

# Efficient Batch Parallel Online Sequential Extreme Learning Machine Algorithm Based on MapReduce

Shan Huang, Botao Wang, Yuemei Chen, Guoren Wang and Ge Yu

**Abstract** With the development of technology and the widespread use of machine learning, more and more models need to be trained to mine useful knowledge from large scale data. It has become a challenging problem to train multiple models accurately and efficiently so as to make full use of limited computing resources. As one of ELM variants, online sequential extreme learning machine (OS-ELM) provides a method to learn from incremental data. MapReduce, which provides a simple, scalable and fault-tolerant framework, can be utilized for large scale learning. In this paper, we propose an efficient batch parallel online sequential extreme learning machine (BPOS-ELM) algorithm for the training of multiple models. BPOS-ELM estimates the Map execution time and Reduce execution time with historical statistics and generates execution plan. BPOS-ELM launches one MapReduce job to train multiple OS-ELM models according to the generated execution plan. BPOS-ELM is evaluated with real and synthetic data. The accuracy of BPOS-ELM is at the same level as those of OS-ELM and POS-ELM. The speedup of BPOS-ELM reaches 10 on a cluster with maximum 32 cores.

**Keywords** Parallel learning • Extreme learning machine • MapReduce • Sequential learning

---

S. Huang (✉) · B. Wang · Y. Chen · G. Wang · G. Yu  
College of Information Science and Engineering, Northeastern University,  
Shenyang 110819, Liaoning, China  
e-mail: huangshan.neu@gmail.com

B. Wang  
e-mail: wangbotao@ise.neu.edu.cn

Y. Chen  
e-mail: 1044210092@qq.com

G. Wang  
e-mail: wanggr@ise.neu.edu.cn

G. Yu  
e-mail: yuge@ise.neu.edu.cn

# 1 Introduction

With the development of technology and the widespread use of machine learning, more and more models need to be trained to mine useful knowledge from large scale data. It has become a challenging problem to train multiple models accurately and efficiently so as to make full use of limited computing resources. For example, in a machine learning organization where high performance computing cluster is a limited resource, researchers must schedule the jobs on the cluster legitimately to make full use of the cluster. For another example, resizable cloud hosting services such as Amazon Elastic Compute Cloud (EC2) [1], which become more and more popular, make it possible to rent large amount of virtual machines by the hour at lower costs than operating a data center year-round. It is important for users to schedule multiple jobs running on this kind of environment as the rented virtual machines are charged by the used time.

Extreme learning machine (ELM) was proposed based on single-hidden layer feed-forward neural networks (SLFNs) [7], and it has been verified to have high learning speed as well as high accuracy [5]. It has also been proved that ELM has universal approximation capability and classification capability [6]. As one of ELM variants, online sequential extreme learning machine (OS-ELM) [8] supports incremental learning.

MapReduce [3] is a well-known framework which supports large scale data processing and analyzing on a large cluster of commodity machines. Recent research has studied on parallelizing ELM [4, 11, 12], however the strategies are not suitable to parallelize OS-ELM. POS-ELM [10] supports training one single OS-ELM model in parallel with MapReduce, but it does not support training multiple OS-ELM models efficiently.

In this paper, we propose an efficient batch parallel online sequential extreme learning machine (BPOS-ELM) algorithm for the training of multiple OS-ELM models on MapReduce. BPOS-ELM first estimates the execution time of Map and Reduce tasks of each OS-ELM based on historical statistics. Then it generates a Map execution plan and a Reduce execution plan with greedy strategy based on the estimations. After that, BPOS-ELM launches a MapReduce job to train multiple OS-ELM models. At the same time, BPOS-ELM collects execution information of selected Map tasks and Reduce tasks, and merges them to historical statistics to improve the accuracy of time estimation. The algorithm is evaluated with real and synthetic data. The accuracy is at the same level as those of OS-ELM and POS-ELM. The speedup reaches 10 on a cluster with maximum 32 cores.

The remainder of this paper is organized as follows. Section 2 describes the batch parallel online sequential learning machine algorithm in detail. An extensive experimental evaluation of BPOS-ELM is presented in Sect. 3. A brief conclusion is presented in Sect. 4.

2 BPOS-ELM

As shown in Fig. 1, BPOS-ELM (1) assigns each model with a unique ID which is used to specify it from the other training models. Then (2) the Map execution time and Reduce execution time are estimated according to historical statistics described with parameters shown in Table 1. After that, (3) A job execution plan is generated according to the estimations. Finally, (4) the generated execution plan is executed to train the models and (5) the actual execution information of selected tasks is collected for future time estimation. The details of execution time estimation, execution plan generation and job execution are described in Sects. 2.1–2.3, respectively.

2.1 Execution Time Estimation

Map execution time is estimated with Inverse Distance Weighted (IDW) [9] interpolation method. First, each parameter of job execution information is mapped to one dimension at a multi-dimensional space, so the historical statistics are mapped to a set of points in the space. Then  $k$  nearest neighbour points of the point to be estimated in the space are selected and used to estimate Map execution time. After that, Inverse Distance Weighted (IDW) [9] interpolation method shown as Formula (1) is used to estimate Map execution time.

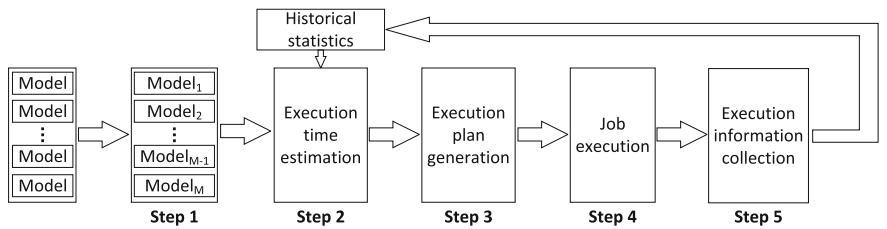


Fig. 1 Execution framework of BPOS-ELM

Table 1 Notations used in BPOS-ELM

Parameter	Description
$B$	Block size
$N$	Number of training data
$D$	Number of attributes in training data set
$M$	Number of training models
$\tilde{N}$	Number of hidden layer nodes
$C$	Number of classifications

$$t_{map}(\mathbf{x}) \approx \begin{cases} \frac{\sum_{i=1}^k w_i(\mathbf{x}) t_i}{k}, & \text{if } d(\mathbf{x}, \mathbf{x}_i) \neq 0 \text{ for all } i \\ t_i, & \text{if } d(\mathbf{x}, \mathbf{x}_i) = 0 \text{ for some } i \end{cases} \quad (1)$$

where  $w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^p}$  is a simple IDW weighting function, as defined by Shepard [9],  $\mathbf{x}$  denotes the parameter vector of point to be predict,  $\mathbf{x}_i$  is the selected  $k$  nearest neighbour points,  $d$  is a given distance from the point  $\mathbf{x}_i$  to point  $\mathbf{x}$  and  $p$  is a positive real number, called the power parameter. Euclidean distance is used to measure the distance between two points. According to the complexity analysis of POS-ELM reduce phase algorithm in [10], the Reduce execution time is estimated by Formula (2).

$$\begin{aligned} t_{red} &\approx \frac{N}{B} (\alpha_1 B^3 + \alpha_2 B^2 \tilde{N} + \alpha_3 B \tilde{N}^2 + \alpha_4 B \tilde{N} C + \alpha_5 (B^2 + BC + \tilde{N}^2 + \tilde{N} C)) \\ &= N (\alpha_1 B^2 + \alpha_2 B \tilde{N} + \alpha_3 \tilde{N}^2 + \alpha_4 \tilde{N} C + \alpha_5 (B + C + \frac{\tilde{N}^2}{B} + \frac{\tilde{N} C}{B})) \end{aligned} \quad (2)$$

where  $\alpha_n (1 \leq n \leq 5)$  are the factors that need to be determined using historical statistics. Then, the Reduce execution time estimation transforms to a multi-parameter regression problem. In this paper, OS-ELM is used as the regression model to solve the problem.

## 2.2 Execution Plan Generation

### 2.2.1 Map Execution Plan Generation

The execution plan generation algorithm of Map phase is shown in Algorithm 1. It first calculates the predictable average execution time of Map tasks (lines 1–3). The models whose estimated Map execution time is less than average time are treated as unit executions during the execution plan generation (lines 5–6). The models whose estimated Map execution time is more than average time are split to multiple unit executions (lines 7–11). After generating the list of unit executions, heuristic algorithm GeneratePlan is executed to generate Map execution plan.

The GeneratePlan algorithm is shown in Algorithm 2, which is used in both Map execution plan generation and Reduce execution plan generation. When the number of unit executions in the list is less than that of tasks, each of the unit execution is assigned to each task (lines 1–3). Otherwise, greedy strategy is used to generate execution plan. *Unassigned* is initialized and used to count the number of unassigned unit executions in the list (line 5). First, the list of unit executions is sorted by

**Algorithm 1:** Map execution plan generation

---

**Input:** *models* [ ] : array of OS-ELM models.  
           *MapNum* : the maximum number of Map tasks in the cluster.

**Result:** *MapPlan* < List < ID, start, end >>[ ] : the Map execution plan.

```

1 for m = 1 to sizeof(models) do
2   | TimeSum = TimeSum + model[i].EstimatedMapTime;
3 AvgTime =  $\frac{\textit{TimeSum}}{\textit{MapNum}}$ ;
4 for m = 1 to sizeof(models) do
5   | if models[m].EstimatedMapTime ≤ AvgTime *  $\alpha$  then
6     | list.add(< models[m].id, 0, models[m].InputSize,
7       | models[m].EstimatedMapTime >);
8   | else
9     | splits =  $\frac{\textit{models}[\textit{m}].\textit{EstimatedMapTime}}{\textit{AvgTime}}$  ;
10    | splitsize =  $\frac{\textit{models}[\textit{m}].\textit{InputSize}}{\textit{splits}}$ ;
11    | for i = 1 to splits do
12      | list.add(< models[m].id, i * splitsize, (i + 1) * splitsize,
13        |  $\frac{\textit{models}[\textit{m}].\textit{EstimatedMapTime}}{\textit{splits}}$  >);
14  |
15  |
16  |
17  |
18  |
19  |
20  |
21  |
22  |
23  |
24  |
25  |
26  |
27  |
28  |
29  |
30  |
31  |
32  |
33  |
34  |
35  |
36  |
37  |
38  |
39  |
40  |
41  |
42  |
43  |
44  |
45  |
46  |
47  |
48  |
49  |
50  |
51  |
52  |
53  |
54  |
55  |
56  |
57  |
58  |
59  |
60  |
61  |
62  |
63  |
64  |
65  |
66  |
67  |
68  |
69  |
70  |
71  |
72  |
73  |
74  |
75  |
76  |
77  |
78  |
79  |
80  |
81  |
82  |
83  |
84  |
85  |
86  |
87  |
88  |
89  |
90  |
91  |
92  |
93  |
94  |
95  |
96  |
97  |
98  |
99  |
100 |

```

12 *MapPlan* = **GeneratePlan**(*list.toArray()*, *MapNum* , *AvgTime* );

---

estimated execution time in descending order (line 6). Then the sorted list is scanned and the unit executions in it are added to the execution plan (lines 7–16). The assigned unit executions are skipped (lines 8–9) and the loop is broken when *Count* exceeds the number of tasks (lines 10–11). After that, the unassigned unit execution which has the longest execution time is added to execution plan (line 12) and the algorithm scans the remaining list to find the suitable unit execution and add it to execution plan recursively (lines 14–15). Finally, the algorithm scans the list of unit executions again and adds the unassigned unit execution to the expected shortest task (lines 17–20).

## 2.2.2 Reduce Execution Plan Generation

The execution plan generation algorithm of Reduce phase is shown in Algorithm 3. The algorithm first calculates the expected average execution time of Reduce tasks (lines 1–3). Then it scans the OS-ELM models and adds them to the list of unit executions (lines 4–5). As the calculations of POS-ELM algorithm in Reduce phase is indivisible, each Reduce task is treated as a unit execution. Since *start* and *end* are not used in Reduce execution plan generation, they are set to 0 to be compatible with **GeneratePlan** algorithm. After that, **GeneratePlan** introduced in Map execution plan generation is executed to generate Reduce execution plan (line 6). At last, the algorithm scans the execution plan and assigns the OS-ELM models in the plan with

---

**Algorithm 2:** GeneratePlan()

---

**Input:** *Tasks* < *ID*, *start*, *end*, *time* > [ ] : array of quadruples, in which each quadruple represents task information of OS-ELM model.

*TaskNum* : the maximum number of tasks that the cluster can hold.

*AvgTime* : the expected average execution time for each task.

**Result:** *Plan* < *List* < *ID*, *start*, *end* >> [ ] : the execution plan.

```

1  if sizeof(Tasks) ≤ TaskNum then
2    for m = 1 to sizeof(Tasks) do
3      Plan[m].add( < Tasks[m].id, Tasks[m].start, Tasks[m].end >);
4  else
5    Unassigned = Size = sizeof(Tasks);
6    SortByTimeInDescendingOrder(Tasks);
7    for i = 1 to Size do
8      if used[i] == true then
9        continue;
10     if Count ≥ TaskNum then
11       break;
12     addToPlan(Count, i)
13     Start = i + 1;
14     for Start ≤ Size do
15       Start = FindAndAdd(Start);
16     Count = Count + 1;
17   for i = 1 to Size do
18     if Unassigned > 0 && used[i] == false then
19       insertIndex = findMinTimeIndex(Time);
20       AddToMapPlan(insertIndex, i);
21 FindMinTimeIndex(Time)
22   for i = 1 to Size do
23     if Time[i] < MinTime then
24       MinTime = Time[i];
25       MinIndex = i;
26   return MinIndex;
27 FindAndAdd(Start)
28   for j = Start to Size do
29     if used[j] == false && Tasks[j].time + Time[Count] ≤ AvgTime *  $\alpha$  then
30       addToMapPlan(Count, j);
31       return j;
32   return j;
33 AddToPlan(P_I, T_I)
34   used[T_I] = true;
35   Time[P_I] = Time[P_I] + Tasks[T_I].time
36   Plan[P_I].add( < Tasks[T_I].id, Tasks[T_I].start, Tasks[T_I].end > )
37   Unassigned = Unassigned - 1;

```

---

**Algorithm 3:** Reduce execution plan generation

---

**Input:** *models* [ ] : array of OS-ELM models.  
           *ReduceNum* : the maximum number of Reduce tasks in the cluster.

**Result:** *ReducePlan* < List < ID, start, end >> [ ] : the Reduce execution plan.

```

1 for m = 1 to sizeof(models) do
2   | TimeSum = TimeSum+model[i].EstimatedReduceTime;
3 AvgTime =  $\frac{TimeSum}{ReduceNum}$ ;
4 for m = 1 to sizeof(models) do
5   | list.add(< models[m].id, 0, 0, models[m].EstimatedReduceTime >);
6 ReducePlan = GeneratePlan(list.toArray(), ReduceNum, AvgTime);
7 for i = 1 to sizeof(ReducePlan) do
8   | for j = 1 to sizeof(ReducePlan[i]) do
9     | Index = FindByID(models, ReducePlan [i][j].ID);
10    | models [Index].ReduceKey = i ;
11 FindByID(list, ID)
12   | for i = 1 to sizeof(list) do
13     | if list[i].ID == ID then
14       |   | return i;

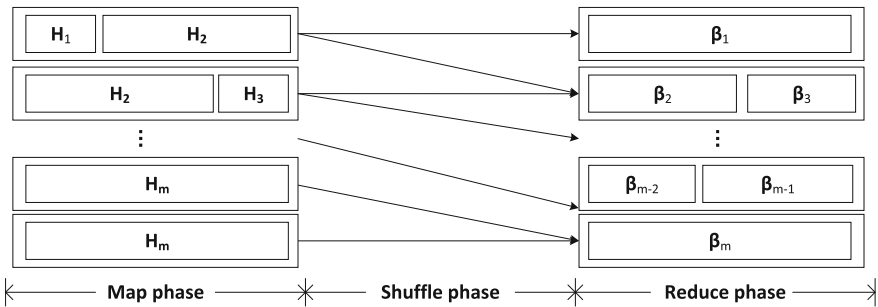
```

---

correct *ReduceKeys* (lines 7–10). The *ReduceKeys* are used to mark which Reduce task that intermediate results should pass to.

### 2.3 Job Execution

Figure 2 shows the execution procedure of BPOS-ELM. Each Map task is responsible for calculating **H** for one OS-ELM model, part of one OS-ELM model or multiple OS-ELM models according to the Map execution plan generated. Each Reduce task



**Fig. 2** Job execution of BPOS-ELM

---

**Algorithm 4:** BPOS-ELM map()
 

---

**Input:** (Key, Value): Key is the OS-ELM model ID, Value is a sample pair  $(x_i, t_i) \in (X_k, T_k)$  where  $0 \leq i \leq |X_k, T_k|$  for the model.  
**Result:**  $m$  : OS-ELM model ID;  
            $k$  : blockID;  
            $ReduceKey_m$  : Key that marks which Reduce task trains the model;  
            $\mathbf{H}_{m,k}$  : Output weight;  
            $\mathbf{T}_{m,k}$  : Observation value vector;

```

1  $m = Key$ ;
2 add to  $block_m$ ;
3  $count_m++$ ;
4 if  $count \geq BLOCK_m$  then
5    $\mathbf{H}_{m,k} = \text{calcH}(block_m)$ ;
6    $\mathbf{T}_{m,k} = \text{calcT}(block_m)$ ;
7   output( $(m, k, ReduceKey_m), (\mathbf{H}_{m,k}, \mathbf{T}_{m,k})$ );
8    $count_m = 0$ ;
9    $k++$ ;
```

---



---

**Algorithm 5:** BPOS-ELM reduce()
 

---

**Input:** Set of (key, value): key is a combination of OS-ELM model ID  $m$ , blockID  $k$  and  $ReduceKey$ . value is a vector pair  $(H_{kb}, T_{kb})$ ;  
**Result:**  $\beta_m$ : output weight vector (corresponding to  $\beta_{m,k}$ ).

```

1  $m = \text{getm}(key)$ ;
2  $\mathbf{H}_{m,k+1} = \text{getH}(value)$ ;
3  $\mathbf{T}_{m,k+1} = \text{getT}(value)$ ;
4  $\mathbf{P}_{m,k+1} = \mathbf{P}_{m,k} - \mathbf{P}_{m,k} \mathbf{H}_{m,k+1}^T (\mathbf{I} + \mathbf{H}_{m,k+1} \mathbf{P}_{m,k} \mathbf{H}_{m,k+1}^T)^{-1} \mathbf{H}_{m,k+1} \mathbf{P}_{m,k}$ ;
5  $\beta_{m,k+1} = \beta_{m,k} + \mathbf{P}_{m,k+1} \mathbf{H}_{m,k+1}^T (\mathbf{T}_{m,k+1} - \mathbf{H}_{m,k+1} \beta_{m,k})$ ;
```

---

is responsible for calculating  $\beta$  for one OS-ELM model or multiple OS-ELM models according to the Reduce execution plan.

The pseudo codes of Map procedure are shown in Algorithm 4. First it collects  $BLOCK_m$  training instances into a buffer  $block_m$  (lines 1–3). After  $BLOCK_m$  training instances are collected (line 4), matrix  $\mathbf{H}_{m,k}$  is calculated (line 5) and  $\mathbf{T}_{m,k}$  is also generated (line 6). After that, a key-value pair is generated as output (line 7). *key* is composed with OS-ELM model ID  $m$ , block ID  $k$  and  $ReduceKey_m$  while *value* includes  $\mathbf{H}_{m,k}$  and  $\mathbf{T}_{m,k}$ . Finally, the counter is cleared (line 8) and the block ID  $k$  is increased by one (line 9).

The pseudo codes of Reduce procedure are shown in Algorithm 5. The output results of Map phase which have the same  $ReduceKey$  are partitioned to the same Reducer and then sorted by  $m$  and  $k$ . The OS-ELM model ID  $m$  is first resolved (line 1). Then  $\mathbf{H}_{m,k}$  and  $\mathbf{T}_{m,k}$  included in *value* are resolved (lines 2–3). Finally, the  $\mathbf{P}_{m,k}$  and  $\beta_{m,k}$  are updated according to the formulas (lines 4–5).



### 3 Experimental Evaluation

#### 3.1 Experimental Setup

In this section, POS-ELM indicates parallel online sequential learning machine algorithm in [10] that trains each OS-ELM model one by one. BPOS-ELM is compared with POS-ELM and OS-ELM algorithms. All the three algorithms are implemented in Java 1.6. Universal java matrix package (UJMP) [2] with version 0.2.5 is used for matrix storage and processing. The activation function of OS-ELM, POS-ELM and BPOS-ELM algorithm is  $g(x) = \frac{1}{1+e^{-x}}$ . The number of hidden layer node is set to 128 in accuracy evaluation and it is set to 64 in training speed evaluation and scalability evaluation.

Hadoop-0.20.2-cdh3u3 is used as our evaluation platform. The Hadoop cluster is deployed on 9 commodity PCs in a high speed Gigabit network, with one PC as the Master node and the others as the Slave nodes. Each PC has an Intel Quad Core 2.66 GHZ CPU, 4 GB memory and CentOS Linux 5.6 operating system. Each PC is set to hold maximum 4 Map or Reduce tasks running in parallel and the cluster is set to hold maximum 32 tasks running in parallel. Each task is configured with 1024M java heap. Other parameters are using the default values of Hadoop.

BPOS-ELM algorithm is evaluated with real data and synthetic data. The real data sets (MNIST,<sup>1</sup> DNA (see footnote 1) and KDDCup99<sup>2</sup>) are mainly used to evaluate training accuracy and testing accuracy. Some attributes of KDDCup99 data set are symbolic-valued attributes which cannot be directly used for BPOS-ELM, POS-ELM or OS-ELM, so we preprocess the data set by mapping symbolic-valued attributes to numeric-valued attributes with the method in [11]. For testing data, we use the KDDcup99 (corrected) evaluation data set by excluding those attack instances which do not belong to the set of attack types in the training data set. The specifications of real data are shown in Table 2.

The synthetic data sets are used for training speed evaluation and scalability evaluation, which are generated by extending based on Flower.<sup>3</sup> The volume and attributes of training data are extended by duplicating the original data in a round-robin way. In one training model group, there are 11 OS-ELM models training with synthetic data sets with  $N$  varies from  $2^0 \times 10^4$  to  $2^{10} \times 10^4$ . The parameters used in scalability evaluation are summarized in Table 3. In the experiments, all the parameters use default values unless otherwise specified.

<sup>1</sup>Downloaded from <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

<sup>2</sup>Downloaded from <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>.

<sup>3</sup>Downloaded from <http://www.datatang.com/data/13152>.

**Table 2** Specifications of real data

Data set	#attributes	#class	#training data	#testing data	Size of test data (KB)
MNIST	780	10	60000	10000	176001.138
DNA	180	3	2000	1186	1126.301
KDDCup99	41	2	4898431	292300	708197.916

**Table 3** Specifications of synthetic data and running parameters for scalability evaluation

Parameter	Value range	Default value
#training data	10k, 20k, 40k, 80k, 160k, 320k, 640k, 1280k, 2560k, 5120k, 10240k	640k
#attributes	64, 128, 256, 512	64
#cores	1, 2, 4, 8, 16, 32	32
#data per block	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024	64
#neurons	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024	64
#classifications	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024	2
#model groups	3, 6, 9, 12, 15, 18, 21, 24, 27, 30	6

## 3.2 Evaluation Results

### 3.2.1 Accuracy Evaluation

Table 4 shows the results of accuracy evaluations with real data. It can be found that the training accuracy and testing accuracy of BPOS-ELM algorithm are at the same level with those of POS-ELM and OS-ELM. The reason for this is that BPOS-ELM algorithm does not change the computation sequence of matrices calculation of OS-ELM.

### 3.2.2 Training Speed Evaluation

Table 5 shows the results of training speed evaluation with real data and synthetic data. As shown in Table 5, the training speed of BPOS-ELM is faster than the training speed of POS-ELM and OS-ELM. For the models training with real data sets, the training speed of BPOS-ELM is only a little faster than that of POS-ELM. The reason for this is that most of the cores are idle in Reduce phase of BPOS-ELM since the number of the training models is less than that of cores and the Reduce tasks are indivisible. For the models training with synthetic data sets, the training speed of BPOS-ELM is much faster than that of POS-ELM and OS-ELM. This is because the cores of the cluster are fully utilized in the Reduce phase of BPOS-ELM algorithm.

**Table 4** Accuracy evaluation with real data

Data set	Algorithm	Training accuracy	Testing accuracy
MNIST	OS-ELM	0.824	0.831
	POS-ELM	0.823	0.830
	BPOS-ELM	0.825	0.831
DNA	OS-ELM	0.845	0.779
	POS-ELM	0.846	0.781
	BPOS-ELM	0.844	0.780
KDDCup99	OS-ELM	0.992	0.856
	POS-ELM	0.991	0.856
	BPOS-ELM	0.992	0.855

**Table 5** Execution time evaluation with real data and synthetic data

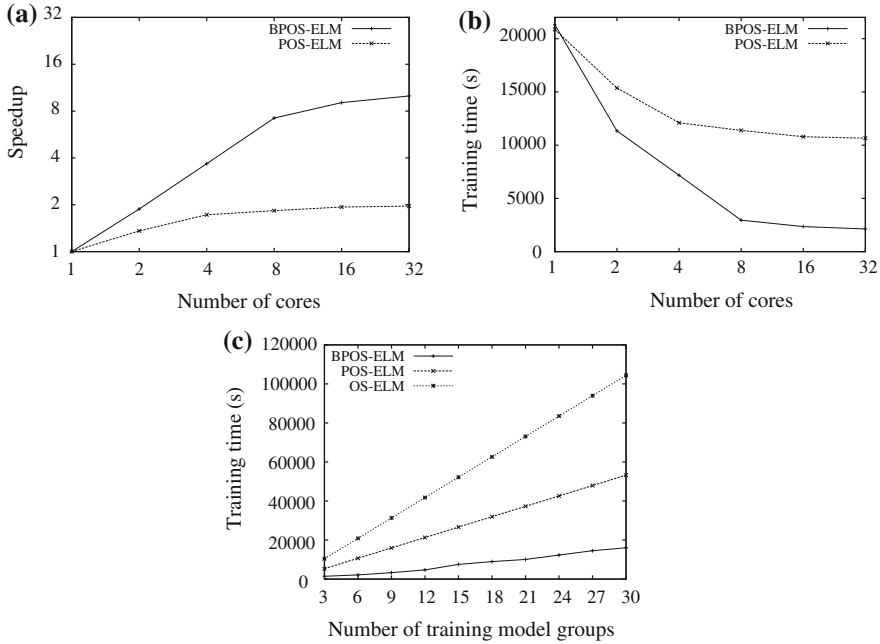
Data set	Algorithm	Training time (s)
Real data	OS-ELM	1146
	POS-ELM	1025
	BPOS-ELM	1021
Synthetic data	OS-ELM	20701
	POS-ELM	10651
	BPOS-ELM	2130

This result also shows that BPOS-ELM trains large scale multiple OS-ELM models efficiently.

### 3.2.3 Scalability Evaluation

Figure 3a shows the scalability (speedup) of BPOS-ELM compared with that of POS-ELM. The speedup of BPOS-ELM reaches 10 when the number of cores increases to 32. It means that BPOS-ELM has good scalability. It benefits from accurate estimations of Map and Reduce execution time and the execution plan which is suitable for parallel processing. It can also be found that the speedup of BPOS-ELM reaches 10 whereas the speedup of POS-ELM only reaches 1.96. The reason is that BPOS-ELM calculates  $\beta_{m,k}$  for different models in parallel instead of calculating them sequentially.

There are several reasons for the changing trend of speedup decreases as the number of cores increases. First, since the Reduce tasks cannot be further split into smaller ones, the execution time of the OS-ELM model which has the longest Reduce execution time does not decrease as the number of cores increases. In this case, the MapReduce job has to wait for the completion of the slowest task. Second, the cost of



**Fig. 3** Scalability evaluation

scheduling tasks among multiple cores increases as the number of cores increases. Third, the memory and the number of I/Os become bottlenecks as the number of cores increases since all the Map tasks and Reduce tasks running on a physical machine share the same memory and disks.

Figure 3b shows the training time of BPOS-ELM compared with that of POS-ELM. The training time of BPOS-ELM is a little longer than that of POS-ELM on one core due to the overhead derived from task scheduling. However, as the number of cores increases, the training time drops significantly and becomes much shorter than that of POS-ELM. It means that BPOS-ELM is more efficient than POS-ELM for training multiple models for the reason that BPOS-ELM trains multiple models in parallel in Reduce phase.

Figure 3c shows the training time of BPOS-ELM with different number of model groups compared with that of POS-ELM and OS-ELM. As shown in Fig. 3c, the training time increases much more slowly than that of POS-ELM and OS-ELM. The reason for this is that BPOS-ELM trains multiple OS-ELM models in parallel in both Map phase and Reduce phase whereas POS-ELM only parallelizes the training in Map phase and OS-ELM does not parallelize the training. It means that BPOS-ELM utilizes computing resources efficiently.

## 4 Conclusions

In this paper, a batch parallel online sequential extreme learning machine (BPOS-ELM) algorithm has been proposed for large scale batch learning. It estimates the execution time of Map and Reduce tasks with historical statistics, and generates a Map execution plan and a Reduce execution plan with greedy strategy based on the estimation. It launches a MapReduce job to train multiple OS-ELM models. The algorithm also collects information of selected tasks in the job and merges it to historical statistics to help to improve the estimation accuracy. BPOS-ELM algorithm is evaluated with real and synthetic data. The experimental results show that the accuracy of BPOS-ELM is at the same level as those of POS-ELM and OS-ELM. The speedup of BPOS-ELM reaches 10 on a cluster with maximum 32 cores. Compared with OS-ELM and POS-ELM, BPOS-ELM trains multiple OS-ELM models more efficiently.

**Acknowledgments** This research was partially supported by the National Natural Science Foundation of China under Grant nos. 61173030, 61272181, 61272182, 61173029, 61332014; and the National Basic Research Program of China under Grant no. 2011CB302200-G.

## References

1. Amazon elastic compute cloud (2015). <http://aws.amazon.com/cn/ec2/>
2. Arndt, H., Bundschuh, M., Naegele, A.: Towards a next-generation matrix library for java. In: Computer Software and Applications Conference, 2009. COMPSAC'09. 33rd Annual IEEE International. vol. 1, pp. 460–467. IEEE (2009)
3. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. *Commun. ACM* **51**(1), 107–113 (2008)
4. He, Q., Shang, T., Zhuang, F., Shi, Z.: Parallel extreme learning machine for regression based on mapreduce. *Neurocomputing* **102**, 52–58 (2013)
5. Huang, G.B., Chen, L.: Convex incremental extreme learning machine. *Neurocomputing* **70**, 3056–3062 (2007)
6. Huang, G.B., Zhou, H., Ding, X., Zhang, R.: Extreme learning machine for regression and multiclass classification. *IEEE Trans. Syst. Man Cybern. Part B: Cybern.* **42**(2), 513–529 (2012)
7. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine. In: Technical Report ICIS/03/2004. School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore (2004)
8. Liang, N.Y., Huang, G.B., Saratchandran, P., Sundararajan, N.: A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Trans. Neural Netw.* **17**(6), 1411–1423 (2006)
9. Shepard, D.: A two-dimensional interpolation function for irregularly-spaced data. In: Proceedings of the 1968 23rd ACM National Conference, pp. 517–524. ACM'68 (1968)
10. Wang, B., Huang, S., Qiu, J., Liu, Y., Wang, G.: Parallel online sequential extreme learning machine based on mapreduce. *Neurocomputing* **149**(Part A), 224–232 (2015)
11. Xiang, J., Westerlund, M., Sovilj, D., Pulkkis, G.: Using extreme learning machine for intrusion detection in a big data environment. In: Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop, pp. 73–82. AISec'14 (2014)
12. Xin, J., Wang, Z., Chen, C., Ding, L., Wang, G., Zhao, Y.: Elm\*: distributed extreme learning machine with mapreduce. *World Wide Web* pp. 1–16 (2013)

Proceedings of ELM-2015 Volume 1

Theory, Algorithms and Applications (I)

Cao, J.; Mao, K.; Wu, J.; Lendasse, A. (Eds.)

2016, IX, 532 p. 213 illus., 93 illus. in color., Hardcover

ISBN: 978-3-319-28396-8