Information Sciences 264 (2014) 104–117

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Fast decorrelated neural network ensembles with random weights

Monther Alhamdoosh, Dianhui Wang *

Department of Computer Science and Computer Engineering, La Trobe University, Melbourne, VIC 3086, Australia

ARTICLE INFO

Article history:
Received 6 November 2013
Received in revised form 27 November 2013
Accepted 20 December 2013
Available online 31 December 2013

Keywords:
Negative correlation learning
Neural network ensembles
Random vector function link networks
Data regression

ABSTRACT

Negative correlation learning (NCL) aims to produce ensembles with sound generalization capability through controlling the disagreement among base learners' outputs. Such a learning scheme is usually implemented by using feed-forward neural networks with error back-propagation algorithms (BPNNs). However, it suffers from slow convergence, local minima problem and model uncertainties caused by the initial weights and the setting of learning parameters. To achieve a better solution, this paper employs the random vector functional link (RVFL) networks as base components, and incorporates with the NCL strategy for building neural network ensembles. The basis functions of the base models are generated randomly and the parameters of the RVFL networks can be determined by solving a linear equation system. An analytical solution is derived for these parameters, where a cost function defined for NCL and the well-known least squares method are used. To examine the merits of our proposed algorithm, a comparative study is carried out with nine benchmark datasets. Results indicate that our approach outperforms other ensembling techniques on the testing datasets in terms of both effectiveness and efficiency.

1. Introduction

In the last two decades, ensemble learning framework has received considerable attention and resulted in many novel machine learning techniques, for instance, bagging, boosting and random forests [1–4]. The base models of an ensemble can be trained individually or collectively. Ensemble methods also differ in the way that they handle the training data during the learning phase. For example, simple averaging ensembles [5,6] use whole training data to learn base models, but with different parameter settings for each, while bagging, boosting and random forests use different parts of the training dataset for each base model to gain more diversity among ensemble components. Bagging randomly resamples replicates from a given training dataset [1]; while Boosting sequentially resamples the training dataset based on mis-classification probability distribution calculated from previous learnt models [2,3]. Random forests, however, create the training data variability based on the input features rather than the training examples. It selects the splitting features for each node in its base decision trees from a random set of input features [4]. On the other hand, the merging weights of base models can be set equally [7], or learnt by using heuristic means or minimizing a cost function [8,9]. Note that boosting ensembles adapt their averaging weights during the course of training weak learners [3]. It is worthy mentioned that ensemble learning is essential to learn complex domain problems that have complex nonlinear relationships among input variables and output variables. In such cases, the hypotheses space of base models is very large and one single hypothesis cannot model the underlying data distribution.

* Corresponding author. Tel.: +61 3 9479 3034; fax: +61 3 9479 3060.
E-mail addresses: mal-hamdoosh@students.latrobe.edu.au (M. Alhamdoosh), dh.wang@latrobe.edu.au (D. Wang).

0020-0255/$ - see front matter Crown Copyright © 2013 Published by Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.12.016

An important issue in ensemble learning algorithms is how to maintain no duplication of base models, i.e., modeling different regions of the features space by different base models without deteriorating their individual accuracies. Meanwhile, the overall performance of ensemble should be maintained [7]. This is called ensemble diversity in literature [7,10]. Evolutionary approaches are widely used to select ensemble component networks with maximum disagreement among their outputs [8,11–13]. However, such evolutionary-based approaches take long time to converge. In [7] the ensemble base models were trained by using cross-validation datasets and a diversity metric was introduced to trade-off the bias-variance–covariance. The bias-variance–covariance decomposition raised the attention to find ensemble models with minimum covariance among the base models so that the individual model performance can be retained [10].

Negative correlation learning (NCL) [14] amends the cost function with a penalty term that weakens the relationship with other individuals and controls the trade-off among the bias, variance and covariance in the ensemble learning [10]. Later, this idea was extended in [15]. It has been noticed that the proposed approach in [13] can automatically determine an optimal size of ensemble by thoroughly exploring the ensemble hypotheses space using NCL. Recently, a regularized version of the negative correlation learning techniques was proposed in [16], aiming at reducing the overfitting risk for noisy data.

In addition to the diversity concern in ensemble learning, the way of generating the component models is crucial. Neural networks with back-propagation learning algorithms are mostly employed for this purpose. Usually, single-layer feed-forward (SLFN) networks are sufficient to problem solving due to its universal approximation capability [17–19]. Unfortunately, the gradient-based learning algorithms for training SLFNs suffer from local minima problem, slow convergence and very poor sensitivity to learning rate setting. To overcome these difficulties, random vector functional-link (RVFL) networks were proposed [20–22], where the weights between the input layer and the hidden layer can be randomly assigned and no need to be tuned. Then, the well-known least square methods can be used to calculate the output weights [23]. This flat-net architecture universally approximates any continuous function and dramatically reduces the training time [22,24]. Our recent work reported in [12] shows that RVFL-based ensembles take some advantages comparing against other existing ensembling methods. It is interesting to see that in most of the cases the resulting ensemble is composed of few component models (4–12). Note that this finding is obtained by selecting the best RVFL candidates from a pool. Because the best candidates selection is implemented by using genetic algorithms, it is a time consuming process to achieve the final ensemble model. Indeed, this motivates us to do further studies in this direction.

Based on our previous studies, this paper aims to develop a fast solution on building neural network ensembles. Our proposed algorithm (termed as DNNE) randomly initializes the hidden layer parameters of base RVFL networks, and then employs the least square method with negative correlation learning scheme to analytically calculate the output weights of these base networks. A minimum norm least square solution is derived and formulated in a matrix form for computational exercises. A comparative study on data regression is carried out. Results over the testing datasets are promising, and supporting a positive statement on performance assessment among bagging, boosting, simple ensembles and random forests.

The remainder of this paper is organized as follows. Basics on RVFL networks, ensemble learners and the negative correlation learning are reviewed in Section 2. Our solution with a detailed description on the proposed learning algorithm is given in Section 3. To evaluate the performance of our approach, some benchmark datasets are employed in this study and results with comparisons and discussions are presented in Section 4. Finally, we conclude this work in the last section.

2. Background

Learning from data is a process of finding a hypothesis $f(x;\theta)$ that estimates an unknown target function $\psi(x)$, where θ is the model parameters to be tuned.The learnt hypothesis can be single or composite model (i.e., ensemble of base hypotheses). Only few works reported the use of random weights in ensemble base networks [10,20] while it has been widely investigated in single neural networks [21,22,25,26]. The theoretical foundation of randomness in neural networks can be deeply understood from the function approximation task with Monte-Carlo (MC) methods. It has been shown in [22] that any continuous function defined on a compact set can be represented by a limit-integral of multivariate continuous function with integration in the parameters space. MC method approximates this multiple integral by drawing random samples of the parameter vector from uniform distribution defined over the limit-integral domain. More likely, the Monte-Carlo estimated accuracy proportionally improves along with an increase of the number of random samples and the approximation error tends to zero as this number goes to infinity. A special case of this general approximation theory becomes the random vector functional link (RVFL) networks, which can be represented as single-layer feed-forward networks (SLFNs). The multivariate continuous functions in the MC method play a role as the activation functions of hidden neurons in the SLFN, and the input weights and the hidden layer biases of the SLFN correspond to the sampled parameters vector in the MC method. Eventually, the output weights of the SLFN function as the estimated parameters in the MC method.

2.1. Review of random basis function approximators

It has been proved that RVFL networks are universal approximators for continuous functions on compact sets and its approximation error converges to zero with order $O\left(C/\sqrt{L}\right)$, where L is the number of basis functions (hidden neurons) and C is a constant [22,26]. An RVFL network can be defined as a SLFN model,

$$f(x;\beta) = \sum_{k=1}^{L} \beta_k G(w_k^T \cdot x + b_k), \tag{1}$$

where $\beta = [\beta_1, \beta_2, \ldots, \beta_L] \in \mathrm{R}^L$ is a parameter vector (the output layer weights); $x \in \mathrm{R}^d$ is the input features vector; $w_k \in \mathrm{R}^d$ and $b_k \in \mathrm{R}$ are nonlinear parameters (input weights and hidden layer biases); d is the number of network inputs; and $G(\cdot)$ is a basis function.

In RVFL network, the parameters of the hidden layer (w_k and b_k) are assigned randomly and independently from the training data; while the linear parameters β_k of the output layer can be tuned using quadratic optimization techniques [22].

Building an RVFL network model requires a training dataset of size N, denoted by $\mathcal{D}_t = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $(x_n, y_n) \in \mathrm{R}^d \times \mathrm{R}$ are pairs of observations. The pairs (x_i, y_i) are i.i.d. samples drawn from an unknown joint probability distribution $p(x, y)$. This implies that $\mathcal{D}_t$ is a realization of a random sequence $D_N = \{Z_1, Z_2, \ldots, Z_N\}$ whose elements are random vectors $Z_n = (X_n, Y_n)$ for $n = 1, \ldots, N$. Now, the mean squares learning error of an RVFL network model f over $\mathcal{D}_t$ can be defined as

$$E_f(\mathcal{D}_t) = \frac{1}{N}\sum_{n=1}^{N} (f(x_n; \beta(\mathcal{D}_t)) - y_n)^2. \tag{2}$$

We write $\beta(\mathcal{D}_t)$ to clarify that the values of the parameter vector β, and thus the hypothesis f, are learnt on the training dataset $\mathcal{D}_t$. Therefore, we will write $f(x; \mathcal{D}_t)$ instead of $f(x_n; \beta(\mathcal{D}_t))$. Commonly, the parameters vector may be estimated through minimizing the following cost function E_f, i.e.,

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \frac{1}{N}\sum_{n=1}^{N} \frac{1}{2} (f(x_n; \mathcal{D}_t) - y_n)^2. \tag{3}$$

Let $\mathcal{D}_v = \{(\acute{x}_1, \acute{y}_1), (\acute{x}_2, \acute{y}_2), \ldots, (\acute{x}_{\acute{N}}, \acute{y}_{\acute{N}})\}$ be a validation dataset of size $\acute{N}$, whose elements are distributed identically but independently of Z_n for all training examples. Then, the generalization error $E_f(\mathcal{D}_v)$ of the RVFL network model f can be defined as the mean squares error (MSE) averaged over all possible realizations of D_N, that is,

$$E_f(\mathcal{D}_v) = E_{D_N}\left\{ \frac{1}{\acute{N}} \sum_{n=1}^{\acute{N}} (f(\acute{x}_n; D_N) - \acute{y}_n)^2 \right\}, \tag{4}$$

where $E_{D_N}\{\cdot\}$ is the expectation with respect to the distribution of the random sequence D_N.

Theorem 1. *The generalization error, given in Eq. (4), of an RVFL learner f can be expressed using the bias/variance dilemma as follows*

$$E_f(\mathcal{D}_v) = \frac{1}{\acute{N}} \sum_{n=1}^{\acute{N}} \left[\overline{Var}(f, \acute{x}_n) + \overline{Bias}(f, \acute{x}_n)^2 \right], \tag{5}$$

where

$$\begin{aligned} \overline{Var}(f, \acute{x}_n) &= E_{D_N}\left\{ \left(f(\acute{x}_n; D_N) - E_{D_N}\{f(\acute{x}_n; D_N)\} \right)^2 \right\}, \\ \overline{Bias}(f, \acute{x}_n) &= E_{D_N}\{f(\acute{x}_n; D_N)\} - \acute{y}_n. \end{aligned} \tag{6}$$

The proof of Theorem 1 can be found in [27]. If a learner model is unbiased, it does not imply a good generalization because it may suffer from a large variance and vice versa. Actually, a balanced trade-off between the bias and variance is required in order to obtain best performance. Interestingly, the bias and variance of a learner can be realized even when a fixed training dataset is used, but with different parameters initializations, as in RVFL networks.

Recently, the feasibility of RVFL networks in learning from data was investigated in [26], and some issues should be carefully paid attention in practice. For instance, the number of basis elements should be sufficiently large and supervised initialization is needed in order to model and compensate system uncertainties. However, these issues can be relaxed when a collection of RVFL networks are combined using Pincus formula [28] and Monte-Carlo approximation procedure so that it results in a model with less variance [20,22]. Alternatively, the problems mentioned in [26] can be overcome by combining multiple RVFL networks trained using negative correlation learning [14].

2.2. Review of negative correlation learning

Given an ensemble of M base models and a training dataset $\mathcal{D}_t$ of N instances, then the output of the ith network on the nth data example is designated by $f_i(x_n; \mathcal{D}_t) \in \mathrm{R}$ and the ensemble collective output is given by

$$\bar{f}(x_n; \mathcal{D}_t) = \sum_{i=1}^{M} \alpha_i f_i(x_n; \mathcal{D}_t), \tag{7}$$

where α_i is the averaging weight of the ith component network and reflects the contribution of this component in the final ensemble decision. In other words, the averaging weights must satisfy $\sum_{i=1}^{M}\alpha_i = 1$ and $0 \leqslant \alpha_i \leqslant 1$. For simplicity, however, uniform averaging weights are adopted in our study i.e., $\alpha_i = \frac{1}{M},\ i = 1,\ldots,M$.

The learning errors of the ensemble base models and the ensemble model can be defined by Eq. (2) by considering the base models independently and the ensemble model as a whole single model, respectively, as follows

$$E_i = \sum_{n=1}^{N}\frac{1}{2}(f_i(x_n) - y_n)^2, \quad i = 1,\ldots,M, \tag{8}$$

$$E_{ens} = \frac{1}{N}\sum_{n=1}^{N}(\bar{f}(x_n) - y_n)^2. \tag{9}$$

Using Eqs. (4) and (7), the generalization error of the ensemble model $\bar{f}$, averaged over all possible training datasets $\mathcal{D}_t$ of size N, is written as follows

$$E_{\bar{f}}(\mathcal{D}_v) = E_{D_N}\left\{\frac{1}{\acute{N}}\sum_{n=1}^{\acute{N}}(\bar{f}(\acute{x}_n; D_N) - \acute{y}_n)^2\right\}, \tag{10}$$

where $\bar{f}(\acute{x}_n; D_N) = \frac{1}{M}\sum_{i=1}^{M} f_i(\acute{x}_n; D_N)$ is the ensemble output given an input $\acute{x}_n$ and learnt on a realization of D_N.

In traditional ensemble learning algorithms, the base models can be trained independently in a parallel way as adopted in Bagging [1] or sequentially as used in Boosting [2,3]. However, such learning algorithms focus only on minimizing the individual learning errors given in Eq. (8). In order to deeply understand the ensemble generalization abilities, Theorem 2 is recalled from [29] and formulated as follows.

Theorem 2. *The generalization error of an ensemble, given in Eq. (10), in which all components are trained using the same dataset can be expressed using the following bias/variance/covariance decomposition*

$$E_{\bar{f}}(\mathcal{D}_v) = \frac{1}{\acute{N}}\sum_{n=1}^{\acute{N}}\left[\frac{1}{M}\overline{Var}(\acute{x}_n) + \left(\frac{M-1}{M}\right)\overline{Cov}(\acute{x}_n) + \overline{Bias}(\acute{x}_n)^2\right] \tag{11}$$

where

$$\overline{Var}(\acute{x}_n) = \frac{1}{M}\sum_{i=1}^{M}E_{D_N}\left\{(f_i - E_{D_N}\{f_i\})^2\right\},$$

$$\overline{Cov}(\acute{x}_n) = \frac{1}{M(M-1)}\sum_{i=1}^{M}\sum_{j\neq i}E_{D_N}\{(f_i - E_{D_N}\{f_i\})(f_j - E_{D_N}\{f_j\})\},$$

$$\overline{Bias}(\acute{x}_n) = \frac{1}{M}\sum_{i=1}^{M}(E_{D_N}\{f_i\} - \acute{y}_n).$$

Notice that $f, f(x_n; D_N)$ and $f(x_n)$ are used interchangeably to denote the output of the model f, trained on a realization of D_N, given an input observation x_n. $\overline{Var}(\acute{x}_n)$ and $\overline{Bias}(\acute{x}_n)$ are actually the average individual variances and biases, respectively. The proof of Theorem 2 can be read in [29]. In fact, managing the covariance term $\overline{Cov}(\acute{x}_n)$ explicitly helps in controlling the disagreement among ensemble components' outputs and hence producing better generalized ensemble model. Some details can be found in [10].

Negative correlation learning (NCL) was proposed to reduce the covariance among ensemble individuals while the variance and bias terms are not increased. Unlike traditional ensemble learning approaches, NCL was introduced to train base models simultaneously in a cooperative manner that decorrelates individual errors E_i [14,30]. Mathematically, the learning error of the ith base model, given in Eq. (8), was modified to include a decorrelation penalty term p_i as follows

$$e_i = \sum_{n=1}^{N}\left[\frac{1}{2}(f_i(x_n) - y_n)^2 + \lambda p_i(x_n)\right], \tag{12}$$

where $\lambda \in [0, 1]$ is a regularizing factor. The penalty term p_i can be designed in different ways depending on whether the ensemble networks are trained sequentially or parallelly. For instance, it could decorrelate the current learning network with all previously learned networks [14]

$$p_i(x_n) = (f_i(x_n) - y_n)\sum_{j=1}^{i-1}(f_j(x_n) - y_n). \tag{13}$$

The penalty term in Eq. (12) can be designed in a way that preserves decorrelation between pairs of networks yet allows alternate networks to be trained independently [14]. In [30], a new penalty term was proposed and formulated as follows

$$p_i(x_n) = (f_i(x_n) - \bar{f}(x_n)) \sum_{j \neq i} (f_j(x_n) - \bar{f}(x_n)). \tag{14}$$

Notice that the penalty term in Eq. (14) reduces the correlation mutually among all ensemble individuals by using the actual ensemble output $\bar{f}(x_n)$ instead of the target function y_n. The work in [10] thoroughly analyzed the characteristics of NCL technique and showed that it really works for multilayer perceptron (MLP) and radial basis function (RBF) networks. More explanation on why negative correlation learning does work for building good ensembles can be found in [10].

3. Decorrelated neural-net ensembles with random weights

The assurance of well-balanced trade-off between the bias and variance of single RVFL learner is a difficult task due to the uncertainties in the learning process that are caused by the random initializations of basis functions. Different initial settings for the RVFL networks lead to different solutions. In order to reduce the impact of these factors on the generalization error of a learning system, a cluster of RVFL networks are combined together to produce efficient predictions [5,22]. In this paper, we propose a new ensemble learning approach that uses RVFL networks as ensemble components and it is fitted in negative correlation learning framework. Since RVFL networks do not require learning all their parameters (basis functions are set randomly), we seek a simple and fast solution to calculate the output weights of the base RVFL networks. This solution should take into account the correlation among the base RVFL networks in the output space and make sure it is at minimum. Although DNNE aims at encouraging the diversity among ensemble components by reducing the correlation among their outputs, it still maintains an overall good ensemble accuracy. Next, we present our method in details.

Given an ensemble of RVFL networks and a training dataset $\mathcal{D}_t$ of N instances, by substituting the penalty term p_i in Eq. (12) with (14), the decorrelated error of the ith individual is expressed as follows

$$e_i = \sum_{n=1}^{N} e_i(x_n) = \sum_{n=1}^{N} \left[\frac{1}{2} (f_i(x_n) - y_n)^2 - \lambda (f_i(x_n) - \bar{f}(x_n))^2 \right]. \tag{15}$$

Note that we use uniform averaging weights to combine ensemble base RVFL networks i.e.,

$$\bar{f}(x_n) = \frac{1}{M} \sum_{i=1}^{M} f_i(x_n) \quad \text{and} \quad \sum_{j \neq i} (f_j(x_n) - \bar{f}(x_n)) = -(f_i(x_n) - \bar{f}(x_n)). \tag{16}$$

Here, we relax the assumption that the ensemble output $\bar{f}(x_n)$ is constant. Moreover, since RVFL networks are used to populate our ensemble, the output of the ith base network, stimulated with an instance x_n, is given by

$$f_i(x_n) = \sum_{j=1}^{L} \beta_{ij} g_{ij}(x_n), \tag{17}$$

where L is the number of basis functions (hidden neurons) in the ith individual RVFL network; β_{ij} is the output weight connecting the jth hidden neuron with the output neuron in the ith base model; $g_{ij}(x_n) = G_{ij}(w_j, b_j, x_n)$ is the output of the jth hidden neuron in the ith base model; and G can be any squashing basis function.

In this study, we assume homogeneous hidden nodes for all base networks, i.e., one type of squashing functions is used for all hidden neurons. As mentioned earlier, the parameters (w_j, b_j) of the basis functions g_{ij} are randomly set while the only parameters to be tuned are the output weights β_{ij}. Therefore, NCL ensemble models attain an optimal performance when the gradient of the error function in Eq. (15) vanishes with respect to the output weights β_{ij}, that is,

$$\begin{aligned} &\bigtriangledown e_i = 0, \quad \text{for} \quad i = 1, \ldots, M; \quad \text{which leads to} \\ &\frac{\partial e_i}{\partial \beta_{ij}} = \sum_{n=1}^{N} \frac{\partial e_i(x_n)}{\partial \beta_{ij}} = 0, \quad \text{for} \quad j = 1, \ldots, L. \end{aligned} \tag{18}$$

We assume here that all base networks have similar architecture and the same dataset is used to train all of them.

Using calculus, we have

$$\begin{aligned} \frac{\partial e_i(x_n)}{\partial \beta_{ij}} &= \frac{\partial}{\partial \beta_{ij}} \left[\frac{1}{2} (f_i(x_n) - y_n)^2 - \lambda (f_i(x_n) - \bar{f}(x_n))^2 \right] = (f_i(x_n) - y_n) g_{ij}(x_n) - 2\lambda (f_i(x_n) - \bar{f}(x_n)) \left(1 - \frac{1}{M}\right) g_{ij}(x_n) \\ &= g_{ij}(x_n)[(f_i(x_n) - y_n) - \gamma (f_i(x_n) - \bar{f}(x_n))], \end{aligned} \tag{19}$$

where

$$\gamma = 2\lambda \left(\frac{M-1}{M} \right). \tag{20}$$

Substituting Eq. (16) in Eq. (19), we get

$$\frac{\partial e_i(x_n)}{\partial \beta_{ij}} = g_{ij}(x_n) \left[\left(1 - \gamma + \frac{\gamma}{M}\right) f_i(x_n) + \frac{\gamma}{M} \sum_{\substack{l \neq i \\ l=1}}^{M} f_l(x_n) - y_n \right]. \tag{21}$$

Using Eq. (17), Eq. (21) can be rewritten as

$$\frac{\partial e_i(x_n)}{\partial \beta_{ij}} = g_{ij}(x_n)\left[\left(1-\gamma+\frac{\gamma}{M}\right)\sum_{k=1}^{L}\beta_{ik}g_{ik}(x_n)+\frac{\gamma}{M}\sum_{\substack{l\neq i\\ l=1}}^{M}\sum_{k=1}^{L}\beta_{lk}g_{lk}(x_n)-y_n\right]. \tag{22}$$

Rearranging Eq. (22) gives

$$\frac{\partial e_i(x_n)}{\partial \beta_{ij}} = \left(1-\gamma+\frac{\gamma}{M}\right)\sum_{k=1}^{L}\beta_{ik}g_{ij}(x_n)g_{ik}(x_n)+\frac{\gamma}{M}\sum_{\substack{l\neq i\\ l=1}}^{M}\sum_{k=1}^{L}\beta_{lk}g_{ij}(x_n)g_{lk}(x_n)-g_{ij}(x_n)y_n. \tag{23}$$

From Eqs. (15) and (22), we have

$$\frac{\partial e_i}{\partial \beta_{ij}} = \left(1-\gamma+\frac{\gamma}{M}\right)\sum_{k=1}^{L}\beta_{ik}\sum_{n=1}^{N}g_{ij}(x_n)g_{ik}(x_n)+\frac{\gamma}{M}\sum_{\substack{l\neq i\\ l=1}}^{M}\sum_{k=1}^{L}\beta_{lk}\sum_{n=1}^{N}g_{ij}(x_n)g_{lk}(x_n)-\sum_{n=1}^{N}g_{ij}(x_n)y_n. \tag{24}$$

Finally, by Eqs. (18) and (24), we obtain

$$\sum_{k=1}^{L}C_1\varphi(i,j,i,k)\beta_{ik}+\sum_{\substack{l\neq i\\ l=1}}^{M}\sum_{k=1}^{L}C_2\varphi(i,j,l,k)\beta_{lk}=\varphi(i,j), \tag{25}$$

where $i=1,\ldots,M;\ j=1,\ldots,L$ and

$$C_1 = 1-\gamma+\frac{\gamma}{M},\quad C_2=\frac{\gamma}{M}, \tag{26}$$

$$\varphi(i,j,k,l)=\sum_{n=1}^{N}g_{ij}(x_n)g_{kl}(x_n),\quad \varphi(i,j)=\sum_{n=1}^{N}g_{ij}(x_n)y_n. \tag{27}$$

Here, C_1 and C_2 are two constants; $\varphi(i,j,k,l)$ represents the correlation between the jth hidden neuron of the ith individual RVFL network and the lth hidden neuron of the kth individual RVFL network; and $\varphi(i,j)$ models the correlation between the jth hidden neuron of the ith base network and the target function $\psi(x)$. By Eqs. (20) and (26), the two constants C_1 and C_2 can be expressed as follows

$$C_1 = 1-2\lambda\frac{(M-1)^2}{M^2},\quad C_2=2\lambda\frac{M-1}{M^2}. \tag{28}$$

Applying Eq. (25) to all individual errors e_i and all output weights β_{ij} ($i=1,\ldots,M;\ j=1,\ldots,L$), a linear system of $M\times L$ equations is derived. Therefore, solving this linear system with respect to β_{ij} yields an RVFL ensemble model. To facilitate this computational task, we write this linear system in a matrix form as follows

$$\mathrm{H}_{\mathrm{corr}}\mathrm{B}_{\mathrm{ens}}=\mathrm{T}_{\mathrm{h}}, \tag{29}$$

where $\mathrm{H}_{\mathrm{corr}}$ is called the hidden correlation matrix, $\mathrm{B}_{\mathrm{ens}}$ is the global output weights matrix and T_{h} is the hidden-target matrix. $\mathrm{H}_{\mathrm{corr}}$ is defined as follows

$$\mathrm{H}_{\mathrm{corr}}=\begin{bmatrix}
C_1\varphi(1,1,1,1) & \ldots & C_1\varphi(1,1,1,L) & \ldots & \ldots & C_2\varphi(1,1,M,1) & \ldots & C_2\varphi(1,1,M,L)\\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\
C_1\varphi(1,L,1,1) & \ldots & C_1\varphi(1,L,1,L) & \ldots & \ldots & C_2\varphi(1,L,M,1) & \ldots & C_2\varphi(1,L,M,L)\\
C_2\varphi(2,1,1,1) & \ldots & C_2\varphi(2,1,1,L) & \ldots & \ldots & C_2\varphi(2,1,M,1) & \ldots & C_2\varphi(2,1,M,L)\\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\
C_2\varphi(2,L,1,1) & \ldots & C_2\varphi(2,L,1,L) & \ldots & \ldots & C_2\varphi(2,L,M,1) & \ldots & C_2\varphi(2,L,M,L)\\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\
C_2\varphi(M,1,1,1) & \ldots & C_2\varphi(M,1,1,L) & \ldots & \ldots & C_1\varphi(M,1,M,1) & \ldots & C_1\varphi(M,1,M,L)\\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\
C_2\varphi(M,L,1,1) & \ldots & C_2\varphi(M,L,1,L) & \ldots & \ldots & C_1\varphi(M,L,M,1) & \ldots & C_1\varphi(M,L,M,L)
\end{bmatrix}_{(ML\times ML)}$$

or $\mathrm{H}_{\mathrm{corr}}(p,q)=\begin{cases}C_1\varphi(m,n,k,l) & \text{if } m=k;\\ C_2\varphi(m,n,k,l) & \text{otherwise;}\end{cases}$

where $p, q = 1, \ldots, M \times L$; $m = \lceil \frac{p}{L} \rceil; n = ((p-1) \bmod L) + 1; k = \lceil \frac{q}{L} \rceil; l = ((q-1) \bmod L) + 1$; and mod is the modulo operation; $\mathrm{B}_{\mathrm{ens}}$ and T_{h} are defined as follows

$$\mathrm{B}_{\mathrm{ens}} = [\beta_{11}, \ldots, \beta_{1L}, \beta_{21}, \ldots, \beta_{2L}, \ldots, \beta_{M1}, \ldots, \beta_{ML}]^T_{ML \times 1},$$
$$\mathrm{T}_{\mathrm{h}} = [\varphi(1,1), \ldots, \varphi(1,L), \varphi(2,1), \ldots, \varphi(2,L), \ldots, \varphi(M,1), \ldots, \varphi(M,L)]^T_{ML \times 1}.$$

Instead of using gradient descent-based quadratic optimization techniques to tune the linear parameters β_{ij}, an analytical solution can be derived by using Eq. (29), that is

$$\widehat{\mathrm{B}}_{\mathrm{ens}} = \mathrm{H}^{-1}_{\mathrm{corr}} \mathrm{T}_{\mathrm{h}}. \tag{30}$$

Algorithm 1. DNNE algorithm.

Require: Training dataset $\mathcal{D}_t$, a basis function $G : \mathrm{R} \mapsto \mathrm{R}$; default is Sigmoid, scaling coefficient $\lambda \in [0, 1]$, the size of base model L, and the number of base models M.
Ensure: Trained DNNE model.
Initialize the architecture of M SLFNs to be the ensemble base models.
Initialize the basis functions of base models randomly i.e., w_j, b_j are randomly set.
Calculate the outputs of the hidden layers of base models for all examples in $\mathcal{D}_t$ i.e., calculate $g_{ij}(x_n)$.
Calculate the constants C_1 and C_2.
for $p \leftarrow 1$ to $M \times L$
  $m \leftarrow \lceil \frac{p}{L} \rceil$
  $n \leftarrow ((p-1) \bmod L) + 1$
  for $q \leftarrow 1$ to $M \times L$ **do**
    $k \leftarrow \lceil \frac{q}{L} \rceil$
    $l \leftarrow ((q-1) \bmod L) + 1$
    if $m = k$ **then**
      $\mathrm{H}_{\mathrm{corr}}[p, q] \leftarrow C_1 \varphi(m, n, k, l)$
    else
      $\mathrm{H}_{\mathrm{corr}}[p, q] \leftarrow C_2 \varphi(m, n, k, l)$
    end if
  end for
end for
$k \leftarrow 1$
for $i \leftarrow 1$ to M **do**
  for $j \leftarrow 1$ to L **do**
    $T_h[k] \leftarrow \varphi(i, j)$
    $k \leftarrow k + 1$
  end for
end for
Calculate $\mathrm{H}^{\dagger}_{\mathrm{corr}}$, the pseudo-inverse inverse of $\mathrm{H}_{\mathrm{corr}}$.
Calculate the estimated global output weights matrix $\widehat{\mathrm{B}}_{\mathrm{ens}}$ from Eq. (31).
for $i \leftarrow 1$ to M **do**
  Output weights of base model $i \leftarrow \widehat{\mathrm{B}}_{\mathrm{ens}}[(i-1)L + 1 : iL]$
end for
return Ensemble model (DNNE).

It has been aware that the matrix $\mathrm{H}_{\mathrm{corr}}$ tends to be ill-conditioned when dealing with real datasets in practice, especially when two or more basis functions are linearly correlated. In fact, the magnitudes of the resulting parameter values β_{ij} will be quite large which is not desirable indeed. To overcome this technical trouble, as done in numerical analysis, we employ the well-known singular value decomposition (SVD) techniques [31] to evaluate the parameters vector in this study. Denoted by $\mathrm{H}^{\dagger}_{\mathrm{corr}}$ as the generalized pseudo-inverse of the matrix $\mathrm{H}_{\mathrm{corr}}$ [23]. Thus, the estimated parameters vector given by Eq. (30) can be evaluated as

$$\widehat{\mathrm{B}}_{\mathrm{ens}} = \mathrm{H}^{\dagger}_{\mathrm{corr}} \mathrm{T}_{\mathrm{h}} \tag{31}$$

Algorithm 1 describes our proposed learning scheme for building neural-net ensembles with random weights. Obviously, it can be seen that our proposed NCL-based ensemble technique with a fast component model builder outperforms other NCL-based ensemble techniques where some classical training algorithms such as back-propagation algorithm are

employed. Based on this understanding, it does not make sense to take a long run to make performance comparisons against BP-based ensembles.

Remark 1. It should be pointed out that, with the best of our knowledge, feed-forward neural networks with random weights assignment have been originally proposed by Schmidt and his co-worker in [21]. Such an idea has been further explored by Igelnik and Pao in [22] where a significant result on universal approximation theorem was established. Recently, this kind of learner models with random parameters has received considerable attention due to its applicability for real-time data processing. In [26], Tyukin and Prokhorov regarded this type of neural networks as random basis function approximators. They provide some deeper insights on the feasibility of such random models for modeling and control applications. Their reported results are quite informative and useful to better understanding the problems associated with the randomness in the learner model. In this paper, we employ the same approach as used in [21] to derive an analytic expression of the weights in output layer, but with a different objective cost function.

Remark 2. The random input weights of base RVFL networks can be selected in different ways in order to improve the performance of the ensemble model. For example, the input weights can be selected so that the feature vectors generated from the hidden neurons are distributed on the surface of a ball centered at the origin of the feature space, but in a higher dimension space. However, this paper does not make any assumption on the selection of input weights. DNNE randomly initializes input weights using a uniform distribution $[-1,+1]$. On the other hand, the hidden feature vectors are scattered in a cube $[0,1]^d$ when a sigmoidal function is used as a basis function. Idealy, the hidden feature vectors of each base RVFL network in the ensemble model covers different regions in the feature space.

4. Performance evaluation

This section investigates the performance of DNNE ensembles and then compares it with other popular ensembling approaches: simple, bagging, boosting and random forests. We tried different configurations for the tested ensembles to figure out the effect of each design parameter on the generalization performance of each ensemble approach.

4.1. Datasets

Nine datasets have been selected to analyze the performance of our method on regression tasks. Among these datasets, four datasets are used to justify the function approximation capabilities of our ensemble method. These functions are: 3-d Mexican Hat, Friedman #1, Friedman #3 and Multi. The domain variables of these functions are randomly generated using a uniform probability distribution $U[a,b]$; where a and b are the lower and upper bounds of the corresponding variables, respectively. The size of each dataset and the constraints on its variables are listed in Table 1. A random Gaussian noise ϵ with zero mean $N(0,0.02)$ was also added to all examples during training in order to reliably measure the learning capabilities of the tested methods. However, testing examples are noise-free. The remaining datasets model real world applications obtained from UCI repository [32], StatLib repository (lib.stat.cmu.edu) and Delve repository (http://www.cs.toronto.edu/delve); below is a brief summary of each dataset.

- California Housing dataset contains 20,640 observations for estimating the median house prices in California, U.S. Each characterized by eight continuous features (median income, housing median age, total rooms, total bedrooms, population, households, latitude, and longitude).
- Quake dataset contains information for 2178 earthquakes occurred between January 1964 and February 1986 and each one was determined by three real variables (focal depth, latitude and longitude).

Table 1
Summary of regression datasets used in our study.

Dataset	Function/dependent variable	Attributes	Size
3-d Mex. hat	$y = sinc\sqrt{x_1^2+x_2^2} = \frac{sin\sqrt{x_1^2+x_2^2}}{\sqrt{x_1^2+x_2^2}} + \epsilon$	$x_i \sim U[-4\pi, 4\pi]$	3000
Friedman #1	$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon$	$x_i \sim U[0,1]$	5000
Friedman #3	$y = tan^{-1}\frac{x_2x_3 - \frac{1}{x_2x_4}}{x_1} + \epsilon$	$x_1 \sim U[0,100]$ $x_2 \sim U[40\pi, 560\pi]$ $x_3 \sim U[0,1]$ $x_4 \sim U[1,11]$	3000
Multi	$y = 0.79 + 1.27x_1x_2 + 1.56x_1x_4 + 3.42x_2x_5 + 2.06x_3x_4x_5 + \epsilon$	$x_i \sim U[0,1]$	4000
Cal. housing	House price	$[x_1, \ldots, x_8] \in R$	20,640
Quake	Earthquake magnitude	$[x_1, \ldots, x_3] \in R$	2178
Space ga	Proportion of votes cast per county	$[x_1, \ldots, x_6] \in R$	3107
Abalone	Abalone age	$[x_1, \ldots, x_8] \in R$	4177
Comp. activ.	CPU running time in user mode	$[x_1, \ldots, x_{12}] \in R$	8192

- Space ga dataset holds 3107 spatial data records on the U.S. county votes cast in the 1980 presidential elections and each record is composed of six input attributes (the population in each county, population in each county with a 12th grade or higher education, the number of owner-occupied housing units, the aggregate income, the X coordinate of the county, and the Y coordinate of the county) and one output dependent variable (logarithm of the proportion of votes cast per county).
- Abalone dataset is used to predict the age of abalone (number of rings in the shell) from eight physical measurements (sex, length, diameter, height, whole weight, shucked weight, viscera weight and shell weight) of 4177 abalones.
- Computer Activity dataset describes the portion of time that CPUs run in user-mode, based on 8192 computer system activities collected from a Sun SPARCstation 20/712 with 2 CPUs and 128 MB of memory running in a multi-user university department, and each system activity is evaluated using 12 system measures (number of reads and number of writes between system memory and user memory, number of system calls of all types, number of system read calls, number of system write calls, number of system fork calls, number of system exec calls, number of characters transferred by read calls, number of characters transferred by write calls, process run queue size, number of memory pages available to user processes, and number of disk blocks available for page swapping).

A summary of all datasets information is presented in Table 1. Note that all input and output attributes were normalized as follows

$$\acute{x} = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{32}$$

where $\acute{x}$ is the new normalized value of the attribute x; and x_{min} and x_{max} are the minimum and maximum values of the attribute x, respectively.

4.2. Experiments setup

Different datasets and different ensemble methods require different parameter values. In all methods, we use the RVFL network with Sigmoid basis function $G(x) = 1/(1 + e^{-x})$ as a base learner. In simple averaging, bagging, boosting and ERWNE, the component RVFL networks are trained using standard least squares method as described in page 142 of [31]; while CART algorithm [33] is used to train the base trees of random forest ensembles. In fact, there are three parameters to be set for our ensemble method DNNE (number of basis functions in RVFL networks L, ensemble size M, and penalty coefficient λ), two parameters for simple averaging ensemble (L and M), three parameters for bagging (L, M, and bootstrap sampling size), four parameters for boosting (L, M, bootstrap sampling size, and threshold ϕ for weighting incorrect predictions). Note that adaboost.RT version [34] is used to compare boosting algorithm with our method since it is one of the best boosting algorithms for data regression.

Exhaustive linear search strategy was adopted to find the optimal values for these parameters. The ensemble size M was searched in the range $[2, 15]$ with step 1 and the number of basis functions L in RVFL networks was searched in the range $[5, 50]$ with step 5. Similarly, the penalty coefficient λ and the boosting threshold ϕ were searched in the range $[0, 1]$ with steps 0.1 and 0.01, respectively. Moreover, 60% of the training examples were frequently sampled for bagging and boosting ensembles. On the other hand, the component decision trees of random forests were used with no pruning and the number of base trees was searched in the range [5, 100] with step 5. The random selection ratio of features was searched in the range $[0.1, 1]$ with step 0.1. All parameter values can be further tuned using other model selection methods, but we are only interested in highlighting the relative performance of DNNE ensemble in comparison with other ensembling methods. It is worthwhile mentioned that we use the same initial weights for the base networks in DNNE and simple averaging ensemble models. This demonstrates the effectiveness of negative correlation learning against simple independent learning. Note that the same RVFL network architecture is used for all ensemble individuals.

All methods were assessed using 10-fold cross-validation procedure. Each dataset is randomly partitioned into ten subsets that initiate 10 runs for each experiment. For each run, two subsets are used for testing and parameters selection and the remaining subsets are used all together for training. The performance measurements from all folds are collected and averaged over ten. Due to the randomness of weights and data sampling, each experiment is simulated ten times for more reliable results. Then, the results from all simulations are averaged. The Root Mean Squares Error (RMSE) metric is calculated on the testing subsets in order to evaluate the generalization capabilities of the tested methods, and it is given by

$$E = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{M}\sum_{i=1}^{M} f_i(x_n) - y_n\right)^2}, \tag{33}$$

where N is the number of testing examples; M is the number of base networks; and $f_i(x_n)$ is the output of the ith base network given an input x_n.

Eventually, the simulations were executed on a computer of 2 processors, each one with 3.0 GHz frequency, and 4.0 GB of RAM. Multiprocessing strategy has been used to run the simulations efficiently.

4.3. Results and discussion

4.3.1. Performance analysis of DNNE algorithm

We thoroughly investigate the performance of our proposed approach in different experimental designs. Fig. 1 plots the testing generalization error of DNNE models (solid lines) and the testing generalization error of simple averaging ensemble models (dashed lines) for different ensemble sizes and for all datasets. At each ensemble size in Fig. 1, the best penalty coefficient and best number of hidden neurons in the RVFL networks are used. We observe from Fig. 1 that increasing the number of base networks is not always beneficial for the ensemble generalization accuracy and the performance of DNNE algorithm is always better than the performance of simple averaging method regardless of the ensemble size. However, Quake dataset reveals an unexpected behavior at four base networks. It can been seen from for four datasets in Fig. 1 that our ensemble approach DNNE gains in accuracy when the ensemble expands up to 12 components (see Quake, Space ga, Abalone and Computer Activity). This can be due to the different datasets complexities. Apparently, 4–12 base networks are sufficient to constitute well generalized ensemble models with random weights using our algorithm.

Not only the ensemble size impacts the generalization performance of an ensembling method. Fig. 2 illustrates how the number of basis functions in the base networks can affect the accuracy of DNNE models (solid lines). In most of the experiments (3d Mexican Hat, Friedman #1, Friedman #3, California Housing, Space ga, Abalone, and Computer Activity), enlarging the hidden layer of base networks can reflect in a better performance. However, the testing RMSE of DNNE tends to become quite stable as the number of basis functions in RVFL networks approaches 40. In the other two datasets, DNNE method benefits from merging the ensemble components rather than from increasing the number of basis functions. 25 and 5 basis functions were sufficient to gain best accuracy for the Multi and Quake problems, respectively. These observations are expected because our algorithm tries to map the problem input space in a way that each base network learns different part of the features space. In addition to that, Fig. 2 reports that ensemble models outperform single models even when random weights are used (compare the solid and dashed lines with the dotted lines). It is also evident from Fig. 2 that DNNE models attain better generalization accuracies than simple averaging models regardless of the size of hidden layers in base networks. The Quake dataset, however, shows comparable performance for both approaches.

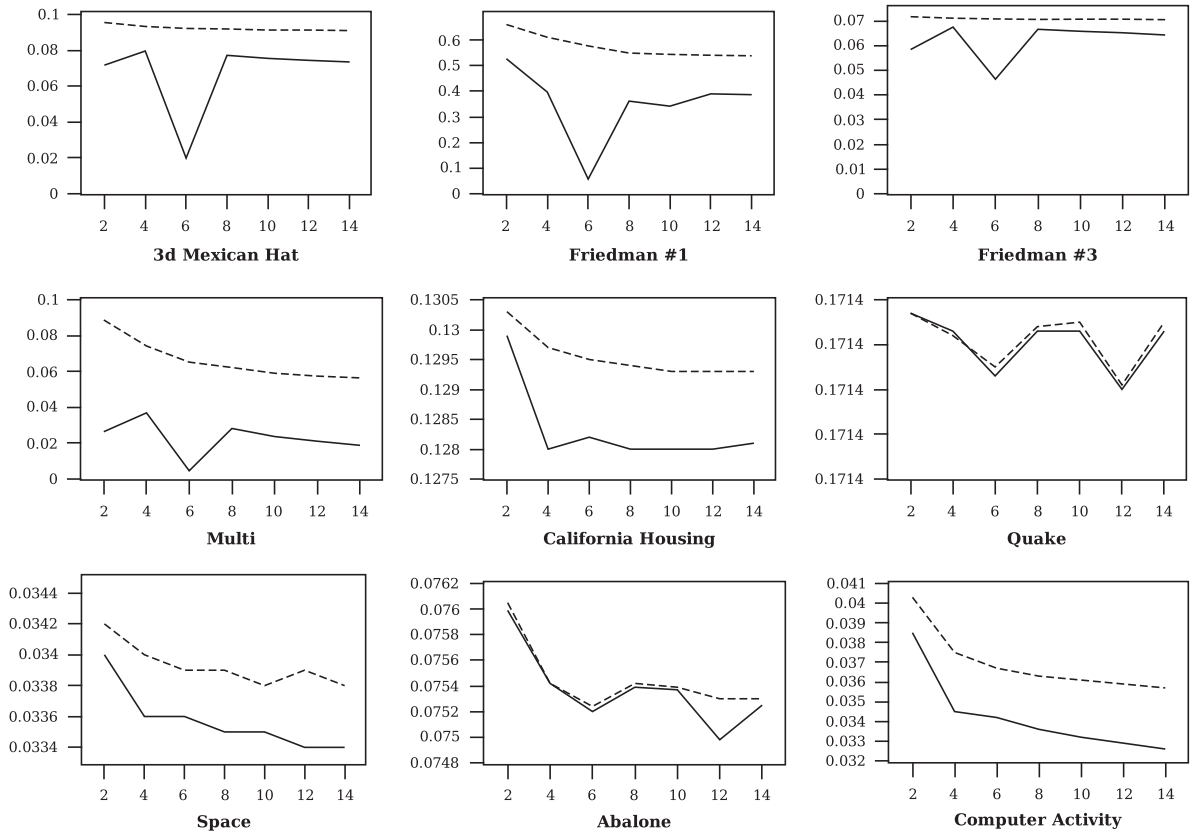

Fig. 1. Average testing RMSEs of DNNE ensembles and simple ensembles for ***different ensemble sizes***. *Solid lines* are for DNNE ensembles and *dashed lines* are for simple averaging ensembles.

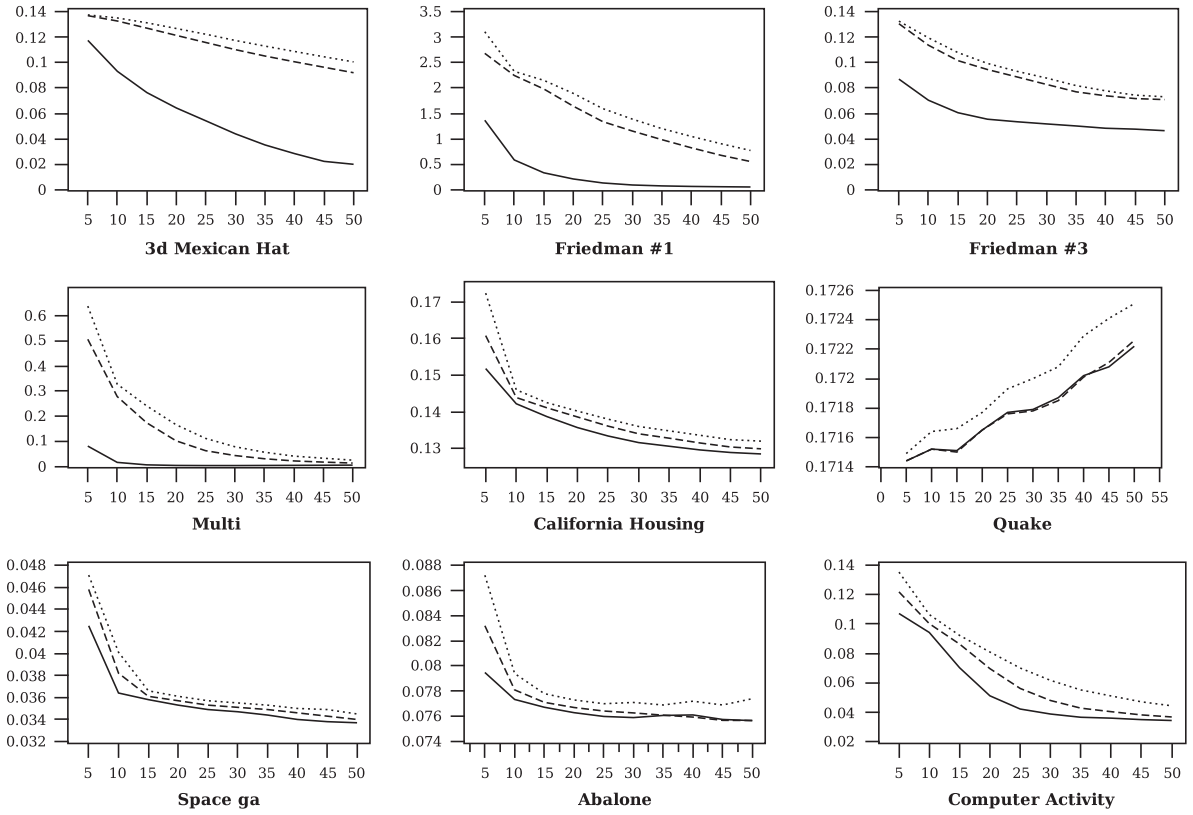

Fig. 2. Average testing RMSE of DNNE ensemble and simple ensemble for ***different base RVFL network sizes***. *Solid lines* are for DNNE ensembles, *dashed lines* are for simple averaging ensembles and *dotted lines* are for individual RVFL networks.

Besides the ensemble size and the number of basis functions in RVFL networks, the penalty coefficient of negative correlation learning plays a key role in the performance of our DNNE algorithm. Fig. 3 presents the testing RMSE of our method in terms of the penalty coefficient and for different base network complexities. The solid, dotted and dashed lines represent the performance measures for 10, 30, and 50 basis functions in the RVFL networks, respectively. Although the penalty coefficient values were searched in the range $[0, 1]$, we do not display the whole scale for sake of graphics clarity and only show the efficient ranges. Fig. 3 demonstrates that the testing generalization error slightly declines as the penalty coefficient value increases. However, the performance of DNNE dramatically improves when $\lambda \geqslant 0.5$. This result can be interpreted due to the scaling factor $1/2$ in the first term of Eq. (15). It forces the learning algorithm to focus on the first term rather than the penalty term when $\lambda < 0.5$. Fig. 3 is also consistent with the results of Fig. 2, that is, increasing the number of basis functions is beneficial for most study cases (can be seen from the solid lines).

It is worthy noted that although the basis functions of base RVFL networks (hidden neurons) work in a linear mode, the proposed learning model (DNNE) could model nonlinear relationships between the input and output variables in the investigated datasets. This linearity emerges because the input weights and biases are randomly fixed and only the output weights of base networks are updated.

4.3.2. Comparison with other ensemble methods

Tables 2 and 3 compare the performance of our method with four popular ensemble approaches: simple averaging, bagging, adaboost.rt and random forests in addition to single RVFL networks and our previous work (denoted by ERWNE in Table 2). The reported results were collected based on the best parameter values for each method and each dataset (can be seen in Table 4). They were selected by using the same search criteria explained in Section 4.2. From Tables 2 and 3, it is evident that our proposed ensemble method DNNE always performs better than other ensemble approaches including our previous work ERWNE [12]. Moreover, the complexity of DNNE models is much simpler than that of other ensemble models in five cases and comparable with others in four cases (compare M_1, M_2, M_3 and M_4 columns in Table 4). Since DNNE and random forest ensemble models have different base learners, it is useful to compare their performance closely. Table 2 shows that DNNE greatly outperforms random forests in all datasets except in Quake dataset where DNNE performs slightly better. The performance of random forests greatly degrades in the four artificial function approximation datasets (3d Mexican Hat, Friedman #1, Friedman #3, and Multi) and this is due to its high sensitivity to noisy data. In terms of ensemble size, the size of

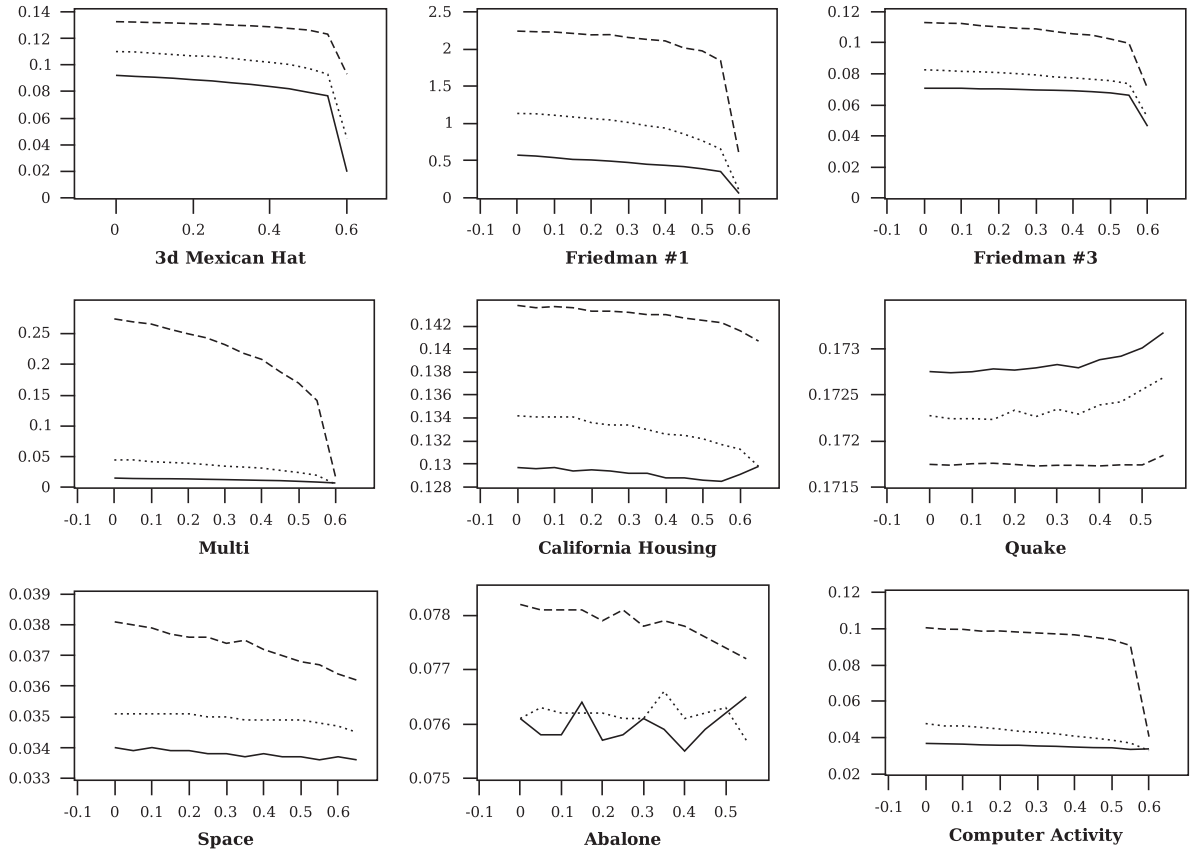

Fig. 3. Average testing RMSE of DNNE ensemble for ***different penalty coefficient values and base network sizes*** with specific number RVFL networks. *Dashed lines* are for DNNE ensembles with 10 hidden neurons in the base networks, *dotted lines* are for DNNE ensembles with 30 hidden neurons and *solid lines* are for DNNE ensembles for 50 hidden neurons.

Table 2
Comparison of testing RMSEs of DNNE, ERWNE, single RVFL network and random forest ensembles.

Dataset	DNNE	ERWNE	Random forest	RVFL network
3-d Mex. hat	**0.0201 ± 0.0024**	0.0847 ± 0.0012	0.1380 ± 0.0001	0.1005 ± 0.0036
Friedman #1	**0.0571 ± 0.0032**	0.3976 ± 0.0151	4.89163 ± 0.001597	0.7738 ± 0.1220
Friedman #3	**0.0465 ± 0.0020**	0.0690 ± 0.0012	0.1380 ± 0.0008	0.0731 ± 0.0020
Multi	**0.0048 ± 0.0003**	0.0383 ± 0.0034	1.1510 ± 0.0006	0.1127 ± 0.0268
Cal. housing	**0.1285 ± 0.0011**	0.1290 ± 0.0015	0.2275 ± 0.0020	0.1320 ± 0.0015
Quake	**0.171420 ± 0.0001**	0.171478 ± 0.0002	0.171970 ± 0.0002	0.171478 ± 0.0003
Space ga	**0.0334 ± 0.0003**	0.0342 ± 0.0006	0.0618 ± 0.0001	0.0346 ± 0.0008
Abalone	**0.0750 ± 0.0006**	0.0757 ± 0.0011	0.1017 ± 0.0005	0.0769 ± 0.0024
Comp. activ.	**0.0326 ± 0.0004**	0.0353 ± 0.0010	0.0829 ± 0.0001	0.0443 ± 0.0042

Table 3
Comparison of testing RMSEs of DNNE, simple and bagging ensembles and Adaboost.Rt.

Dataset	DNNE	Simple	Bagging	Adaboost.RT
3-d Mex. hat	**0.0201 ± 0.0024**	0.0920 ± 0.0017	0.0910 ± 0.0012	0.0908 ± 0.0012
Friedman #1	**0.0571 ± 0.0032**	0.5571 ± 0.0488	0.5322 ± 0.0327	0.5326 ± 0.0250
Friedman #3	**0.0465 ± 0.0020**	0.0710 ± 0.0009	0.0705 ± 0.0007	0.0722 ± 0.0008
Multi	**0.0048 ± 0.0003**	0.0640 ± 0.0092	0.0126 ± 0.0008	0.0495 ± 0.0074
Cal. housing	**0.1285 ± 0.0011**	0.1299 ± 0.0007	0.1292 ± 0.0005	0.1290 ± 0.0004
Quake	**0.171420 ± 0.0001**	0.171421 ± 0.0001	0.171434 ± 0.0002	0.171445 ± 0.0002
Space ga	**0.0334 ± 0.0003**	0.0339 ± 0.0002	0.0339 ± 0.0004	0.0346 ± 0.0007
Abalone	**0.0750 ± 0.0006**	0.0753 ± 0.0006	0.0756 ± 0.0011	0.0757 ± 0.0010
Comp. activ.	**0.0326 ± 0.0004**	0.0357 ± 0.0005	0.0358 ± 0.0007	0.0378 ± 0.0010

Table 4
Optimal parameter values of each dataset for the results in Tables 3 and 2.

Dataset	DNNE			Bagging		Adaboost.RT			Forest	
	M_1[a]	L_1[b]	λ[c]	M_2[a]	L_2[a]	M_3[a]	L_3[a]	ϕ[d]	M_4[a]	f[e]
3-d Mex. hat	**6**	50	0.6	15	50	15	50	0.15	30	0.1
Friedman #1	**6**	50	0.6	15	50	15	50	0.05	75	1.0
Friedman #3	**6**	50	0.6	14	50	15	50	0.05	75	1.0
Multi	**6**	25	0.6	15	50	15	35	0.05	100	0.9
Cal. housing	**4**	50	0.55	14	50	15	50	0.05	15	0.1
Quake	12	5	0.1	11	5	15	5	0.1	60	0.3
Space ga	**4**	50	0.5	12	50	15	50	0.05	25	1.0
Abalone	**12**	45	0.5	15	45	15	45	0.05	60	0.1
Comp. activ.	**12**	50	0.5	14	50	15	50	0.05	85	1.0

[a] The ensemble size.
[b] The number of basis functions in the RVFL base networks.
[c] The penalty coefficient of the negative correlation learning formula.
[d] The incorrect predictions threshold.
[e] The features random selection percentage.

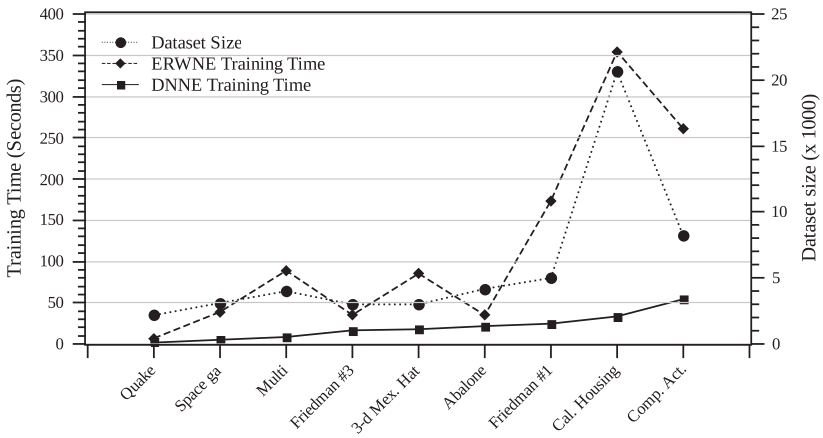

Fig. 4. Comparison of average training time of our current DNNE algorithm and our previous ERWNE algorithm.

random forest models is much greater than that of DNNE models, i.e., few networks are used in DNNE models compared with many decision trees used in RF models.

As mentioned in Section 1, we are looking for an efficient solution that reduces the training time of ensemble models. Fig. 4 illustrates the average training times of our current DNNE algorithm and our previous ERWNE algorithm that was developed in [12] along with the datasets sizes. It is evident that DNNE runs efficiently regardless of the dataset size while ERWNE suffers when the dataset size dramatically increases, as can be seen from Cal. Housing and Comp. Act. datasets. Note that the training time of DNNE for Comp. Act. is greater than the training time for Cal. Housing because the number of base models in Comp. Act. ensembles is larger (see Table 4).

5. Conclusions

Evolutionary approaches combined with gradient-based learning algorithms are mainly employed to build neural-net ensembles with high diversity among their base networks, while the overall ensemble accuracy is well maintained. However, these approaches, including our previous work in [12], are time consuming and exhaustively explore the ensemble hypotheses space. In this paper, we proposed an effective and efficient solution to build ensemble models in a very short time. The majority of RVFL networks' weights (hidden layer weights and biases) are randomly assigned, and the rest of weights can be computed analytically based on the negative correlation learning (NCL) and least squares methods. The results show that our proposed algorithm is promising for data regression applications. It should be pointed out that this is the first study to investigate the effectiveness of least squares method in building ensemble models with random weights.

Although DNNE algorithm is effective and efficient, it still has some limitations. For instance, the performance of our DNNE is sensitive to the value of regularizing factor λ. It is interesting to do further research on how to reduce the influence of this free parameter on the algorithm performance. Another limitation of DNNE is the high computational complexity when it attempts to solve problems that require large number of RVFL networks with more hidden neurons. The computa-

tional cost is mainly dominated by computing the matrix H_{corr} and its pseudo-inverse. One way to alleviate this computational burden is to partition the whole training dataset into some subsets for building compact RVFL networks. Moreover, the pseudo-inverse matrix can be calculated using more efficient algorithms.

This work sets a basis for future research on building advanced neural-net ensembles with random weights. As can be seen that other basis functions, such as kernel functions used in support vector machines (SVMs) and Radial Basis Functions (RBFs), can be also employed in RVFL networks. As for if such a replacement can result in better performance, further empirical studies with comparisons are needed. Furthermore, with some adjustments, the proposed algorithm in this paper can be easily extended for data classification problems.

References

[1] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[2] R.E. Schapire, The strength of weak learnability, Mach. Learn. 5 (2) (1990) 197–227.
[3] Y. Freund, Boosting a weak learning algorithm by majority, in: Proceedings of the 3rd Annual Workshop on Computational Learning Theory, Rochester NY, USA, 1990, pp. 202–216.
[4] L. Breiman, Radnom Forests – Random Features, Tech. Rep. 567, University of California, Berkeley CA, USA, 1999.
[5] L. Hansen, P. Salamon, Neural network ensembles, IEEE Trans. Pattern Anal. Mach. Intell. 12 (10) (1990) 993–1001.
[6] M.P. Perrone, L.N. Cooper, When Networks Disagree: Ensemble Methods for Hybrid Neural Networks, Tech. Rep. 61, Brown University, Providence RI, USA, 1993.
[7] A. Krogh, J. Vedelsby, Neural network ensembles, cross validation and active learning, in: Advances in Neural Information Processing Systems, MIT Press, 1995, pp. 231–238.
[8] X. Yao, Y. Liu, Making use of population information in evolutionary artificial neural networks, IEEE Trans. Syst., Man Cybernet. 28 (3) (1998) 417–425.
[9] D.H. Wolpert, Stacked generalization, Neural Netw. 5 (1992) 241–259.
[10] G. Brown, J.L. Wyatt, P. Tiňo, Managing diversity in regression ensembles, J. Mach. Learn. Res. 6 (2005) 1621–1650.
[11] D.W. Opitz, J.W. Shavlik, Generating accurate and diverse members of a neural network Ensemble, in: Advances in Neural Information Processing Systems, MIT Press, 1996, pp. 535–541.
[12] D. Wang, M. Alhamdoosh, Evolutionary extreme learning machine ensembles with size control, Neurocomputing 102 (2013) 98–110.
[13] Y. Liu, X. Yao, T. Higuchi, Evolutionary ensembles with negative correlation learning, IEEE Trans. Evol. Comput. 4 (4) (2000) 380–387.
[14] B. Rosen, Ensemble learning using decorrelated neural networks, Connect. Sci. 8 (1996) 373–384.
[15] Y. Liu, X. Yao, Ensemble learning via negative correlation, Neural Netw. 12 (10) (1999) 1399–1404.
[16] H. Chen, X. Yao, Regularized negative correlation learning for neural network ensembles, IEEE Trans. Neural Netw. 20 (12) (2009) 1962–1979.
[17] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[18] M. Leshno, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Netw. 6 (1993) 861–867.
[19] J. Park, I.W. Sandberg, Universal approximation using radial-basis-function networks, Neural Comput. 3 (2) (1991) 246–257.
[20] B. Igelnik, Y.-H. Pao, S. LeClair, C.Y. Shen, The ensemble approach to neural-network learning and generalization, IEEE Trans. Neural Netw. 10 (1) (1999) 19–30.
[21] W. Schmidt, M. Kraaijveld, R. Duin, Feedforward neural networks with random weights, in: Proceedings of 11th IAPR International Conference on Pattern Recognition Methodology and Systems, 1992, pp. 1–4.
[22] B. Igelnik, Y.-H. Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link net, IEEE Trans. Neural Netw. 6 (6) (1995) 1320–1329.
[23] C.R. Rao, S.K. Mitra, Generalized Inverse of Matrices and Its Applications, Wiley, New York, 1971.
[24] Y.-H. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Comput. Mag. 25 (5) (1992) 76–79.
[25] D.J. Albers, J.C. Sprott, W.D. Dechert, Routes to chaos in neural networks with random weights, Int. J. Bifurcat. Chaos 8 (7) (1998) 1463–1478.
[26] I. Tyukin, D. Prokhorov, Feasibility of random basis function approximators for modeling and control, in: Proceedings of IEEE Multi-Conference on Systems and Control, Saint Petersburg, Russia, 2009, pp. 1391–1396.
[27] S. Geman, E. Bienenstock, R. Doursat, Neural networks and the bias/variance dilemma, Neural Comput. 4 (1) (1992) 1–58.
[28] M. Pincus, A closed form solution of certain types of constrained optimization problems, Oper. Res. 16 (1968) 690–694.
[29] N. Ueda, R. Nakano, Generalization error of ensemble estimators, in: Proceedings of IEEE International Conference on Neural Networks, Washington DC, USA, 1996, pp. 90–95.
[30] Y. Liu, X. Yao, Negatively correlated neural networks can produce best ensembles, Aust. J. Intell. Inform. Process. Syst. 4 (3/4) (1997) 176–185.
[31] C.M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 2006.
[32] A. Frank, A. Asuncion, UCI Machine Learning Repository, 2010. <http://archive.ics.uci.edu/ml>.
[33] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth and Brooks/Cole Advanced Books and Software, Monterey CA, USA, 1984.
[34] D.L. Shrestha, D.P. Solomatine, Experiments with AdaBoost.RT, an improved boosting scheme for regression, Neural Comput. 18 (7) (2006) 1678–1710.