Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Negative effects of sufficiently small initial weights on back-propagation neural networks*

Yan LIU[1,2], Jie YANG[1], Long LI[3], Wei WU[‡1]

([1]*School of Mathematical Sciences, Dalian University of Technology, Dalian 116024, China*)

([2]*School of Information Science and Engineering, Dalian Polytechnic University, Dalian 116034, China*)

([3]*Department of Mathematics and Computational Science, Hengyang Normal University, Hengyang 421002, China*)

E-mail: liuyan@dlpu.edu.cn; yangjiee@dlut.edu.cn; long_li1982@163.com; wuweiw@dlut.edu.cn

Received Jan. 11, 2012; Revision accepted June 21, 2012; Crosschecked July 6, 2012

Abstract: In the training of feedforward neural networks, it is usually suggested that the initial weights should be small in magnitude in order to prevent premature saturation. The aim of this paper is to point out the other side of the story: In some cases, the gradient of the error functions is zero not only for infinitely large weights but also for zero weights. Slow convergence in the beginning of the training procedure is often the result of sufficiently small initial weights. Therefore, we suggest that, in these cases, the initial values of the weights should be neither too large, nor too small. For instance, a typical range of choices of the initial weights might be something like $(-0.4,-0.1)\cup(0.1,0.4)$, rather than $(-0.1,0.1)$ as suggested by the usual strategy. Our theory that medium size weights should be used has also been extended to a few commonly used transfer functions and error functions. Numerical experiments are carried out to support our theoretical findings.

Key words: Neural networks, Back-propagation, Gradient learning method, Convergence
doi:10.1631/jzus.C1200008 **Document code:** A **CLC number:** TP18

1 Introduction

Feedforward neural networks with hidden layers are probably the most widely used neural networks, and they are usually trained by the back-propagation (BP) algorithm based on the gradient descent method (Castillo *et al.*, 2000; Wu *et al.*, 2005; Zhang *et al.*, 2006; Liu *et al.*, 2008). Over the years, neural networks have received considerable attention and achieved many promising results (Xiong *et al.*, 2007; Qi *et al.*, 2009; Bao *et al.*, 2010; Deng *et al.*, 2010), and thus has been successfully applied to resolve problems such as pattern recognition, forecasting, nonlinear control, and regression analysis (Biswajeet and Saied, 2010; Pradhan and Buchroithner, 2010; Pradhan *et al.*, 2010; Pradhan, 2011; Wang *et al.*, 2011). When a feedforward neural network is trained with the BP algorithm, the performance depends on several factors, such as choice of learning parameters, cost function, and network topology (Drago and Ridella , 1992). In the literature, a number of studies on the choice of weights emerged. We mention a few related references on the choice of the initial values of the weights. An algorithm is introduced for determining the optimal initial weights of feedforward neural networks based on a linear algebraic method (Yam *et al.*, 1997). An optimal initialization of bias and initial weight vectors based on multidimensional geometry is proposed (Yam *et al.*, 2001). It is suggested to choose the set of initial weights from a few sets of weights in a statistic point of view (Kathirvalavakumar and

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 11171367) and the Fundamental Research Funds for the Central Universities, China

Thangavel, 2003). An optimization methodology for neural network weights and architectures was introduced to generate an automatic process with high classification performance and low complexity (Ludermir *et al.*, 2006). The sensitivity of Adalines to weights perturbation is discussed by means of the hypercube model and analytical geometry method (Zeng *et al*, 2006). The initial values are analyzed using the split-complex BP algorithm (Yang *et al.*, 2008).

An important and commonly adopted strategy is that the initial weights are chosen to be small in magnitude in order to prevent premature saturation in the training procedure (Elman, 1993; Hagan *et al.*, 1996; Ham and Kostanic, 2001). We point out in this paper the other side of the story: In some cases, slow convergence in the beginning of the training procedure is the result of sufficiently small initial weights.

To elaborate, let us recall why generally the initial weights should be small. The transfer (or activation) functions for the hidden layer and the output layer of the BP neural networks are often chosen to be sigmoid functions, such as $g(x) = 1/(1+\mathrm{e}^{-x})$ with its saturation value being $0/1$. Then, large values of the weights $\rightarrow$ large value of x $\rightarrow$ small value of the derivative $g'(x) \rightarrow$ slow convergence. Hence, a commonly applied strategy is to choose the values of the initial weights randomly from a small symmetrical range, say, $(-0.1, 0.1)$.

However, in some special cases, e.g., some widely used benchmark classification problems, the gradients of the error functions are $\mathbf{0}$ not only for infinitely large weights but also for zero weights. (Henceforth we shall use $\mathbf{0}$ to denote a vector whose components are all zero. Its dimension may be different in different places and can be determined from the context.) Hence, in such cases, similarly we have: small values of the weights $\rightarrow$ small values of the gradients $\rightarrow$ slow convergence. The objective and motivation of this study is to demonstrate that based on this observation, when solving these classification problems in numerical experiments, the initial values of the weights should be neither too large, nor too small. For instance, a typical range for choosing the initial weights might be something like $(-0.4, -0.1) \cup (0.1, 0.4)$.

2 BP algorithm based on the gradient descent method

Consider a feedforward neural network with three layers for the pattern classification task. The numbers of neurons for the input, hidden, and output layers are l, n, and 1, respectively. The structure is shown in Fig. 1. Let the input training examples be $\boldsymbol{\xi}^j \in \mathbb{R}^l$ $(j = 1, 2, \ldots, J)$, and the corresponding desired outputs be $O^j \in \mathbb{R}$ $(j = 1, 2, \ldots, J)$. We denote the weight matrix connecting the input and the hidden layers by $\boldsymbol{V} = (v_{ij})_{n\times l}$, and we let $\boldsymbol{v}_i = (v_{i1}, v_{i2}, \ldots, v_{il})^{\mathrm{T}} \in \mathbb{R}^l$ $(i = 1, 2, \ldots, n)$ denote the weight vector connecting the input layer and the ith hidden neuron.

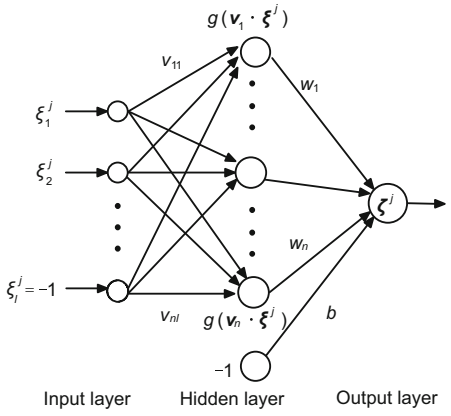

Fig. 1 Structure of a back-propagation network with three layers

We remark that there are actually bias terms involved in the neural system. However, following a common strategy, we set the last component ξ_l^j $(j = 1, 2, \ldots, J)$ of the input vectors to be -1, so the last component v_{il} of $\boldsymbol{v}_i$ stands for the bias term for the ith hidden neuron. This strategy allows us not to write explicitly the bias terms in the description of our problem.

The weight vector connecting the hidden and the output layers is denoted by $\tilde{\boldsymbol{w}} = (w_1, w_2, \ldots, w_n)^{\mathrm{T}} \in \mathbb{R}^n$. Let $g : \mathbb{R} \longrightarrow \mathbb{R}$ be the transfer function for both the hidden and the output layers, which is typically, but not necessarily, of sigmoid type. For convenience, we introduce the following vector function of $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^{\mathrm{T}} \in \mathbb{R}^n$:

$$\tilde{G}(\boldsymbol{x}) = (g(x_1), g(x_2), \ldots, g(x_n))^{\mathrm{T}}. \qquad (1)$$

For any given input $\boldsymbol{\xi} \in \mathbb{R}^l$, the output of the hidden

layer is

$$\tilde{G}(\boldsymbol{V}\boldsymbol{\xi}) = (g(\boldsymbol{v}_1 \cdot \boldsymbol{\xi}), g(\boldsymbol{v}_2 \cdot \boldsymbol{\xi}), \ldots, g(\boldsymbol{v}_n \cdot \boldsymbol{\xi})), \quad (2)$$

and the final output of the network is

$$\boldsymbol{\zeta} = g\left(\tilde{\boldsymbol{w}} \cdot \tilde{G}(\boldsymbol{V}\boldsymbol{\xi}) - b\right), \quad (3)$$

where b is the bias for the output neuron. Similarly as above, we write

$$\begin{cases} \boldsymbol{w} = (w_1, w_2, \ldots, w_n, b)^{\mathrm{T}}, \\ G(\boldsymbol{x}) = (g(x_1), g(x_2), \ldots, g(x_n), -1)^{\mathrm{T}}. \end{cases} \quad (4)$$

Then Eq. (3) can be simplified as

$$\boldsymbol{\zeta} = g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi})). \quad (5)$$

The popular square error function $E(\boldsymbol{w}, \boldsymbol{V})$ is given by

$$E(\boldsymbol{w}, \boldsymbol{V}) := \frac{1}{2}\sum_{j=1}^{J}(O^j - \boldsymbol{\zeta}^j)^2 = \frac{1}{2}\sum_{j=1}^{J}\left[O^j - g\left(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j)\right)\right]^2. \quad (6)$$

Other types of error functions can also be used (Section 5). The aim of network training is to find the optimal weights $(\boldsymbol{w}^*, \boldsymbol{V}^*)$ such that

$$E(\boldsymbol{w}^*, \boldsymbol{V}^*) = \min E(\boldsymbol{w}, \boldsymbol{V}). \quad (7)$$

First let us introduce the batch, or off-line, training method. We start from an arbitrary initial value $(\boldsymbol{w}^0, \boldsymbol{V}^0)$, and proceed to refine it iteratively by the following rules:

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k + \triangle\boldsymbol{w}^k,\ k = 0, 1, \ldots, \quad (8a)$$

$$\boldsymbol{v}_i^{k+1} = \boldsymbol{v}_i^k + \triangle\boldsymbol{v}_i^k,\ i = 1, 2, \ldots, n, k = 0, 1, \ldots, \quad (8b)$$

where (η is the learning rate)

$$\triangle\boldsymbol{w}^k = -\eta E_{\boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{V}) = -\eta\sum_{j=1}^{J}[(O^j - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))) \cdot g'(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))G(\boldsymbol{V}\boldsymbol{\xi}^j)], \quad (9a)$$

$$\triangle\boldsymbol{v}_i^k = -\eta E_{\boldsymbol{v}_i}(\boldsymbol{w}, \boldsymbol{V}) = -\eta\sum_{j=1}^{J}[(O^j - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))) \cdot g'(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))w_i g'(\boldsymbol{v}_i \cdot \boldsymbol{\xi}^j)\boldsymbol{\xi}^j]. \quad (9b)$$

Next, we introduce the online gradient method. Let $\{\boldsymbol{\xi}^{(k)}, O^{(k)}\}_{k=0}^{\infty} \in \mathbb{R}^l \times \mathbb{R}$ be the training sequence of input-target pairs, such that each $(\boldsymbol{\xi}^{(k)}, O^{(k)})$ is chosen from $\{(\boldsymbol{\xi}^j, O^j) | j = 1, 2, \ldots, J\}$ stochastically. Now, in place of Eqs. (9a) and (9b) we have

$$\triangle\boldsymbol{w}^k = -\eta\left(O^{(k)} - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^{(k)}))\right) \cdot g'\left(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^{(k)})\right)G(\boldsymbol{V}\boldsymbol{\xi}^{(k)}), \quad (10a)$$

$$\triangle\boldsymbol{v}_i^k = -\eta\left(O^{(k)} - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^{(k)}))\right) \cdot g'\left(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^{(k)})\right)w_i g'(\boldsymbol{v}_i \cdot \boldsymbol{\xi}^{(k)})\boldsymbol{\xi}^{(k)}. \quad (10b)$$

3 When and why sufficiently small initial weights are ineffective

In this section, we present three theorems to show that, under certain conditions, the gradient of the square error function for zero weights is, or nearly is, $\mathbf{0}$. In such a case, it is clear that small initial weights in the training procedure (8)−(10) will lead to slow convergence since the gradient of the square error function will be very small.

Theorem 1 Consider the three-layer neural network (3) with the batch BP training algorithm (8)−(9). Then, the gradient of the square error function (6) for zero weights is $\mathbf{0}$, provided that one of the following two conditions is satisfied:

(i) $g'(0) = 0$;

(ii) The network is used to solve a classification problem such that the numbers of the two classes of training samples are equal and that $g(0) = (O^+ + O^-)/2$, where O^+ and O^- are the two target outputs for the two classes, respectively.

Proof The gradients of the square error function $E(\boldsymbol{w}, \boldsymbol{V})$ with respect to the weights $\boldsymbol{w}$ and $\boldsymbol{V}$ are shown below:

$$E_{\boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{V}) = -\sum_{j=1}^{J}[(O^j - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))) \cdot g'\left(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j)\right)G(\boldsymbol{V}\boldsymbol{\xi}^j)], \quad (11)$$

where $G(\boldsymbol{V}\boldsymbol{\xi}^j) = (g(\boldsymbol{v}_1 \cdot \boldsymbol{\xi}^j), \ldots, g(\boldsymbol{v}_n \cdot \boldsymbol{\xi}^j), -1)^{\mathrm{T}}$,

$$E_{\boldsymbol{v}_i}(\boldsymbol{w}, \boldsymbol{V}) = -\sum_{j=1}^{J}[(O^j - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))) \cdot g'(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))w_i g'(\boldsymbol{v}_i \cdot \boldsymbol{\xi}^j)\boldsymbol{\xi}^j]. \quad (12)$$

At $\boldsymbol{u} \equiv (\boldsymbol{w}, \boldsymbol{V}) = \boldsymbol{0}$, we have

$$G(\boldsymbol{V}\boldsymbol{\xi}^j)|_{\boldsymbol{u}=\boldsymbol{0}} = (g(0), g(0), \ldots, g(0), -1)^{\mathrm{T}}, \quad (13)$$

$$\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j)|_{\boldsymbol{u}=\boldsymbol{0}} = 0, \quad (14)$$

$$g\left(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j)\right)|_{\boldsymbol{u}=\boldsymbol{0}} = g(0), \quad (15)$$

$$g'\left(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j)\right)|_{\boldsymbol{u}=\boldsymbol{0}} = g'(0). \quad (16)$$

Thus,

$$E_{\boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{V})|_{\boldsymbol{u}=\boldsymbol{0}} = -\sum_{j=1}^{J}[(O^j - g(0))g'(0) \cdot (g(0), \ldots, g(0), -1)^{\mathrm{T}}]. \quad (17)$$

Apparently, provided that Condition (i) or (ii) in Theorem 1 holds, there will be

$$E_{\boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{V})|_{\boldsymbol{u}=\boldsymbol{0}} = \boldsymbol{0}. \quad (18)$$

The following evaluation is simple:

$$E_{\boldsymbol{v}_i}(\boldsymbol{w}, \boldsymbol{V})|_{\boldsymbol{u}=\boldsymbol{0}} = \boldsymbol{0}, \quad i = 1, 2, \ldots, n. \quad (19)$$

This completes the proof.

Similarly the following two theorems can be proven.

Theorem 2 Consider the three-layer neural network (3) with online BP training algorithm (10). Then, the gradient of the square error function (6) for zero weights is $\boldsymbol{0}$, provided that $g'(0) = 0$ is satisfied.

Theorem 3 If the transfer function satisfies $g(0) = 0$, then for both the batch algorithm (9) and the online algorithm (10), the gradients of the square error function $E(\boldsymbol{w}, \boldsymbol{V})$ with respect to $\tilde{\boldsymbol{w}} = (w_1, w_2, \ldots, w_n)^{\mathrm{T}}$ and $\boldsymbol{v}_i$ $(i = 1, 2, \ldots, n)$ are all $\boldsymbol{0}$, but the partial derivative with respect to the bias b for the output neuron might not be zero.

Remark The condition $g'(0) = 0$ in Theorems 1 and 2 is satisfied by, for example, the popular radial basis functions which are often used for approximation problems. Condition (ii) in Theorem 1 is satisfied by many benchmark classification problems as shown in the next section. The condition $g(0) = 0$ in Theorem 3 is satisfied by many popular transfer functions.

4 Numerical experiments

In this section, we carry out seven numerical experiments as shown in the seven tables below to support our theoretical findings. The first four experiments satisfy Condition (ii) in Theorem 1, the fifth satisfies Condition (i) in Theorem 1, the sixth is for an online BP training problem satisfying the condition in Theorem 2, and the seventh satisfies the condition in Theorem 3.

We start by investigating the behavior of the three-layer BP network based on the gradient descent method in batch mode for the following four classification problems (Tables 1−4): (1) square-circle classification problem, (2) continuous XOR classification problem, (3) two-spiral classification problem, (4) 5-bit parity classification problem. The total numbers of the training samples are 400, 400, 98, and 32, respectively, each of which is divided into two classes with an equal number of samples. Thus, if a sigmoid function is used as the transfer function, then the BP network for solving these benchmark problems will satisfy Condition (ii) in Theorem 1. Fig. 2 shows the training samples of the first three problems. The square error function is employed and the transfer function for both hidden and output layers is the logsig function $g(x) = 1/(1 + \mathrm{e}^{-x})$. Other training parameters are chosen as follows: the learning rate is 0.1; the numbers of the hidden neurons are $6, 6, 30, 10$, respectively; the maximum numbers of iterations are 5000, 5000, 50 000, and 10 000, respectively.

In these four, as well as in the other three, experiments, the initial weights are randomly generated in certain ranges as specified in the tables. The 'success rate' indicates the percentage of the training samples that are successfully classified in the end of the training iteration. The 'average error' in the tables is the result of averaging over 10 training procedures.

Table 1 Results for the square-circle classification problem

Range of initial weights	Average error	Average success rate (%)
$[-0.005, 0.005]$	63.9776	51.23
$[-0.01, 0.01]$	66.1019	53.64
$[-0.02, 0.02]$	27.5531	81.79
$[-0.04, 0.04]$	14.9078	90.20
$[-0.08, 0.08]$	2.9076	99.00
$[-0.1, 0.1]$	2.9728	98.80
$[-0.5, 0.5]$	1.6304	99.50
$[-1, 1]$	1.4826	99.55
$[-5, 5]$	2.8227	98.63
$[-8, 8]$	7.1010	96.64
$[-15, 15]$	27.0526	86.00

Table 2 Results for the continuous XOR classification problem

Range of initial weights	Average error	Average success rate (%)
$[-0.005, 0.005]$	64.3643	49.58
$[-0.01, 0.01]$	65.3097	52.83
$[-0.02, 0.02]$	64.9070	53.90
$[-0.04, 0.04]$	39.7133	75.40
$[-0.08, 0.08]$	20.1743	85.53
$[-0.1, 0.1]$	19.4074	87.68
$[-0.5, 0.5]$	5.6972	97.07
$[-1, 1]$	2.5869	99.30
$[-5, 5]$	2.4817	99.00
$[-10, 10]$	25.0396	86.35

Table 3 Results for the two-spiral classification problem

Range of initial weights	Average error	Average success rate (%)
$[-0.005, 0.005]$	14.1214	53.61
$[-0.01, 0.01]$	14.1138	53.06
$[-0.05, 0.05]$	14.1380	53.47
$[-0.2, 0.2]$	14.1167	53.37
$[-0.6, 0.6]$	11.8426	66.43
$[-1, 1]$	6.8395	82.55
$[-2, 2]$	3.9638	90.41
$[-5, 5]$	2.6730	94.78
$[-8, 8]$	4.9588	90.00
$[-10, 10]$	7.7583	84.29

Table 4 Results for the 5-bit parity classification problem

Range of initial weights	Average error	Average success rate (%)
$[-0.005, 0.005]$	4.0000	50.31
$[-0.01, 0.01]$	4.0000	52.19
$[-0.08, 0.08]$	3.6704	62.50
$[-0.1, 0.1]$	3.1627	78.44
$[-0.4, 0.4]$	1.7266	100.00
$[-0.6, 0.6]$	1.3226	99.88
$[-1, 1]$	1.1630	99.75
$[-5, 5]$	0.9334	92.84
$[-8, 8]$	1.8992	85.44
$[-10, 10]$	2.3455	81.75

We see clearly in these experiments that, as revealed by our theorems, up to a certain critical point, smaller initial weights result in slower convergence and larger initial weights lead to better convergence. But after that point, as is well known to the neural network community, the convergence increasingly worsens for larger initial weights.

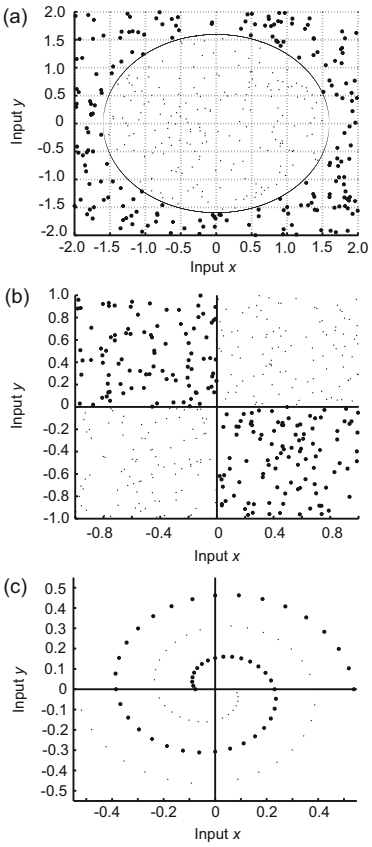

Fig. 2 Training samples for the square-circle problem (a), continuous XOR problem (b), and two-spiral problem (c)

Next, consider in Table 5 a three-layer BP neural network based on the gradient descent method in batch mode to accomplish the square-circle classification task, in which the transfer function is $g(x) = \mathrm{e}^{-x^2/2}$ satisfying Condition (i) in Theorem 1, i.e., $g'(0) = 0$. We let the numbers of samples of the two classes (with ideal outputs 1 and 0, respectively) be 200 and 150, respectively. Thus, it dissatisfies Conditions (ii) in Theorem 1. The number of hidden units is 6, the learning rate is 0.002, and the maximum number of training epochs is 5000. Table 5 shows the average results for 10 trials.

Table 6 is for a three-layer BP neural network based on the online gradient method to accomplish the three-bit parity classification task. The training patterns are supplied to the network in a completely stochastic order. $g(x) = \mathrm{e}^{-x^2/2}$ satisfies the condition $g'(0) = 0$ in Theorem 2. The number of hidden

Table 5 Results for the square-circle classification problem

Range of initial weights	Average error	Average success rate (%)
$[-0.005, 0.005]$	42.7096	57.14
$[-0.01, 0.01]$	30.5714	75.14
$[-0.02, 0.02]$	18.9219	86.37
$[-0.06, 0.06]$	12.5217	94.11
$[-0.08, 0.08]$	10.6432	95.80
$[-0.2, 0.2]$	10.1697	96.14
$[-0.8, 0.8]$	8.6064	97.91
$[-1, 1]$	11.3832	94.69
$[-5, 5]$	29.6105	82.34
$[-10, 10]$	36.7854	72.71

units is 5, the learning rate is 0.1, and the maximum number of training epochs is 5000. The average results for 10 trials are given in Table 6. Note that a similar situation occurs here for the online gradient learning method under the condition $g'(0) = 0$. But from our numerical experience, under Condition (ii) in Theorem 1 (which is a very important case since it is often satisfied when sigmoid functions are used), very small initial weights cause much less trouble for the online gradient method than for the off-line (batch) gradient method. The reason is that at each training step, the online gradient vector is generally not $\mathbf{0}$, though its expectation is, when the two classes have the same number of samples. Maybe this should be viewed as another advantage of the online gradient method over the off-line gradient method for neural networks.

Table 6 Results of the online gradient method for the 3-bit classification problem

Range of initial weights	Average error	Average success rate (%)
$[-0.005, 0.005]$	1.0698	50.00
$[-0.04, 0.04]$	1.1245	50.00
$[-0.06, 0.06]$	1.0425	60.00
$[-0.1, 0.1]$	0.9461	72.50
$[-0.5, 0.5]$	0.5615	95.00
$[-0.8, 0.8]$	0.4934	91.25
$[-2, 2]$	0.4951	88.75
$[-5, 5]$	0.9742	76.25
$[-8, 8]$	1.0262	73.75
$[-10, 10]$	1.4650	60.00

Table 7 is for a three-layer BP neural network based on the gradient descent method in batch mode to accomplish the continuous XOR classification task, with the transfer function $g(x) = \tanh x = (\mathrm{e}^{\beta x} - \mathrm{e}^{-\beta x})/(\mathrm{e}^{\beta x} + \mathrm{e}^{-\beta x})$ satisfying $g(0) = 0$, which is the condition in Theorem 3. We let the numbers of samples of the two classes (with ideal outputs -1 and 1, respectively) be 200 and 150, respectively. Thus, the conditions in Theorem 1 are not met. The number of hidden units is 8, the learning rate is 0.01, and the maximum number of training epochs is 5000. Table 7 gives the average results for 10 trials. We see that in this case, very small initial weights also lead to slow convergence as predicted by Theorem 3.

Table 7 Results for the continuous XOR classification problem

Range of initial weights	Average error	Average success rate (%)
$[-0.001, 0.001]$	195.5337	55.31
$[-0.004, 0.004]$	160.4651	63.26
$[-0.008, 0.008]$	89.5904	82.31
$[-0.02, 0.02]$	68.7012	88.17
$[-0.06, 0.06]$	37.1312	93.40
$[-0.08, 0.08]$	22.0969	96.29
$[-0.2, 0.2]$	7.3421	99.26
$[-0.8, 0.8]$	4.4031	99.94
$[-2, 2]$	19.1917	97.83
$[-5, 5]$	87.6273	87.91
$[-8, 8]$	109.4125	84.37

5 Other error functions and transfer functions

In this section, we try to apply our theory to a few other commonly used error functions and transfer functions for feedforward neural networks.

Consider the following error functions (Li *et al.*, 2001):

1. Square error function $E^1(\boldsymbol{w}, \boldsymbol{V})$ (Eq. (6)).
2. Measure of the cross entropy

$$E^2(\boldsymbol{w}, \boldsymbol{V}) := \sum_{j=1}^{J} \left[\frac{1}{2}(1 + O^j) \ln \frac{1 + O^j}{1 + g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))} + \frac{1}{2}(1 - O^j) \ln \frac{1 - O^j}{1 - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))} \right]. \tag{20}$$

3. Cauchy error function

$$E^3(\boldsymbol{w}, \boldsymbol{V}) := \sum_{j=1}^{J} \ln\{1 + [O^j - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))]^2\}. \tag{21}$$

4. Logistical error function ($\alpha > 0, \beta > 0$)

$$E^4(\boldsymbol{w}, \boldsymbol{V}) := \frac{\beta}{\alpha} \ln\{\cosh[\alpha(O^j - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j)))]\}. \tag{22}$$

5. A generalized error function ($0 \le \lambda \le 1$)

$$E^5(\boldsymbol{w}, \boldsymbol{V}) := \sum_{j=1}^{J} \{O^j[O^j - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))] + \frac{\lambda}{2}[g^2(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j)) - (O^j)^2]\}. \tag{23}$$

6. A linear combination error function

$$E^6(\boldsymbol{w}, \boldsymbol{V}) := \sum_{j=1}^{J} \left\{ \frac{1}{2}[O^j - g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))]^2 + \frac{1}{20}\left[(1+O^j)\ln\frac{1+O^j}{1+g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))} + (1-O^j)\ln\frac{1-O^j}{1-g(\boldsymbol{w} \cdot G(\boldsymbol{V}\boldsymbol{\xi}^j))}\right]\right\}. \tag{24}$$

To evaluate the gradients of the error functions with respect to weights, we first note that the gradient $E_{\boldsymbol{v}_i}(\boldsymbol{w}, \boldsymbol{V})|_{\boldsymbol{u}=\boldsymbol{0}} = \boldsymbol{0}$ is obviously true (cf. Eq. (19)). Similarly, the gradients with respect to weight vector $\boldsymbol{w}$ for these error functions can be easily computed.

Furthermore, we list five commonly used transfer functions as follows (Li *et al.*, 2001):

1. Logistic function

$$g_1(x) = \frac{1}{1+\mathrm{e}^{-2\beta x}}, \quad \beta > 0. \tag{25}$$

2. Hyperbolic tangent function

$$g_2(x) = \tanh x = \frac{\mathrm{e}^{\beta x} - \mathrm{e}^{-\beta x}}{\mathrm{e}^{\beta x} + \mathrm{e}^{-\beta x}}, \quad \beta > 0. \tag{26}$$

3. Inverse tangent function

$$g_3(x) = \frac{2}{\pi}\arctan(\beta x), \quad \beta > 0. \tag{27}$$

4. Error function

$$g_4(x) = \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_0^x \mathrm{e}^{-u^2}\mathrm{d}u. \tag{28}$$

5. Radial basis function

$$g_5(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right). \tag{29}$$

Some important values of the five transfer functions are given in Table 8, which will be used for our computation of the gradients with respect to the weights.

Table 8 Values of the transfer functions

	$g(-\infty)$	$g(0)$	$g(+\infty)$	$g'(0)$
$g_1(x)$	0	0.5	1	$\beta/2$
$g_2(x)$	−1	0	1	β
$g_3(x)$	−1	0	1	$2\beta/\pi$
$g_4(x)$	−1	0	1	$2/\sqrt{\pi}$
$g_5(x)$	0	1	0	0

In Table 9, we investigate whether our theories can be applied to different combinations of the above listed six error functions and five transfer functions. Our suggestion is: do not choose very small initial weights in the cases marked 'Y' or 'S' in the table.

Table 9 Gradients of different error functions with respect to w at the origin

	E^1	E^2	E^3	E^4	E^5	E^6
g_1	Y(ii)	N	Y(ii)	N	S(iii)	N
g_2	Y(ii)	S(iii)	Y(ii)	S(iii)	S(iii)	S(iii)
g_3	Y(ii)	S(iii)	Y(ii)	S(iii)	S(iii)	S(iii)
g_4	Y(ii)	S(iii)	Y(ii)	S(iii)	S(iii)	S(iii)
g_5	Y(i or ii)	Y(i)	Y(i)	Y(i)	Y(i)	Y(i)

E^1–E^6: error functions; g_1–g_5: transfer functions. Y (resp. N): the corresponding gradient of the error function with respect to $\boldsymbol{w}$ is (resp. is not) $\boldsymbol{0}$; S: the gradient of the error function with respect to $\tilde{\boldsymbol{w}}$ is $\boldsymbol{0}$ except for the bias b. '(i)' and '(ii)' correspond to Conditions (i) and (ii) in Theorem 1 respectively, and '(iii)' stands for the condition $g(0) = 0$ in Theorem 3

6 Conclusions

It is a well-known and widely adopted strategy that in the network training of feedforward neural networks, the initial weights are chosen to be small in magnitude in order to prevent premature saturation in the training procedure. However, this paper points out that in some cases, very small initial weights may be ineffective in that they result in slow convergence in the beginning of the training procedure. Our results suggest that, in these cases, the initial values of the weights should be neither too large, nor too small. For instance, a typical range for choosing the initial weights might be something like $(-0.4, -0.1) \cup (0.1, 0.4)$, rather than $(-0.1, 0.1)$ as suggested by the usual strategy. Numerical experiments are carried out to support our theoretical findings.

First, we reveal that the gradient of the square

error function will be **0** if the transfer function $g(x)$ satisfies at $x = 0$ one of the two conditions: (i) $g'(0) = 0$, for either the batch or online BP gradient method; (ii) The batch BP gradient method is applied, and the network is used to solve a classification problem such that the numbers of the two classes of training samples are equal and that $g(0) = (O^+ + O^-)/2$, where O^+ and O^- are the target outputs for the two classes, respectively.

Second, if the transfer function satisfies (iii) $g(0) = 0$, then for both the batch and online BP gradient methods, the gradient of the square error function is **0**, except that the partial derivative with respect to the bias b of the output neuron might not be zero.

Our conclusion for these cases is drawn through the following argument: small values of the weights $\rightarrow$ small values of the gradients $\rightarrow$ slow convergence.

A few commonly used transfer functions and error functions are investigated. Our theory can be extended to most combinations of these transfer functions and error functions.

References

Bao, J., Chen, Y., Yu, J., 2010. A regeneratable dynamic differential evolution algorithm for neural networks with integer weights. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **11**(12):939-947. [doi:10.1631/jzus.C1000137]

Biswajeet, P., Saied, P., 2010. Comparison between prediction capabilities of neural network and fuzzy logic techniques for landslide susceptibility mapping. *Dis. Adv.*, **3**(2):26-34.

Castillo, P.A., Carpio, J., Merelo, J.J., Prieto, A., Rivas, V., Romero, G., 2000. Evolving multilayer perceptrons. *Neur. Process. Lett.*, **12**(2):115-128. [doi:10.1023/A:1009684907680]

Deng, Y., He, X., Zhao, J., Xiong, Y., Shen, Y., Jiang, J., 2010. Application of artificial neural network for switching loss modeling in power IGBTs. *J. Zhejiang Univ.-Sci. C (Comput. & Electron.)*, **11**(6):435-443. [doi:10.1631/jzus.C0910442]

Drago, G.P., Ridella, S., 1992. Statistically controlled activation weight initialization. *IEEE Trans. Neur. Networks*, **3**(4):627-631. [doi:10.1109/72.143378]

Elman, J.L., 1993. Learning and development in neural networks: the importance of starting small. *Cognition*, **48**(1):71-99. [doi:10.1016/0010-0277(93)90058-4]

Hagan, M.T., Demuth, H.B., Beale, M., 1996. Neural Network Design. PWS Publishing Company, Boston, MA.

Ham, F.M., Kostanic, I., 2001. Principles of Neurocomputing for Science and Engineering. McGraw-Hill, New York.

Kathirvalavakumar, T., Thangavel, P., 2003. A new learning algorithm using simultaneous perturbation with weight initialization. *Neur. Process. Lett.*, **17**(1):55-68. [doi:10.1023/A:1022919300793]

Li, Z., Wu, W., Zhang, H., 2001. Convergence of on-line gradient methods for two-layer feedforward neural networks. *J. Math. Res. Exp.*, **21**:219-228.

Liu, M., Zhang, M., Yan, G., 2008. A new neural network model for the feedback stabilization of nonlinear systems. *J. Zhejiang Univ.-Sci. A*, **9**(8):1015-1023. [doi:10.1631/jzus.A0720122]

Ludermir, T.B., Yamazaki, A., Zanchettin, C., 2006. An optimization methodology for neural network weights and architectures. *IEEE Trans. Neur. Networks*, **17**(6):1452-1459. [doi:10.1109/TNN.2006.881047]

Pradhan, B., 2011. An assessment of the use of an advanced neural network model with five different training strategies for the preparation of landslide susceptibility maps. *J. Data Sci.*, **9**(1):65-81.

Pradhan, B., Buchroithner, M.F., 2010. Comparison and validation of landslide susceptibility maps using an artificial neural network model for three test areas in Malaysia. *Envir. Eng. Geosci.*, **16**(2):107-126. [doi:10.2113/gseegeosci.16.2.107]

Pradhan, B., Youssef, A.M., Varathrajoo, R., 2010. Approaches for delineating landslide hazard areas using different training sites in an advanced artificial neural network model. *Geo-spat. Inf. Sci.*, **13**(2):93-102. [doi:10.1007/s11806-010-0236-7]

Qi, H., Zhao, H., Liu, W., Zhang, H., 2009. Parameters optimization and nonlinearity analysis of grating eddy current displacement sensor using neural network and genetic algorithm. *J. Zhejiang Univ.-Sci. A*, **10**(8):1205-1212. [doi:10.1631/jzus.A0820564]

Wang, J., Wu, W., Zurada, J.M., 2011. Deterministic convergence of conjugate gradient method for feedforward neural networks. *Neurocomputing*, **74**(14-15):2368-2376. [doi:10.1016/j.neucom.2011.03.016]

Wu, W., Feng, G., Li, Z., Xu, Y., 2005. Deterministic convergence of an online gradient method for BP neural networks. *IEEE Trans. Neur. Networks*, **16**(3):533-540. [doi:10.1109/TNN.2005.844903]

Xiong, Y., Wu, W., Kang, X., Zhang, C., 2007. Training pi-sigma network by online gradient algorithm with penalty for small weight update. *Neur. Comput.*, **19**(12):3356-3368. [doi:10.1162/neco.2007.19.12.3356]

Yam, J.Y.F., Chow, T.W.S., 2001. Feedforward networks training speed enhancement by optimal initialization of the synaptic coefficients. *IEEE Trans. Neur. Networks*, **12**(2):430-434. [doi:10.1109/72.914538]

Yam, Y.F., Chow, T.W.S., Leung, C.T., 1997. A new method in determining initial weights of feedforward neural networks for training enhancement. *Neurocomputing*, **16**(1):23-32. [doi:10.1016/S0925-2312(96)00058-6]

Yang, S.S., Siu, S., Ho, C.L., 2008. Analysis of the initial values in split-complex backpropagation algorithm. *IEEE Trans. Neur. Networks*, **19**(9):1564-1573. [doi:10.1109/TNN.2008.2000805]

Zeng, X.Q., Wang, Y.F., Zhang, K., 2006. Computation of Adalines' sensitivity to weight perturbation. *IEEE Trans. Neur. Networks*, **17**(2):515-519. [doi:10.1109/TNN.2005.863418]

Zhang, N., Wu, W., Zheng, G., 2006. Convergence of gradient method with momentum for two-layer feedforward neural networks. *IEEE Trans. Neur. Networks*, **17**(2):522-525. [doi:10.1109/TNN.2005.863460]