Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Data Mining (CIDM 2007)

A Novel Complex-Valued Counterpropagation Network

Prem K. Kalra, Deepak Mishra & Kanishka Tyagi
Department of Electrical Engineering
Indian Institute of Technology Kanpur, India
E-mail: dkmishra@iitk.ac.in, kanishkaugee@gmail.com, kalra@iitk.ac.in

Abstract— **The Counterpropagation network is a combination of competitive network (Kohonen layer) and Grossberg outstar structure. In this paper we have proposed a complex valued representation on conventional forward only counterpropagation network. Many researchers have investigated the computational capabilities of neuron models for real values only. The novel part of the paper is, while considering the complex values equal weightage is given to both the real and imaginary parts. A vectored approach is taken to compute the complex numbers while implementing it with complex valued counterpropagation network (CVCPN). The proposed network is tested on benchmark problem (two spiral problem), Julia's set, rotational transformations and color image compression. The complex valued counterpropagation network (CVCPN) exhibits less percentage of misclassification and error rate is considerably smaller when compared to the equivalent model in Backpropagation network. The learning of intermediate forms of vector classes, manipulation with complex numbers, criterion for winning neuron, and the results of the proposed network with various benchmark and classification problems are discussed.**

I. Introduction

Till now complex valued neural network was actively used in backpropagation algorithm with complex weights and complex valued neuron-activation functions, but this network suffers with many limitations like it is slow with a large set of data, lack of bounded and analytic complex nonlinear activation functions in complex plane[1]. Several approaches have been suggested to process the complex data using backpropagation algorithm [2]. Due to strong power of generalization and simplicity in calculations, Counterpropagation network has an advantage over backpropagation network.

Here the complex number theory is successfully applied on forward only Counterpropagation network. Instead of linear reduction in kohonen layer's learning rate a new approach of exponentially reducing the learning rate have been used. Many algorithms have been developed in recent years that works on neural computations techniques with complex values [3]. Many of them have applied complex values on backpropagation algorithm using multilayer or multiplicative neurons [2]. The processing of complex numbers with Counterpropagation network is another plausible approach, which we have explored in this paper.

Section II, describes the topology, training rules of the Counterpropagation network. Section III, describes the mathematical approach for sthe complex-valued Counterpropagation network (CVCPN) and the algorithm used. Section IV, shows the experimental results obtained from the proposed network. Section V, summarizes on the proposed model as well as the possible future work.

II. Background

A. Forward-Only Counterpropagation Network

The network was introduced by Hecht-Nielson(1987,88)[4]. It is a combination of Kohonen network which clusters training vectors by the adaptive vector quantization and approximates the centroid of each vector class in a self organizing fashion [5][6] and Grossberg outstar network which adapt the associated vectors through a supervised training method [4][7]. *m*, the output vector *y* of the Kohonen layer becomes

$$[y_1,y_2...y_m]=[0\ 0\ ...1]$$

Such a response is generated as a result of lateral inhibitions within the layer which needs to be activated during recall mode [8]. The second layer is called Grossberg layer due to its *outstar learning mode* (Grossberg 1974,1982). The Grossberg layer with weights v_{ij}, functions in the following manner:

$$z= \Gamma(Vy)$$

With diagonal elements of the operator Γ being a sgn function operating component wise on entries of the vector *Vy*. The supervised learning rule for this layer is similar to *outstar learning rule*.

B. Weight Updation Rules

Kohonen weight updation: When an input vector *x* is applied to the input layer, the unit which has the closest weight vector to the input vector is defined as the winner unit. Generally euclidean distance is used as the criterion for deciding the winner unit. For a particular i^{th} hidden neuron,given a set of input value presented to input layer, $neth_i$ is calculated as follow:

$$neth_i = \sum_{m=1}^{n} \|x_m - w_{im}\| \tag{1}$$

where $\|..\|$ is the euclidean distance between two points. *n* is the total number of input neurons. Let the winning unit in the

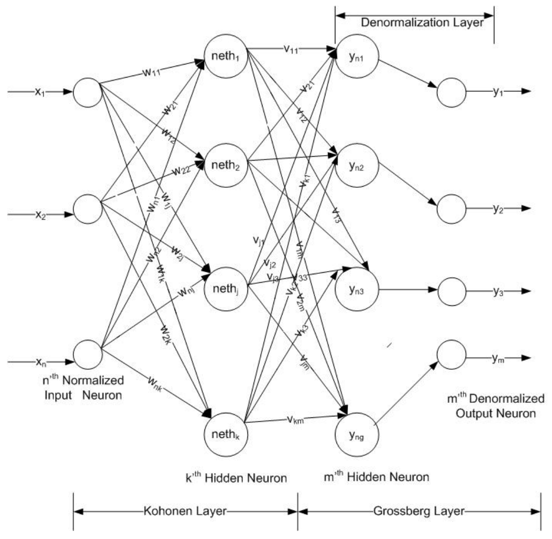

Fig. 1. Counterpropagation Network

hidden layer be the j^{th} neuron then the weight vector of the winner unit is updated by the following rule:

$$w_j(t+1) = w_j(t) + \alpha(x - w_j(t)) \tag{2}$$

where α is the learning rate which decreases with the number of iterations The Kohonen weights are adjusted till stable clusters are formed. The set of input vectors are thus classified by this learning [9].

Grossberg weight updation: After the above Kohonen weight updation, the weight vector w_j is kept fixed and the Grossberg weight vector v_j between competitive layer and output layer are adjusted by again applying the input vector to the input layer. For i^{th} hidden neuron, *neth*$_i$ is again calculated using

$$neth_i = \sum_{m=1}^{n} \|x_m - w_{im}\| \tag{3}$$

n is the total number of input neurons.When all *X* have been fed into input layer then winner is decided using minimum distance criterion.
Update v_j for only those which are connected to the winning neuron and the output, according to

$$v_j(t+1) = v_j(t) + \beta(y - v_j(t)) \tag{4}$$

Here *y* is the desired output, *j* is the winning hidden neuron. β is increased with the number of iterations.
Above steps are repeated till desired convergence criterion is met.

III. The Proposed Model

In its simplest version, CPN is able to perform vector to vector mapping similar to heteroassociative memory networks. This actually gave us the clue to treat complex numbers in vectored manner and give real and imaginary part equal significance thus following a real-valued approach. The weights used in the network are also taken in complex form.

A. Network Architecture

The input layer contain *n* cells for distributing training vector *x*, Kohonen layer with *k* cells produces output *neth*. This *neth* being a real number is compared with other *k-1* cells and minimum *neth* is declare as winner. A *fan-in* structure, from all input cells is thus created to the winning cell in Kohonen layer. During training stage each complex value is fed into Kohonen network for the self organizing classification. The classification is done separately on real and imaginary part of input values by a defined algorithm(described in the next sub-section).

B. Training of input vector in Kohonen layer

We assume that $w^r(t)$ and $w^i(t)$ are the real and imaginary components of Kohonen layer weight vector. After updation they become $w^r(t+1)$ and $w^i(t+1)$ respectively. Then,

$$w^r(t+1) = w^r(t) + \alpha(x^r - w^r(t)) \tag{5}$$
$$w^i(t+1) = w^i(t) + \alpha(x^i - w^i(t)) \tag{6}$$

The network is trained until there is no significant change in the updated weight and the old weight i.e

$$w^r(t+1) = w^r(t) \tag{7}$$
$$w^i(t+1) = w^i(t) \tag{8}$$

In that case the real and imaginary value of the input vector is copied on to its corresponding weight vector. When this situation arises the training is said to be completed and this is the stopping condition for the network. The weight vectors have now settled near the centroid of each cluster. These clusters are different for real and imaginary parts. The architecture of the proposed model is similar to the conventional CPN network. The criterion for deciding the winning element is an important factor and needs an explanation here.

In competitive stage two other criterion for finding *neth* in Kohonen layer were tested:
1) Selecting by dot product

$$neth_i = min[\sum_{k=1}^{n}(x_k w_{ki})] \tag{9}$$

2) Converting Cartesian into Polar coordinates. We convert the cartesian form of x_k=($x_k{}^r$,$x_k{}^i$) into polar form (r_1,Θ_1) and similarly for w_{kj}=($w_{kj}{}^r$,$w_{kj}{}^i$) into polar form (r_2,Θ_2).*neth* is calculated as:

$$neth_i = min[\sqrt{r_1^2 + r_2^2 - 2r_1r_2cos(\Theta_1 - \Theta_2)}] \tag{10}$$

where r_1,r_2,Θ_1,Θ_2 are the polar coordinates of the input complex number and the connected weight between hidden unit and the input neuron.

Out of the above the dot product does not give satisfactory results with non-binary values, converting cartesian to polar coordinates does give results with non binary values but requires large training time and number of hidden neuron. Therefore Euclidean distance method or the nearest neighbor method is found to be most suitable. The reason for taking the minimum distance (and not the maximum) is that the distance which is minimum is closest to a particular cluster, we then update the weights in such a manner so as to bring it more closer to the cluster thus moving that value to the centroid of the cluster. We know that if x_1 and x_2 are two complex vectors, then the Euclidean distance between them is given by

$$|x_1 - x_2| = \sqrt{({x_1}^r - {x_2}^r)^2 + ({x_1}^i - {x_2}^i)^2} \tag{11}$$

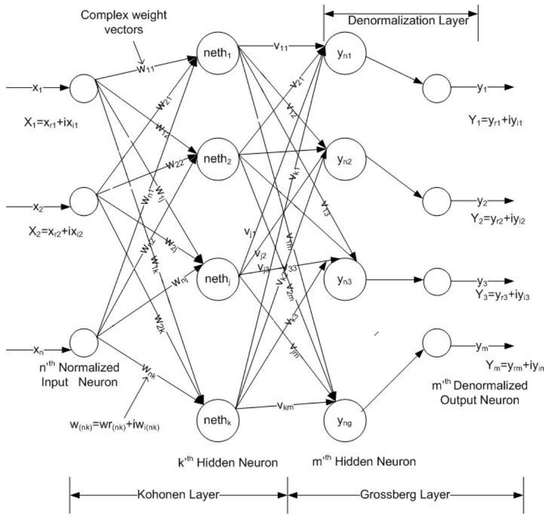

Fig. 2. Forward-Only Complex-Valued Counterpropagation Network

If *n* be the total input neurons and *k* be the total number of hidden layer neurons, then let x_1,x_2...x_n are the complex input given in the Instar layer. w^r_{11},w^r_{12}...w^r_{1k} are the real and w^i_{11},w^i_{12}...w^i_{1k}be the imaginary components of the weights connecting the input x_1 and the $k^{'th}$ neurons of Kohonen layer, similarly other weights are connecting input neuron and the hidden layer. The Euclidean distance between x_1 and w_{11}them is :

$$x_1=x^r_1+\mathrm{i}x^i_1$$
$$w_{11}=w^r_{11}+\mathrm{i}w^i_{11}$$
$$x_1\text{-}w_{11}=(x^r_1\text{-}w^r_{11})+\mathrm{i}(x^i_1\text{-}w^i_{11})$$
$$|x_1 - w_{11}|=\sqrt{(x^r_1 - w^r_{11})^2 + (x^i_1 - w^i_{11})^2}$$

This is the Euclidean distance between any two points in a complex plane. For a particular set of values of input vectors x_1,x_2,x_3...x_u...x_n we calculate the following real valued quantity,

$$neth_1 = \sum_{u=1}^{n} \sqrt{[(x^r_u - w^r_{u1})^2 + (x^i_u - w^i_{u1})^2]} \tag{12}$$

Similarly,

$$neth_2 = \sum_{u=1}^{n} \sqrt{[(x^r_u - w^r_{u2})^2 + (x^i_u - w^i_{u2})^2]} \tag{13}$$

Generalizing it we get;

$$neth_k = \sum_{u=1}^{n} \sqrt{[(x^r_u - w^r_{uk})^2 + (x^i_u - w^i_{uk})^2]} \tag{14}$$

The minimum of all *neth* ($neth_1$,$neth_2$....$neth_k$) is calculated and is considered as winner for that set of values of *x*. Figure

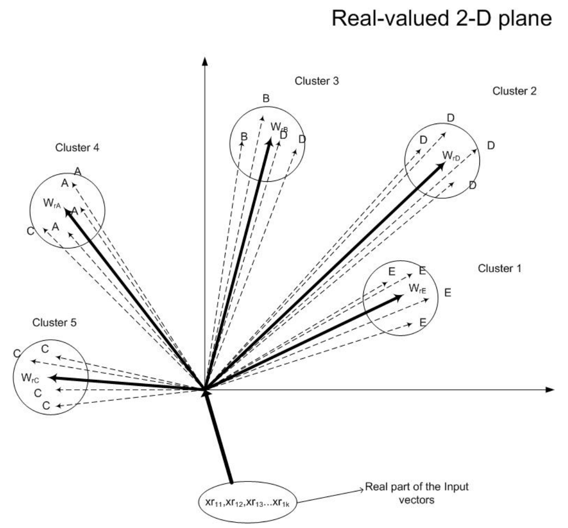

Fig. 3. Clustering process for real part of Input vectors

3 gives a pictorial representation of how the clustering process take place after the complete training of network. For the sake of simplicity we have shown clustering only for the real part of input vectors, similar clustering takes place for the imaginary part also. Here A,B...E are the input vectors (real numbers) and W_{rA},W_{rB}...W_{rE} are the real components of weight vectors. It is only after the training that these weight vectors are oriented at the center of the circle. This is what we mean when we say that the weights are adjusted to the centroid of the cluster. The bold line from origin to each cluster is the weight vector and is the mean of all values of the vectors coming in that circle. Observe that cluster 3 is not properly clustered as orientation of vector D (which should be in cluster 2) is into this cluster. Same is the case with cluster 4 where vector C is wrongly directing towards cluster 4. This is a practical situation and is the cause of error and misclassification. This can be removed by either with more training or proper selection of α and β.

An important point to be noted here is that the disperancies as whether the number will lie in 1,2,3,4 quadrants are removed. That is to say that we'll get a unique Euclidean distance even if the magnitude of number is same. Therefore we have spread out the clusters and now they are not confined in a single quadrant. This is the advantage of using a vectored approach, that we have implemented in our paper.

C. Training the Outstar Layer

Once the training in Kohonen layer is completed and all the input vector have been clustered (both real and imaginary) the vector *x* now makes a *fan out* connection from Kohonen layer to the Grossberg layer and only weights in the Grossberg layer are updated which are connected to the winning neuron. After the training stage, output of the CVCPN are separate in real and imaginary parts. They are again combined after the output layer to give the complex output.

If each input vector in a cluster maps to a different output vector, then the outstar learning procedure will enable the outstar to reproduce the average of those output vectors when any member of the class is presented to the inputs of the CVCPN. A stuck vector problem is seen if we generate the weights in the range of [-1 1]. Also to avoid the condition of orthogonality of randomly generated weight vectors we have used weight vectors in the range of [0 1]

D. Algorithm for training Complex-Valued Counterpropagation Network

Kohonen-layer training

The input data is first normalized (normalization is done on real and imaginary part separately). We have used linear normalization technique to normalize the input vector if they are not in the range of [-1,1]. If *x* is the unnormalized vector then its normalized form is given by

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{15}$$

where x_{max}, x_{min} are the maximum and minimum values of the input data presented to the network.
We have varied α exponentially from 0.9 to 0.1. The smaller the value of α, less it will go far from the centroid region of the cluster units.

Grossberg layer training

The weights connecting the winning hidden neuron and output neuron are updated.The previous weights of Kohonen layer are not updated at this stage. The learning rate parameter β is also exponentialy increased from 0.1 to 0.5 in steps of 0.01.

The reason why the paramenters α & β are exponentially changed is that the large number of values that we get as compared to when we use linear change in these learning parameters values. Due to these large number of values, the centroid of the data set can be precisely located (for Kohonen layer) and the centroidal values obtained can be accurately copied on the outpur layer (for grossberg layer).
We assume that,
Total number of input neurons=n
Total number of Kohonen layer neurons=k
Total number of Grossberg layer neurons=g

The formal algorithm is as follows:

Phase 1 :
1. Initialize weights $\in$(0,1) randomly.
2. **repeat**
Apply each of the normalized input to the competitive layer.
• **for** m=1,..., n, **do** :
(a) Calculate the distance between m^{th}input neuron and j^{th} kohonen weight vector according to:

$$neth_j = \|x_m - w_{mj}\| \tag{16}$$

where $\|\cdots\|$ represents the Euclidean distance and j=1,2,3...k.

$$neth_j = \sum_{m=1}^{n} \sqrt{[(x^r{}_m - w^r{}_{mj})^2 + (x^i{}_m - w^i{}_{mj})^2]} \tag{17}$$

$x^r{}_m$ is the real part of the m^{th} input vector, $w^r{}_{mj}$ is the real part of weight connecting the m^{th} input vector to the j^{th} kohonen layer neuron. $x^i{}_u$ and $w^i{}_{mj}$ are the imaginary components input and kohonen weight vector respectively.

(b) Find the winning cluster and call its index k^* such that $neth_k^* \leq neth_j$, j=1,...,k..

(c) Update the weights connecting the winning cluster unit and the input vector according to

$$\begin{aligned} w^r(t+1)_{mk^*} &= w^r(t)_{mk^*} + \alpha_t(x^r{}_m - w^r(t)_{mk^*}) \\ w^i(t+1)_{mk^*} &= w^i(t)_{mk^*} + \alpha_t(x^i{}_m - w^i(t)_{mk^*}) \end{aligned} \tag{18}$$

As done in conventional algorithms of CPN, here there is no need to set the output of cluster or hidden units as 1 or 0 depending on that minimum distance.

end-for;

• Learning rate α is decreased gradually during the training algorithm according to :

$$\alpha_t = \alpha_0 e^{-a/a_0} \tag{19}$$

• α_0 is set to 0.9, a_0 is taken as 10 and *a*= 1,2,..q. *a* is incremented after each set of input pattern is fed into the network;
until performance is satisfactory or computational bounds are exceeded;

Phase 2 :
1. Fix α at current level (from Phase 1) and initialize β;
2. **repeat**
• **for** m=1,..., n, **do** :

(a) Calculate the distance between m^{th}input vector and kohonen weight vector according to:

$$neth_j = \sum_{m=1}^{n} \sqrt{[(x^r{}_m - w^r{}_{mj})^2 + (x^i{}_m - w^i{}_{mj})^2]} \tag{20}$$

(b) Find a winning kohonen layer neuron with index l such that $neth_l \leq neth_j$, j=1,...,k.

(c) Adjust the outstar layer weights connecting the winning l^{th} neuron to the grossberg layer g^* according to :

$$\begin{aligned} v^r(t+1)_{lg^*} &= v^r(t)_{lg^*} + \beta_t({y^r}_{g^*} - v^r(t)_{lg^*}) \\ v^i(t+1)_{lg^*} &= v^i(t)_{lg^*} + \beta_t({y^i}_{g^*} - v^i(t)_{lg^*}) \quad (21) \\ g^* &= 1, 2, 3...g \end{aligned}$$

end-for;

• Decrease β, by a small constant in the same exponential manner as α;

until performance is satisfactory or computational bounds are exceeded.

E. Algorithm for testing or recalling from CVCPN

The CVCPN functions in the recall mode as a nearest match look-up table. The difference from the usual table look-up is that the weight vector ar obtained by the training algorithm, rather than in an *adhoc* manner. The input vector x^r+ix^i finds the weight vector ${w^r}_m$+i${w^i}_m$ that is its closest match among *k* vectors available in the hidden layer.Then the outstar weights ${v^r}_m$+i${v^i}_m$ which are fanning out from the winning m^{th} Kohonen's neuron are updated. The output given by the network is the statistical averages of the similar kinds of input given in the training period.

Step 1: For each input value,repeat the step 2-4.
Step 2: Store the values of real and imaginary part of Grossberg weights that are connected to the hidden cluster unit and output neuron in different variables M and N.
Step 3: If the input was given in a normalized form de-normalize them,else skip this step.
Step 4: The vector M+iN is the desired output.

IV. Illustrative Examples

For each problem, Intel(M) (Celeron Mobile), 1.5 GHz, CPU with 256 MB RAM is used for simulation work with MATLAB as the simulation software. Training and testing data were normalized wherever necessary. It is observed that the proposed model exhibits a more efficient learning in each case. The number of misclassification is very less as compared to an equivalent model of complex valued approach using backpropagation algorithm[2]. It is due to the reason that in the Kohonen Layer of CVCPN unsupervised and in Grossberg Layer supervised learning is taking place. It is actually the combination of two independent layer of different learning rule that makes up the CPN and this basic structure is preserved in our model also.

A. Julia's Data Set

Julia's set are non-euclidean objects whose dimension can be any real number. A filled julia's set corressponding to the given values of the parameters a and b of the dynamical system is the set of initial conditions (x(0),y(0)) resulting in bounded system trajectories. The data comes from a set of sequence of complex numbers called mandelbrot set. The mandelbrot set of equations are taken as

$$\begin{aligned} z_{k+1} &= z_k^2 + c, \ [z_0 = 0 \ and \ c \in C] \\ z(k) &= x(k) + iy(k) \quad (22) \\ c &= a + ib \end{aligned}$$

Simplifying it we get

$$\begin{aligned} x(k+1) &= x^2(k)s - y^2(k) + a \quad (23) \\ y(k+1) &= 2x(k)y(k) + b \end{aligned}$$

We have tested our network with two modifications: The first method is, we take input & output of the 10 × 10 matrix into the single neuron. The results obtained are given below in Table I. We observe that the training and testing time is more but the number of misclassification is less. This is because due to the fact that since the single neuron is processing the whole data, so time taken will be more. But this form of network gives less number of misclassification.

To take as the first set of initial conditions we set the following values of

c=(a+ib)=0.2+i0.65
k=2
v=10

where:
c is a complex number used in the map f(z)=z^2 + c
k gives the number of iterations
v determines the number of points on the x-axis

TABLE I
Performance of single input CVCPN for c=0.2+i0.65, k=2, v=10

S. No.	Parameter	CVCPN
1	Number of hidden neurons	1000
2	Total Training and Testing time in minutes	2.36
3	Number of Parameters	60
4	Learning rate	(α) 0.9-0.1
		(β) 0.1-0.5
5	Misclassification	11

Another way of inputing the values is to take the 10 × 10 matrix in 10 neurons. One way of doing this is that we take ten input neurons and feed the 10 × by 10 matrix. But in that way we are actually training each neuron with only 10 input values. The results obtained are not good and misclassification are drastically increased. In our model we reshaped in a way that we take 2 neuron as input and train them with 50 neuron. In this the training set for each neuron is increased and the misclassification are reduced significantly. Same values of *c,k,v* are taken,and results are tabulated in Table II.

TABLE II
PERFORMANCE OF TWO INPUT CVCPN FOR C=0.2+I0.65, K=2, V=10

S. No.	Parameter	general CVCPN
1	Number of hidden neurons	500
2	Total Training and Testing time in minutes	1.10
3	Number of Parameters	60
4	Learning rate	(α) 0.9-0.1
		(β) 0.1-0.5
5	Misclassification	18

TABLE III
CVCPN PERFORMANCE FOR ROTATIONAL TRANSFORMATION

S. No.	Parameter	CVCPN
1	Number of hidden layer neuron	500
2	Training and Testing time (in sec)	22
3	Percentage Misclassification	0.52 %
4	Number of Parameters	31
5	Angle of rotation (in degree)	90
6	Learning rate	(α) 0.9-0.1
		(β) 0.1-0.5

B. Rotational transformation

Here our aim is to show that CVCPN is capable of generalizing a transformation of complex number, when treated as vectors. The original set of vector was transformed to 90^o clockwise. To convert the problem in complex domain the real part of the complex input is used as the x-coordinate of the data point and imaginary component as y-coordinate. Instead of choosing points on a line to train the network [10] [11], we trained it with points that are bounded within a rectangle of known dimensions i.e. the points lie in a 2 dimensional rectangular plane. These points which are complex in nature are rotated by 90^o clockwise, as a result the plane gets rotated by 90^o. This is the desired output and CVCPN is trained with these data sets obtained. In testing the network we presented a complex data set that resembled english alphabet 'T', when plotted. Figure 4 shows the testing figure used for transformation and the results obtained from the network. Circled dots are the desired output 'T' whereas the astrik dots is the 'T' learned by the network. Table III tabulates the performance of the network.

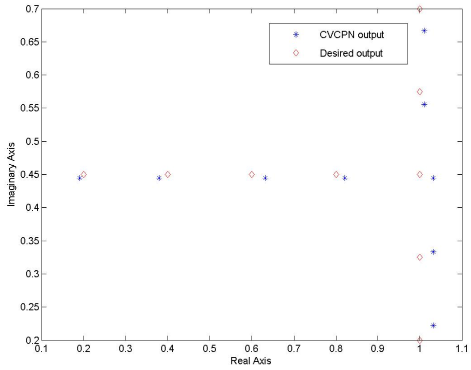

Fig. 4. Training data points and output results for Rotational Transformation

The time taken by the network to generalize the transformation is very less when compared to either of real and complex BPA.

C. Two Spiral Problem

The two spiral problem is found to be extremely tough classification and generalization problem for BPA family. The problem when solved by a standard BPA takes more than 4000 iterations with a target error of 0.0001 [12]. The problem is composed of two interwoven spirals in x-y plane, one training to be a member of Class A, the other to Class B. Although the classification is done in a highly non-linear region, the problem can easily be solved using self look-up table. The archimedean spirals are given by following polar equation:
$r=\Theta$
where r is the radius vector and Θ is the polar angle. In cartesian form we represent it by:
$x=\Theta\cos(\Theta+\Psi)$
$y=\Theta\sin(\Theta+\Psi)$
Ψ is the angle of rise of the spiral. In actual problem statement the two spiral were chosen to be π radians apart. As the neural network perform continuous mapping, the points in the vicinity of the spirals should also get correctly identified. Therefore we obtained the data set for spirals which are close to each other. Total of 194 I/O values are taken for training and testing the network. For uniformity in output result, 40-20-40 criterion is used i.e. an output is considered to be a logical zero if it is in the lower 40 percent of the output range, a one if it is in the upper 40 percent, and indeterminate (and therefore incorrect) if it is in the middle 20 percent of the range. We found that using CVCPN only 1 misclassification was obtained with its testing and training time is significantly reduced as tabulated in Table IV.

TABLE IV
CVCPN PERFORMANCE FOR TWO SPIRAL PROBLEM

S. No.	Parameter	CVCPN
1	Number of hidden layer neuron	500
2	Training and Testing time (in sec)	29
3	Percentage Misclassification	0.52 %
4	Number of Parameters	36
5	Learning rate	(α) 0.9-0.1
		(β) 0.1-0.5

D. Color Image Compression

Two major problems have been tackled while using the complex-valued CPN with image compression.
1) Mathematically any color image is a 3 dimensional matrix, and to process it in complex domain, we have to reduce it in either 2 or 1 dimension.

2) The 3 components associated with color image, red green and blue components, when viewed together impart color to any colored image. The problem that is faced is that all the three components have equal contribution in the final colored image. If the dimensionality reduction is done then it is difficult to decide which two components have to be merged and which component to leave intact so that it is processed directly with the network.

We took the advantage of the fact that human eye is more sensitive towards luminance of a image therefore we converted the image into YCbCr color space. In this space Y represents the luminance, Cb the blue component and Cr as the red component. Therefore the strategy we followed is that we combined the Cb and Cr components to a single component. It is then combined with the Y component and fed into the network in complex form. The output obtained is then broken down into the 3 components and converted back to RGB color space and displayed.

TABLE V
CVCPN PERFORMANCE FOR COLOR IMAGE COMPRESSION

S. No.	parameter	lena	mandril	text
1	Entropy (reconstructed image)	7.3216	5.2794	1.5623
2	Entropy Loss	0.0297	2.4073	0.5287
3	PSNR	28.9195	22.7347	18.9415
4	RMSE	9.1320	18.6124	28.8049
5	SNR	21.4591	11.8918	13.6010
6	Hidden neurons	64	128	8
7	Compression Ratio	4:1	2:1	8:1

V. Conclusion

The advantages of complex valued counterpropagation network are that the training rules used by CVCPN are not computationally complex when compared to the complex valued backpropagation network which are very costly in their computational resources. Though there have been a number of optimization methods invented but the overall complexity is still higher for backpropagation networks. CVCPN is fast in generalizing (rotational transformation problem), the number of misclassification is also very less as compared to its counterpart in real domain. This is because a self look up table approach is the basis of counterpropagation network which has been preserved in our network also.
It can be observed from the results that training and testing time is not so high and performance of the network is highly depended upon the winning criterion that we are taking.

Use of a more robust and faster algorithm that can do better clustering, spatial arrangement, adding a biasing or conscience to the network thus making use of dead or inactive neurons in the hidden layer, unsticking the stuck vector and use of other methods for dimensionality reduction like embedding magnitude of complex number in its phase are further areas that need to be explored with complex valued counterpropagation network.

References

[1] Silverman H., Complex variables, Houghton, Newark, USA, 1975.
[2] A.Yadav,D.Mishra,S.Ray and P.K. Kalra,"Representation of Complex-Valued Neural Networks:A Real-Valued Approach",IEEE Procc. on intelligent sensing and information processing, pp.331-335, 2005.
[3] Akira Hirose, "Complex-Valued Neural Network:Theories and Applications", World Scientific,2003.
[4] R.Hecht-Nielson,"Application of Counterpropagation Network", IEEE Trans.Neural Networks,vol 1,pp.131-141,1988.
[5] R.Hecht-Nielson,"Counterpropagation Networks", Applied Optics,vol.26, pp.4979-4984,1987.
[6] T.Kohonen,"Self Organization and Associative Memory", 2nd Ed., New York:Springler-Verlag,1988.
[7] G.Carpenter and S.Grossberg,"A massively parallel architecture for a self-organizing neural pattern recognition machine" Computer Verion,Graphics and Image Processing, vol.37, pp54-115, 1987.
[8] J.M.Zurada, "Introduction to Artificial Neural Systems", Jaico Publishing House,2005.
[9] J.A.Freeman and D.Skapura, "Neural Networks Algorithms,Applications and Programming Techniques", Addison Wesley Longman.
[10] T.Nitta,"An Extension of the Backpropagation Algorithm to Complex Numbers", Elsevier, Neural Networks, vol 10, no.8, pp.1391-1415, 1997.
[11] Kshitij Wat,"Complex Valued Based Neural Networks and Algorithms for their Implementation", M.Tech Thesis (2003), Department of Electrical Engineering, Indian Institute of Technology, Kanpur.
[12] A.Prashanth,"Investigation On Complex Variable Based Backpropagation Algorithm and Applications", Ph.D. thesis (2003), Department of Electrical Engineering, Indian Institute of Technology, Kanpur.