Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE @ DIRECT®

ELSEVIER International Journal of Approximate Reasoning 35 (2004) 137–161

INTERNATIONAL JOURNAL OF APPROXIMATE REASONING

www.elsevier.com/locate/ijar

A diffusion-neural-network for learning from small samples ☆

Chongfu Huang [a,*], Claudio Moraga [b]

[a] *Institute of Resources Science, Beijing Normal University, 19, Xinjiekouwai Street, Beijing 100875, China*

[b] *Department of Computer Science, Computer Engineering and Computing Education, University of Dortmund, 44221 Dortmund, Germany*

Received 1 June 2003; accepted 1 June 2003

Abstract

Neural information processing models largely assume that the patterns for training a neural network are sufficient. Otherwise, there must exist a non-negligible error between the real function and the estimated function from a trained network. To reduce the error, in this paper, we suggest a diffusion-neural-network (DNN) to learn from a small sample consisting of only a few patterns. A DNN with more nodes in the input and layers is trained by using the deriving patterns instead of original patterns. In this paper, we give an example to show how to construct a DNN for recognizing a non-linear function. In our case, the DNN's error is less than the error of the conventional BP network, about 48%. To substantiate the special case arguments, we also study other two non-linear functions with simulation technology. The results show that the DNN model is very effective in the case where the target function has a strong non-linearity or a given sample is very small.
© 2003 Elsevier Inc. All rights reserved.

Keywords: Neural network; Non-linear function; Fuzzy information; Information diffusion

☆ Project supported by a Mercator Visiting Professorship of the German Research Society DFG, granted to Prof. Chongfu Huang at the University of Dortmund.

* Corresponding author. Fax: 86-10-62208178.

E-mail addresses: nortzw@irs.bnu.edu.cn, hchongfu@bnu.edu.cn (C. Huang).

0888-613X/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.ijar.2003.06.001

1. Introduction

Artificial neural networks have received extensive attention during the last two decades. It is well known that they can solve many practical problems as pattern recognition [21], function approximation [24], system identification [17], time series forecasting, etc. [5,20].

Neuro-fuzzy modeling is concerned with the extraction of models from numerical data representing the behaviour of a system. The models in this case are rule-based and use the formalism of fuzzy logic, i.e. they consists of sets of fuzzy "if-then" rules with possibly several premises [18].

Neural information processing models largely assume that: (i) the patterns are compatible; (ii) the learning patterns for training a neural network are sufficient.

If the patterns are contradictory, the neural network does not converge because the adjustments of weights and thresholds do not know where to turn. In 1996, Huang and Ruan [13] used the information diffusion method [7,8] and the falling shadow theory [25] to construct an information diffusion network (IDN) based on BP algorithm to solve the problem of contradictory patterns. An IDN always converges. For every result of IDN method, its reliability can be analysed conveniently. In 1999, Huang and Leung [12] suggested a hybrid fuzzy-neural-network to estimate the relationship between isoseismal area and earthquake magnitude. In the model, the information diffusion method is employed to construct fuzzy "if-then" rules as many as the given observations. Integrating the rules to form an information-diffusion-approximate-reasoning estimator (IDAR), we can change contradictory patterns to be compatible for training a BP network. The hybrid-model estimator is more precise than the linear-regression estimator, and more stable than the conventional BP-neural-network estimator. These two hybrid models put forward the case that contradictory patterns can be learned by neural networks. In other words, with the help of the information diffusion technique [10,14], we have resolved the problem related to contradictory patterns.

If the learning patterns are insufficient, it is impossible to recognize a non-linear system, i.e, there must exist a non-negligible error between the real function and the estimated function from a trained network. Developing the information diffusion technique, in this paper, we suggest another hybrid model to reduce the error of estimated function from a BP network trained by a small sample.

2. Conventional BP networks trained by small samples

A neural network can be understood [16] as a mapping $f : R^p \to R^q$, defined by

$$y = f(x) = \varphi(Wx),$$

where $x \in R^p$ ($R^p = \mathbb{R}^p$, $\mathbb{R}$ is the set of real numbers) is the input vector, $y \in R^q$ is the output vector. The weight matrix W is a $p \times q$ matrix and φ is a non-linear function that is often referred to as the activation function. The typical activation function is the Sigmoid function

$$\varphi(x) = \frac{1}{1 + e^{-\alpha x}}, \quad \alpha > 0.$$

The mapping f can be decomposed into a chaining of mappings; the result is a multi-layer network

$$R^p \to R^m \to \cdots \to R^n \to R^q.$$

The algorithm for computing W is often called the training algorithm. The most popular neural networks are the multi-layer back-propagation networks whose training algorithm is the well-known gradient descent method. Such networks are called BP networks.

A conventional BP network (CBPN) consists of an input layer (the first layer), an output layer (the last layer), and some hidden layers. To recognize a function with p input variables and q output variable, in general, we set p nodes in the input layer and q nodes in the output layer. In other words, the number of nodes in the first and last layer is just equal to the number of input and output variables, respectively.

Relationships between variables are most often recognized by learning neural networks with data or patterns collected. The approach is also called adaptive pattern recognition [19]. For the majority of cases, the applied neural networks, from a statistical point of view, solve conditional estimation problems [15]. The celebrated back propagation error algorithm used for training feed forward neural networks is shown to be a special case of gradient optimization in the sense of mean squared error [22]. Feed forward neural networks are analyzed in [27] for consistent estimation of conditional expectation functions, which optimize the expected squared error.

In the learning phase of training such a network, we present the pattern $\underline{x}_p = \{i_{pi}\}$ as input and ask that the network adjust the set of weights in all the connecting links and also all the thresholds in the nodes such that the desired outputs $\underline{y}_p = \{t_{pk}\}$ are obtained at the output nodes. Once this adjustment is made, we present another pair of $\underline{x}_p = \{i_{pi}\}$ and $\underline{y}_p = \{t_{pk}\}$, and ask that the network learn that association also. In fact, we ask that the network find a single set of weights and biases that will satisfy all the (input, output) pairs presented to do it. This process can pose a very strenuous learning task and is not always readily accomplished.

In general, the outputs $\{o_{pk}\}$ of the network will not be the same as the target or desired values $\{t_{pk}\}$. For each pattern, the square of the error is

$$E_p = \frac{1}{2}\sum_k (t_{pk} - o_{pk})^2$$

and the average system error is

$$E = \frac{1}{2P}\sum_p \sum_k (t_{pk} - o_{pk})^2, \tag{1}$$

where P is the sample size and the one-half scaling factor is inserted for mathematical convenience. A true gradient search for minimum system error should be based on the minimization of expression (1).

A number of authors have discussed the property of universal approximation with respect to neural networks. For example, in 1989 Cybenko [2] and Funahashi [4] showed that any continuous function can be approximated by a neural network with one internal hidden layer using sigmoidal non-linearities. Also in 1989 Hornik et al. [6] proved that multi-layer networks using arbitrary Squashing functions can approximate any continuous function to any degree of accuracy, provided enough hidden units are available. However, in 1995, Wray and Green [28] proved that, due to the fact that networks are implemented on computers, the property of universal approximation (to any degree of accuracy) does not hold in practice.

Their results come from an assume that we can get patterns as many as we need to train networks. Otherwise, what will happen? Let us consider a simple non-linear function

$$y = x^2, \quad x \in [0, 1]. \tag{2}$$

Our task is to learn the function with the following sample:

$$\begin{aligned} A &= \{(\underline{x}_p, \underline{y}_p) \,|\, p = 1, 2, 3, 4, 5\} \\ &= \{(0,0), (1/4, 1/16), (1/2, 1/4), (3/4, 9/16), (1, 1)\}. \end{aligned} \tag{3}$$

A conventional BP network (Fig. 1) with one input node, and one output node can be used to learn from this small sample. The number of input and output nodes correspond to the number of input and output dimensions, respectively.

It is acknowledged that is difficult to determine the number of hidden nodes. That is why numerous algorithms can be found [1,3,11,26]. Many have believed that a network with the minimum number of hidden nodes is the best generalizing network. This is not true. It is quite easy to show by example that sometimes a network with more than the minimum number of hidden nodes generalizes better. Our interest of this paper is not this topic. The approach we use to find the number of hidden nodes is to start with a few numbers of nodes,

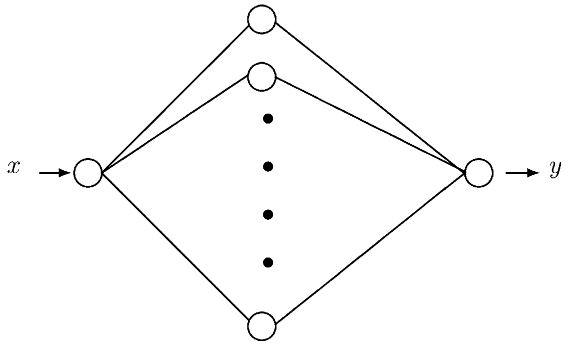

Fig. 1. The architecture of a conventional BP network to learn with the sample in (3). It is a topology 1–K–1 BP network.

then slightly increase the number, until no significant improvement is noted. In our case, the result of this approach reveals: the number of hidden nodes is 15.

Let the momentum rate be $\eta = 0.9$ and the learning rate be $\alpha = 0.7$. After 6 000 000 iterations, the normalized system error is 0.0000000773. Fig. 2 shows the real function $y = x^2$ in the thick curve and the estimated function of trained BP network in the thin curve. We could get the result that the estimated one is not close to the real one for many values of x in $[0, 1]$.

We may increase the sample size, which must always be of benefit, but it may imply more expense if data collection is costly, as when, for example, dangerous real measurements or complex technical experiments have to be performed. In this situation we would like a new viewpoint which would help us to improve estimated functions of trained BP networks on a small sample.

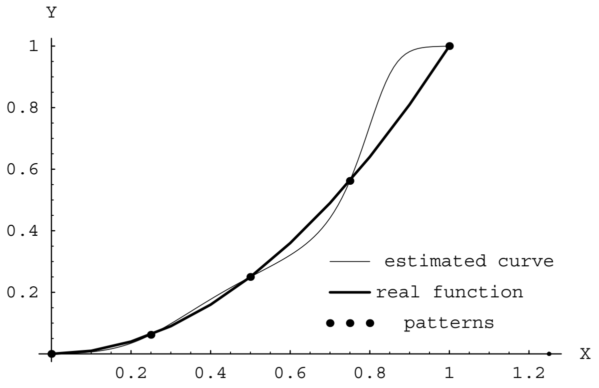

Fig. 2. Comparison between the real function (thick curve) and the estimated function (thin curve) from a BP network with topology 1–15–1.

3. Small-sample problem and information diffusion techniques

From a small sample, any classical network cannot recognize a non-linear function. Is there any techniques to ease the problem? The answer is positive. The principle of information diffusion [14] asserts that, when we use an incomplete-data set to estimate a relationship, there must exist reasonable diffusion means to change observations into fuzzy sets to partly fill the gap caused by incompleteness and improve non-diffusion estimate.

The information diffusion techniques were suggested to deal with so-called small-sample problem by fuzzy set theory.

Let $X = \{x_1, x_2, \dots, x_n\}$ be a given sample, called a data set, drawn from a population Ω. We assume that X will be employed to estimate a relation R in Ω.

Definition 1. Give a sample X. Let R be the real relation we want to know. X is called a correct-data set to R if and only if there exists a model γ in which we can obtain an estimate R_X^γ such that $R_X^\gamma = R$.

Let the size of X be n. Let U be the universe of the relation R described by Ω. For example, the universe of the relation $y = 1.5x$ is $\mathbb{R} \times \mathbb{R}$, i.e., $U = R^2$.

The set of all random samples with size n drawn from Ω is called the *n-sample space* of Ω, denoted by $\mathscr{X}_n$. The set of all models by which we can estimate R with a given sample is called the *operator space*, denoted by Γ. $\forall X \in \mathscr{X}_n, \forall \gamma \in \Gamma$, we use $r_X^\gamma(u)$ to denote the estimate of R at a point $u \in U$ with X by γ.

Definition 2. Let $X \in \mathscr{X}_n$. X is called a *handicapped sample* if and only if $\forall \gamma \Gamma$, $\exists u \in U$, such that

$$|r_X^\gamma(u) - r(u)| > 0. \tag{4}$$

Definition 3. $\mathscr{X}_n$ is called an *incomplete sample space* of Ω if and only if $\exists X \in \mathscr{X}_n$ and X is handicapped.

Definition 4. $X \in \mathscr{X}_n$ is called an *incomplete-data set* if and only if $\mathscr{X}_n$ is incomplete.

An incomplete-data set X is called an *incomplete sample*. In a situation with an incomplete sample, we say that we face a *small-sample problem*.

Definition 5. Let X be a sample and V be a subset of U. A mapping from $X \times V$ to $[0,1]$

$$\begin{aligned} &\mu : X \times V \to [0,1], \\ &(x,v) \mapsto \mu(x,v) \quad \forall (x,v) \in X \times V \end{aligned} \tag{5}$$

is called an *information diffusion* of X on V if it is decreasing: $\forall x \in X$, $\forall v', v'' \in V$, if $\|v' - x\| \leqslant \|v'' - x\|$ then $\mu(x, v') \geqslant \mu(x, v'')$. μ is called a *diffusion function* and V is called a *monitoring space*.

Definition 6. The *trivial diffusion* is defined by

$$\mu(x, u) = \begin{cases} 1, & \text{if } u = x, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Definition 7. $\mathscr{D}(X) = \{\mu(x, u) \,|\, x \in X, u \in U\}$ is called the *sample of fuzzy sets* (FS) derived from X on U by information diffusion.

Definition 8. Let X be a given sample which can be used to estimate a relationship R by the operator γ. If the estimate is calculated by using the FS $\mathscr{D}(X)$, the estimate is called the *diffusion estimate* of R, denoted by

$$\widetilde{R}(\gamma, \mathscr{D}(X)) = \{\gamma(\mu(x_i, u)) \,|\, x_i \in X, u \in U\}, \tag{7}$$

where $\mu(x_i, u)$ is a diffusion function of X on U.

Correspondingly, the estimated relationship $\widehat{R}$ that directly comes from a given sample X by an operator γ is denoted as $\widehat{R}(\gamma, X)$. It is called the *non-diffusion estimate* of R.

Corollary 1. *A trivial diffusion estimate is a non-diffusion estimate.*

One easily verifies any kernel function $K(x)$ [23] as a diffusion function is sufficient and conservative.

The principle of information diffusion. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a given sample which can be used to estimate a relationship R on universe U. If and only if X is incomplete, there must exist a diffusion function $\mu(x_i, u)$ and a corresponding operator γ that leads to a diffusion estimate $\widetilde{R}(\gamma, \mathscr{D}(X))$ such that it is nearer to the real R than any non-diffusion estimate.

The principle of information diffusion guarantees the existence of reasonable diffusion functions to improve the non-diffusion estimates when the given samples are incomplete. In other words, when X is incomplete, there must exist some approach to pick up fuzzy information of X for more precisely estimating a relationship as function approximation. However, the principle does not provide any indication on how to find the diffusion functions.

Although the principle is given as an assertion, it holds, at least, in the case of estimating a probability density function (pdf), as proven in [7].

After researching the similarities of information and molecules in diffusion action, we can obtain a partial differential equation [7] to represent the information diffusion. Solving the equation, we obtain a diffusion function

$$\mu(x,u) = \frac{1}{h\sqrt{2\pi}} \exp\left[-\frac{(x-u)^2}{2h^2}\right] \tag{8}$$

named *normal diffusion*, because the function is just the same as the normal pdf.

According to computer simulation test results, we suggest the following formula to calculate h

$$h = \begin{cases} 0.6841(b-a) & \text{for } n = 5, \\ 0.5404(b-a) & \text{for } n = 6, \\ 0.4482(b-a) & \text{for } n = 7, \\ 0.3839(b-a) & \text{for } n = 8, \\ 2.6851(b-a)/(n-1) & \text{for } n \geqslant 9, \end{cases} \tag{9}$$

where

$$b = \max_{1 \leqslant i \leqslant n}\{x_i\}, \quad a = \min_{1 \leqslant i \leqslant n}\{x_i\}.$$

h is called *diffusion coefficient*.

4. Deriving patterns by information diffusion

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a given sample with input a and output b, i.e., $x_i = (a_i, b_i)$, that will be used to train a BP network.

That data X is incomplete implies that the patterns are insufficient. In other words, we need more patterns to train the BP network for obtaining a more accurate estimate of input–output relation. The new patterns obtained from this X are called *derivative patterns from X*.

According to the principle of information diffusion we know that, there must exist a diffusion function μ and a corresponding operator that lead to a diffusion estimate nearer the real input–output relation.

The simplest model based on this principle to derive patterns is suggested in [9] where an observation $x \in \mathbb{R}$ is used to derive 10–50 points through a pseudo-random generator controlled by pdf

$$p(u) = \frac{1}{h\sqrt{2\pi}} \exp\left[-\frac{(x-u)^2}{2h^2}\right], \quad u \in \mathbb{R}. \tag{10}$$

By this way, we can reduce the error of estimate of pdf from which X was drawn.

Developing the above model to be a 2-dimensional model for deriving patterns from $x_i = (a_i, b_i)$ is good in theory, but does it work in practice? In the case of training a BP network, the experiment failed. The reason is that the deriving patterns may be contradictory. The neural network does not converge. The more the derivative patterns, the more difficult the training.

The failure of the conventional BP network for a small sample tells us

(1) We need a new model for deriving patterns to improve BP estimate.
(2) We need a new topology BP network to replace the 1–K–1 network for learning derivative patterns.

Let us consider what we can do with the normal diffusion function shown in (8). This $\mu(x,u)$ is a membership function of a fuzzy set that can be written as

$$\mu_x(u) = \mu(x,u), \quad x \in X, \ u \in \mathbb{R}. \tag{11}$$

According to the relation between the membership and possibility [29], $\mu_x(u_0)$ can be translated to be the possibility that $u = u_0$. Thus, we get a possibility distribution $\pi'(u)$ on $\mathbb{R}$ associated with variable u from x

$$\pi'_x(u) = \mu_x(u), \quad u \in \mathbb{R}. \tag{12}$$

The normal diffusion derives the following possibility distribution

$$\pi'_x(u) = \frac{1}{h\sqrt{2\pi}} \exp\left[-\frac{(x-u)^2}{2h^2}\right], \quad u \in \mathbb{R}. \tag{13}$$

The nearer the u is to x, the larger the possibility to be equal to x. The largest possibility is $1/(h\sqrt{2\pi})$. The corresponding normalized distribution is

$$\pi_x(u) = \exp\left[-\frac{(x-u)^2}{2h^2}\right], \quad u \in \mathbb{R}, \tag{14}$$

where the largest possibility is 1.

Recall that our derivation problem is to design a model using the n labeled patterns $(a_i, f(a_i))$, $i = 1, 2, \ldots, n$ (n is small as $n = 5$) to estimate mapping f as precisely as possible. In general, we use (x_i, y_i) to denote $(a_i, f(a_i))$. We want to find $y = f(x)$ by a BP network on the labeled patterns (x_i, y_i).

If the linear correlation coefficient r that measures the strength of the relationship between the paired x and y values in a sample is 1, the function f must be a line. In other words, for a given sample X with $r = 1$, it is unnecessary to derive any new pattern. It seems fairly obvious that r is an important information for deriving patterns. It can be calculated by

$$r_{xy} = \frac{l_{xy}}{\sqrt{l_{xx} l_{yy}}}, \tag{15}$$

where

$$l_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2, \quad l_{yy} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

and

$$l_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}),$$

where

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i.$$

For example, using formula (15) on the patterns in (3), we obtain

$$r = \frac{0.625}{\sqrt{0.625 \times 0.6797}} = 0.9589 \approx 0.96.$$

Naturally we let

$$\exp\left[-\frac{(x-u)^2}{2h^2}\right] = 0.96$$

to find u from x as a derivative point. However, the neural network still does not converge.

In fact, we never consider the normal diffusion with the coefficient calculated by formula (9) can fit any case. For our case that a diffusion function is employed to derive patterns to improve training BP networks, the normal diffusion and its coefficient may not be the best choice. The derivative patterns may be more contradictory so that they produce convergence problems. However we know that, the nearer the derived patterns are to the given pattern, the smaller the contradiction. A linear correlation coefficient r also provides some information to assign the distance from a given pattern to a derived pattern. When $r = 1$ we assign value 0 to the distance between a given point x and its derivative point u. When $r = 0.96$ we assign 0.999999 to the distance. In this way, we can avoid convergence problems.

In general, if the linear correlation coefficient of a sample

$$X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

is $r = 0.9 + m \times 10^{-2}$, $m = 1, 2, \ldots, 9$, we can use

$$\exp\left[-\frac{(x_i - x)^2}{2h_x^2}\right] = \underbrace{0.9\ldots9}_{\text{there are } m \text{ 9s}} \tag{16}$$

and

$$\exp\left[-\frac{(y_i - y)^2}{2h_y^2}\right] = \underbrace{0.9\ldots9}_{\text{there are } m \text{ 9s}}, \tag{17}$$

where both h_x and h_y can be obtained by using formula (9), to derive new pattern (x, y).

We define a mapping ψ to represent this change

$$\begin{aligned} &\psi : r \to poss \text{ (possibility)}, \\ &\psi(1) \mapsto 1, \\ &\psi(0.9 + m \times 10^{-2}) \mapsto \underbrace{0.9\ldots9}_{m\ 9\text{s}}\ \forall r \in \{0.91, 0.92, \ldots, 0.99\}. \end{aligned} \tag{18}$$

Let

$$\exp\left[-\frac{(x-u)^2}{2h^2}\right] = \psi(r). \tag{19}$$

We have

$$u = x \pm \sqrt{-2h^2 \ln \psi(r)}. \tag{20}$$

Hence, from x_i we can derive three points

$$\begin{aligned} &x_i' = x_i - \sqrt{-2h_x^2 \ln \psi(r)} \text{ with } poss = \psi(r), \\ &x_i \text{ with } poss = 1, \\ &x_i'' = x_i + \sqrt{-2h_x^2 \ln \psi(r)} \text{ with } poss = \psi(r). \end{aligned} \tag{21}$$

Also from y_i we have

$$\begin{aligned} &y_i' = y_i - \sqrt{-2h_y^2 \ln \psi(r)} \text{ with } poss = \psi(r), \\ &y_i \text{ with } poss = 1, \\ &y_i'' = y_i + \sqrt{-2h_y^2 \ln \psi(r)} \text{ with } poss = \psi(r). \end{aligned} \tag{22}$$

On the assumption that $y = f(x)$ is increasing, from one pattern (x_i, y_i), we obtain three patterns

$$\begin{aligned} &(x_i', y_i') \text{ with } poss = \psi(r), \\ &(x_i, y_i) \text{ with } poss = 1, \\ &(x_i'', y_i'') \text{ with } poss = \psi(r). \end{aligned} \tag{23}$$

5. Diffusion-neural-network

Using the above deriving model, from a given sample

$$X = \{(x_i, y_i) \mid i = 1, 2, \ldots, n; x_i, y_i \in \mathbb{R}\}, \tag{24}$$

where each pattern has one input value and one output value, on the assumption that $y = f(x)$ is increasing, we obtain $3n$ derivative patterns so that we have a new sample

$$\begin{aligned}X' = \{&((x'_1, \psi(r)), (y'_1, \psi(r))), ((x_1, 1), (y_1, 1)), ((x''_1, \psi(r)), (y''_1, \psi(r))), \ldots,\\ &((x'_n, \psi(r)), (y'_n, \psi(r))), ((x_n, 1), (y_n, 1)), ((x''_n, \psi(r)), (y''_n, \psi(r)))\},\end{aligned}$$

where r is calculated by formula (15), ψ is defined by (18), and x'_i, y'_i, x''_i, y''_i, $i = 1, 2, \ldots, n$, are calculated by using (21) and (22). In this new sample each pattern has two input values and two output values. This sample also can be written as

$$X' = \{((x_{dj}, poss_j), (y_{dj}, poss_j)) \,|\, j = 1, 2, \ldots, 3n\}, \tag{25}$$

where

$$x_{dj} = \begin{cases} x'_i, & \text{if } j = 3i-2, \\ x_i, & \text{if } j = 3i-1, \quad i = 1, 2, \ldots n, \\ x''_i & \text{if } j = 3i, \end{cases}$$

$$y_{dj} = \begin{cases} y'_i, & \text{if } j = 3i-2, \\ y_i, & \text{if } j = 3i-1, \quad i = 1, 2, \ldots n \\ y''_i & \text{if } j = 3i, \end{cases}$$

and

$$poss_j = \begin{cases} \psi(r), & \text{if } j = 3i-2 \text{ or } j = 3i, \\ 1, & \text{if } j = 3i-1, \end{cases} \quad i = 1, 2, \ldots n.$$

Above derivative sample can be used to train a conventional BP-neural-network with two nodes in the input layer, and two nodes in the output layer. That is a topology 2–K–2 BP network to replace 1–K–1 network for learning derivative patterns. Its architecture can be shown by Fig. 3, called *diffusion-neural-network* (DNN). In the input layer, one node is for the variable x, another is for the possibility *poss*. In output layer, one node is for the variable y, another is for the possibility *poss*.

We can regard x_0 as $(x_0, 1)$ to be an input of the trained DNN. If the output of the DNN is (y_0, z) and z is near 1, we say that the estimated value is y_0. In this way, we can get a new estimated function from a given sample X.

Let us define the benefit from the DNN model.

We suppose that the real function is $y = f(x)$. We use the notation $y = CBPN(x)$ for the input x producing output y calculated by a conventional BP network. We use $y = DNN(x)$ for the input x producing output y calculated by a DNN.

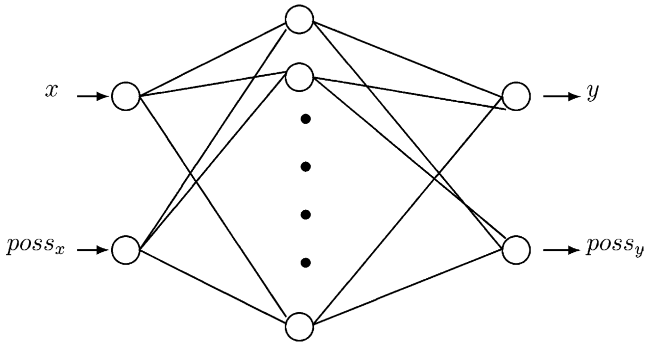

Fig. 3. The architecture of diffusion-neural-network (DNN) to learn with the derivative sample in (25). It is a topology 2–K–2 BP network.

To measure the error between the real function and an estimated function, we take some points in domain of the real function to make a set

$$U = \{u_1, u_2, \dots, u_m\}. \tag{26}$$

Let

$$s_{CBPN} = \frac{1}{m}\sum_{j=1}^{m}(f(u_j) - CBPN(u_j))^2 \tag{27}$$

and

$$s_{DNN} = \frac{1}{m}\sum_{j=1}^{m}(f(u_j) - DNN(u_j))^2, \tag{28}$$

called the average square errors.

Obviously, $s_{DNN} < s_{CBPN}$ means the DNN estimate is nearer the real function than the estimate from the conventional BP network. In this case, the benefit of DNN is defined by

$$B = \frac{s_{CBPN} - s_{DNN}}{s_{CBPN}} \times 100\%. \tag{29}$$

It means DNN can reduce the error of CBPN estimate in B.

6. An example

We will study

$$\begin{aligned} X &= \{(x_i, y_i) \mid i = 1,2,3,4,5\} \\ &= \{(0,0),(1/4,1/16),(1/2,1/4),(3/4,9/16),(1,1)\} \end{aligned}$$

to show the advantage of the DNN model. The given sample is the same as the A in (3).

First, we derive more patterns from the given sample.

Using formula (9), we have diffusion coefficients

$$h_x = 0.6841\left(\max_{1\leqslant i\leqslant n}\{x_i\} - \min_{1\leqslant i\leqslant n}\{x_i\}\right) = 0.6841,$$

$$h_y = 0.6841\left(\max_{1\leqslant i\leqslant n}\{y_i\} - \min_{1\leqslant i\leqslant n}\{y_i\}\right) = 0.6841.$$

Using formula (15), we obtain linear correlation coefficient $r = 0.96$.

According to the definition of $\psi(r)$ given by (18) and using formulas (21) and (22), from one pattern in X we can obtain tree derivative patterns. For example, from the second pattern $(x_2, y_2) = (1/4, 1/16)$, we obtain

$$x_2' = x_2 - \sqrt{-2h_x^2 \ln \psi(r)} = 1/4 - \sqrt{-2(0.6841)^2 \ln 0.999999} = 0.249026,$$

$$x_2'' = x_2 + \sqrt{-2h_x^2 \ln \psi(r)} = 1/4 + \sqrt{-2(0.6841)^2 \ln 0.999999} = 0.250974,$$

$$y_2' = y_2 - \sqrt{-2h_y^2 \ln \psi(r)} = 1/16 - \sqrt{-2(0.6841)^2 \ln 0.999999} = 0.061526,$$

$$y_2'' = y_2 + \sqrt{-2h_y^2 \ln \psi(r)} = 1/16 + \sqrt{-2(0.6841)^2 \ln 0.999999} = 0.063473.$$

Therefore, from pattern $(1/4, 1/16)$, we obtain three patterns

$(0.249026, 0.061526)$ with $poss = 0.999999$,

$(0.25, 0.0625)$ with $poss = 1$,

$(0.250974, 0.063473)$ with $poss = 0.999999$.

Hence, we obtain a derivative sample

$$
\begin{aligned}
&X' = \\
&\{((-0.000973, 0.999999), (-0.000973, 0.999999)), ((0, 1), (0, 1)), \\
&\quad ((0.000973, 0.999999), (0.000973, 0.999999)), \\
&\quad ((0.249026, 0.999999), (0.061526, 0.999999)), \\
&\quad ((0.25, 1), (0.0625, 1)), ((0.250974, 0.999999), (0.063473, 0.999999)), \\
&\quad ((0.499026, 0.999999), (0.249026, 0.999999)), ((0.5, 1), (0.25, 1)), \\
&\quad ((0.500974, 0.999999), (0.250974, 0.999999)), \\
&\quad ((0.749026, 0.999999), (0.561526, 0.999999)), \\
&\quad ((0.75, 1), (0.5625, 1)), ((0.750974, 0.999999), (0.563474, 0.999999)), \\
&\quad ((0.999026, 0.999999), (0.999026, 0.999999)), ((1, 1), (1, 1)), \\
&\quad ((1.000974, 0.999999), (1.000974, 0.999999))\}.
\end{aligned} \tag{30}
$$

Second, we use the derivative sample to train a DNN with two nodes in the input layer, one hidden layer with 15 nodes, and two nodes in the output layer. Let the momentum rate be $\eta = 0.9$ and the learning rate be $\alpha = 0.7$. After 6 000 000 iterations, the normalized system error is 0.0000004569. Fig. 4 shows the real function $y = x^2$ in solid curve and the estimated functions from a CBPN and a DNN. We could get the result that the estimated function from the DNN is closer to the real function than the one from the CBPN.

Finally, we study the benefit of DNN model for this calculation case.

We take some points with step $t = 0.01$ in domain $[0, 1]$ of the real function $y = x^2$ to make the following set:

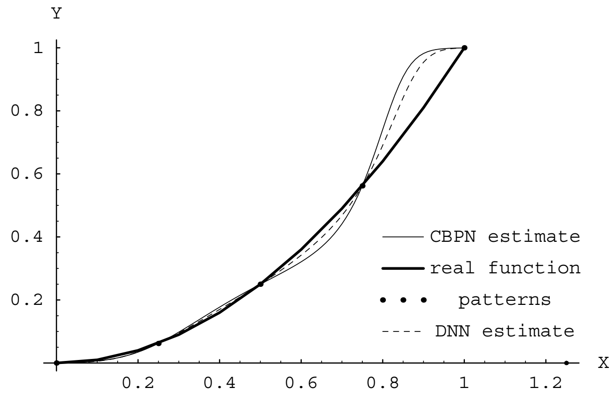

Fig. 4. Estimating $y = x^2$ by a conventional BP network (CBPN) and a diffusion-neural-network (DNN).

$$U = \{u_j \mid j = 1, 2, \ldots, 101\} = \{0, 0.01, \ldots, 1\}. \tag{31}$$

Using the real function, from U we obtain real function values whose form a set

$$V = \{v_j \mid v_j = u_j^2, u_j \in U\} = \{0, 0.0001, \ldots, 1\}.$$

Using the trained CBPN that gives the CBPN estimate curve of Fig. 4, we obtain corresponding estimated values

$$\begin{aligned} V_{\text{CBPN}} &= \{v_j \mid v_j = CBPN(u_j), u_j \in U\} \\ &= \{0.000747, 0.000940, \ldots, 0.999538\}. \end{aligned}$$

Using the trained DNN that gives the DNN estimate curve of Fig. 4, we obtain

$$V_{DNN} = \{v_j \mid v_j = DNN(u_j), u_j \in U\} = \{0.000353, 0.000483, \ldots, 0.999422\}.$$

Using formulas (27) and (28), we have

$$s_{CBPN} = 0.004528, \quad s_{DNN} = 0.002358.$$

Therefore

$$B = \frac{0.004528 - 0.002358}{0.004528} \times 100\% = 48\%.$$

It is interesting to note, if we choose the normalized system error the same as in the trained DNN (that is larger than one in the trained CBPN) to control training of a conventional BP network, we can get a smaller s_{CBPN}. After 302 792 iterations, it arrives to a system error of 0.0000004569. Now, $s_{CBPN} = 0.003496$. That is, a larger learning error leads to a smaller estimate error. In this case $B = 33\%$.

According to the theory of the artificial neural network, the smaller the system error, the higher accurate the estimated function. The reason why this result is in contradiction with the theory is that the given patterns are so few. However it cannot lead to say that the larger the system error, the smaller the estimate error. Furthermore, it is more difficult to find a fit system error to control training. Combining above benefits calculated by different controlling parameters, roughly speaking, we can reduce the estimate error about 40% with the DNN model.

Before closing this section we would like to address the issues of scalability and complexity. Under the scalability of a system, usually its sensitivity to an (increasing) number of data points is understood. In the present case, we would have to speak of "reverse scalability", since our concern is the performance of the proposed system under a decreasing number of data points for training. The proposed system is robust to sparse training data. As the number of training data points increases, the performance of the system approaches that of a classical feed-forward neural network trained by a gradient descend algorithm. Under the complexity of a system, the number of elementary computation steps relative to the size of the sample is understood. Since the proposed system first generates possible new training points and then trains the neural network, the computational complexity will be higher than that of a classical neural network using some form of backpropagation, however, since the proposed system is meant to work with a small number of training points, the real increase in computing time will not be of importance in the context of improved performance.

To substantiate the special case arguments, we need more numerical results to compare the CBPN model and DNN model. We employ simulation technology to fulfill it.

7. Simulation experiments with small samples

In this section, we use computer simulation to prove that DNN is better than CBPN when a given sample $X = \{(x_i, y_i) \mid i = 1, 2, \dots, n\}$ of a non-linear function $y = f(x)$ is small.

As well known, the Sigmoid function used in BP networks as the activation function has output value belonging to $[0, 1]$. To avoid any calculation normalizing X to be in $[0, l] \times [0, 1]$, we only consider the X such that $X \subset [0, l] \times [0, 1]$.

7.1. Model description

Let A be a set consisting of n generated pseudo-random observations x_i $(i = 1, 2, \dots, n)$ drawn from the uniform distribution $U(0, 1)$. A is called an input-sample. For a given function $y = f(x)$, we can have an output-sample $B = \{y_i \mid y_i = f(x_i), x_i \in A\}$. We use $X = \{(x_i, y_i) \mid i = 1, 2, \dots, n\}$ to recognize $y = f(x)$ by a CBPN and a DNN, respectively.

We consider the cases that $n = 5, 7, 9$. We employ a topology 1–15–1 BP network to be a CBPN, and a topology 2–15–2 BP network to be a DNN. To train these networks, we use $\eta = 0.9$ to be the momentum rate and $\alpha = 0.7$ to be the learning rate. For controlling the training, we set the number of iterations to be 600 000 and the normalized system error to be 0.00001. In other words, a neural network will be regarded as a trained network when it learns 600 000 times or its normalized system error is less or equal to 0.00001.

To reduce the error of the simulation experiments, we do 10 simulation experiments with different seed numbers (randomly given) for each size n. The average square error of estimates is a better index to show the quality of a model. Suppose we have done N simulations, and $s(i)$ is the result from the ith simulation, the average square error is then defined by

$$s = \frac{1}{N}\sum_{i=1}^{N} s(i). \tag{32}$$

There are many kinds of non-linear functions. However if a function has continuous derivatives up to $(n+1)$th order, then this function can be expanded in the following fashion:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \cdots + \frac{f^{(n)}(a)(x-a)^n}{n!} + R_n,$$

where R_n, called the remainder after $n+1$ terms, is given by

$$R_n = \int_a^x f^{(n+1)}(u)\frac{(x-u)^n}{n!}\,\mathrm{d}u = \frac{f^{(n+1)}(\xi)(x-a)^{n+1}}{(n+1)!}, \quad a < \xi < x.$$

When this expansion converges over a certain range c^x, that is, $\lim_{n\to\infty} R_n = 0$, then the expansion is called the *Taylor Series* of $f(x)$ expanded about a.

If $a = 0$ the series is called the *MacLaurin Series*

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \cdots$$

Therefore, strictly speaking, if we want to prove the performance of a model with respect to all continuous non-linear functions, we have to study its performance for every term. Practically speaking, because $\lim_{n\to\infty} R_n = 0$, the main part of a function consists of the first four terms. It is enough to study these terms. According to the definition of the mapping ψ for deriving patterns, we know that, for a linear function, DNN performs as CBPN. In last section, we showed that, DNN is better than CBPN to estimate $y = x^2$. Hence, the next task is to study $y = x^3$. Then, we can consider the case including x, x^2 and x^3. Without loss in generality, we will study

$$y = f_1(x) = 0.01x + 0.02x^2 + 0.9x^3, \quad x \in [0,1]. \tag{33}$$

Considering the importance of the exponential function for engineering, we also study

$$y = f_2(x) = 1 - \mathrm{e}^{-2x^4}, \quad x \in [0,1]. \tag{34}$$

In other words, if $s_{DNN} < s_{CBPN}$ holds for both functions, then we almost obtain the conclusion: DNN is better than CBPN.

We use Program 1 to generate uniform random numbers x_i, $i = 1, 2, \ldots, n$, based on seed number SEED.

Program 1 (*Generator of Random Numbers Obeying Uniform Distribution* $U(0,1)$).

```
      PROGRAM MAIN
      INTEGER N,SEED
      REAL X(9)
      READ(*,*)N,SEED
      IX = SEED
      DO 10 I = 1,N
      K1 = IX/60466
      IX = 35515*(IX-K1*60466)-K1*33657
      IF(IX.LT.0)IX = IX+2147483647
      K1 = IX/102657
      IX = 20919*(IX-K1*102657)-K1*1864
      IF(IX.LT.0)IX = IX+2147483647
      RANUN = FLOAT(IX)/2.147483647e9
      X(I) = RANUN
   10 CONTINUE
      WRITE(*,20)(I,X(I),I = 1,N)
   20 FORMAT(1X,5('X(',I1,') = ',F5.3,', '))
      STOP
      END
```

For example, input $N = 5$ (sample size) and SEED = 37589 (seed number) to Program 1, it will give $X(1) = 0.200$, $X(2) = 0.521$, $X(3) = 0.493$, $X(4) = 0.371$, $X(5) = 0.084$. Then, we obtain an input-sample

$$A = \{x_i \mid i = 1, 2, \ldots, 5\} = \{0.200, 0.521, 0.493, 0.371, 0.084\}.$$

By $y = 0.01x + 0.02x^2 + 0.9x^3$, we obtain an output-sample

$$B = \{y_i \mid i = 1, 2, \ldots, 5\} = \{0.010, 0.138, 0.118, 0.052, 0.001\}.$$

We use the set of patterns

$$\begin{aligned} X &= \{(x_i, y_i) \mid i = 1, 2, \ldots, 5\} \\ &= \{(0.200, 0.010), (0.521, 0.138), (0.493, 0.118), (0.371, 0.052), (0.084, 0.001)\} \end{aligned}$$

to recognize $y = f(x)$ by a CBPN and a DNN, respectively.

We also use the points of the U given in (31) as inputs to calculate outputs of function f. The square error s is calculated on these points.

7.2. Estimating $y = 0.01x + 0.02x^2 + 0.9x^3$

Tables 1–3 show the results of the simulation experiments with $N = 10$ (number of simulations) for sample size $n = 5, 7,$ and 9, respectively.

Table 1
Using five random patterns to estimate $y = 0.01x + 0.02x^2 + 0.9x^3$

Seed number	r	$\psi(r)$	s_{CBPN}	s_{DNN}
37 589	0.96	0.999999	0.005918892	0.003287310
471	0.95	0.99999	0.000587282	0.000363450
84 378	0.95	0.99999	0.000560380	0.000442065
4 455 556	0.94	0.9999	0.002898453	0.001890389
4	0.95	0.99999	0.004610664	0.003405603
14	0.96	0.999999	0.001247012	0.000741144
123	0.95	0.99999	0.000204686	0.000296314
41 356	0.95	0.99999	0.000254109	0.000199072
70	0.96	0.999999	0.002537641	0.001325982
90 089	0.96	0.999999	0.000120842	0.000030299
Average square error			0.001893996	0.001198163

Table 2
Using seven random patterns to estimate $y = 0.01x + 0.02x^2 + 0.9x^3$

Seed number	r	$\psi(r)$	s_{CBPN}	s_{DNN}
452	0.94	0.9999	0.000802205	0.000561230
41	0.95	0.99999	0.000588861	0.000513334
869 403	0.95	0.99999	0.001148254	0.001023722
30	0.95	0.99999	0.000276856	0.000176103
3067	0.95	0.99999	0.000161748	0.000168508
88	0.94	0.9999	0.002691227	0.001866422
4 763 200	0.94	0.9999	0.002037587	0.001560040
90 000	0.94	0.9999	0.000954938	0.000450177
777 758	0.94	0.9999	0.000284984	0.000248202
8977	0.94	0.9999	0.000111734	0.000027444
Average square error			0.000905840	0.000659518

Table 3
Using nine random patterns to estimate $y = 0.01x + 0.02x^2 + 0.9x^3$

Seed number	r	$\psi(r)$	s_{CBPN}	s_{DNN}
90	0.94	0.9999	0.003694762	0.003432359
666	0.96	0.999999	0.000100587	0.000093889
34 562	0.95	0.99999	0.000023885	0.000020703
789	0.94	0.9999	0.000051348	0.000018279
5555	0.96	0.999999	0.001914853	0.001346447
1132	0.94	0.9999	0.000432030	0.000444494
87 988	0.96	0.999999	0.000136089	0.000139782
2000	0.95	0.99999	0.000126611	0.000125036
3214	0.96	0.999999	0.000209642	0.000201720
5566	0.95	0.99999	0.000134331	0.000126790
Average square error			0.000682414	0.000594950

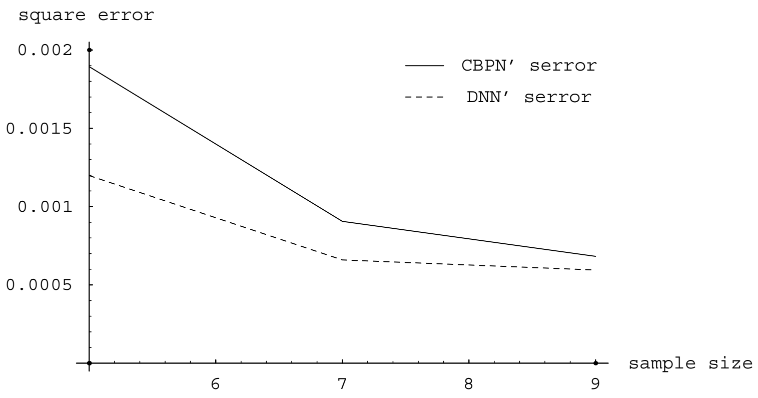

Fig. 5. Square errors of CBPN model and DNN model to estimate $y = 0.01x + 0.02x^2 + 0.9x^3$ with a small sample.

We use Fig. 5 to show the average square errors of CBPN and DNN. Obviously, DNN is better than CBPN. Furthermore, Fig. 5 suggests that the advantage of DNN will disappear as increasing the sample size n.

For $n = 5$, 7, 9, using formula (29), we obtain the benefit of DNN with respect to CBPN as the following:

$$B_5 = \frac{0.001893996 - 0.001198163}{0.001893996} \times 100\% = 37\%,$$

$$B_7 = \frac{0.000905840 - 0.000659519}{0.000905840} \times 100\% = 27\%,$$

$$B_9 = \frac{0.000682414 - 0.000594950}{0.000682414} \times 100\% = 13\%.$$

7.3. Estimating $y = 1 - e^{-2x^4}$

Tables 4–6 show the results of the simulation experiments with $N = 10$ (number of simulations) for sample size $n = 5, 7$, and 9, respectively.

Fig. 6 shows the average square errors of CBPN and DNN to estimate $y = 1 - e^{-2x^4}$ with a small sample, size is n. Here, DNN is also better than CBPN.

The corresponding benefits are

$$B_5 = \frac{0.005474657 - 0.004292307}{0.005474657} \times 100\% = 22\%,$$

$$B_7 = \frac{0.000111606 - 0.000091506}{0.000111606} \times 100\% = 18\%,$$

Table 4
Using five random patterns to estimate $y = 1 - e^{-2x^4}$

Seed number	r	$\psi(r)$	s_{CBPN}	s_{DNN}
112 233	0.95	0.99999	0.000085535	0.000063195
123	0.96	0.99999	0.000014915	0.000014481
10 000	0.96	0.999999	0.000020523	0.000011072
765	0.96	0.999999	0.000133007	0.000120072
45 298	0.96	0.999999	0.000209563	0.000168604
1000	0.96	0.999999	0.000027055	0.000018111
330	0.95	0.99999	0.000039449	0.000028573
56 300	0.94	0.9999	0.000073281	0.000056224
560	0.94	0.9999	0.000056001	0.000001335
89 000	0.96	0.999999	0.054087236	0.042441401
Average square error			0.005474657	0.004292307

Table 5
Using seven random patterns to estimate $y = 1 - e^{-2x^4}$

Seed number	r	$\psi(r)$	s_{CBPN}	s_{DNN}
990	0.96	0.999999	0.000074837	0.000044239
444 332	0.94	0.9999	0.000420641	0.000360700
43 000	0.96	0.999999	0.000063083	0.000038376
10 000	0.96	0.999999	0.000024781	0.000018266
33 333	0.96	0.999999	0.000032107	0.000021862
888 888	0.96	0.999999	0.000035635	0.000018213
909	0.94	0.9999	0.000023586	0.000006794
55 221	0.95	0.99999	0.000284857	0.000293719
20 100	0.94	0.9999	0.000119015	0.000099723
2100	0.94	0.9999	0.000037516	0.000013169
Average square error			0.000111606	0.000091506

Table 6
Using nine random patterns to estimate $y = 1 - e^{-2x^4}$

Seed number	r	$\psi(r)$	s_{CBPN}	s_{DNN}
5760	0.95	0.99999	0.000037451	0.000028500
33 333	0.95	0.99999	0.000036811	0.000030885
45 673	0.96	0.999999	0.000108590	0.000050375
10 000	0.96	0.999999	0.000024254	0.000022614
7788	0.96	0.999999	0.000198299	0.000210473
70 000	0.94	0.9999	0.000139202	0.000120949
87	0.95	0.99999	0.000022641	0.000017435
6000	0.94	0.9999	0.000102992	0.000036721
640 000	0.96	0.999999	0.000079564	0.000066711
1234	0.95	0.99999	0.000077390	0.000081819
Average square error			0.000082719	0.000066648

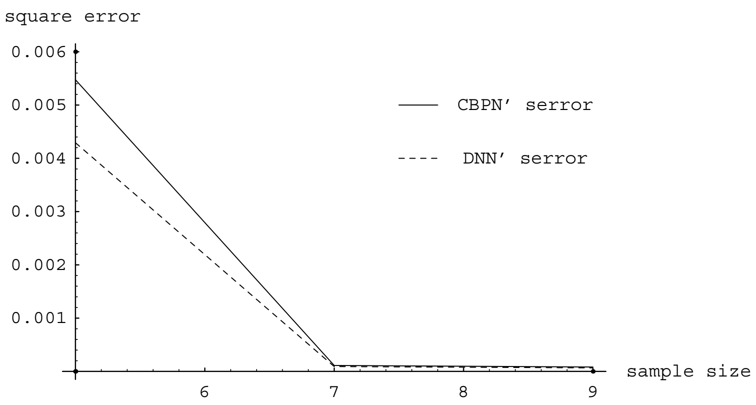

Fig. 6. Square errors of CBPN model and DNN model to estimate $y = 1 - e^{-2x^4}$ with a small sample.

$$B_9 = \frac{0.000082719 - 0.000066648}{0.000082719} \times 100\% = 19\%.$$

8. Conclusion and discussion

This paper shows how the principle of information diffusion can be used to derive more patterns to partly fill up the gaps in a small sample and then we can get a better result when we employ a neural network to recognize a non-linear function. The hybrid model integrating the deriving model and a corresponding back-propagation neural network is called a diffusion-neural-network (DNN).

This paper gives an example and some simulation results to show that DNN is better that CBPN (conventional BP network) to estimate a non-linear function with a small sample consisting of 5–9 patterns. In the case that we use sample $X = \{(0,0),(1/4,1/16),(1/2,1/4),(3/4,9/16),(1,1)\}$ to estimate $y = x^2$, a DNN can reduce error about 48%.

DNN can be considered as a primary model based on the principle of information diffusion to improve accuracy of artificial neural networks with respect to small samples. We have shown that, although the performance of the suggested deriving model satisfied us, this new approach is limited by the quality of the selected diffusion function. Until now we know little about how to get a good quality diffusion function to mine fuzzy information in a small sample. In some sense, the normal diffusion function used in this paper is an experimental function. The function cannot fit any case. To develop DNN model to be more powerful, the first we have to do is to discover more diffusion functions for a variety of populations from which patterns are drawn, and consider other evaluation criteria besides the square error.

Acknowledgements

The first author would like to thank the University of Dortmund, Germany, for providing an excellent research environment during August of 2000 to August of 2001, under a Mercator Visiting Professorship of the German Research Society (DFG). The authors thank Mr. Xingguang Yuan who, using the programs provided by authors, did some simulation experiments in the Section 7.

References

[1] B. Amirakian, H. Nishimura, What size network is good for generalization of a specific task of interest? Neural Networks 7 (2) (1994) 321–329.
[2] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals, and Systems 2 (4) (1989) 303–314.
[3] L. Fletcher, V. Katkovnik, F.E. Steffens, Optimizing the number of hidden nodes of a feedforward artificial neural network, in: Proceedings of the International Joint Conference on Neural Networks, Anchorage, USA, 1998, pp. 1608–1612.
[4] K. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (3) (1989) 183–192.
[5] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice-Hall, Englewood Cliffs, 1994.
[6] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (5) (1989) 359–366.
[7] C.F. Huang, The principle of information diffusion and thought computation and their applications in earthquake engineering, Ph.D. Thesis, Beijing Normal University, Beijing, 1993 (in Chinese).
[8] C.F. Huang, Principle of information diffusion, Fuzzy Sets and Systems 91 (1) (1997) 69–90.
[9] C.F. Huang, Deriving samples from incomplete data, in: Proceedings of FUZZ-IEEE'98, Anchorage, USA, 1998, pp. 645–650.
[10] C.F. Huang, Information diffusion techniques and small sample problem, International Journal of Information Technology and Decision Making 1 (2) (2002) 229–249.
[11] S.C. Huang, Y.F. Huang, Bounds on the number of hidden neurons in multilayer perceptrons, IEEE Transactions of Neural Network 2 (1) (1991) 47–55.
[12] C.F. Huang, Y. Lueng, Estimating the relationship between isoseismal area and earthquake magnitude by hybrid fuzzy-neural-network method, Fuzzy Sets and Systems 107 (2) (1999) 131–146.
[13] C.F. Huang, D. Ruan, Information diffusion principle and application in fuzzy neuron, in: D. Ruan (Ed.), Fuzzy Logic Foundations and Industrial Applications, Kluwer Academic Publishers, Boston, 1996, pp. 165–189.
[14] C.F. Huang, Y. Shi, Towards Efficient Fuzzy Information Processing—Using the Principle of Information Diffusion, Physica-Verlag, Heidelberg, 2002.
[15] P. Kulczycki, Estimating conditional distributions by neural networks, in: Proceedings of the International Joint Conference on Neural Networks, Anchorage, USA, 1998, pp. 344–1349.

[16] K. Van Ha, Hierarchical radial basis function networks, in: Proceedings of the International Joint Conference on Neural Networks, Anchorage, USA, 1998, pp. 1893–1898.
[17] S. Lu, T. Basar, Robust nonlinear system identification using neural network models, IEEE Transactions of Neural Network 9 (3) (1998) 407–429.
[18] C. Moraga, Neuro-fuzzy modeling of compensating systems, in: P. Sinčák, J. Vaščák (Eds.), Quo Vadis Computational Intelligence? Physica-Verlag, Heidelberg, 2000, pp. 385–398.
[19] Y.H. Pao, Adaptive Pattern Recognition and Neural Networks, Addison-Wesley, Reading, MA, 1989.
[20] K.S. Patrick, Artificial Neural Systems Foundations, Paradigms, Applications, and Implementations, McGraw-Hill, New York, 1990.
[21] B.D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge, 1996.
[22] D.E. Rumelhart, J.L. McClelland, the PDP Research Group, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vols. 1&2, MIT Press, Cambridge, MA, 1986.
[23] B.W. Silverman, Density Estimation for Statistics and Data Analysis, Chapman and Hall, London, 1986.
[24] P.P. van der Smagt, Minimisation methods for training feedforward networks, Neural Networks 7 (1) (1994) 1–11.
[25] P.Z. Wang, Fuzzy Sets and Falling Shadows of Random Sets, Beijing Normal University Press, Beijing, 1985 (in Chinese).
[26] Z.N. Wang, C. Dimassimo, M.T. Tham, A.J. Morris, A procedure for determining the topology of multilayer feedforward neural networks, Neural Networks 7 (2) (1994) 291–300.
[27] H. White, Connectionist nonparametric regression: multilayer feedforward network can learn arbitrary mappings, Neural Networks 3 (5) (1990) 535–549.
[28] J. Wray, G.G.R. Green, Neural networks, approximation theory, and finite precision computation, Neural Networks 8 (1) (1995) 31–37.
[29] L.A. Zadeh, Fuzzy sets as a basis for a theory of possibility, Fuzzy Sets and Systems 1 (1) (1978) 3–28.