Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Single-layered complex-valued neural network for real-valued classification problems

Md. Faijul Amin[a], Kazuyuki Murase[a,b,*]

[a] Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui, 3-9-1 Bunkyo, Fukui 910-8507, Japan
[b] Research and Education Program for Life Science, University of Fukui, 3-9-1 Bunkyo, Fukui 910-8507, Japan

ARTICLE INFO

Article history:
Received 1 June 2007
Received in revised form
25 April 2008
Accepted 29 April 2008
Communicated by T. Heskes
Available online 9 May 2008

Keywords:
Activation function
Classification
Complex-valued neural networks
Generalization
Phase-encoding

ABSTRACT

This paper presents a model of complex-valued neuron (CVN) for real-valued classification problems, introducing two new activation functions. In this CVN model, each real-valued input is encoded into a phase between 0 and π of a complex number of unity magnitude, and multiplied by a complex-valued weight. The weighted sum of inputs is then fed to an activation function. Both the proposed activation functions map complex values into real values, and their role is to divide the net-input (weighted sum) space into multiple regions representing the classes of input patterns. Gradient-based learning rules are derived for each of the activation functions. The ability of such CVN is discussed and tested with two-class problems, such as two- and three-input Boolean problems, and the symmetry detection in binary sequences. We show here that the CVN with both activation functions can form proper boundaries for these linear and nonlinear problems. For solving n-class problems, a complex-valued neural network (CVNN) consisting of n CVNs is also studied. We defined the one exhibiting the largest output among all the neurons as representing the output class. We tested such single-layered CVNNs on several real world benchmark problems. The results show that the classification ability of single-layered CVNN on unseen data is comparable to the conventional real-valued neural network (RVNN) having one hidden layer. Moreover, convergence of the CVNN is much faster than that of the RVNN in most cases.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Complex numbers are used to express real-world phenomena like signal amplitude and phase, and to analyze various mathematical and geometrical relationships. In order to directly process complex values by artificial neural networks, the complex-valued neural network (CVNN) as well as the complex back-propagation (CBP) algorithm have been developed [3,7,8,10,16]. The properties of the CVNN and CBP have been studied [5,11], and the CVNN is shown to be powerful in applications such as adaptive radar image processing [15], and optical image processing [2,4]. Further extension to multidimensional values has been attempted as well [13]. Some researchers recently have also applied CVNN on real-valued classification problems [1,9].

We are aware of two approaches for the application of CVNN to real-valued classification problems. In Ref. [9], each real-valued input is phase encoded between 0 and $\pi/2$ of unity magnitude complex number, so that the complex-valued neuron (CVN) can receive complex-valued inputs. The role of CVN is two-fold, aggregation and threshold operations. The former role is to aggregate the inputs multiplied by the connection weights, and the latter is to determine the class label by using an activation function. They showed that the CVN is successful in classifying all two-input Boolean functions, and 245 among 256 three-input Boolean functions. The learning algorithm, however, includes a reciprocal of partial derivatives. When the partial derivatives approach zero and the reciprocals become very large, the learning process may become unstable, especially when real world problems are evaluated. In a recent work [1], a multilayer feed-forward architecture of multivalued neuron is proposed. The model encodes real-valued inputs by phases between 0 and 2π of unity magnitude complex numbers, and determines the class label by the complex-valued output, based on the output's vicinity to the roots of unity. It has been shown that the model was able to solve the parity n and two spirals problems, and could perform better in "sonar" benchmark and the Mackey–Glass time series prediction problems.

In this paper, we propose two activation functions that map complex values to real-valued outputs. The role of the activation functions is to divide the net-input (sum of weighted inputs)

* Corresponding author at: Department of Human and Artificial Intelligence Systems, University of Fukui, 3-9-1 Bunkyo, Fukui 910-8507, Japan. Tel.: +81 776 27 8774; fax: +81 776 27 8420.
E-mail address: murase@synaspe.his.fukui-u.ac.jp (K. Murase).

0925-2312/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2008.04.006

space into multiple regions representing the classes. Since the net-input of a CVN is a complex number, it is a two-dimensional space. Both the proposed activation functions are differentiable with respect to real and imaginary parts of the net-input. As a result, a gradient-based learning rule can be derived. To present complex-valued inputs to the CVN, real-valued inputs are phase encoded between 0 and π. We will show in the following sections that such a CVN is able to solve linear and nonlinear classification problems such as two-input Boolean functions, 253 among 256 three-input Boolean functions, and symmetry detection in binary sequences.

To solve n-class problems, we considered a single-layered CVNN (without hidden layer) consisting of n CVNs described above, and formulated the learning and classification scheme. The single-layered CVNN has been applied and tested on various real world benchmark problems. Experimental results showed that the generalization ability of the single-layered CVNN is comparable to the conventional two-layered (with one hidden layer in addition to input and output layer) real-valued neural network (RVNN). It is noteworthy that in the proposed single-layered CVNN, the architecture selection problem does not exist. In the multilayered RVNNs which are considered as universal approximators [6], in contrast, the architecture selection is crucial to achieve better generalization and faster learning abilities.

The remainder of the paper is organized as follows. In Section 2, we discuss the model of CVN along with the proposed activation functions. In Section 3, we develop a gradient-based learning rule for training the CVN. In Section 4, we discuss the ability of the single CVN for some binary-valued classification problems. In Section 5, performances of the single-layered CVNN consisting of multiple CVNs are compared to those of the conventional two-layered RVNN on some real world benchmark problems. Finally, Section 6 gives a discussion and concluding remarks.

2. Complex-valued neuron (CVN) model

Since the CVN processes complex-valued information, it is necessary to map real input values to complex values in order to solve real-valued classification problems. After such mapping, the neuron processes information in a way similar to the conventional neuron model except that all the parameters and variables are complex-valued, and computations are performed according to complex algebra. As illustrated in Fig. 1, the neuron, therefore, first sums up the weighted complex-valued inputs and the threshold value to represent its internal state for the given input pattern, and then the weighted sum is fed to an activation function which maps the internal state (complex-valued weighted sum) to a real value. Here, the activation function combines the real and imaginary parts of the weighted sum.

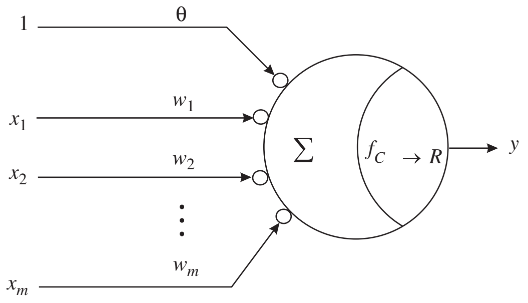


Fig. 1. Model of a complex neuron. The sign $\sum$ sums up the weighted inputs $w_j x_j (1 \leqslant j \leqslant m)$ and the bias θ. $f_{C\to R}$ is an activation function that maps the complex-valued internal state to a real-valued output y.

2.1. Phase encoding of the inputs

This section explains how the real-valued information is presented to a CVN. Consider (X, c) as an input example, where $X \in R^m$ represents the vector for m attributes of the example, and $c \in \{0, 1\}$ denotes the class of the input pattern. We need a mapping $R^m \to C^m$ to process the information with the CVN. One such mapping for each element of the vector X can be done by the following transformation:

$$\text{Let } x \in [a, b], \quad \text{where} \quad a, b \in R, \quad \text{then } \varphi = \frac{\pi(x-a)}{(b-a)} \tag{1}$$

and

$$z = e^{i\varphi} = \cos\varphi + i\sin\varphi \tag{2}$$

Here $i = \sqrt{-1}$. Eq. (1) is a linear transformation which maps $x \in [a, b]$ to $\varphi \in [0, \pi]$. Then by Euler's formula, as given by Eq. (2), a complex value z is obtained. When a real variable moves in the interval of $[a, b]$, the above transformation will move the complex variable z over the upper half of a unit circle. As shown in Fig. 2, the variation on a real line is thus now the variation of phase φ over the unit circle.

Some facts about the transformation are worth noting. Firstly, the transformation retains relational property. For example, when two real values x_1 and x_2 hold a relation $x_1 \leqslant x_2$, the corresponding complex values have the same property in their phases as such, $phase(z_1) \leqslant phase(z_2)$. Secondly, the spatial relationship among the real values is also retained. For example, two real values x_1 and x_2 are farthest from each other when $x_1 = a$ and $x_2 = b$. The transformed complex values z_1 and z_2 are also farthest from each other as shown in Fig. 2. Thirdly, the interval $[0, \pi]$ is better than the interval $[0, 2\pi]$ as we loose the spatial relationship among the variables in the latter. For example, $x_1 = a$ and $x_2 = b$ will be mapped to the same complex value since $e^{i0} = e^{i2\pi}$. It is worth noting that interval $[0, \pi/2]$ can also be used for phase encoding. However, experiments presented in Section 5 and Table 9 show that learning convergence is faster in the case of the interval $[0, \pi]$. Finally, the transformation can be regarded as a preprocessing step. The preprocessing is commonly used even in RVNNs in order to map input values into a specified range, and so on. The transformation in the proposed CVN, therefore, does not increase any additional stage for the process with neural networks. In fact, the above transformation does not loose any information from the real values; rather it lets a CVN process the information in a more powerful way.

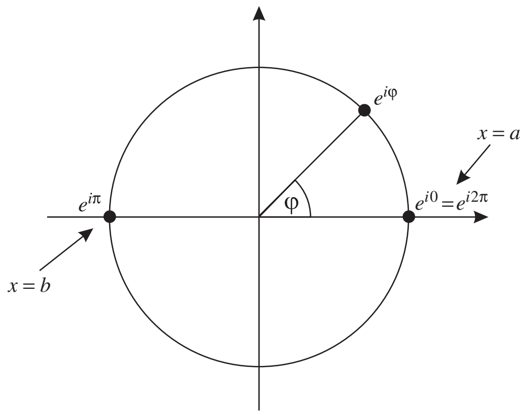


Fig. 2. Phase encoded inputs. When a real value x moves in the interval $[a, b]$, corresponding complex value moves over the upper half of unit circle. $x = a$, and $x = b$ are mapped by e^{i0}, and $e^{i\pi}$, respectively.

2.2. Activation function

When a CVN sums up inputs multiplied by the connection weights, the weighted sum is fed to an activation function to make the output bounded. In the real-valued case, activation functions are generally differentiable and bounded over the entire real domain, which makes it possible to derive a gradient-based backpropagation (BP) algorithm. But from Liouville's theorem, there is no complex-valued function except a constant which is differentiable and bounded at the same time. However, this is not as frustrating as it seems. For solving real-valued classification problems, we can combine real and imaginary parts of the weighted sum in some useful ways.

In this paper, we propose two activation functions that map complex values to real values for classification problems. Their role is to divide the neuron's net-input (sum of weighted inputs) space into multiple regions for identifying the classes, as is the role of the real-valued neuron (RVN) in the classification problems.

Let the net-input of a complex neuron be $z=u+\mathrm{i}v$, then we define two activation functions, 1 and 2, by Eqs. (3) and (4), respectively:

$$f_{C\to R}(z)=\sqrt{(f_R(u))^2+(f_R(v))^2} \tag{3}$$

$$f_{C\to R}(z)=(f_R(u)-f_R(v))^2 \tag{4}$$

where $f_R(x)=1/(1+\exp(-x))$ and $x,u,v\in R$. Both the activation functions combine the real and imaginary parts, but in different ways. The real and imaginary parts are first passed through the same sigmoid function individually. Thus each part becomes bounded within the interval (0,1). The activation function 1 in Eq. (3) gives the magnitude of the resulting complex value, while the activation function 2 in Eq. (4) gives the square of the difference between real and imaginary parts. Fig. 3 illustrates the output value y of both the activation functions in the domain of real and imaginary parts of net-input, u and v, respectively. It is noteworthy that defining activation functions as above makes it possible to compute the partial derivatives $\partial f/\partial u$ and $\partial f/\partial v$, which in turn makes it possible to derive gradient-based learning rules.

As seen in Fig. 3, both the CVN with the proposed activation functions saturate in four regions. On the contrary, the real-valued sigmoid function saturates only in two regions. Thus a RVN can solve only linearly separable problems. We show in the later sections that saturation in four regions of the proposed CVN significantly improves their classification ability.

3. Learning rule

Here we derive a gradient-descent learning rule for a CVNN composed of multiple CVNs described in the previous section. We deal with the CVNN without any hidden units. Consider an $m-n$ CVNN, where m and n are the numbers of inputs and outputs, respectively. Let $X=[x_1\cdots x_m]^{\mathrm{T}}$ be an m dimensional complex-valued input vector, $W_k=[w_{k1}\cdots w_{km}]^{\mathrm{T}}$ the complex-valued weight vector of neuron k, and θ_k the complex-valued bias for the kth neuron. To express the real and imaginary parts, let $x_j=x_j^R+\mathrm{i}x_j^I$, $w_{kj}=w_{kj}^R+\mathrm{i}w_{kj}^I$, and $\theta_k=\theta_k^R+\mathrm{i}\theta_k^I$. Then the net-input z_k, and the output y_k of the kth neuron are given by

$$z_k=\sum_{j=1}^{m}w_{kj}x_j+\theta_k \tag{5}$$

$$z_k=\left(\sum_{j=1}^{m}(w_{kj}^Rx_j^R-w_{kj}^Ix_j^I)+\theta_k^R\right)+\mathrm{i}\left(\sum_{j=1}^{m}(w_{kj}^Ix_j^R+w_{kj}^Rx_j^I)+\theta_k^I\right) \tag{6}$$

$$z_k=z_k^R+\mathrm{i}z_k^I \tag{7}$$

and

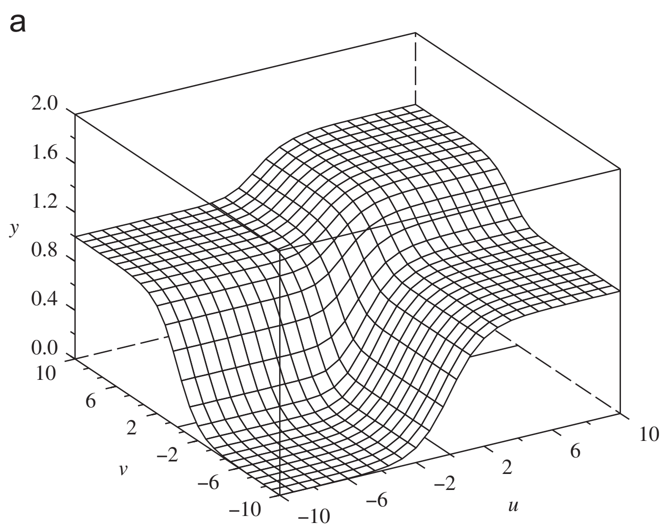
$$y_k=f_{C\to R}(z_k) \tag{8}$$

If the desired output of the kth neuron is d_k, then the error function to be minimized during the training is given by

$$E=\left(\frac{1}{2}\right)\sum_{k=1}^{n}(d_k-y_k)^2=\left(\frac{1}{2}\right)\sum_{k=1}^{n}e_k^2 \tag{9}$$

During the training, the biases and the weights are updated according to the following equations:

$$\Delta\theta_k=-\eta\frac{\partial E}{\partial\theta_k^R}-\mathrm{i}\eta\frac{\partial E}{\partial\theta_k^I}=\Delta\theta_k^R+\mathrm{i}\,\Delta\theta_k^I \tag{10}$$



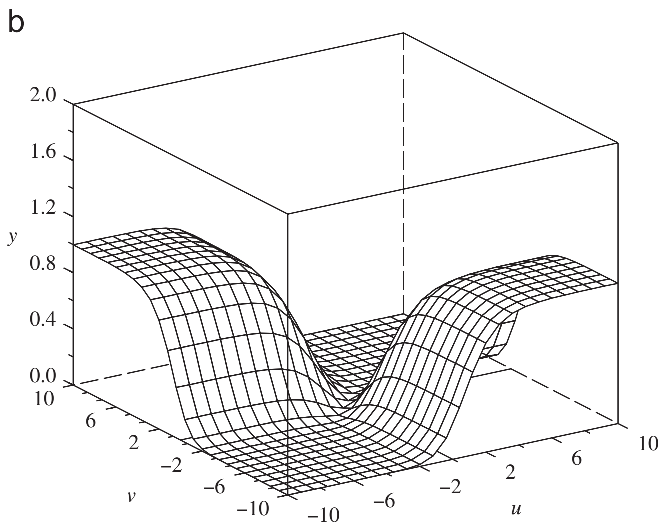


Fig. 3. The proposed activation functions 1 and 2 that map complex-value $u+\mathrm{i}v$ to real-value y: (a) activation function 1 is the magnitude $|f_R(u)+\mathrm{i}f_R(v)|$ of sigmoided real and imaginary parts, and (b) activation function 2 is the square of differences $(f_R(u)-f_R(v))^2$ between sigmoided real and imaginary parts.

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}^R} - \mathrm{i}\eta \frac{\partial E}{\partial w_{kj}^I} = \Delta w_{kj}^R + \mathrm{i}\,\Delta w_{kj}^I = \bar{x}_j\,\Delta\theta_k \tag{11}$$

where η is the learning rate and $\bar{x}_j$ is the complex conjugate of the complex number x_j.

If the activation function is given by Eq. (3), then

$$\Delta\theta_k^R = \eta e_k \frac{f_R(z_k^R)}{y_k} f_R'(z_k^R) \tag{12}$$

$$\Delta\theta_k^I = \eta e_k \frac{f_R(z_k^I)}{y_k} f_R'(z_k^I) \tag{13}$$

where $f_R(u) = 1/(1+\exp(-u))$ and $f_R'(u) = f_R(u)(1-f_R(u))$, $u \in R$. If the activation function is given by Eq. (4), then

$$\Delta\theta_k^R = 2\eta e_k (f_R(z_k^R) - f_R(z_k^I)) f_R'(z_k^R) \tag{14}$$

$$\Delta\theta_k^I = 2\eta e_k (f_R(z_k^I) - f_R(z_k^R)) f_R'(z_k^I) \tag{15}$$

Using Eqs. (12) and (13) in Eqs. (10) and (11), connection weights can be updated, which gives the learning rule of the CVNN for the activation function 1. Similarly, Eqs. (14) and (15) can be used for activation function 2.

4. Ability of CVN

A CVN is more powerful than an RVN, since it is able to solve linearly non-separable problems too. This ability is explained in this section for two- and three-input Boolean functions, and for the symmetry detection in binary sequences.

4.1. Two-input Boolean problems

There are 16 possible Boolean functions for two inputs. Some are linearly separable, while others are not. An RVN can solve only linearly separable problems since it gives a straight line as a decision boundary. To say this another way, the activation function of such a neuron saturates only in two regions. But a CVN with the proposed activation functions saturates in four regions and thus can achieve higher classification ability. In fact, a CVN could solve all the two-input Boolean functions. First, we explain it graphically for the XOR problem. Inputs x_1 and x_2 in Table 1 show the input values after phase-encoding discussed in Section 2.1. Let the weighted sum or the net-input of the neuron be $z = w_1x_1 + w_2x_2 + \theta = u + \mathrm{i}v$, where θ is the bias. After training the neuron, the net-input for each input pattern should produce the output near the desired output value shown in Table 1 by the mapping of $y = f_{C\rightarrow R}(z)$.

The net-inputs for the four input patterns in XOR problem and the contour diagram of output y are shown in Fig. 4. Among the four regions of saturation in the activation function 1 (Fig. 4(a)), the net-inputs after training were located in three regions; upper left, lower left, and lower right since the desired outputs were either zero or one. In case of the activation function 2 (Fig. 4(b)), all four regions were used. As is apparent in Fig. 4, the CVN with both the proposed activation functions was able to solve the XOR problem, while an RVN cannot solve such a linearly non-separable problem.

Table 1
Input–output mapping for XOR problem after phase encoding of inputs

Inputs		Output
x_1	x_2	y
1+i0	1+i0	0
1+i0	−1+i0	1
−1+i0	1+i0	1
−1+i0	−1+i0	0

Parameters obtained after the training, for all possible two-input Boolean functions, are given in Table 2. Each sequence $Y = y_1y_2y_3y_4$ denotes a Boolean function, where $y_1 = f(0,0)$, $y_2 = f(0,1)$, $y_3 = f(1,0)$, and $y_4 = f(1,1)$. For example, the sequence 0110 is the XOR problem, 0001 is the AND problem, 0111 is the OR problem, and so on. It can be checked from Table 2 that with the given parameters a CVN can solve all the two-input Boolean functions.

4.2. Three-input Boolean problems

For three inputs, the number of different Boolean functions is 256. Each of these functions can be expressed with a sum of products or *minterms*. Michel and Awwal [9] have reported that their CVN model could solve 245 among the 256 functions. Our CVN model could solve 253 (out of 256) functions with the activation function 2. The unsolved functions were $\sum m(0,1,2,4,7)$, $\sum m(0,3,5,6)$, and $\sum m(1,2,4,7)$. As for example, $\sum m(1,2,4,7)$ denotes the function $f(a,b,c) = a'b'c{+}a'bc'{+}ab'c'{+}abc$. Similarly, $\sum m(0,3,5,6) = a'b'c'{+}a'bc{+}ab'c{+}abc'$. When the CVN was applied with the activation function 1, it could solve 250 functions. In this case, the unsolved functions were $\sum m(0,3,5,6)$, $\sum m(1,2,4,7)$, $\sum m(1,2,4)$, $\sum m(1,2,7)$, $\sum m(1,4,7)$, and $\sum m(3,5,6)$. Considering the number of functions solved, our CVN model with both the activation functions could perform better than the model of Michel and Awwal [9].

From the classification point of view, the number of dichotomies is half the number of Boolean functions, i.e., there are 128 dichotomies for all possible three-input Boolean classification problems. This is because any function and its complement form a single classification problem or a dichotomy. Thus the functions $\sum m(0,3,5,6)$ and $\sum m(1,2,4,7)$, which are complements to each other, form one dichotomy, the so-called parity problem. Note that both the activation functions could solve either a function or its complement or both the function and its complement, except for the parity problem. In other words, our CVN model could solve all the three-input Boolean classification problems except the parity problem. It turns out that while solving the classification problems with the proposed activation functions, one scheme of target output setting (e.g., 0 for one class and 1 for the other) may perform better than the alternative scheme. However, the results presented above imply that the problem is less in the case of activation function 2 than for activation function 1.

The reason why activation function 2 works better is that we can use all the four regions of net-input space of a CVN in this case. It can be seen from Fig. 3(b) that the function uses two regions (where the function produces values near 0) for one class and the other two regions (where the function produces values near 1) for the other class. As a result, a CVN with the activation function 2 can map the input patterns on all the four regions. In case of activation function 1, we can use only one region (where the function produces values near 0) for one class, and two regions for the other class (regions producing values near 1) keeping the other region unused (as the function produces values near $\sqrt{}2$). The regions can be seen from Fig. 3(a). Since while solving Boolean functions the target outputs are set to 0 or 1, the CVN can map the input patterns on the three regions, in case of activation function 1. This issue is again addressed in the next section when the CVN is applied to the symmetry detection problem.

In general, the reason why our CVN could not solve some of the three-input Boolean functions may be because of the inability to create appropriate decision boundaries. Nevertheless, the number

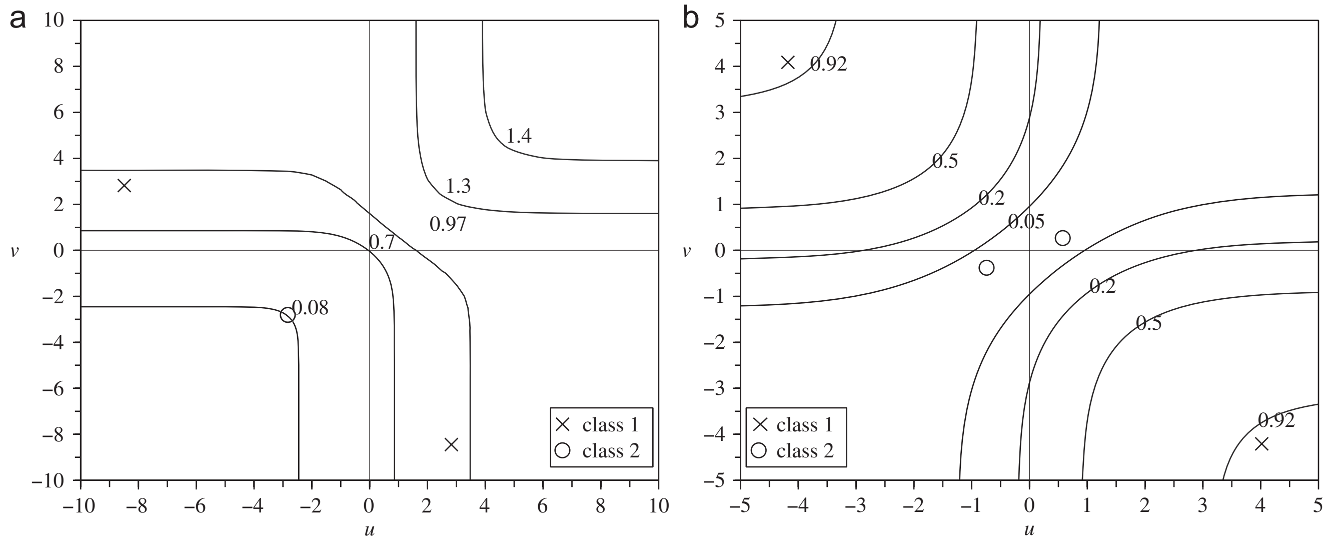


Fig. 4. Net-inputs of a CVN for XOR problem and contour diagram for activation functions 1 and 2 in (a) and (b), respectively. Curved lines indicate the points taking the same output values as indicated by the numbers. Horizontal and vertical lines are the real and imaginary axes, respectively. Cross mark [×] denotes the net-input for the input pattern belonging to class 1 (target output 1), and circle [O] denotes the net-input belonging to class 0 (target output 0).

Table 2
Learned weights and bias of CVN for two-input Boolean functions

Function	Activation 1			Activation 2		
$y_1y_2y_3y_4$	θ	w_1	w_2	θ	w_1	w_2
0000	−3.61−3.61i	0.00+0.00i	0.00+0.00i	−0.15+0.33i	−0.19−0.28i	−0.33−0.34i
0001	−2.97−2.98i	−1.83−1.99i	−1.83−1.99i	−2.53+0.17i	−0.96−2.68i	2.26−1.13i
0010	−2.97−2.98i	−1.79−2.03i	1.79+2.03i	1.58−2.03i	−2.69−0.74i	−0.29−2.64i
0011	−1.68−1.02i	−1.93−2.59i	0.00+0.00i	−2.27+2.32i	2.41−2.34i	0.05+0.07i
0100	−2.97−2.97i	1.86+1.96i	−1.86−1.96i	−2.36+1.05i	1.03+2.74i	2.57−0.25i
0101	−1.25−1.47i	0.03−0.03i	−2.36−2.14i	−2.27+2.32i	−0.03−0.03i	2.40−2.35i
0110	−2.83−2.81i	−2.83+2.82i	2.83−2.82i	−0.08−0.06i	1.72−2.24i	−2.38+1.19i
0111	−1.64+0.39i	−2.70−0.39i	1.61−2.90i	−1.74+1.98i	−1.74+1.98i	4.05−4.02i
1000	−2.95−2.96i	2.46+1.37i	2.46+1.37i	−2.31+0.96i	−2.48+0.33i	0.98+2.67i
1001	−2.83−2.83i	−2.83+2.83i	−2.83+2.83i	−0.06−0.07i	−2.13+2.64i	−2.54+2.03i
1010	−1.66−1.05i	−0.01+0.01i	1.95+2.56i	2.36−2.20i	−0.08−0.08i	2.31−2.46i
1011	1.58−1.89i	−1.58−2.17i	2.88−1.15i	1.50−2.17i	−4.10+3.98i	−1.50+2.17i
1100	−1.31−1.42i	2.30+2.19i	0.00+0.00i	2.22−2.50i	2.45−2.17i	−0.07−0.03i
1101	1.81−1.91i	2.91−0.93i	−1.81−1.95i	−4.01+4.07i	−1.99+1.74i	1.99−1.74i
1110	−1.05−0.84i	2.82−0.84i	−1.05+2.83i	−2.25+1.38i	2.25−1.38i	−3.95+4.13i
1111	1.11+0.57i	0.08−0.09i	−0.19+0.21i	−4.66+4.67i	0.02+0.01i	−0.01+0.00i

of three-input Boolean problems solved by the CVN is suggestive to explore the ability of CVN on some other nonlinear problems, for example, the symmetry detection in binary sequences.

4.3. Symmetry detection problem

The symmetry detection problem is detecting whether or not binary activity levels of a one-dimensional input array is symmetrical about the center point. An example of input–output mapping, for the input length of three bits, is shown in Table 3. It is a linearly non-separable problem and hence cannot be solved by an RVN.

4.3.1. Solving with one CVN

We tested the CVN on symmetry detection problems of various input lengths ranging from 2 to 10 bits. A single CVN with the proposed activation functions could solve all of the problems. For training the CVN, we set the target values as follows. With the activation function 1, the target was set to 0 if the input pattern was symmetric, and 1 if asymmetric. With activation function 2, the target was 1 if symmetric, and 0 if asymmetric.

Table 3
Input–output mapping of a symmetry detection problem with three inputs

Inputs			Output
x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Output 1 means corresponding input pattern is symmetric, and 0 means asymmetric.

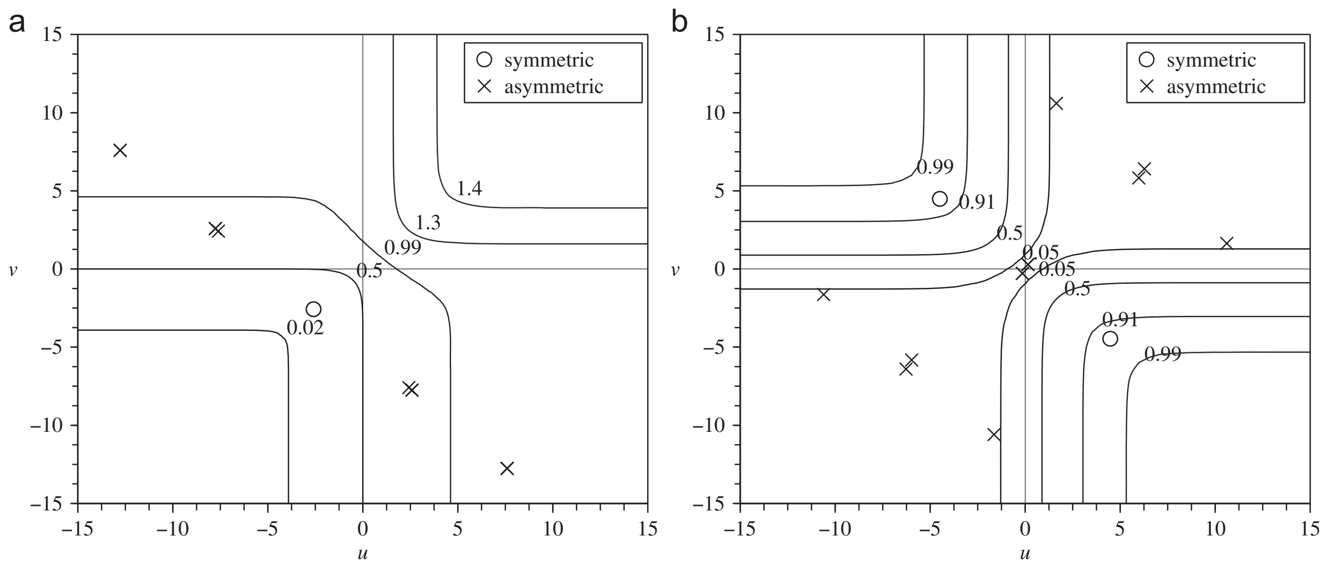


Fig. 5. Net-inputs of a single CVN for sixteen possible input patterns in the four-input symmetry detection problem on the contour diagrams of (a) activation function 1, and (b), activation function 2.

Solutions for the four-input problem obtained with activation functions 1 and 2 are shown in the contour graphs (a) and (b) in Fig. 5, respectively. Though activation function 1 saturates in four regions, we can expect that three regions will be used because the target output values are set to 0 and 1. The other region for which outputs are near $\sqrt{2}$ will not be used. As expected, experimental results presented in Fig. 5(a) showed that the net-inputs for 16 different input patterns were distributed over 3 regions, according to the symmetry and asymmetry, after the training. For activation function 2, on the other hand, all four regions of saturation can be used, and the experimental results in Fig. 5(b) shows that the input patterns were distributed in four regions according to the symmetry and asymmetry.

We then reversed the target output settings for both the activation functions. That is, with activation function 1, the target was set to 1 if the input pattern was symmetric, and 0 if asymmetric, while with activation function 2, the target was set to 0 if symmetric, and 1 if asymmetric. In these settings, the CVN with activation function 1 could solve the symmetry detection problem up to the length of 3 bits, and with activation function 2 up to 5 bits. This degraded result can be explained by the imbalance in number of input patterns assigned to each of the output values. That is, as the length of inputs increases, the percentage of symmetric inputs among all the possible input patterns decreases. For example, there are 50% symmetric patterns for the symmetry detection problem of 3 bits, while only 6.25% and 3.125% for that of 9 and 10 bits, respectively. The detailed explanation is given below.

Nonlinearity comes from the fact that input patterns having numerically very different values, and being spatially far different in location belong to the same class. When the same weight vector is multiplied with different input vectors of the same class, the weighted sum is different. The activation function 1 has two regions of saturation for the target value of 1 and only one region for the target value of 0 (Fig. 3(a)). Since most of the input patterns are asymmetric for higher dimensional problems having the target output 0, in case of reverse target setting, only one region is not enough to accommodate all the asymmetric input patterns having numerically very different values. With the other target setting, 0 for symmetry and 1 for asymmetry, activation function 1 could solve higher dimensional problems as there are two well spread regions for output values near the value of 1, and most of the input patterns are asymmetric having the target value of 1. The same argument can be applied to the CVN with the activation function 2. That is, in case of reverse target output settings (0 for symmetric and 1 for asymmetric), though there are two regions of saturation for the target output of 1, the regions are not as spread as the regions for 0, while most of the input patterns are asymmetric.

Table 4
Average number of misclassifications by two CVNs for symmetry detection problems

Length	Sym.[a]+Asym.[b]	Misclassified.[c]/total patterns	
		Activation 1	Activation 2
2	2+2	0/4	0/4
3	4+4	0/8	0/8
4	4+12	0/16	0/16
5	8+24	0/32	0/32
6	8+56	0/64	0/64
7	16+112	0/128	0.8/128
8	16+240	16/256	6.6/256
9	32+480	32/512	8.8/512
10	32+992	32/1024	32/1024

Averages are taken over 10 independent runs. Superscript letters a–c refer to the number of symmetric patterns, number of asymmetric patterns, and average number of misclassifications, respectively.

4.3.2. *Solving with two CVNs*

To solve the output encoding problem described above, one solution is to use two CVNs, each with the opposite target output settings. If the target value for one CVN is 1, then that of the other CVN is 0. Output class of a given input pattern is decided according to the CVN with the larger output. Results for using two CVNs for the symmetry detection problems are summarized in Table 4. Here the average of 10 independent runs is shown. Two CVNs with activation function 1 could solve the symmetry detection problem up to seven bits, and with activation function 2 up to 6 bits. The reason for these degraded results in comparison to using a single CVN is that although response from one of the CVN is near its target value, the output of the other neuron degrades the result more strongly. This can be seen from Table 5,

Table 5
Outputs of two CVNs with activation function 2 for the seven-input symmetry detection problem in a single run

Bit patterns	Neuron 1 output (target)	Neuron 2 output (target)	Bit patterns	Neuron 1 output (target)	Neuron 2 output (target)
0000000	0.942(1)	0.548(0)	1000000	0.000(0)	0.528(1)
0000001	0.000(0)	0.532(1)	1000001	0.942(1)	0.512(0)
0000010	0.000(0)	0.947(1)	1000010	0.000(0)	0.943(1)
0000011	0.000(0)	0.944(1)	1000011	0.000(0)	0.940(1)
0000100	0.033(0)	0.948(1)	1000100	0.000(0)	0.944(1)
0000101	0.000(0)	0.945(1)	1000101	0.033(0)	0.941(1)
0000110	0.038(0)	0.993(1)	1000110	0.000(0)	0.992(1)
0000111	0.000(0)	0.992(1)	1000111	0.038(0)	0.992(1)
0001000	0.942(1)	0.546(0)	1001000	0.000(0)	0.526(1)
0001001	0.000(0)	0.530(1)	1001001	0.942(1)	0.510(0)
0001010	0.000(0)	0.947(1)	1001010	0.000(0)	0.943(1)
0001011	0.000(0)	0.944(1)	1001011	0.000(0)	0.940(1)
0001100	0.033(0)	0.948(1)	1001100	0.000(0)	0.944(1)
0001101	0.000(0)	0.945(1)	1001101	0.033(0)	0.941(1)
0001110	0.038(0)	0.993(1)	1001110	0.000(0)	0.992(1)
0001111	0.000(0)	0.992(1)	1001111	0.038(0)	0.991(1)
0010000	0.033(0)	0.948(1)	1010000	0.000(0)	0.944(1)
0010001	0.000(0)	0.945(1)	1010001	0.033(0)	0.941(1)
0010010	0.000(0)	0.993(1)	1010010	0.000(0)	0.992(1)
0010011	0.000(0)	0.992(1)	1010011	0.000(0)	0.992(1)
0010100	**0.942(1)**	**0.993(0)**	1010100	0.000(0)	0.992(1)
0010101	0.000(0)	0.992(1)	**1010101**	**0.942(1)**	**0.992(0)**
0010110	0.000(0)	0.953(1)	1010110	0.000(0)	0.949(1)
0010111	0.000(0)	0.950(1)	1010111	0.000(0)	0.946(1)
0011000	0.033(0)	0.948(1)	1011000	0.000(0)	0.944(1)
0011001	0.000(0)	0.945(1)	1011001	0.033(0)	0.941(1)
0011010	0.000(0)	0.993(1)	1011010	0.000(0)	0.992(1)
0011011	0.000(0)	0.992(1)	1011011	0.000(0)	0.991(1)
0011100	**0.942(1)**	**0.993(0)**	1011100	0.000(0)	0.992(1)
0011101	0.000(0)	0.992(1)	**1011101**	**0.942(1)**	**0.992(0)**
0011110	0.000(0)	0.952(1)	1011110	0.000(0)	0.949(1)
0011111	0.000(0)	0.950(1)	1011111	0.000(0)	0.946(1)
0100000	0.000(0)	0.947(1)	1100000	0.000(0)	0.943(1)
0100001	0.000(0)	0.944(1)	1100001	0.000(0)	0.940(1)
0100010	**0.942(1)**	**0.992(0)**	1100010	0.000(0)	0.992(1)
0100011	0.000(0)	0.992(1)	**1100011**	**0.942(1)**	**0.991(0)**
0100100	0.000(0)	0.993(1)	1100100	0.000(0)	0.992(1)
0100101	0.000(0)	0.992(1)	1100101	0.000(0)	0.992(1)
0100110	0.033(0)	0.953(1)	1100110	0.000(0)	0.950(1)
0100111	0.000(0)	0.951(1)	1100111	0.033(0)	0.947(1)
0101000	0.000(0)	0.946(1)	1101000	0.000(0)	0.943(1)
0101001	0.000(0)	0.944(1)	1101001	0.000(0)	0.940(1)
0101010	**0.942(1)**	**0.992(0)**	1101010	0.000(0)	0.992(1)
0101011	0.000(0)	0.992(1)	**1101011**	**0.942(1)**	**0.991(0)**
0101100	0.000(0)	0.992(1)	1101100	0.000(0)	0.992(1)
0101101	0.000(0)	0.992(1)	1101101	0.000(0)	0.991(1)
0101110	0.033(0)	0.953(1)	1101110	0.000(0)	0.949(1)
0101111	0.000(0)	0.950(1)	1101111	0.033(0)	0.947(1)
0110000	0.038(0)	0.993(1)	1110000	0.000(0)	0.992(1)
0110001	0.000(0)	0.992(1)	1110001	0.038(0)	0.992(1)
0110010	0.033(0)	0.953(1)	1110010	0.000(0)	0.950(1)
0110011	0.000(0)	0.951(1)	1110011	0.033(0)	0.947(1)
0110100	0.000(0)	0.953(1)	1110100	0.000(0)	0.949(1)
0110101	0.000(0)	0.950(1)	1110101	0.000(0)	0.946(1)
0110110	0.942(1)	0.541(0)	1110110	0.000(0)	0.520(1)
0110111	0.000(0)	0.525(1)	1110111	0.942(1)	0.503(0)
0111000	0.038(0)	0.992(1)	1111000	0.000(0)	0.992(1)
0111001	0.000(0)	0.992(1)	1111001	0.038(0)	0.991(1)
0111010	0.033(0)	0.953(1)	1111010	0.000(0)	0.949(1)
0111011	0.000(0)	0.950(1)	1111011	0.033(0)	0.947(1)
0111100	0.000(0)	0.952(1)	1111100	0.000(0)	0.949(1)
0111101	0.000(0)	0.950(1)	1111101	0.000(0)	0.946(1)
0111110	0.942(1)	0.539(0)	1111110	0.000(0)	0.517(1)
0111111	0.000(0)	0.523(1)	1111111	0.942(1)	0.501(0)

where the result of a single run for the seven-bit symmetry detection problem with activation function 2 is given. Bold outputs denote the misclassification. Response of one of the neurons (neuron 2) degraded the result strongly even though the other neuron (neuron 1) gave the correct output.

Nevertheless, though limitations exist in the output assignment for the classes, a single CVN is at least more powerful than a single RVN as it can solve linear as well as nonlinear problems to some considerable extent. Especially, activation function 2 seems to have less output-assignment problem. The application of CVNN will be justified in the next section where we apply multiple CVNs on real world benchmark problems.

5. Single-layered complex-valued neural network (CVNN) on real world classification problems

Discussion in the previous sections is suggestive for exploring the ability of multiple CVNs, which we call the single-layered complex-valued neural network, on real world classification problems. To compare CVNN's performance with the conventional two-layered RVNN, both the neural networks are tested on some real world benchmark problems. We used Proben1 [14] datasets for the experiments. In this section, first we give a short description of the datasets, and then compare the learning convergence and generalization ability of CVNN and RVNN.

5.1. Description of the datasets

Proben1 is a collection of learning problems well suited for supervised learning. Most of the problems are classification problems regarding diagnosis. In our study, we used five of them. Each of the problems is available in three datasets, which vary in the ordering of examples. For example, the *cancer* problem consists of *cancer1*, *cancer2* and *cancer3* datasets. Short descriptions of these five problems are given below.

5.1.1. Cancer

This is a problem on diagnosis of breast cancer. The task is to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. Input attributes are for instance the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei. The data are comprised of 9 inputs, 2 outputs and 699 examples, of which 65.5% are benign examples. These data were originally obtained from University of Wisconsin Hospitals, Madison, from Dr. Wolberg and Mangasarian [17].

5.1.2. Card

The task is to predict the approval or non-approval of a credit card to a customer. Each example represents a card application and the output describes whether the bank (or similar institution) granted the card or not. There are 51 inputs, 2 outputs, and 690 examples.

5.1.3. Diabetes

This is on diagnosis of diabetes of Pima Indians. The task is to decide whether a Pima Indian individual is diabetes positive or not, based on personal data (age, number of times pregnant) and the results of medical examination (e.g. blood pressure body mass index, result of glucose tolerance test, etc.). It has 8 inputs, 2 outputs, and 768 examples.

5.1.4. Glass

Here the task is to classify glass types. The results of a chemical analysis of glass splinters (percent content of eight different elements) plus the refractive index are used to classify the samples to be either float processed or non-float processed building windows, vehicle windows, containers, tableware, or head lamps. This task is motivated by forensic needs in criminal investigation. The data have 9 inputs, 6 outputs, and 214 examples.

Table 6
Number of parameters for single-layered CVNNs and two-layered RVNNs

Dataset			CVNN	RVNN	
Name	Input	Output	Parameters	Hidden units	Parameters
Cancer	9	2	40	3	38
Card	51	2	208	4	218
Diabetes	8	2	36	3	35
Glass	9	6	120	7	118
Soybean	82	19	3154	31	3181

Parameters include the connection weights and the biases of the neurons. For CVNN, each weight and bias are counted as two parameters, since they have real and imaginary parts.

5.1.5. Soybean

The task is to recognize 19 different diseases of soybeans. The discrimination is performed based on a description of bean (e.g. whether its size and color are normal), the plants (e.g. the size of spots on the leafs, whether these spots have a halo, whether plant growth is normal, whether roots are rotted), and information about the history of the plant's life (whether changes in crop occurred in the last year or last 2 years, whether seeds were treated, the environment temperature). There are 35 inputs, 19 outputs, and 683 examples in the data.

5.2. Experimental setup

All the datasets discussed above contain real-valued data. The inputs were phase encoded as discussed in Section 2.1. Class labels were encoded by 1-of-n encoding for n-class problem using output values 0 and 1. To determine the class, a *winner-takes-all* method was used, i.e., for a given input pattern, the output neuron with the highest activation designated the class. Regarding the architecture of neural networks, the single-layered CVNN consisted of n CVNs, while two-layered RVNN architecture was selected in such a way that the number of parameters for both the CVNN and RVNN were nearly equal. For CVNNs, each complex-valued weight was counted as two parameters since it has real and imaginary parts. Table 6 shows the number parameters for the CVNNs and the RVNNs used in the experiments.

The backpropagation algorithm was used to train the RVNN, while the learning rules discussed in Section 3 were used to train the CVNN. During the training, we kept the learning rate at 0.1 for both the networks. Weights and biases (in case of CVNN, real and imaginary parts) were initialized with random numbers from a uniform distribution in the range of $[-0.5, 0.5]$.

We used the partitioning of the dataset into training, validation, and test sets as given in Proben1 [14]. The partitioning is shown in Table 7. We trained both the networks over the training set for 1000 epochs, except for the *Glass* and *Soybean* problems. During these 1000 epochs, the weight parameters with the minimum mean squared error (MSE) over the validation set, was stored and applied on the test set. We calculated this validation error by using the same formulation as Eq. (16), except with the validation set instead of training set. Validation error was measured after every five epochs during the training process. For the *Glass* problem we trained the RVNN up to 5000 epochs, and for the *Soybean* 4000 epochs. CVNN was trained up to 1000 epochs for all the datasets.

5.3. Learning convergence

When a neural network is trained with input patterns, the error on the training set decreases gradually with the epochs. We used the following formula to measure the mean squared training error over an epoch:

$$E = \frac{1}{2P}\sum_{p=1}^{P}\sum_{i=1}^{n}(d_{pi} - y_{pi})^2 \qquad (16)$$

where p is the pattern number, P the total number of training patterns, and n is the number of output neurons.

Fig. 6 shows how the mean squared training error changed over the epochs for the two-layered RVNN, and the single-layered CVNN with both the proposed activation functions on the *Diabetes* dataset. For both the activation functions, CVNN converged much faster than the RVNN. Especially, CVNN with activation function 2 converged faster than with activation function 1. In the following section, we show it quantitatively for all the datasets used in the experiments.

5.4. Results

Here we compare the generalization ability of the single-layered CVNN and two-layered RVNN on the basis of misclassification on test dataset. We also compare the number of epochs required to reach the minimum MSE during the training over validation set. Table 8 summarizes the results. Results were taken over 20 independent runs for each of the datasets.

As shown in Table 8, the test set errors obtained with activation function 1 tended to be more or less similar to those with activation function 2. For six datasets, *Cancer3*, *Diabetes2*, *Glass1*, *Glass2*, *Glass3* and *Soybean1*, the average classification errors of CVNN were a little bit less than those of the RVNN, while for the others a little bit more than those of the RVNN. Single-layered CVNNs, with both the activation functions, thus exhibit comparable generalization ability to that of the RVNNs.

Regarding the learning convergence, single-layered CVNNs required far less training cycles (epochs), in almost all the cases, to reach the minimum validation error than the RVNN counterpart as can be seen in Table 8. In other words, learning convergence of CVNN is faster than that of the RVNN. CVNN with activation function 2 seem to have even faster convergence than that of activation function 1 in most of the cases.

In Section 2.1, we mentioned that both the intervals $[0, \pi]$ and $[0, \pi/2]$ may be used for the phase encoding of real-valued inputs. Table 9 shows the number of epochs to reach the minimum validation error in both the encoding schemes, with each of the activation functions. As is apparent from the table, the learning convergence is faster for the interval $[0, \pi]$ in most cases. The difference in generalization ability (test set error) was insignificant and thus the results are not shown.

Table 7
Total number of examples, and partitioning of the datasets into training, validation and test sets

Dataset	Total patterns	Training set	Validation set	Test set
Cancer	699	350	175	174
Card	690	345	173	172
Diabetes	768	384	192	192
Glass	214	107	54	53
Soybean	683	342	171	170

6. Discussion

Two new activation functions for the CVN have been proposed in this study. These are suitable for applying the CVNN to real-valued classification problems as well as for deriving a gradient-based learning algorithm. It is shown that encoding real-valued inputs into phases, and processing information in complex domain, improves classification ability of a neuron. Capability of the proposed CVN is illustrated by solving two-input Boolean functions, three-input Boolean functions, and the symmetry detection in binary sequences of various lengths. A single-layered CVNN consisted of multiple CVNs with the decision making in a *winner-takes-all* fashion is applied to real world benchmark problems.

We explained graphically the nature of the proposed activation functions. Both the activation functions have four regions of saturation. We exploited three regions for activation function 1, and all four regions for activation function 2, since target outputs were given either 0 or 1. Such saturation regions improve the classification ability of a CVN. Michel et al. [9] have reported that their model is able to solve 245 out of 256 three-input Boolean

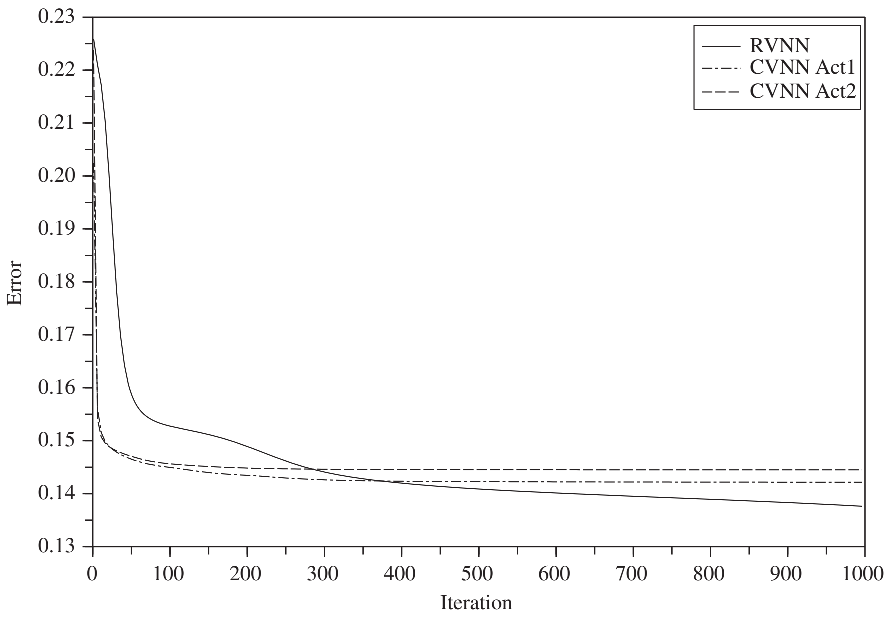


Fig. 6. Learning processes of the two-layered RVNN, and single-layered CVNNs with activation function 1 (CVNN Act1) and activation function 2 (CVNN Act2).

Table 8
Average classification error on the test set, and the average number of epochs required to reach the minimum validation error

Dataset	Test set error ± S.D.			Epoch at minimum validation error		
	RVNN	CVNN		RVNN	CVNN	
		Act. 1	Act. 2		Act. 1	Act. 2
Cancer1	1.44 ± 0.29	2.56 ± 0.29	2.50 ± 0.28	133.25	30.00	23.25
Cancer2	3.71 ± 0.35	4.37 ± 0.29	4.48 ± 0.30	69.00	21.25	16.25
Cancer3	4.48 ± 0.35	4.02 ± 0.00	4.02 ± 0.00	69.25	27.25	20.25
Card1	14.48 ± 0.75	14.94 ± 0.80	14.80 ± 0.61	30.75	7.25	6.00
Card2	17.94 ± 0.76	17.82 ± 1.07	17.79 ± 1.17	16.25	4.50	6.25
Card3	19.13 ± 0.96	18.98 ± 1.20	19.16 ± 1.58	33.75	10.25	10.00
Diabetes1	23.72 ± 0.60	25.60 ± 0.39	25.57 ± 0.50	630.25	83.75	40.00
Diabetes2	26.17 ± 0.84	23.78 ± 0.80	24.71 ± 0.85	313.75	94.50	45.50
Diabetes3	21.90 ± 0.64	21.28 ± 0.76	21.02 ± 0.59	416.25	7.00	6.00
Glass1	35.28 ± 5.38	32.64 ± 1.95	29.72 ± 3.67	4590.50	194.00	131.25
Glass2	53.87 ± 1.67	46.79 ± 6.05	44.72 ± 5.61	217.50	388.75	110.25
Glass3	50.38 ± 7.62	39.06 ± 5.71	40.75 ± 5.21	4195.25	311.00	131.50
Soybean1	10.62 ± 1.23	8.29 ± 0.44	9.32 ± 1.62	3545.50	220.75	209.75
Soybean2	7.44 ± 1.52	6.88 ± 0.57	8.47 ± 2.55	3452.50	299.75	155.00
Soybean3	8.65 ± 1.54	9.68 ± 1.22	10.18 ± 2.50	3122.25	140.00	116.25

Averages are taken over 20 independent runs. Act. 1 and 2 refer to activation function 1 and activation function 2, respectively.

Table 9
Average number of epochs required to reach the minimum validation error for the phase encoding $[0, \pi/2]$ and $[0, \pi]$

Dataset	Activation 1		Activation 2	
	$\pi/2$	π	$\pi/2$	π
Cancer1	61.25	30.00	42.00	23.25
Cancer2	39.50	21.25	27.75	16.25
Cancer3	50.75	27.25	38.25	20.25
Card1	12.50	7.25	7.75	6.00
Card2	6.00	4.50	6.00	6.25
Card3	26.50	10.25	27.50	10.00
Diabetes1	900.50	83.75	881.50	40.00
Diabetes2	996.00	94.50	567.25	45.50
Diabetes3	19.00	7.00	12.50	6.00
Glass1	743.75	194.00	688.75	131.25
Glass2	310.50	388.75	334.75	110.25
Glass3	474.25	311.00	461.25	131.50
Soybean1	376.00	220.75	400.50	209.75
Soybean2	382.75	299.75	325.00	155.00
Soybean3	215.00	140.00	259.50	116.25

Averages are taken over 20 independent runs.

functions, while the proposed CVN could solve 253 (out of the 256) functions. Solving the symmetry detection problem in binary sequences by a CVN up to six bits has been reported by Nitta [12]. We showed here that the proposed CVN could solve up to 10 bits. However, there is a limitation of output encoding for high dimensional symmetry detection problems. Using two CVNs, this problem can be solved to a considerable extent.

The single-layered CVNN is much faster in learning than the RVNN in terms of number of epochs for reaching the minimum validation error. Generalization ability of the single-layered CVNN is comparable to two-layered RVNN on real world benchmark problems. One advantage of using such a single-layered CVNN instead of RVNN for the classification problems is that the architecture selection problem does not exist in the case of the single-layered CVNN.

Though there is no complex-valued activation function having differentiability and boundedness at the same time in the entire complex domain, one can treat real and imaginary parts of net-input of a CVN individually, and then can combine them meaningfully. Also making the dimensionality high (for example, quaternions) may be a new direction for enhancing the ability of neural networks. We think solving real-valued problems more efficiently by the CVNN deserves further research.

Acknowledgement

This study was supported by grants to K.M. from Japanese Society for promotion of Sciences and Technology, Yazaki memorial foundation for Science and Technology, and the University of Fukui.

References

[1] I. Aizenberg, C. Moraga, Multilayer feedforward neural network based on multi-valued neurons and a backpropagation learning algorithm, Soft Comput. 11 (2) (2007) 169–183.
[2] H. Aoki, Application of complex-valued neural networks for image processing, in: A. Hirose (Ed.), Complex-Valued Neural Networks: Theories and Applications, vol. 5, World Scientific Publishing, Singapore, 2003, pp. 181–204.
[3] G. Georgiou, C. Koutsougeras, Complex backpropagation, IEEE Trans. Circuits Syst. II 39 (5) (1992) 330–334.
[4] A. Hirose, Applications of complex-valued neural networks to coherent optical computing using phase-sensitive detection scheme, Inf. Sci.—Appl. 2 (2) (1994) 103–117.
[5] A. Hirose, Complex-Valued Neural Networks, Springer, Berlin, 2006.
[6] K. Hornik, M. Stinchombe, H. White, Multilayer Feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.
[7] T. Kim, T. Adali, Fully complex multilayer perceptron for nonlinear signal processing, J. VLSI Signal Process. Syst. Signal Image Video Technol. Special Issue: Neural Networks Signal Process. 32 (2002) 29–43.
[8] H. Leung, S. Haykin, The complex backpropagation algorithm, IEEE Trans. Signal Process. 39 (9) (1991) 2101–2104.
[9] H.E. Michel, A.A.S. Awwal, Enhanced artificial neural networks using complex-numbers, in: Proceedings of IJCNN'99 International Joint Conference on Neural Networks, vol. 1, 1999, pp. 456–461.
[10] T. Nitta, An extension of the backpropagation algorithm to complex numbers, Neural Networks 10 (8) (1997) 1391–1415.
[11] T. Nitta, On the inherent property of the decision boundary in complex-valued neural networks, Neurocomputing 50 (C) (2003) 291–303.
[12] T. Nitta, Solving the XOR problem and the detection of symmetry using a single complex-valued neuron, Neural Networks 16 (8) (2003) 1101–1105.
[13] T. Nitta, Three-dimensional vector valued neural network and its generalization ability, Neural Inf. Process.—Lett. Rev. 10 (10) (2006) 237–242.
[14] L. Prechelt, PROBEN1—A Set of Benchmarks and Benchmarking Rules for Neural Network Training Algorithms, University of Karlsruhe, Germany, Available FTP: ⟨ftp.ira.uka.de/pub/neuron/proben1.tar.gz⟩.
[15] A.B. Suksmono, A. Hirose, Adaptive interferometric radar image processing by complex-valued neural networks, in: A. Hirose (Ed.), Complex-Valued Neural

Networks: Theories and Applications, vol. 5, World Scientific Publishing, Singapore, 2003, pp. 277–301.
[16] B. Widrow, J. McCool, M. Ball, The complex LMS algorithm, Proc. IEEE 63 (4) (1975) 719–720.
[17] W.H. Wolberg, O.L. Mangasarian, Multisurface method of pattern separation for medical diagnosis applied to breast cytology, Proc. Natl. Acad. Sci. 87 (1990) 9193–9196.

Md. Faijul Amin is currently a master course student at the Department of Human and Artificial Intelligence Systems, University of Fukui, Japan. He received B.Sc. in computer science and engineering at Bangladesh University of Engineering and Technology (BUET) in 2004, and joined as a lecturer at the Department of Computer Science and Engineering of Khulna University of Engineering and Technology (KUET). He is a member of the Institute of Engineers in Bangladesh (IEB). His research interest includes artificial neural networks and its mathematical foundations.

Kazuyuki Murase is a professor at the Department of Human and Artificial Intelligence Systems, Graduate School of Engineering, University of Fukui, Japan, since 1999. He received M.E. in electrical engineering from Nagoya University in 1978, Ph.D. in biomedical engineering from Iowa State University in 1983. He Joined as a research associate at Department of Information Science of Toyohashi University of Technology in 1984, as an associate professor at the Department of Information Science of Fukui University in 1988, and became professor in 1992. He is a member of The Institute of Electronics, Information and Communication Engineers (IEICE), The Japanese Society for Medical and Biological Engineering (JSMBE), The Japan Neuroscience Society (JSN), The International Neural Network Society (INNS), and The Society for Neuroscience (SFN). He serves on the Board of Directors of Japan Neural Network Society (JNNS), as a Councilor of Physiological Society of Japan (PSJ) and as a Councilor of Japanese Association for the Study of Pain (JASP).