ELSEVIER

Chemometrics and Intelligent Laboratory Systems 64 (2002) 27–43

Chemometrics and intelligent laboratory systems

www.elsevier.com/locate/chemometrics

An improved differential evolution algorithm in training and encoding prior knowledge into feedforward networks with application in chemistry

Chong-wei Chen [a], De-zhao Chen [a,*], Guang-zhi Cao [b]

[a]Department of Chemical Engineering, Zhejiang University, Hangzhou 310027, PR China
[b]College of Electrical Engineering, Zhejiang University, Hangzhou 310027, PR China

Received 4 May 2002; accepted 22 May 2002

Abstract

Prior-knowledge-based feedforward networks have shown superior performance in modeling chemical processes. In this paper, an improved differential evolution (IDEP) algorithm is proposed to encode prior knowledge simultaneously into networks in training process. With regard to monotonic prior knowledge, IDEP algorithm employs a *flip* operation to adjust those prior-knowledge-violating networks to conform to the monotonicity. In addition, two strategies, Levenberg–Marquardt descent (LMD) strategy and random perturbation (RP) strategy, are adopted to speed up the differential evolution (DE) in the algorithm and prevent it from being trapped by some local minimums, respectively. To demonstrate the IDEP algorithm's efficiency, we apply it to model two chemical curves with the increasing monotonicity constraint. For comparison, four network-training algorithms without prior-knowledge constraints, as well as three existing prior-knowledge-based algorithms (which have some relationship and similarities with IDEP algorithm), are employed to solve the same problems. The simulation results show that IDEP's performance is better than all other algorithms. As a conclusion, IDEP algorithm and its promising prospective will be discussed in detail at the end of this paper.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Prior knowledge; Feedforward network; Improved differential evolution; Flip operation; Levenberg–Marquardt descent strategy; Random perturbation strategy

1. Introduction

Three-layer feedforward networks have been commonly used in modeling chemical processes because it can approximate arbitrary complex function with any precision [1,2]. Generally, training networks to approximate functions can be formulated as minimization of an error function, such as the summed square error between target and actual outputs over all training data, by iteratively adjusting connection weights. The networks trained in this way will only depend on the training data, and no inherent information of the chemical process is considered during the modeling.

However, functional relationship of the actual chemical process probably possesses some special properties, such as monotonicity and concavity, which exist apart from the sample data. Such property

[*] Corresponding author.
E-mail address: dzchen@mail.hz.zj.cn (D. Chen).

is called prior knowledge of the model. Usually, prior knowledge is obtained according to some first-principle models so it is reliable. But the actual training data are finite, sparse and noisy. In addition, they may not be able to embody the prior knowledge very well so the network models that completely rely on these data may be inaccurate and even violate the prior knowledge. Accordingly, its performance and practicability are degraded. Moreover, since never used in modeling process explicitly, the well-grounded prior knowledge of the chemical process is overlooked and wasted.

To solve this problem, some researchers, such as Joerding and Meador [3], Thompson and Kramer [4] and Chen et al. [5,6], proposed some methods to encode prior knowledge into neural networks, called prior-knowledge-based methods. Prior-knowledge-based methods can be classified into three classes, namely weight constraint (WC) methods, architecture constraint (AC) methods and data constraint (DC) methods [3,6]. The networks trained by these methods have turned out to perform better in simulation experiments [5,6], and they can solve the overfitting problem very well [6]. However, the existing prior-knowledge-based methods still have some disadvantages. The existing WC methods, such as Joerding's penalty function (J.PF) method [3,6], are not very good at approximation. The existing AC methods, such as exponential weight (EW) method [6], provide relatively little freedom in searchable weight space so it is usually difficult to obtain the training goal. The existing DC methods, such as interpolation (IP) method [6], may possibly not impose prior knowledge on networks very well in some situations. Therefore, it is quite necessary and desirable to develop a method that can combine the information of both the prior knowledge and the training data into neural networks simultaneously and efficiently.

On the other hand, a class of stochastic search and optimization methods, called evolutionary algorithms (EAs), has received considerable and increasing interest over the last decade [7]. Based on the biological evolution in nature, these algorithms apply the principle of survival of the fittest to produce successively better approximations to an optimum. EAs have also been applied to train neural networks by reformulating the training process as the evolution of connection weights and the error function as the fitness to be optimized [8,9]. The evolutionary training algorithms are preferable to the first-order gradient-descent-based training algorithms in their reliability and faster convergence speed [10]. However, because networks' weight space is real-valued, large and very complex, it is still a relatively time-consuming task for normal EAs, such as Goldberg's [11] SGA, to train networks. Fortunately, Storn and Price [12] present a differential evolution (DE) algorithm. It turns out to be one of the best genetic algorithms for solving real-valued problems because it can find the best or satisfactory solutions very fast. Therefore, it is soon adopted to train neural networks [13], which is called NC-DET algorithm in this paper. Despite the great speed improvement, NC-DET is still much slower when compared with some second-order optimization algorithms, such as the conjugate gradient algorithm [14,16] and Levenberg–Marquardt (LM) algorithm [15]. Therefore, it is necessary to improve NC-DET algorithm in order to make it a practicable training algorithm for neural networks.

In this paper, an improved differential evolution (IDEP) algorithm is proposed to train neural networks and encode domain-related prior knowledge into them simultaneously. In IDEP algorithm, with regard to monotonic prior knowledge, a *flip* operation is employed to adjust those prior-knowledge-violating networks to conform to the monotonicity. In addition, some other genetic operations, e.g. selection, also help encode prior knowledge into networks, and two strategies, Levenberg–Marquardt descent (LMD) strategy and random perturbation (RP) strategy, are employed to speed up the evolution process and prevent it from converging at some local minimums, respectively.

To validate its effectiveness, IDEP algorithm is applied to model the true boiling point curve of crude oil and the curve of effect of pressure on entropy with prior knowledge of increasing monotonicity. After comparing with the simulation results of other network modeling methods, including four network-training algorithms without prior-knowledge constraints and three existing prior-knowledge-based algorithms, we conclude that IDEP algorithm is faster, more reliable and robust, and it can encode prior knowledge into networks very well.

At the end of this paper, the simulation results and key parts of IDEP algorithm are analyzed in detail. In addition, its promising prospective and future works are discussed as a conclusion.

2. Neural network and prior knowledge

In this section, the feedforward network and prior knowledge of increasing monotonicity used in this paper will be introduced in detail.

2.1. Structure of the feedforward network

A commonly used three-layer feedforward network model *net* is illustrated in Fig. 1. Here, the first layer is the input layer, which accepts an m-dimensional input vector $\boldsymbol{x}$. The second layer has n hidden nodes, and each one uses the following tansig function $g(x)$ to process the weighted input:

$$g(x) = \frac{2}{1+e^{-2x}} - 1 \qquad (2\text{-}1)$$

The third layer is the output layer, which has q nodes and uses linear transfer function. The output vector is a q-dimension vector $\hat{\boldsymbol{y}}$, which is the prediction of the target output $\boldsymbol{y}$. Let $\boldsymbol{W}^{[2]}$ and $\boldsymbol{W}^{[3]}$ denote the weight matrices between layers one and two and layers two and three, respectively, including the biases. $w_{ij}^{[h]}$, the element of $\boldsymbol{W}^{[h]}$, denotes the interconnection weight between the jth node of the $(h-1)$th layer and the ith node of the hth layer. $b_i^{[h]}$ denotes the bias to the ith node of the hth layer ($h=2$, 3). Thus, the network model *net* can be expressed by the following function:

$$\hat{\boldsymbol{y}}(\boldsymbol{x}) = f(\boldsymbol{W},\boldsymbol{x}) = \boldsymbol{W}^{[3]} g(\boldsymbol{W}^{[2]}\boldsymbol{x}). \qquad (2\text{-}2)$$

For brevity, we will substitute $\boldsymbol{W}$ for $\boldsymbol{W}^{[2]}$ and $\boldsymbol{W}^{[3]}$ in some of the formulas.

In this paper, with regard to monotonic prior knowledge, the input and output vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ are both one-dimensional vectors, i.e. $m=q=1$. Therefore, in the following text, they will be denoted by scalars x and y, respectively. In addition, (x_i, y_i), $i=1, 2, \ldots, l$ represents the training data accordingly.

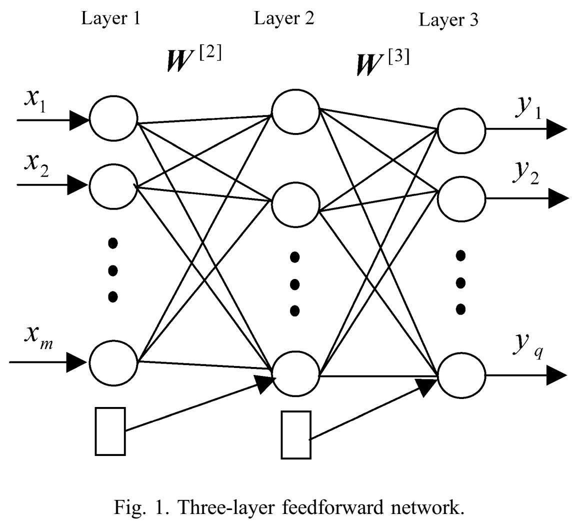

Fig. 1. Three-layer feedforward network.

2.2. Parameters for network training

The general parameters for network training are: the performance function (PE), the training goal (*goal*) and the maximum number of iterations (*epochs*). The training will stop if the number of iterations exceeds *epochs* or PE gets lower than *goal*. In the following text, we will take $\mathrm{PE} = \mathrm{SSE} = \sum_i (y_i - \hat{y}(x_i))^2$ as the performance function unless there is a special explanation.

2.3. Prior knowledge of increasing monotonicity

A function $y=f(x)$ is called to be monotonically increasing in its domain D_x if:

$$\forall x_1, x_2 \in D_x, \text{ if } x_1 < x_2 \text{ then } f(x_1) \le f(x_2). \qquad (2\text{-}3)$$

Many chemical curves, such as the true boiling point curve of the crude oil, the curve of effect of pressure on entropy and the curve of enthalpy on temperature, have been known as monotonically increasing, which is a prior knowledge. If we use feedforward networks to approximate those curves, it is important to insure that the networks conform to the prior knowledge.

2.4. Performance criteria for network modeling methods

Generally, the performances of different network modeling methods are inspected in the following three aspects: the approximation accuracy, the prediction ability to the testing data and the network model's conformance of the prior knowledge.

For each method, the cross-validation [17,18] is employed to inspect the approximation accuracy and prediction ability of a trained network. That is, we take out a datum from the sample data as temporary 'testing data' in turn, and use the remaining data to model a network and predict on the testing data. The mean square error (MSE) of the training data for one model in cross-validation is:

$$\mathrm{MSE} = \frac{1}{l}\sum_i (y_i - \hat{y}(x_i))^2, \tag{2-4}$$

where l is the total number of the training data. The approximation accuracy of each method is the mean of the MSE (denoted by $\overline{\mathrm{MSE}}$).

The relative prediction error on the current testing data for one model in cross-validation is:

$$r_{\mathrm{e}} = \frac{y - \hat{y}(x)}{y}. \tag{2-5}$$

The prediction ability for each training method can be measured by the mean of the $|r_{\mathrm{e}}|$ values (denoted by $\overline{|r_{\mathrm{e}}|}$) and the standard deviation of the r_{e} values (denoted by $\delta_{r_{\mathrm{e}}}$) of the different models in cross-validation modeled by that method.

As for the conformance of the prior knowledge, in this paper, we need to analyze the nonmonotonic intervals of the network function $\hat{y}$ versus x. To do this, we calculate the points where $\hat{y}'(x)=0$ by a numerical method, and exclude the inflexions to determine the nonmonotonic intervals. Because the nonmonotonic intervals cannot be averaged among different models, we total the times when the model has one or several nonmonotonic intervals in cross-validation. Moreover, we plot the curve and give out the nonmonotonic interval for one model selected at random (we just select the first model) in cross-validation as the delegate. If the selected model has more than one nonmonotonic interval, the interval closest to minus infinity will be given out in the performance data.

3. Some existing prior-knowledge-based methods

3.1. J.PF method

Joerding and Meador [3] presented J.PF method to impose the increasing monotonicity on feedforward networks. In their method, a sufficient (but not necessary) condition for $\hat{y}(x)$ to satisfy the increasing monotonicity is deduced:

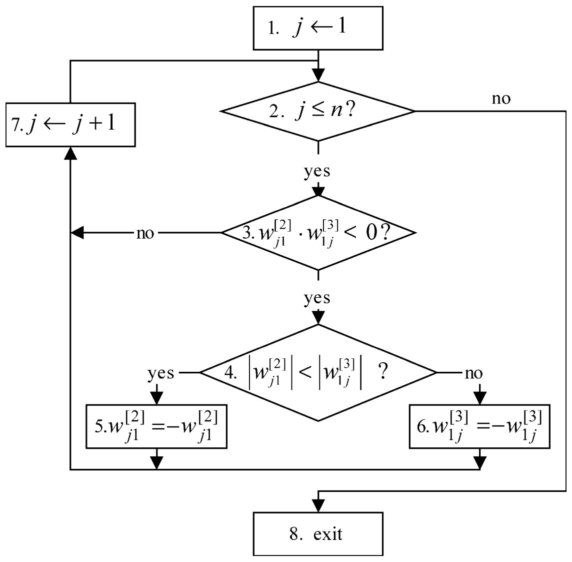
1. $j \leftarrow 1$
2. $j \le n$?
no
7. $j \leftarrow j+1$
yes
no
3. $w_{j1}^{[2]} \cdot w_{1j}^{[3]} < 0$?
yes
yes
4. $|w_{j1}^{[2]}| < |w_{1j}^{[3]}|$?
no
5. $w_{j1}^{[2]} = -w_{j1}^{[2]}$
6. $w_{1j}^{[3]} = -w_{1j}^{[3]}$
8. exit

Fig. 2. Flip operation.

$$w_{j1}^{[2]} w_{1j}^{[3]} > 0, \; j = 1, \; 2, \; \ldots, \; n, \tag{3-1}$$

Based on this, J.PF method utilizes the following performance function to eliminate the nonmonotonic intervals in network models:

$$\mathrm{PE}(\boldsymbol{W}) = \mathrm{SSE}(\boldsymbol{W}) + L(\boldsymbol{W}) \tag{3-2}$$

where $L(\boldsymbol{W}) = \sum_j \mu(\theta w_{j1}^{[2]} w_{1j}^{[3]})(e^{\theta w_{j1}^{[2]} w_{1j}^{[3]}} - 1)$ is a penalty function with

$$\mu(\theta w_{j1}^{[2]} w_{1j}^{[3]}) = \begin{cases} 1 & \theta w_{j1}^{[2]} w_{1j}^{[3]} > 0 \\ 0 & \text{otherwise} \end{cases},$$

here, θ is a modulatory constant.

In practice, due to the introduction of the penalty function, J.PF's approximation accuracy and prediction ability are not very good. Moreover, sometimes, it even cannot ensure all of the models in cross-validation are monotonically increasing [6].

3.2. EW method

Chen et al. [6] derive the following sufficient but not necessary condition for $\hat{y}(x)$ to be monotonically increasing:

$$w_{j1}^{[2]} > 0, \text{ and } w_{1j}^{[3]} > 0 \; j = 1, 2, \ldots, n, \tag{3-3}$$

which is obviously stronger than the condition in Eq. (3-1). In addition, according to Eq. (3-3), they proposed EW method, which replaces the original $w_{j1}^{[2]}$ and $w_{1j}^{[3]}$ in network with $e^{w_{j1}^{[2]}}$ and $e^{w_{1j}^{[3]}}$ ($j = 1,2,\ldots,n$) to perform all the weight adjustments and network computations (the biases are intact and computed in the original way). Clearly, networks working in this way are certain to satisfy Eq. (3-3) and, hence, necessarily conform to the increasing monotonicity. However, since Eq. (3-3) is a strengthened sufficient condition, it permits little freedom for the training, which makes it difficult to obtain the training goal under some circumstances, thus, hampering the approximation accuracy and prediction ability.

3.3. IP method

IP method [6] is a kind of DC method. It interpolates some data satisfying the prior knowledge into original sample data, and then utilizes these compound data to train the network; during the training process, the added artificial data will be deleted gradually. This method utilizes the interpolation points to embody the given prior knowledge. Gener-

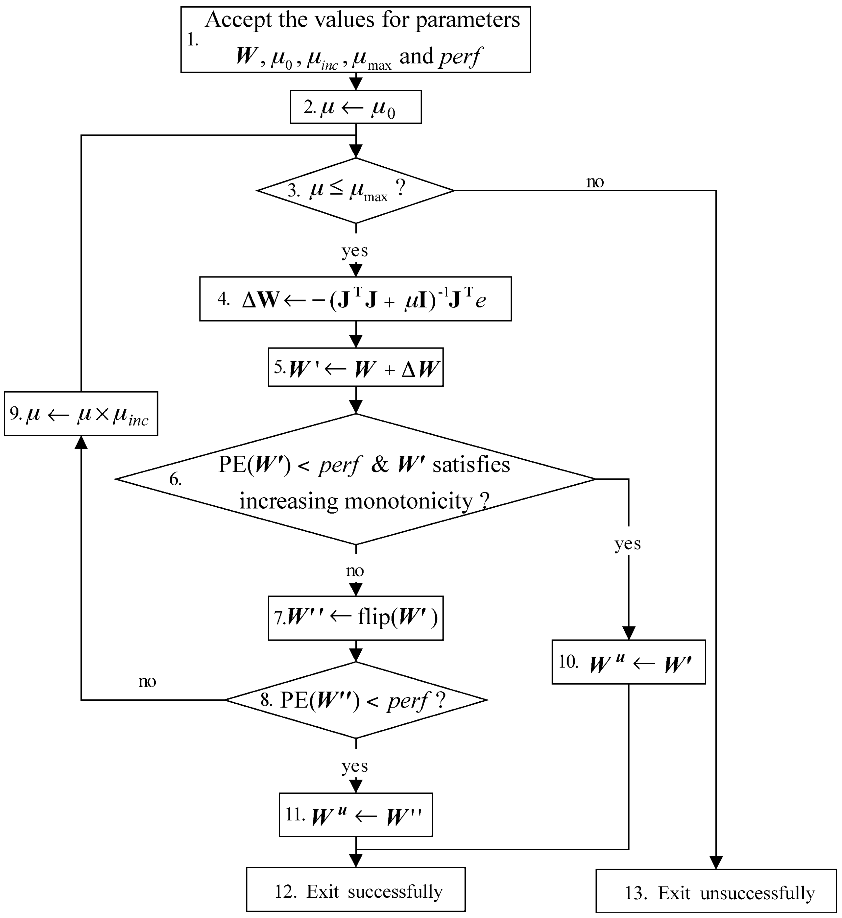
1. Accept the values for parameters $W, \mu_0, \mu_{inc}, \mu_{max}$ and $perf$
2. $\mu \leftarrow \mu_0$
3. $\mu \le \mu_{max}$?
no
yes
4. $\Delta W \leftarrow -(J^T J + \mu I)^{-1} J^T e$
5. $W' \leftarrow W + \Delta W$
9. $\mu \leftarrow \mu \times \mu_{inc}$
6. PE(W') < $perf$ & W' satisfies increasing monotonicity ?
yes
no
7. $W'' \leftarrow$ flip(W')
10. $W^u \leftarrow W'$
no
8. PE(W'') < $perf$?
yes
11. $W^u \leftarrow W''$
12. Exit successfully
13. Exit unsuccessfully

Fig. 3. LMD strategy.

ally, the interpolation points are acquired by using first-principle models and they are equally spaced with respect to the original data, so that the addition and deletion of the interpolation points will not have a significant impact on network models. In IP method, a parameter d, called the interpolation density, is used to control the number of the artificial interpolation data sets.

IP method possesses good approximation accuracy and prediction ability. However, sometimes, it cannot ensure that all of the models in cross-validation are monotonically increasing.

We will see the above properties of J.PF, EW and IP methods from the examples in Section 6.

4. NC-DET and DEP algorithms

Training a feedforward network can be formulated as solving the following unconstrained optimization problem:

$$\min_{\boldsymbol{W}} \mathrm{PE}(\boldsymbol{W}) \tag{4-1}$$

where the weight matrix $\boldsymbol{W}$ of the network is the optimization variable. Accordingly, encoding the prior knowledge of increasing monotonicity into a network is equivalent to solving the following constrained optimization problem:

$$\min_{\boldsymbol{W}} \mathrm{PE}(\boldsymbol{W})$$

$$\text{s.t. } f(\boldsymbol{W},x_1) \leq f(\boldsymbol{W},x_2), \forall x_1 < x_2 \in D_x. \tag{4-2}$$

All the algorithms in optimization theory can be applied to problems (4-1) and (4-2) and so can the EAs. In this section, we will employ NC-DET algorithm to solve problem (4-1) and its variant DEP algorithm to solve problem (4-2).

4.1. NC-DET algorithm

Jouni and Miika [13] adopt DE algorithm [12,20] to train feedforward networks. Their training algorithm is called NC-DET algorithm in this paper. It is a parallel direct search algorithm that utilizes N_p individuals $\boldsymbol{W}_i^G$ ($i=1, \ldots, N_\mathrm{p}$) as a population for each generation G. Each individual $W_i^G=[w_1, w_2, \ldots, w_n]_i^G$ consists of all the weight parameters in a feedforward network. PE($\boldsymbol{W}^G$) is the fitness of the individual $\boldsymbol{W}^G$. The smallest fitness in each generation corresponds to the performance function value in network training process. The basic structure of NC-DET algorithm is described as follows.

(1) Initialization operation: initialize the weight parameters $\boldsymbol{W}_i^0$ ($i=1, \ldots, N_\mathrm{p}$) with Nguyen and Widrow's [19] initialization method, and let $G \leftarrow 0$.

(2) For each individual $\boldsymbol{W}_i^G$ ($i=1, 2, \ldots, N_\mathrm{p}$) in the Gth generation, do steps 3–6 to produce the population of the ($G+1$)th generation.

(3) Mutation operation: in this operation, a perturbed individual $\hat{W}_i^{G+1}$ is generated according to:

$$\hat{\boldsymbol{W}}_i^{G+1} = \boldsymbol{W}_\mathrm{p}^G + F(\boldsymbol{W}_j^G - \boldsymbol{W}_k^G) \tag{4-3}$$

with $j, k, p \in [1,N_\mathrm{p}]$, integer and mutually different and $F>0$.

The integers j, k and p are chosen randomly from the interval $[1,N_\mathrm{p}]$, and are different from the running index i. $F\in[0,2]$ is a real constant factor that

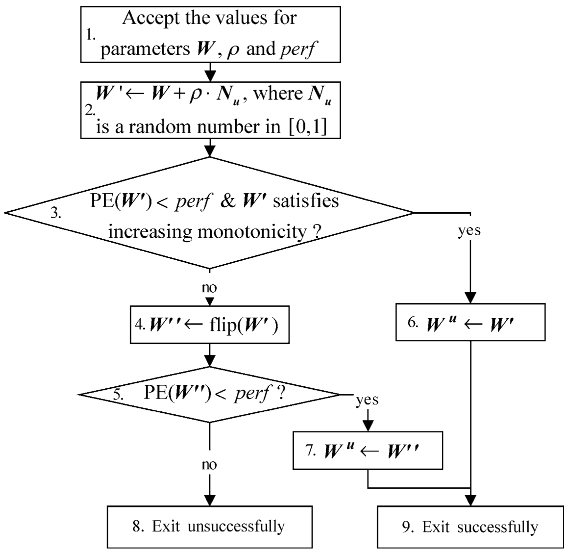
1. Accept the values for parameters $\boldsymbol{W}$, ρ and $perf$
2. $\boldsymbol{W}' \leftarrow \boldsymbol{W} + \rho \cdot N_u$, where N_u is a random number in [0,1]
3. PE($\boldsymbol{W}'$) < $perf$ & $\boldsymbol{W}'$ satisfies increasing monotonicity ?
yes
no
4. $\boldsymbol{W}'' \leftarrow$ flip($\boldsymbol{W}'$)
6. $\boldsymbol{W}^u \leftarrow \boldsymbol{W}'$
5. PE($\boldsymbol{W}''$) < $perf$?
yes
7. $\boldsymbol{W}^u \leftarrow \boldsymbol{W}''$
no
8. Exit unsuccessfully
9. Exit successfully

Fig. 4. RP strategy.

controls the amplification of the differential variation ($\boldsymbol{W}_j^G - \boldsymbol{W}_k^G$). Note that the individual $\boldsymbol{W}_\mathrm{p}^G$, which is perturbed to yield $\hat{\boldsymbol{W}}_i^{G+1}$, has no relationship to $\boldsymbol{W}_i^G$, but is a randomly chosen population member.

(4) Cross-over operation: to increase the diversity of the offspring in the next generation, the perturbed individual $\hat{\boldsymbol{W}}_i^{G+1}$ and the current individual $\boldsymbol{W}_i^G$ are selected by a binomial distribution to perform the cross-over operation to generate the offspring. In the

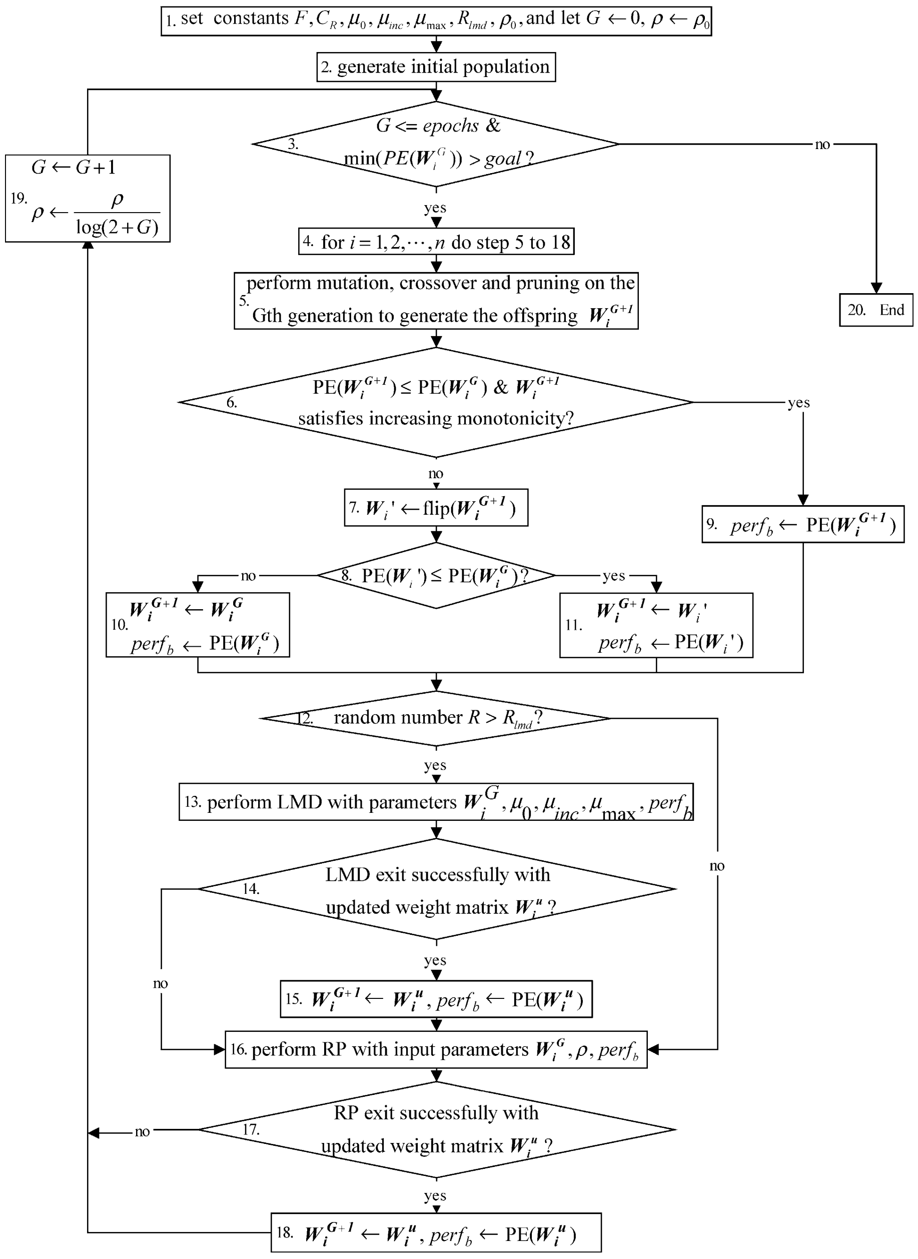
1. set constants $F, C_R, \mu_0, \mu_{inc}, \mu_{\max}, R_{lmd}, \rho_0$, and let $G \leftarrow 0$, $\rho \leftarrow \rho_0$
2. generate initial population
3. $G <= epochs$ & $\min(PE(\boldsymbol{W}_i^G)) > goal$?
no
20. End
yes
4. for $i = 1, 2, \cdots, n$ do step 5 to 18
5. perform mutation, crossover and pruning on the Gth generation to generate the offspring $\boldsymbol{W}_i^{G+1}$
6. $\mathrm{PE}(\boldsymbol{W}_i^{G+1}) \le \mathrm{PE}(\boldsymbol{W}_i^G)$ & $\boldsymbol{W}_i^{G+1}$ satisfies increasing monotonicity?
yes
no
7. $\boldsymbol{W}_i' \leftarrow \mathrm{flip}(\boldsymbol{W}_i^{G+1})$
9. $perf_b \leftarrow \mathrm{PE}(\boldsymbol{W}_i^{G+1})$
8. $\mathrm{PE}(\boldsymbol{W}_i') \le \mathrm{PE}(\boldsymbol{W}_i^G)$?
no
yes
10. $\boldsymbol{W}_i^{G+1} \leftarrow \boldsymbol{W}_i^G$ $perf_b \leftarrow \mathrm{PE}(\boldsymbol{W}_i^G)$
11. $\boldsymbol{W}_i^{G+1} \leftarrow \boldsymbol{W}_i'$ $perf_b \leftarrow \mathrm{PE}(\boldsymbol{W}_i')$
12. random number $R > R_{lmd}$?
yes
no
13. perform LMD with parameters $\boldsymbol{W}_i^G, \mu_0, \mu_{inc}, \mu_{\max}, perf_b$
14. LMD exit successfully with updated weight matrix $\boldsymbol{W}_i^u$?
yes
no
15. $\boldsymbol{W}_i^{G+1} \leftarrow \boldsymbol{W}_i^u$, $perf_b \leftarrow \mathrm{PE}(\boldsymbol{W}_i^u)$
16. perform RP with input parameters $\boldsymbol{W}_i^G, \rho, perf_b$
17. RP exit successfully with updated weight matrix $\boldsymbol{W}_i^u$?
no
yes
18. $\boldsymbol{W}_i^{G+1} \leftarrow \boldsymbol{W}_i^u$, $perf_b \leftarrow \mathrm{PE}(\boldsymbol{W}_i^u)$
19. $G \leftarrow G+1$ $\rho \leftarrow \dfrac{\rho}{\log(2+G)}$

Fig. 5. IDEP algorithm.

cross-over operation, the *j*th gene of the *i*th individual of the next generation is produced from the perturbed individual $\hat{\boldsymbol{W}}_i^{G+1}=[\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_n]_i^{G+1}$ and the current individual $\boldsymbol{W}_i^G=[w_1, w_2, \ldots, w_n]_i^G$ as follows:

$$w_{ki}^{G+1} = \begin{cases} w_{ki}^G, & \text{if a random number} > C_R \\ \hat{w}_{ki}^{G+1}, & \text{otherwise;} \end{cases}, \quad k = 1, \ldots, n, \tag{4-4}$$

where the cross-over factor $C_R \in [0,1]$ is set by users.

(5) Pruning operation: after some generations' evolution, the absolute value of some individuals' genes may become so large as to ill-conditioned computations. In this case, the offspring $\boldsymbol{W}_i^{G+1}$ is pruned to solve this problem:

$$w_{ki}^{G+1} = \begin{cases} -B_p + 2RB_p, & \text{if } |w_{ki}^{G+1}| > B_p \\ w_{ki}^{G+1}, & \text{otherwise;} \end{cases}, \quad k = 1, \ldots, n, \tag{4-5}$$

where *R* is a random number in the interval [0,1] and B_p is the bound of the absolute value of connection weights. Generally, B_p is a large positive number, and in this paper, it is always set to 10,000 for all DE-based algorithms, including NC-DET, DEP and IDEP.

(6) Evaluation operation: the offspring $\boldsymbol{W}_i^{G+1}$ competes one-to-one with its parent $\boldsymbol{W}_i^G$. This competition implies that the parent is replaced by its offspring if the offspring's fitness is equal or smaller than its parent's. Otherwise, the parent is retained in the next generation. The evaluation operation is, therefore, expressed as:

$$W_i^{G+1} = \begin{cases} W_i^{G+1} & \text{if } PE(W_i^{G+1}) \le PE(W_i^G) \\ W_i^G & \text{otherwise} \end{cases}, \tag{4-6}$$

(7) $G \leftarrow G+1$

(8) Repeat of steps 2–7 until either the number of generation *G* exceed max generation number *epochs* or the smallest fitness in the current generation, i.e. min(PE($\boldsymbol{W}_i^G$)), drops below *goal*.

(9) Take the individual $W_b{}^{Gf}$ with the smallest fitness from the last generation G_f as the resulting trained weight matrix for the feedforward network.

Compared with gradient-based training algorithms, NC-DET algorithm is faster, and it does not depend on gradient information of the performance function. In addition, NC-DET algorithm is generally much less sensitive to initial training conditions. Generally, it can find a near-optimal set of connection weights globally, while a gradient descent algorithm can only find a local optimum in a neighborhood of the initial solution.

However, although NC-DET algorithm is good at global search, it performs poorly in localized search, which makes it much slower than some second-order search algorithms, such as the conjugate gradient algorithm [14,16], quasi-Newton algorithm [21] and LM training algorithm [15]. In addition, usually, the individuals yielded by NC-DET algorithm tend to cluster closely at early stage of the evolution [22,23]. Therefore, as observed in Eqs. (4-3) and (4-

Table 1
Data for the true boiling point of a crude oil

	No.														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p (%)	0	1.53	2.34	3.46	4.40	5.36	6.79	8.21	9.76	11.24	12.78	15.03	17.19	19.59	21.09
t (°C)	75	90.32	107	129	143	165	191	210	230	246	260	280	295	312	321
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
p (%)	23.58	24.75	26.34	28.05	30.71	34.18	35.57	40.95	44.50	47.89	50.21	51.73	54.42	57.32	59.77
t (°C)	339	350	365	380	403	439	467	480	481	482	490	500	510	512	513

4), the mutation and cross-over operations can hardly generate new better individuals. This results in a premature convergence and hence reduces the global search ability of the algorithm. Finally, NC-DET algorithm performs without considering prior knowledge constraints, so the trained networks may violate it greatly.

4.2. DEP algorithm

We ever presented a variant of NC-DET algorithm, called DEP algorithm, which is able to encode the prior knowledge of increasing monotonicity into networks during its training process. DEP algorithm is identical to NC-DET except the initialization and evaluation operations.

In DEP's initialization operation, the initial population is first generated as step 1 of NC-DET algorithm. Then each individual in the population will be checked whether it conforms to the increasing monotonicity or not. Those violating the prior knowledge will be adjusted to satisfy it though a certain operation (in this paper, we use the flip operation, which will be described in detail in Section 5.1). Thus, after the operation, no individuals in the initial population will violate the prior knowledge.

In DEP's evaluation operation, if $\boldsymbol{W}_i^{G+1}$ satisfies the prior knowledge of increasing monotonicity, it will competes one-to-one with its parent $\boldsymbol{W}_i^G$. Otherwise, $\boldsymbol{W}_i^{G+1}$ is excluded right away and the parent $\boldsymbol{W}_i^G$ is retained to the next generation.

From DEP's initialization and evaluation operations, we can see that during the evolution process DEP excludes all the individuals that violate the prior knowledge. Therefore, networks resulted from such a training process will consequentially satisfy the prior knowledge.

DEP algorithm imposes the prior knowledge constraint on the evolution process, which reduces the probability of introducing new individuals into the population through the evaluation operation when compared with NC-DET algorithm. Therefore, DEP algorithm is more likely to cluster and prematurely converge at an early stage of the evolution and it is more difficult for DEP algorithm to find a global optimum, which may lead to a poor approximation to the training data. The simulation results in Section 6 confirm it.

In order to approximate the training data with high accuracy and encode the prior knowledge as well, an IDEP algorithm is proposed in the following section. Compared with NC-DET and DEP algorithms, it is more fast, reliable and robust. In Section 6, two chemical examples are employed to demonstrate its effectiveness.

5. IDEP algorithm

As an improved version of DE, IDEP algorithm trains networks with both the training data and the prior knowledge information of increasing monotonicity. The key parts of IDEP algorithm include flip operation, LMD strategy and RP strategy, and we will introduce them respectively in this section.

5.1. Flip operation

In Section 3.1, we state that Eq. (3-1) is a sufficient condition for a network to be monotonically increasing in its domain. Based on this, a flip operation is introduced to force a network to conform to the increasing monotonicity by changing the signs of its weights. Flip operation works as a subroutine, and it accepts the weight matrix $\boldsymbol{W}$ of a neural network as the input parameter.

The diagram of flip operation is shown in Fig. 2.

In Fig. 2, step 3 verifies whether the product of the weight pairs $w_{j1}^{[2]}$ and $w_{1j}^{[3]}$ satisfies Eq. (3-1) or not. If not, the sign of the one among $w_{j1}^{[2]}$ and $w_{1j}^{[3]}$ with a smaller absolute value will be changed. Intuitively, it seems to flip symmetrically around the origin on the real axis, so we call it flip operation. The purpose of choosing the smaller absolute weight value is to change the original weight matrix as little as possible. This procedure is performed on all of the weight pairs of every hidden node. Therefore, after flip operation, the weight matrix will certainly conform to the increasing monotonicity.

Note that Eq. (3-1) is not a necessary condition for a network to be monotonically increasing. Therefore, there must exist some networks that conform to increasing monotonicity but not satisfy Eq. (3-1). Therefore, when applied on such networks, flip operation is not expected to rescue a prior-knowledge-violated network (since it already satisfied the prior knowledge),

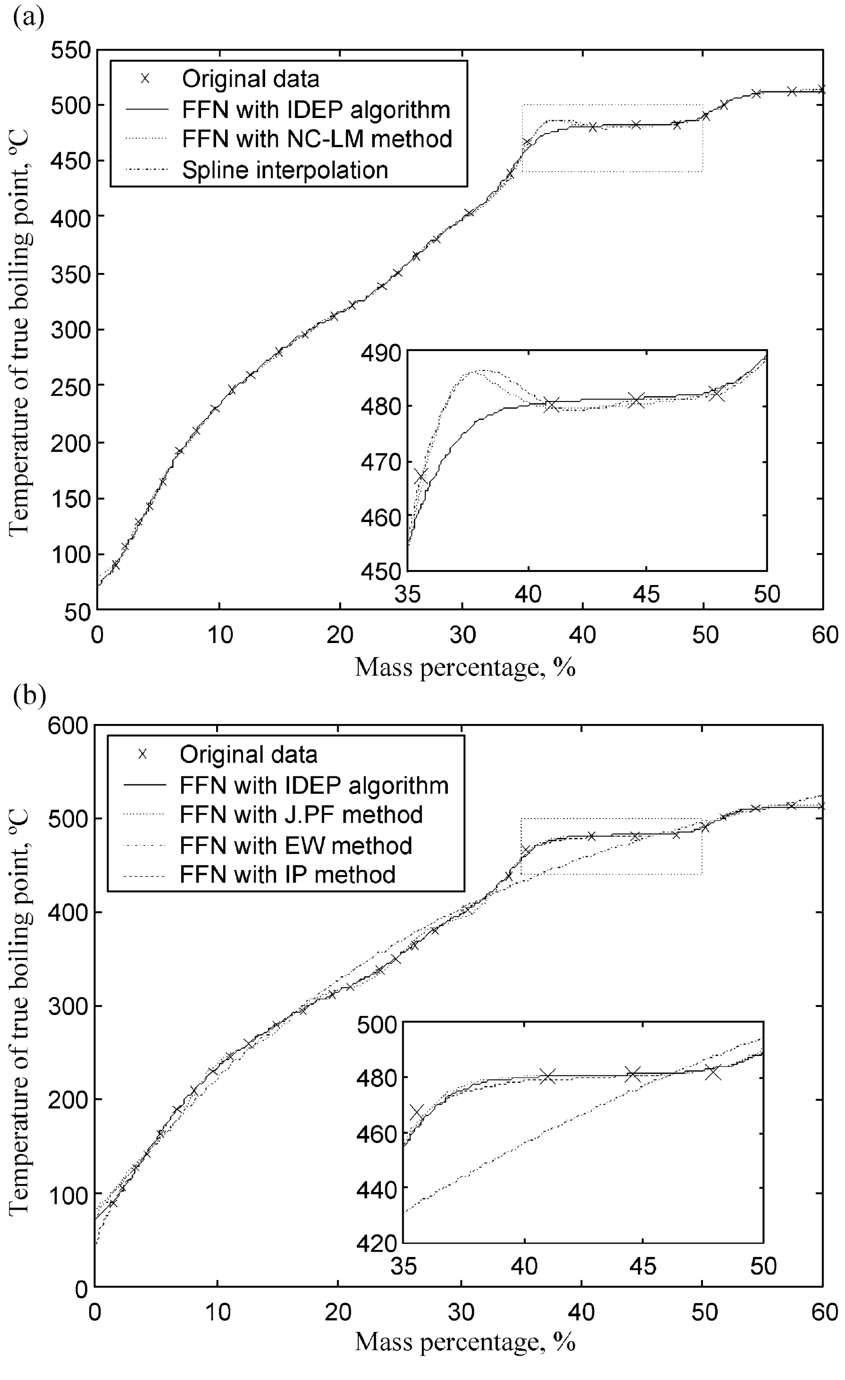

Fig. 6. (a) Comparison between the different methods in modeling the true boiling point curve of crude oil. (b) Comparison between the different methods in modeling the true boiling point curve of crude oil.

Table 2
Performance of the different prior-knowledge-based methods in modeling the true boiling point of a crude oil

Method	Mean of approximation accuracy ($\overline{\text{MSE}}$)	Mean of relative prediction error ($\overline{\lvert r_e \rvert}$)	S.D. of relative prediction error (δ_{r_e})	Number of nonmonotonic models/total models in cross-validation	Nonmonotonic interval of the first model in cross-validation
NC-LM	2.9431×10^{-6}	0.0545	0.0939	27/28	[37.80%, 41.70%]
NC-CGF	4.1291×10^{-5}	0.2473	0.8648	24/28	[39.70%, 44.70%]
NC-DET	8.4616×10^{-5}	35,199.7	186,095	23/28	None
IDEP	3.3430×10^{-6}	0.0333	0.0640	0/28	None
DEP	1.0951×10^{-4}	0.1317	0.5191	0/28	None
J.PF	1.3046×10^{-4}	0.2341	0.6861	10/28	None
IP	2.2268×10^{-6}	0.0401	0.1046	1/28	None
EW	1.2115×10^{-4}	0.0531	0.1006	0/28	None

but to produce a new network that both differs from the original one and satisfies the prior knowledge.

5.2. LMD strategy

In Section 4.2, we claim DEP algorithm performs poorly in a localized search, which will decrease its speed dramatically. To overcome this disadvantage, IDEP algorithm employs LMD strategy to accelerate the local search and convergence.

LMD strategy is based on the LM algorithm [15]. LM algorithm is developed to approach second-order training speed without having to compute the Hessian matrix. We assume the absolute error is $e = y_i - \hat{y}(x_i)$ at the training datum (x_i, y_i). When the performance function is SSE($\boldsymbol{W}$), LM algorithm uses the approximation to the Hessian matrix in the following weight update:

$$\Delta \boldsymbol{W} = -(\boldsymbol{J}^T \boldsymbol{J} + \mu \boldsymbol{I})^{-1} \boldsymbol{J}^T e, \tag{5-1}$$

where $\boldsymbol{J}$ is the Jacobian matrix, which contains the first derivations of the network errors with respect to the weights and biases; μ is an modulatory parameter.

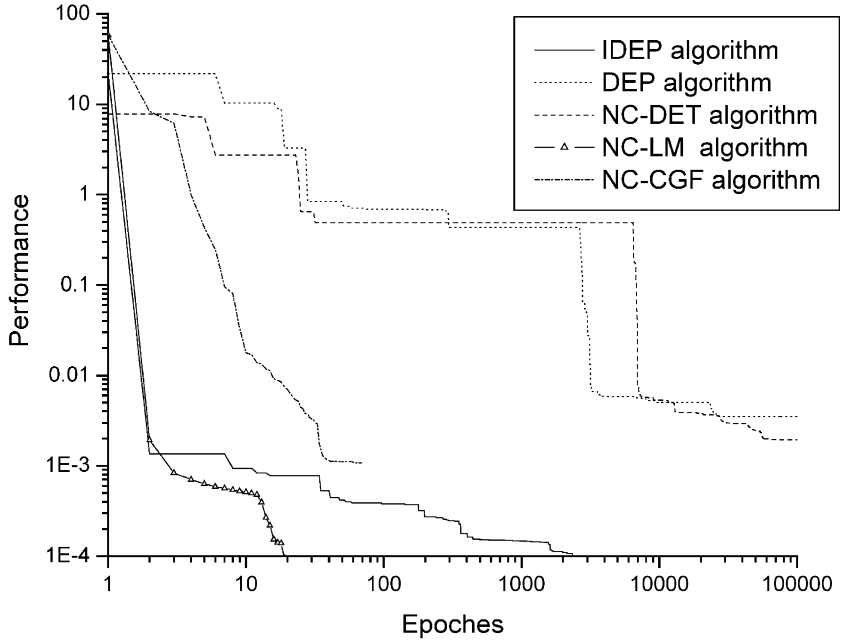

Fig. 7. Training progress of the different methods in modeling the true boiling point curve of crude oil (the first model in cross-validation).

Table 3
Performance of the different algorithms in modeling the true boiling point of a crude oil (the first model in cross-validation)

Training algorithm	$\overline{\text{Gen}}$	$\overline{\text{PE}_\text{f}}$	C_g/C_t
IDEP	4231.1	9.9986×10^{-5}	10/10
DEP	100,001	0.0036	0/10
NC-DET	100,001	0.0019	0/10
NC-CGF	74.2	0.0011	0/10
NC-LM	22.2	8.6658×10^{-5}	10/10

When μ is zero, it is just Newton's method [24], using the approximate Hessian matrix. When μ is very large, it becomes a gradient descent with a small step size. Since Newton's method is faster and more accurate near an error minimum, the aim is to shift towards Newton's method as quickly as possible. Thus, μ is set to a small initial value and is increased when a tentative step would increase the performance function.

LMD strategy works as an independent subroutine with five input arguments: the weight matrix $\boldsymbol{W}$; the initial value, increasing speed and maximum value of the modulatory parameter μ_0, μ_{inc} and μ_{max}, respectively; and the goal of the performance function *perf* (it must be less than or equal to SSE($\boldsymbol{W}$)).

The diagram of LMD strategy is shown in Fig. 3.

As we state in Section 5.1, when PE($\boldsymbol{W}'$) $\geq$ *perf* and the network satisfies increasing monotonicity in step 6, the purpose of flip operation in step 7 is to introduce a new valid network so as to increase the diversity of the population. In this sense, flip operation can also be regarded as a kind of perturbation on the original network. Similar cases happen in the diagram of RP strategy and IDEP algorithm.

If an updated weight matrix $\boldsymbol{W}^u$ is found by which the network is constructed has a smaller performance function value than *perf* and meanwhile conforms to increasing monotonicity, LMD strategy will exit successfully. Otherwise, the modulatory parameter μ will multiplied by μ_{inc} and steps 3–8 will be repeated until μ exceeds μ_{max}. In such a case, LMD strategy will exit unsuccessfully.

5.3. RP strategy

Although LMD strategy can accelerate the convergence greatly, too fast descent may result in a local minimum or premature convergence. To solve this problem, RP (random perturbation) strategy is employed to retain the diversity and prevent the evolution from being trapped by some local minimums.

Similar to flip operation and LMD strategy, RP strategy works like a subroutine, which accepts three input parameters: the weight matrix $\boldsymbol{W}$, the magnitude of the noise ρ, and the goal of the performance function *perf* (it must be less than or equal to PE($\boldsymbol{W}$)).

The diagram of RP strategy is shown in Fig. 4.

If a randomly perturbed weight matrix $\boldsymbol{W}^u$ is found by which the network is constructed has a smaller performance function value than *perf* and meanwhile conforms to increasing monotonicity, then RP strategy will exit successfully. Otherwise, RP strategy will exit unsuccessfully.

5.4. IDEP algorithm

The diagram of IDEP algorithm is shown in Fig. 5.

In IDEP algorithm, the initialization in step 2 and mutation, cross-over and pruning operations in step 5

Table 4
Data for the effect of pressure on entropy for a crude oil[a]

No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
P_r	0.2	0.25	0.3	0.35	0.40	0.45	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95	1.00	1.50	2.00	2.50	3.00	3.50	4.00	4.50	5.00	5.50	6.00	6.50	0.00	7.50	8.00	8.50	9.00
S	0	0.23	0.29	0.33	0.40	0.46	0.51	0.60	0.64	0.71	0.79	0.84	0.93	1.10	1.32	1.71	2.55	3.22	3.30	3.37	3.45	3.51	3.55	3.60	3.61	3.63	3.66	3.67	3.68	3.69	3.70	3.70	3.70

[a] Reduced temperature $T_r = 1.00$.

are same as those in DEP algorithm. Within the loop from step 4 to 18, first an offspring $\boldsymbol{W}_i^{G+1}$ is produced (step 5), then flip operation produces a variant of $\boldsymbol{W}_i^{G+1}$ that satisfies the prior knowledge (step 7), LMD and RP strategies try to find a better individual than $\boldsymbol{W}_i^G$ in a local region by using a LM-like descent method (step 13) or through random perturbation on $\boldsymbol{W}_i^G$ (step 16), respectively. All candidates produced by these operations and strategies, as well as $\boldsymbol{W}_i^G$, will compete together according to their fitness and conformance of the prior knowledge. Only the one that not only has the smallest fitness but also satisfies the prior knowledge will survive.

Too frequent performance of LMD strategy may possibly consume a lot of time and lead to premature or convergence at local minimum. To avoid this, we select a random number R in interval [0,1) and perform LMD strategy only when $R > R_{\mathrm{lmd}}$, where R_{lmd} is a predefined number in interval (0,1). Thus, R_{lmd} is used to control the frequency of performing LMD strategy.

In RP strategy, ρ is used to control the magnitude of noise. At the early stage of the training process, a large value of ρ can help to escape local minimums and enlarge the search region. In addition, at the later stage, the algorithm appeal to a small value of ρ since a large one may decrease the convergence speed. Therefore, an adjustable ρ may work better than a constant one. In IDEP algorithm, ρ is initialized to an arbitrary constant ρ_0 (step 1) and decrease at a logarithmic speed (step 19), which is borrowed from the inhomogeneous simulated annealing algorithm. In experiments, we find such descent is relatively slow and smooth, which is helpful to reach a global minimum.

6. Applications in chemistry

6.1. Modeling the true boiling point curve of crude oil

Crude oil is a very complicated mixture mainly containing different kinds of hydrocarbons, organic sulfur, nitrogen and oxygen compounds and trace inorganic compounds. Each component has different boiling point. Usually, we plot the true boiling point curve of crude oil by taking the mass percentage p of the distilled component as the abscissa, and the distilled temperature t as the ordinate. The curve can reflect the composition of the distilled crude oil. Therefore, to build a model that takes p as the independent variable and t as the dependent variable is an important problem in petrochemical industry [25]. We can build nonparametric models on the sample data, such as spline curve and neural networks. Moreover, we also know that t is monotonically increasing in p over its domain, which is the prior knowledge of the model. Therefore, we must make sure that the models conform to the prior knowledge.

In this paper, IDEP algorithm written using Matlab 5.3 is applied to modeling the true boiling point curve of crude oil. The sample data of a certain kind of crude oil are listed in Table 1. Here the mass percentage p corresponds to x, and the distilled temperature t corresponds to y. In addition, in order to speed up the training process, t is normalized to $(t - \min\{t\})/(2(\max\{t\} - \min\{t\}))$ beforehand. Note that because of the possibility of the nonmonotonicity between the sample data 22 and 23, we will not extract them as the testing data in cross-validation. The neural network uses a 1-7-1 structure. The training parameters are: PE $= \mathrm{SSE} = \sum_i (y_i - \hat{y}(x_i))^2$, $goal = 1 \times 10^{-4}$ and $epochs = 100{,}000$. The parameters of IDEP algorithm are $N_{\mathrm{p}} = 100$, $C_{\mathrm{R}} = 0.9$, $F = 0.9$, $\rho_0 = 1$, $\mu_0 = 0.001$, $\mu_{\mathrm{inc}} = 10$, $\mu_{\max} = 1 \times 10^{10}$ and $R_{\mathrm{lmd}} = 0.05$.

The $\hat{y}(x)$–x curve of the first model trained by IDEP algorithm in cross-validation is illustrated in Fig. 6a. For comparison, the curve of the model trained by NC-LM algorithm, and the cubic spline interpolation curve proposed by Hu [26] are also plotted in Fig. 6a. In addition, the curves of the models trained by another three prior-knowledge-based methods, J.PF, IP and EW are plotted in Fig. 6b. The detailed performance data of the methods mentioned above, as well as the NC-CGF, NC-DET and DEP algorithms, are listed in Table 2. Here, the parameters for NC-DET and DEP algorithm are $N_{\mathrm{p}} = 100$, $C_{\mathrm{R}} = 0.9$, $F = 0.9$, which are same as those in IDEP algorithm. The parameters for NC-LM algorithm are $\mu_0 = 0.001$, $\mu_{\mathrm{inc}} = 10$, $\mu_{\max} = 1 \times 10^{10}$, also same as those in IDEP algorithm. The parameters for NC-CGF algorithm take the default values in Matlab [27]. The parameters for J.PF, IP and EW methods are same as those in Ref. [6].

From Table 2, we can see the model trained by IDEP algorithm exhibits very good approximation accuracy. More importantly, it also shows outstanding prediction ability. Both its $\overline{|r_e|}$ and δ_{r_e} are the smallest among the different methods, which implies that it can predict with high accuracy and stability. Moreover, we see IDEP does not produce any model that violates the prior knowledge in cross-validation. Now let us also come to analyze the other algorithms according to their simulation results: DEP algorithm does not predict well since it cannot approximate the sample data as desired; NC-DET and NC-CGF algorithms do not incorporate any prior knowledge constraints, and they are also inferior in approximation, therefore, their resulting models contain many nonmonotonic intervals as we suppose, and their approximation and prediction results are worse, even absurd, such as $\overline{|r_e|} = 35199.7$. J.PF, IP and EW methods also have some disadvantages compared with IDEP algorithm, and the reasons will be discussed in detail in Section 7.

On the other hand, in order to compare the speed and search ability of different algorithms, we employ these algorithm (IDEP algorithm, as well as NC-DET, DEP, NC-CGF and NC-LM algorithm) to build the first model in cross-validation for 10 times with different random seeds. In addition, for each algorithm, the first one of the 10 evolution processes is plotted Fig. 7, which describes the relationship between the performance and the evolution epoch. Table 3 lists the detailed performance data of these algorithms, including the average number of generations or epochs for 10 times (denoted by $\overline{\text{Gen}}$), the mean of the smallest fitness in the last generation (denoted by $\overline{\text{PE}_\text{f}}$), and the ratio of the times that the training goal is achieved to the total training times (denoted by C_g/C_t, where $C_t = 10$). From these data, we can see that IDEP algorithm has achieved the training goal all the 10 times. However, NC-DET, DEP and NC-CGF, have never achieved it. NC-CGF is early-stopped because it reaches the minimum step size. Moreover, to obtain the same approximation accuracy, IDEP algo-

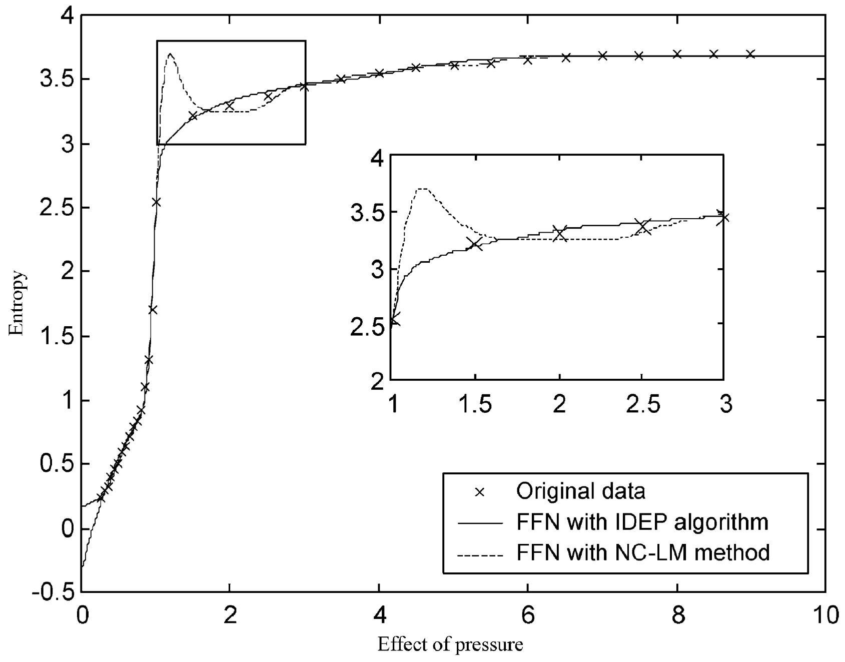

Fig. 8. Comparison between IDEP algorithm and other methods in modeling pressure effect on entropy of crude oil.

rithm requires fewer epochs than NC-DET, DEP and NC-CGF algorithm, but more epochs than NC-LM algorithm. Note that each epoch of EAs may consume more time than non-EA algorithms, and the actual training time depends on many factors, which we will not discuss here.

6.2. Modeling the curve of effect of pressure on entropy

Other than the true boiling point curve, we also verify the methods by modeling the curve of effect of pressure on entropy. The sample data [28] for the effect of pressure P_r on entropy S (reduced temperature $T_r = 1.00$) of a kind of crude oil are listed in Table 4. From the curve in Fig. 7 H1.5 [28], we can see that the effect of pressure on entropy is monotonically increasing over its domain, which is the prior knowledge of the model. Simulation network models can replace the curve. However, the model trained by NC method will be nonmonotonic at some intervals. Here, IDEP algorithm is applied to overcome the problem. In the simulation, P_r and S are normalized to $(P_r)/(10)$ and $(S - \min\{S\})/(2(\max\{S\} - \min\{S\}))$ before further process. The sample data 17, 18 and 19 are not extracted as the testing data in cross-validation because of the possibility of the nonmonotonicity among them. The structure of the neural network is 1-7-1 and *t*he training parameters are: PE $=$ SSE $= \sum_i (y_i - \hat{y}(x_i))^2$, $goal = 1 \times 10^{-3}$ and $epochs = 100{,}000$, all other parameters are same as those in modeling the true boiling point curve. The simulation curves are plotted in Fig. 8, and the detailed performance data are listed in Table 5, from which we can see IDEP outperforms all other methods again.

7. Analysis and conclusion

7.1. Discussion of IDEP algorithm

The key parts of IDEP algorithm are flip operation, LMD and RP strategy. We will discuss them respectively in this section.

The main purpose of flip operation is to adjust the weights of a network so as to conform the prior knowledge constraints. In addition, it can generate a new valid network to increase the diversity of the population so that it can prevent the premature during the evolution.

In LMD strategy, μ_0 and $\mu_{\max}$ define the search range of LM algorithm, and μ_{inc} defines the increasing speed of μ during the search. Usually, a broad range and a small increasing speed mean a relatively thorough search, but it will consume more time. On the contrary, a narrow range and a large speed lead to a fast but inexact search, which will probably miss some feasible solutions. Therefore, when using LMD strategy, a compromise between consumed time and search accuracy must be considered. Usually the above parameters are selected according to experience. However, a set of adaptive parameters may work better. We will discuss this in other papers.

The control parameters of RP strategy are the initial magnitude of noise ρ_0 and its decreasing strategy. Generally a large ρ_0 and slow decreasing speed can

Table 5
Performance of the different prior-knowledge-based methods in modeling the effect of pressure on entropy for a crude oil

Method	Mean of approximation accuracy ($\overline{\text{MSE}}$)	Mean of relative prediction error ($\overline{\lvert r_e \rvert}$)	S.D. of relative prediction error (δ_{r_e})	Number of nonmonotonic models/total models in cross-validation	Nonmonotonic interval of the first model in cross-validation
NC-LM	2.9577×10^{-5}	0.1062	0.3042	20/30	[1.20, 1.72]
NC-CGF	1.9952×10^{-4}	0.0754	0.1472	30/30	[1.57, 2.05]
NC-DET	3.1147×10^{-4}	0.3381	1.2574	25/30	[0.001, 0.029]
IDEP	2.9483×10^{-5}	0.0465	0.1012	0/30	None
DEP	1.0135×10^{-3}	0.1866	0.5006	0/30	None
J.PF	5.4455×10^{-4}	0.1298	0.3409	14/30	None
IP	2.6481×10^{-5}	0.0667	0.1907	10/30	[1.08, 1.18]
EW	3.0292×10^{-5}	0.0738	0.1861	0/30	None

increase the probability to obtain more feasible solutions, but will also increase the evolution time. Accordingly, a small ρ_0 and fast decreasing speed will work contrarily. This is also a compromise, which can be solved by experience or adaptive methods.

7.2. IDEP as a prior-knowledge-based algorithm

IDEP is a kind of WC methods according to the classification in Refs. [3,6]. It restricts the weight matrixes by excluding all the individuals that violate the prior knowledge during the evolution, so the final solution will strictly conform to the prior knowledge. On the contrary, almost all the DC methods cannot guarantee to satisfy the prior knowledge, e.g. IP method. Neither are some of the WC methods, e.g. J.PF method. As to the AC methods, although they inherently conform to the prior knowledge, usually they demand some skillful design.

EW method introduces the exponential computation based on a strengthened condition, which makes the approximation difficult. Therefore, the approximation and prediction ability of EW are worse than those of IDEP algorithm.

Now let us compare IDEP with J.PF method. Both J.PF method and IDEP algorithm are based on optimization methods. J.PF employs the exterior penalty function method, which will induce networks to satisfy the prior knowledge since those violating the prior knowledge will be penalized. However, it still cannot ensure that the resulting models will completely satisfy the prior knowledge. Moreover, the introduction of the penalty function will also affect the approximation accuracy. That is the reason why the approximation accuracy of J.PF is not very good. On the other hand, IDEP algorithm is based on EAs, which operates on a population of weight matrixes to perform optimization. Networks that violate the prior knowledge will not be penalized like J.PF does, but simply be rejected once found. Such a strategy is feasible because there are so many individuals in population that the rejection of several individuals will not have a significant impact on the solutions as a whole. In addition, because no penalty function is used during the training process, the approximation accuracy of IDEP algorithm remains high.

In this paper, we only discuss the prior knowledge of increasing monotonicity. However, it is easy to generalize it to the decreasing monotonicity and other kinds of prior knowledge by modifying the flip operation accordingly.

7.3. IDEP as a training algorithm

By employing LMD strategy to speed up the local search, the converge speed of IDEP algorithm is much faster than NC-DET, DEP, and NC-CGF algorithms. But due to the prior knowledge constraint, it is still slower than NC-LM algorithm.

During the population evolution, flip operation, LMD and RP strategy can increase the probability of introducing new individuals into the population and help algorithm to find a global or satisfactory minimum, which means IDEP algorithm possesses a good approximation accuracy. On the contrary, NC-DET and DEP algorithm is prone to premature, and NC-CGF algorithm will often get trapped in a local minimum.

Therefore, IDEP algorithm is a fast, high efficiency, and robust algorithm for network training. In addition, by slightly modifying IDEP algorithm, it can be generalized to a wide range of network training problems besides the prior knowledge constraint.

8. Conclusion and future investigations

This paper proposes an IDEP algorithm and discusses its application in modeling chemical curves with the increasing monotonicity constraint. IDEP algorithm works very well in encoding the prior knowledge into networks during network training. However, many further works remain to be done along the following research directions:

• The methodology of encoding any kind of prior knowledge into networks by modifying the current IDEP algorithm.

• The strategy of the parameter selection and adjustment in IDEP algorithm, such as an adaptive strategy:

- Accurate theoretical analysis for the astringency and complexity of IDEP algorithm.
- The generalization from IDEP algorithm to a common training algorithm for feedforward networks and other kinds of neural networks.

Acknowledgements

This project was supported by the National Natural Science Foundation (20076041) of China.

References

[1] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.

[2] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Networks 3 (1990) 551–560.

[3] W.H. Joerding, J.L. Meador, Encoding a priori information in feedforward networks, Neural Networks 4 (1991) 847.

[4] M.L. Thompson, M.A. Kramer, Modeling chemical processes using prior knowledge and neural networks, AIChE Journal 40 (8) (1994) 1328–1340.

[5] C.-W. Chen, D.-Z. Chen, X.Q. Ye, S.-X. Wu, Feedforward networks based on prior knowledge and its application in modeling the true boiling point curve of the crude oil, Journal of Chemical Engineering of Chinese Universities 15 (4) (2001) 351–356.

[6] C.-W. Chen, D.-Z. Chen, S.-X. Wu, Prior-knowledge-based feedforward network simulation of true boiling point curve of crude oil, Computers and Chemistry 25 (2001) 541–550.

[7] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, New York, 1996.

[8] D. Whitley, T. Starkweather, C. Bogart, Genetic algorithms and neural networks: optimizing connections and connectivity, Parallel Computing 14 (3) (1990) 347–361.

[9] R.K. Belew, J. McInerney, N.N. Schraudolph, Evolving networks: using genetic algorithm with connectionist learning. Computer Science and Engineering Department (C-014), University of California, San Diego, Technical Report CS90-174 (revised), 1991.

[10] X. Yao, Evolving artificial neural networks, Proceedings of the IEEE Special Issue on Computational Intelligence, 1999, pp. 1423–1447.

[11] D.E. Goldberg, Genetic Algorithm in Search. Optimization and Machine Learning, Addison-Wesley Publishing, New York, USA, 1989.

[12] R. Storn, K. Price, Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, 1995.

[13] L. Jouni, L. Miika, http://www.it.lut.fi/project/nngenetic/index, 2000.

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

[15] M.T. Hagan, M. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Transactions on Neural Networks 5 (6) (1994) 989–993.

[16] R. Fletcher, C.M. Reeves, Function minimization by conjugate gradients, Computer Journal 7 (1964) 149–154.

[17] H.A. Martens, P. Dardenne, Validation and verification of regression in small data sets, Chemometrics and Intelligent Laboratory Systems 44 (1998) 99–121.

[18] H.A. Martens, T. Naes, Multivariate Calibration, Wiley, Chichester, UK, 1989.

[19] D. Nguyen, B. Widrow, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, Proceedings of the International Joint Conference on Neural Networks 3, 1990, pp. 21–26.

[20] R. Storn, K.V. Price, Minimizing the real functions of the ICEC '96 contest by differential evolution, IEEE Conference on Evolutionary Computation, 1996, pp. 842–844, Nagoya.

[21] J.E. Dennis, R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[22] F.S. Wang, J.P. Chiou, Optimal control and optimal time location problems of differential–algebraic systems by differential evolution, Industrial and Engineering Chemistry Research 36 (1997) 5348–5357.

[23] F.S. Wang, J.P. Chiou, Hybrid method of evolutionary algorithms for static and dynamic optimization problems with application to a fed-batch fermentation process, Computers and Chemical Engineering 23 (1999) 1277–1291.

[24] M. Avriel, Nonlinear Programming: Analysis and Methods, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[25] S.-X. Hu, J.-N. Tang, Petroleum Processing 18 (2) (1987) 1–10.

[26] S.-X. Hu, Study on the vapor–liquid equilibrium calculation of the distilled component of crude oil using spline function, Journal of East China Petroleum Institute 1 (1983) 1–18.

[27] H. Demuth, M. Beale, Neural Network Toolbox: User's Guide, MathWorks, USA, 1998.

[28] American Pertroleum Institute, Technical Data Book—Petroleum Refining, Port City Press, Washington, DC, 1970, pp. 7–198.