Single Layer Complex Valued Neural Network with Entropic Cost Function

Luís A. Alexandre

Dept. Informatics, Univ. Beira Interior
R. Marquês d'Ávila e Bolama, 6201-001 Covilhã
and IT - Instituto de Telecomunicações, Covilhã, Portugal
lfbaa@ubi.pt

Abstract. This paper presents the adaptation of a single layer complex valued neural network (NN) to use entropy in the cost function instead of the usual mean squared error (MSE). This network has the good property of having only one layer so that there is no need to search for the number of hidden layer neurons: the topology is completely determined by the problem. We extend the existing stochastic MSE based learning algorithm to a batch MSE version first and then to a batch minimum error entropy (MEE). We present experiments showing the the proposed algorithms are competitive with other learning machines.

Keywords: Complex valued NN, Entropic cost function, MSE, MEE.

1 Introduction

Complex valued neural networks (CVNNs) have been gaining considerable attention [1,2,3,4]. The benefits of using a complex valued NN come when dealing with specific types of data, such as wave phenomena [1, 5] where there is the need of processing phase and amplitude information.

The key feature of these networks is related to how the product of complex numbers work. Let's compare what happens if we consider a 2D input to a neuron in the following two cases: first, the traditional real valued case where the neuron has a weight associated with each input; second the complex value case where a single complex weight is used for a single complex valued input. The two cases are represented in figure 1. Consider the real numbers a, b, w_1 and w_2. In a real value neuron the 2D input consisting of values a and b gets multiplied by the respective weights w_1 and w_2 giving an input to the neuron of $aw_1 + bw_2$. In the case represented in the lower part of the figure, we have the same input values a and b but now as real and imaginary parts of a complex input $z = a+ib$. The weight are also part of a single complex weight $w = w_1+iw_2$. The neuron now sees this input as the product $zw = aw_1 - bw_2 + i(aw_2 + bw_1)$. If we write this result using amplitude and phase representation of complex numbers, say, $z = \sqrt{a^2 + b^2}e^{i\tan^{-1}(b/a)}$ and $w = \sqrt{w_1^2 + w_2^2}e^{i\tan^{-1}(w_2/w_1)}$, we get $zw = \sqrt{a^2 + b^2}\sqrt{w_1^2 + w_2^2}e^{i(\tan^{-1}(b/a)+\tan^{-1}(w_2/w_1))}$. This means that the product of complex numbers is really just multiplying the amplitudes and adding the phases.

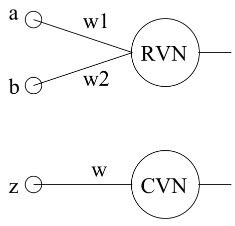

Fig. 1. Example of how a real valued (above) and a complex valued (below) neuron deal with a 2D input. $z = a + ib$ and $w = w_1 + iw_2$. See the text for details.

Traditionally, neural networks have used mean squared error as cost functions [6]. Recently however, it has been shown [7,8] that there may be advantages in using a cost function based on an information theoretical approaches, such as the minimization of the entropy of the errors (MEE). A concrete example is the increase in convergence during the training of recurrent NNs that were observed in [9,10].

In this paper we show a batch learning algorithm for the single layer complex value neural network proposed in [3] and proceed to derive the MEE based learning algorithm for this network.

The rest of the paper is organized as follows: the next section contains a presentation of the single layer complex valued NN, section 3 contains the derivation of the batch versions of the learning algorithm (with MSE and MEE); the following sections contains experiments and section 5 contains the conclusions.

2 Complex Valued NN

2.1 One Layer CVNN

In this subsection we follow closely [3].

Consider an input space with m features. The neuron input is the complex vector $x = x^R + ix^I$ where x^R is the real and x^I is the imaginary part, such that $x \in \mathbb{C}^m$. The complex unit is $i = \sqrt{-1}$. The weight matrix $w \in \mathbb{C}^m$ can also be written as $w = w^R + iw^I$. The net input of neuron k is given by

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

where $\theta_k \in \mathbb{C}$ is the bias for neuron k and can be written as $\theta_k = \theta_k^R + i\theta_k^I$.

Given the complex multiplication of $w_k x$, this can be further written as

$$z_k = z_k^R + iz_k^I = \left(\theta_k^R + x^R w_k^R - x^I w_k^I\right) + i\left(\theta_k^I + x^R w_k^I + x^I w_k^R\right) \tag{2}$$

Note that,

$$x^R w_k^R = \sum_{j=1}^{m} x_j^R w_{kj}^R \tag{3}$$

The k neuron output is given by $y_k = f(z_k)$ where $f : \mathbb{C} \to \mathbb{R}$ is the activation function and $y_k \in \mathbb{R}$. The activation function used is $f(z_k) = (s(z_k^R) - s(z_k^I))^2$ where $s(\cdot)$ is the sigmoid function $s(x) = \frac{1}{1+\exp(-x)}$.

Given the form of this activation function, it is possible to solve non-linear classification problems whereas in the case of a real valued neural network with only one layer (such as a simple perceptron) this would not be possible.

Now that we know how a single neuron obtains its output, we will see how to train a network composed of a single layer with N of these complex valued neurons.

To train the network in a stochastic learning approach we need to obtain the weights that minimize the following error functional

$$E(w) = \frac{1}{2}\sum_{k=1}^{N}(t_k - y_k)^2 \tag{4}$$

where $t_k \in \mathbb{R}$ represents the target output for neuron k. This is the mean squared error functional (MSE) that is traditionally used in the learning algorithm of NNs, only in this case it depends on a complex weight matrix, w.

To minimize (4) we find its derivative w.r.t. the weights:

$$\frac{\partial E}{\partial w_{kj}^R} = -2e_k(s(z_k^R) - s(z_k^I))\left(s'(z_k^R)x_j^R - s'(z_k^I)x_j^I\right) \tag{5}$$

The previous expression is the derivative w.r.t. the real weights but a similar one should be made w.r.t. the imaginary weights.

To obtain the weights we use the gradient descent rule, and update the weights at each iteration (t), that is, after the presentation of each training pattern to the network, using

$$w_{kj}^R(t) = w_{kj}^R(t-1) + \Delta w_{kj}^R(t) \tag{6}$$

with (gradient descent: go opposite to the derivative of E w.r.t. the weights)

$$\Delta w_{kj}^R(t) = -\eta\frac{\partial E}{\partial w_{kj}^R} = 2\eta e_k(s(z_k^R) - s(z_k^I))\left(s'(z_k^R)x_j^R - s'(z_k^I)x_j^I\right) \tag{7}$$

A similar derivation can be made for the case of the imaginary part of the weights, yielding

$$\Delta w_{kj}^I(t) = -\eta\frac{\partial E}{\partial w_{kj}^I} = 2\eta e_k(s(z_k^I) - s(z_k^R))\left(s'(z_k^R)x_j^I + s'(z_k^I)x_j^R\right) \tag{8}$$

It is possible to show that the final expressions for the adjustment of the real and imaginary parts of the bias are

$$\Delta\theta_k^R = 2\eta e_k(s(z_k^R) - s(z_k^I))s'(z_k^R) \tag{9}$$

and

$$\Delta\theta_k^I = 2\eta e_k(s(z_k^I) - s(z_k^R))s'(z_k^I) \tag{10}$$

3 Batch Learning

In this section we present the batch version of the algorithm presented in the previous section.

First change is on the functional that should be minimized: now it contains the error contributions from all the L patterns in the training set:

$$E(w) = \frac{1}{2L} \sum_{l=1}^{L} \sum_{k=1}^{N} (t_k - y_k)^2 \qquad (11)$$

The only difference to the stochastic approach presented earlier is that instead of updating the weights after each pattern is presented to the network, we sum the values of Δw_{kj} and $\Delta \theta_k$ obtained after each pattern is presented to the network and only update the weights after all patterns have been shown to the network (after an epoch).

3.1 MEE for Learning in Batch Mode

Now we propose the use of the minimization of the entropy of the errors (MEE) instead of the minimization of the mean squared error (MSE) as the optimization principle behind the learning for this network.

This type of training needs a batch mode algorithm because we have to estimate the distribution of the errors for updating the weights, so we need several of these errors to obtain a good estimate.

In [11] it is shown that the minimization of the error entropy (in particular, Renyi's entropy) results in the minimization of the divergence between the joint pdfs of input-target and input-output signals. This suggests that the distribution of the output of the system is converging to the distribution of the targets. Also, when the entropy is minimized, for the classification case and under certain mild conditions, implies that the error must be zero (see proof in [12]).

As we saw above, the error $e_j = t_j - y_j$ represents the difference between the target t_j of the j neuron and its output y_j. We will replace the MSE of the variable e_j for its MEE counterpart. First it is necessary to estimate the pdf of the error. For this we use the Parzen window approach $\hat{f}(e_j) = \frac{1}{Lh} \sum_{i=1}^{L} K\left(\frac{e_j - e_i}{h}\right)$ where h represents the bandwidth of the kernel K and L is the number of patterns in the training set. The kernel used is the Gaussian kernel given by $K(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$. Renyi's quadratic entropy is given by $H_{R2}(x) = -\log\left(\int_C (f(x))^2 dx\right)$ where C is the support of x and $f(\cdot)$ is its density function. Note that this last equation can be seen as the logarithm of the expected value of the pdf: $-\log E[f(x)]$. This justifies the use of the following estimator for H_{R2}: $\hat{H}_{R2}(x) = -\log\left(\frac{1}{L} \sum_{i=1}^{L} f(x_i)\right)$.

Once we plug the estimator of the pdf into this last expression, we get the final expression of the entropy of the error (the cost function)

$$\hat{H}_{R2}(e_j) = -\log\left(\frac{1}{L^2h}\sum_{i=1}^{L}\sum_{u=1}^{L}K\left(\frac{e_i-e_u}{h}\right)\right) \tag{12}$$

Note that instead of the time complexity for the MSE which is $O(L)$, the MEE approach has $O(L^2)$ complexity.

To find how to minimize this cost function, we follow a similar approach to the one done above for the MSE. Note first that to minimize equation (12) is the same as to maximize the argument of the logarithm, which we call J (ignoring the constant factors):

$$J = \sum_{i=1}^{L}\sum_{u=1}^{L}K\left(\frac{e_i-e_u}{h}\right) \tag{13}$$

First we find the derivative of J w.r.t. the real weights:

$$\frac{\partial J}{\partial w_{kj}^R} = \frac{1}{h}\sum_{i=1}^{L}\sum_{u=1}^{L}K'\left(\frac{e_i-e_u}{h}\right)\left(\frac{\partial e_i}{\partial w_{kj}^R} - \frac{\partial e_u}{\partial w_{kj}^R}\right) \tag{14}$$

The term $\frac{\partial e_i}{\partial w_{kj}^R}$ is given by $-\frac{\partial y_i}{\partial w_{kj}^R}$. This gives the following

$$\begin{aligned}\frac{\partial J}{\partial w_{kj}^R} = \frac{2}{h}\sum_{i=1}^{L}\sum_{u=1}^{L}K'\left(\frac{e_i-e_u}{h}\right)&((s(z_i^R)-s(z_i^I))(s'(z_i^R)x_j^R - s'(z_i^I)x_j^I)- \\ &(s(z_u^R)-s(z_u^I))(s'(z_u^R)x_j^R - s'(z_u^I)x_j^I))\end{aligned} \tag{15}$$

We will again use the gradient to guide the search for the weights, but in this case it is a gradient ascent since we wish to maximize J. So, the weight update at each iteration (t) will be guided by

$$\Delta w_{kj}^R(t) = \eta\frac{\partial J}{\partial w_{kj}^R} \tag{16}$$

A similar derivation can be done for the case of the imaginary weights. The expression equivalent to (15) is

$$\begin{aligned}\frac{\partial J}{\partial w_{kj}^I} = \frac{2}{h}\sum_{i=1}^{N}\sum_{u=1}^{N}K'\left(\frac{e_i-e_u}{h}\right)&((s(z_i^I)-s(z_i^R))(s'(z_i^R)x_j^I + s'(z_i^I)x_j^R)- \\ &(s(z_u^I)-s(z_u^R))(s'(z_u^R)x_j^I + s'(z_u^I)x_j^R))\end{aligned} \tag{17}$$

The update equations for the thresholds can be obtained by finding $\frac{\partial J}{\partial \theta_k^R}$ and $\frac{\partial J}{\partial \theta_k^I}$. These equations are

$$\begin{aligned}\frac{\partial J}{\partial \theta_k^R} = \frac{2}{h}\sum_{i=1}^{N}\sum_{u=1}^{N}K'\left(\frac{e_i-e_u}{h}\right)&((s(z_i^R)-s(z_i^I))s'(z_i^R)- \\ &(s(z_u^R)-s(z_u^I))s'(z_u^R))\end{aligned} \tag{18}$$

and

$$\frac{\partial J}{\partial \theta_k^I} = \frac{2}{h}\sum_{i=1}^{N}\sum_{u=1}^{N} K'\left(\frac{e_i - e_u}{h}\right)((s(z_i^I) - s(z_i^R))s'(z_i^I) - (s(z_u^I) - s(z_u^R))s'(z_u^I)) \tag{19}$$

4 Experiments

4.1 Datasets

We tried to find datasets were measurements were made with real and imaginary parts (complex numbers) because we suspected that these would be the most adequate settings for the type of network we are studying. Unfortunately it is very hard to find this type of data. We used an artificial dataset to simulate complex data and a real one, with actual complex measurements.

The artificial generated problem (Checkerboard) is a 2 by 2 grid of points with alternate classes (similar to the XOR problem). It contains 400 points, 100 per grid position and 200 per class. In this case we consider that the value of the X coordinate of a point is the real part of a complex measurement and the Y coordinate is the imaginary part.

The second is a breast cancer dataset. It consists of electrical impedance measurements that were performed on 120 samples of freshly excised breast tissue. The problem has 6 classes, 120 points and 24 features (real and imaginary parts of 12 measurements of impedance at different frequencies) [13].

The data was centered and reduced for all algorithms with the exception of the SVM where a normalization in the interval [-1,1] was done for each feature. We used the LIBSVM [14] implementation.

4.2 Results

The results are in table 1. This table contain the average error and standard deviation of 30 repetitions of a two-fold cross-validation. We show also the results using SVM with RBF kernel (best value obtained for g varying from 2.2 to 0.8 in steps of 0.2, for C=10 and C=100), k-NN (best value from k=1, 3, 5 and 7) and the C4.5 decision tree. For the MEE version there were 3 results for each value of the learning rate, one for each of values of the kernel bandwidth used (1.0, 1.2 and 1.4). We only show the best to save space. The presented results for the CVNNs were the best values obtained when the training run for 4000 epochs, which were evaluated at 20 epochs intervals on the test set.

The results for the Checkerboard problem are very impressive: the CVNN is able to attain almost perfect classification and the second best method, the SVM with RBF, is still a bit behind. In this dataset, the batch MEE version is also the best for the tested values of the parameters, when compared with the other two versions. For the Checkerboard problem we also show the more informative balanced error rate since this is a two class problem (we cannot show this value for the second dataset since it has 6 classes).

Table 1. Average error and balanced error (BER), in percentage, with standard deviation for 30 repetitions of a two fold-cross validation for both datasets

Dataset ->	Checkerboard			Breast cancer	
Method	Parameters	Error (std)	BER (std)	Parameters	Error (std)
SVM RBF	g=1.8, C=10	2.92 (0.60)	5.37 (1.14)	g=1.0, C=10	31.83 (3.23)
k-NN	$k = 1$	4.48 (0.98)	7.18 (1.38)	$k = 5$	34.42 (2.99)
C4.5	-	25.22 (0.29)	49.91 (0.55)	-	35.28 (5.28)
Stochastic	$\eta = 0.09$	0.51 (0.38)	0.38 (0.34)	$\eta = 0.09$	32.11 (5.68)
Batch MSE	$\eta = 0.09$	0.60 (0.43)	0.47 (0.39)	$\eta = 0.09$	33.25 (6.05)
Batch MEE	$\eta = 0.09, h = 1.0$	0.30 (0.22)	0.22 (0.21)	$\eta = 0.09, h = 1.0$	33.14 (5.55)
Stochastic	$\eta = 0.07$	0.43 (0.26)	0.32 (0.29)	$\eta = 0.07$	32.69 (5.30)
Batch MSE	$\eta = 0.07$	0.48 (0.39)	0.37 (0.33)	$\eta = 0.07$	33.25 (6.05)
Batch MEE	$\eta = 0.07, h = 1.4$	0.32 (0.31)	0.24 (0.28)	$\eta = 0.07, h = 1.4$	33.47 (6.19)
Stochastic	$\eta = 0.05$	0.57 (0.50)	0.46 (0.54)	$\eta = 0.05$	33.64 (5.07)
Batch MSE	$\eta = 0.05$	0.45 (0.30)	0.33 (0.26)	$\eta = 0.05$	33.03 (5.26)
Batch MEE	$\eta = 0.05, h = 1.4$	0.33 (0.24)	0.22 (0.16)	$\eta = 0.05, h = 1.0$	33.00 (4.94)
Stochastic	$\eta = 0.03$	0.72 (0.55)	0.59 (0.51)	$\eta = 0.03$	33.17 (6.18)
Batch MSE	$\eta = 0.03$	0.58 (0.36)	0.44 (0.34)	$\eta = 0.03$	32.94 (5.74)
Batch MEE	$\eta = 0.03, h = 1.4$	0.37 (0.22)	0.27 (0.21)	$\eta = 0.03, h = 1.0$	33.50 (4.43)
Stochastic	$\eta = 0.01$	0.68 (0.32)	0.57 (0.34)	$\eta = 0.01$	33.28 (5.66)
Batch MSE	$\eta = 0.01$	0.68 (0.54)	0.59 (0.57)	$\eta = 0.01$	33.61 (5.29)
Batch MEE	$\eta = 0.01, h = 1.0$	0.23 (0.31)	0.18 (0.27)	$\eta = 0.01, h = 1.0$	34.58 (3.84)

For the Brest Cancer problem, the SVM with RBF was the best classifier. The CVNN came in second place. Within the 3 variants of the CVNN, the best results were obtained by the stochastic version. The MEE based version showed in general (4 out of 5) smaller standard deviations in the results. The exception was for $\eta = 0.07$.

5 Conclusions

In this paper we showed how to extend the previous existing single layer complex valued neural network to batch MSE training and batch MEE training. We present some experiments showing the validity of the proposals. It is interesting to see that in one of the experiments (Checkerboard), the CVNN improves substantially the results of other approaches. It would be important to try to understand what are the features of this dataset that make CVNNs so adequate to it, but this is beyond the scope of the present work. As future work, we would like to try to accelerate the MEE based algorithm, since it is quadratic in the number of data points. A possibility is the application of a mixed batch-sequential approach as in [15].

Acknowledgments. We acknowledge Prof. Marques de Sá for providing the datasets used in the experiments.

References

1. Hirose, A.: Complex-valued neural networks: The merits and their origins. In: Proceedings of the 2009 International Joint Conference on Neural Networks, pp. 1209–1216 (2009)
2. Mandic, D.P., Goh, V.S.L.: Complex valued nonlinear adaptive filters. John Wiley & Sons, Chichester (2009)
3. Amin, M., Murase, K.: Single-layered complex-valued neural network for real-valued classification problems. Neurocomputing 72, 945–955 (2009)
4. Savitha, R., Suresh, S., Sundararajan, N., Saratchandran, P.: A new learning algorithm with logarithmic performance index for complex-valued neural networks. Neurocomputing 72, 3771–3781 (2009)
5. Hirose, A.: Complex-Valued Neural Networks. Springer, Heidelberg (2006)
6. Haykin, S.: Neural Networks: A Comprehensive Foundation. MacMillan College Publishing Company, Inc., Basingstoke (1994)
7. Silva, L., Felgueiras, C., Alexandre, L., Marques de Sá, J.: Error entropy in classification problems: A univariate data analysis. Neural Computation **18**(9) (September 2006) 2036–2061
8. Silva, L.M., Marques de Sá, J., Alexandre, L.A.: The MEE principle in data classification: A perceptron-based analysis. Neural Computation 22(10), 2698–2728 (2010)
9. Alexandre, L., Marques de Sá, J.: Error Entropy Minimization for LSTM Training. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006. LNCS, vol. 4131, pp. 244–253. Springer, Heidelberg (2006)
10. Alexandre, L.: Maximizing the zero-error density for RTRL. In: 8th IEEE International Symposium on Signal Processing and Information Technology - ISSPIT 2008. IEEE Press, Sarajevo (December 2008)
11. Erdogmus, D., Principe, J.: An error-entropy minimization algorithm for supervised training of nonlinear adaptive systems. IEEE Trans. Signal Processing 50(7), 1780–1786 (2002)
12. Santos, J., Alexandre, L., Marques de Sá, J.: The error entropy minimization algorithm for neural network classification. In: Lofti, A. (ed.) Proceedings of the 5th International Conference on Recent Advances in Soft Computing, Nottingham, United Kingdom, pp. 92–97 (December 2004)
13. Silva, J., Sá, J., Jossinet, J.: Classification of breast tissue by electrical impedance spectroscopy. Medical & Biological Engineering & Computing 38(1), 26–30 (2000)
14. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001), Software, `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
15. Santos, J., Marques de Sá, J., Alexandre, L.: Neural networks trained with the EEM algorithm: Tuning the smoothing parameter. In: 6th WSEAS Int. Conference on Neural Networks, Lisbon, Portugal, vol. 4, pp. 295–300 (June 2005)