Journal of Computer and Communications, 2014, 2, 172-181
Published Online March 2014 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2014.24023

Scientific Research

Multi-Valued Neuron with Sigmoid Activation Function for Pattern Classification

Shen-Fu Wu, Yu-Shu Chiou, Shie-Jue Lee

Department of Electrical Engineering, National Sun Yat-sen University
Email: sfwu@water.ee.nsysu.edu.tw, yschiou@water.ee.nsysu.edu.tw, leesj@mail.ee.nsysu.edu.tw

Received December 2013

Abstract

Multi-Valued Neuron (MVN) was proposed for pattern classification. It operates with complex-valued inputs, outputs, and weights, and its learning algorithm is based on error-correcting rule. The activation function of MVN is not differentiable. Therefore, we can not apply backpropagation when constructing multilayer structures. In this paper, we propose a new neuron model, MVN-sig, to simulate the mechanism of MVN with differentiable activation function. We expect MVN-sig to achieve higher performance than MVN. We run several classification benchmark datasets to compare the performance of MVN-sig with that of MVN. The experimental results show a good potential to develop a multilayer networks based on MVN-sig.

Keywords

Pattern Classification; Multi-Valued Neuron (MVN); Differentiable Activation Function; Backpropagation

1. Introduction

The discrete multi-valued neuron (MVN) was proposed by N. Aizenberg and I. Aizenberg in [1] for pattern classification. The neuron operates with complex-valued inputs, outputs, and weights. Its inputs and outputs are mapped onto the complex plane. They are located on the unit circle, and are exactly the k^{th} roots of unity. The activation function of MVN k-valued logic maps a set of the k^{th} roots of unity on itself. Two discrete-valued MVN learning algorithms are presented in [2]. They are based on error-correcting learning rule and are derivative-free. This makes MVN have higher functionality than sigmoidal or radial basis function neurons.

The multilayer feedforward neural network based on MVN (MLMVN) was introduced in [3,4]. This model can achieve good performance using simpler structures. MLMVN learning rule is heuristic error backpropagation due to the fact that the activation function of MVN is not differentiable. The error with certain neuron is retrieved from next layer and evenly shared among the neurons connected from the former layer and itself. We can not apply function optimization methods to this model because of the activation function. This property led us to develop a multi-valued neuron with a differentiable activation function.

How to cite this paper: Wu, S.-F., Chiou, Y.-S. and Lee, S.-J. (2014) Multi-Valued Neuron with Sigmoid Activation Function for Pattern Classification. *Journal of Computer and Communications*, **2**, 172-181. http://dx.doi.org/10.4236/jcc.2014.24023

In this paper, we propose a new neuron model, MVN-sig, to simulate the mechanism of MVN with differentiable activation function. We consider the activation function of MVN as a function of the argument of a weighted sum. We stack multiple sigmoid functions to approximate this multiple step function. Hence, we can obtain a differentiable input/output mapping and we apply a naive gradient descent method as its learning rule. We expect MVN-sig to achieve better performance than MVN.

The rest of the paper is organized as follows. Section 2 briefly describes MVN and its activation function. Section 3 presents MVN-sig and the sigmoid activation function. The learning algorithm of MVN-sig is described in detail. Section 4 presents the results of three experiments. Finally, conclusions and discussions are given in Section 5.

2. Multi-Valued Neuron

2.1. Discrete MVN

A discrete-valued MVN is a function mapping from a n-feature input onto a single output. This mapping is described by a multiple-valued (k-valued) function of n-feature instances, $f(x_1,\ldots,x_n)$, which uses $n+1$ complex-valued weights:

$$f(x_1,\ldots,x_n) = P(\omega_0 + \omega_1 x_1 + \ldots + \omega_n x_n) \tag{1}$$

where x_1, x_n are the features of an instance, on which the performed function depends, and ω_0, ω_1, ω_n are the weights. The values of the function and of the features are complex. They are the k^{th} roots of unity: $\varepsilon^j = exp(i2\pi j/k)$, $j \in 0,1,\ldots,k-1$, and i is an imaginary unity. P is the activation function of the neuron:

$$P(z) = \exp(\frac{i2\pi j}{k}),\ if\ \frac{2\pi j}{k} \le arg(z) < \frac{2\pi(j+1)}{k} \tag{2}$$

where $j = 0,1,\ldots,k-1$ are values of the k-valued logic, $z = \omega_0 + \omega_1 x_1 + \ldots + \omega_n x_n$ is the weighted sum, and $arg(z)$ is the argument of the complex number z. Equation (2) is illustrated in **Figure 1**.

Equation (2) divides the complex plane into k equal sectors and maps the whole complex plane onto a subset of points belonging to the unit circle. This subset corresponds exactly to a set of the k^{th} roots of unity.

The MVN learning is reduced to the movement along the unit circle and is derivative-free. The movement is determined by the error which is the difference between the desired and actual output. The error-correcting learning rule and the corresponding learning algorithm for the discrete-valued MVN were described in [5] and modified by I. Aizenberg and C. Moraga [3]:

$$\omega_i^{r+1} = \omega_i^r + \frac{C_r}{(n+1)\,|\,z_r\,|}(\varepsilon^q - \varepsilon^s)\bar{x}_i, \tag{3}$$

for $i = 0,1,\ldots,n$, where $\bar{x}_i$ is the input of i^{th} feature with the components complex-conjugated, n is the number of the input features, ε^q is the desired output of the neuron, $\varepsilon^s = P(z)$ is the actual output of the

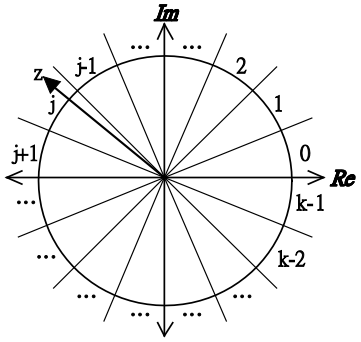

Figure 1. Geometrical interpretation of the discrete-valued MVN activation function.

neuron (see **Figure 2**), r is the number of the learning epoch, ω_i^r is the current weighting of the i^{th} feature, ω_i^{r+1} is the following weighting of the i^{th} feature after correction, C_r is the constant part of the learning rate (it may always equal to 1), and $|z_r|$ is the absolute value of the weighted sum obtained on the r^{th} epoch. The factor $\frac{1}{|z_r|}$ is useful when learning non-linear functions with a number of high irregular jumps. Equation (3) ensures that the corrected weighted sum moves from sector s to sector q (see **Figure 2(a)**). The direction of this movement is determined by the error $\delta = \varepsilon^q - \varepsilon^s$. The convergence of the learning algorithm was proven in [6].

2.2. Continuous MVN

The activation function Equation (2) is piece-wise discontinuous. This function can be modified and generalized for the continuous case in the following way. When $k \to \infty$ in Equation (2), the angle value of the sector (see in **Figure 1**) will approach to zero. The activation function is transformed as follows:

$$P(z) = \exp(i\arg(z)) = \frac{z}{|z|} \tag{4}$$

where z is the weighted sum, $\arg(z)$ is the argument of complex number z, and $|z|$ is the modulus of the complex number z. The activation function Equation (4) maps the weighted sum into the whole unit circle (see **Figure 2(b)**). Equation (2) maps only to the discrete subsets of the points belonging to the unit circle. Equation (2) and Equation (4) are both not differentiable, but their differentiability is not required for MVN learning. The Learning rule of the continuous-valued MVN is shown as follows:

$$\omega_i^{r+1} = \omega_i^r + \frac{C_r}{(n+1)|z_r|}(\varepsilon^q - \varepsilon^s)\overline{x}_i = \omega_i^r + \frac{C_r}{(n+1)|z_r|}(\varepsilon^q - \frac{z}{|z|})\overline{x}_i, \tag{5}$$

for $i = 0,1,\ldots,n$.

3. MVN with Sigmoid Activation Function

The learning algorithm of MVN is reduced to the movement along the unit circle on the complex plane. They are based on error-correcting learning rule and are derivative-free. Therefore, we can not use chain rule for error backpropagation to construct multilayer networks using MVN. In [3], I. Aizenberg and C. Moraga introduced a heuristic approach to construct the multilayer feedforward neural networks based on MVN. This inspired us to develop a differentiable activation function to which we can apply backpropagation learning rule. We expect the new learning rule to have more execution time but it can achieve better performance than MVN on a single neuron.

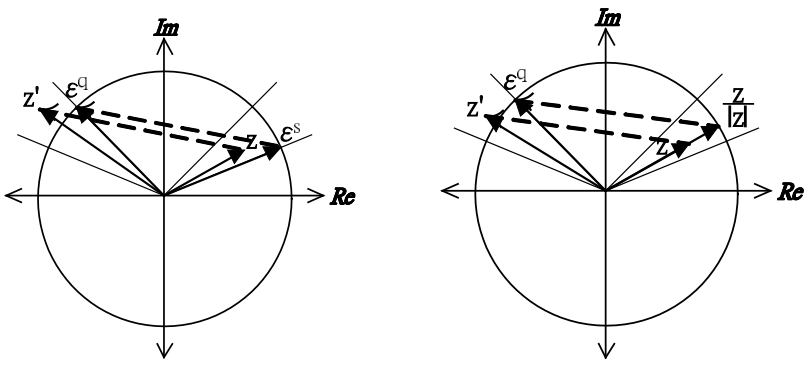

Figure 2. Geometrical interpretation of the MVN learning rule: (a) Discrete-valued MVN; and (b) Continuous-valued MVN.

3.1. Multi-Valued Sigmoid Activation Function

The basic idea of the multi-valued sigmoid activation function is to approximate the functionality of MVN using multiple sigmoid functions. In [7], E. Wilson presented a similar model for robot thruster control and farther applications. The original activation function of MVN is written in Equation (2). This activation function operates with the argument of weighted sum. Therefore, we can transfer this activation function into a function of argument which is illustrated in **Figure 3**. The corresponding MVN presentation is in **Figure 4**. This form of original activation function of MVN is a combination of multiple step functions, which can be approximated by stacking multiple sigmoid functions:

$$f_i(\theta_1) = \frac{2\pi / k}{1+\exp^{-c(\theta_1-\tau_i)}}$$
$$F(\theta_1) = \sum_{i=1}^{k-1} f_i(\theta_1) = \sum_{i=1}^{k-1} \frac{2\pi / k}{1+\exp^{-c(\theta_1-\tau_i)}} \quad (6)$$

where $F(\theta_1)$ is the multi-valued sigmoid activation function, θ_1 is the argument of weighted sum, τ_i is the argument of i^{th} root of unity and $f_i(\theta_1)$ is the sigmoid function setting on τ_i. The activation function $F(\theta_1): \theta_1 \to \theta_2$ in Equation (6) is a mapping from the argument of weighted sum onto the argument of certain root of unity. Hence, we use Euler's formula in Equation (7) to transfer the outcome to a complex value.

$$e^{i\theta_2} = cos(\theta_2) + isin(\theta_2) \quad (7)$$

where θ_2 is the outcome of activation function $F(\theta_1)$. We can approach the functionality of MVN activation function in Equation (2) by using Equations (6) and (7). Therefore, we can build a new model architecture which

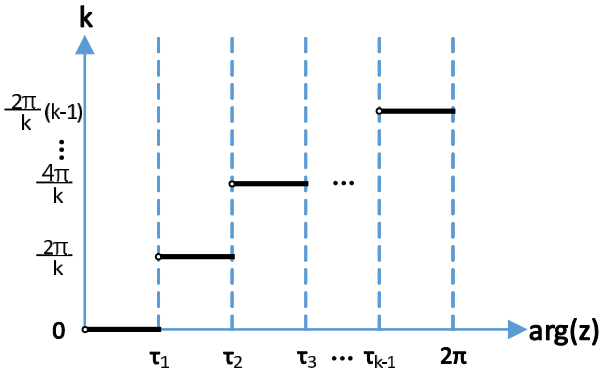

Figure 3. MVN activation function $\{0, \tau_1, \cdots, \tau_{k-1}\}$ represents the set of roots of unity by their arguments.

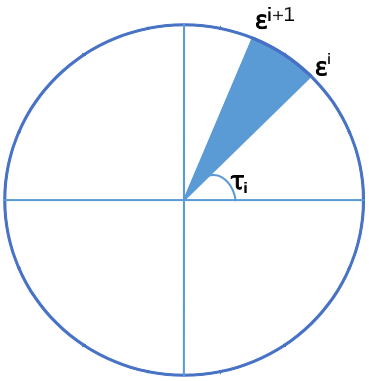

Figure 4. Geometrical interpretation of MVN activation function by τ_i.

can not only solve multi-valued logic problems but also be differentiable. We call this model "MVN-sig" and its model architecture is illustrated in **Figure 5**.

Since the activation function is no longer discrete-valued, we need to determine how to obtain the corresponding label for classification problems. The most intuitive method is to equally divide the mapped range of the activation function. This label judgement method is illustrated in **Figure 6**.

3.2. Learning Single Neuron Using Gradient Descent Method

Because the MVN-sig a complex-valued neuron, the output of MVN-sig can be derived from input:

$$x_j = x_{j,re} + x_{j,im}, \quad w_j = w_{j,re} + w_{j,im} \tag{8}$$

where $x_{j,re}$ and $x_{j,im}$ indicate the real and imaginary parts of j^{th} input variable, respectively. Likewise, $w_{j,re}$ and $w_{j,im}$ indicate the real and imaginary parts of j^{th} weight. Thus, the weighted sum of a given n-variable instance can be written as:

$$z = \sum_{j=0}^{n} w_j x_j = \sum_{j=0}^{n} (w_{j,re} x_{j,re} - w_{j,im} x_{j,im}) + i \sum_{j=0}^{n} (w_{j,im} x_{j,re} + w_{j,re} x_{j,im}) \tag{9}$$

Let the output of MVN-sig be $\hat{y}_t = a_t + ib_t = cos(\theta_2) + isin(\theta_2) = e^{i\theta_2}$. For an instance t, a_t and b_t are the real and imaginary parts of MVN-sig output, respectively.

$$\begin{aligned} \hat{y}_t &= a_t + ib_t = cos(\theta_2) + isin(\theta_2) = cos(F(\theta_1)) + isin(F(\theta_1)) \\ &= cos(F(arg(z))) + isin(F(arg(z))) \end{aligned} \tag{10}$$

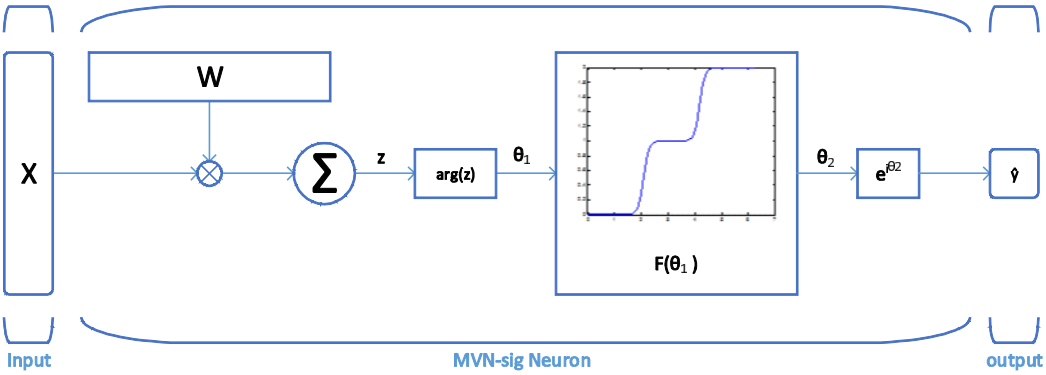

Figure 5. MVN-sig model architecture.

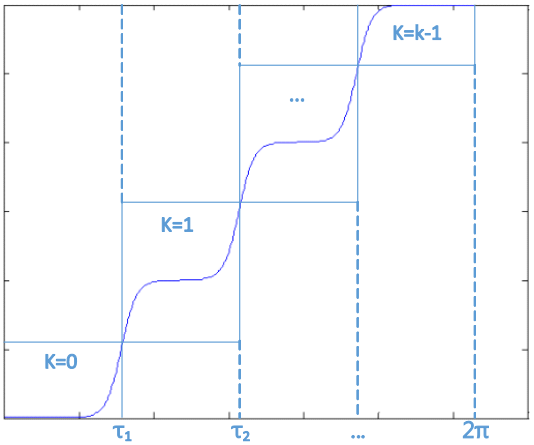

Figure 6. Label judgement.

The error function for an instance t is defined as:

$$E_t = \frac{1}{2}\delta_t \bar{\delta}_t, \ \delta_t = y_t - \hat{y}_t \tag{11}$$

where y_t is the desired output and $\hat{y}_t$ is the actual output of MVN-sig neuron. δ_t is the error between the desired value and the output. $\bar{\delta}_t$ signifies the complex conjugate of δ_t. Let the desired output be $y_t = \alpha_t + i\beta_t$. $\delta_t = y_t - \hat{y}_t = (\alpha_t - a_t) + i(\beta_t - b_t)$. Hence, the error function E_t is a real scalar function:

$$E_t = \frac{1}{2}\delta_t \bar{\delta}_t = \frac{1}{2}\{(\alpha_t - a_t)^2 + (\beta_t - b_t)^2\} \tag{12}$$

In order to use the chain rule to find the gradient of error function E_t, we have to calculate both real and imaginary parts independently. In Equation (12), we observe that E_t is a function of both $(\alpha_t - a_t)$ and $(\beta_t - b_t)$, and a_t and b_t are both functions of $w_{j,re}$ and $w_{j,im}$. We defined the gradient of the error function E_t with respect to the complex-valued weight w_j as follows:

$$\Delta w_j = -\eta\left(\frac{\partial E_t}{\partial w_j}\right) \overset{def}{=} -\eta\left(\frac{\partial E_t}{\partial w_{j,re}} + i\frac{\partial E_t}{\partial w_{j,im}}\right) \tag{13}$$

The gradient of the error function with respect to the real and imaginary parts of w_j can be written as:

$$\begin{aligned}\frac{\partial E_t}{\partial w_{j,re}} &= \frac{\partial E_t}{\partial a_t}\left(\frac{\partial a_t}{\partial F(arg(z))}\frac{\partial F(arg(z))}{\partial arg(z)}\frac{\partial arg(z)}{\partial w_{j,re}}\right) \\ &+ \frac{\partial E_t}{\partial b_t}\left(\frac{\partial b_t}{\partial F(arg(z))}\frac{\partial F(arg(z))}{\partial arg(z)}\frac{\partial arg(z)}{\partial w_{j,re}}\right)\end{aligned} \tag{14}$$

$$\begin{aligned}\frac{\partial E_t}{\partial w_{j,im}} &= \frac{\partial E_t}{\partial a_t}\left(\frac{\partial a_t}{\partial F(arg(z))}\frac{\partial F(arg(z))}{\partial arg(z)}\frac{\partial arg(z)}{\partial w_{j,im}}\right) \\ &+ \frac{\partial E_t}{\partial b_t}\left(\frac{\partial b_t}{\partial F(arg(z))}\frac{\partial F(arg(z))}{\partial arg(z)}\frac{\partial arg(z)}{\partial w_{j,im}}\right)\end{aligned} \tag{15}$$

We can combine Equation (14) and Equation (15) into Equation (13):

$$\begin{aligned}\Delta w_j = -\eta\{&\frac{\partial E_t}{\partial a_t}\frac{\partial a_t}{\partial F(arg(z))} + \frac{\partial E_t}{\partial b_t}\frac{\partial b_t}{\partial F(arg(z))}\} \\ &\{\frac{\partial F(arg(z))}{\partial arg(z)}\}\{\frac{\partial arg(z)}{\partial w_{j,re}} + i\frac{\partial arg(z)}{\partial w_{j,im}}\}\end{aligned} \tag{16}$$

Firstly, we derive each term within the first braces in Equation (16) from Equation (12) and Equation (10):

$$\frac{\partial E_t}{\partial a_t} = -(\alpha_t - a_t) = -Re(\delta_t) \tag{17}$$

$$\frac{\partial E_t}{\partial b_t} = -(\beta_t - b_t) = -Im(\delta_t) \tag{18}$$

$$\frac{\partial a_t}{\partial F(arg(z))} = \frac{\partial cos(F(arg(z)))}{\partial F(arg(z))} = -sin(F(arg(z))) \tag{19}$$

$$\frac{\partial b_t}{\partial F(arg(z))} = \frac{\partial sin(F(arg(z)))}{\partial F(arg(z))} = cos(F(arg(z))) \tag{20}$$

where $Re(\delta_t)$ and $Im(\delta_t)$ represent the real and imaginary parts of δ_t. Secondly, the second braces in

Equation (16) contains the derivative of the multi-valued sigmoid activation function. We can derive this activation function in Equation (6):

$$\frac{\partial F(arg(z))}{\partial arg(z)} = \frac{\partial}{\partial arg(z)} \sum_{i=1}^{k-1} \frac{2\pi / k}{1+\exp^{-c(arg(z)-\tau_i)}} = \sum_{i=1}^{k-1} \frac{2\pi}{k} \frac{c\exp^{-c(arg(z)-\tau_i)}}{1+\exp^{-c(arg(z)-\tau_i)}} \tag{21}$$

Finally, the third braces in Equation (16) contains the derivatives of the argument of weighted sum z. The derivation is shown as follows:

$$\frac{\partial arg(z)}{\partial w_j} = \frac{\partial}{\partial w_j}(tan^{-1}\frac{Im(z)}{Re(z)}) = \frac{Re(z)Im'(z) - Re'(z)Im(z)}{Re^2(z) + Im^2(z)} \tag{22}$$

where $Re(z)$ and $Im(z)$ represent the real and imaginary parts of weighted sum z and $Re'(z)$ and $Im'(z)$ represent their derivatives, respectively. From Equation (9), we can simplify this equation:

$$\frac{\partial Re(z)}{\partial w_{j,re}} = Re(x_j), \frac{\partial Im(z)}{\partial w_{j,re}} = Im(x_j), \tag{23}$$

$$\frac{\partial Re(z)}{\partial w_{j,im}} = -Im(x_j), \frac{\partial Im(z)}{\partial w_{j,im}} = Re(x_j) \tag{24}$$

where $Re(x_j)$ and $Im(x_j)$ represent the real and imaginary parts of j^{th} input variable x. Thus, the derivatives of the argument of weighted sum with respect to the real ($w_{j,re}$) and imaginary $w_{j,im}$ parts is shown as follows:

$$\frac{\partial arg(z)}{\partial w_{j,re}} = \frac{1}{|z|^2}(Re(z)Im(x_j) - Im(z)Re(x_j)) \tag{25}$$

$$\frac{\partial arg(z)}{\partial w_{j,im}} = \frac{1}{|z|^2}(Re(z)Re(x_j) + Im(z)Im(x_j)) \tag{26}$$

$$\frac{\partial arg(z)}{\partial w_{j,re}} + i\frac{\partial arg(z)}{\partial w_{j,im}} = i\frac{z}{|z|^2}\bar{x}_j \tag{27}$$

where $\bar{x}_j$ represents the complex conjugate of j^{th} input variable x. From Equations (17)-(21) and (27), we can generate the learning rule of j^{th} input weighting Δw_j:

$$\Delta w_j = -\eta\{Re(\delta_t)sin(\theta_2) - Im(\delta_t)cos(\theta_2)\}\{\sum_{i=1}^{k-1} \frac{2\pi}{k} \frac{c\exp^{-c(\theta_1-\tau_i)}}{1+\exp^{-c(\theta_1-\tau_i)}}\}\{i\frac{z}{|z|^2}\bar{x}_j\} \tag{28}$$

where θ_1 is the argument of the weighted sum and θ_2 is the output of multi-valued sigmoid activation function ($F(\theta_1)$).

3.3. Stopping Criteria

We combine two stopping criteria to make sure the learning process stops when the output is accurate and stable. Firstly, we continue to iterate until the difference between the network response and the target function reaches some acceptable level. In Equation (29), we keep tracking the mean squared error until it reaches a given value λ. Secondly, we check the relative change in the training error is small using Equation (30).

$$\frac{1}{N}\sum_{t=1}^{N}\delta_t\bar{\delta}_t \le \lambda \tag{29}$$

$$\frac{|E(w^t) - E(w^{t-1})|}{\frac{1}{2}\{E(w^t) - E(w^{t-1})\}} \le \Lambda \tag{30}$$

where N is the total number of learning instances, δ_t is the error between desired value and estimation, and $E(w^t)$ and $E(w^{t-1})$ represent the mean squared errors at t^{th} and $(t-1)^{th}$ learning epochs.

Figure 7 shows two examples of error convergence. Vertical and horizontal axis represent mean squared error and time epoch, respectively. In **Figure 7(a)**, if we only use Equation (30) as the stopping criterion, the learning process will stop early and has worse training and testing accuracy. On the other hand, if we do not use Equation (30), it will take too much time to converge. In **Figure 7(b)**, we can see the learning process is relatively stable after 300 epochs. We monitored the testing accuracy at 300 epoch and at the end of learning process. They did not change much during this interval and that is why we use Equation (29) and Equation (30) as our stopping criteria.

3.4. MVN-Sig Learning Algorithm

We develop the learning algorithm based on the multi-valued sigmoid function and the gradient descent method. The convergence of the learning algorithm can be proven based on the convergence of the gradient descent method in [8]. The implementation of the proposed learning algorithm in one iteration consists of the following steps:

procedure MVN-sig
This is one learning epoch with N learning instances
Let $i = 1$ represents the i^{th} instance
Let z be the current value of weighted sum.
$P(z) = \varepsilon^s$ /⋆ Equation (10) ⋆/
$\delta \leftarrow \varepsilon^q - \varepsilon^s$
while $i \le N$ **do**
Check equation with activation function.
for j = 0 to n **do** /⋆ n-variable instance ⋆/
$w_j^{r+1} \leftarrow w_j^r + \Delta w_j$ /⋆ Equation (28) ⋆/
end for
$\tilde{z} = w_0^{r+1} + w_1^{r+1}x_1 + \ldots + w_n^{r+1}x_n$
$\delta \leftarrow \varepsilon^q - P(\tilde{z})$
$i = i + 1$
end while
Iterates until the outcome meets stopping criteria. /⋆ Equation (29) and Equation (30) ⋆/
end procedure

4. Simulation Results

The proposed strategies and learning algorithms are implemented and checked over a given three benchmark

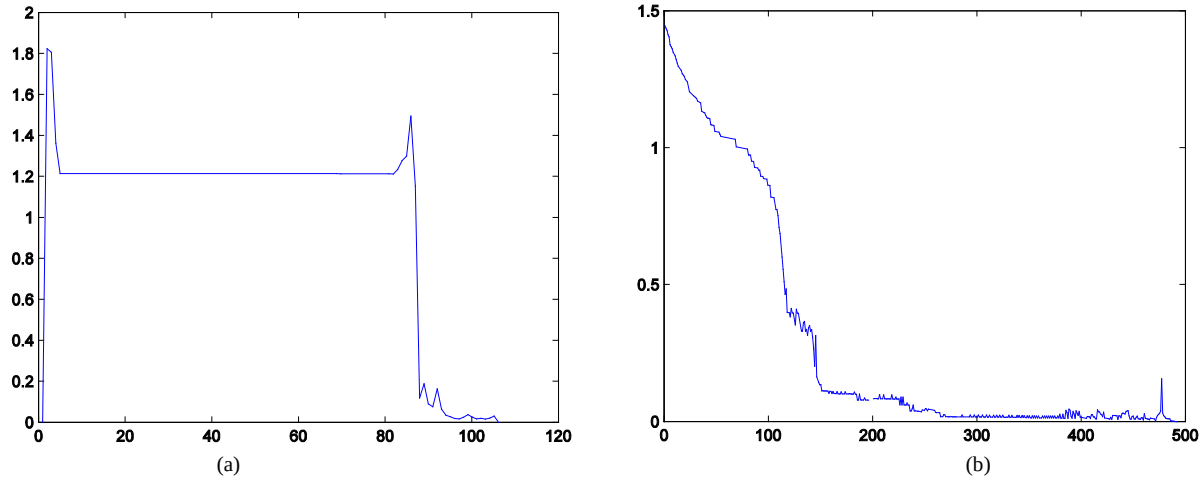

Figure 7. MSE-epoch plots.

datasets [9]. The software simulator is written in Matlab R2011b (64-bit) running on a computer with AMD Phenom II X4 965 3.4 GHz CPU and 8 GB RAM.

4.1. Wine Dataset

This dataset is downloaded from the website of UC Irvine Machine Learning Repository [9]. It contains 178 instances and, for each instance, there are 13 real-valued input variables. The instances belongs to any of the 3 classes that indicates 3 different types of wines.

To solve this classification problem, we use 5-fold cross-validation. The first fold has 136 instances in the training set and 42 instances in the testing set. The other 4 folds have 144 instances in the training set and 34 instances in the testing set, respectively. For MVN, we used the testing set to compute their average classification performance after learning with the training set completely with no training error, *i.e*., training accuracy being 100%. For MVN-sig, we computed their average classification performance after learning process met the stopping criteria. The testing accuracy obtained by MVN and MVN-sig, respectively, for this dataset, is shown in **Table 1**. From this table, we can see that MVN-sig achieves a better testing accuracy than MVN.

4.2. Iris Dataset

This well known dataset is also downloaded from the website of UC Irvine Machine Learning Repository [9]. It contains 150 instances and, for each instance, there are 4 real-valued input variables. The instances are belonging to any of the 3 classes (Setosa: 0, Versicolour: 1 and Virginica: 2).

To solve this classification problem, we use 5-fold cross-validation. The dataset is randomly divided into 120 instances of training set and 30 instances of testing set. For MVN, the learning process will not stop after 10,000 iterations if we want to learn whole instances with no error, *i.e*., training accuracy being 100%. The training accuracy of our MVN-sig algorithm is from 96% to 99%, and we choose 96% to be the stopping criterion for MVN learning. For MVN-sig,we computed their average classification performance after learning process met the stopping criteria. The testing accuracy obtained by MVN and MVN-sig, respectively, for this dataset, is shown in **Table 1**. From this table, we can see that MVN-sig achieves a better testing accuracy than MVN.

4.3. Breast Cancer Wisconsin (Diagnostic) Dataset

This is also downloaded from the website of UC Irvine Machine Learning Repository [9]. It contains 569 instances and, for each instance, there are 32 real-valued input variables. The instances are belonging to 2 classes (malignant: 0 and benign: 1).

To solve this classification problem, we use 5-fold cross-validation. The first fold has 452 instances in the training set and 117 instances in the testing set. The other 4 folds have 456 instances in the training set and 113 instances in the testing set, respectively. For MVN, we used continuous learning rule in Equation (5) since the discrete one is not suitable for binary classification. We used the testing set to compute their average classification performance after learning with the training set completely with no training error, *i.e*., training accuracy being 100%,. For MVN-sig,we computed their average classification performance after learning process met the stopping criteria. The testing accuracy obtained by MVN and MVN-sig, respectively, for this dataset, is shown in **Table 1**. From this table, we can see that MVN-sig achieves a better testing accuracy than MVN.

5. Conclusions and Discussions

The parameter c in Equation (6) affects the converging time and performance. We can develop a parameterfree model since the activation function of MVN-sig is differentiable. But if we make this model parameter-free, the

Table 1. Accuracy comparison of MVN and MVN-sig.

	MVN	**MVN-sig**
Wine	94.404	94.980
Iris	93.200	95.870
Diagnostic	86.936	87.480

execution time is increased tremendously. In this paper, we select suitable parameters for each experiment.

The simulation results obtained with benchmark datasets show MVN-sig meets our expectation of original thoughts. From the result of Iris dataset, we can see if we loosen the MVN stopping criterion, MVN-sig can still achieve better testing accuracy. But the critical downside is the much higher time complexity. In this paper, we just apply a simple gradient descent method to the neuron. More complicated techniques for reducing back-propagation time can be applied.

There are three main concerns about the future work of the MVN-sig neuron. Firstly, we have to develop a multilayer neural network using MVN-sig and investigate its performance. Secondly, to reduce its execution time we need to do more research on the learning algorithm. Finally, we have to find its pros and cons in the real-world applications.

References

[1] Aizenberg, N.N. and Aizenberg, I.N. (1992) CNN Based on Multivalued Neuron as a Model of Associative Memory For Grey Scale Images. In *Proceedings of Second IEEE International Workshop on Cellular Neural Networks and their Applications* (*CNNA*-92), 36-41. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=274330

[2] Aizenberg, I., Aizenberg, N.N. and Vandewalle, J.P. (2000) Multi-Valued and Universal Binary Neurons: Theory, Learning and Applications. Springer. http://dx.doi.org/10.1007/978-1-4757-3115-6

[3] Aizenberg, I. and Moraga, C. (2007) Multilayer Feed Forward Neural Network Based on Multi-Valued Neurons (mlmvn) and a Backpropagation Learning Algorithm. *Soft Computing—A Fusion of Foundations, Methodologies and Applications*, **11**, 169-183. http://www.springerlink.com/index/T645547TN41006G0.pdf

[4] Aizenberg, I., Paliy, D.V., Zurada, J.M. and Astola, J. T. (2008) Blur Identification by Multilayer Neural Network Based on Multivalued Neurons. **19**, 883-898. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4479859

[5] Aizenberg, I. (2010) Periodic Activation Function and a Modified Learning Algorithm for the Multivalued Neuron. **21**, 1939-1949. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5613940

[6] Aizenberg, I. (2011) Complex-Valued Neural Networks with Multi-Valued Neurons. Springer.

[7] Wilson, E. (1994) Backpropagation Learning for Systems with Discrete-Valued Functions. *Proceedings of the World Congress on Neural Networks*, San Diego, California, June.

[8] Hagan, M.T., Demuth, H.B., Beale, M.H., *et al.* (1996) Neural Network Design. Thomson Learning Stamford, CT.

[9] UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/index.html