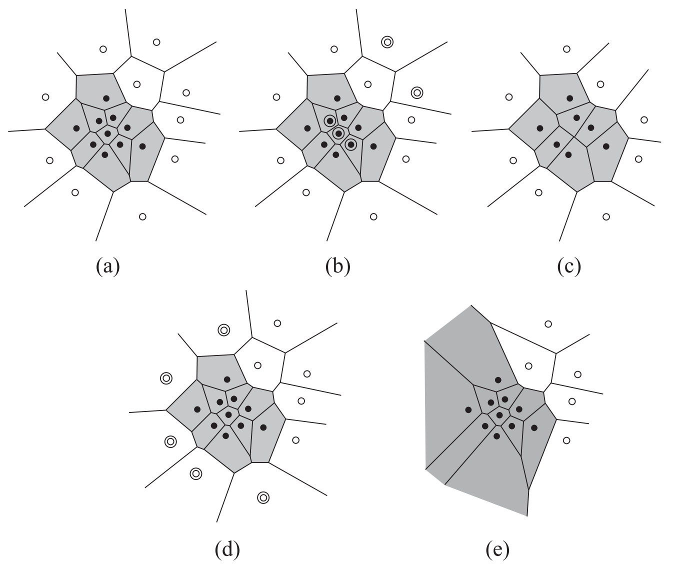


Fig. 11. (a) The original VoD. (b) The VoD with a set of selected target seeds. (c) The updated VoD after removing the target seeds in (b). (d) The VoD with another set of selected target seeds. (e) The updated VoD after removing the target seeds in (d).

The experimental results are shown in Table 1. In Table 1, Column 1 lists the benchmarks of the classification problems [25] ,[26]. Columns 2 and 3 are the number of dimensions and classes of the benchmark, respectively. The next three columns are the number of hyperplanes, $|HP|$, the number of Voronoi cells, $|cell|$, and the number of neurons, $|neuron|$, in the original VoD-based ANNs, respectively. The next three columns are the corresponding results after the minimization. Note that the number of neurons is the summation of the numbers of hyperplanes, Voronoi cells, and the classes. The last three columns list the reduction percentage of the neurons in the ANNs, the isolated seeds percentage in the training data set, and the required CPU time for minimization, respectively. The experimental results are divided into two groups based on the magnitude of percentage of isolated seeds: one is that with greater than or equal to 50 percent of isolated seeds; the other is that with less than 50 percent of isolated seeds.

For example, the benchmark *banana* has two dimensions and two classes. Its VoD-based ANN has 15851 hyperplanes and 5,291 cells. Note that we have removed the repeated cells before deriving the VoD-based ANN. Therefore, the total number of used neurons is 21,144. After the minimization, 5,078 hyperplanes and 1,698 cells are left. The total number of neurons is reduced to 6,778. That is, 67.9 percent neurons are removed after the minimization. The required CPU time for minimization is 0.9 seconds.

According to Table 1, our minimization algorithm can reduce the number of neurons in the resultant ANNs from 7 to 94 percent. It is clear that this reduction ratio varies with different benchmarks. Specifically, for the first group with greater than or equal to 50 percent of isolated seeds, the reduction of neurons count is 81.2 percent on average. For the other group, on the contrary, the reduction of neuron count is 20.4 percent on average. The reason for this result is the variation of the percentage of isolated seeds in the training data set.

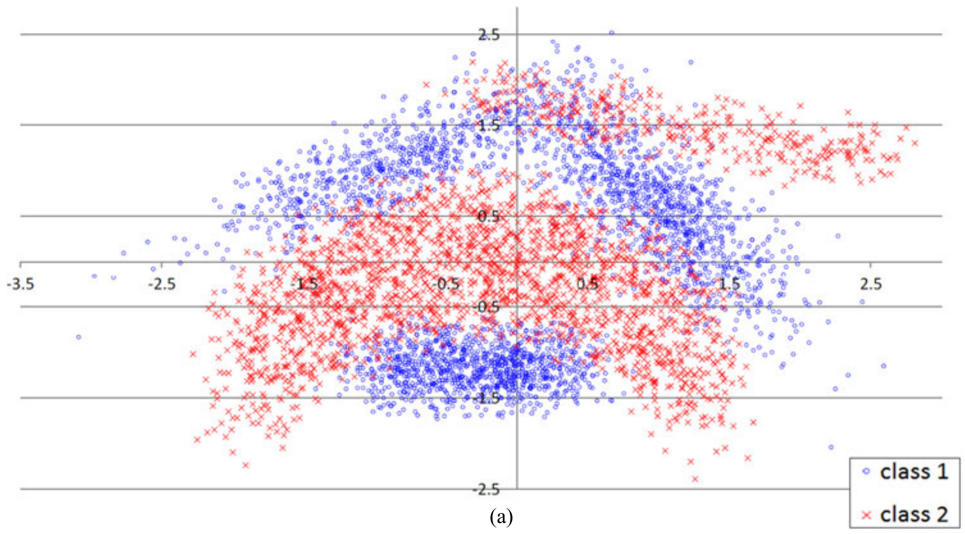
According to Table 1, the number of isolated seeds in the training data set directly affects the performance of our minimization approach. That is, if the seeds within the same class are close to each other and far from the seeds in other classes, such as *banana*, there will be more isolated seeds, and the new ANN can be minimized significantly by our approach. Fig. 12a is the data distribution of the benchmark *banana*. Since the data within the same class are close to each other, there are many isolated seeds in the corresponding VoD. Thus, the reduction of neuron count is remarkable. Fig. 12b is the remaining seeds after removing isolated seeds. On the other hand, the reduction of neuron count in the benchmarks *haberman* and *newthyroid* are not impressive. This is because there are few isolated seeds in the training data set. As a result, the number of neurons that can be removed is limited.

The proposed minimization approach for ANNs is based on its VoD. Thus, if the VoD of the original benchmark cannot be constructed due to certain reasons, e.g., a high dimension of training data in the feature space, the proposed approach will fail. This is the inherent

TABLE 1
The Experimental Results on the Number of Neurons Before and After the Proposed Algorithm

Group A											
			Before			After					
Benchmark	$\|Dimension\|$	$\|Class\|$	$\|HP\|$	$\|Cell\|$	$\|Neuron\|$	$\|HP\|$	$\|Cell\|$	$\|Neuron\|$	Reduction (%)	Isolated seed (%)	Time (s)
banana	2	2	15851	5291	21144	5078	1698	6778	67.9	67.9	0.90
monk-2	6	2	336	432	770	20	20	42	94.5	95.4	1.73
average	-	-	-	-	-	-	-	-	81.2	-	1.32

Group B											
			Before			After					
Benchmark	$\|Dimension\|$	$\|Class\|$	$\|HP\|$	$\|Cell\|$	$\|Neuron\|$	$\|HP\|$	$\|Cell\|$	$\|Neuron\|$	Reduction (%)	Isolated seed (%)	Time (s)
haberman	3	2	1663	283	1948	1550	256	1808	7.2	9.5	0.09
iris	4	3	1734	147	1884	1297	111	1411	25.1	24.5	0.13
data_bank.	4	2	20882	1380	22232	10851	696	11549	48.1	48.4	1.66
newthyroid	5	3	4491	215	4709	4073	196	4272	9.3	8.8	0.96
phoneme	5	2	177090	5349	182441	154980	4628	159610	12.2	13.5	107.02
average	-	-	-	-	-	-	-	-	20.4	-	21.97



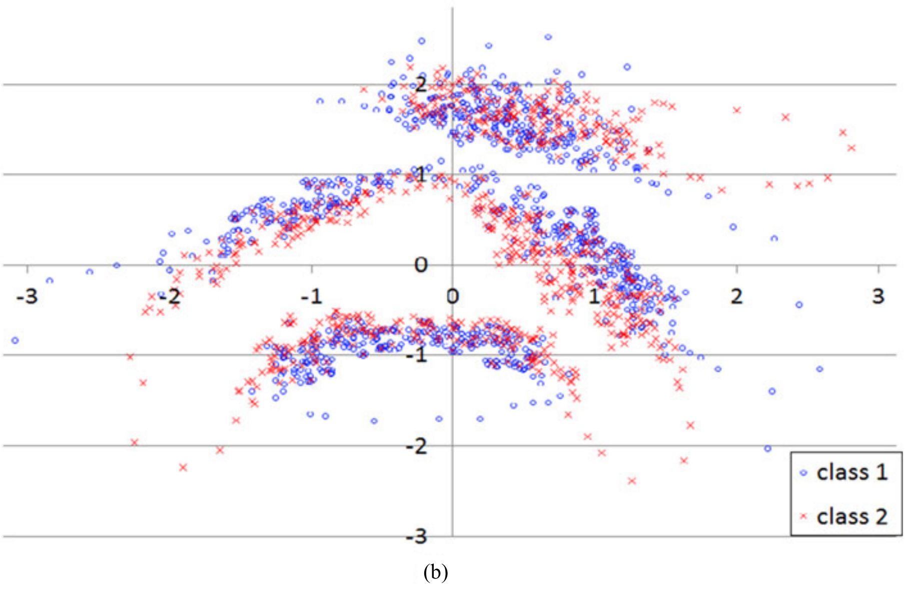


Fig. 12. (a) The training data distribution of the benchmark *banana*. (b) The remaining seeds.

limitation of the proposed approach. Other dimension reduction techniques [10] may be applied before deriving the VoD for improving the applicability of this approach. When the VoD can be derived, our minimization algorithm can be applied successfully.

5 Conclusion and Future Work

This paper presents the first minimization algorithm for reducing the neuron count in a VoD-based ANN while preserving its functionality. This minimization can be achieved when the seeds are clustered substantially. The experimental results show that the reduction percentage of neuron count is from 7 to 94 percent. Our future work are two directions: the first one is to apply the dimension reduction techniques in the construction of VoD such that the proposed minimization algorithm is applicable to more benchmarks. The other is to propose an algorithm that accepts some error tolerance compared with VoD-based ANN, but is more efficient.

Acknowledgments

This work is supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 103-2221-E-007-125-MY3, MOST 103-2221-E-155-069, MOST 104-2220-E-155-001, NSC 100-2628-E-007-031-MY3, NSC 101-2221-E-155-077, NSC 101-2628-E-007-005, NSC 102-2221-E-007-140-MY3, and NSC 102-2221-E-155-087.

References

[1] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa, “The quickhull algorithm for convex hulls,” *ACM Trans. Math. Softw.*, vol. 22, no. 15, pp. 469–483, Dec. 1996.
[2] V. Beiu, J. M. Quintana, and M. J. Avedillo, “VLSI implementations of threshold logic-A comprehensive survey,” *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1217–1243, Sep. 2003.
[3] C. M. Bishop, *Neural Networks for Pattern Recognition*. London, U.K.: Oxford Univ. Press, 1995.
[4] N. K. Bose and A. K. Garga, “Neural network design using voronoi diagrams,” *IEEE Trans. Neural Netw.*, vol. 4, pp. 778–787, Sep. 1993.
[5] Y. Chen and F. Bastani, “ANN with two-dendrite neurons and its weight initialization,” in *Proc. Int. Joint Conf. Neural Netw.*, Jun. 1992, pp. 139–146.

[6] H. S. M. Coxeter, *Regular Polytopes*. Mineola, NY, USA: Dover Publications, 1973.

[7] S. Devi, D. C. Panda, S. S. Pattnaik, B. Khuntia, and D. K. Neog, "Initializing artificial neural networks by genetic algorithm to calculate the resonant frequency of single shorting post rectangular patch antenna," in *Proc. IEEE Antennas Propag. Soc. Int. Symp.*, Jun. 2003, pp. 144–147.

[8] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*. Boston, USA: Pws Pub., 1996.

[9] R. Hecht-Nielsen, "Theory of the backpropagation neural network" in *Proc. Int. Joint Conf. Neural Netw.*, 1989, pp. 593–605.

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

[11] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural Netw.*, vol. 3, no. 1, pp. 23–43, Jan. 1990.

[12] Q. Li and D. W. Tufts, "Synthesizing neural networks by sequential addition of hidden nodes," in *Proc. IEEE Int. Conf. Neural Netw.*, 1994, pp. 708–713.

[13] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, Apr. 1987.

[14] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Netw.*, vol. 6, no. 4, pp. 525–533, 1993.

[15] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights," in *Proc. Int. Joint Conf. Neural Netw.*, Jun. 1990, pp. 21–26.

[16] F. P. Preparata and M. I. Shamos, *Computational Geometry*. Berlin, Germany: Springer-Verlag, 1985.

[17] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge, U.K.: Cambridge Univ. press, 1996.

[18] R. Rojas, *Neural Networks: a Systematic Introduction*. Berlin, Germany: Springer, 1996.

[19] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 1, pp. 23–38, Jan. 1998.

[20] T. D. Sanger, "Optimalunsupervised learning in a single-layer linearfeedforward neural network," *Neural Netw.*, vol. 2, no. 6, pp. 459–473, 1989.

[21] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001.

[22] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.

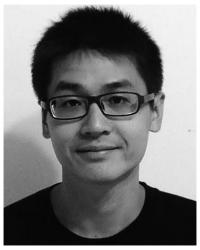
[23] Y. Wang, H. Yu, L. Ni, G.-B. Huang, M. Yan, C. Weng, W. Yang, and J. Zhao, "An energy-efficient nonvolatile In-memory computing architecture for extreme learning machine by domain-wall nanowire devices," *IEEE Trans. Nanotechnol.*, vol. 14, no. 6, pp. 998–1012, Nov. 2015.

[24] (2016). [Online]. Available: https://en.wikipedia.org/wiki/Voronoi_diagram

[25] (2015). [Online]. Available: https://archive.ics.uci.edu/ml/

[26] (2015). [Online]. Available: http://www.keel.es/

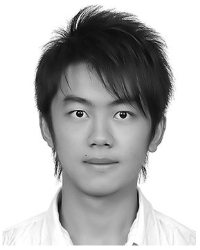
[27] (2015). [Online]. Available: http://www.qhull.org/



Chen-Yu Lin received the BS and MS degrees from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2013 and 2015, respectively. His research interests include logic synthesis and optimization for emerging technologies.

Yung-Chih Chen received the BS, MS, and PhD degrees from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2003, 2005, and 2011, respectively. He is currently an assistant professor with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan. His current research interests include logic synthesis, design verification, and design automation for emerging technologies.

Chun-Yao Wang (S'00-M'03) is a professor with the Department of Computer Science, National Tsing Hua University, Hsinchu. His two research results were nominated as Best Papers in the 2009 IEEE Asia and South Pacific Design Automation Conference and the 2010 IEEE/ACM Design Automation Conference, respectively. He is a member of the IEEE.



Ching-Yi Huang received the BS degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2009, where he is currently working toward the PhD degree in the Department of Computer Science. His research interests include logic synthesis, optimization, verification for very large-scale integrated designs, and automation for emerging technologies.

Chiou-Ting Hsu (M'98-SM'13) received the PhD degree in computer science and information engineering from the National Taiwan University, Taipei, Taiwan, in 1997. From 1998 to 1999, she was with Philips Innovation Center, Taipei, Philips Research, as a senior research engineer. She joined the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, as an assistant professor in 1999 and is currently a professor. She received the Citation Classic Award from Thomson ISI in 2001 for her paper "Hidden digital watermarks in images". She has served on the editorial board of *Advances in Multimedia* (2006-2012) and *IEEE Transactions on Information Forensics and Security* (2012-2015), and is currently an associate editor of the *Journal of Visual Communication and Image Representation and EURASIP Journal on Image and Video Processing*. Her research interests include image processing, digital image forensics, computer vision, and machine learning. She is a senior member of the IEEE.