Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Tensor Decomposition Based Approach for Training Extreme Learning Machines

Nikhitha K. Nair, Asharaf S.

Indian Institute of Information Technology and Management-Kerala (IIITM-K), Thiruvananthapuram, India

ARTICLE INFO

Article history:
Received 30 November 2016
Received in revised form 4 June 2017
Accepted 4 July 2017
Available online xxxx

Keywords:
ELM
TENSOR
PARAFAC
TUCKER
ALS
HOSVD

ABSTRACT

Conventional Extreme Learning Machines utilize Moore–Penrose generalized pseudo-inverse to solve hidden layer activation matrix and perform analytical determination of output weights. Scalability is the major concern to be addressed in Extreme Learning Machines while dealing with large dataset. Motivated by these scalability concerns, this paper proposes a novel tensor decomposition based Extreme Learning Machine which utilize PARAFAC and TUCKER decomposition based techniques in a SPARK platform. This proposed Extreme Learning Machine achieve reduced training time and better accuracy when compared with a conventional Extreme Learning Machine.

1. Introduction

Extreme Learning Machine has become a significant research topic in the field of machine intelligence and big data analytics due to its vibrant characteristics such as extremely fast training rate and universal regression/classification and approximation capability. ELM tend to be an effective solution for the drawbacks of single hidden layer feed-forward neural networks. It has wide applications in domains such as face classification [1,2], image segmentation [1,3] and human action recognition [1,4].

Traditional feed-forward neural networks exhibit potentials such as (1) approximation of complex nonlinear mappings directly from input samples; (2) inefficient handling of large classes of phenomena using different classical parametric techniques [5]. These neural networks have much slower learning algorithms than desired.

Extreme Learning Machines exhibit important characteristics such as 1) introducing minimum error from training process. 2) requirement of smallest normalised weights in the entire process and providing perfect generalisation performance [5,7,10]. These machines reveal uniqueness property with minimum normalised least-squares solution of $H\beta = T$, which is $\hat{\beta} = H^{+}T$ [5,7,10,11]. The most expensive computation in Extreme Learning Machines exist in the process of computing Moore–Penrose generalised inverse of hidden layer matrix [12]. Some of the most popular methods for computing this matrix include singular value decomposition method, iterative method, orthogonalization method and orthogonal method [12–15]. When nonsingular matrices are to be handled, both iterative and orthogonalization method show poor performance [12,13]. The singular value decomposition method tend to be an accurate method. This method prone to be time-consuming when massive data are handled [12,13,16].

The CANDECOMP and TUCKER decomposition techniques are two major tensor decompositions used in the field of data mining, classification and big data analytics. In case of tensors, there does not exist a single generalized Singular Value Decomposition. Both these techniques are considered as extensions of Singular Value Decomposition to adapt higher dimensions [25].

This paper contribute to develop a Tensor decomposition based Extreme Learning Machine on a Apache Spark Platform, which overcome the shortcomings of conventional Extreme Learning Machines implemented on Apache Spark Platform, whose leaning ability is pathetic to huge dataset. PARAFAC decomposition based tensor (ELM-PARAFAC) and TUCKER decomposition based tensor (ELM-TUCKER) are the two proposed scalable tensor decomposition based Extreme Learning Machines. Utilizing these tensor techniques, factor matrices are generated from the training data tensor, which is then trained using Extreme Learning Machines. Therefore, training massive data are made effective with these tensor decomposition techniques.

The rest of this paper is formulated as follows. Section 2 discuss some of the work related to the proposed approach. Section 3 contribute a brief overview of standard Extreme Learning Machines

E-mail addresses: nikhitha.mphilcs3@iiitmk.ac.in (N.K. Nair), asharaf.s@iiitmk.ac.in (Asharaf S.).

http://dx.doi.org/10.1016/j.bdr.2017.07.002
2214-5796/

and its algorithm, which is commonly used in classification and regression tasks. Section 4 discuss a brief overview of tensors and its basic preliminaries. Section 5 reflect the core part of this work, which discuss the real problem faced by standard Extreme Learning Machines, introduce the methodology adopted for solving the threat and implementation of proposed algorithm. Section 6 compare the performance of standard Extreme Learning Machines with that of proposed Tensorized Extreme Learning Machines. Conclusions and future scope of this work is drawn in Section 7.

2. Related work

Till now, Support Vector Machines are extensively being used for classification. These machines exhibit learning features such as: a) Mapping training data to a higher dimensional feature space with the help of a nonlinear feature mapping function b) Solution for maximum margin separation of two different classes determined using standard optimizing method. There is intensive computational complexity in the process of training support vector machines since quadratic programming is utilized [8].

The neural networks are normally trained with the help of a finite training dataset [5,7]. Huang and Babri [5,7,9] show that a single-hidden layer feed-forward neural network is capable to learn 'N' distinct observations with atmost 'N' hidden nodes using any nonlinear activation functions. These feed-forward neural networks were mostly used in combination with back-propagation (BP) algorithms for optimizing different parameters of neural networks. These algorithms belong to the criteria of gradient based methods. The main issues addressed by these methods include low convergence rate and problem of not achieving the proper global minimum [6]. The backpropogation algorithms deal with several issues such as: 1) Learning algorithm achieves low convergence rate, when learning rate is too small. 2) Learning algorithm stops easily at the local minimum, when the learning algorithm is located much far above the specified global minimum. 3) The traditional neural networks become over-trained by using these algorithms and achieve worse generalisation performance [7].

Infinitely differentiable activation functions allow random assigning of hidden layer biases and input weights in single-hidden layer feed-forward neural network. After randomly choosing weights in the input layer and biases in the hidden layer, the entire single layer feed-forward neural networks are treated as a linear system. The analytical determination of output weights of the neural networks are performed using simple inverse operation in the output matrices of the hidden layer [5].

These all concepts are combined together to form a simple generalized learning algorithm named Extreme Learning Machines (ELM). These Extreme Learning Machines have proven to exhibit thousand of times faster training speed when compared with traditional feed-forward network algorithms. It also achieve better performance in generalization [5,7].

The computation of Moore–Penrose generalised inverse of hidden layer matrix tend to be the most expensive and tedious computation while dealing with Extreme Learning Machines. According to Courrieu, fast computation of Moore–Penrose generalised inverse matrices are achieved using a full rank Cholesky factorization method [12,13]. Tensor product matrix Extreme Learning Machine (TPM-ELM) is another approach stated for computing Moore–Penrose pseudo-inverse of a large matrix [12,16,17]. Different variants of Extreme Learning Machines exist to overcome their shortcomings, which includes pruning ELM, two-stage ELM, voting-based ELM, fully complex ELM, incremental ELM, error-minimized ELM and symmetric ELM [39].

Tensor decomposition techniques tend to be among the emerging tools to analyze multi-dimensional aspects of data in many engineering and scientific fields such as machine learning, signal processing and chemometrics [18–22].

For improved data analysis, data with multiple aspects or modalities are collected, correlated and organized as tensors. There is a loss of internal structure information, when matrix factorization techniques are applied to matricized data. Tensor decomposition techniques provide multiple perspective stereoscopic view rather than flatten view provided by matrix factorization methodologies [18].

In matrix, there is a quadratic nature of increase in number of elements as product of rows and columns. In third-order tensors, number of elements in it increases cubically, as product of columns, rows and tubes. Thus for handling large tensors, there is always a need for tensor decomposition techniques [23].

Over these years, tensors are used for various purposes in the field of data mining and analytics [24]. There are different tensor decomposition approaches that are applied to multi-dimensional data. The higher-order generalizations of Singular Value Decomposition include Tucker decomposition, PARAFAC decomposition (parallel factor analysis) or CANDECOMP (canonical decomposition), Higher-Order Singular Value Decomposition (HO-SVD) and Higher-Order Orthogonal Iteration (HOOI) [23].

PARAFAC decomposition is used for decomposing a large tensor into a sum of rank-one tensors for finding the latent factors. TUCKER decomposition is mostly used for tensor compression as well as finding relations that exist between the latent factors [18, 25–29]. Since there exist an interaction among all pairs of factors, TUCKER decomposition tends to be considered as a generalised version of CANDECOMP decomposition [26].

The tensorization refers to the process of utilizing lower-dimensional original data for creating data tensors that have practical existence [30]. In case of data analysis, higher-order tensors are obtained by the process of tensorization of large-scale matrices or vectors. Nowadays, tensor decomposition techniques are applied for low tensor rank approximations. This works well for big data analytics [31].

Dealing with tensors and their decomposition approaches, intermediate data explosion problem is one of the major issues faced when handling large tensors [25–27]. GIGATENSOR, a scalable approach to tensor decomposition is capable of handling tera-scale tensors using MapReduce framework and their open source implementation in Hadoop platform [27].

The Alternating Least Square (ALS) approach is one of the popular methods used in PARAFAC and TUCKER tensor decomposition. While considering three variables and fixing two of them, the pseudo-inverse calculated by this method gives the solution of minimal norm for those variables [27–29].

The big data analytics always tends to adapt novel technologies that has the capability to handle immense datasets with high speed. Multiway analysis through tensor networks and tensor decompositions prone to be such emerging technologies to handle multidimensional big datasets. The idea behind these tensor decompositions are generating factor matrices by decomposing large training data tensors while interconnected lower-order tensors represent higher-order tensors using tensor networks. The challenge behind these tensor decompositions include finding an approach to analyze large-scale multiway data and perform different tasks such as classification, blind-source separation and compression [32].

Tensor decompositions have also shown immense applications in the area of multi-task learning. In this, multiple related tasks are learned simultaneously so that re-use of knowledge segregated from each task takes place. The multi-task ideas which is based on matrix factorization are generalised to tensor factorization, so that knowledge is shared flexibly in connected and convolutional deep neural networks [33].

Please cite this article in press as: N.K. Nair and Asharaf S., Tensor Decomposition Based Approach for Training Extreme Learning Machines, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.07.002

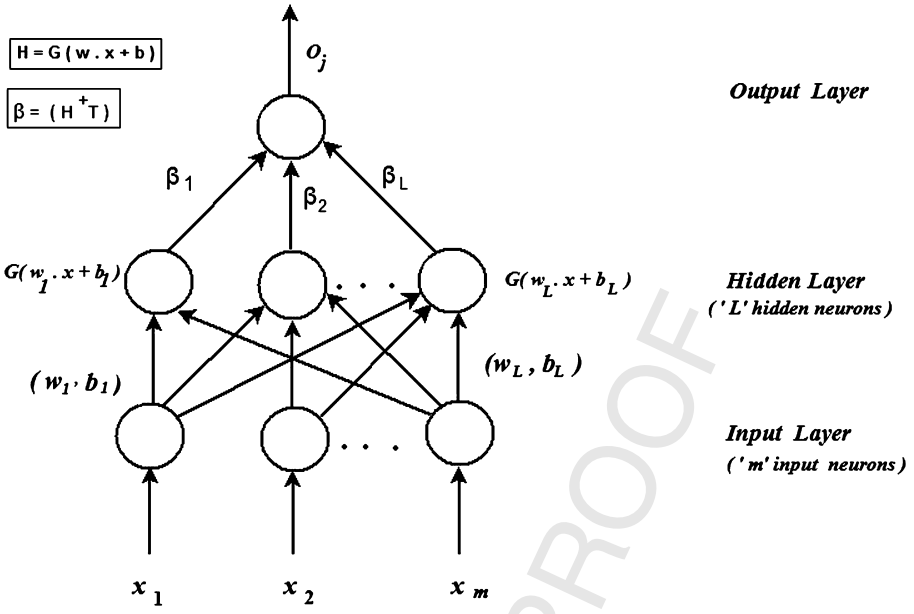

Fig. 1. Representation of Extreme Learning Machines.

The Higher-Order Singular Value Decomposition (HO-SVD) of a tensor is also utilized for handwritten digit classification. In this, training dataset is compressed using HOSVD or the basis matrices which describes unknown digits are computed using HO-SVD [34].

Distributed Extreme Learning Machines are also implemented in a MapReduce framework which utilises the parallel computing ability of MapReduce and handles massive training data. The computation of product of two large matrices in Extreme Learning Machines are transformed into a summation which fits the MapReduce methodology well [41]. Parallelised Extreme Learning Machines for the regression tasks are also implemented using MapReduce methodology [42]. Another approach to design distributed Extreme Learning Machines include multiple submodels of Extreme Learning Machines are trained in a distributed fashion using the distributed data blocks and a complete Extreme Learning Machine model is generated by combining these submodels [43]. Extreme Learning Machines are also introduced in a Spark platform for Resilient Distributed Dataset (RDD) programming and massive data processing. In this scenario, vSphere VMware platform provides an environment for carrying out parallel experiments [44].

3. A brief overview of standard Extreme Learning Machines

Extreme Learning Machines (ELM) map the input data to L-dimensional random feature space and generates the output function [6,10,36,37] of the form

$$f_L = \sum_{i=1}^{L} \beta_i h_i(x) = h(x)\beta \tag{1}$$

where $\beta = [\beta_1 \ldots \beta_L]^{\mathbf{T}}$ represents the output weight matrix between the hidden layer nodes and output layer nodes and $\mathbf{h}(\mathbf{x}) = [\mathbf{G_1}(\mathbf{x}).....\mathbf{G_L}(\mathbf{x})]$ represents the outputs of hidden nodes for the input data '**x**' [6,10,36,37].

ELMs are used to resolve the learning problems [1–17,35–38] of type

$$H\beta = T \tag{2}$$

where 'β' represents weight vector connecting the hidden nodes and output nodes [1–17,35–38].

$$\beta = [\beta_1.....\beta_L]^T_{L*m} \tag{3}$$

'**H**' represents hidden layer randomized output matrix [1–17, 35–38].

$$H = \begin{bmatrix} h(x_1) \\ h(x_2) \\ \vdots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & .... & h_L(x_1) \\ \vdots & & \\ h_1(x_N) & .... & h_L(x_N) \end{bmatrix}_{N*L} \tag{4}$$

'**T**' represents target label matrix [1–17,35–38].

$$T = \begin{bmatrix} t(x_1^T) \\ t(x_2^T) \\ \vdots \\ t(x_N^T) \end{bmatrix} = \begin{bmatrix} t_{11} & .... & t_{1m} \\ \vdots & & \\ t_{N1} & .... & t_{Nm} \end{bmatrix}_{N*m} \tag{5}$$

Extreme Learning Machines uses a set of '**N**' distinct samples (x_i, t_i) where $\mathbf{x_i} \in \mathbb{R}^{\mathbf{m}}$ and $t_i \in \mathbb{R}$ [1–17,35–38]. The single hidden layer feed-forward neural network having '**L**' number of hidden neurons exhibit output [7], which is written in the form

$$t_j = \sum_{i=1}^{L} \beta_i g(w_i x_j + b_i); \quad j = \epsilon[1, N] \tag{6}$$

Fig. 1 shows a representation of Standard Extreme Learning Machine. The training process of a single hidden layer feed-forward neural networks involves two main stages: (a) Feature Mapping randomly (b) Solving the parameters that are linear [10]. The first stage involves using some nonlinear mapping function to map the input data to some feature space. This is accomplished by initializing the hidden layer parameters randomly by Extreme Learning Machines.

3.1. Standard Extreme Learning Machine algorithm

Moore–Penrose Pseudo-inverse solution of a system of linear equations, used for calculating the output weights of Extreme Learning Machines can be explained as follows:

Suppose there is a requirement to solve a system of linear equations in which there are equations more in number than unknowns. Then the system can be simply described by the following equation:

$$\vec{T} = H\vec{X} \tag{7}$$

If the matrix H tends to be a square matrix, then we can proceed using the above equation: $\vec{T} = H\vec{X}$

$$H^{-1}\,\vec{T} = H^{-1}H\,\vec{X}\,H^{-1}\,\vec{T} = I\,\vec{T} \tag{8}$$

(where I tends to be an Identity Matrix).

$$\Rightarrow H^{-1}\,\vec{T} = \vec{T} \tag{9}$$

Now if a condition exists such that the matrix H is not a square matrix, then usual H^{-1} does not exists, instead there arises the importance of calculating the pseudo-inverse of matrix H.

$\vec{T} = H\,\vec{X}$

$H^{+}\,\vec{T} \approx H^{+}\,H\,\vec{X}$ (Moore–Penrose pseudo-inverse multiplied on both sides of the equation)

$H^{+}\,\vec{T} \approx I\,H\,\vec{X}'$ ($H^{+}H = I$ where I is Identity Matrix)

$$H^{+}\,\vec{T} \approx \vec{X} \tag{10}$$

Thus, Moore–Penrose pseudo-inverse can be computed using the following equation:

$$H^{+} = (H^{T}\,H)^{-1}\,H^{T} \tag{11}$$

where H^{+} represents the Moore–Penrose pseudo-inverse of the matrix H.

As mentioned in paper [7], Extreme Learning Machines exhibit the following properties:

(1) Extreme Learning Machines possess minimized training error.
(2) The norm of output weights of ELM tends to be smallest.
(3) The unique property of minimum norm least square solution of $H\beta = T$.

The algorithm of Standard Extreme Learning Machines includes the following consecutive steps:

Algorithm 1: Training Algorithm of Standard ELM.

Input: *Suppose we have a sample of training dataset* $\mathbf{S} = [(\mathbf{x_i}, \mathbf{t_i})|\mathbf{x_i} \in \mathbf{R^m}, \mathbf{t_i} \in \mathbf{R}, \mathbf{i} = \mathbf{1}, \mathbf{2}, \mathbf{3}, ...\mathbf{N}]$, *number of hidden nodes in the ELM 'L' and hidden node output function* ***G(w,b,x)***.

Output: *Output weights* β *of the ELM obtained from Moore–Penrose generalized inverse of hidden layer output matrix '****H****'.*

1 *Input weights and biases involved in the Extreme Learning Machines are assigned randomly.*
2 *The hidden layer output matrix '****H****' is calculated using Moore–Penrose inverse of a matrix.*
3 *Output weights calculated by multiplying Moore–Penrose pseudo-inverse of the hidden layer output matrix '****H****' and transpose of target samples.*

4. Overview of tensors and its preliminaries

Tensors can be recognised as a mutidimensional extension of two-way arrays such as matrices, which has two indices for representing the pair of rows and columns (i, j). In the case of tensors, which are characterised by three or more indices $(i, j, k, l, ...)$, represents higher-order arrays or higher-order tensors [19,26,28,29]

The mode or order of a tensor is defined by the number of dimensions which exists in tensor. The zero-order tensors are denoted as scalars, first-order tensors as vectors and second-order tensors as matrices. The higher-order tensors are the tensors possessing more than two number of dimensions [19,28,29].

In mathematical science, bold calligraphic alphabets are used to denote tensors such as $\mathcal{A}$, matrices are denoted by bold capital alphabets such as '**B**' and bold small alphabets are used to denote vectors such as '**v**'. The $\mathbf{v_i}$ denotes the representation of ith element of a vector, $\mathbf{b_{i,j}}$ indicates the representation of element (i, j) of a matrix **B** and $a_{i,j,k,l}$ denotes the representation of element (i, j, k, l) of a fourth-order tensor $\mathcal{A}$ [19,22,23,28,29].

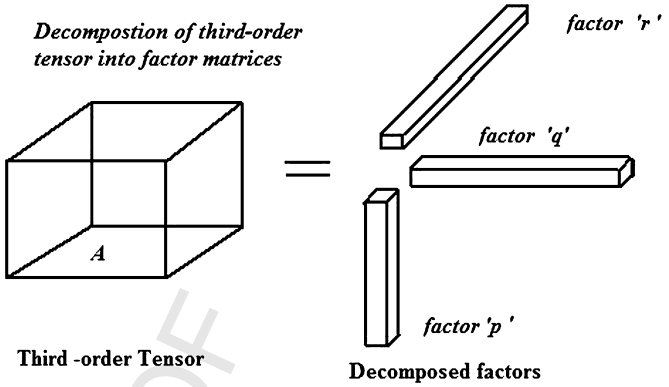

Fig. 2. Representation of a third-order tensor.

The Fig. 2 shows a generalized third-order tensor and their decomposition into three factors.

Subtensors are created from original data tensors by keeping subset of indices fixed in various applications.The subtensors can be fibers when they are vector-valued or can be slices when they are matrix-valued. Tensor fibers are one-dimensional subset of a tensor, obtained by fixing all the indices but one index of the original tensor [19,28,29].

In a third-order tensor, row fibers represented as $a_{i:k}$, column fibers represented as $\boldsymbol{a_{:jk}}$ and tube fibers represented as $a_{ij:}$ [19, 24,28,29].

Tensor slices tends to be two-dimensional subset of original tensors obtained by fixing all the indices but two indices of the original tensor [19,24,28,29]. In a third-order tensor, lateral slice represented as $\boldsymbol{a_{:j:}}$, horizontal slice represented as $\boldsymbol{a_{i::}}$ and frontal slice denoted as $\boldsymbol{a_{::k}}$. Fig. 3 provides a representation of slices in third-order tensors.

4.1. Basic operations performed on tensors

- Matricizing: Reshaping or reformatting is required for performing various manipulations in tensors. Matrix unfolding is a particular case of performing such reshaping of tensors into matrices [19,22,23,28–30]. Fig. 4 shows an outlook of matricization performed on third-order tensor.
  The final outcome of such an unfolding, result in flattening of tensors into matrices. The elements in a N-way array or tensors are then rearranged into a matrix. Fig. 5 shows an example of unfolding performed on third-order tensor.
  Given a third-order tensor $Z \in \mathbb{R}^{I\times J\times K}$, then $Z_{(n)}$ denote mode-n matricization performed on tensor Z [19,22,23,28, 29]which includes

$$Z_{(1)} = [Z_1, Z_2.....Z_K] \in \mathbb{R}^{I\times(JK)} \tag{12}$$

$$Z_{(2)} = [Z_1^T, Z_2^T, ...Z_K^T] \in \mathbb{R}^{J\times(IK)} \tag{13}$$

$$Z_{(3)} = [vec(Z_1), ...vec(Z_1)]^T \in \mathbb{R}^{K\times(IJ)} \tag{14}$$

- Tensor Addition: Addition of two tensors possible based upon the equality of their dimensions. Suppose there are two tensors $\mathcal{P}$ and $\mathcal{Q}$ such that $\mathcal{P}$ belongs to $\mathbb{R}$ over $n_1 \times n_2 \times n_3 \times$ and $\mathcal{Q}$ belongs to $\mathbb{R}$ over $n_1 \times n_2 \times n_3$, then there exists an additive tensor $\mathcal{S}$ such that $\mathcal{S}$ belongs to $\mathbb{R}$ over $n_1 \times n_2 \times n_3$ where $s_{i,j,k} = p_{i,j,k} + q_{i,j,k}$ [19,28,29].
- Transpose of a Tensor: The transpose of a third-order tensor is achieved when first face of the tensor is preserved and subsequent faces are reversed [19,28,29].

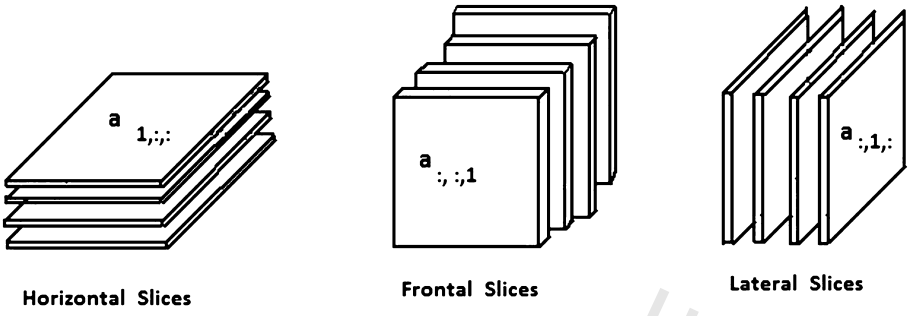

Fig. 3. Representation of tensor slices.

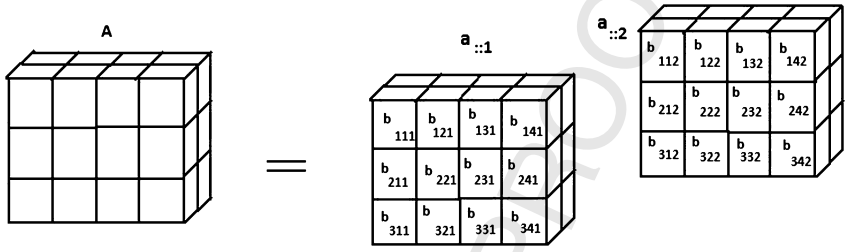

Fig. 4. Example of unfolding a third-order tensor in three modes.

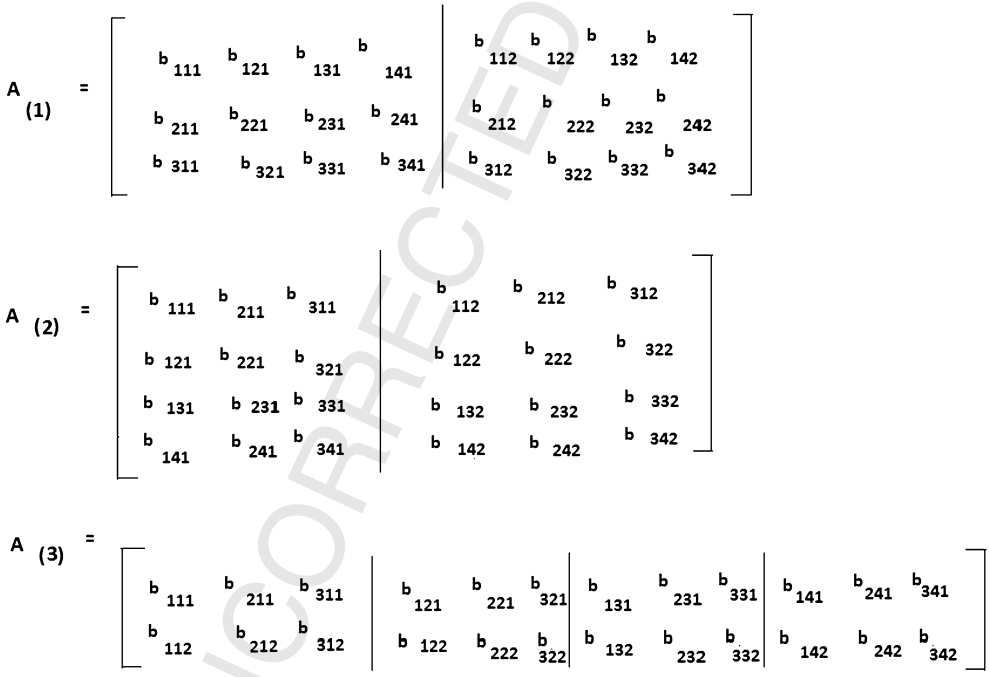

Fig. 5. Example of unfolding a third-order tensor in three modes.

- Multiplication of Tensor: Two third-order tensors can be multiplied only when first matrix tends to be a block circulant matrix and the second matrix tends to be a block column vector, obtained from the consecutive faces of the corresponding matrices. The multiplication of a tensor times a vector, tensor times a matrix and tensor times a tensor is possible in numerous scenarios. The n-mode product performed on various applications gives the tensor $\mathcal{A} \in \mathbb{C}^{I_1 \times I_2 \times \ldots \times I_N}$ and a matrix $B \in \mathbb{C}^{J \times I_n}$, then $(\mathcal{A} \cdot B)_{i_1 \ldots i_{n-1} j i_{n+1} \ldots \ldots i_N} = \sum_{i_n=1}^{I_n} a_{i_1 i_2 \ldots \ldots i_N} b_{j i_n}$ [19,27–29].

  The Kronecker product denoted by $\otimes$, Khatri–Rao product denoted by $\odot$, and Hadamard product denoted by $*$ are the commonly used tensor products used for various tensor manipulations [19,22,25–29,32].

4.2. Basic decompositions performed on tensors

The CANDECOMP/PARAFAC decomposition and Tucker decomposition models are the two most popular decompositions performed on tensors [18–20,25–31].

- PARAFAC decomposition: The process of decomposing a tensor in order to obtain sum of rank-one tensors is known as Parallel factor analysis or Canonical Decomposition [18,22,25–31]. Fig. 6 provides a representation of PARAFAC decomposition applied on higher order tensors.
  Given a three-way tensor $\mathcal{A}$ and an approximation rank U, the factor matrices can be defined as the combination of vectors from rank-one components.

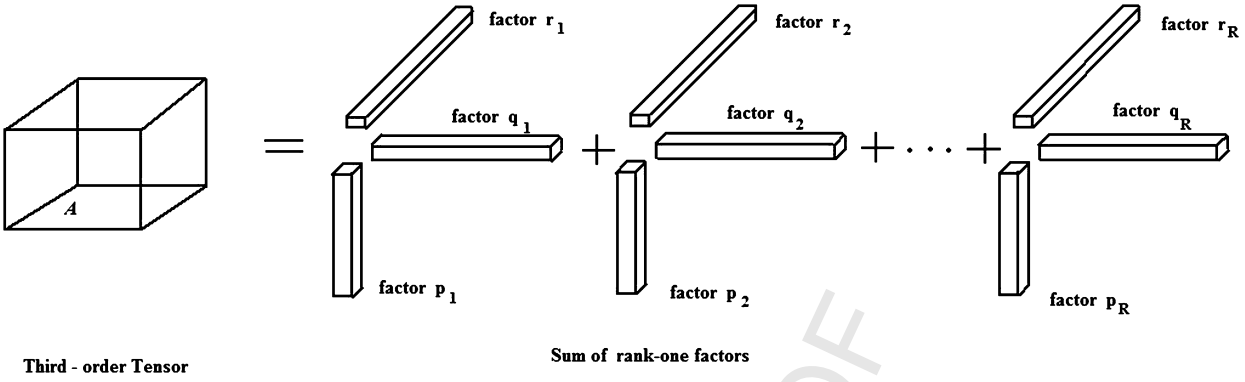

Fig. 6. Representation of PARAFAC decomposition.

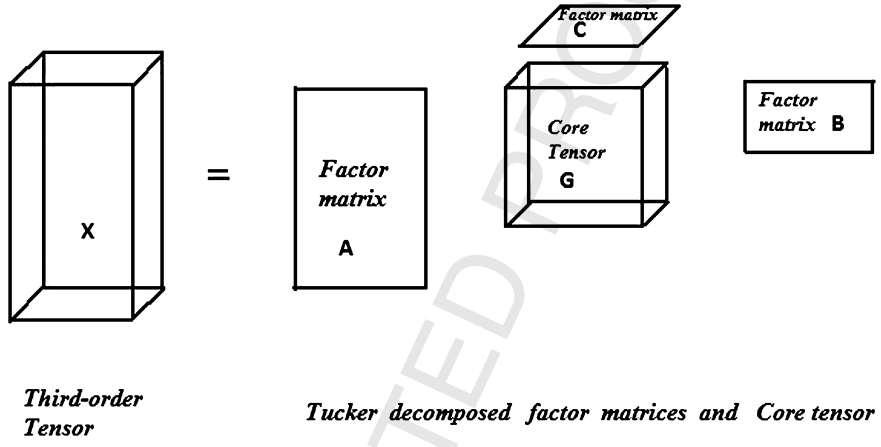

Fig. 7. Representation of Tucker decomposition of a three-way tensor.

For example, given a third-order tensor $\mathbf{a} \in \mathbb{R}^{I*J*K}$, we can write the tensor [18,19,25–29] in the form

$$a \approx \sum_{u=1}^{U} p_u \circ q_u \circ r_u \tag{15}$$

where $p_u \in \mathbb{R}^I$, $q_u \in \mathbb{R}^J$ and $r_u \in \mathbb{R}^K$ for $u = 1, \ldots, U$.
The factor matrices includes [18,19,25–29]:

$$\mathcal{A}_{(1)} \approx P(R \odot Q)^T \tag{16}$$

$$\mathcal{A}_{(2)} \approx Q(R \odot P)^T \tag{17}$$

$$\mathcal{A}_{(3)} \approx R(Q \odot P)^T \tag{18}$$

- TUCKER decomposition: Tucker decomposition can be interpreted as a Higher Order generalization of Singular Value Decomposition (HOSVD) [18,22,23,25]. It decomposes a tensor into a core tensor multiplied (or transformed) by a matrix along each mode [18–26,28–31].

Fig. 7 provides a representation of TUCKER decomposition applied on higher order tensors. Thus, in the three-way case where $x \in \mathbb{R}^{I*J*K}$, TUCKER decomposition [18,19,22,23,25–29] can be represented in the form

$$\mathcal{X} \approx G * A_1 * B_2 * C_3 = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{s=1}^{\mathbb{S}} G_{pqs} a_p \circ b_q \circ c_s = [G; A, B, C]. \tag{19}$$

Here, $A \in \mathbb{R}^{I*P}$, $B \in \mathbb{R}^{J*Q}$, and $C \in \mathbb{R}^{K*S}$ are the factor matrices.

The tensor 'G' $\in R^{P*Q*S}$ is called the core tensor and its entries show the level of interaction between the different components [18,25–29].

By applying SVD to $X_{(n)}$, $n = 1, 2, \ldots, N$, the Tucker decomposition of a given tensor is obtained, which is referred to as the Higher-Order SVD (HOSVD) [18,19,22–25,28,29]. It has wide applications in multidimensional data analysis.

5. Tensor decomposition based ELM

There is a tendency for the traditional machine learning techniques to fail when they face large volume of data. They are incapable of loading all the data into the memory at once. However, volume of data in real life applications, becoming larger and even the standard Extreme Learning Machines cannot handle immense data in an efficient manner.

In standard Extreme Learning Machines, most of the time is wasted on the computation of matrix Moore–Penrose pseudo-inverse in the output vector.

The proposed Extreme Learning Machine has the capability to handle massive data, while reducing the training time and maintaining a comparable level of accuracy.

The proposed system is approached by considering two different stages. The first stage contributes to building a standard Extreme Learning Machine using randomly assigned hidden node parameters such as input weights and biases. This is followed by hidden layer output matrix calculation using Moore–Penrose pseudo-inverse operation and calculating the output weights related to the Extreme Learning Machines.

This model of Extreme Learning Machine is comparatively better than other machine learning techniques such as artificial neural networks and support vector machines.

Please cite this article in press as: N.K. Nair and Asharaf S., Tensor Decomposition Based Approach for Training Extreme Learning Machines, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.07.002

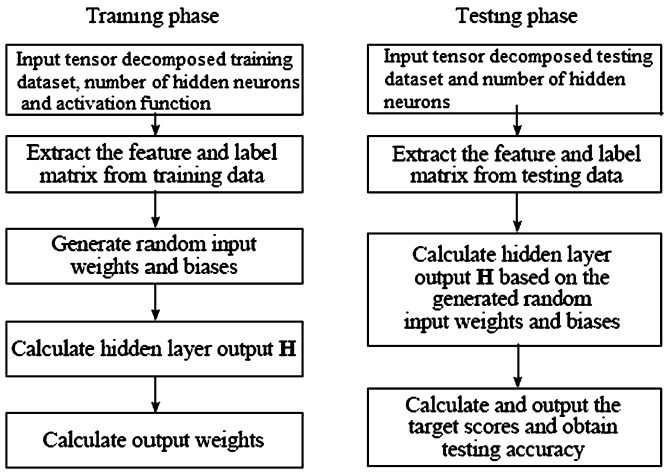

Fig. 8. Workflow of tensor decomposition based Extreme Learning Machine.

After analyzing the entire working, it was found that this basic model takes more time in its training process when volume of data becomes larger and larger. This increase in training time is due to large time spent on calculating the hidden layer output matrix in order to find the output weights.

The second stage attempts to reduce the time dealing with calculation of hidden layer output matrix and thereby reducing the training time. The approach attempted for this purpose includes introducing tensor decomposition techniques into the standard Extreme Learning Machines.

5.1. Implementation of tensor decomposition based Extreme Learning Machine

Fig. 8 illustrates the work flow of a tensor based Extreme Learning Machine. The working of a tensor based Extreme Learning Machine consists of Training and Testing phase.

The tensor based training dataset is first fed into the ELM for training. For the training phase in ELM, three arguments such as training dataset samples, number of hidden neurons '**L**' and an activation function '**G**' are used. The feature matrix and label matrix are extracted from the training dataset. Next step is to generate random input weights '**W**' and random biases '**b**'.

The hidden layer output matrix '**H**' is calculated based on the extracted feature matrix, input weight matrix '**W**', biases '**b**' and specified activation function such sigmoid, tanh, hardlims, tribas or sine. Output weights are calculated, which involves calculating the Moore–Penrose inverse of the hidden layer matrix '**H**'.

The testing phase involves using a testing dataset, and then extracting the expected labels and feature matrix from the given testing dataset.

Next hidden layer output matrix '**H**' is obtained based on random input weights, biases and specified activation function. The target scores are calculated in order to obtain the actual labels. The overall accuracy is obtained by comparing the actual labels with the expected labels.

5.2. Training algorithm for tensor decomposition based Extreme Learning Machines

The training Algorithm for Tensor decomposition based Extreme Learning Machines includes the following consecutive steps:

Algorithm 2: Training Algorithm for tensor decomposition based Extreme Learning Machine.

Input: *Suppose we have a sample of training dataset* $\mathbf{S} = [(\mathbf{x_i}, \mathbf{t_i}) \mid \mathbf{x_i} \in \mathbf{R}, \mathbf{t_i} \in \mathbf{R}, i = 1, 2, 3, ...N]$, *number of hidden nodes in the ELM '**L**' and hidden node output function **G(w,b,x).***

Output: *Output weights β of the ELM obtained from Moore–Penrose generalized inverse of hidden layer decomposed output matrix '**H**'.*

1 *Input weights and biases involved in the Extreme Learning Machines are assigned randomly.*
2 *Tensor decomposition performed on the large tensorised data and fed into the Extreme Learning Machine for training.*
3 *The hidden layer output matrix '**H**' calculated on the decomposed tensorised data using Moore–Penrose pseudo-inverse operation on the matrix.*
4 *Output weights calculated by multiplying Moore–Penrose pseudo-inverse of the hidden layer decomposed tensorised output matrix '**H**' and transpose of target samples.*

5.3. Enforcing CANDECOMP/PARAFAC decomposition technique on Extreme Learning Machines

The PARAFAC decomposition on Extreme Learning Machines has the peculiarity to present higher dimension tensors as sum of rank-one tensors [18,19,28,29]. The PARAFAC decomposition when applied on tensors can be written as combination of three matrices P, Q, R. The uth column of P contains p_u, uth column of Q contains q_u and uth column of R contains r_u.

Furthermore, the factor matrices can be made unit norm by introducing a term which is scalar such as λ_u to each rank-one factor. This scalar term is introduced after normalizing each column of the obtained three factor matrices, which is accumulated after performing tensor decomposition. The PARAFAC decomposition [18,19,22,28,29] when applied on ELM computed using the equation

$$a \approx \sum_{u=1}^{L} \lambda_u p_u \circ q_u \circ r_u \tag{20}$$

The Alternating Least Squares (ALS) method considered to be the most popular method used for fitting the PARAFAC decomposition. For a third-order tensor, the algorithm consists of three steps, where each step represents conditional update of only one of the three factor matrices at a time, given the other two. The key idea is to minimize the sum of squares between the original tensor and the factorized model of the tensor [19,28,29].

When training dataset is given, the ALS method compute the least square subproblems in order to obtain the three factor matrices P, Q and R. Given three initial factor matrices P^0, Q^0 and R^0, the ALS algorithm first solves for P by fixing Q and R, then solves for Q by fixing P and R, and solves for R by fixing P and Q. There is a conditional update of all the three factor matrices. This process continues iteratively until some convergence criteria gets satisfied. When the maximum number of iterations is reached or least square cost for two consecutive iterations stops changing significantly, then the stoping criteria of ALS is achieved.

5.4. Enforcing TUCKER decomposition technique on Extreme Learning Machine

The computation of Tucker Decomposition of tensors are commonly carried out using Higher Order Singular Value Decomposition [19,28,29]. In this approach, computation of singular value decomposition takes place on each n-mode matricized component.

The HOSVD can be written in terms of n-mode product of tensors [22–24,28,29] as

$$A = S \times_1 \mathrm{U} \times 2\mathrm{V} \times_3 \mathrm{W} \tag{21}$$

Algorithm 3: PARAFAC decomposition based Extreme Learning Machine.

Input: Tensor: $x \in \mathbb{R}^{I*J*K}$, rank R, Maximum Iterations T, hidden node output function $G(a, b, x)$, and the number of hidden nodes L.
Output: Output weights β using Moore–Penrose generalized inverse of hidden layer output matrix H.

Initialise factor matrices P, Q and R;
for $i = 1, \ldots..$ Maximum Iterations ***do***
  Factor Matrix $P \leftarrow$ Multiplying Mode (1) of tensor with Khatri–Rao product of (R and Q) and pseudo-inverse of Hadamard product of ($R^T R$ and $Q^T Q$).
  Normalise the columns of factor matrix A using the scalar term λ.
  Factor Matrix $Q \leftarrow$ Multiplying Mode (2) of tensor with Khatri–Rao product of (R and P) and pseudo-inverse of Hadamard product of ($R^T R$ and $P^T P$).
  Normalise the columns of factor matrix B using the scalar term λ.
  Factor Matrix $R \leftarrow$ Multiplying Mode (3) of tensor with Khatri–Rao product of (Q and P) and pseudo-inverse of Hadamard product of ($Q^T Q$ and $P^T P$).
  Normalise the columns of factor matrix C using the scalar term λ.
    if stopping or convergence criterion is met ***then***
      break the for loop;
    end if
end for
return *PARAFAC decomposed tensor, which is given to Extreme Learning Machine for training and calculating the output weights β.*

where $\mathcal{A}$, S belongs to $\mathbf{R^{I\times J\times K}}$, U belongs to $\mathbf{R^{I\times I}}$, V belongs to $\mathbf{R^{J\times J}}$ and W belongs to $\mathbf{R^{K\times K}}$.

The Higher Order Singular value decomposition technique, always applicable for higher order tensors, as it can be obtained by taking singular value decomposition of each of the flattening matrices.

Algorithm 4: TUCKER decomposition based ELM.

Input: Tensor: $T \in \mathbf{R^{I_1\times I_2\times I_N}}$, rank R, hidden node output function $G(a, b, x)$, and the number of hidden nodes C.
Output: Output weights β using Moore–Penrose generalized inverse of hidden layer output matrix H.

Construct different modes of matrix $\mathbf{T_{(k)}}$ that corresponds to tensor $\boldsymbol{T}$.
Compute the singular value decomposition on $\mathbf{T_{(k)}}$ using orthogonal matrix $\mathbf{U_k}$, diagonal matrix $\sum_{\mathbf{k}}$ and orthogonal matrix $\mathbf{V_k^T}$. Store the left singular vectors $\mathbf{U_k}$.
The core tensor $\boldsymbol{G}$ is tensor $\boldsymbol{T}$ times different tensor basis obtained by factor matrices $\mathbf{\prod_{n=1}^{N} U^T}$.
Return the Tucker decomposed tensor, which is given to Extreme Learning Machine for training and calculating the output weights β.

By applying singular value decomposition to each modes of tensor $\mathbf{T_{(n)}}$, the tucker decomposition of a given tensor is obtained; which is referred to as Higher Order Singular Value Decomposition.

The TUCKER decomposed tensor having core tensor and the factor matrices are given to the ELM for training and calculating the output weights.

6. Empirical results

This section presents experimental results obtained from different tensor decomposition techniques applied on standard Extreme Learning Machines.

6.1. Datasets used in the implementation of tensor based ELM

The dataset used for implementing Tensor decomposition based Extreme Learning Machines are from UCI Machine Learning Repository.

- MNIST handwritten dataset[1]: The process of digit recognition deals with classifying data in the MNIST dataset. The dataset contains 42000 training samples with 784 features along with labels. Each row in the dataset includes a label that forms the handwritten digit as the first column. Remaining columns represents pixel data color values that ranges from 0 to 255. The test dataset contains 8400 samples with 784 features along with labels [34].
- KDD Cup 1999 dataset[2]: KDDCup 1999 intrusion detection dataset contains 60000 training samples with 41 features along with one label. The test dataset contains 24000 samples with 41 features along with one label. The label values indicate normal or malicious attacks. The abnormal/malicious attack values in the training dataset further subdivided into four main attack groups and twenty two specific attack types. The four main attack groups include: Denial of Service (DoS); User to Root (U2R); Remote to Local (R2L); and Probing Attack. The twenty two specific attack types include land, neptune, smurf, pod, back, teardrop, portsweep, ipsweep, satan, nmap, multihop, spy, phf, warezclient, guess_passwd, ftp_write, warezmaster, imap, buffer_overflow, loadmodule, perl and rootkit. The KDD Cup 1999 dataset features are classified into three groups: Basic Features, Content Features, Time-based Traffic Features, Host-based Traffic Features [39,40].
- Statlog (Landsat Satellite) dataset[3]: Statlog dataset contains multi-spectral values of pixels of a satellite image. The training dataset contains 4435 samples with 36 attributes and 7 decision classes. Twenty percent of the training samples are used for testing. The classes include red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble, mixture class (all types present) and very damp grey soil.
- Statlog (Shuttle) dataset[4]: Shuttle dataset contains 43500 training samples with 9 attributes. Forty percent of the training samples are used for testing. The dataset include the following classes: Rad Flow, Fpv Close, Fpv Open, High, Bypass, Bpv Close, Bpv Open.
- Letter Recognition dataset[5]: Letter Recognition dataset identify large number of pixel displays as belonging to one of the twenty six capital letters in English alphabet. The training dataset contains 20000 samples with 16 attributes with one letter category. Twenty percent of the training samples are used for testing including labels. Labels used for identification are twenty six capital letters in English alphabet.
- Mushroom dataset[6]: Mushroom dataset include descriptions of 23 species of grilled mushrooms of Lepiota and Agaricus Family. The dataset contains 8124 training samples and 22 attributes with one category of labels. The labels used for classification include edible, poisonous or unknown.

6.2. Performance evaluation

For evaluation, standard Extreme Learning Machine is built using randomly assigned hidden node parameters, calculating hidden layer output matrix and determining the output weights. The entire process of building a standard Extreme Learning Machine, focused on attempting the training process using different datasets such as MNIST handwritten dataset, KDD Cup 1999 dataset, Statlog (Landsat Satellite) dataset, Statlog (Shuttle) dataset, Letter Recognition dataset and Mushroom dataset and predicting their accuracy.

[1] http://yann.lecun.com/exdb/mnist/.
[2] https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
[3] https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite).
[4] https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle).
[5] https://archive.ics.uci.edu/ml/datasets/letter+recognition.
[6] https://archive.ics.uci.edu/ml/datasets/mushroom.

Please cite this article in press as: N.K. Nair and Asharaf S., Tensor Decomposition Based Approach for Training Extreme Learning Machines, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.07.002

Table 1
Training time results of basic ELM, ELM-TUCKER and ELM-PARAFAC.

Training time (s) – MNIST dataset			
Activation	Basic ELM	ELM-TUCKER	ELM-PARAFAC
Sine	12332.981	10643.794	10555.316
Tribas	12296.731	10474.998	10414.283
Sigmoid	12298.065	10484.763	10421.326

The second stage attempts to reduce the time taken for calculating the hidden layer output matrix and there by reducing the training time of ELM.

Tensor decomposition based approaches such as PARAFAC decomposition and TUCKER decomposition techniques are adopted for the training process of ELM.

The implementation of entire work is carried out in a fast, generic and easy to use big data computing framework called Apache Spark 1.6.1. The Spark works on Resilent Distributed Datasets (RDD) which tend to be an immutable collection of data objects. Here a single Apache Spark server is used and RDDs are created by default parallelizing of an existing collection in the driver program. The Basic ELM, ELM-TUCKER and ELM-PARAFAC are implemented on Apache Spark 1.6.1. The programming language used for the entire implementation is Python.

6.2.1. Performance evaluation on MNIST handwritten dataset

Table 1 provides a comparative analysis of basic ELM, ELM-TUCKER and ELM-PARAFAC on the basis of training time (in seconds) using MNIST handwritten dataset. Basic ELM, ELM-TUCKER, ELM-PARAFAC are implemented on Apache Spark 1.6.1 and treated with different activation functions such as sine, tribas and sigmoid.

The results reveal more time in the training process of Basic ELM, when volume of data becomes larger and larger. This increase in training time is due to more time spent on calculating the hidden layer output matrix in order to find the output weights. There is reduction in training time by ELM-TUCKER and ELM-PARAFAC than that of Basic ELM in different activation functions. The Basic ELM, ELM-TUCKER and ELM-PARAFAC are implemented on Apache Spark 1.6.1.

Training time of Basic ELM versus ELM-TUCKER and ELM-PARAFAC using MNIST handwritten dataset is graphically represented in Fig. 9. The ELM-TUCKER and ELM-PARAFAC reveals a reduced training time when compared with that of Basic ELM.

Table 2 shows a comparison of testing accuracy, precision and recall when Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with activation functions such as sine, tribas and sigmoid. ELM-Tucker and ELM-PARAFAC shows increased accuracy than that of basic ELM using MNIST handwritten dataset.

6.2.2. Performance evaluation on KDD Cup 1999 dataset

Table 3 provides a comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC on the basis of training time (in seconds) using KDD CUP 1999 dataset. Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with different activation functions such as sigmoid, tanh, hardlims and tribas. The results reveal reduced training time by ELM-TUCKER and ELM-PARAFAC than that of basic ELM in different activation functions. The Basic ELM shows memory issues when handling KDDCup99 data samples above 60000.

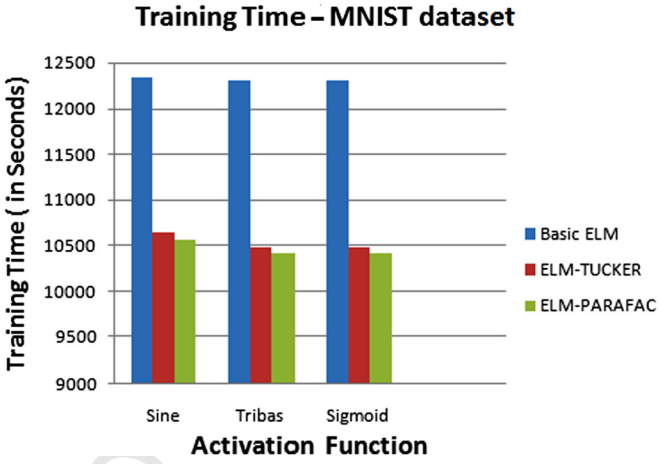

Fig. 9. Comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC.

Table 3
Training time results of basic ELM, ELM-TUCKER and ELM-PARAFAC.

Training time (s) – KDDCUP99 dataset			
Activation function	Basic ELM	ELM-TUCKER	ELM-PARAFAC
Tanh	14198.517	12258.568	12182.942
Sigmoid	14190.885	12280.145	12165.450
Hardlims	14185.134	11677.938	11592.199
Tribas	14176.046	11669.836	11488.908

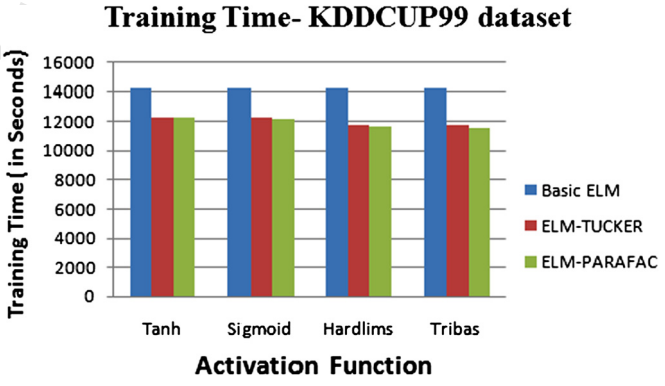

Fig. 10. Comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC.

The Basic ELM, ELM-TUCKER and ELM-PARAFAC are implemented on Apache Spark 1.6.1.

Training time of Basic ELM versus ELM-TUCKER and ELM-PARAFAC using KDDCUP99 dataset is graphically represented in Fig. 10. The ELM-TUCKER and ELM-PARAFAC reveals a reduced training time when compared with that of Basic ELM.

Table 4 shows a comparison of testing accuracy when Basic ELM, ELM-PARAFAC and ELM-PARAFAC are treated with activation functions such as sigmoid, tanh, hardlims and tribas.

A comparable testing accuracy is achieved when Extreme Learning Machine is treated with tensor decomposition techniques. The results reveal that by incorporating tensor decomposition techniques such as PARAFAC and TUCKER decomposition, training time

Table 2
testing accuracy results of basic ELM, ELM-TUCKER and ELM-PARAFAC-MNIST dataset.

Activation	Basic ELM			ELM-TUCKER			ELM-PARAFAC		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Sine	0.0991	0.10	0.10	0.2154	0.21	0.22	0.227	0.22	0.23
Tribas	0.25011	0.23	0.25	0.4527	0.46	0.45	0.4780	0.48	0.48
Sigmoid	0.8595	0.86	0.86	0.8732	0.88	0.87	0.8790	0.88	0.88

Please cite this article in press as: N.K. Nair and Asharaf S., Tensor Decomposition Based Approach for Training Extreme Learning Machines, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.07.002

Table 4
Testing accuracy results of basic ELM, ELM-TUCKER and ELM-PARAFAC-KDDCUP99 dataset.

Activation	Basic ELM			ELM-TUCKER			ELM-PARAFAC		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Tanh	0.9954	0.99	0.98	0.9964	0.99	0.99	0.9964	0.99	0.99
Sigmoid	0.9959	0.97	0.95	0.9959	0.99	0.99	0.9962	0.98	0.98
Hardlims	0.9629	0.95	0.96	0.9976	0.97	0.97	0.9982	0.99	0.99
Tribas	0.9968	0.98	0.97	0.9972	0.99	0.99	0.9987	0.98	0.99

Table 5
Training time results of basic ELM and ELM-PARAFAC.

Training time (s) – Satellite dataset			
Activation function	Basic ELM	ELM-TUCKER	ELM-PARAFAC
Hardlims	124.215	116.012	108.011
Tribas	124.228	102.624	98.927

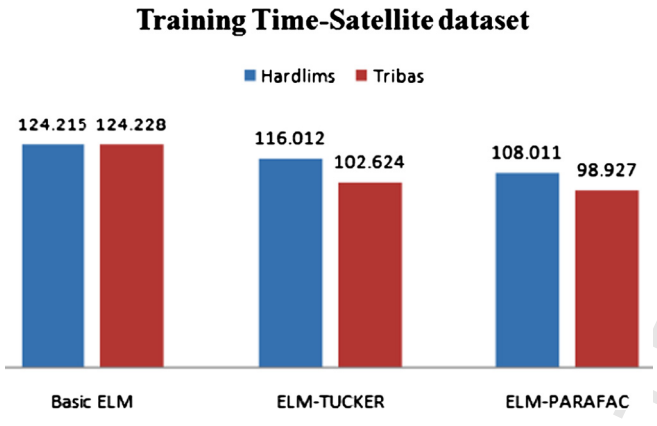

Fig. 11. Comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC.

in Extreme Learning machine is reduced than that of basic Extreme Learning Machines.

6.2.3. Performance evaluation on satellite dataset

Table 5 provides a comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC on the basis of training time (in seconds) using Satellite dataset. Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with different activation functions such as hardlims and tribas. The results reveal slight variation in training time by ELM-PARAFAC and ELM-TUCKER than that of basic ELM in different activation functions since size of the dataset is small. The Basic ELM, ELM-TUCKER and ELM-PARAFAC are implemented on Apache Spark 1.6.1.

Training time of Basic ELM versus ELM-TUCKER and ELM-PARAFAC using Satellite dataset is graphically represented in Fig. 11. The ELM-TUCKER and ELM-PARAFAC reveals a reduced training time when compared with that of Basic ELM.

Table 6 shows a comparison of testing accuracy when Basic ELM and ELM-PARAFAC are treated with activation functions such as sigmoid, tanh, hardlims and tribas.

A comparable testing accuracy is achieved when Extreme Learning Machine is treated with tensor decomposition techniques. The results reveal that by incorporating tensor decomposition techniques such as TUCKER decomposition, training time in Extreme

Table 7
Training time results of basic ELM, ELM-TUCKER and ELM-PARAFAC.

Training time (s) – SHUTTLE dataset			
Activation function	Basic ELM	ELM-TUCKER	ELM-PARAFAC
Tribas	6716.550	6376.865	6315.289
Tanh	6725.434	6680.117	6617.240
Hardlims	6706.932	6629.858	6584.301

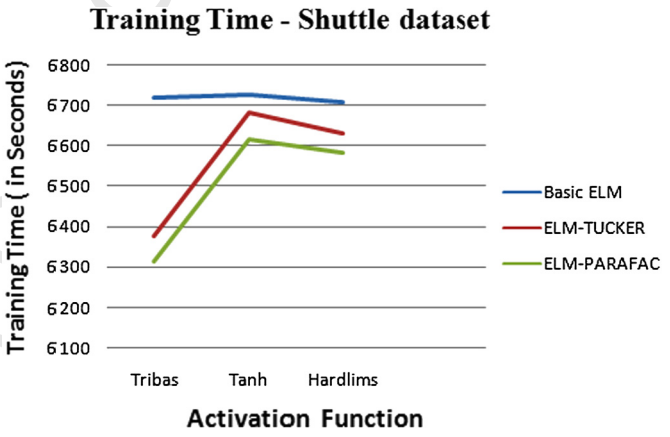

Fig. 12. Comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC.

Learning machine is reduced than that of basic Extreme Learning Machines.

6.2.4. Performance evaluation on Shuttle dataset

Table 7 provides a comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC on the basis of training time (in seconds) using shuttle dataset. Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with different activation functions such as tanh, hardlims and tribas. Results reveal ELM-TUCKER and ELM-PARAFAC have reduced trained time when compared with Basic ELM. The Basic ELM, ELM-TUCKER and ELM-PARAFAC are implemented on Apache Spark 1.6.1.

Training time of Basic ELM versus ELM-TUCKER and ELM-PARAFAC using Shuttle dataset is graphically represented in Fig. 12. The ELM-TUCKER and ELM-PARAFAC reveals a reduced training time when compared with that of Basic ELM.

Table 8 shows a comparison of testing accuracy when Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with activation functions such as sigmoid, tanh, hardlims and tribas.

A comparable testing accuracy is achieved when Extreme Learning Machine is treated with tensor decomposition techniques. The results reveal that by incorporating tensor decomposition tech-

Table 6
Testing accuracy results of basic ELM, ELM-TUCKER and ELM-PARAFAC-Satellite dataset.

Activation	Basic ELM			ELM-TUCKER			ELM-PARAFAC		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Hardlims	0.9825	0.96	0.94	0.9932	0.99	0.99	0.9982	0.99	0.98
Tribas	0.9943	0.98	0.97	0.9976	0.99	0.98	0.9987	0.98	0.99

Please cite this article in press as: N.K. Nair and Asharaf S., Tensor Decomposition Based Approach for Training Extreme Learning Machines, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.07.002

Table 8
Testing accuracy results of basic ELM, ELM-TUCKER and ELM-PARAFAC-SHUTTLE dataset.

Activation	Basic ELM			ELM-TUCKER			ELM-PARAFAC		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Tribas	0.9568	0.95	0.94	0.9752	0.97	0.98	0.9787	0.98	0.97
Tanh	0.9653	0.95	0.93	0.9812	0.97	0.98	0.9964	0.99	0.99
Hardlims	0.9629	0.95	0.96	0.9676	0.96	0.97	0.9782	0.98	0.97

Table 9
Training time results of basic ELM, ELM-TUCKER and ELM-PARAFAC.

Training time (s) – Letter Recognition dataset			
Activation function	Basic ELM	ELM-TUCKER	ELM-PARAFAC
Tribas	2502.307	2413.627	2210.872
Tanh	2507.960	2401.842	2198.813
Sigmoid	2504.618	2478.125	2173.671
Hardlims	2501.627	2465.783	2178.354

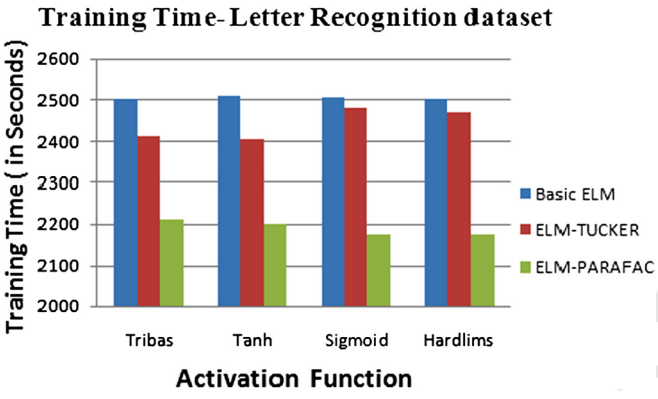

Fig. 13. Comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC.

niques such as PARAFAC and TUCKER decomposition, training time in Extreme Learning machine is reduced than that of basic Extreme Learning Machines.

6.2.5. Performance evaluation on Letter Recognition dataset

Table 9 provides a comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC on the basis of training time (in seconds) using Letter Recognition dataset. Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with different activation functions such as tanh, hardlims, sigmoid and tribas. Results reveal ELM-TUCKER and ELM-PARAFAC have reduced trained time when compared with Basic ELM. The Basic ELM, ELM-TUCKER and ELM-PARAFAC are implemented on Apache Spark 1.6.1.

Training time of Basic ELM versus ELM-TUCKER and ELM-PARAFAC using Letter Recognition dataset is graphically represented in Fig. 13. The ELM-TUCKER and ELM-PARAFAC reveals a reduced training time when compared with that of Basic ELM.

Table 10 shows a comparison of testing accuracy when Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with activation functions such as sigmoid, tanh, hardlims and tribas.

A comparable testing accuracy is achieved when Extreme Learning Machine is treated with tensor decomposition techniques. The

Table 11
Training time results of basic ELM, ELM-TUCKER and ELM-PARAFAC.

Training time (s) – Mushroom dataset			
Activation function	Basic ELM	ELM-TUCKER	ELM-PARAFAC
Tribas	412.883	388.762	385.128
Tanh	414.840	386.351	362.901
Sigmoid	413.797	381.147	376.561
Hardlims	412.699	382.347	378.903

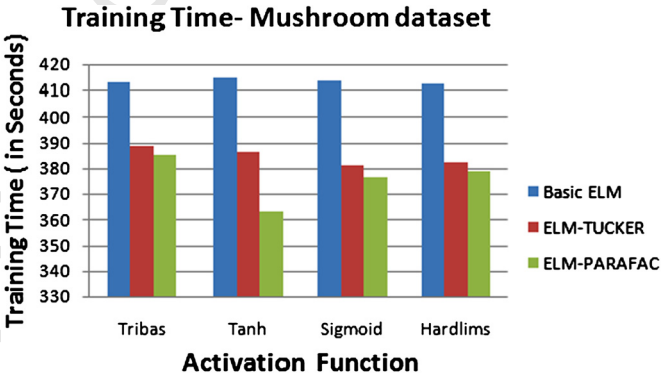

Fig. 14. Comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC.

results reveal that by incorporating tensor decomposition techniques such as PARAFAC and TUCKER decomposition, training time in Extreme Learning machine is reduced than that of basic Extreme Learning Machines.

6.2.6. Performance evaluation on Mushroom dataset

Table 11 provides a comparative analysis of Basic ELM, ELM-TUCKER and ELM-PARAFAC on the basis of training time (in seconds) using Mushroom dataset. Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with different activation functions such as hardlims, tribas, tanh and sigmoid. The results reveal slight variation in training time by ELM-PARAFAC than that of basic ELM in different activation functions since size of the dataset is small. The Basic ELM, ELM-TUCKER and ELM-PARAFAC are implemented on Apache Spark 1.6.1.

Training time of Basic ELM versus ELM-TUCKER and ELM-PARAFAC using Mushroom dataset is graphically represented in Fig. 14. The ELM-TUCKER and ELM-PARAFAC reveals a reduced training time when compared with that of Basic ELM.

Table 12 shows a comparison of testing accuracy when Basic ELM, ELM-TUCKER and ELM-PARAFAC are treated with activation functions such as sigmoid, tanh, hardlims and tribas.

Table 10
Testing accuracy results of basic ELM, ELM-TUCKER and ELM-PARAFAC-LETTER Recognition dataset.

Activation	Basic ELM			ELM-TUCKER			ELM-PARAFAC		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Tribas	0.6857	0.68	0.71	0.7832	0.78	0.80	0.8012	0.80	0.79
Tanh	0.8590	0.85	0.73	0.9023	0.91	0.90	0.9254	0.92	0.91
Sigmoid	0.8830	0.85	0.86	0.9034	0.90	0.92	0.9251	0.92	0.91
Hardlims	0.8102	0.80	0.79	0.8375	0.83	0.83	0.834	0.84	0.83

Please cite this article in press as: N.K. Nair and Asharaf S., Tensor Decomposition Based Approach for Training Extreme Learning Machines, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.07.002

Table 12
TESTING ACCURACY RESULTS OF BASIC ELM, ELM-TUCKER AND ELM-PARAFAC- MUSHROOM DATASET.

Activation	Basic ELM			ELM-TUCKER			ELM-PARAFAC		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
Tribas	0.9621	0.96	0.96	0.9712	0.97	0.98	0.9787	0.97	0.97
Tanh	0.9676	0.97	0.96	0.9812	0.97	0.98	0.9946	0.99	0.98
Sigmoid	0.9619	0.95	0.96	0.9686	0.96	0.97	0.9782	0.98	0.97
Hardlims	0.9629	0.96	0.96	0.9676	0.96	0.97	0.9782	0.98	0.97

A comparable testing accuracy is achieved when Extreme Learning Machine is treated with tensor decomposition techniques. The results reveal that by incorporating tensor decomposition techniques such as PARAFAC and TUCKER decomposition, training time in Extreme Learning machine is reduced than that of basic Extreme Learning Machines. The proposed ELM-TUCKER and ELM-PARAFAC are better than Basic ELM in training time and accuracy. The generalisation performance of ELM-TUCKER and ELM-PARAFAC is achieved better than Basic ELM. The implementation of entire work is done on Apache Spark version 1.6.1 platform which provided an environment for running large dataset.

7. Conclusions and future scope

Extreme Learning Machine is an emerging learning paradigm that offers reduced training time and simplicity in implementation in terms of efficiency. Apart from its peculiarities, Extreme Learning Machines faces scalability issues when enormous datasets needs to be handled. Reducing the training time to few seconds is of great challenge in ELM.

Tensor decomposition techniques are promising tools in the field of big data analytics, as they have the capability to bring multiple modes and aspects of data to a unified framework. These techniques have potential to indulge in applications involving data compression, low-rank approximation, visualization, and feature extraction of high dimensional multi-way data.

Two most popularly used variants of tensor decomposition techniques are TUCKER decomposition and PARAFAC decomposition. Dealing with scalability issues, Tucker decomposition and PARAFAC decomposition expresses better generalisation performance than that of Basic ELM.

Different variants of TUCKER decomposition and CANDECOMP/PARAFAC decomposition can be introduced to handle scalability and dimensionality reduction in Extreme Learning Machines.

Map-reduce framework implementation in 'Tensor Decomposition Based Extreme Learning Machines' can provide a 'Distributed Tensor Decomposition Based Extreme Learning Machine', which can handle tremendous and enormous data explosion problems of standard Extreme Learning Machines.

Conflict of interest

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant finantial support for this work that could have influenced its outcome.

References

[1] Jiexiong Tang, Chenwei Deng, Guang-Bin Huang, Extreme learning machine for multilayer perceptron, IEEE Trans. Neural Netw. Learn. Syst. 27 (4) (2016) 809–821.
[2] Abdul Adeel Mohammed, et al., Human face recognition based on multidimensional PCA and extreme learning machine, Pattern Recognit. 44 (10) (2011) 2588–2597.
[3] Chen Pan, et al., Leukocyte image segmentation by visual attention and extreme learning machine, Neural Comput. Appl. 21 (6) (2012) 1217–1227.
[4] Rashid Minhas, et al., Human action recognition using extreme learning machine based on visual vocabularies, Neurocomputing 73 (10) (2010) 1906–1917.
[5] Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1) (2006) 489–501.
[6] Gao Huang, et al., Trends in extreme learning machines: a review, Neural Netw. 61 (2015) 32–48.
[7] Guang-Bin Huang, Qin-Yu Zhu, Chee-Kheong Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Proceedings of 2004 IEEE International Joint Conference on Neural Networks, vol. 2, IEEE, 2004.
[8] Guang-Bin Huang, et al., Extreme learning machine for regression and multiclass classification, IEEE Trans. Syst. Man Cybern., Part B, Cybern. 42 (2) (2012) 513–529.
[9] Guang-Bin Huang, Haroon A. Babri, Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions, IEEE Trans. Neural Netw. 9 (1) (1998) 224–229.
[10] Shifei Ding, et al., Extreme learning machine: algorithm, theory and applications, Artif. Intell. Rev. 44 (1) (2015) 103–115.
[11] Shifei Ding, Xinzheng Xu, Ru Nie, Extreme learning machine and its applications, Neural Comput. Appl. 25 (3–4) (2014) 549–556.
[12] Tao Dou, Xu Zhou, Fast computation methods for extreme learning machines, in: 2nd International Conference on Intelligent Computing and Cognitive Informatics, ICICCI 2015, 2015.
[13] Vasilios N. Katsikis, Dimitrios Pappas, Fast computing of the Moore–Penrose inverse matrix, Electron. J. Linear Algebra 17 (1) (2008) 637–650.
[14] Adi Ben-Israel, Thomas.N.E. Grenville, Generalized Inverses: Theory and Applications, Springer-Verlag, Berlin, 2002.
[15] Gene H. Golub, Charles F. Van, Matrix Computations, 3rd ed., Johns Hopkins University Press, MD, 1996.
[16] Vasilios N. Katsikis, Dimitrios Pappas, Athanassios Petralias, An improved method for the computation of the Moore–Penrose inverse matrix, Appl. Math. Comput. 217 (23) (2011) 9828–9834.
[17] Ross MacAusland, The Moore–Penrose Inverse and Least Squares, 2014.
[18] Guoxu Zhou, Andrzej Cichocki, Shengli Xie, Decomposition of big tensors with low multilinear rank, preprint, arXiv:1412.1885, 2014.
[19] Andrzej Cichocki, et al., Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-Way Data Analysis and Blind Source Separation, John Wiley & Sons, 2009.
[20] Nicholas D. Sidiropoulos, Rasmus Bro, Georgios B. Giannakis, Parallel factor analysis in sensor array processing, IEEE Trans. Signal Process. 48 (8) (2000) 2377–2388.
[21] Lieven De Lathauwer, Josphine Castaing, Jean-Franois Cardoso, Fourth-order cumulant-based blind identification of underdetermined mixtures, IEEE Trans. Signal Process. 55 (6) (2007) 2965–2973.
[22] Pierre Comon, Tensors: a brief introduction, IEEE Signal Process. Mag. 31 (3) (2014) 44–53.
[23] Peter D. Turney, Empirical evaluation of four tensor decomposition algorithms, preprint, arXiv:0711.2023, 2007.
[24] Berkant Savas, Algorithms in Data Mining Using Matrix and Tensor Methods, 2008.
[25] Inah Jeon, et al., Mining billion-scale tensors: algorithms and discoveries, VLDB J. (2016) 519–544.
[26] Inah Jeon, et al., Haten2: billion-scale tensor decompositions, in: 2015 IEEE 31st International Conference on Data Engineering, IEEE, 2015.
[27] U. Kang, et al., Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2012.
[28] Tamara G. Kolda, Brett W. Bader, Tensor decompositions and applications, SIAM Rev. 51 (3) (2009) 455–500.
[29] Tamara Gibson Kolda, Multilinear Operators for Higher-Order Decompositions, Department of Energy, United States, 2006.
[30] Andrzej Cichocki, et al., Tensor decompositions for signal processing applications: from two-way to multiway component analysis, IEEE Signal Process. Mag. 32 (2) (2015) 145–163.
[31] Lars Grasedyck, Daniel Kressner, Christine Tobler, A literature survey of low-rank tensor approximation techniques, GAMM-Mitt. 36 (1) (2013) 53–78.
[32] Andrzej Cichocki, Era of big data processing: a new approach via tensor networks and tensor decompositions, preprint, arXiv:1403.2048, 2014.

Please cite this article in press as: N.K. Nair and Asharaf S., Tensor Decomposition Based Approach for Training Extreme Learning Machines, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.07.002

[33] Yongxin Yang, Timothy Hospedales, Deep multi-task representation learning: a tensor factorisation approach, preprint, arXiv:1605.06391, 2016.
[34] Berkant Savas, Lars Eldén, Handwritten digit classification using higher order singular value decomposition, Pattern Recognit. 40 (3) (2007) 993–1003.
[35] Guang-Bin Huang, et al., Local receptive fields based extreme learning machine, IEEE Comput. Intell. Mag. 10 (2) (2015) 18–29.
[36] Erik Cambria, et al., Extreme learning machines [trends & controversies], IEEE Intell. Syst. 28 (6) (2013) 30–59.
[37] Guang-Bin Huang, An insight into extreme learning machines: random neurons, random features and kernels, Cogn. Comput. 6 (3) (2014) 376–390.
[38] Ming-Bin Li, et al., Fully complex extreme learning machine, Neurocomputing 68 (2005) 306–314.
[39] Adetunmbi A. Olusola, Adeola S. Oladele, Daramola O. Abosede, Analysis of KDD'99 intrusion detection dataset for selection of relevance features, in: Proceedings of the World Congress on Engineering and Computer Science, vol. 1, 2010.
[40] Harshit Saxena, Vineet Richariya, Intrusion detection in KDD99 dataset using SVM-PSO and feature reduction with information gain, Int. J. Comput. Appl. 98 (6) (2014).
[41] Junchang Xin, et al., ELM*: distributed extreme learning machine with MapReduce, World Wide Web 17 (5) (2014) 1189–1204.
[42] Qing He, et al., Parallel extreme learning machine for regression based on MapReduce, Neurocomputing 102 (2013) 52–58.
[43] Jiaoyan Chen, et al., MR-ELM: a MapReduce-based framework for large-scale ELM training in big data era, Neural Comput. Appl. 27 (1) (2016) 101–110.
[44] Tiantian Liu, et al., Parallelization of a series of extreme learning machine algorithms based on spark, in: 2016 IEEE/ACIS 15th International Conference on Computer and Information Science, ICIS, IEEE, 2016.

Please cite this article in press as: N.K. Nair and Asharaf S., Tensor Decomposition Based Approach for Training Extreme Learning Machines, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.07.002