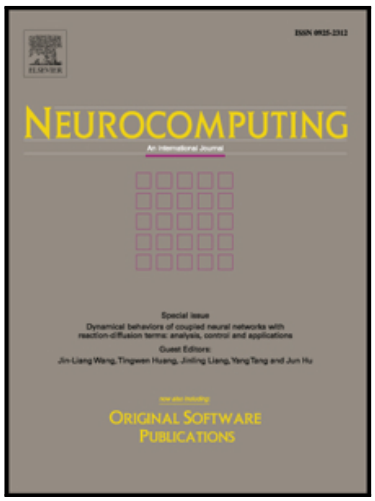


Distributed Extreme Learning Machine with Alternating Direction Method of Multiplier

Minnan Luo, Lingling Zhang, Jun Liu, Jun Guo, Qinghua Zheng

PII: S0925-2312(17)30204-7
DOI: 10.1016/j.neucom.2016.03.112
Reference: NEUCOM 18003

To appear in: *Neurocomputing*

Received date: 26 September 2015
Revised date: 16 March 2016
Accepted date: 22 March 2016

Please cite this article as: Minnan Luo, Lingling Zhang, Jun Liu, Jun Guo, Qinghua Zheng, Distributed Extreme Learning Machine with Alternating Direction Method of Multiplier, *Neurocomputing* (2017), doi: 10.1016/j.neucom.2016.03.112

Distributed Extreme Learning Machine with Alternating Direction Method of Multiplier

Minnan Luo*[1], Lingling Zhang[1], Jun Liu[1], Jun Guo[2], Qinghua Zheng[1]

[1]*SPKLSTN Lab,*
Department of Computer Science, Xian Jiaotong University, Xi'an, China, 710049

[2]*Hardware Department,*
School of Computer Science and Technology, Northwest University, Xi'an, China, 710127

Abstract

Extreme learning machine, as a generalized single-hidden-layer feedforward network, has achieved much attention for its extremely fast learning speed and good generalization performance. However, big data often makes a challenge in large scale learning of extreme learning machine due to the memory limitation of single machine as well as the distributed manner of large scale data in many applications. For the purpose of relieving the limitation of memory with big data, in this paper, we exploit a novel distributed model to implement the extreme learning machine algorithm in parallel for large-scale data set, namely Distributed Extreme Learning Machine (DELM). A corresponding algorithm is developed on the basis of alternating direction method of multipliers which has shown its effectiveness in distributed convex optimization. Finally, extensive experiments on some benchmark data sets are carried out to illustrate the effectiveness and superiority of the proposed DELM method with an analysis on the performance of speedup, scaleup and sizeup.

Keywords: Extreme learning machine, neurone work, alternating direction method of multiplier

1. Introduction

Extreme learning machine is a generalized single-hidden-layer feedforward network, where the parameters of hidden layer feature mapping are generated randomly according to any continuous probability distribution [1] instead of being tuned by gradient descent based algorithms. As a result, extreme learning machine achieves extremely fast learning speed and better performance of generalization. The ELM technique performs effectively and have been applied in many applications of machine learning such as classification [2, 3], clustering [4] and regression [5]. C. W. Deng and G. B. Huang highlighted the new trends of multi-layer learning with extreme learning machine [6]. Extreme learning machine is also used in many real life applications, for example, S. Shamshirband and et. al use extreme learning machine to estimate the wind speed distribution [7]; Deng et. al proposed an efficient image super-resolution approach based on extreme learning machine to reconstruct the high-frequency components containing details [8].

It is noteworthy that traditional extreme learning machine is often implemented on a single machine, and therefore it is inevitable to suffer from the limitation of memory with large scale data set. Especially in the era of big data, the data set scale is usually extremely large and the data is often very high-dimensional for detailed information [9, 10]. On the other hand, it is actually necessary to deal with the data set in different machines due to the following two reasons: (1) The data set is stored and collected in a distributed manner because of the large scale of applications; (2) It is impossible to collect all of data together for the reason of confidentiality and the data set can be only accessed on their own machine. Based on the analysis above, how to implement extreme learning machine with respect to the data set which located in different machines becomes a key problems.

In previous work, some parallel or distributed ex-

Email addresses: minnluo@mail.xjtu.edu.cn (Minnan Luo*[1]), zhanglingling@stu.xjtu.edu.cn (Lingling Zhang[1]), liukeen@mail.xjtu.edu.cn (Jun Liu[1]), guojun@nwu.edu.cn (Jun Guo[2]), qhzheng@mail.xjtu.edu.cn (Qinghua Zheng[1])

treme learning machine have been implemented to meet the challenge of large-scale data set [11, 12]. For example, Q. He and et. al took advantages of the distributed environment provided by MapReduce [13] and propose an parallel extreme learning machine on the basis of MapReduce via designing the proper $\langle key, value\rangle$ pairs [14]. X. Wang and et. al focused on the issue of parallel ELM and propose M^3 extreme learning machine on the basis of min-max modular network, namely as [15]. This approach decomposes the classification problem into several small subproblems and trains individual ELM for each subproblem; in the end, M^3-network is adopted to ensemble the individual classifiers together. Additionally, A. Akusok et. al exploited a complete approach which successfully utilize high-performance extreme learning machine toolbox for big data [16].

Besides the distributed framework mentioned above, alternating direction method of multipliers (ADMM) that aims to find a solution to global problem by solving local subproblems coordinately, is also an effective optimization method for distributed convex optimization [9, 17]. It is noteworthy that C. Zhang investigated a large-scale distributed linear classification algorithm in the framework of ADMM and achieves a significant speedup over some other classifiers [18]. C.Y. Lu et. al proposed a fast proximal ADMM with parallel splitting method to further reduce the per-iteration complexity for multi-blocks problem [19].

Motivated by the advantages of ADMM in distributed optimization problem, in this paper, we exploit a novel distributed extreme learning machine (DELM) to implement the extreme learning machine on multiple machines in parallel. This approach relieve the limitation of memory with large scale data set to some extent. It is different from the traditional extreme learning machine, where all of data are loaded onto one processor and share a common output weight vector of hidden layer. Instead, the proposed DELM method allows large-scale data set to be stored in distributed manner by associating each processor a output weight vector, where each output weight vector can be determined in parallel since it depends only on the corresponding sub-dataset. In the framework of ADMM, the shared output weight vector across all processors is derived by combining all output weight vectors with an included regularization vector.

The remainder of this paper is organized as follows. In Section 2, notations and preliminaries about extreme learning machine are reviewed briefly. Section 3 focuses on the formulated optimization problem of distributed extreme learning machine. A corresponding distributed convex optimization algorithm is developed for the proposed DELM on the basis of ADMM in Section 4. Extensive experiments on well-known benchmark data sets are conducted in Section 5 to illustrate the effectiveness and superiority of the proposed method. Conclusion is given in Section 6.

2. Principles of extreme learning machine

Extreme learning machine refer to a kind of single-hidden-layer feedforward neural network, where the hidden layer need not be tuned. Formally, the output function of extreme learning machine is formulated as

$$f_L(\mathbf{x}) = \sum_{j=1}^{L} \beta_j h_j(\mathbf{x}) = \mathbf{h}(\mathbf{x})\beta, \tag{1}$$

where $\beta = \left(\beta^1, \beta^2, \cdots, \beta^L\right)^\intercal \in \mathbb{R}^L$ is the output weights vector of the hidden layer with L nodes, which needs to be estimated analytically; Feature mapping $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^L$ maps input variable $\mathbf{x} \in \mathbb{R}^n$ to L-dimensional hidden-layer feature space such that

$$\mathbf{h}(\mathbf{x}) = \left(h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_L(\mathbf{x})\right), \tag{2}$$

where the component $h_j(\mathbf{x}) = G(\mathbf{a}_j, b_j, \mathbf{x})$ $(j = 1, 2, \cdots, L)$ denotes the output function of j-th hidden node, which is known to users by generating the parameters $\left\{\left(\mathbf{a}_j, b_j\right) : \ j = 1, 2, \cdots, L\right\}$ randomly according to any continuous probability distribution [20]. In general, Sigmoid function

$$G(\mathbf{a}, b, \mathbf{x}) = \frac{1}{1 + \exp\left(-\left(\mathbf{a}^\intercal \mathbf{x} + b\right)\right)},$$

Gaussian function

$$G(\mathbf{a}, b, \mathbf{x}) = \exp\left(-b\left\|\mathbf{x} - \mathbf{a}\right\|^2\right)$$

and some other nonlinear activation functions are usually adopted to generate the probability distribution in the framework of extreme learning machine. In addition, some kernel sparse coding methods are also developed for the requirement of many applications [21, 22].

For data set $D = \left\{\left(\mathbf{x}_k^\intercal, t_k\right) : \ \mathbf{x}_k \in \mathbb{R}^n, t_k \in \mathbb{R}, k = 1, 2, \cdots, N\right\}$ that consists of N training data points $\mathbf{x}_k$ with actual output t_k $(k = 1, 2, \cdots, N)$, extreme learning machine randomly generates input weights and estimates the output weight vector β by minimizing the training error with the l_2-norm of output weights for better generalization, i.e.,

$$\begin{aligned}\beta &= \arg\min_{\beta} \frac{1}{2}\left\|\beta\right\|_2^2 + \frac{C}{2}\sum_{k=1}^{N}\left(\mathbf{h}(\mathbf{x}_k)\beta - t_k\right)^2 \\ &= \arg\min_{\beta} \frac{1}{2}\left\|\beta\right\|_2^2 + \frac{C}{2}\left\|H\beta - T\right\|_2^2 \end{aligned} \tag{3}$$

where C is the trade-off parameter between the training error and the regularization; $T = (t_1, t_2, \cdots, t_N)^\intercal \in \mathbb{R}^N$ denotes the actual output vector; $H \in \mathbb{R}^{N\times L}$ represents the hidden layer output matrix, i.e.,

$$H = \begin{pmatrix} \mathbf{h}(\mathbf{x}_1) \\ \mathbf{h}(\mathbf{x}_2) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{pmatrix} = \begin{pmatrix} h_1(\mathbf{x}_1) & h_2(\mathbf{x}_1) & \cdots & h_L(\mathbf{x}_1) \\ h_1(\mathbf{x}_2) & h_2(\mathbf{x}_2) & \cdots & h_L(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_N) & h_2(\mathbf{x}_N) & \cdots & h_L(\mathbf{x}_N) \end{pmatrix}.$$

It is evident that the closed solution of optimization (3) can be achieved as

$$\beta = H^\intercal \left(\frac{I}{C} + HH^\intercal\right)^{-1} T, \tag{4}$$

and therefore the output of conventional extreme learning machine satisfies

$$f_L(\mathbf{x}) = \mathbf{h}(\mathbf{x}) H^\intercal \left(\frac{I}{C} + HH^\intercal\right)^{-1} T. \tag{5}$$

If the feature mapping $\mathbf{h}$ is not known to users, a kernel matrix $\Omega = \left(\Omega_{i,j}\right) \in \mathbb{R}^{N\times N}$ with respect to data set D can be calculated as

$$\Omega_{i,j} = h(\mathbf{x}_i) \cdot h\left(\mathbf{x}_j\right) = K\left(\mathbf{x}_i, \mathbf{x}_j\right),$$

where the kernel function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is usually defined as Gaussian function

$$K\left(\mathbf{x}_i, \mathbf{x}_j\right) = G\left(\mathbf{x}, \mathbf{x}_j, \sigma\right) = \exp\left(-\frac{\left\|\mathbf{x} - \mathbf{x}_j\right\|^2}{\sigma^2}\right).$$

According to equation (5), the output of extreme learning machine with kernel function K is formulated as

$$\begin{aligned} f_K(\mathbf{x}) &= \mathbf{h}(\mathbf{x}) H^\intercal \left(\frac{I}{C} + HH^\intercal\right)^{-1} T \\ &= \left(K(\mathbf{x}, \mathbf{x}_1), K(\mathbf{x}, \mathbf{x}_2), \cdots, K(\mathbf{x}, \mathbf{x}_N)\right)\left(\frac{I}{C} + \Omega\right)^{-1} T. \end{aligned} \tag{6}$$

As we all known that the conventional extreme learning machine is usually implemented on a single machine and is inevitable to suffer from the limitation of memory with large scale data set. Indeed, it is definitely essential to handle large scale data which are located on different machines, for example, in some specific applications, the large scale data are collected and stored in a distributed manner and can only be accessed on their own machines; It is impossible to collect all data together due to the confidentiality.

In addition, traditional extreme learning machine is computation-intensive with large scale data because of the high complexity for inverse conversion of the large hidden layer output matrix H (see Eq. (5) for details). This situation will be worse in the case of kernel matrix $\Omega \in R^{N\times N}$ since the number of nodes is equal to the number of samples with large scale data set.

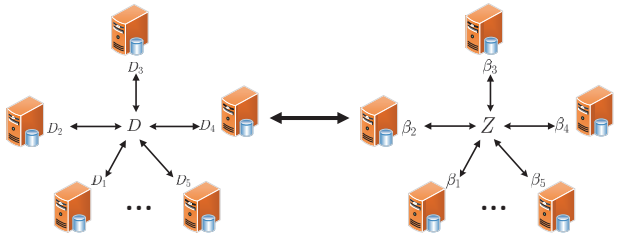


Figure 1: Distributed extreme learning machine.

3. Distributed extreme learning machine

To combat the issues mentioned above, in this section, we extend the traditional extreme learning machine to its distributed version and propose a new distributed extreme learning machine to process the situations where large scale data are located in different machines.

For data set $D = \left\{\left(\mathbf{x}_k^\intercal, t_k\right) : \mathbf{x}_k \in \mathbb{R}^n, t_k \in \mathbb{R}, k = 1, 2, \cdots, N\right\}$, Let $\{D_1, D_2, \cdots, D_m\}$ be a partition of the entire data such that $D = \bigcup_{i=1}^m D_i$. For better representation of, we assume that each subset D_i is located in different machines for $i = 1, 2, \cdots, m$. The optimization problem of DELM is formulated as

$$\begin{aligned} \min_{\beta_1, \beta_2, \cdots, \beta_m, \mathbf{z}} \quad & \frac{1}{2}\|\mathbf{z}\|_2^2 + \frac{C}{2}\sum_{i=1}^m \sum_{j\in D_i} \left(h(\mathbf{x}_j)\beta_i - t_j\right)^2 \\ \text{subject to} \quad & \beta_i = \mathbf{z} \quad (i = 1, 2, \cdots, m) \end{aligned} \tag{7}$$

where, different from the traditional extreme learning machine where all data points in data set D share a common output weight vector, we associate each data set D_i with a local variable $\beta_i = (\beta_{i1}, \beta_{i2}, \cdots, \beta_{iL})^\intercal \in \mathbb{R}^L$ $(i = 1, 2, \cdots, m)$ which represents the corresponding output weight vector. It is noteworthy that the samples in data set D_i are just processed in the i-th processor, and therefore the output weight vector β_i can be determined in parallel. Moreover, a common global variable $\mathbf{z}$ is introduced in the objective function of DELM to integrate all output weight vectors β_i $(i = 1, 2, \cdots, m)$. It is called the global consensus problem since the constraint is that all the local variables should agree [23]. This strategy guarantees to learn a shared output weight vector with

respect to data set D. See Figure 1 for better understanding).

Similar to the traditional extreme learning machine, the parameters of hidden node output function shared by all data set D_i $(i = 1, 2, \cdots, m)$ are generated randomly according to any continuous probability distribution in the framework of DELM. Thus, we can reformulate the optimization problem (7) as follows for better representation,

$$\begin{aligned} \min_{\beta_1, \beta_2, \cdots, \beta_m, \mathbf{z}} \quad & \frac{1}{2}\|\mathbf{z}\|_2^2 + \frac{C}{2}\sum_{i=1}^{m}\|H_i\beta_i - T_i\|_2^2 \\ \text{subject to} \quad & \beta_i = \mathbf{z} \quad (i = 1, 2, \cdots, m) \end{aligned} \tag{8}$$

where $H_i \in \mathbb{R}^{N_i \times L}$ is the hidden-layer output matrix with respect to the data set D_i, i.e.,

$$\begin{aligned} H_i &= \begin{pmatrix} \mathbf{h}(\mathbf{x}_1) \\ \mathbf{h}(\mathbf{x}_2) \\ \vdots \\ \mathbf{h}(\mathbf{x}_{N_i}) \end{pmatrix} \\ &= \begin{pmatrix} h_1(\mathbf{x}_1) & h_2(\mathbf{x}_1) & \cdots & h_L(\mathbf{x}_1) \\ h_1(\mathbf{x}_2) & h_2(\mathbf{x}_2) & \cdots & h_L(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_{N_i}) & h_2(\mathbf{x}_{N_i}) & \cdots & h_L(\mathbf{x}_{N_i}) \end{pmatrix}; \end{aligned}$$

$T_i \in \mathbb{R}^{N_i}$ denotes the actual output corresponding to data set D_i; $N_i = |D_i|$ $(i = 1, 2, \cdots, m)$ refers to the number of data points in data set D_i with $\sum_{i=1}^{m} N_i = N$.

Associated with the sub-datasets D_i $(i = 1, 2, \cdots, m)$, the corresponding hidden-layer output matrixes $H_i \in \mathbb{R}^{N_i \times L}$ $(i = 1, 2, \cdots, m)$ are utilized in the proposed distributed extreme learning machine instead of H with respect to the entire large scale data in traditional extreme learning machine. In this case, the expensive computational cost problem of inverse conversion for large matrix H is avoided to some extent. At the same time, the proposed DELM method makes it possible to implement extreme learning machine algorithm in parallel.

4. ADMM based algorithm for DELM

In this section, we first introduce the notations and preliminaries of ADMM, and then exploit an ADMM based distributed algorithm to solve the proposed optimization problem for DELM.

4.1. ADMM

ADMM is an effective optimization method to solve the following composite optimization problem [9]

$$\begin{aligned} \min \quad & g(\mathbf{z}) + f(\beta) \\ \text{s.t.} \quad & A\beta + B\mathbf{z} = \mathbf{c} \end{aligned} \tag{9}$$

where functions f and g are all convex. With the augmented Lagrangian for (9),

$$\begin{aligned} &\mathcal{L}_\rho(\mathbf{z}, \beta, \lambda) \\ &= g(\mathbf{z}) + f(\beta) + \lambda^T(A\beta + B\mathbf{z} - \mathbf{c}) + \frac{\rho}{2}\|A\beta + B\mathbf{z} - \mathbf{c}\|_2^2, \end{aligned}$$

the algorithm of ADMM consists of iterations

$$\beta^{k+1} = \arg\min_{\beta} \mathcal{L}_\rho(\beta, \mathbf{z}^k, \lambda^k) \tag{10}$$

$$\mathbf{z}^{k+1} = \arg\min_{\mathbf{z}} \mathcal{L}_\rho(\beta^{k+1}, \mathbf{z}, \lambda^k)) \tag{11}$$

$$\lambda^{k+1} = \lambda^k + \rho\left(A\beta^{k+1} + B\mathbf{z}^{k+1} - \mathbf{c}\right) \tag{12}$$

where $\rho > 0$ and k denotes the iteration time. It has been proved that the iterations formulated above satisfy the following theoretical guarantees of convergence:

1. $\mathbf{r}^k = A\beta^k + B\mathbf{z}^k - \mathbf{c} \to 0$ as $k \to +\infty$;
2. $f(\beta^k) + g(\mathbf{z}^k)$ converges to the optimal objective function as $k \to +\infty$.

It is evident that ADMM is a simple but powerful algorithm that aims to find a solution to a global problem by solving local subproblems coordinately [9]. In the framework of ADMM, β^{k+1} is often added with the previous value of $\mathbf{z}^k$ in optimizations (10) and (11) for fast convergence, i.e.,

$$\hat{\beta}^{k+1} = t\beta^{k+1} + (1 - t)\mathbf{z}^k$$

where $t \in [1.5, 1.8]$ is usually used [24]. This technique makes a further improvement and affect efficiency of ADMM significantly.

4.2. Algorithm for DELM

In this section, we exploit an ADMM based algorithm to solve the proposed optimization problem (8). Let $\beta := \{\beta_1, \beta_2, \cdots, \beta_m\}$ and $\lambda := \{\lambda_1, \lambda_2, \cdots, \lambda_m\}$ with $\lambda_i \in \mathbb{R}^L$ for $i = 1, 2, \cdots, m$. We consider the corresponding augmented Lagrangian function given by

$$\begin{aligned} \mathcal{L}_\rho(\beta, \mathbf{z}, \lambda) = &\frac{1}{2}\|\mathbf{z}\|_2^2 + \frac{C}{2}\sum_{i=1}^{m}\|H_i\beta_i - T_i\|_2^2 \\ &+ \sum_{i=1}^{m}\left(\lambda_i^\intercal(\beta_i - \mathbf{z}) + \frac{\rho}{2}\|\beta_i - \mathbf{z}\|_2^2\right). \end{aligned} \tag{13}$$

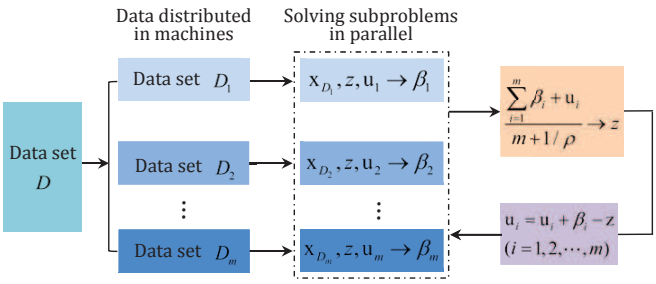


Figure 2: Distributed extreme learning machine with ADMM.

Starting from some initial vectors $\beta^0 = \left\{\beta_1^0, \beta_2^0, \cdots, \beta_m^0\right\}, \lambda^0 = \left\{\lambda_1^0, \lambda_2^0, \cdots, \lambda_m^0\right\}$ and $\mathbf{z}^0$, the variables at iteration $k \geq 0$ are updated in the framework of ADMM as

$$\beta^{k+1} = \arg\min_{\beta} \mathcal{L}_\rho\left(\beta, \mathbf{z}^k, \lambda^k\right) \tag{14}$$

$$\mathbf{z}^{k+1} = \arg\min_{\mathbf{z}} \mathcal{L}_\rho\left(\beta^{k+1}, \mathbf{z}, \lambda^k\right) \tag{15}$$

$$\lambda_i^{k+1} = \lambda_i^k + \rho\left(\beta_i^{k+1} - \mathbf{z}^{k+1}\right) \tag{16}$$

for $i = 1, 2, \cdots, m$. Because the Lagrangian $\mathcal{L}_\rho$ is separable in β_i, we can solve the optimization problem (14) in parallel, i.e.,

$$\beta_i^{k+1} = \arg\min_{\beta_i} \frac{C}{2}\left\|H_i\beta_i - T_i\right\|_2^2 + (\lambda_i^k)^\intercal\left(\beta_i - \mathbf{z}^k\right) + \frac{\rho}{2}\left\|\beta_i - \mathbf{z}^k\right\|_2^2 \tag{17}$$

for $i = 1, 2, \cdots, m$. Let $\lambda_i = \rho\mathbf{u}_i$ $(i = 1, 2, \cdots, m)$ for better representation, we write down an equivalent problem of optimization (17) as

$$\begin{aligned}\beta_i^{k+1} &= \arg\min_{\beta_i} \mathcal{A}_i \\ &= \frac{C}{2}\left\|H_i\beta_i - T_i\right\|_2^2 + \frac{\rho}{2}\left\|\beta_i - \mathbf{z}^k + \mathbf{u}_i^k\right\|_2^2,\end{aligned} \tag{18}$$

The necessary condition for the minimum of optimization problem (18) is derived by setting the partial derivatives with respect to β_i to zero, i.e.,

$$\frac{\partial \mathcal{A}_i}{\partial \beta_i} = CH_i^\intercal\left(H_i\beta_i - T_i\right) + \rho\left(\beta_i - \mathbf{z}^k + \mathbf{u}_i^k\right) = 0. \tag{19}$$

Therefore, we have

$$\beta_i^{k+1} = \left(\frac{\rho}{C}I + H_i^\intercal H_i\right)^{-1}\left[\frac{\rho}{C}\left(\mathbf{z}^k - \mathbf{u}_i^k\right) + H_i^\intercal T_i\right].$$

On the other hand, with $\lambda_i = \rho\mathbf{u}_i$ $(i = 1, 2, \cdots, m)$,

Algorithm 1 Distributed ELM with ADMM

Input: Data set $D = \left\{\left(\mathbf{x}_j^\intercal, t_j\right) : \mathbf{x}_j \in \mathbb{R}^n, t_j \in \mathbb{R}, j = 1, 2, \cdots, N\right\}$ with a partition of all data indices $\{D_1, D_2, \cdots, D_m\}$.

Initialization: $\mathbf{z}^0 = 0, \mathbf{u}_i^0 = 0$ $(i = 1, 2, \cdots, m)$, ρ, t.

1: **for** $k = 0, 1, 2, \cdots$ **do**

2: **Update** β_i^{k+1} **in parallel:**

$$\beta_i^{k+1} = \left(\frac{\rho}{C}I + H_i^\intercal H_i\right)^{-1}\left[\frac{\rho}{C}\left(\mathbf{z}^k - \mathbf{u}_i^k\right) + H_i^\intercal T_i\right]$$

$$\hat{\beta}_i^{k+1} = t\beta_i^{k+1} + (1-t)\,\mathbf{z}^k$$

$(i = 1, 2, \cdots, m)$.

3: **Update** $\mathbf{z}^{k+1}$**:** $\mathbf{z}^{k+1} = \frac{\sum_{i=1}^m\left(\hat{\beta}_i^{k+1} + \mathbf{u}_i^k\right)}{m+1/\rho}$;

4: **Update** $\mathbf{u}_i^{k+1}$**:** $\mathbf{u}_i^{k+1} = \mathbf{u}_i^k + \hat{\beta}_i^{k+1} - \mathbf{z}^{k+1}$ $(i = 1, 2, \cdots, m)$.

5: **end for**

Output: Output weights vector $\mathbf{z}$ with respect to data set D.

the optimization problem (15) is rewritten as

$$\begin{aligned}\mathbf{z}^{k+1} &= \arg\min_{\mathbf{z}} \mathcal{B} \\ &= \frac{1}{2}\left\|\mathbf{z}\right\|_2^2 + \sum_{i=1}^m \frac{\rho}{2}\left\|\beta_i^{k+1} - \mathbf{z}\right\| + \rho\left(\mathbf{u}_i^k\right)^\intercal\left(\beta_i^{k+1} - \mathbf{z}\right).\end{aligned} \tag{20}$$

Thus, the closed form solution for optimization problem (20) is obtained by setting the partial derivatives with respect to $\mathbf{z}$ to zero; and we have

$$\mathbf{z}^{k+1} = \frac{\sum_{i=1}^m\left(\beta_i^{k+1} + \mathbf{u}_i^k\right)}{m + 1/\rho}.$$

It is evident that the update of $\mathbf{z}^{k+1}$ is indeed an averaging step over the present local output weight vectors β_i and $\mathbf{u}_i$ $(i = 1, 2, \cdots, m)$ to some extent [9].

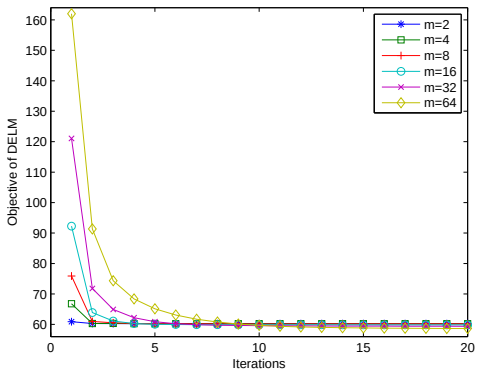


Figure 3: The objective function of DELM with respect to different number of processors.

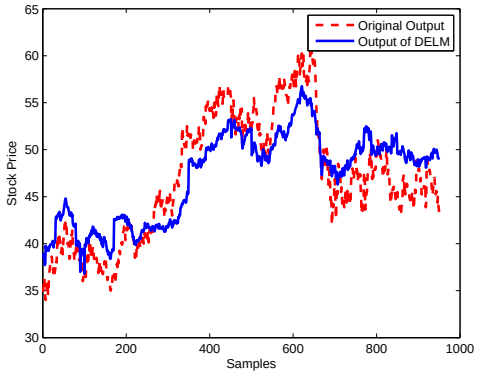


Figure 4: Regression accuracy of DELM.

Based on the analysis above, we summarize the algorithm for DELM on the basis of ADMM in Algorithm 1 which is also visualized by Figure 2 for better understanding.

5. Experiments

In this section, extensive experiments on well-known benchmark data sets are carried out to illustrate the effectiveness and superiority of the proposed DELM. Firstly, we show that the efficient of ADMM based algorithm for the optimization problem of DELM . Secondly, we design experiments to demonstrate the optimal processors and parallel performance of DELM in terms of speedup, scaleup and sizeup, which are the most popular evaluation metrics for parallel algorithm. Finally, we compare the performance of DELM with the conventional extreme learning machine in real-world benchmark data sets for various tasks, including regression, binary and multiclass classification.

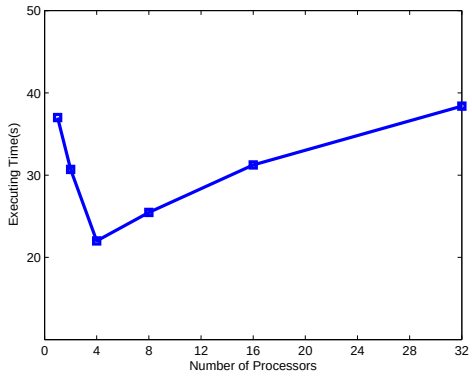


Figure 5: Computing time under different number of processors.

5.1. Experiments for effectiveness of handling optimization

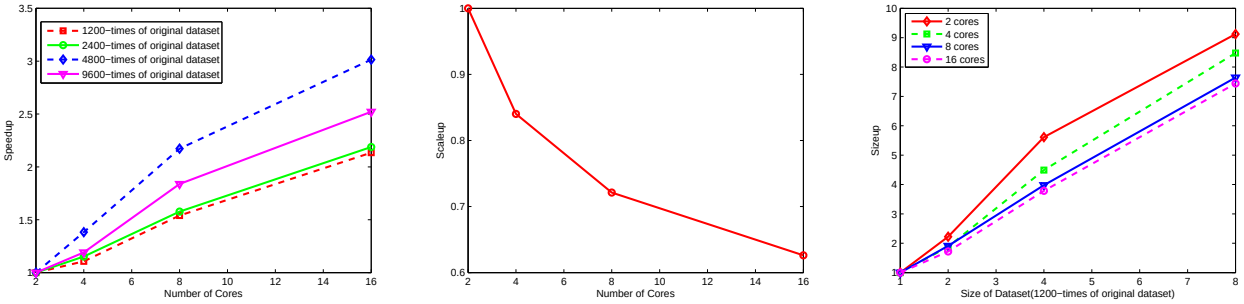
In this section, the dataset *stock* that come from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ ml/datasets.html) is utilized to illustrate the effectiveness of the proposed DELM algorithm. This data set is obtained from practical stock pricing problem and consists of 950 instances with nine independent variables together with one dependent variable. In this experiment, the front 900 instances are used as training dataset, and the rest 50 instances are used as testing dataset.

5.1.1. Convergence and accuracy

We set the value of parameters $\rho = 1$ and $t = 1.5$. The number of hidden nodes is set to 30. The number of processors $m = 2, 4, 6, 8, 16, 32, 64$ are respectively utilized in the framework of DELM. We copy the training data to 1200 times and split the overall training instances evenly among the processors. With respect to different numbers of processors, in Figure 3, we plot the corresponding value of objective function as the increase of iteration times. It indicates that the objective function for different numbers of processors converges within no more than 15 times iterations though slightly more number of iterations are required with respect to more large number of processors. Therefore, the proposed DELM achieves fast convergence on the basis of ADMM. We also show the regression accuracy of DELM in Figure 4. The average root mean square error (RMSE) of training data is 4.456, and the average RMSE of testing data is 4.448.

5.1.2. Experiments for optimal processors

Considering a practical case that users need to assign the large scale data to several processors, in this section,



(a) Speedup (b) Scaleup (c) Sizeup

Figure 6: Parallel performance of DELM.

we conduct experiments to discuss the optimal processors by assigning different number of processors over data set of stock which is copied to 2400-times of the original dataset. The experiments run on a computer with $8 * 2.8GHZ$ cores and 8GB memory. As shown in Figure 5, the total computing time of DELM decreases with the increase of number of processors and reaches its minimum when the number of processors is 4. After that, with the increase of processors, the computing time increases gradually because of the extra overhead. Note that the procedure includes the communication time among processors and the management time. As a result, we can conclude that there is a tradeoff between the computing time of each processor and the extra overhead among processors.

5.1.3. *Experiments for parallel performance of DELM*

In this section, three popular evaluation metrics, namely speedup, scaleup, and sizeup are leveraged to analyze the performance of the proposed DELM. We denote the computing time of n-times data on m computers by $T_{n*data}(m)$, and then the meanings of speedup, scaleup, and sizeup are given as follows,

$$Speedup(m) = \frac{T_{n*data}(1)}{T_{n*data}(m)} \tag{21}$$

$$Scaleup(m) = \frac{T_{1*data}(1)}{T_{m*data}(m)} \tag{22}$$

$$Sizeup(n) = \frac{T_{n*data}(m)}{T_{1*data}(m)} \tag{23}$$

Specifically, given n-times data, the speedup of DELM is calculated according to the formula (21). The perfect parallel system demonstrates linear speedup. The value of scaleup is calculated according to (22), which is used to evaluate the scalability of DELM. With varying values of m, the parallel algorithm has better adaptability if the scaleup value has been 1.0 or less. The equation (23) describes the calculation of the sizeup. It demonstrates the time cost that spends on a given number of computers m when the dataset size becomes n-times larger than the original dataset.

In this experiment, we firstly evaluate the speedup of DELM by using 1200-times, 2400-times, 4800-times and 9600-times of the original training dataset with the cores number varying from 2 to 16. Next, to demonstrate the scaleup performance of DELM, we run scalability experiments on the system where the datasets size of 1200-times, 2400-times, 4800-times, 9600-times of original training dataset with 2, 4, 8 and 16 cores, respectively. Finally, we run the parallel algorithm on 1200-times, 2400-times, 4800-times and 9600-times of original training datasets with fixed the number of cores to evaluate the performance of sizeup for DELM.

As shown in Figure 6a, with the increase of dataset size, the speedup of DELM algorithm tends to be approximately linear, especially for the large-scale dataset. To the best of our knowledge, the linear speed is difficult to achieve because of the extra communication time. In particular, with the increase of data size, there are more communication cost among processors. The scaleup and sizeup performances of DELM are shown in Figure 6b and Figure 6c, respectively. It indicates that the scaleup of DELM has a decline trend gradually with the increase of the dataset size and the cores number. As a result, the proposed DELM algorithm achieves great scalability and sizeup performance.

5.2. *Comparison experiments*

In order to verify the better performance of DELM, in this section, we conduct extensive experiments on several benchmark data sets with respect to 3 binary classification cases, 3 multiclass classification cases and 3

Table 1: Benchmark Datasets of experiments.

Dataset	Task	Instances	Train	Test	Feature	Times
Banknote	Binary Class	1372	915	457	4	1600
Diabetes	Binary Class	768	512	256	8	2400
Liver	Binary Class	345	230	115	6	4800
Glass	Multi Class	214	142	72	10	4800
Seed	Multi Class	210	140	70	7	4800
Wine	Multi Class	178	118	60	12	4800
Housing	Regression	506	337	169	14	2400
Stock	Regression	950	900	50	10	2400
Yacht	Regression	308	205	103	7	4800

Table 2: Performance Comparison of ELM and DELM: Binary/Multi-Class Datasets.

Dataset	ELM		DELM	
	Rate(%)	Dev(%)	Rate(%)	Dev(%)
Banknote	97.68	0.30	97.59	0
Diabetes	76.66	0.90	77.03	0.93
Liver	66.48	1.58	68.70	1.52
Glass	75.91	6.77	81.11	1.98
Seed	99.99	0.01	98.50	0.32
Wine	97.50	5.32	98.83	1.72

Table 3: Performance Comparison of ELM and DELM: Regression Datasets.

Dataset	ELM		DELM	
	Tr-RMSE	Te-RMSE	Tr-RMSE	Te-RMSE
Housing	4.87	4.66	4.21	3.99
Stock	1.95	1.80	1.73	1.62
Yacht	5.27	5.31	5.14	5.26

regression cases. All of the data sets are taken from UCI Machine Learning Repository. We describe the details of data sets in Table 1, where the column "Times" shows that how many times we copy the data set in the experiment.

For fair comparison, we set the number of hidden nodes to 30 for all of the experiments; the Sigmoid function is adopted for the output of each hidden node; the value of parameter C is set to 1 in the framework of both traditional extreme learning machine and the proposed DELM. Additionally, we set the value of parameters $\rho = 1$, $t = 1.5$ in DELM.

Some simulation results are given in our experiments including the average testing accuracy, the corresponding standard deviation (Dev), and the value of RMSE associated with training data and testing data. Table 2 and Table 3 show the performance comparison of ELM and DELM. It indicates that DELM could always achieve comparable performance to extreme learning machine in binary class, multiclass, and regression. Therefore, the proposed DELM could not only relieve the limitation of memory with big data, but also get a comparable performance.

6. Conclusion

In this paper, we exploit an effective distributed extreme learning machine to implement the traditional ELM algorithm in parallel for large-scale data set. This method takes advantages of ADMM algorithm and find a solution to a global problem by solving local subproblems coordinately. In such a way, the proposed method can relieves the limitation of memory of single machine to some extent and can handle some situations where large scale data are collected and stored in distributed manner. Extensive experiments on well-known benchmark data sets demonstrate the effectiveness and superiority of the proposed DELM.

Acknowledgments

This work was supported in part by the National Science Foundation of China under Grant 61502377, Grant 61532015, and Grant 61532004, in part by the National Key Research and Development Program of China under Grant 201s6YFB1000903, and in part by the China Post-Doctoral Science Foundation under Grant 2015M582662.

References

[1] G. Huang, L. Chen, C. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Transaction on Neural Networks 17 (4) (2006) 879–892.

[2] X. Li, W. Mao, W. Jiang, Extreme learning machine based transfer learning for data classification, Neurocomputing 174 (2016) 203–210.
[3] X. Zhao, X. Bi, G. Wang, C. Wang, H. Yang, Uncertain xml documents classification using extreme learning machine, Neurocomputing 174 (2016) 375–382.
[4] Y. Peng, W. Zheng, B. Lu, An unsupervised discriminative extreme learning machine and its applications to data clustering, Neurocomputing 174 (2016) 250–264.
[5] X. Luo, X. Chang, X. Ban, Regression and classification using extreme learning machine based on l_1-norm and l_2-norm, Neurocomputing 174 (2016) 179–186.
[6] C. Deng, G. Huang, J. Xu, J. Tang, Extreme learning machines: new trends and applications, Science China Information Sciences 58 (2) (2015) 1–16.
[7] S. Shahaboddin, M. Kasra, C. Tong, D. Petković, Application of extreme learning machine for estimation of wind speed distribution, Climate Dynamics (2015) 1–15.
[8] A. Le, B. Bir, Image super-resolution by extreme learning machine, in: Proceedings of the 19th International Conference on Image Processing (ICIP), Florida, USA, 2012, pp. 2209–2212.
[9] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends in Machine Learning 3 (1) (2010) 1–122.
[10] X. Chang, Y.-L. Yu, Y. Yang, A. G. Hauptmann, Searching persuasively: Joint event detection and evidence recounting with limited supervision, in: ACM Conference on Multimedia (MM), 2015.
[11] S. Huang, B. Wang, J. Qiu, J. Yao, G. Wang, G. Yu, Parallel ensemble of online sequential extreme learning machine based on mapreduce, Neurocomputing 174 (2016) 352–367.
[12] J. Xin, Z. Wang, L. Qu, G. Yu, Y. Kang, A-elm*: adaptive distributed extreme learning machine with mapreduce, Neurocomputing 174 (2016) 368–374.
[13] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.
[14] X. Wang, Y. Chen, H. Zhao, B. Lu, Parallel extreme learning machine for regression based on mapreduce, Neurocomputing 102 (2013) 52–58.
[15] Q. He, T. Shang, F. Zhuang, Z. Shi, Parallelized extreme learning machine ensemble based on min-max modular network, Neurocomputing 128 (2014) 31–41.
[16] A. Akusok, K. Bjork, M. Yoan, A. Lendasse, High-performance extreme learning machines: a complete toolbox for big data applications, IEEE Access 3 (2015) 1011–1025.
[17] R. Zhang, K. James, Asynchronous distributed admm for consensus optimization, in: Proceedings of the 31st International Conference on Machine Learning (ICML-14), Beijing, China, 2014, pp. 1701–1709.
[18] C. Zhang, H. Lee, K. shin, Efficient distributed linear classification algorithms via the alternating direction method of multipliers, in: Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS), Canary Islands, Spain, 2012, pp. 1398–1406.
[19] C. Lu, H. Li, Z. Lin, S. Yan, Fast proximal linearized alternating direction method of multiplier with parallel splitting, arXiv preprint arXiv:1511.05133.
[20] G. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trasaction on Systems, Man, and Cybernetics-Part B: Cybernetics 42 (2) (2012) 513–529.
[21] H. Liu, D. Guo, F. Sun, Object recognition using tactile measurements: Kernel sparse coding methods, IEEE Transactions on Instrumentation and Measurement 65 (3) (2016) 656–665.
[22] H. Liu, D. Guo, F. Sun, Robust exemplar extraction using structured sparse coding, IEEE Transactions on Neural Networks and Learning Systems 26 (8) (2015) 1816–1821.
[23] A. Nedić, O. Asuman, Distributed subgradient methods for multi-agent optimization, IEEE Transaction on Automatic Control 54 (1) (2009) 48–61.
[24] J. Eckstein, Parallel altrenating direction multiplier decomposition of convex programs, Optimization Theory and Applications 80 (1) (1994) 39–62.

Minnan Luo received the Ph. D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2014. Currently, she is an Assistant Professor in the School of Electronic and Information Engineering at Xi'an Jiaotong University. She is also a Post-Doctoral Researcher with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA. Her research interests include machine learning and optimization, video analysis, cross-media retrieval and fuzzy system identification and modeling.

Lingling Zhang is currently working toward the MS degree in computer science at Xi'an Jiaotong University. She received the BS degree in Computing Science from Xi'an Jiaotong University in 2015. Her research interests include cross-media information mining and E-learning.

Jun liu received the B.S, M.S and Ph.D degrees in computer science in 1995, 1998, and 2004 from Xi'an Jiaotong University, China. He is currently a professor in the School of Electronic and Information Engineering at Xi'an Jiaotong University. His research interests include text mining, data mining, Intelligent Network learning Environment and multimedia E-learning.

Guo Jun is an associated Professor of Northwest University. He Graduated from National University of Defence Technology in 1990, got the B. E. degree of computer science and engineering. In 1997 he got the Master degree at Northwest Institute of Nuclear Technology. He got Doctor degree from Northwest University in 2007 and worked as post doctoral fellow in Tsinghua University during 2008-2010. At the meantime he held a visiting scholarship at Waseda University in Japan for one year. His research interests including embedded computing,natural language processing and AI algorithm.

Qinghua Zheng received the B.S. degree in computer software in 1990, the M.S. degree in computer organization and architecture in 1993, and the Ph.D. degree in system engineering in 1997 from Xi'an Jiaotong University, China. He was a Postdoctoral Researcher at Harvard University in 2002. He is currently a professor in Xi'an Jiaotong University, and the dean of the Department of Computer Science. His research areas include computer network security, intelligent E-learning theory and algorithm, multimedia e-learning, and trustworthy software.