Neurocomputing van der Pauw function for the measurement of a semiconductor's resistivity without use of the learning rate of weight vector regulation

Li Hongli(李宏力)[1,2], Sun Yicai(孙以材)[1,†], Wang Wei(王伟)[1], and Harry Hutchinson[1]

[1]Hebei University of Technology, Tianjin 300130, China
[2]Tianjin Vocational Institute, Tianjin 300410, China

Abstract: Van der Pauw's function is often used in the measurement of a semiconductor's resistivity. However, it is difficult to obtain its value from voltage measurements because it has an implicit form. If it can be expressed as a polynomial, a semiconductor's resistivity can be obtained from such measurements. Normally, five orders of the abscissa can provide sufficient precision during the expression of any non-linear function. Therefore, the key is to determine the coefficients of the polynomial. By taking five coefficients as weights to construct a neuronetwork, neurocomputing has been used to solve this problem. Finally, the polynomial expression for van der Pauw's function is obtained.

Key words: measurement of the semiconductor's resistivity; van der Pauw function; reversal development; neurocomputing; polynomial match; learning rate of weight vector regulation

DOI: 10.1088/1674-4926/32/12/122002 **EEACC:** 2520

1. Introduction

Van der Pauw's function, f, is often used in the measurement of a semiconductor's resistivity R_s[1,2]:

$$R_{\mathrm{s}} = \frac{\pi}{\ln 2}\frac{v_1+v_2}{I} f\left(\frac{v_1}{v_2}\right), \tag{1}$$

where v_1 and v_2 are the measured voltages, I is the injection current, and f is van der Pauw's function:

$$\frac{(v_1/v_2)-1}{(v_1/v_2)+1} = \frac{\operatorname{arcosh}\left(\dfrac{1}{2}\exp\dfrac{\ln 2}{f}\right)}{\ln 2/f}, \quad v_1 \geqslant v_2. \tag{2}$$

However, it is difficult to obtain values of f from any measured values of v_1/v_2 because it has an implicit form (Eq. (2)). If we express it as a polynomial:

$$f = 1 + w_1(v_1/v_2) + w_2(v_1/v_2)^2 + w_3(v_1/v_2)^3 + w_4(v_1/v_2)^4 + w_5(v_1/v_2)^5.$$

The material's resistivity will be obtained from the measurement values (i.e. v_1/v_2), by using Eq. (1):

$$R_{\mathrm{s}} = \frac{\pi}{\ln 2}\frac{v_1+v_2}{I}\left[1 + w_1(v_1/v_2) + w_2(v_1/v_2)^2 + w_3(v_1/v_2)^3 + w_4(v_1/v_2)^4 + w_5(v_1/v_2)^5\right].$$

Normally, five orders of the abscissa, x, provide sufficient precision to express any non-linear function. Therefore, the key is to obtain the coefficients, w_i. Neurocomputing has been used to solve this problem.

2. Neurocomputing[3,4]

The five-order polynomial can be expressed as $y = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + w_5x^5$ for the van der Pauw function. Here $y = f - 1$ and $x = v_1/v_2$. If the polynomial is correct, w_i will be optimized. Therefore, we search for such w_i while satisfying $x = 1$ to 10, which is known as the global reversal development of van der Pauw's function.

2.1. Sampling

We obtain approximate values of van der Pauw's function, f, at several points v_1/v_2 (i.e. x), relying on local reversal development. The results are shown in Table 1 and taken as the so-called samples for neurocomputing. Here 10 samples are sufficient for the global reversal development of van der Pauw's function. Such sample values of van der Pauw's function are denoted as y_j^* to discriminate y_j, where $j = 1$ to 10. The former are expectated values whereas the latter are computed values: $y_j = \sum_i^5 w_i x_j^i$.

The object of the neurocomputing is to regulate w_i such that y_j tends to each known sample y_j^* respectively.

2.2. Neuronetwork

Figure 1 shows a neuronetwork comprising five weights, w_i ($i = 1, 2, \cdots, 5$). The input layer comprises five weights, i.e. the vector W, with the corresponding x^1, x^2, x^3, x^4, x^5. The hidden layer comprises x_j, which are points, j, on abscissa, x. The output layer comprises different computed y_j. The samples are y_j^*, which are the values of van der Pauw's function at x_j. The errors are the differences between y_j and y_j^*. The output from the hidden layer is $Y = X^TW$, i.e. $W = (w_1, w_2, w_3, w_4, w_5)^T$,

† Corresponding author. Email: syc@hebut.edu.cn

Received 17 September 2011, revised manuscript received 29 September 2011

Table 1. The approximate values of f at several v_1/v_2, relying on local reversal development ($f_1 = 1$ when $v_1/v_2 = 1$).

v_1/v_2	2 ± 0.00001	3 ± 0.00001	4 ± 0.00001	5 ± 0.00001	6 ± 0.00001	7 ± 0.00001	8 ± 0.00001	9 ± 0.00001	10 ± 0.00001
f	0.96028	0.90876	0.86092	0.82061	0.789283	0.76297	0.738119	0.71500	0.699259
$f-1$	−0.03972	−0.09124	−0.139708	−0.17939	−0.210717	0.23703	−0.261882	0.28500	−0.300741

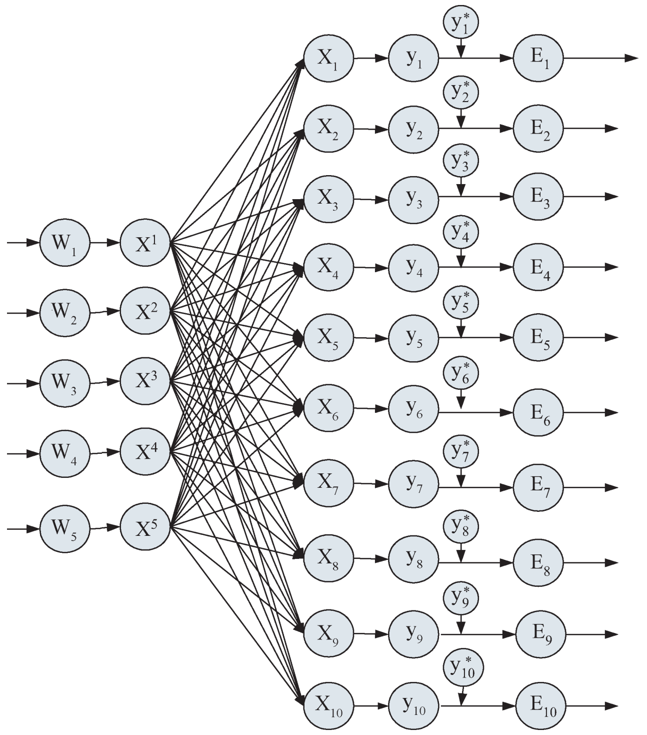

Fig. 1. The constructed neuronetwork.

$$Y = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8, y_9, y_{10})^T = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_9 \\ y_{10} \end{bmatrix}$$

$$= \begin{bmatrix} x_1 & x_1^2 & x_1^3 & x_1^4 & x_1^5 \\ x_2 & x_2^2 & x_2^3 & x_2^4 & x_2^5 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_9 & x_9^2 & x_9^3 & x_9^4 & x_9^5 \\ x_{10} & x_{10}^2 & x_{10}^3 & x_{10}^4 \cdot & x_{10}^5 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}. \tag{3}$$

For example, $y_1 = w_1x_1 + w_2x_1^2 + w_3x_1^3 + w_4x_1^4 + w_5x_1^5$, $y_{10} = w_1x_{10} + w_2x_{10}^2 + w_3x_{10}^3 + w_4x_{10}^4 + w_5x_{10}^5$.

The arrows in Fig. 1 indicate the mathematical relationships (in accordance with Eq. (3)). This neuronetwork can also be expressed as a simple symbol, which is shown in the first row of Fig. 2.

2.3. Errors and their variations

There are 10 errors $E_j = y_j - y_j^*$ (j = 1 to 10) corresponding to 10 samples that arise from arbitral weight vector

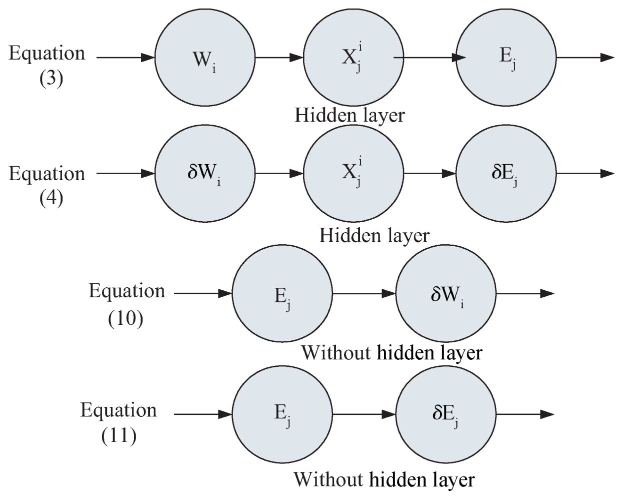

Fig. 2. Four symbols for different equations.

W: $E_j = y_j - y_j^* = \sum_i^5 w_i x_j^i$ (j = 1 to 10). Variations in the weights will induce corresponding variations of errors:

$$\delta E_j(k) = \delta(y_j - y_j^*) = \sum_i^5 \delta w_i x_j^i, \quad j = 1, 2, \cdots, 10. \tag{4}$$

In a manner similar to Eq. (3), Equation (4) can also be realized via a neuronetwork. A difference as compared to the neuronetwork of Fig. 1 is found in the input layer comprising δw_i and the output layer comprising δE_j; it should be noted that the hidden layer is the same in both cases. It can be simplified as another symbol, shown in the second row of Fig. 2.

2.4. Selection of the variations of weights

2.4.1. Prevailing weight training[3, 4]

Normally, weight training any neural network relies on a learning rate that will efficiently guide the weight vector W to a location that yields the desired network performance. There have been many different approaches to solving this problem. A commonly encountered goal is to move W to a position that minimizes some particular neural network performance function, such as the mean square error deviation. It is desirable to select a W vector so that the mean square error of y when compared with y' is minimized. In other words, the goal is to find an optimum W that minimizes the mean square error. The half sum of the square error is

$$F(w) = \frac{1}{2}\frac{1}{10}\sum_{j=1}^{10}\left|E_j\right|^2 = \frac{1}{2}\frac{1}{10}\sum_{j=1}^{10}\left(y_j' - \sum_i^5 w_i x_j^i\right)^2. \tag{5}$$

The vector $-\nabla F$ points the direction in which $F(w)$ will decrease at the fastest possible rate. If $-\nabla F$ could be calculated or estimated, the weight vector will be moved in the " downhill" direction by a small amount. The problem is estimating $\nabla_{\mathrm{wi}} F$[3, 4].

$$\nabla_{\mathrm{wi}} F(w) = \frac{1}{10}\sum_{j=1}^{10}(y'_j - y_j)\nabla_w\left(-\sum_i^5 w_i x_j^i\right) = \frac{1}{10}\sum_{j=1}^{10}(y'_j - y_j)(-x_j^i). \quad (6)$$

If the error, $y_j - y'_j$, is denoted by E_j then

$$\nabla_{\mathrm{wi}} F(w) = -\frac{1}{10}\sum_{j=1}^{10}E_j x_j^i = -\frac{1}{10}\sum_{j=1}^{10}E_j x_j^i. \quad (7)$$

Thus a weight updated law[3, 4] will converge to a correct W from any starting W_0 value:

$$\delta w_i = w_i(k+1) - w_i(k) = -\alpha_i \frac{1}{10}\sum_{j=1}^{10}E_j x_j^i, \quad (8)$$

$$\alpha_i = -\delta w_i \left(\frac{1}{10}\sum_{j=1}^{N}E_j x_j^i\right)^{-1}, \quad (9)$$

where α is a positive constant known as the learning rate[3, 4]. However, α is not easily calculated and is typically selected by trial and error[3, 4]. In general, if α is too large, the weight vector will not converge and, if α is too small, convergence will take an unduly long period of time.

There are some common variants of the above equation. The momentum variant is given by the following equation $W(k+1) = W(k) + (1-\alpha)E_j + \mu[W(k) - W(k-1)]$, where the value of μ is considerably larger than α, such as being chosen as 0.9 (typically, $0.01 < \alpha < 10$)[3, 4].

Although there are a great many learning laws that can adjust the weight vector W, a difficulty exists in selecting the learning rate α and the momentum variant value μ. Therefore, a problem arises, which is whether or not the choice of the learning rate can be simplified in neurocomputing. The answer is positive. This paper addresses simplifying and solving this problem.

2.4.2. The difficulties encountered by the prevailing weight training

Now consider the denominator in the right-hand side of Eq. (9), then

$$i = 1,\quad \sum_{j=1}^{10}E_j x_j = E_1x_1 + E_2x_2 + \cdots + E_{10}x_{10},$$

$$i = 2,\quad \sum_{j=1}^{10}E_j x_j^2 = E_1x_1^2 + E_2x_2^2 + \cdots + E_{10}x_{10}^2,$$

$$\cdots$$

$$i = 5,\quad \sum_{j=1}^{10}E_j x_j^5 = E_1x_1^5 + E_2x_2^5 + \cdots + E_{10}x_{10}^5. \quad (10)$$

If $x_{10} = 10$, $x_1 = 1$, then $x_{10}^5 = 10^5$. Thus $E_1 + 8E_2 + 243E_3 + 1024E_4 + 3125E_5 + \cdots + 10^5E_{10}$ is quite different from $E_1 + 2E_2 + 3E_3 + 4E_4 + 5E_5 + \cdots + 10E_{10}$ respectively for α_5 and α_1. A polynomial of any non-linear function, there is $|w_5| \ll |w_1|$ (as can be appreciated from Section 3), so there should be $|\delta w_5| \ll |\delta w_1|$ to move weights a small step. So values of $|\alpha_5|$ are much lower than values of $|\alpha_1|$. They differ from each other by at least three orders. Therefore, the learning rate, α, is difficult to select from Eq. (9). Additionally, Equations (4) and (8) give

$$\delta E_1 = x_1\delta w_1 + \cdots + x_1^5\delta w_5 = -\frac{1}{10}\left(\alpha_1\sum_{j=1}^{10}E_jx_j + \cdots + \alpha_5\sum_{j=1}^{10}E_jx_j^5\right),$$

$$\delta E_{10}(k) = x_{10}\delta w_1 + \cdots + x_{10}^5\delta w_5 = -\left(\alpha_1\sum_{j=1}^{10}E_jx_j + \cdots + \alpha_5 10^4\sum_{j=1}^{10}E_jx_j^5\right).$$

So,

$$\partial(\delta E_1)/\partial\alpha_1 : \partial(\delta E_{10})/\partial\alpha_1 : \partial(\delta E_1)/\partial\alpha_5 : \partial(\delta E_{10})/\partial\alpha_5 = \frac{1}{10}\sum_{j=1}^{10}E_jx_j : \sum_{j=1}^{10}E_jx_j : \frac{1}{10}\sum_{j=1}^{10}E_jx_j^5 : 10^4\sum_{j=1}^{10}E_jx_j^5.$$

The influences of the learning rates upon the respective errors are quite different, which emphasizes the difficulties in choosing learning rates and their variations. References [3, 4] have pointed out: typically, $0.01 < \alpha < 10$, with a value of 0.1 often being a starting value. Choosing and adjusting α for training samples is an art that neurocomputing practitioners need to learn. If it takes 10^3 times to choose a correct learning rate for one weight, then it would take 10^{10} to 10^{15} times to choose the learning rates for five weights. Therefore, this case is only appropriate to the following neurocomputing expression, which is different from Eq. (3):

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_9 \\ y_{10} \end{bmatrix} = \cdot \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ x_9^{(1)} & x_9^{(2)} & x_9^{(3)} \\ x_{10}^{(1)} & x_{10}^{(2)} & x_{10}^{(3)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix},$$

where (i) represents different physical quantities, such as temperatures, pressure, etc. Normally, the number of physical quantities involved is lower than five. Therefore, (i) is not the power on x_j, but a symbol, representing some physical quantity. Thus, $\sum_{j=1}^{N} E_j x_j^{(i)}$ are the same for one w_i and the learning rate is easily chosen because the influence of the learning rates on the errors to the samples depends upon the ratio of the sample's values $x_j^{(i)}$ for different physical quantities (i):

$$\partial(\delta E_1)/\partial\alpha_1 : \partial(\delta E_{10})/\partial\alpha_1 = x_1^{(1)} : x_{10}^{(1)},$$

$$\partial(\delta E_1)/\partial\alpha_3 : \partial(\delta E_{10})/\partial\alpha_3 = x_1^{(3)} : x_{10}^{(3)},$$

$$\partial(\delta E_1)/\partial\alpha_1 : \partial(\delta E_1)/\partial\alpha_3 = x_1^{(1)}\sum_{j=1}^{N} E_j x_j^{(1)} : x_1^{(3)}\sum_{j=1}^{N} E_j x_j^{(3)},$$

$$\partial(\delta E_{10})/\partial\alpha_1 : \partial(\delta E_{10})/\partial\alpha_3 = x_{10}^{(1)}\sum_{j=1}^{N} E_j x_j^{(1)} : x_{10}^{(3)}\sum_{j=1}^{N} E_j x_j^{(3)}.$$

If $x_j^{(i)}$ is normalized for different physical quantities (i), for example, $x_1^{(i)} = 0.1, \cdots, x_{10}^{(i)} = 1$, then $x_1^{(1)}\sum_{j=1}^{N} E_j x_j^{(1)} = x_1^{(3)}\sum_{j=1}^{N} E_j x_j^{(3)}$ and $x_{10}^{(1)}\sum_{j=1}^{N} E_j x_j^{(1)} = x_{10}^{(3)}\sum_{j=1}^{N} E_j x_j^{(3)}$. So,

$$\partial(\delta E_1)/\partial\alpha_1 : \partial(\delta E_1)/\partial\alpha_3 = x_1^{(1)}\sum_{j=1}^{N} E_j x_j^{(1)} : x_1^{(3)}\sum_{j=1}^{N} E_j x_j^{(3)} = 1,$$

$$\partial(\delta E_{10})/\partial\alpha_1 : \partial(\delta E_{10})/\partial\alpha_3 = x_{10}^{(1)}\sum_{j=1}^{N} E_j x_j^{(1)} : x_{10}^{(3)}\sum_{j=1}^{N} E_j x_j^{(3)} = 1.$$

It can be appreciated from the last two expressions above that $\alpha_1 = \alpha_2 = \alpha_3$ are the same and are independent of the physical quantities involved. In this case, prevailing weight training techniques[3, 4] are appropriate for easily choosing a universal learning rate by trial and error or empirically using the above expressions. However, we believe that it is difficult to choose a universal expression for the learning rate (Eq. (9)) for different weights in the case of polynomial matching.

2.4.3. Adopting an universal expression of the variations of weights

Therefore, we propose to abandon the above types of weight training that rely on selecting learning rates in favor of adopting a universal expression for the variations of weights for global reversal development for any non-linear function.

From Eq. (4), it can be seen that the variations of each error are dependent upon the variations of weights. Whether errors will be reduced depends upon the selection of the magnitude of the variations of weights. We select the δw_i as follows:

$$\delta W_i = -\frac{1}{m}\frac{1}{x_m^i}\left\{E_i + \frac{1}{m}\left[(1-\delta_{k,j})\sum_{k=6}^{m}|E_k|\right] + \delta_{kj}E_k\right\},$$
$$i = 1, 2, \cdots, 5, j = 1, 2, \cdots, m, k = 6, 7, \cdots, m, \tag{11}$$

where m is the maximum number of nodes in the hidden layer, i.e. number of samples, a maximum of five weight components (coefficients of the polynomial) is selected and j and k are the number of nodes. E_i are the deviations of the computing function $Y = X^T W$ relative to the expectations Y' for points $j = i = 1$ to 5 and E_k are for points $j = k = 6$ to m respectively on the abscissa, the i of x_m^i is the power of x_m, δw_i denotes variation of w_i. The number m in x_m^i is required for a hidden layer with m nodes. Now introducing each δw_i into Eq. (4) and choosing $m = 10$, i.e. the numbers of samples, one obtains:

$$\delta E_1 = -\frac{1}{10}\left(\left\{\frac{1}{10}E_1 + \left(\frac{1}{10}\right)^2 E_2 + \left(\frac{1}{10}\right)^3 E_3 + \left(\frac{1}{10}\right)^4 E_4 + \left(\frac{1}{10}\right)^5 E_5 + \frac{1}{10}\left[\frac{1}{10} + \left(\frac{1}{10}\right)^2 + \left(\frac{1}{10}\right)^3 + \left(\frac{1}{10}\right)^4 + \left(\frac{1}{10}\right)^5\right](|E_6| + |E_7| + |E_8| + |E_9| + |E_{10}|)\right\}\right),$$

$$\delta E_2 = -\frac{1}{10}\left(\left\{\frac{2}{10}E_1 + \left(\frac{2}{10}\right)^2 E_2 + \left(\frac{2}{10}\right)^3 E_3 + \left(\frac{2}{10}\right)^4 E_4 + \left(\frac{2}{10}\right)^5 E_5 + \frac{1}{10}\left[\frac{2}{10} + \left(\frac{2}{10}\right)^2 + \left(\frac{2}{10}\right)^3 + \left(\frac{2}{10}\right)^4 + \left(\frac{2}{10}\right)^5\right](|E_6| + |E_7| + |E_8| + |E_9| + |E_{10}|)\right\}\right),$$

$$\cdots$$

$$\delta E_5 = -\frac{1}{10}\left(\left\{\frac{5}{10}E_1 + \left(\frac{5}{10}\right)^2 E_2 + \left(\frac{5}{10}\right)^3 E_3 + \left(\frac{5}{10}\right)^4 E_4 + \left(\frac{5}{10}\right)^5 E_5 + \frac{1}{10}\left[\frac{5}{10} + \left(\frac{5}{10}\right)^2 + \left(\frac{5}{10}\right)^3 + \left(\frac{5}{10}\right)^4 + \left(\frac{5}{10}\right)^5\right](|E_6| + |E_7| + |E_8| + |E_9| + |E_{10}|)\right\}\right),$$

$$\delta E_6 = -\frac{1}{10}\left\{\frac{6}{10}E_1 + \left(\frac{6}{10}\right)^2 E_2 + \left(\frac{6}{10}\right)^3 E_3 + \left(\frac{6}{10}\right)^4 E_4 + \left(\frac{6}{10}\right)^5 E_5 + \frac{1}{10}\left[\left(\frac{6}{10} + \left(\frac{6}{10}\right)^2 + \left(\frac{6}{10}\right)^3 + \left(\frac{6}{10}\right)^4 + \left(\frac{6}{10}\right)^5\right)\right](10E_6 + |E_7| + |E_8| + |E_9| + |E_{10}|)\right\},$$

$$\cdots$$

$$\delta E_8 = -\frac{1}{10}\left\{\frac{8}{10}E_1 + \left(\frac{8}{10}\right)^2 E_2 + \left(\frac{8}{10}\right)^3 E_3 + \left(\frac{8}{10}\right)^4 E_4 + \left(\frac{8}{10}\right)^5 E_5 + \frac{1}{10}\left[\frac{8}{10} + \left(\frac{8}{10}\right)^2 + \left(\frac{8}{10}\right)^3 + \left(\frac{8}{10}\right)^4 + \left(\frac{8}{10}\right)^5\right](|E_6| + |E_7| + 10E_8 + |E_9| + |E_{10}|)\right\},$$

$$\cdots$$

$$\delta E_{10} = -\frac{1}{10}\left\{E_1 + E_2 + E_3 + E_4 + E_5 + \frac{1}{10}[(1+1+1+1+1)](|E_6| + |E_7| + |E_8| + |E_9| + 10E_{10})\right\}. \quad (12)$$

The above equations are independent of each other without any coupling. If all E_j are known, any δE_j will be directly obtained, being no need to go through the hidden layer, which is in contrast to Eq. (3).

This is a key advantage of Eq. (11). The second advantage is its universality for different orders and ease of compiling the program.

2.5. Error iteration reducing

Errors can be reduced by relying on iterations with their variations, depending on the magnitude of variations. The principle can be essentially demonstrated via three examples, which are given below.

2.5.1. One-dimensional (sample) error iteration reduction

The relationship between errors for successive iterations can be represented as $E(k+1) = E(k) + \delta E(k)$ and $\delta E(k) = -nE(k)$ during iterations of the error E. If $n = 0.1$, 0.25, 0.5, then E will decrease to 0.729, 0.422, 0.125 respectively after three iterations and will tend to zero after many iterations. If $n = 1$, then E will immediately decrease to zero after one iteration. If $n = 1.2$, 1.4, 1.8, then E will assume negative values (−0.2, −0.4, −0.8) respectively after one iteration, while converging to −0.008, −0.064, −0.0512 after three iterations and will tend to zero after many iterations. If $n = 2$, then E will oscillate, taking either values of $-E$ or $+E$ for successive iterations. If n is larger than 2, then E will diverge.

2.5.2. Two-dimensional (sample) error iteration reduction

The relationship between errors for successive iterations can be expressed as $E_1(k+1) = E_1(k) + \delta E_1(k)$, $E_2(k+1) = E_2(k) + \delta E_2(k)$ and $\delta E_1(k) = -\frac{1}{2}\left[(E_1(k) + \frac{1}{2}E_2(k))\right]$, $\delta E_2(k) = -\frac{1}{2}\left[E_2(k) + \frac{1}{4}E_1(k)\right]$ during iterations of the error E. After one time iteration, it will be respectively obtained that $E_1(1) = \frac{1}{2}E_1 - \frac{1}{4}E_2$, $E_2(1) = \frac{1}{2}E_2 - \frac{1}{8}E_1$ and after two times iterations $E_1(2) = \frac{9}{32}E_1 - \frac{3}{16}E_2$, $E_2(2) = \frac{9}{32}E_2 - \frac{2}{16}$ for an initial E_1 and E_2. It can be appreciated that each error reduces progressively depending upon the ratio factors (here −1/2 as an example) taken for δE_i in two-dimensional error iteration reduction.

2.5.3. Six-dimensional (sample) error iteration reduction

According to Eq. (11), we select the following variation of weights:

$$\delta W_i = -\frac{1}{6}\frac{1}{x_6^i}\left\{E_i + \frac{1}{6}\left[(1-\delta_{6,j})|E_6|\right] + \delta_{6j}E_6\right\}, \quad i = 1,2,\cdots,5,\ j = 1,2,\cdots,6. \quad (13)$$

Substituting it into Eq. (4), gives the following equations:

$$\delta E_1 = -\frac{1}{6}\left\{\frac{1}{6}E_1 + \left(\frac{1}{6}\right)^2 E_2 + \left(\frac{1}{6}\right)^3 E_3 + \left(\frac{1}{6}\right)^4 E_4 + \left(\frac{1}{6}\right)^5 E_5 + \frac{1}{6}\left[\frac{1}{6} + \left(\frac{1}{6}\right)^2 + (\frac{1}{6})^3 + \left(\frac{1}{6}\right)^4 + \left(\frac{1}{6}\right)^5\right]|E_6|\right\},$$

$$\delta E_2 = -\frac{1}{6}\left\{\frac{2}{6}E_1 + \left(\frac{2}{6}\right)^2 E_2 + \left(\frac{2}{6}\right)^3 E_3 + \left(\frac{2}{6}\right)^4 E_4 + \left(\frac{2}{6}\right)^5 E_5 + \frac{1}{6}\left[\frac{2}{6} + \left(\frac{2}{6}\right)^2 +]\left(\frac{2}{6}\right)^3 + \left(\frac{2}{6}\right)^4 + \left(\frac{2}{6}\right)^5\right]|E_6|\right\},$$

$$\cdots$$

$$\delta E_5 = -\frac{1}{6}\left\{\frac{5}{6}E_1 + \left(\frac{5}{6}\right)^2 E_2 + \left(\frac{5}{6}\right)^3 E_3 + \left(\frac{5}{6}\right)^4 E_4 + \left(\frac{5}{6}\right)^5 E_5 + \frac{1}{6}\left[\frac{5}{6} + \left(\frac{5}{6}\right)^2 + \left(\frac{5}{6}\right)^3 + \left(\frac{5}{6}\right)^4 + \left(\frac{5}{6}\right)^5\right]|E_6|\right\},$$

$$\delta E_6 = -\frac{1}{6}\left[E_1 + E_2 + E_3 + E_4 + E_5 + \frac{5}{6}E_6\right]. \quad (14)$$

Equation (14) is independent and uncoupled. This is the foundation of performing an error iteration reduction. Each variation of error for k iterations, $\delta E_j(k)$, can be directly obtained for all errors by solving these independent equations (14). A symbol can represent solving these equations (14 or 12): $E_j(k) \to \delta E_j(k)$, here the hidden layer is deleted. If we use error iteration reduction: $E_j(k) \to \delta E_j(k)$, and $E_j(k+1) = E_j(k) + \delta E_j(k)$, each error can tend gradually to zero. Tables 2 and 3 show the error reductions during the iterations for 6 samples with different initial errors. It should be noted, however, that there is a slight oscillation. Here the error iterations do not involve any weights. The calculations are performed independently. We call this "error iteration reduction". This is the third advantage of using Eq. (11).

Table 2. The magnitude of each E_j ($j = 1, 2, \cdots, 6$) after several iterations for the first case.

Iteration time	0	1	5	10	20	30	50	100	1000	3000	8000
E_1	−5	−3.5490	−2.9211	−1.4838	−1.5692	−1.3019	−0.4063	−0.0834	−0.0145	−0.0027	−3.9569 (10^{-5})
E_2	−4.2	−0.7630	−0.7395	1.3689	−0.2516	0.1722	0.8099	0.4708	0.0410	0.0077	1.1558 (10^{-4})
E_3	−3.3	2.9344	0.0167	2.2428	−1.5294	−0.2861	−0.5830	−0.6787	−0.2143	−0.0398	−5.9355 (10^{-4})
E_4	−2.5	7.7444	−1.9528	1.0640	−3.2411	0.1222	−2.0059	−0.8100	0.2748	0.0512	7.6777 (10^{-4})
E_5	−1.2	14.8214	−7.4114	−0.1790	0.7284	7.6156	1.0679	1.1813	−0.1843	−0.0343	−5.1036 (10^{-4})
E_6	0	24.3000	5.9695	17.0460	30.1814	17.5932	3.8469	0.2386	0.0056	−4.2807 (10^{-4})	−2.0573 (10^{-5})

Table 3. The magnitude of each E_j ($j = 1, 2, \cdots, 6$) after several iterations computing for the second case.

Iteration time	0	1	5	10	20	30	50	100	1000	3000	8000
E_1	0	−0.1970	−0.9149	−0.8242	−0.3012	−0.2235	−0.1513	−0.0148	−3.8821 (10^{-4})	−7.0384 (10^{-5})	−1.0784 (10^{-6})
E_2	1.2	0.4629	−1.0120	−0.8148	0.0863	−0.0047	0.0374	0.0785	0.0015	2.8928 (10^{-4})	4.2720 (10^{-6})
E_3	2.5	0.4695	−1.0694	−0.9412	0.1154	−0.2706	−0.1371	−0.0644	−0.0064	−0.0012	−1.7835 (10^{-5})
E_4	3.3	−1.5039	−0.7361	−1.4164	−0.5398	−1.0449	−0.5792	−0.2778	0.0099	0.0018	2.7523 (10^{-5})
E_5	4.2	−6.0083	2.4410	−0.9201	−0.7443	−0.6726	0.1831	0.2895	−0.0038	−7.0469 (10^{-4})	−1.0625 (10^{-5})
E_6	5	−18.050	15.0960	2.2149	−0.6972	−1.1066	−0.8653	−0.0465	0.0047	8.6666 (10^{-4})	1.2927 (10^{-5})

2.5.4. Ten-dimensional (sample) error iteration reduction

As per the six dimensional case above, if we use error iteration reduction: $E_j(k) \to \delta E_j(k)$, and $E_j(k+1) = E_j(k) + \delta E_j(k)$, all ten errors will progressively tend to zero even though there is a minor amount of oscillation. Here $E_j(k) \to \delta E_j(k)$ represents a solution to uncoupling Eq. (10). For the sake of brevity, we do not present the iteration details but show only the results for the weight optimization in Section 3.2.

3. The neurocomputing course

3.1. Two computing flows

Now we consider the computing course. Figure 3 shows the computing flow of the so-called error iteration reduction. Firstly, any arbitral W_i will be imputed. The subscript t, such as $1, 2, \ldots$, at W_{it} indicates t-th computing iteration for W_i. W_{i1} represents the initial weights for the first computing iteration, x_j represents the hidden layer, composed of ten points (nodes) on abscissa x, y_{j1} is the output from nodes j in the first neurocomputing iteration. The deviations of y_{j1} to samples y_j^*, i.e. errors E_{j1}, are equal to $y_{j1} - y_j^*$. The symbol $E_j \to \delta E_j$ represents a solution to uncoupling Eq. (12). After the first computing iteration, the hidden later is deleted because each δE_j can be directly computed from different E_j according to Eq. (12).

Figure 3 shows another computing flow of so-called weight iteration optimization: $w_i(k+1) = w_i(k) + \delta w_i(k)$. These two computing flows are coupled to each other by Eq. (11) of δw_i, which are associated with all errors E_j. The iterations are repeated many times until each $E_j < \varepsilon$ (for example, 0.0001). At the end, they all gradually tend to zero with each w_i approaching to an optimum. The object is to obtain the optimum weights without any deviations E_j to $Y^* = X^T W'$. The compilation of a computer program to implement the above is simple because the computing formula $\delta w_i(k)$ is its universality, being appropriate to any quantities of orders without a need to choose learning rates empirically[3, 4].

3.2. Selecting the initial weights before iterations

Consideration ought to be given to selecting the initial weights in advance of commencing the iterations.

Convergence will take a long time if the initial weights are selected too arbitrarily (as can be appreciated from Tables 2 and 3). In particular, δw_5 may be very small ($X_m^5 = 10^5$ is very significant in the denominator) in each iteration step of weight regulations. Guidance for selecting the initial weights for the iterations is given below:

The van der Pauw's function can be expressed as the following polynomial:

$$f = 1 + w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + w_5 x^5 + \zeta. \quad (15)$$

Setting $w_2 = 0.1\alpha w_1$, $w_3 = 0.1\beta w_2$, $w_4 = 0.1\gamma w_3$, $w_5 = 0.1\delta w_4$ and introducing them into Eq. (15), gives $f = 1 + w_1 x + 0.1\alpha w_1 x^2 + 0.1^2 \alpha\beta w_1 x^3 + 0.1^3 \alpha\beta\gamma w_1 x^4 + 0.1^4 \alpha\beta\gamma\delta w_1 x^5$.

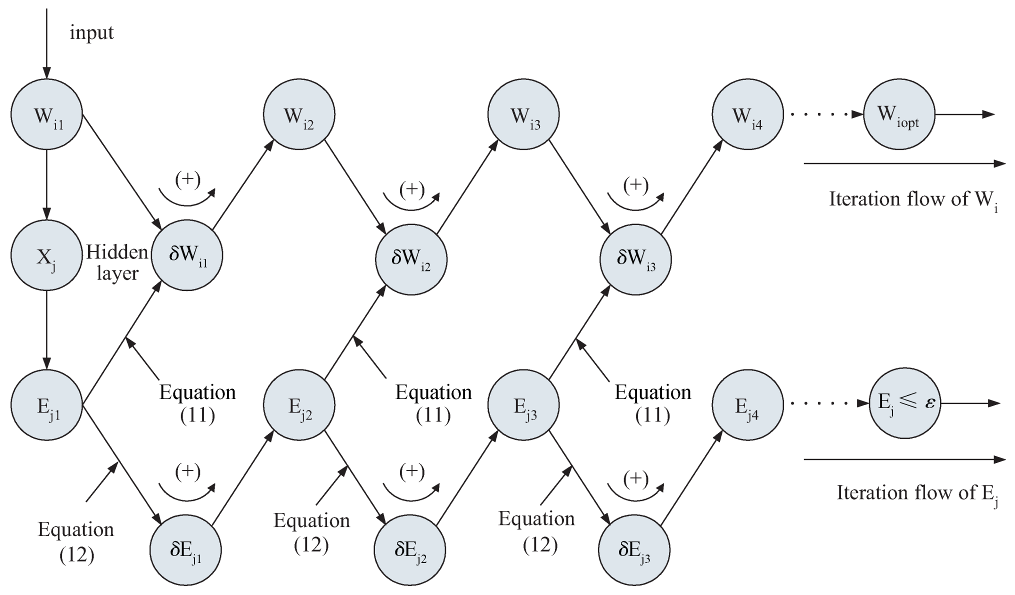

Fig. 3. Two computing flows of error iteration reduction and weight optimization.

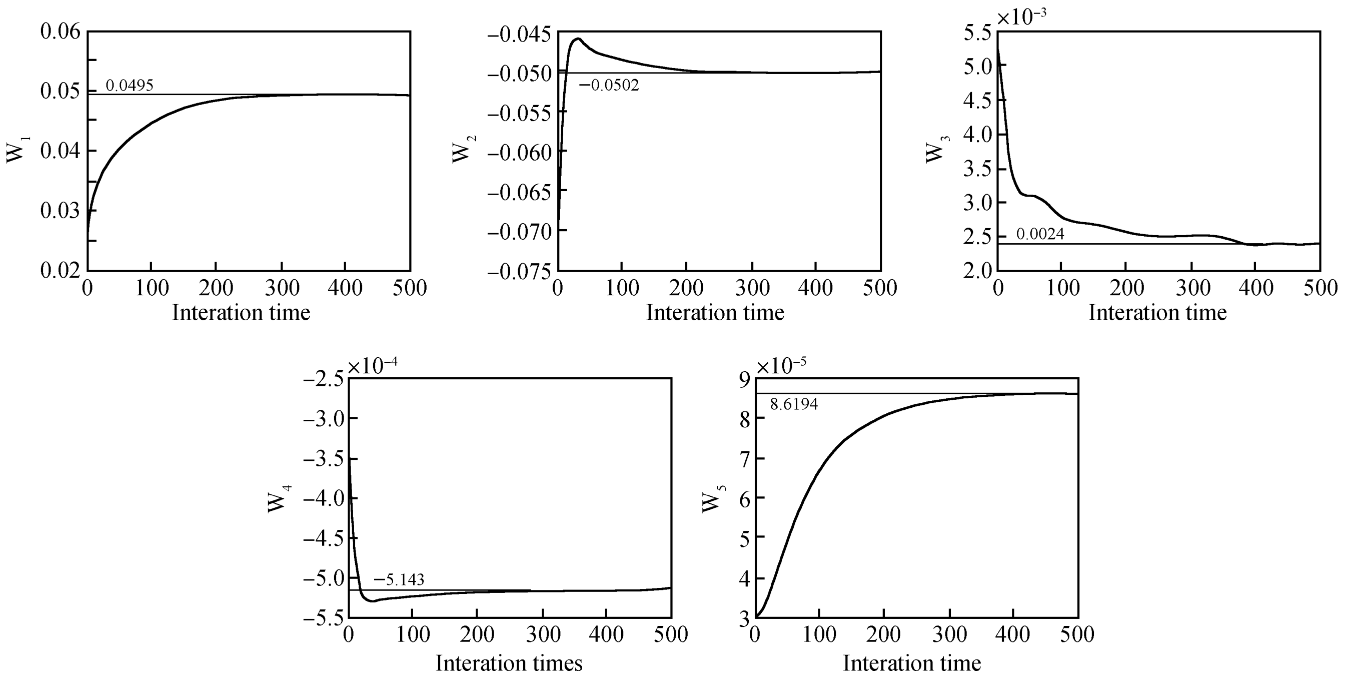

Fig. 4. Each weight varies with the iteration times to arrive at its optimum.

When $x = 1$, $f_1 = 1 + w_1(1 + 0.1\alpha + 0.1^2\alpha\beta + 0.1^3\alpha\beta\gamma + 0.1^4\alpha\beta\gamma\delta)$. Therefore, $\alpha = 10(\frac{f_1-1}{w_1} - 1)/\{1+0.1\beta\,[1+0.1\gamma(1+0.1\delta)]\} \approx -11$, where $f_1 = 1$ and setting $\beta = \gamma = \delta = -1$ as approximate values.

When $x = 2$, we have had $f_2 \cong 0.9604$, which is obtained by local reversal development (see Table 1) and $f_2 = 1 + 2w_1\{(1-(1.1)2 + (0.11)4 - (0.011)8 + (0.0011)16\} = w_1(-1.66)$. Therefore, $w_1 = (f_2 - 1)/(-1.66) = 0.0384$.

Due to $\beta = \gamma = \delta = -1$ being approximate values, five approximate weights can be obtained as the beginning values to initiate the neurocomputing:

$$w_1 = 0.0384,\quad w_2 = -0.04224,\quad w_3 = 0.004224,$$
$$w_4 = -0.0004224,\quad w_5 = 0.00004224. \tag{16}$$

Figure 4 illustrates the variation of each weight with the number of iteration to arrive its optimum by taking Eq. (11) to adjust the weight vector, relying on the error iteration reduction. It can be seen that the iteration times are reduced because the initial weights are properly selected.

Thus, van der Pauw's function can be expressed as the following polynomial:

$$\begin{aligned} f &= 1 + 0.0323772171x - 0.0403767815x^2 \\ &\quad + 0.0085788146x^3 - 0.0007769267x^4 \\ &\quad + 0.0000260362x^5, \\ &\qquad x = v_1/v_2. \end{aligned} \tag{17}$$

3.3. Comparison between different methods

The values of van der Pauw's function at $v_1/v_2 = 1, 2, \cdots, 10$ for Eq. (17) are close to the sample values shown in Table 1. The polynomial expression for van der Pauw's function can also been obtained[5] using a normalized method of polynomial matching for the non-linear signal[6]:

$$f = 1 + 0.03237715x - 0.04037679x^2 + 0.00857882x^3 - 0.00077693x^4 + 0.00002604x^5,$$

$$x = v_1/v_2. \qquad (18)$$

Equations (17) and (18) are substantively the same. However, the neurocomputing is more accurate because there are only five samples for the latter[5].

4. Conclusion

It is feasible to take:

$$\delta W_i = -\frac{1}{m}\frac{1}{x_m^i}\left\{E_i + \frac{1}{m}\left[(1-\delta_{k,j})\sum_{k=6}^{m}|E_k|\right] + \delta_{kj}E_k\right\}$$

to adjust a weight vector. Relying on error iteration reduction, coupled by δw_i, each weight will converge to an optimum value with a small magnitude of oscillation from a suitable starting value.

References

[1] Sun Y, Shi J, Meng Q. Measurement of sheet resistance of cross microareas using a modified Van der Pauw method. Semicond Sci Technol, 1996, 11: 805

[2] Shi J, Sun Y. New method of calculating the correction factor for the measurement of sheet resistance of a square sample with a square four point probe. Rev Sci Instrum, 1997, 68: 1814

[3] Robert H N. Neurocomputing. New York: Addison–Wesley Publishing Company, 1990

[4] Mirszal A R. Artificial intelligence. London: Chapman and Hall, 1990

[5] Wang J, Sun Y. Realization of Van der Pauw function's reversal development with high accuracy using a normalized method of polynomial match. Chinese Journal of Semiconductors, 2003, 24(8): 817

[6] Sun Y, Meng Q, Chen Z, et al. Normalizing the polynomial-match for the non-linear signal in transducers. Sensors and Actuators A, 2005, 125: 76