Neurocomputing 149 (2015) 464–471

Contents lists available at ScienceDirect

Neurocomputing

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

Elastic extreme learning machine for big data classification

Junchang Xin [a,*], Zhiqiong Wang [b], Luxuan Qu [b], Guoren Wang [a]

[a] College of Information Science & Engineering, Northeastern University, China
[b] Sino-Dutch Biomedical & Information Engineering School, Northeastern University, China

ARTICLE INFO

Article history:
Received 22 August 2013
Received in revised form
25 September 2013
Accepted 30 September 2013
Available online 10 September 2014

Keywords:
Extreme learning machine
Big data classification
Incremental learning
Decremental learning
Correctional learning

ABSTRACT

Extreme Learning Machine (ELM) and its variants have been widely used for many applications due to its fast convergence and good generalization performance. Though the distributed ELM* based on MapReduce framework can handle very large scale training dataset in big data applications, how to cope with its rapidly updating is still a challenging task. Therefore, in this paper, a novel Elastic Extreme Learning Machine based on MapReduce framework, named Elastic ELM (E^2LM), is proposed to cover the shortage of ELM* whose learning ability is weak to the updated large-scale training dataset. Firstly, after analyzing the property of ELM* adequately, it can be found out that its most computation-expensive part, matrix multiplication, can be incrementally, decrementally and correctionally calculated. Next, the Elastic ELM based on MapReduce framework is developed, which first calculates the intermediate matrix multiplications of the updated training data subset, and then update the matrix multiplications by modifying the old matrix multiplications with the intermediate ones. Then, the corresponding new output weight vector can be obtained with centralized computing using the update the matrix multiplications. Therefore, the efficient learning of rapidly updated massive training dataset can be realized effectively. Finally, we conduct extensive experiments on synthetic data to verify the effectiveness and efficiency of our proposed E^2LM in learning massive rapidly updated training dataset with various experimental settings.

1. Introduction

Today, we are surrounded by data. People upload videos, take pictures on their mobile phones, text friends, update their Facebook status, leave comments around the web, and so forth. Machines, too, are generating and keeping more and more data [1]. It is difficult to estimate how much total data are stored as electronically all over the world. For example: Facebook stored almost 10 billion pictures, it is about 1PB; New York Stock Exchange will produce 1TB exchange data; The Internet Archive store the data about 2PB, and it is increasing at least in speed of 20TB data per month [2]. Thus, the era of Big Data [3] has arrived. Big Data is known for its 4Vs [4]: V1 (Volume), it involves a great volume of data; V2 (Variety), the data cannot be structured into regular database tables; V3 (Velocity), the data is produced with great velocity and must be captured and processed rapidly; V4 (Value), high commercial value but low density value, meaning that sometimes there is a very big volume of data to process before finding valuable needed information.

Due to the characteristics of excellent generalization performance, little human intervene, and rapid training speed, Extreme Learning Machine (ELM) [5–10] has recently attracted more and more researchers' attention [11]. Because of its advantage, ELM can be applied in many fields and displayed a significant result [12–26]. As a variant of ELM, distributed ELM (i.e. PELM [27] and ELM* [28]) based on MapReduce [29–31] can resolve the V1 (Volume) problem of Big Data. However, it is quite common in big data classifications that some new training data arrived, some old training data expired and some error training data corrected. For example, a large number of patients all over the world go to hospital every day. Some patients' conditions are clear and consistent, and they got doctors' diagnosis; other patients, whose conditions have been changed from the last diagnosis, need to be further examined by the doctors again; the others' conditions are re-confirmed after the consultation from doctors. All the above situations can result in updates of the training dataset on Computer-Aided Diagnosis (CAD).

A simple and direct way is to retraining the ELM* [28] using the whole training dataset. Obviously, this kind of method is time-consuming, since the training dataset is very large. Therefore, it is essential to improve the ELM* [28] algorithm, in order to support the functionalities such as incremental learning, decremental learning, and correctional learning. There is a great overlap between the old training

* Corresponding author.
E-mail address: xinjunchang@ise.neu.edu.cn (J. Xin).

http://dx.doi.org/10.1016/j.neucom.2013.09.075
0925-2312/

dataset and the new one, if the overlapped information can be used, the retraining cost will be reduced greatly. Therefore, in this paper, an Elastic ELM (E^2LM) is proposed to improve ELM* [28] with the abilities of incremental learning, decremental learning, and correctional learning. The contributions of this paper can be summarized as follows.

(1) We prove theoretically that the most expensive computation part in ELM* can be incrementally, decrementally and correctionally calculated, which indicate that ELM* can be extended to support incremental learning, decremental learning, and correctional learning.
(2) A novel Elastic Extreme Learning Machine (E^2LM) based on MapReduce framework is proposed, which can enhance the training performance of ELM* for handling the rapid updated massive training dataset.
(3) Last but not least, our extensive experimental studies using synthetic data show that our proposed E^2LM can learn the rapid updated massive training dataset effectively, which can fulfill the requirements of many big data classification applications.

The remainder of the paper is organized as follows. Section 2 briefly introduces traditional ELM and distributed ELM*. The theoretical foundation and the computational details of the proposed E^2LM approach are introduced in Section 3. The experimental results are reported in Section 4 to show the effectiveness and efficiency of our E^2LM approach. Finally, we conclude this paper in Section 5.

2. Background

In this section, we describe the background for our work, which includes a brief overview of ELM and ELM*.

2.1. ELM

ELM [5,6] has been originally developed for single hidden-layer feedforward neural networks (SLFNs) and then extended to the "generalized" SLFNs where the hidden layer need not be neuron alike [7,8]. ELM first randomly assigns the input weights and the hidden layer biases, and then analytically determines the output weights of SLFNs. It can achieve better generalization performance than other conventional learning algorithms at a extremely fast learning speed. Besides, ELM is less sensitive to user-specified parameters and can be deployed faster and more conveniently [9,10].

For N arbitrary distinct samples $(\mathbf{x}_j, \mathbf{t}_j)$, where $\mathbf{x}_j = [x_{j1}, x_{j2}, \ldots, x_{jn}]^T \in \mathbb{R}^n$ and $\mathbf{t}_j = [t_{j1}, t_{j2}, \ldots, t_{jm}]^T \in \mathbb{R}^m$, standard SLFNs with L hidden nodes and activation function $g(x)$ are mathematically modeled as

$$\sum_{i=1}^{L} \boldsymbol{\beta}_i g_i(\mathbf{x}_j) = \sum_{i=1}^{L} \boldsymbol{\beta}_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{o}_j \qquad (j = 1, 2, \ldots, N) \tag{1}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^T$ is the weight vector connecting the ith hidden node and the input nodes, $\boldsymbol{\beta}_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the weight vector connecting the ith hidden node and the output nodes, b_i is the threshold of the ith hidden node, and $\mathbf{o}_j = [o_{j1}, o_{j2}, \ldots, o_{jm}]^T$ is the jth output vector of the SLFNs [5].

The standard SLFNs with L hidden nodes and activation function $g(x)$ can approximate these N samples with zero error. It means $\sum_{j=1}^{L} \| \mathbf{o}_j - \mathbf{t}_j \| = 0$ and there exist $\boldsymbol{\beta}_i$, $\mathbf{w}_i$ and b_i such that

$$\sum_{i=1}^{L} \boldsymbol{\beta}_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j \quad (j = 1, 2, \ldots, N) \tag{2}$$

The equation above can be expressed compactly as follows:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{3}$$

where $\mathbf{H}(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_L, b_1, b_2, \ldots, b_L, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_L)$

$$= [h_{ij}] = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & g(\mathbf{w}_2 \cdot \mathbf{x}_1 + b_2) & \ldots & g(\mathbf{w}_L \cdot \mathbf{x}_1 + b_L) \\ g(\mathbf{w}_1 \cdot \mathbf{x}_2 + b_1) & g(\mathbf{w}_2 \cdot \mathbf{x}_2 + b_2) & \ldots & g(\mathbf{w}_L \cdot \mathbf{x}_2 + b_L) \\ \vdots & \vdots & \vdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & g(\mathbf{w}_2 \cdot \mathbf{x}_N + b_2) & \ldots & g(\mathbf{w}_L \cdot \mathbf{x}_N + b_L) \end{bmatrix}_{N \times L} \tag{4}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_{11} & \beta_{12} & \cdots & \beta_{1m} \\ \beta_{21} & \beta_{22} & \cdots & \beta_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{L1} & \beta_{L2} & \cdots & \beta_{Lm} \end{bmatrix}_{L \times m} \quad \text{and}$$

$$\mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & \ldots & t_{1m} \\ t_{21} & t_{22} & \ldots & t_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ t_{N1} & t_{N2} & \ldots & t_{Nm} \end{bmatrix}_{N \times m} \tag{5}$$

$\mathbf{H}$ is called the hidden layer output matrix of the neural network and the ith column of $\mathbf{H}$ is the ith hidden node output with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. The smallest norm least-squares solution of the above linear system is

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T} \tag{6}$$

where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized the inverse of matrix $\mathbf{H}$. Then the output function of ELM can be modeled as follows.

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\mathbf{H}^{\dagger}\mathbf{T} \tag{7}$$

2.2. ELM*

The orthogonal projection method can be efficiently used in ELM [10]: $\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$ if $\mathbf{H}^T\mathbf{H}$ is nonsingular or $\mathbf{H}^{\dagger} = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}$ if $\mathbf{H}\mathbf{H}^T$ is nonsingular. According to the ridge regression theory, it suggests that a positive value $1/\lambda$ is added to the diagonal of $\mathbf{H}^T\mathbf{H}$ or $\mathbf{H}\mathbf{H}^T$ in the calculation of the output weights β, and the resultant solution is stable and tends to have better generalization performance [10]. Moreover, in big data applications, we can easily have $N \gg L$. Thus, the size of $\mathbf{H}^T\mathbf{H}$ is much smaller than that of $\mathbf{H}\mathbf{H}^T$, we can get

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{8}$$

and the corresponding output function of ELM is

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{9}$$

Let $\mathbf{U} = \mathbf{H}^T\mathbf{H}$, $\mathbf{V} = \mathbf{H}^T\mathbf{T}$, and we can get,

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{U}\right)^{-1}\mathbf{V} \tag{10}$$

According to Eq. (4), we have $h_{ij} = g(\mathbf{w}_j \cdot \mathbf{x}_i + b_j)$, and $h_{ij}^T = h_{ji} = g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i)$. Thus, we can further have,

$$u_{ij} = \sum_{k=1}^{N} h_{ik}^T h_{kj} = \sum_{k=1}^{N} h_{ki} h_{kj} = \sum_{k=1}^{N} g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) g(\mathbf{w}_j \cdot \mathbf{x}_k + b_j) \tag{11}$$

$$v_{ij} = \sum_{k=1}^{N} h_{ik}^T t_{kj} = \sum_{k=1}^{N} h_{ki} t_{kj} = \sum_{k=1}^{N} g(\mathbf{w}_i \cdot \mathbf{x}_k + b_i) t_{kj} \tag{12}$$

According to Eqs. (11) and (12), the process of calculating $\mathbf{U}$ and $\mathbf{V}$ are both decomposable. Thus, we can use MapReduce framework to speedup the computation of output weight vector β. The process of calculating matrices $\mathbf{U}$ and $\mathbf{V}$ based on MapReduce framework is shown in Algorithm 1 [28].

The algorithm has two classes, Class Mapper (Lines 1–20) and Class Reducer (Lines 21–26). Class Mapper contains three methods, Initialize (Lines 2–4), Map (Lines 5–14) and Close (Lines 15–20), while Class Reducer only contains one method, Reduce (Lines 22–26).

In the Initialize method of Mapper, we initialize two arrays, u and v, which are used to store the intermediate summation of the elements in matrices $\mathbf{U}$ and $\mathbf{V}$ respectively. In the Map method of Mapper, first, we initial the local variable h (Line 6), and then resolve training record s, which means dividing s into training data $\mathbf{x}_k$ and its corresponding training result t (Line 7). Moreover, we calculate the partial result of hidden layer output matrix corresponding to $\mathbf{x}_k$ (Lines 8–9). Next, we calculate the partial summation of elements in matrix $\mathbf{U}$ and $\mathbf{V}$ respectively, and store the results into their corresponding local variables u and v (Lines 10–14). In the Close method of Mapper, the intermediate summation stored in u and v is emitted by the mapper (Lines 16–20). In the Reduce method of Reducer, first, we initialize a temporary variable uv (Line 23). And then, we combine the intermediate summation of different mappers which have the same Key, and furthermore, get the final summation of the corresponding element of the Key (Lines 24–25). Finally, we store the results into the distributed file system (Line 26).

Algorithm 1. ELM*-**U**, **V** on MapReduce.

```
1   class Mapper
2     method Initialize()
3       u= new AssociativeArray;
4       v= new AssociativeArray;
5     method Map(sid id, sample s)
6       h= new AssociativeArray;
7       (x, t)=Parse(s);
8       for i=1 to L do
9         h[i]=g(w_i · x+b_i);
10      for i=1 to L do
11        for j=1 to L do
12          u[i,j]=u[i,j]+h[i]h[j];
13        for j=1 to m do
14          v[i,j]=v[i,j]+h[i]t[j];
15    method Close()
16      for i=1 to L do
17        for j=1 to L do
18          context.write(triple ('U', i, j), sum u[i,j]);
19        for j=1 to m do
20          context.write(triple ('V', i, j), sum v[i,j]);
21  class Reducer
22    method Reduce(triple p, sum [s_1, s_2, ...])
23      uv=0;
24      for all sum s ∈ [s_1, s_2, ...] do
25        uv=uv+s;
26        context.write(triple p, sum uv);
```

Then, the process of ELM* is shown in Algorithm 2 [28]. First, we generate L pairs of hidden node parameters $(\mathbf{w}_i, b_i)$ randomly (Lines 1–2). And then, we calculate matrices $\mathbf{U}$ and $\mathbf{V}$ according to the input parameters and randomly generate parameters using Algorithm 1 (Line 3). Finally, using Eq. (10), we solve the output weight vector β (Line 4).

Algorithm 2. ELM*.

```
1  for i=1 to L do
2    Randomly generate hidden node parameters (w_i, b_i);
3  Calculate U=H^T H, V=H^T T with MapReduce;
4  Calculate the output weight vector β=(I/λ+U)^{-1}V;
```

3. Elastic ELM

In this section, we explain the theoretical foundation and the computational details of the proposed E^{2}LM approach. The incremental learning, decremental learning and correctional learning methods are presented in Sections 3.1, 3.2 and 3.3 respectively.

3.1. Incremental learning

The original training dataset $\mathcal{N}=\{(\mathbf{x}_j,\mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\mathbf{t}_j\in\mathbb{R}^m,j=1,2,\ldots,N\}$ is denoted as $\mathcal{N}=(\mathbf{X},\mathbf{T})$. Then, according to Eq. (4), we can get the hidden layer output matrix $\mathbf{H}$ from $\mathbf{X}$. Thus, we have $(\mathbf{H},\mathbf{T})$.

The newly arrived training dataset $\Delta\mathcal{N}=\{(\mathbf{x}_j,\mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\mathbf{t}_j\in\mathbb{R}^m,j=N+1,N+2,\ldots,N+\Delta N\}$ is denoted as $\Delta\mathcal{N}=(\Delta\mathbf{X},\Delta\mathbf{T})$. Then, according to Eq. (4), we can get the hidden layer output matrix $\Delta\mathbf{H}$ from $\Delta\mathbf{X}$. Thus, we have $(\Delta\mathbf{H},\Delta\mathbf{T})$.

Now, the new training dataset after $\Delta\mathcal{N}$ arrived is $\mathcal{N}^+=(\mathbf{X}+\Delta\mathbf{X},\mathbf{T}+\Delta\mathbf{T})=\{(\mathbf{x}_j,\mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\mathbf{t}_j\in\mathbb{R}^m,j=1,2,\ldots,N,N+1,N+2\ldots,N+\Delta N\}$. The new hidden layer output matrix $\mathbf{H}^+$ can be easily derived from $\mathbf{H}$ and $\Delta\mathbf{H}$.

Thus, we have $(\mathbf{H}^+,\mathbf{T}^+)$, where $\mathbf{H}^+=\begin{bmatrix}\mathbf{H}\\ \Delta\mathbf{H}\end{bmatrix}$, $\mathbf{T}^+=\begin{bmatrix}\mathbf{T}\\ \Delta\mathbf{T}\end{bmatrix}$.

Then, according to the matrix multiplication operator, we have

$$\mathbf{H}^{+T}\mathbf{H}^+=\begin{bmatrix}\mathbf{H}\\ \Delta\mathbf{H}\end{bmatrix}^T\begin{bmatrix}\mathbf{H}\\ \Delta\mathbf{H}\end{bmatrix}=[\mathbf{H}^T\ \ \Delta\mathbf{H}^T]\begin{bmatrix}\mathbf{H}\\ \Delta\mathbf{H}\end{bmatrix}=\mathbf{H}^T\mathbf{H}+\Delta\mathbf{H}^T\Delta\mathbf{H} \tag{13}$$

$$\mathbf{H}^{+T}\mathbf{T}^+=\begin{bmatrix}\mathbf{H}\\ \Delta\mathbf{H}\end{bmatrix}^T\begin{bmatrix}\mathbf{T}\\ \Delta\mathbf{T}\end{bmatrix}=[\mathbf{H}^T\ \ \Delta\mathbf{H}^T]\begin{bmatrix}\mathbf{T}\\ \Delta\mathbf{T}\end{bmatrix}=\mathbf{H}^T\mathbf{T}+\Delta\mathbf{H}^T\Delta\mathbf{T} \tag{14}$$

Let $\mathbf{U}^+=\mathbf{H}^{+T}\mathbf{H}^+$ and $\Delta\mathbf{U}=\Delta\mathbf{H}^T\Delta\mathbf{H}$, we can get $\mathbf{U}^+=\mathbf{U}+\Delta\mathbf{U}$. Similar to Eq. (11), the element δu_{ij} of matrix $\Delta\mathbf{U}$ can be expressed as follows:

$$\delta u_{ij}=\sum_{k=N+1}^{N+\Delta N}g(\mathbf{w}_i\cdot\mathbf{x}_k+b_i)g(\mathbf{w}_j\cdot\mathbf{x}_k+b_j) \tag{15}$$

Let $\mathbf{V}^+=\mathbf{H}^{+T}\mathbf{T}^+$ and $\Delta\mathbf{V}=\Delta\mathbf{H}^T\Delta\mathbf{T}$, we can get $\mathbf{V}^+=\mathbf{V}+\Delta\mathbf{V}$. Similar to Eq. (12), the element δv_{ij} of matrix $\Delta\mathbf{V}$ can be expressed as follows:

$$\delta v_{ij}=\sum_{k=N+1}^{N+\Delta N}g(\mathbf{w}_i\cdot\mathbf{x}_k+b_i)t_{kj} \tag{16}$$

According to Eqs. (15) and (16), the process of calculating $\Delta\mathbf{U}$ and $\Delta\mathbf{V}$ are both decomposable. Thus, we can use MapReduce framework to speedup their computation, and the process of calculating matrices $\Delta\mathbf{U}$ and $\Delta\mathbf{V}$ based on MapReduce framework is similar to Algorithm 1. Then, the process of Incremental Learning (IL) approach in E^{2}LM is shown in Algorithm 3. Firstly, we get the preserved matrices $\mathbf{U}$ and $\mathbf{V}$ (Line 1) and the L pairs of hidden node parameters $(\mathbf{w}_i,b_i)$ (Lines 2–3). Next, we calculate matrices $\Delta\mathbf{U}$ and $\Delta\mathbf{V}$ according to the preserved parameters using Algorithm 1 (Line 4). And then, update the corresponding matrices $\mathbf{U}$ and $\mathbf{V}$ using $\Delta\mathbf{U}$ and $\Delta\mathbf{V}$ (Line 5). Finally, using Eq. (10), we solve the new output weight vector β (Line 6). The E^{2}LM incremental learning process for newly arrived training dataset $\Delta\mathcal{N}$ finishes after the complement of the training process above, and then we can predict the outputs of other data according to Eq. (9).

Algorithm 3. Incremental Learning.

```
1  Get the preserved matrices U and V;
2  for i=1 to L do
3    Get the preserved hidden node parameters (w_i, b_i);
4  Calculate ΔU=ΔH^T ΔH, ΔV=ΔH^T ΔT with MapReduce;
5  Update the matrices U=U+ΔU and V=V+ΔV;
6  Calculate the new output weight vector β=(I/λ+U)^{-1}V;
```

3.2. Decremental learning

The original training dataset $\mathcal{N}=\{(\mathbf{x}_j,\mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\mathbf{t}_j\in\mathbb{R}^m,j=1,2,\ldots,N\}$ is denoted as $\mathcal{N}=(\mathbf{X},\mathbf{T})$. Then, according to Eq. (4), we can get the hidden layer output matrix $\mathbf{H}$ from $\mathbf{X}$. Thus, we have $(\mathbf{H},\mathbf{T})$.

The expired training dataset $\overline{\Delta}\mathcal{N}=\{(\mathbf{x}_j,\ \mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\ \mathbf{t}_j\in\mathbb{R}^m,\ j=N-\overline{\Delta}N+1,\ N-\overline{\Delta}N+2,\ldots,N-1,N\}$ is denoted as $\overline{\Delta}\mathcal{N}=(\overline{\Delta}\mathbf{X},\ \overline{\Delta}\mathbf{T})$. Then, according to Eq. (4), we can get the hidden layer output matrix $\overline{\Delta}\mathbf{H}$ from $\overline{\Delta}\mathbf{X}$. Thus, we have $(\overline{\Delta}\mathbf{H},\overline{\Delta}\mathbf{T})$.

Now, the remaining training dataset after $\overline{\Delta}\mathcal{N}$ expired is $\mathcal{N}^-=(\mathbf{X}-\overline{\Delta}\mathbf{X},\ \mathbf{T}-\overline{\Delta}\mathbf{T})=\{(\mathbf{x}_j,\ \mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\ \mathbf{t}_j\in\mathbb{R}^m,\ j=1,2,\ldots,N-\overline{\Delta}N\}$, The remaining hidden layer output matrix $\mathbf{H}^-$ can be easily derived from $\mathbf{H}$ and $\overline{\Delta}\mathbf{H}$.

Thus, we have $(\mathbf{H}^-,\mathbf{T}^-)$, where $\mathbf{H}=[\frac{\mathbf{H}^-}{\overline{\Delta}\mathbf{H}}]$, $\mathbf{T}=[\frac{\mathbf{T}^-}{\overline{\Delta}\mathbf{T}}]$.

Then, according to the matrix multiplication operator, we have

$$\mathbf{H}^T\mathbf{H}=\begin{bmatrix}\mathbf{H}^-\\ \overline{\Delta}\mathbf{H}\end{bmatrix}^T\begin{bmatrix}\mathbf{H}^-\\ \overline{\Delta}\mathbf{H}\end{bmatrix}=[\mathbf{H}^{-T}\ \overline{\Delta}\mathbf{H}^T]\begin{bmatrix}\mathbf{H}^-\\ \overline{\Delta}\mathbf{H}\end{bmatrix}=\mathbf{H}^{-T}\mathbf{H}^-+\overline{\Delta}\mathbf{H}^T\overline{\Delta}\mathbf{H}$$

Thus,

$$\mathbf{H}^{-T}\mathbf{H}^-=\mathbf{H}^T\mathbf{H}-\overline{\Delta}\mathbf{H}^T\overline{\Delta}\mathbf{H} \tag{17}$$

$$\mathbf{H}^T\mathbf{T}=\begin{bmatrix}\mathbf{H}^-\\ \overline{\Delta}\mathbf{H}\end{bmatrix}^T\begin{bmatrix}\mathbf{T}^-\\ \overline{\Delta}\mathbf{T}\end{bmatrix}=[\mathbf{H}^{-T}\ \overline{\Delta}\mathbf{H}^T]\begin{bmatrix}\mathbf{T}^-\\ \overline{\Delta}\mathbf{T}\end{bmatrix}=\mathbf{H}^{-T}\mathbf{T}^-+\overline{\Delta}\mathbf{H}^T\ \overline{\Delta}\mathbf{T}$$

Thus,

$$\mathbf{H}^{-T}\mathbf{T}^-=\mathbf{H}^T\mathbf{T}-\overline{\Delta}\mathbf{H}^T\overline{\Delta}\mathbf{T} \tag{18}$$

Let $\mathbf{U}^-=\mathbf{H}^{-T}\mathbf{H}^-$ and $\overline{\Delta}\mathbf{U}=\overline{\Delta}\mathbf{H}^T\overline{\Delta}\mathbf{H}$, we can get $\mathbf{U}^-=\mathbf{U}-\overline{\Delta}\mathbf{U}$. Similar to Eq. (11), the element $\overline{\delta}u_{ij}$ of matrix $\overline{\Delta}\mathbf{U}$ can be expressed as follows:

$$\overline{\delta}u_{ij}=\sum_{k=N-\overline{\Delta}N+1}^{N}g(\mathbf{w}_i\cdot\mathbf{x}_k+b_i)g(\mathbf{w}_j\cdot\mathbf{x}_k+b_j) \tag{19}$$

Let $\mathbf{V}^-=\mathbf{H}^{-T}\mathbf{T}^-$ and $\overline{\Delta}\mathbf{V}=\overline{\Delta}\mathbf{H}^T\overline{\Delta}\mathbf{T}$, we can get $\mathbf{V}^-=\mathbf{V}-\overline{\Delta}\mathbf{V}$. Similar to Eq. (12), the element $\overline{\delta}v_{ij}$ of matrix $\overline{\Delta}\mathbf{V}$ can be expressed as follows:

$$\overline{\delta}v_{ij}=\sum_{k=N-\overline{\Delta}N+1}^{N}g(\mathbf{w}_i\cdot\mathbf{x}_k+b_i)t_{kj} \tag{20}$$

According to Eqs. (19) and (20), the process of calculating $\overline{\Delta}\mathbf{U}$ and $\overline{\Delta}\mathbf{V}$ is both decomposable. Thus, we can use MapReduce framework to speedup their computation, and the process of calculating matrices $\overline{\Delta}\mathbf{U}$ and $\overline{\Delta}\mathbf{V}$ based on MapReduce framework is similar to Algorithm 1. Then, the process of Decremental Learning (DL) approach in E^{2}LM is shown in Algorithm 4. Firstly, we get the preserved matrices $\mathbf{U}$ and $\mathbf{V}$ (Line 1) and the L pairs of hidden node parameters $(\mathbf{w}_i,b_i)$ (Lines 2–3). Next, we calculate matrices $\overline{\Delta}\mathbf{U}$ and $\overline{\Delta}\mathbf{V}$ according to the preserved parameters using Algorithm 1 (Line 4). And then, update the corresponding matrices $\mathbf{U}$ and $\mathbf{V}$ using $\overline{\Delta}\mathbf{U}$ and $\overline{\Delta}\mathbf{V}$ (Line 5). Finally, using Eq. (10), we solve the new output weight vector β (Line 6). The E^{2}LM decremental learning process for expired training dataset $\overline{\Delta}\mathcal{N}$ finishes after the complement of the training process above, and then we can predict the outputs of other data according to Eq. (9).

Algorithm 4. Decremental Learning.

1 Get the preserved matrices $\mathbf{U}$ and $\mathbf{V}$;
2 **for** $i=1$ to L **do**
3 Get the preserved hidden node parameters $(\mathbf{w}_i,b_i)$;
4 Calculate $\overline{\Delta}\mathbf{U}=\overline{\Delta}\mathbf{H}^T\overline{\Delta}\mathbf{H}$, $\overline{\Delta}\mathbf{V}=\overline{\Delta}\mathbf{H}^T\overline{\Delta}\mathbf{T}$ with MapReduce;
5 Update the matrices $\mathbf{U}=\mathbf{U}-\overline{\Delta}\mathbf{U}$ and $\mathbf{V}=\mathbf{V}-\overline{\Delta}\mathbf{V}$;
6 Calculate the new output weight vector $\boldsymbol{\beta}=(\mathbf{I}/\lambda+\mathbf{U})^{-1}\mathbf{V}$;

3.3. Correctional learning

The original training dataset $\mathcal{N}=\{(\mathbf{x}_j,\mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\mathbf{t}_j\in\mathbb{R}^m,j=1,2,\ldots,N\}$ is denoted as $\mathcal{N}=(\mathbf{X},\mathbf{T})$. Then, according to Eq. (4), we can get the hidden layer output matrix $\mathbf{H}$ from $\mathbf{X}$. Thus, we have $(\mathbf{H},\mathbf{T})$.

The error training dataset $\tilde{\Delta}\mathcal{N}=\{(\mathbf{x}_j,\ \mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\ \mathbf{t}_j\in\mathbb{R}^m,\ j=N-\tilde{\Delta}N+1,N-\tilde{\Delta}N+2,\ldots,N-1,N\}$ (Denoted as $\tilde{\Delta}\mathcal{N}=(\tilde{\Delta}\mathbf{X},\ \tilde{\Delta}\mathbf{T})$) is corrected to training dataset $\tilde{\Delta}\mathcal{N}'=\{(\mathbf{x}'_j,\ \mathbf{t}'_j)|\mathbf{x}'_j\in\mathbb{R}^n,\ \mathbf{t}'_j\in\mathbb{R}^m,\ j=N-\tilde{\Delta}N+1,N-\tilde{\Delta}N+2,\ldots,N-1,N\}$ (Denoted as $\tilde{\Delta}\mathcal{N}'=(\tilde{\Delta}\mathbf{X}',\ \tilde{\Delta}\mathbf{T}')$). Then, according to Eq. (4), we can get the hidden layer output matrix $\tilde{\Delta}\mathbf{H}$ from $\tilde{\Delta}\mathbf{X}$. Thus, we have $(\tilde{\Delta}\mathbf{H},\tilde{\Delta}\mathbf{T})$. Similarly, we have $(\tilde{\Delta}\mathbf{H}',\tilde{\Delta}\mathbf{T}')$.

Now, the new training dataset after $\tilde{\Delta}\mathcal{N}$ corrected to $\tilde{\Delta}\mathcal{N}'$ is $\mathcal{N}'=(\mathbf{X}-\tilde{\Delta}\mathbf{X}\ +\tilde{\Delta}\mathbf{X}',\mathbf{T}-\tilde{\Delta}\mathbf{T}\ +\tilde{\Delta}\mathbf{T}')=\{(\mathbf{x}_j,\ \mathbf{t}_j)|\mathbf{x}_j\in\mathbb{R}^n,\ \mathbf{t}_j\in\mathbb{R}^m,\ j=1,2,\ldots,N-\tilde{\Delta}N\}\cup\{(\mathbf{x}'_j,\mathbf{t}'_j)|\mathbf{x}'_j\in\mathbb{R}^n,\ \mathbf{t}'_j\in\mathbb{R}^m,\ j=N-\tilde{\Delta}N+1,N-\tilde{\Delta}N+2\ldots,N-1,N\}$, The remaining hidden layer output matrix $\mathbf{H}'$ can be easily derived from $\mathbf{H}$, $\tilde{\Delta}\mathbf{H}$ and $\tilde{\Delta}\mathbf{H}'$.

Thus, we have $(\mathbf{H}',\mathbf{T}')$, where

$$\mathbf{H}=\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}\end{bmatrix},\quad \mathbf{T}=\begin{bmatrix}\mathbf{T}^-\\ \tilde{\Delta}\mathbf{T}\end{bmatrix},\quad \mathbf{H}'=\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}'\end{bmatrix},\quad \mathbf{T}'=\begin{bmatrix}\mathbf{T}^-\\ \tilde{\Delta}\mathbf{T}'\end{bmatrix}.$$

Then, according to the matrix multiplication operator, we have

$$\mathbf{H}^T\mathbf{H}=\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}\end{bmatrix}^T\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}\end{bmatrix}=[\mathbf{H}^{-T}\ \tilde{\Delta}\mathbf{H}^T]\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}\end{bmatrix}=\mathbf{H}^{-T}\mathbf{H}^-+\tilde{\Delta}\mathbf{H}^T\tilde{\Delta}\mathbf{H}$$

$$\mathbf{H}'^T\mathbf{H}'=\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}'\end{bmatrix}^T\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}'\end{bmatrix}=[\mathbf{H}^{-T}\ \tilde{\Delta}\mathbf{H}'^T]\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}'\end{bmatrix}=\mathbf{H}^{-T}\mathbf{H}^-+\tilde{\Delta}\mathbf{H}'^T\tilde{\Delta}\mathbf{H}'$$

Thus,

$$\mathbf{H}'^T\mathbf{H}'=\mathbf{H}^T\mathbf{H}-\tilde{\Delta}\mathbf{H}^T\tilde{\Delta}\mathbf{H}+\tilde{\Delta}\mathbf{H}'^T\tilde{\Delta}\mathbf{H}' \tag{21}$$

$$\mathbf{H}^T\mathbf{T}=\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}\end{bmatrix}^T\begin{bmatrix}\mathbf{T}^-\\ \tilde{\Delta}\mathbf{T}\end{bmatrix}=[\mathbf{H}^{-T}\ \tilde{\Delta}\mathbf{H}^T]\begin{bmatrix}\mathbf{T}^-\\ \tilde{\Delta}\mathbf{T}\end{bmatrix}=\mathbf{H}^{-T}\mathbf{T}^-+\tilde{\Delta}\mathbf{H}^T\tilde{\Delta}\mathbf{T}$$

$$\mathbf{H}'^T\mathbf{T}'=\begin{bmatrix}\mathbf{H}^-\\ \tilde{\Delta}\mathbf{H}'\end{bmatrix}^T\begin{bmatrix}\mathbf{T}^-\\ \tilde{\Delta}\mathbf{T}'\end{bmatrix}=[\mathbf{H}^{-T}\ \tilde{\Delta}\mathbf{H}'^T]\begin{bmatrix}\mathbf{T}^-\\ \tilde{\Delta}\mathbf{T}'\end{bmatrix}=\mathbf{H}^{-T}\mathbf{T}^-+\tilde{\Delta}\mathbf{H}'^T\tilde{\Delta}\mathbf{T}'$$

Thus,

$$\mathbf{H}'^T\mathbf{T}'=\mathbf{H}^T\mathbf{T}-\tilde{\Delta}\mathbf{H}^T\tilde{\Delta}\mathbf{T}+\tilde{\Delta}\mathbf{H}'^T\tilde{\Delta}\mathbf{T}' \tag{22}$$

Let $\mathbf{U}'=\mathbf{H}'^T\mathbf{H}'$, $\tilde{\Delta}\mathbf{U}=\tilde{\Delta}\mathbf{H}^T\tilde{\Delta}\mathbf{H}$ and $\tilde{\Delta}\mathbf{U}'=\tilde{\Delta}\mathbf{H}'^T\tilde{\Delta}\mathbf{H}'$, we can get $\mathbf{U}'=\mathbf{U}-\tilde{\Delta}\mathbf{U}+\tilde{\Delta}\mathbf{U}'$. Similar to Eq. (11), the elements $\tilde{\delta}u_{ij}$ and $\tilde{\delta}u'_{ij}$ of matrices $\tilde{\Delta}\mathbf{U}$ and $\tilde{\Delta}\mathbf{U}'$ can be expressed as follows:

$$\tilde{\delta}u_{ij}=\sum_{k=N-\tilde{\Delta}N+1}^{N}g(\mathbf{w}_i\cdot\mathbf{x}_k+b_i)g(\mathbf{w}_j\cdot\mathbf{x}_k+b_j) \tag{23}$$

$$\tilde{\delta}u'_{ij}=\sum_{k=N-\tilde{\Delta}N+1}^{N}g(\mathbf{w}_i\cdot\mathbf{x}'_k+b_i)g(\mathbf{w}_j\cdot\mathbf{x}'_k+b_j) \tag{24}$$

Let $\mathbf{V}'=\mathbf{H}'^T\mathbf{T}'$, $\tilde{\Delta}\mathbf{V}=\tilde{\Delta}\mathbf{H}^T\tilde{\Delta}\mathbf{T}$ and $\tilde{\Delta}\mathbf{V}'=\tilde{\Delta}\mathbf{H}'^T\tilde{\Delta}\mathbf{T}'$, we can get $\mathbf{V}'=\mathbf{V}-\tilde{\Delta}\mathbf{V}+\tilde{\Delta}\mathbf{V}'$. Similar to Eq. (12), the elements $\tilde{\delta}v_{ij}$ and $\tilde{\delta}v'_{ij}$ of matrices $\tilde{\Delta}\mathbf{V}$ and $\tilde{\Delta}\mathbf{V}'$ can be expressed as follows:

$$\tilde{\delta}v_{ij}=\sum_{k=N-\tilde{\Delta}N+1}^{N}g(\mathbf{w}_i\cdot\mathbf{x}_k+b_i)t_{kj} \tag{25}$$

$$\tilde{\delta}v'_{ij}=\sum_{k=N-\tilde{\Delta}N+1}^{N}g(\mathbf{w}_i\cdot\mathbf{x}'_k+b_i)t'_{kj} \tag{26}$$

According to Eqs. (23), (24), (25) and (26), the process of calculating $\tilde{\Delta}\mathbf{U}$, $\tilde{\Delta}\mathbf{U}'$, $\tilde{\Delta}\mathbf{V}$ and $\tilde{\Delta}\mathbf{V}'$ are all decomposable. Thus, we can use MapReduce framework to speedup their computation, and the processes of calculating matrices $\tilde{\Delta}\mathbf{U}$ and $\tilde{\Delta}\mathbf{V}$, $\tilde{\Delta}\mathbf{U}'$ and $\tilde{\Delta}\mathbf{V}'$ based on MapReduce framework are both similar to Algorithm 1. Then, the process of Correctional Learning (CL) approach in E^{2}LM is shown in Algorithm 4. Firstly, we get the preserved matrices $\mathbf{U}$ and $\mathbf{V}$ (Line 1) and the L pairs of hidden node parameters $(\mathbf{w}_i,b_i)$ (Lines 2–3). Next, we calculate matrices $\tilde{\Delta}\mathbf{U}$ and $\tilde{\Delta}\mathbf{V}$ according to the preservedparameters using Algorithm 1 in one MapReduce job (Line 4), and calculate matrices $\tilde{\Delta}\mathbf{U}'$ and $\tilde{\Delta}\mathbf{V}'$ according to the preserved parameters using Algorithm 1 in another MapReduce job (Line 5). And then, update the

corresponding matrices $\mathbf{U}$ and $\mathbf{V}$ using $\tilde{\Delta}\mathbf{U}$, $\tilde{\Delta}\mathbf{U}'$, $\tilde{\Delta}\mathbf{V}$ and $\tilde{\Delta}\mathbf{V}'$ (Line 6). Finally, using Eq. (10), we solve the new output weight vector β (Line 7). The E^{2}LM correctional learning process for corrected training dataset $\tilde{\Delta}\mathcal{N}$ to $\tilde{\Delta}\mathcal{N}'$ finishes after the complement of the training process above, and then we can predict the outputs of other data according to Eq. (9).

Table 1
Experimental parameters.

Parameter	Range and default
Dimensionality	10, 20, 30, 40, **50**
Number of hidden nodes	100, 150, **200**, 250, 300
Number of records	3M(1.4G), 4M(1.86G), **5M**(2.3G), 6M(2.8G), 7M (3.27G)
Number of nodes	1, 2, 3, 4, 5, 6, 7, **8**
Update ratio	2%, 4%, 6%, 8%, **10**%

Algorithm 5. Correctional Learning.

1 Get the preserved matrices $\mathbf{U}$ and $\mathbf{V}$;
2 **for** $i=1$ to L **do**
3 Get the preserved hidden node parameters $(\mathbf{w}_i, b_i)$;
4 Calculate $\tilde{\Delta}\mathbf{U}=\tilde{\Delta}\mathbf{H}^T\tilde{\Delta}\mathbf{H}$, $\tilde{\Delta}\mathbf{V}=\tilde{\Delta}\mathbf{H}^T\tilde{\Delta}\mathbf{T}$ with MapReduce;
5 Calculate $\tilde{\Delta}\mathbf{U}'=\tilde{\Delta}\mathbf{H}'^T\tilde{\Delta}\mathbf{H}'$, $\tilde{\Delta}\mathbf{V}'=\tilde{\Delta}\mathbf{H}'^T\tilde{\Delta}\mathbf{T}'$ with MapReduce;
6 Update the matrices $\mathbf{U}=\mathbf{U}-\tilde{\Delta}\mathbf{U}+\tilde{\Delta}\mathbf{U}'$ and $\mathbf{V}=\mathbf{V}-\tilde{\Delta}\mathbf{V}+\tilde{\Delta}\mathbf{V}'$;
7 Calculate the new output weight vector $\boldsymbol{\beta}=(\mathbf{I}/\lambda+\mathbf{U})^{-1}\mathbf{V}$;

4. Performance evaluation

In this section, the performance of our Elastic ELM is evaluated in detail with various experimental settings. We first describe the platform used in our experiments in Section 4.1. Then we present and discuss the experimental results of the evaluation in Section 4.2.

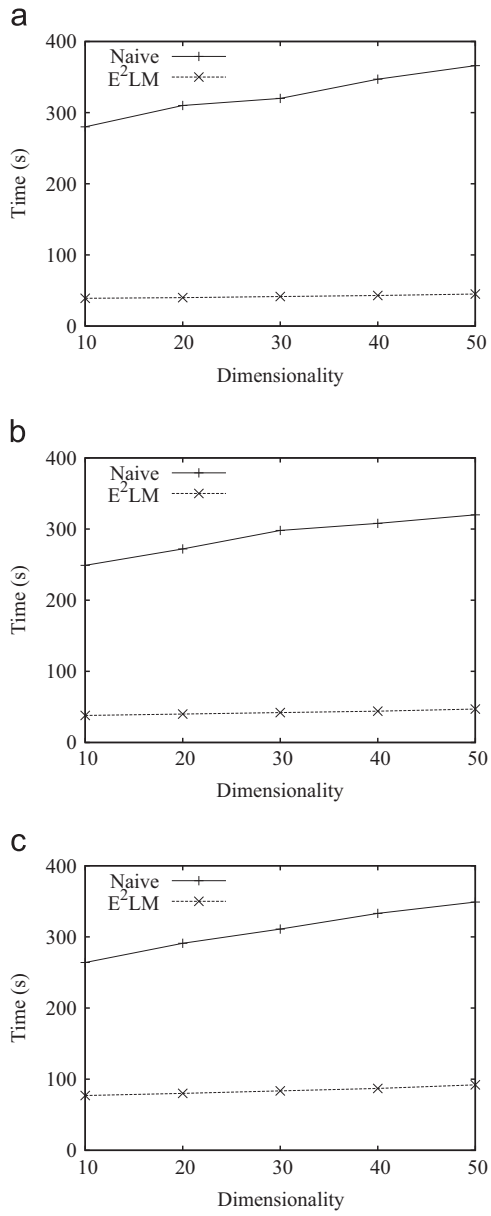

Fig. 1. The influence of dimensionality. (a) Incremental learning, (b) decremental learning, (c) correctional learning.

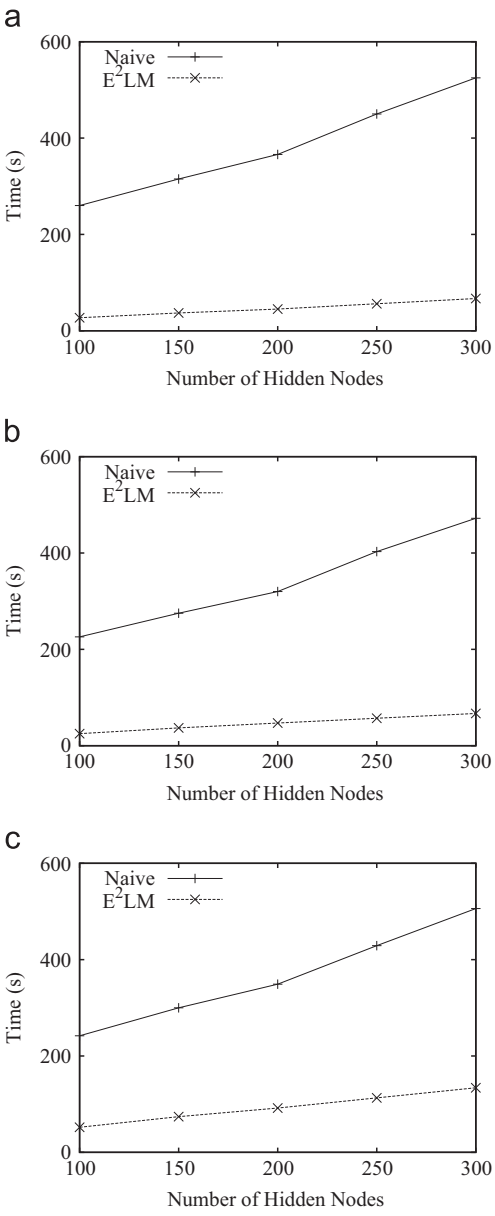

Fig. 2. The influence of number of hidden nodes. (a) Incremental learning, (b) decremental learning, (c) correctional learning.

4.1. Experimental platform

All of our experiments are run on a cluster with 9 computers which are connected in a high speed Gigabit network. Each computer has an Intel Quad Core 2.66GHz CPU, 4GB memory and CentOS Linux 5.6. One computer is set as the Master node and the others are set as the Slave nodes. We use Hadoop version 0.20.2 and configure it to run up to 4 map tasks or 4 reduce tasks concurrently per node. Therefore, at any point in time, at most 32 map tasks or 32 reduce tasks can run concurrently in our cluster. Table 1 summarizes the parameters used in our experimental evaluation, along with their ranges and default values shown in bold. In each experiment, we vary a single parameter, while setting the remainders to their default values. The followings are algorithms that have been evaluated.

- Naive: Retraining ELM* [28] with the whole training dataset.
- E^{2}LM: our proposed Elastic Extreme Learning Machine.

The activation function in the experimental evaluation is sigmoid (i.e. $g(x) = 1/(1+\exp(-x))$), and the value of λ is 1. Compared with Naive, E^{2}LM only improves the learning efficiency on the updated training dataset, but not the classification accuracy. As a result, the experiments mainly compare the training time of each algorithm, without examining the classification accuracy of the algorithm. Since the activation function and the value of λ have large influence on the classification accuracy and little influence on the training time, we hence did not test the influence of activation function and the value of λ to training time.

4.2. Experimental results

Firstly, we investigate the performances of Naive and E^{2}LM, when the dimensionality of training data changes. As shown in Fig. 1, with the changing of dimensionality, the running time of Naive and E^{2}LM all increase slightly. With the increase of dimensionality, the running time for calculating corresponding row in hidden layer output matrix for each data record increases slightly, and make the running time for MapReduce process increases slightly. While the number of hidden nodes does not change, the

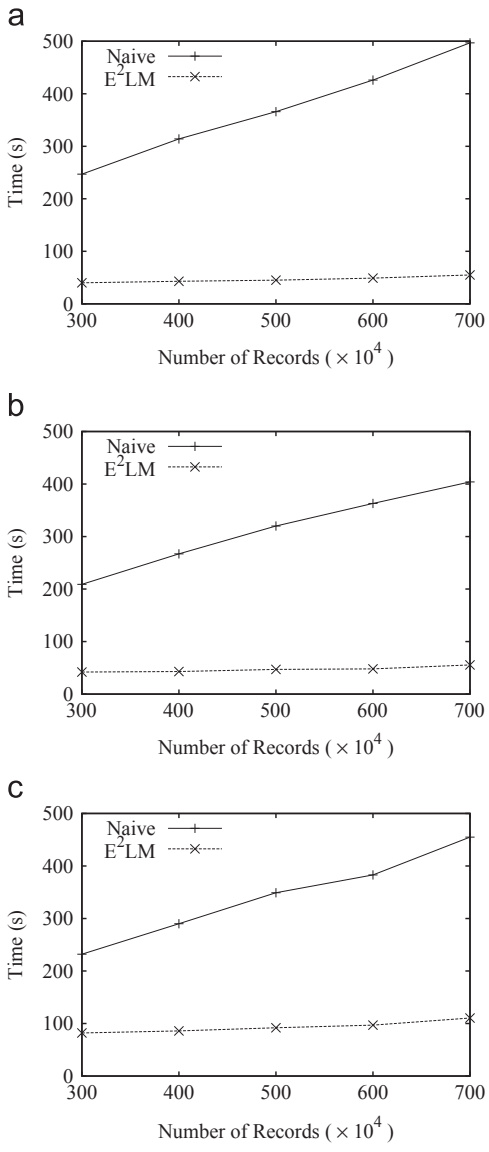

Fig. 3. The influence of number of records. (a) Incremental learning, (b) decremental learning, (c) correctional learning.

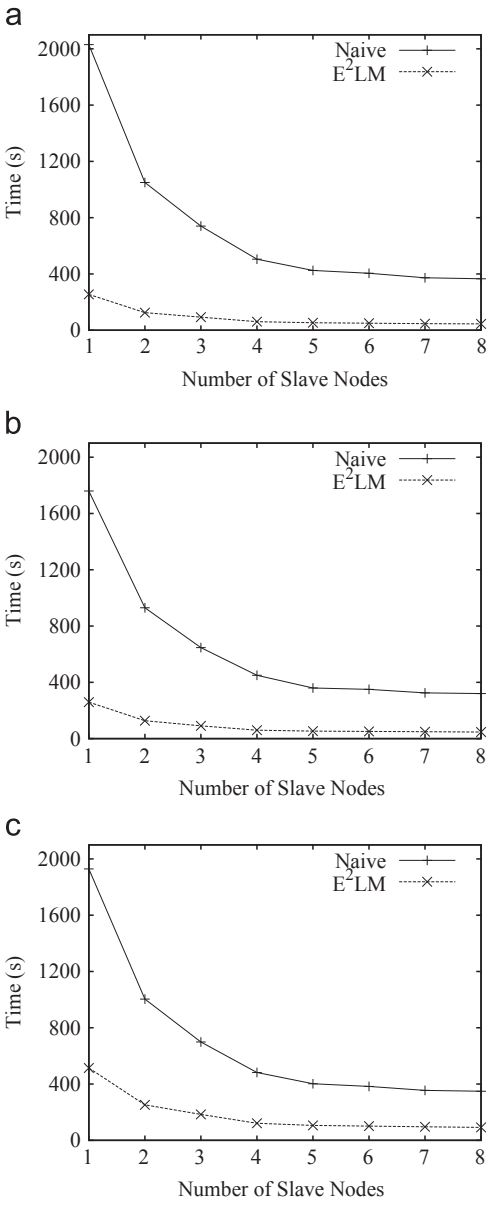

Fig. 4. The influence of number of slave nodes. (a) Incremental learning, (b) decremental learning, (c) correctional learning.

amount of data transmitting in the cluster has no apparent increase, and the time of the intermediate results transmitting in cluster does not also increase. Therefore, the training time increases slightly with the increase of dimensionality. It is obvious that the running time of E^{2}LM is shorter than Naive approach. Because the updated training dataset is much smaller than the whole one, times for calculating the updated parts of **U** and **V** are much shorter than recalculating the whole **U** and **V** directly. Therefore, the performance of E^{2}LM is greatly optimal to Naive approach.

Secondly, we consider the influence of the number of hidden nodes. As illustrated in Fig. 2, with the number of hidden nodes increases, the running time increases. When the number of hidden nodes increases, the size of matrices **U** and **V** is also enlarged. That is to say, the amount of intermediate results of MapReduce is enlarged. This does not only increases the computing time of MapReduce job, but also increases the transmitting time of the intermediate results in MapReduce cluster. Therefore, the training time increases with the increase of the number of hidden nodes. The relationship between the performances of Naive and E^{2}LM is similar to those shown in Fig. 1, which indicates the effectiveness of our E^{2}LM.

Then, we investigate the influence of the number of training records. Fig. 3 shows that with the increase of the number of training records, the running time increases. As the number of training records increases, the total amount of data that MapReduce needs to process is enlarged, which increases both the computing time of MapReduce job and the transmitting time of the intermediate results in MapReduce cluster. Therefore, the training time increases with the increase of the number of training records. Similar to the influence of the number of hidden nodes, the running time of E^{2}LM is shorter than Naive approach, which further proves the effectiveness of our E^{2}LM.

Next, we consider the influence of the number of working slave nodes in MapReduce Cluster. Fig. 4 shows that with the number of working slave nodes increases, the running time decreases rapidly. And it is reasonable that the increasing amount of working slave nodes makes the tasks that can be run by Map and Reduce increase at the same time. As a result, it improves the parallelism of computing and process efficiency. Therefore, the training time decreases when the amount of working slave nodes increases. The same, the running time of E^{2}LM is always shorter than Naive approach.

Finally, we discuss the influence of the training data update ratio. As shown in Fig. 5, while the training data update ratio increases, the running time of E^{2}LM increases slightly. As training data update ratio increases, the updated dataset is enlarged, so the running time of E^{2}LM is increased. The running time of Naive approach increases slightly in incremental learning, decreases slightly in decremental learning, and it is stable in correctional learning because the amounts of data evolved are increase, decremental and stable respectively. As a matter of course, the running time of E^{2}LM is always shorter than Naive.

5. Conclusions

In order to overcome the weakness of ELM* in learning updated large-scale training dataset, we propose a novel Elastic Extreme Learning Machine (E^{2}LM) based on MapReduce framework. E^{2}LM can support incremental learning, decremental learning and correctional learning of large scale training dataset. Specifically, through analyzing the character of ELM*, we found that the most computation-expensive part, matrix multiplication, can be incrementally, decrementally and correctionally calculated. Then, we present the computational details of E^{2}LM. E^{2}LM first calculates the intermediate matrix multiplications of the updated training data subset. Next, it updates the matrix multiplications by modifying the old matrix multiplications with the intermediate ones. Then, the corresponding new output weight vector can be obtained with centralized computing using the update the matrix multiplications. Finally, in the Cluster environment,

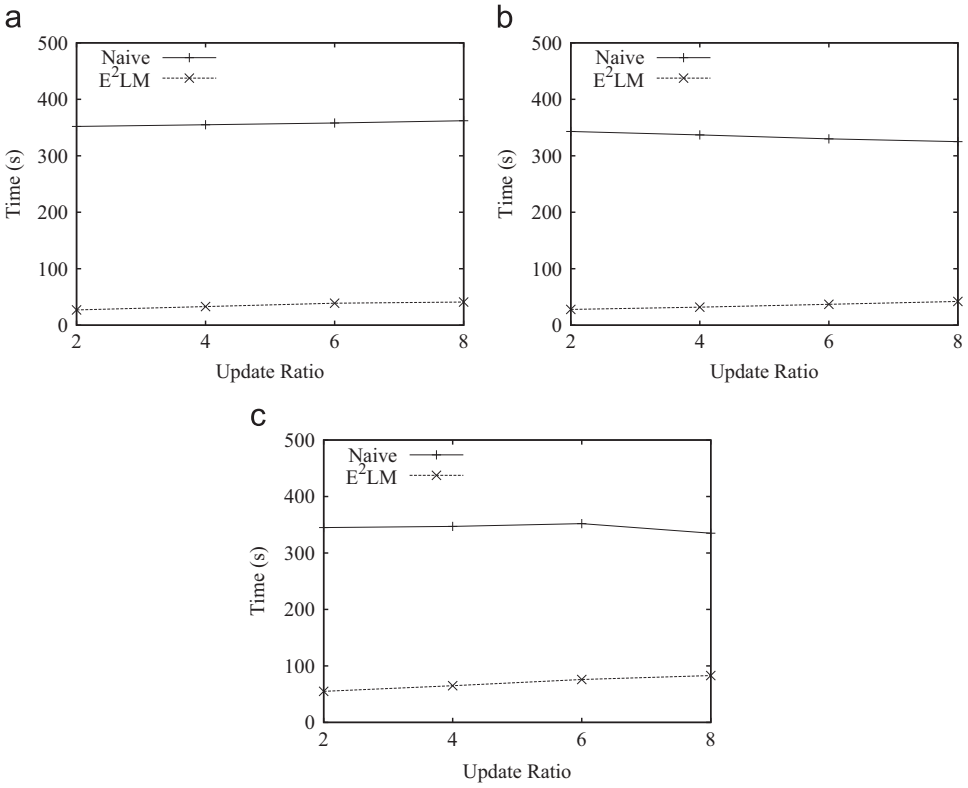

Fig. 5. The influence of update ratio. (a) Incremental learning, (b) decremental learning, (c) correctional learning.

we use synthetic data to do a detailed validation of the effectiveness and efficiency of E^{2}LM. The experiment results show that E^{2}LM can efficiently learn the rapid updated massive training dataset in big data classification applications.

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China under Grant nos. 60933001, 61025007, 61100022 and 61472069; the National Basic Research Program of China under Grant no. 2011CB302200-G; the 863 Program under Grant no. 2012AA011004, and the Fundamental Research Funds for the Central Universities under Grant no. N110404009.

References

[1] S. Wang, H. Wang, X. Qin, X. Zhou, Architecting big data: challenges, studies and forecasts, Chin. J. Comput. 34 (10) (2011) 1741–1752.
[2] A.B. Patel, M. Birla, U. Nair, Addressing big data problem using hadoop and map reduce, in: Proceedings of the Nirma University International Conference on Engineering (NUiCONE 2012) December 6–8, 2012, pp. 1–5.
[3] R.T. Kaushik, K. Nahrstedt, T*: A data-centric cooling energy costs reduction approach for big data analytics cloud, in: Proceedings of the High Performance Computing, Networking, Storage and Analysis (SC 2012), Salt Lake City, Utah, USA, November 10–16, 2012, pp. 1–11.
[4] D. Garlasu, A big data implementation based on grid computing, in: Proceedings of the Roedunet International Conference (RoEduNet 2013), January 17–19, 2013, pp. 1–4.
[5] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.
[6] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892.
[7] G.-B. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (16–18) (2007) 3056–3062.
[8] G.-B. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neurocomputing 71 (16–18) (2008) 3460–3468.
[9] G.-B. Huang, X. Ding, H. Zhou, Optimization method based extreme learning machine for classification, Neurocomputing 74 (1–3) (2010) 155–163.
[10] G.-B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trans. Syst. Man Cybern. B: Cybern. 42 (2) (2012) 513–529.
[11] G.-B. Huang, D.H. Wang, Y. Lan, Extreme learning machines: a survey, Int. J. Mach. Learn. Cybern. 2 (2) (2011) 107–122.
[12] R. Zhang, G.-B. Huang, N. Sundararajan, P. Saratchandran, Multi-category classification using an extreme learning machine for microarray gene expression cancer diagnosis, IEEE/ACM Trans. Comput. Biol. Bioinformatics 4 (3) (2007) 485–495.
[13] Q.-Y. Zhu, A.K. Qin, P.N. Suganthan, G.-B. Huang, Evolutionary extreme learning machine, Pattern Recognit. 38 (10) (2005) 1759–1763.
[14] G. Wang, Y. Zhao, D. Wang, A protein secondary structure prediction framework based on the extreme learning machine, Neurocomputing 72 (1–3) (2008) 262–268.
[15] X. Zhao, G. Wang, X. Bi, P. Gong, Y. Zhao, XML document classification based on elm, Neurocomputing 74 (16) (2011) 2444–2451.
[16] Y. Sun, Y. Yuan, G. Wang, An os-elm based distributed ensemble classification framework in p2p networks, Neurocomputing 74 (16) (2011) 2438–2443.
[17] B. Wang, G. Wang, J. Li, B. Wang, Update strategy based on region classification using elm for mobile object index, Soft Computing, no. http://dx.doi.org/10.1007/s00500-012-0821-9, 2012, pp. 2444–2451.
[18] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, N. Sundararajan, On-line sequential extreme learning machine, in: Proceedings of the IASTED International Conference on Computational Intelligence (CI 2005), Calgary, Alberta, Canada, July 4–6, 2005, pp. 232–237.
[19] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate on-line sequential learning algorithm for feedforward networks, IEEE Trans. Neural Netw. 17 (6) (2006) 1411–1423.
[20] H.-J. Rong, G.-B. Huang, N. Sundararajan, P. Saratchandran, On-line sequential fuzzy extreme learning machine for function approximation and classification problems, IEEE Trans. Syst. Man Cybern. B 39 (4) (2009) 1067–1072.
[21] X. Wang, Q. Shao, Q. Miao, J. Zhai, Architecture selection for networks trained with extreme learning machine using localized generalization error model, Neurocomputing 102 (1) (2013) 3–9.
[22] J. Zhai, H. Xu, X. Wang, Dynamic ensemble extreme learning machine based on sample entropy, Soft Comput. 16 (9) (2012) 1493–1502.
[23] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, OP-ELM: Optimally pruned extreme learning machine, IEEE Trans. Neural Netw. 21 (1) (2010) 158–162.
[24] Y. Miche, M. van Heeswijk, P. Bas, O. Simula, A. Lendasse, Trop-elm: a double-regularized ELM using LARS and tikhonov regularization, Neurocomputing 74 (16) (2011) 2413–2421.
[25] H.-J. Rong, Y.-X. Jia, G.-S. Zhao, Aircraft recognition using modular extreme learning machine, Neurocomputing 128 (2014) 166–174.
[26] B. Xu, Y. Pan, D. Wang, F. Sun, Discrete-time hypersonic flight control based on extreme learning machine, Neurocomputing 128 (2014) 232–241.
[27] Q. He, T. Shang, F. Zhuang, Z. Shi, Parallel extreme learning machine for regression based on mapreduce, Neurocomputing 102 (2) (2013) 52–58.
[28] J. Xin, Z. Wang, C. Chen, L. Ding, G. Wang, Y. Zhao, ELM*: distributed extreme learning machine with mapreduce, World Wide Web, 2013, pp. 1–16.
[29] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: Proceedings of Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6–8, 2004, pp. 137–150.
[30] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[31] J. Dean, S. Ghemawat, MapReduce: a flexible data processing tool, Commun. ACM 53 (1) (2010) 72–77.

Junchang Xin received B.Sc., M.Sc. and Ph.D. in computer science and technology from the Northeastern University, P.R. China, in July 2002, March 2005, and July 2008, respectively. He visited the National University of Singapore (NUS) as post-doctoral visitor (April 2010 – April 2011). He is currently an associate professor in the Department of Computer Science, Northeastern University, P.R. China. His research interests include big data, cloud computing, data management over wireless sensor network and uncertain data management. He has published more than 40 research papers.

Zhiqiong Wang received her Master's degree in applied computer technology (Computer Applications Technology) from Northeastern University in 2008. She visited the National University of Singapore (NUS) in 2010 and the Chinese University of Hong Kong (CUHK) in 2013 as the academic visitor. Currently, she is a lecturer of the Sino-Dutch Biomedical and Information Engineering School of Northeastern University, and a Ph.D. candidate of computer software and theory in Northeastern University. Her main research interests are: biomedical, biological data processing, cloud computing, and machine learning. She has published more than 20 papers in journals and conferences.

Luxuan Qu received her Bachelor's degree in automatic from the Northeast Dianli University in 2010. Currently, she is a master student of the Sino-Dutch Biomedical and Information Engineering School of Northeastern University. Her main research interests are: biological data processing, machine learning, and cloud computing.

Guoren Wang received B.Sc., M.Sc. and Ph.D. from the Department of Computer Science, Northeastern University, P.R. China, in 1988, 1991 and 1996, respectively. He worked as a post-doctor in Kyushu University, Japan (March 1996 – March 1997). He visited several universities in the world as research visitor and visiting professor, including Kyushu University, Japan (September 1997), the Chinese University of Hong Kong, China (January 2001, February 2004), China and Carleton University, Canada (September 2003 – July 2003). Currently, he is a professor in the Department of Computer Science, Northeastern University, P.R. China. His research interests are XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems, and P2P data management. He has published more than 100 research papers.