Int J Syst Assur Eng Manag (June 2022) 13(Suppl. 2):S925–S940
https://doi.org/10.1007/s13198-021-01394-3

ORIGINAL ARTICLE

Parallel training models of deep belief network using MapReduce for the classifications of emotions

Gaurav Agarwal[1,2] · **Hari Om**[1]

Received: 21 June 2021 / Revised: 3 September 2021 / Accepted: 9 September 2021 / Published online: 31 October 2021

Abstract In this paper we present two parallel models for the training of the deep belief networks (DBNs) based on the map-reduce framework. In both models we used more than one computer for the training of DBNs in layer by layer manner following the positive and the negative phase. It is well know that training of DBNs requires large amount of time so with the help proposed models computation can be performed in lesser amount of time with respect to the standalone DBN. By performing experiments on Ryerson Audio-Visual Database of Emotional Speech and Song and Toronto Emotional Speech Set data set it has been observed that the first proposed model i.e. First parallel map-reduced based deep belief network (FParMRBDBN) is shown significantly improvement in the computation time. While the second proposed model i.e. second parallel map-reduced based deep belief network (FParMRBDBN) is shown significantly accelerates the training speed of DBNs. Moreover, both proposed models have given significant results in class classification of the emotions as well.

Keywords Deep belief network · MapReduce · Hidden layer · Restricted Boltzmann machine · Neurons

✉ Gaurav Agarwal
gaurav13shaurya@gmail.com

Hari Om
hariom4india@gmail.com

[1] Department of Computer Science & Engineering, The Indian Institute of Technology (Indian School of Mines), Police Line Road, Main Campus IIT (ISM, Hirapur, Sardar Patel Nagar), Dhanbad, Jharkhand 826004, India

[2] Department of Computer Science and Engineering, Raj Kumar Goel Institute of Technology, Ghaziabad, India

1 Introduction

In recent years, big data has gotten a force from industry and as well as from the scholarly community. Numerous associations are consistently gathering giant datasets from different sources like, the World Wide Web (WWW), social networking websites, and sensor systems. In Big data, a new world of opportunities. (2012), big data is presented as a term that incorporates the utilization of various strategies like to catch, process, investigate, and visualize the large datasets in a measurable time period (Zikopoulos et al. 2012). Now a day's big data is characterized with 7 V's:

1. Volume: Volume refers to the size of the data.
2. Variability: Data for which the meaning is changing constantly.
3. Visualization: The data in a manner that's readable and accessible.
4. Veracity: The trustworthiness of the data in terms of accuracy.
5. Variety: The different types of the data.
6. Velocity: The speed of data generation.
7. Value: Just having of big data is of no use unless we can turn it into value.

Neural networks (NNs), enlivened by the structure of an organic mind, have been effectively applied in many fields, for example, natural language processing (NLP) and pattern recognition (PR) (Dahl et al. 2012; Hinton et al. 2012; Wei et al. 2017; Ouyang et al. 2017; Ren et al. 2017). It has been demonstrated that by expanding the size of the network models with respect to the quantity of network parameters, the quantity of the training instances or samples and the profundity of network models i.e. the quantity

of the hidden layers in the NN, classification precision can be improved significantly (Dan et al. 2010; Le et al. 2011). Both hypothetical researchers and viable applications considered the expanding of profundity of NNs is the most beneficial among these available strategies (Bengio 2009; Simonyan and Zisserman 2014; Szegedy et al. 2015; He et al. 2016). The most well-known one of the artificial neural network (ANN) is Back-propagation neural network (BPNN), capable of assuming any constant nonlinear function with an enough number of neurons which works on arbitrary precision (Hagan et al. 1996). With the giant datasets, back propagation method is generally used for training BPNN and because of the large dataset it requires a large amount of time to compute (Gu et al. 2013). So, to utilize the NN to its full extent one has to speed up the computation process, for this parallel computing is one of the good options. For parallelization a scalable parallel ANN has been proposed by Long and Gupta by using message passing interface (MPI) (Kumar et al. 2002; Message Passing Interface 2015; Long and Gupta 2008). It is also important that MPI was intended for information concentrated applications with high performance prerequisites. MPI offers almost no help in adaptation to internal failure. In the event where a flaw happens, a MPI computation ought to be begun from the starting point. Subsequently, MPI isn't reasonably good for large information applications like big data, which will last for few hours when fault occurs (Liu et al. 2015). At the same time PC limits the size of the bigger network models because of the memory constrained. Now for the training of the NNs at a faster rate, researchers are using the graphics card. Preparing the NN with GPUs, Oh and Jung has reported a speedup factor of 20 (Oh and Jung 2004). Later on, a convolution neural network (CNN) which is faster by four times when compared with the CNNs based on CPU (Chellapilla et al. 2006). This CNN is based on GPUs, and now days GPUs are playing a vital role in machine learning applications. In 2014, 2015 and 2016 Simonyan et al., Szegedy et al., He et al. respectively, developed a model based on large number of GPUs, b which the NNs are becoming deeper and larger. However, hardware has becomes a bottleneck again as the size of models increase. Recently, many researchers are using the concept of cloud computing (CC) to solve the computation problem of the larger NN related to the bottle neck (Huqqani et al. 2014). Without considering the accuracy aspect of the parallel NN Gu et al. proposed a network which was implemented to solve and to speed up the computation of a parallel NN, networks uses memory network processing techniques.

Deep Belief Network (DBN), Long short-term memory (LSTM), Convolution neural network (CNN), auto encoders (AE) are some of the extensively used deep learning models (Zhao et al. 2018). Out of all these CNN shows the best results when it comes to extraction of the features through incrusted and gushed learning and it has been deployed for handing number of problems like music mood classification (Agarwal and Om 2021), segmentation process of the video data(Senger and Mukhopadhyay 2019), face detection as well as disease diagnosis etc. At the same time LSTM is advantageous for Speech Recognition as well as for the weather forecasting. DBN can perform well in number of problems too like interpretation of the facial expression, analysis of the time series, text dependent and non-text dependent learning. Out of all the models of deep learning, DBN is the most widely used model because DBN ensures better results in performance metrics in terms of precision, accuracy, error, specificity. Working of DBNs are almost similar to the working of the human brain, as in human brains process the neurons to have an optimal solution the same is the working of DBN. DBN can be thought as a human brain that can be viewed as a complex structure having neurons in multiple layers formed in stacked format on top of one another (Gong 2021).

However, the process of training of DBN is very time consuming when the DBN encounters the giant datasets. So, in real life applications or experiments the performance of the algorithm in terms of speed is one of the major concerns; thus methods that can accelerate the training of DBN must be investigated. In recent past, large numbers of parallel computing frameworks have been developed to train large NNs. Out of the developed frameworks; SparkNet implements a distributed algorithm by using the framework of Spark based on batch processing to train deep networks (Meng et al. 2016). To compute a particular task using batch-processing framework, dataset is bifurcated into number of batches and finally allotted to individual processor. It simply means, to optimize the purpose, multiple replicas of the same model are trained on different processors by using variable data batches. However, the memory size of an individual system limits the scale of the model because all the available systems using the same model (Guang shi, Jiangshe Zhang 2020). Moreover, this type of scattered processing of the batches is called the parallelism of the data or Data Parallelism. Data parallelism is easy to implement with the use of MapReduce framework generally called as Mapper and Reducer. To bridge the research gap between the sequential and parallel computing as well as training of the DBN models for the large datasets, two models have been proposed in this paper. Using which any researcher can reduce the training time of the Deep Belief Networks in real time by enhancing the processing ability of the giant datasets. In this study two models have been proposed to train the DBN's in lesser time and more accurately with the use of Hadoop ecosystem.

This paper proposed a Hadoop based parallel deep belief networks (HPDBN) for speeding up the computation, so that time requires for the analysis of larger datasets may get reduce. One can use the MapReduce for the distributed processing and it's become a de facto standard computing model with the Mahout for Machine Learning algorithms. Figure 1 represents the Hadoop ecosystem components and its architecture. In this ecosystem, the default layer for the data storage is HDFS i.e. Hadoop Distributed File System. The most important component of Apache Hadoop is HDFS because user can store giant datasets till the user wants. Data will be available in this until the data will be removed by the user itself after the required analysis. By default HDFS creates three (03Nos) replicas of the data, so that the proper distribution of the data across the cluster can be done. Because of this HDFS provides quick and reliable data access. Name Node, Data Node and Secondary Name Node are the three important components of HDFS. Out of all Name Node is considered as the master node so that a track of all the data stored in the cluster can be kept, while slave nodes are the data nodes. MapReduce framework is known as Mapper and Reducer. The input to the map-reduce is in the form of $<$ key, value $>$ pair so that intermediary records can be maintained in the form of $<$ key, value $>$. Total number of outputs from reducer may be same or different from the input provided. Output result from the mapper only generated after the execution of the mappers on the chunks of data available on the different data nodes, each mapper produces the output for the chunk of data it executed. Input file blocks is the key factor to decide number of mappers to be executed by the map-reduce. Intermediate values of the mappers are reduced by the reducer. Basically the overall working of this phase is in three steps shuffle, Sort and Reduce. The final output is in the form of $<$ key, (list of values) $>$ for each pairs in the given input. All the tasks of MapReduce are handled by two daemons known as Task and Job tracker. For easy and efficient analysis of the giant datasets Yahoo developed a tool known as Pig. An optimized high level language Pig Latin is used. Large data sets can be easily handled by pig because its provide parallelism of giant datasets. For querying, analysis and data summarization a similar language to SQL is introduced by Facebook known as HiveQL. For exporting and importing of the data in Hadoop related components like Hbase, HDFS or Hive SQOOP framework is used. It allows data imports, copies data quickly, parallelized data transfer, efficient data analysis and mitigates excessive loads. Sinks, Channel and Sources are the three primary structures of Flume. It gathers the data from origin and reverts to the resting location. In Directed Acyclic Graph one can express the workflow using the Oozie. All the Hadoop jobs like Pig, MapReduce, Scoop or Hive are taken care by Oozie framework. Dependencies like time and data are responsible for the execution of the Oozie's workflow. For having operational services coordination in a Hadoop ecosystem Zookeeper is responsible and it provides reliable, robust, fast access with coordination among all the data chunks available at the data nodes. Naming registry is to be done for the distributed system of this eco system; zookeeper is taken care of this. It also provides distributed and synchronization services to the HDFS. For making Hadoop system enable for the machine learning (ML) Mahout is one of the most important components. With Mahout one can implement various ML algorithms. The rest of the paper is organized as follows. Material and method section focuses on brief introduction of Deep Belief Networks (DBNs), the introduction and formation of our proposed three kinds of parallel deep belief networks (ParDBNs). Experimental results of proposed ParDBNs are discussed in results and discussion section. Finally, we end the paper with conclusions in Sect. 4.

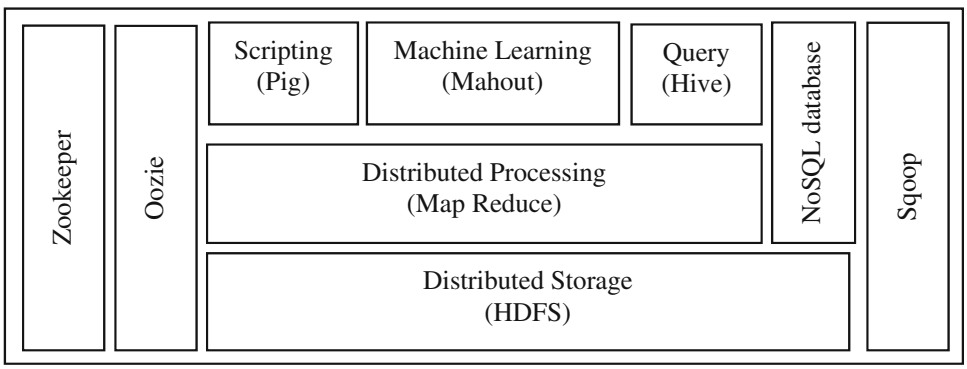

Fig. 1 Hadoop ecosystem components and its architecture (Ashlesha and Tugnayat 2018)

Hadoop ecosystem components and its architecture [29]

2 Materials and methods

This paper attempts to explore the possibility of parallelism in training the DBN and applies them to increase the accuracy, precision and reduce the training time. At the same time effectiveness of the models will also be one of the major concerned. Normally data parallelization and structure are the two important aspects of the parallel computing. Designing of the complicated algorithms can be treated as the structure for the parallel computing and breaking of large datasets in to data subsets to distribution of chunks to the computing nodes are termed as data parallelization. When there is a need of processing the data in parallel manner then the most important model is always a MapReduce programming model. This model is capable to compute the numerous autonomous operations in any order. In parallel processing, the order of operations is not a matter of concern because the equations result does not change from the order of operations, that's why it is also called commutative operations. Commutativity can apply to complex operations and even processes, as long as they don't manipulate the same memory. MapReduce delivers a programming model which abstracts many complexities of parallel processing. The MapReduce implementation performs much of the "wiring" associated with parallel processing, leaving the developer to implement relatively simple methods. The use of MapReduce does come with some constraints, making it less appropriate for some tasks. MapReduce models are optimized for tasks where a large number of key*value input lists must be processed somewhat independently. MapReduce map() method must be commutative, in order for the MapReduce implementation to make use of parallelization. MapReduce enables the parallelization across hundreds and even thousands of CPU's.

Two kinds of ParDBNs have been proposed in this paper and called as First Parallel Deep Belief Network using Map Reduce (FParMRBDBN) and Second Parallel Deep Belief Network using Map Reduce (SParMRBDBN), both are with the diverse communication and synchronization policies. Along with this, the various basic models like DBNs, Restricted Boltzmann Machines (RBMs) are briefly reviewed. In 1986, Hinton and Seinowski proposed the Boltzmann machine, later a modified version of the Boltzmann machine was proposed by Paul Smolensky which was renowned as a Restricted Boltzmann machine.

In RBM, there exist two layers the first layer is known as the visible layer while the second one is called as hidden layer. Visible features of the input data are given as an input to the visible layer and the high level features or unidentified features are represented by the hidden layer (https://medium.com/datadriveninvestor/deep-learning-restricted-boltzmann-machine-b76241af7a92.). As shown in Fig. 2, RBM is represented as an undirected acyclic graph because each and every neuron of the hidden layer is connected to the every neuron of the visible layer i.e. full connection between the two layers with no intra layer links are allowed in RBM. If we have h neurons in the hidden layer and v neurons in the visible layer, then RBM can be expressed as their joint probability distribution. Concept of RBM was initially introduced by Paul Smolensky in 1986 and it gained big popularity in recent years. RBM has set a benchmark for classification, collaborative filtering, topic modeling, regression, feature learning, and dimensionality reduction. For the positive phase, the hidden bias helps the RBM, while during the negative phase the biases of visible layers are helpful to reconstructions of the inputs.

We have denoted neurons in visible and hidden layers as v and h respectively. Neurons in the visible layer are denoted from 1 to m, vi (i = 1, 2,…, m), while in hidden layers are denoted as 1 to n as hj (j = 1, 2,…, n). Here, m and n are simply represents the number of neurons in the visible and hidden layer respectively. It is presuming that every neuron in RBM must satisfy a binary distribution, denoted as $v_i \in [0, 1]$ and $h_j \in [0, 1]$. Moreover, each neuron in RBM is associated with an activation function σ(x), generally sigmoid activation function is selected when the number of classes are more than two. When a neuron is activated then the value of output is always be equal to expression represents the weighted input plus the bias. The input for the upper layer is always the outputs of the neurons in the lower layer. A DBN can be considered as a non-convolutional network which can be formed by merging the several RBMs. The basic structure of DBN has been shown in Fig. 3, in which every two corresponding layers are termed as RBM and denoted substantially as RBM_1 to RBM_n with input layer and n hidden layers. DBN can also be seen as an unsupervised probabilistic based deep learning algorithm. ID of DBN is composed of a multi-layer latent variable, having a random pattern or probability distribution which may not be predicted precisely but may be statistically analyzed statistically. Binary

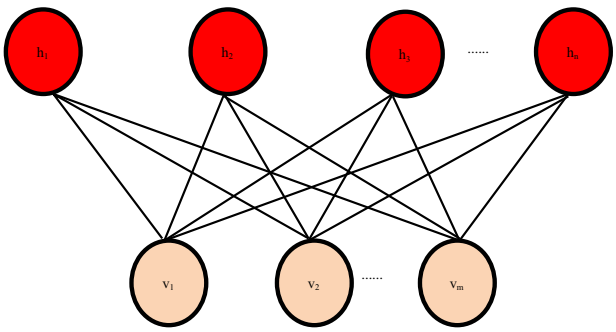

Fig. 2 Visible and hidden layers representation in RBM

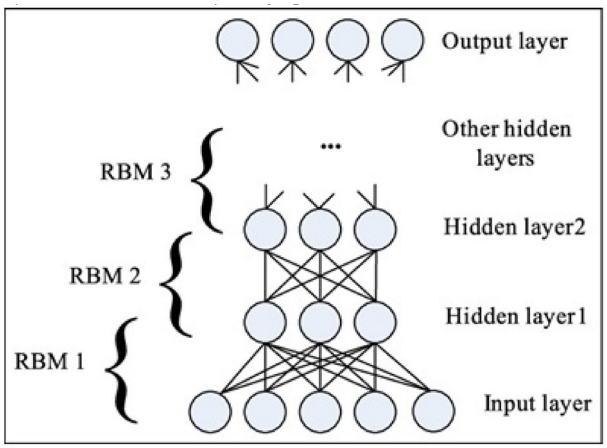

Fig. 3 Architecture of DBN

values are associated with the latent variable, that's why they also known as hidden layer neurons or feature detectors. DBN is a hybrid model, in which the first two layers can be seen as an undirected graph and the rest of layers have directed connections to the top layers like a directed acyclic graph (DAG). Architecture of DBN is shown in Fig. 3.

In DBN, the last or the lowest layer receives the input data and this layer is termed as or the visible layer. This layer accepts either the binary data or the real data. There exists no intra layer connections likes RBM (Mohamed et al. 2009; Bengio et al. 2013). Correlations present in the data are captured by the hidden neuron and represents as features. Any two layers in DBN are connected by the symmetrical weights W stored in a form of matrix. Every neuron in each layer is connected to every neuron in the each neighbouring layer. Greedy learning algorithm is used for the pre training of DBN**.** Dependency of variables of one layer over the other in DBN can be determined by the generative weights, these weights can be learned by using the layer by layer approach in top down fashion in greedy learning algorithm. On the top two hidden layers of DBN several steps of Gibb's sampling are executed. The sample defined by the top two hidden layers of RBM is essentially drawing so that a single pass of the parental sample can be executed through the rest of the model from the visible layer neuron to draw a sample. A single bottom up pass is quite enough to estimate the values of hidden variables in each layer. Input data vector available at the bottom layer is responsible for the pre-training of the DBN and then for the fine tuning it uses the generator weight in the reverse direction.

Geoffrey Hinton proposed the greedy layer wise training algorithm for DBN. In this one layer of DBN is trained at a time in an unsupervised manner. For the simplicity complexity of DBN is divided into small chunks of easily manageable chunks so a multilayer DBN is divided into number of RBMs and these simple models are learned sequentially. Because it is always easy to train a less deep network rather than training a deeper network. To have a different representation of the data this greedy algorithm allows each and every model to traverse in the sequence manner.

For the better understanding let us consider an example as shown in Fig. 4. It has one visible or Input layer and three hidden layers, the last hidden layer is also considered as output layer. The input layer is having three neurons and the first hidden layer is having two neurons while second and third hidden layer is having three and two hidden neurons respectively. by considering this example let us see how the DBN with greedy algorithm can be applied.

Firstly, all the layers of the network are frozen except the first layer. Training data is used to train the first layer greedily. Individual activation probabilities for the neurons of the first hidden layer are computed. In this we parallel updated all the hidden neurons of the first hidden layer. This phase of computation is called as the positive phase and computed as Eq. 1 and 2, where bias associated with hidden neurons are b_1 and b_2, $P(H_{ij} = 1|V)$ represents the output probability of the hidden neurons, b_i is the bias with hidden neuron, $W_{ij}V_i$ is the sum of weight between neuron i in the visible layer and the neuron j in the hidden layer with input and $\sigma(*)$ represents the sigmoid function.

$$P(H_{11} = 1|V) = \sigma(b_1 + W_{11}V_1 + W_{21}V_2 + W_{31}V_3) \quad (1)$$

$$P(H_{12} = 1|V) = \sigma(b_2 + W_{12}V_1 + W_{22}V_2 + W_{32}V_3) \quad (2)$$

In second step, we have to construct the visible neurons by using negative phase which is similar to positive phase by using Eq. 3, 4, 5, here a1, a2 and a3 are the biases associated with the visible neurons, this step is also known as reconstructing of visible neurons from hidden neurons. $P(V_i = 1|H_i)$ represents the output probability of the visible neurons, a_i is the bias with visible neuron, $W_{ij}V_i$ is the sum of weight and input and $\sigma(*)$ represents the sigmoid function.

$$P(V_1 = 1|H_1) = \sigma(a_1 + W_{11}H_{11} + W_{12}H_{12}) \quad (3)$$

$$P(V_2 = 1|H_1) = \sigma(a_2 + W_{21}H_{11} + W_{22}H_{12}) \quad (4)$$

$$P(V_3 = 1|H_1) = \sigma(a_3 + W_{31}H_{11} + W_{32}H_{12}) \quad (5)$$

In last step, one has to update all associated weights in greedy layer wise learning using Eq. 6. In this, the result of the difference of positive and negative phase is multiplied by L i.e. the learning rate and then added to the initial value of the weight.

$$UpdW_{11} = W_{11} + L * (P(H_{11} = 1|V) - P(V_1 = 1|H_1) \quad (6)$$

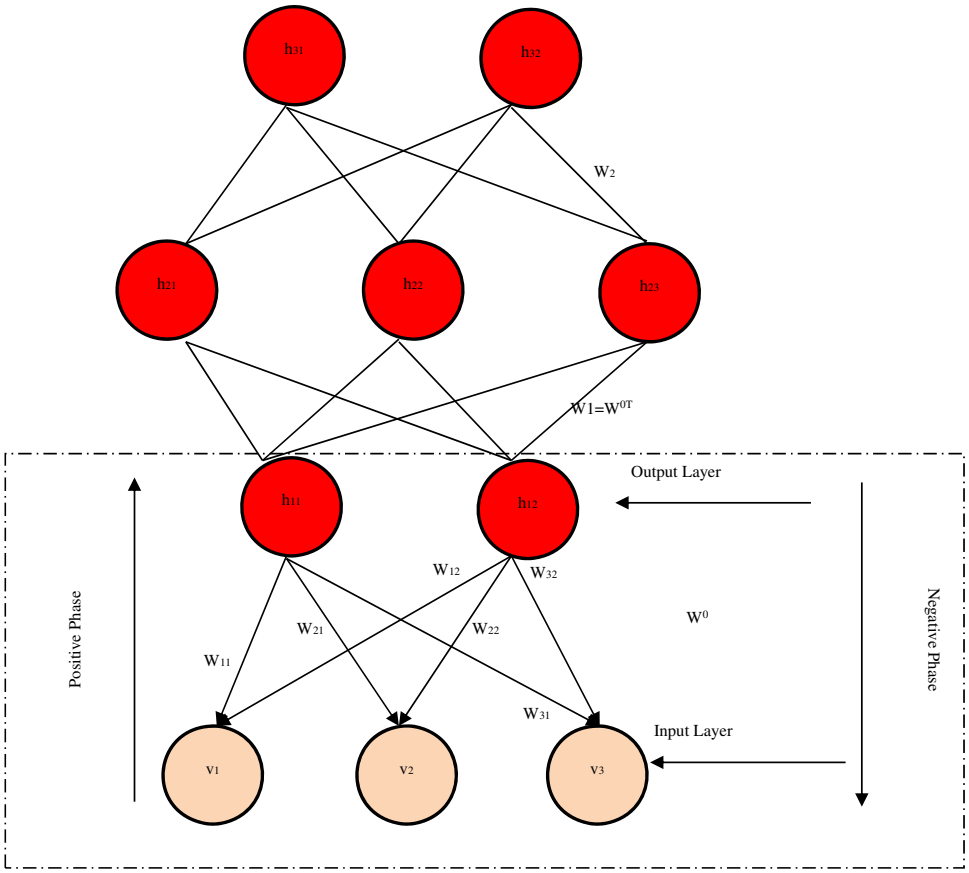

Fig. 4 Example of DBN

Now, for the second hidden layer the first hidden layer will act as an input and so on. Each progressive layer takes yield of the past layer as a contribution to create a yield. Yield created is another portrayal of information where circulation is more straightforward. Weights of the first RBM will be transposed to become the weights for the second RBM. Now repeat the same process for all the RBMs. For training the next RBM, hidden neurons or yield of previous RBM will be the input for the next RBM. We calculate both the phases and update all the weights associated with it. This process will be repeated till we reach the last hidden layer. Here a and b are the bias associated with the visible and hidden layers neurons. For the example repeat the process for the last RBM and calculate the contrastive divergence using the Gibb's sampling.

Positive Phase

$$P(H_{21} = 1|H_1) = \sigma(b_{21} + W_{11}H_{11} + W_{21}H_{12}) \quad (7)$$

$$P(H_{22} = 1|H_1) = \sigma(b_{22} + W_{12}H_{11} + W_{22}H_{12}) \quad (8)$$

$$P(H_{23} = 1|H_1) = \sigma(b_{23} + W_{13}H_{11} + W_{23}H_{12}) \quad (9)$$

Negative Phase

$$P(H_{11} = 1|H_2) = \sigma(a_{11} + W_{11}H_{21} + W_{12}H_{22} + W_{13}H_{23}) \quad (10)$$

$$P(H_{12} = 1|H_2) = \sigma(a_{12} + W_{21}H_{21} + W_{22}H_{22} + W_{23}H_{23}) \quad (11)$$

Now, the positive and negative phases can be represented using the general form of equation. In genric form the Eqs. 1, 2 and 7 to 9 can be written as Eq. 12 and Eq. 3 to 5 and 10, 11 can be written as Eq. 13.

$$P(h_j = 1|V) = \sigma\left(b_j + \sum_{i=1}^{n} v_i w_{ij}\right), \quad j = 1, 2, \ldots, m \quad (12)$$

where w_{ij} indicates the weight between ith visible node and jth hidden node, $\sigma(*)$ represents the sigmoid function, and the bias of jth hidden node is denoted by b_j. Likewise, when a hidden vector $h(h_1, \ldots\ldots, h_j, \ldots\ldots h_m)$ is identified, then the probability activation of the ith visible node is calculated by Eq. (2).

$$P(V_i = 1|h) = \sigma\left(a_i + \sum_{j=1}^{m} h_j w_{ij}\right), \quad i = 1, 2, \ldots\ldots, n \tag{13}$$

where the ith visible node's bias is represented as a_i. Further, with k hidden layers the layer-wise pre-training is employed. Based on the input sample size x, the activation $A_k(x)$ of the kth hidden layer is evaluated as

$$A_k(x) = \sigma(b_k + W_k\sigma(\ldots + W_2\sigma((b_1 + W_1 x)\ldots)) \tag{14}$$

where, W_k and b_k are the weight matrices and the hidden bias vectors of the k^{th} RBM. The DBN uses the deep architecture to get the fair feature representation of the layer-wise pre-training. The Eq. 6 i.e. updation of the weight in generic form can be written as:

$$UpdW_{ij} = W_{ij} + L * \left(Positive(E_{ij}) - Negative(E_{ij})\right) \tag{15}$$

In Eq. 15, $UpdW_{ij}$ is the updated weight, W_{ij} is the weight that is to be updated, L is the learning rate and $Positive(E_{ij}) - Negative(E_{ij})$ is the difference of positive and negative phase.

Greedy layer wise pre training identifies feature detector. Features are slightly modified by fine tuning to get the boundaries right for the specific category. It also helps to discriminate between different classes better associated with the input. Moreover, the accuracy of the model can be improved by adjusting the weights during fine tuning process. And finally, the sigmoid function of Eq. 12 and 13 can be solved by using Eqs. 16 and 17 respectively. Conditional probability of one hidden neuron is given by v through Eq. 16 and conditional probability of one visible neuron can be computed from Eq. 17, given an input vector v and h

$$P(h_i = 1|v) = \frac{1}{1 + e^{\left(-\left(b_j + w_j v_i\right)\right)}} \tag{16}$$

$$P(v_i = 1|h) = \frac{1}{1 + e^{(-(a_i + w_i h_i))}} \tag{17}$$

2.1 Proposed models of ParDBNs

Now days, many researchers are using the network cluster developed with the help of commodity computers for dealing the data intensive applications. In recent past, most popular computing models are Phoenix, Mars, and Hadoop framework (Ashlesha and Tugnayat 2018) of the MapReduce. Out of all the available frameworks because of the open source Hadoop framework is widely accepted by the community. HDFS is there in Hadoop for the management of the data. In Hadoop cluster Name node is considered as a very special node because it holds all the meta data related to the cluster and for running the jobs or for any type of processing cluster is comprises of the number of Data nodes. Data nodes are also responsible for the execution of the Mapper functions (map) and reducer functions. On assigning a job to the cluster, the job is bifurcated into the small data chunks of either 32 MB or 64 MB and finally saved into HDFS. To have data integrity in the cluster, by default each data chunk have three replicas but this may be increase or decrease as per the user requirement. In a Hadoop cluster, rack awareness policy is followed by the mappers to copy and to read the data from the nodes based on data location. And finally reducer generates the final output and stores it back into HDFS.

FParMRBDBN is used for the classification of the datasets where the testing data is in very huge volume. Let us consider a testing sample $test_i = \{l_1, l_2, l_3, \ldots, l_{le}\}$, $test_i \in TEST$, *where*

a. $test_i$, denotes a testing sample.
b. Here the dataset is TEST;
c. le is length of $test_i$; the total number of inputs of a DBN is le;
d. $test_i|target_m|$ is the inputs format of DBN
e. $target_m$, represents the expected output if $test_i$ is a testing sample.
f. "test" and "train," are the two values for the *type* field, if test value is set then type is of $test_i$; and $target_m$ if test is not set.

Hadoop distributed file system (HDFS) initially saves the files that contain samples. Each file comprises of; portion of the testing sample and all the training samples. Therefore, the numbers of mappers required are finalized by the file number n. The file data is the input of FParMRBDBN. Each mapper initializes a DBN as soon as an algorithm starts. As a result, the number of DBNs in the cluster is equal to n. To have architecture neutral model all the DBNs using the same structure with exactly the same value of parameters. Input is read by each mapper is in the form of $test_i|target_m|type$ from HDFS and process the data. $test_i$ is the input to the DBN if type field is having value test. The output of every hidden layer is calculated using Eq. 16, and negative phase is calculated using Eq. 17. The new weights will be computed for every layer and each mapper starts the processing of its DBN for the next layer. All the training samples are processed with the positive and negative phase and the output class is generated.

By running the positive and negative phase each mapper classifies samples labelled as "test". In the proposed model, small portion of the testing dataset is classified by the individual mapper this result in the improved efficiency of the model. The output generated by each mapper is in the form of $instance_i|O_{jn}$, where $instance_i$ represents the

key and O_{jn} is the nth mapper output. Now, all outputs of all the mappers are merged by the reducer and generate the output in the form of $test_i|O_{jn}$ into Hadoop file system known as HDFS. In this O_{jn} is the final class to which sample $test_i$ belong. Figure 5 shows the architecture of FParMRBDBN and Algorithm 1 shows the pseudo code.

$$(mapper_i, DBN_i, train_i) \rightarrow classifier_i \tag{19}$$

With a part of the training dataset each $classifier_i$ is trained to reduce the computation cost. The drawback of this policy is results in the significantly degradation of the classification accuracy of the mapper because to train the DBN only the portion of the training data is used which is a very critical issue. To overcome this issue in SParMRBDBN a number of weak learners are clubbed to creates the strong learners so that the classification accuracy can be maintained.

Proposed pseudo code for FParMRBDBN

Input: TEST, TRAIN.
Output: emotion (O)
Assumptions: m mappers and one reducer

Begin
1: One DBN is constructed by each mapper with le inputs, O outputs and in hidden layer number of neurons is h .
2: Initialize every link w_{ij} from hidden layer to visible or vice versa with a random value.
3: $\forall\ train\ \in TRAIN,\ train_i\ = \{l_1, l_2, l_3, \ldots, l_{le}\}$; Input $l_i \rightarrow le_i$, unit j in hidden layer compute.
(a) Positive phase: update all the neurons of the hidden layer in parallel i.e. compute the positive statistics for edge E_{ij} positive (E_{ij}), which is $P(H_j = 1|V)$.
(b) For the hidden layer neurons, the activation probabilities of the individual neuron can be given by using equation 12 and sigmoid can be calculated using equation 16.
(c) Negative phase: The visible neurons are reconstructed by using the similar technique, i.e. compute the negative statistics for edge E_{ij} Negative (E_{ij}), which is $P(V_i = 1|H)$.
(d) The activation probabilities of the individual neuron of the visible layer can be specified by using equation 13 and sigmoid can be calculated using equation 17.
(e) Update the weight of edge: the updated weight of the edge can be given using equation 15.
4: Repeat step 3 until there is a unit j in hidden layer and training terminates.
5: Divide TEST into $\{t_1, t_2, t_3, \ldots, t_m\}$,$\cup_{i=0}^{m}\ t_i = TEST$
6: Each mapper inputs $test_j = \{l_1, l_2, l_3, \ldots, l_{le}\}, test_j \in t_i$
7: Execute step 3 and 4.
8: Mapper outputs $< test_j, O_j >$
9: Reducer collects and merges all $< test_j, O_j >$
10: Repeat steps 6 to 9 until training is traversed.
11: Reducer outputs emotion (O).
End

Second Parallel Deep Belief Network using Map Reduce (SParMRBDBN) is proposed based on the concept, where training data is more than the testing data i.e. the quantity of the training data is much more than of testing data in DBN. Let us consider a training dataset *TRAIN* with *m* samples. For training by the mapper in SParMRBDBN, the dataset *TRAIN* is to be divided into *n* data samples out of which each data sample $train_i$ is individually processed.

$$TRAIN = \cup_{train=1}^{train=n} train_i, \{\forall\ train\ \in train_i | train\ \notin train_n,\ i \neq n\} \tag{18}$$

In hadoop cluster a single DBN is maintained by single mapper, and for every DBN in $mapper_i$, data sample $train_i$ is work as input for the training data. On the basis of trained parameters each DBN in a mapper produces an output classifier class.

Concept of Bootstrapping: Miscellaneous classifications from one training dataset have been considered simpler than the case of finding a strong learner (https://medium.com/datadriveninvestor/deep-learning-restricted-boltzmann-machine-b76241af7a92.). For this there exist a number of techniques; most commonly used technique is to remodel training datasets by applying the concept of bootstrapping and majority voting. Balanced bootstrap samples can be created by simply constructing a string of samples like $instance_1, instance_2, instance_3, \ldots, instance_n$ and repeat the sequence B times to achieve a sequence of $target_1, target_2, target_3, \ldots, target_{Bn}$. From $target_{p1}, target_{p2}, target_{p3}, \ldots, target_{pn}$ first bootstrap

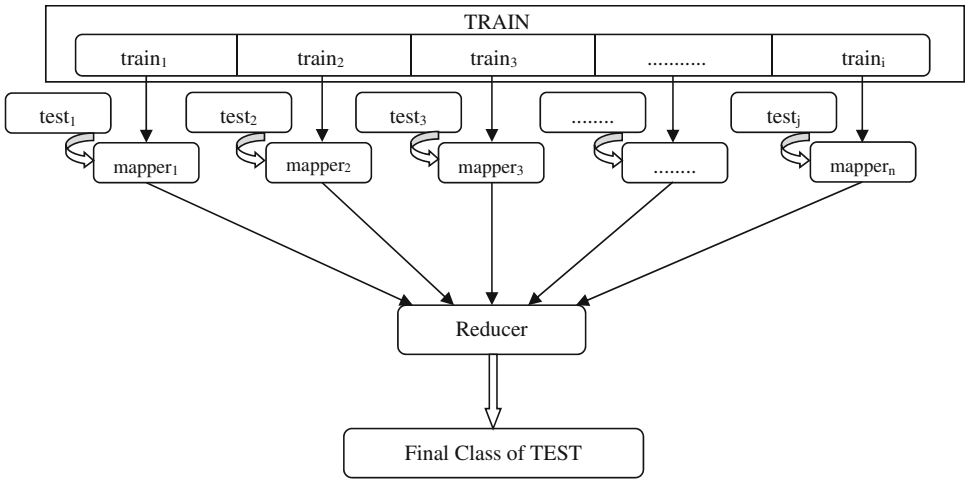

Fig. 5 Architecture of proposed FParMRBDBN

sample has been created and similarly to created second bootstrap sample we have $target_{p(n+1)}, target_{p(n+2)}, \ldots, target_{p(2n)}$ and the process continues until the creation of *Bth* bootstrapping sample by $target_{p((B-1)n+1)}, target_{p((B-1)n+2)}, \ldots, target_{p(Bn)}$.

Majority Voting: It performs classifications based on the maximum votes of the base level classifiers (https://medium.com/datadriveninvestor/deep-learning-restricted-boltzmann-machine-b76241af7a92.). The prediction P_i of the *ith* classifier can be defined as $P_{i,j} \in \{1, 0\}$, $i = 1, 2, \ldots, C_C$ *and* $j = 1, 2, \ldots, c$ where C_C is the number of classes of the classifiers. If the *cth* classifier chooses class *j*, then $P_{i,j} = 1$, *otherwise* $P_{i,j} = 0$. then, the prediction for class *k* is calculated as

$$P_{i,j} = \max_{j=1}^{j=c} \sum_{i=1}^{I} P_{i,j} \tag{20}$$

SParMRBDBN first generates the number of subsets from the complete training data by applying balanced bootstrapping.

$$balanced\ bootstrapping \rightarrow \{train_1, train_2, train_3, \ldots, train_n\}, \bigcup_{i=1}^{n} train_i = TRAIN \tag{21}$$

where $train_i$ represents the *ith* subset from the entire dataset and it directly belongs to the dataset *TRAIN,* the total number of subsets in dataset *TRAIN is n*. Each $train_i$ is saved in HDFS as a single file. Each sample $train_k = \{l_1, l_2, l_3, \ldots, l_{le}\}$, $train_k \in TRAIN_i$, is defined in the format of $instance_i|target_k|type$, where.

1. One bootstrapped sample $train_k$ is represented by $instance_k$, which works as an input of DBN.
2. The total number of inputs of the DBN are *le*.
3. The desirable output will be $target_k$, if $instance_k$ is a training sample.
4. "test" and "train," are the two values for the *type* field, which is equal to the type of $instance_k$; if "test" value is set and $target_k$ field should be left empty.

In SParMRBDBN one DBN is constructed by one mapper and all the associated weights and biases are initialized by the random values lie between -1 to 1 for all the neurons. Now the mapper is having one record as an input and in the form of $instance_i|target_k|type$. Firstly, the mappers retrieves the type of the sample by parse the input data in such a manner that if train is the value of type then the sample is directly feed as an input into the visible layer i.e. the input layer. The output of the every hidden layer is calculated using Eq. 16, and negative phase is calculated using Eq. 17. The new weights will be computed for every layer and then the DBN in every mapper begins the dissemination process for the next layer. Repeat the positive and negative phase until all the training samples are processed and the output class are generated.

In this, the classification class of the sample is computed by every mapper at the last hidden layer i.e. at the output layer. The intermediate output of the each mapper is in the form of key and output of the nth mapper i.e. $instance_i|O_{jn}$. Finally, all the mappers outputs are collected by the reducer and all the outputs of the same key have been merged together. Now, the reducer runs majority voting using Eq. 20 and outputs the result of $instance_i$ into HDFS in the form of $instance_k|r_k$, *where* r_k represents the voted classification result of $instance_k$. Figure 6 represents the model of SParMRBDBN and Algorithm 2 describes the pseudo code.

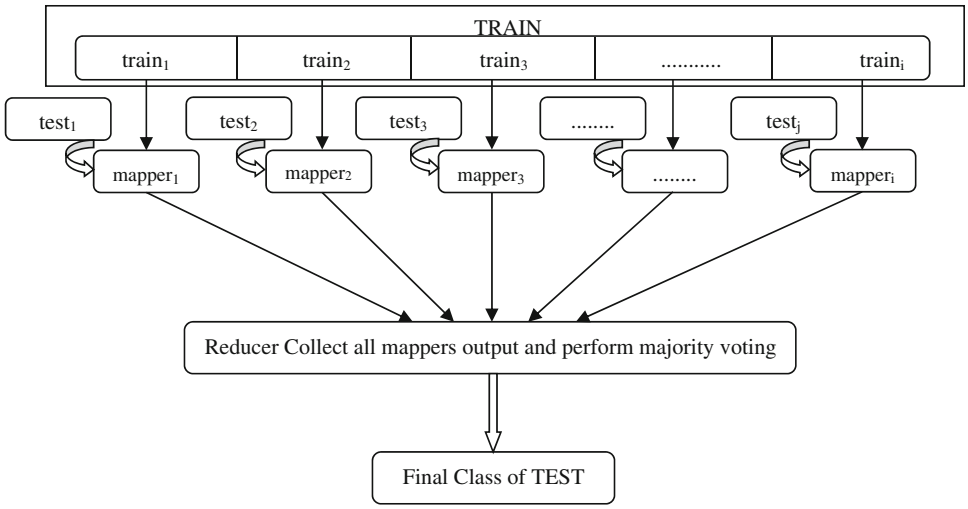

Fig. 6 Architecture of proposed SParMRBDBN

Proposed pseudo code for SParMRBDBN
Input: TEST, TRAIN. **Output:** emotion (O) **Assumptions:** m mappers and one reducer
Begin 1: One DBN is constructed by each mapper with le inputs, O outputs and in hidden layer number of neurons is h . 2: Initialize every link w_{ij} from hidden layer to visible or vice versa with a random value. 3: perform bootstrapping using equation 19 4: $\forall\ train\ \in TRAIN,\ train_i = \{l_1, l_2, l_3, \dots, l_{le}\}$; Input $l_i \rightarrow le_i$, unit j in hidden layer compute. (a) Positive phase: update all the neurons of the hidden layer in parallel i.e. compute the positive statistics for edge E_{ij} positive (E_{ij}), which is $P(H_j = 1\|V)$. (b) For the hidden layer neurons, the activation probabilities of the individual neuron can be given by using equation 12 and sigmoid can be calculated using equation 16. (c) Negative phase: The visible neurons are reconstructed by using the similar technique, i.e. compute the negative statistics for edge E_{ij} Negative (E_{ij}), which is $P(V_i = 1\|H)$. (d) The activation probabilities of the individual neuron of the visible layer can be specified by using equation 13 and sigmoid can be calculated using equation 17. (e) Update the weight of edge: the updated weight of the edge can be given using equation 15. 5: Repeat step 4 until there is a unit j in hidden layer and training terminates. 6: Divide TEST into $\{t_1, t_2, t_3, \dots, t_m\}$,$\cup_{i=0}^{m}\ t_i = TEST$ 7: Each mapper inputs $test_j = \{l_1, l_2, l_3, \dots, l_{le}\}, test_j \in t_i$ 8: Execute step 4 and 5. 9: Mapper outputs $< test_j, O_j >$ 10: Reducer collects and merges all $< test_j, O_{jm} >, m = (1,2,3, \dots n)$ 11: for each $test_j$ compute equation 20. 12: Repeat steps 7 to 11 until training is traversed. 13: Reducer outputs emotion (O). **End**

3 Results and discussions

In this paper, two parallel DBNs using Hadoop cluster have been proposed. To analyse the performance of the proposed models a Hadoop cluster was built, this cluster is comprising of twenty-five (25 Nos) PC out of which twenty-three (23 Nos) PCs act as the Data nodes, one (1No) PC is the secondary name node and one (1 No) PC is the Name node. The complete cluster details is shown in Table 1

Two datasets RAVDESS and TESS has been used in this study. The datasets are measured on the basis of total sample numbers, sample length, element range and class number. Sample number represents the total no of samples in a particular dataset, Sample length represents the range

Table 1 Cluster Details

Sr. No	Machine type	Quantity	Configuration	Workload pattern /cluster type
1	Name node	01	OS: Ubuntu 14.04LTS, Disk: 250 GB SSD, Memory: 32 GB RAM, CPU: Core i5, 5th Gen 2.3Ghz Quad Core	Balanced workload
2	Secondary name node	01	OS: Ubuntu 14.04LTS, Disk: 250 GB SSD, Memory: 32 GB RAM, CPU: Core i5, 5th Gen 2.3 Ghz Quad Core	Balanced workload
3	Data node	23	OS: Ubuntu 14.04LTS, Disk: 500 GB SATA HDD 7200RPM, Memory: 4 GB, CPU: Corei3, 5th Gen 2.0 Ghz Quad core	Cluster type
4	Root	01	20 GB	Acts as storage location in server
5	Swap memory	01	2 × Memory	Acts as secondary memory, when actual RAM gets occupied
6	Bandwidth	–	200Mbps to 1 Ghz	–
7	Switch	02	TP-Link TL –SF1016D 16-Port 10/100 Mbps	–

Table 2 Dataset Details

Sr. no	Data set	Sample number	Sample length	Element range	Class number
1	TESS	2800	1–2	(0.6)	7
2	RAVDESS	1440	3–4	(0.7)	7

duration of the input sample, and element range represents the class number to which the input sample belongs while class number represents the total number of available classes in the dataset.

Dataset Description: Total of two English datasets has been used in this paper. Descriptions of both the datasets, Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS) and Toronto Emotional Speech Set (TESS) are as shown in Table 2.

Toronto Emotional Speech Set (TESS): TESS dataset was modelled on the North-western University Auditory Test No. 6. From Toronto, two actresses aged 64 and 26 years were recruited. By the two actresses, a set of 200 target words were spoken in the carrier phrase and the recordings were made of the set describing the seven emotions (https://www.kaggle.com/ejlok1/toronto-emotional-speech-set-tess) (Agarwal and Om 2020) the basic details has been shown in Table 3.

Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): The RAVDESS is an approved multimodal dataset of emotional song and speech. The dataset is gender balanced including professional actors, vocalizing lexically-coordinated statements in a neutral North American accent. Each emotion is generated at two stages of emotional force, with an extra neutral emotion. All the settings are presented in the face, voice and face-voice arrangements (Agarwal and Om 2020; Livingstone and Russo 2018) Description of factor level coding of RAVDESS filenames are given in Table 4

We have implemented a two-layer as well as three-layer DBN with twelve numbers of neurons in the hidden layer. The Hadoop cluster is maintained with twelve numbers of mappers and with one reducer. For calculating the precision P of the proposed models total number of samples varies from 20 to 600 and Eq. 22 has been used. Similarly, the computation efficiency of the proposed models has been computed on the dataset size vary from 2 MB to 1 GB. Each model was executed eight times and the average of all has considered as final result.

$$P = \frac{CR}{CR + WR} \times 100\% \tag{22}$$

where CR are the correctly recognized samples and WR Specifies wrongly recognized samples.

Here in the proposed models the basic use of DBN is to find out the input neurons. The number of the output neurons will be as per our datasets either 07 or 08 for TESS or RAVDESS dataset respectively. The activation function used here is sigmoid based on data that is differentiable and hence continuous. One has to decide the number of neurons in each and every hidden layer with number of epochs and iterations in each epoch.

The precision of FParMRBDBN has been done using the variable number of samples from the training dataset. The maximum number of the testing as well as the training samples is six hundred only (600 Nos). There is data duplication in the large number of samples. Twelve mappers have been used for the precision results of

Table 3 Description of TESS Dataset

Sr. no	Identifier	Coding description of factors level
1	Modality	01 = Audio Only
2	Channel	01 = Words
3	Emotion	01 = Neutral, 02 = surprise, 03 = Happy, 04 = Angry, 05 = Fear, 06 = Disgust, 07 = Sad
4	Intensity	01 = Normal, 02 = Strong
5	Words	01 = First Word, …., 200 = Two Hundredth Word
6	Repetition	01 = None
7	Actor	01 = Young Female Actor, 02 = Old Female Actor

Table 4 Description of RAVDESS Dataset

Sr. no	Identifier	Coding description of factors level
1	Modality	01 = Audio–Video, 02 = Video Only, 03 = Audio Only
2	Channel	01 = Speech, 02 = Song
3	Emotion	01 = Neutral, 02 = Calm, 03 = Happy, 04 = Sad, 05 = Angry, 06 = Fearful, 07 = Disgust, 08 = Surprised
4	Intensity	01 = Normal, 02 = Strong
5	Statement	01 = “Kids are talking by the door”, 02 = “Dogs are sitting by the door”
6	Repetition	01 = First repetition, 02 = Second repetition
7	Actor	01 = First Actor, 02 = Second Actor, 03 = Third Actor, 04 = Fourth Actor, …., 24 = Twenty Fourth Actor

FParMRBDBN. It has been observed that with the increase of the training samples the precision accuracy go on increasing and it reaches 100% precision for the TESS and 97.53 for the RAVDESS. During the experiment it has also been noticed that FParMRBDBN behaves quite similarly to that of the standalone DBN, as FParMRBDBN runs the hadoop to distribute the data instead of distribution of the DBN among the Hadoop nodes. Similarly for the evaluation of SParMRBDBN, six hundred testing samples and six hundred testing samples have been deployed. The training sample subsets have been used for the training of the mappers and based on majority voting with bootstrapping it gives the classification class for all the six hundred testing samples. Each mapper of SParMRBDBN uses training samples varying from twenty to six hundred as an input. It has been observed that with the increase of the training samples the precision based on majority voting goes on increasing. Again in SParMRBDBN TESS reached to 100% while RAVDESS reaches to 98.97%, which is a bit higher with respect to FParMRBDBN. The overall Precision percentage (%) of proposed FParMRBDBN and SParMRBDBN on TESS dataset and on RAVDESS dataset has been compared and reflected in Figs. 7 and 8 respectively.

The clear observation is, because of majority voting proposed SParMRBDBN performs better than proposed FParMRBDBN and also the precision of SParMRBDBN is more stable than that of FParMRBDBN for the classification purpose. The stability of both the proposed models has been shown in Fig. 9. The TESS dataset shows that SParMRBDBN is more stable when compared with FParMRBDBN based on precision.

For the computation efficiency experiments has been carried out using the TESS and RAVDESS datasets. Experiments are performed using twelve mappers to analyse the efficiency of FParMRBDBNand of SParMRBDBN. For this data size varies from 1 to 900 MB. FParMRBDBN simply outperforms the standalone DBN and same has been presented in Fig. 10. Another important observation is about the computation overhead, it has been clearly observed that standalone DBN overhead is too low in comparison with the proposed model till the dataset size is below 23 MB and the overhead increases exponentially with the data size more than 23 MB. The operating cost of the proposed FParMRBDBN model is low because the testing data is to be distributed in the Hadoop cluster among twenty three data nodes, and all the twenty three data nodes must run in parallel for the class classification. And for the SParMRBDBN, once the data size goes beyond 54 MB it starts increasing rapidly. Performance of the proposed parallel SParMRBDBN is far better than that of standalone DBN but a bit lower than the proposed FParMRBDBN.

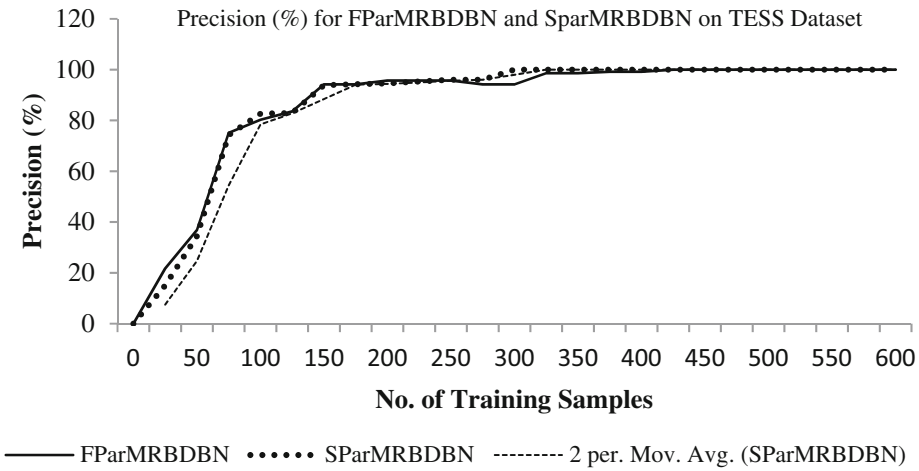

Fig. 7 Precision percentage (%) Comparison of proposed FParMRBDBN and SParMRBDBN on TESS dataset

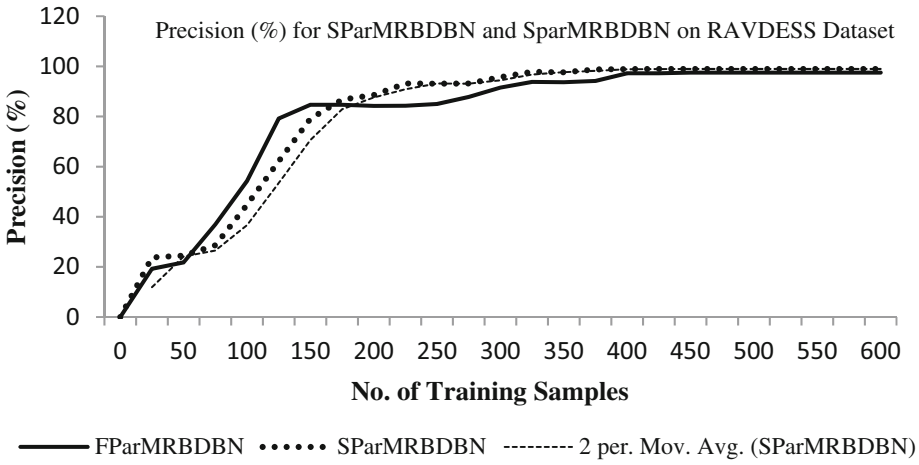

Fig. 8 Precision percentage (%) comparison of proposed FParMRBDBN and SParMRBDBN on RAVDESS dataset

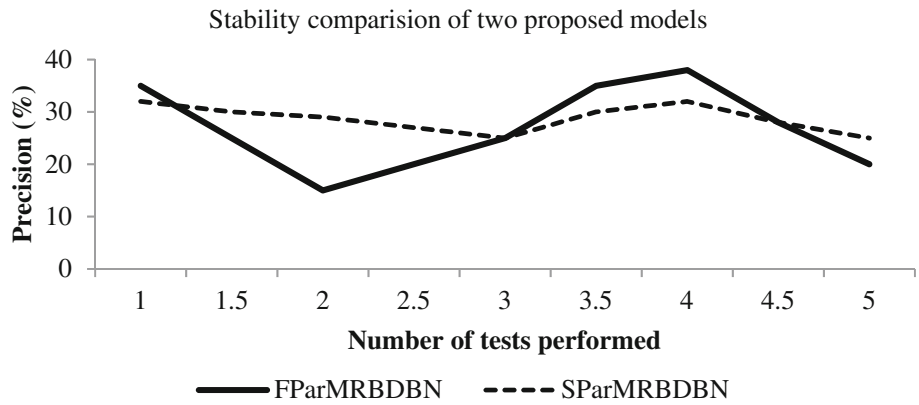

Fig. 9 stability comparison of the two proposed models based on precision (%)

For the time complexity analysis of the both proposed models with general DBN, let us consider t_{DBN} is the required training time for each and every DBN, t_{ftt} is the time that is required for the fine tuning of the complete network, and t_{cc} is the time judgement time for the classifier committee, on the basis of these assumptions the FparMRBDBN and SparMRBDBN is having the training time as follows:

$$T_{DBNtr} = t_{DBN} + l \times t_{ftt} + t_{cc} \tag{23}$$

where l represents the total number of classifiers in the proposed model at the same time following the normal boosting the training time changes to as shown in Eq. 24.

$$T_{DBNbo} = l \times t_{DBN} + l \times t_{ftt} + t_{cc} \tag{24}$$

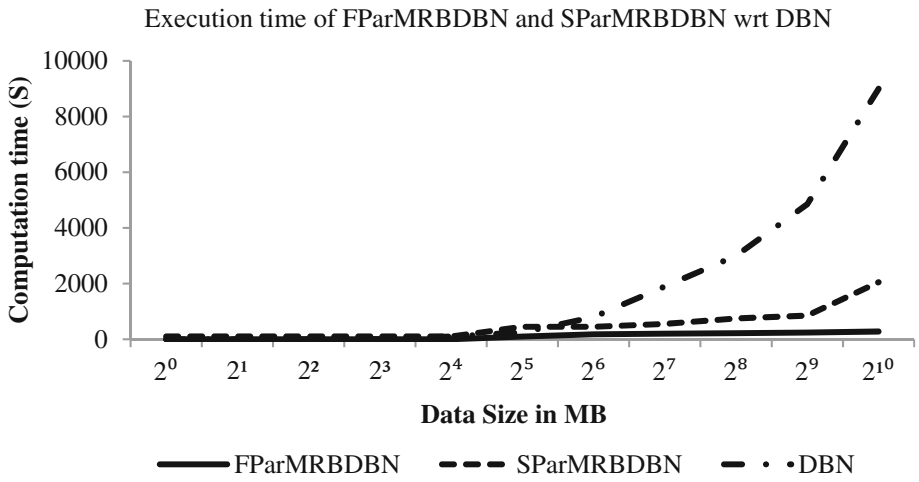

Fig. 10 Execution time of FParMRBDBN and SParMRBDBN wrt DBN

DBN can be trained by the two well defined phases that are pre-training and fine-tuning. To capture the features of the input training sample, pre-training phase is to give a DBN with the appropriate weights and biases while to adjust the weights and bias accurately the fine-tuning phase is using the error backpropagation (BP) algorithm based on the weights and biases obtained from the pre-training phase. Table 5 represents the comparison of training and testing error rate with respect to number of hidden layers used and also the iterations performed for pre training as well as for the fine tuning so here we compare the performance of proposed DBNs with DBN. Training has been performed in batches containing hundred data samples to a total of six hundread data samples. For the fine tuning in this paper conjugate gradient descent has been used for both two hidden layers and three hidden layers. Table 5 clearly indicates that proposed SparMRBDBN outperforms both DBN and FparMRBDBN on each and every iteration of the experiments. For the comparision 0.1 learning rate with 0.1 initialization of weights has been used. However, by training the network several times and observing its performance on the both TESS and RAVDESS dataset has been validated.

For assuring the proper working of the proposed models both available datasets i.e. TESS and RAVDESS has been bifurcated into two equal parts i.e. 50% training and 50% testing. Then the proposed models have been executed for the formation of Table 5. From Table 5, it is very clear that the performance of SparMRBDBN is best among the three. We started from two hidden layers and goes up to three. Number of iterations used for pre training and fine tuning are 6, 10, 20 and 50 for both hidden layers. We have tested the proposed models and compared with DBN on the basis of training error rate and test error rate. It has been observed that as the number of iterations goes on increasing from 2 to 50 either for hidden layers will be two or three the training and testing error goes on decreasing. With two hidden layers and six iterations training and testing rate are 2.58, 2.64 and 1.81 and 3.66, 1.86 and 1.73 for DBN, FparMRBDBN and SparMRBDBN respectively. When fifty iterations with two hidden layers taking into consideration the training error rate is zero and test error

Table 5 Comparison of training and testing error rate based on number of hidden layers and iterations

Sr. No	Hidden Layers	Iterations		Training error rate (%)			Test error rate (%)		
		Pre training	Fine tuning	DBN	FparMRBDBN	SparMRBDBN	DBN	FparMRBDBN	SparMRBDBN
1	2	6	6	2.58	2.64	1.81	3.66	1.86	1.73
2	2	10	10	0.62	0.51	0.12	2.10	1.64	0.82
3	2	20	20	0.02	0.00	0.00	1.78	0.54	0.24
4	2	50	50	0.00	0.00	0.00	1.68	0.18	0.21
5	3	6	6	3.44	2.59	0.96	3.99	1.83	1.46
6	3	10	10	0.91	1.03	0.27	2.40	1.76	1.41
7	3	20	20	0.09	0.02	0.00	1.82	1.21	1.28
8	3	50	50	0.00	0.00	0.00	1.71	0.47	1.14

rate is 1.68, 0.18 and 0.21 for DBN, FparMRBDBN and SparMRBDBN respectively. Similarly, when we considered three hidden layers and six iterations training and testing rate are 3.44, 2.59 and 0.96 and 3.99, 1.83 and 1.46 for DBN, FparMRBDBN and SparMRBDBN respectively. When fifty iterations have been considered with three hidden layers the training error rate is zero while test error rate is 1.71, 0.47 and 1.14 for DBN, FparMRBDBN and SparMRBDBN respectively. Clearly, the complete discussion shows that the proposed models outperform the DBN on the aspect of training and testing error rate.

4 Conclusion and future scope

In this paper, two parallel DBNs models have been proposed called as FParMRBDBNand SParMRBDBN based on the computing model of MapReduce. To deal with the giant size of the datasets DFS file structure has been utilized with a complete cluster size of twenty five nodes. Paper concludes that overhead of the computation can be reduced extremely if number of nodes can be used in parallel manner either in FParMRBDBN or SParMRBDBN. Because of the concept of majority voting as well as bootstrapping SParMRBDBN is more feasible and more considerable as a fully distributed DBN in a cluster based programming environment but it also incurs overhead because of regular start and stop of mappers and reducer in hadoop framework. The unique strength of SparMRBDBN is maintaining almost the same time complexity as of DBN. Top layers of the proposed model utilizes different weighted data to focus on more efficient class classification, while the lower layers of the belief networks share weights for feature extraction. The result shows that the proposed methods are computationally efficient and can be readily used for practical applications. In near future the proposed models can be enhanced for the different chunks of the data blocks used by the Hadoop ecosystem. Models can be modified for the use of 32/64 MB of the data blocks so that the training time depending upon the block size can be optimized.

Funding Not Applicable.

Data availability The data that support the findings of this study are available with the corresponding author, upon reasonable request.

Declarations

Conflicts of interest Gaurav Agarwal and Hari OM declare that they have no conflict of interest.

Consent to participate Due acknowledgement/citation have been given to all.

Code availability The code that support the findings of this study are available with the corresponding author, upon reasonable request.

Ethics approval This chapter does not contain any studies with human participants/animals performed by any of the authors.

References

Agarwal G, Om H (2020) Performance of deer hunting optimization based deep learning algorithm for speech emotion recognition. Int J Multimed Tools Appl 2020:1

Agarwal G, Om H (2021) An efficient supervised framework for music mood recognition using autoencoder-based optimised support vector regression model. IET Signal Proc. https://doi.org/10.1049/sil2.12015

Ashlesha S, Tugnayat RM (2018) A review of Hadoop Ecosystem for Bigdata. Int J Comput Appl 180(14):1

Bengio Y (2009) Learning deep architectures for AI. Found Trends Mach Learn 2(1):1–127

Bengio Y, Courville A, Vincent P (2013) Representation learning: a review and new perspectives. IEEE Trans Pattern Anal Mach Intell 35(8):1798–1828

Chellapilla K, Puri S, Simard P (2006) High performance convolution neural networks for document processing. In: 10th international workshop on frontiers in handwriting recognition, Suvisoft

Dahl GE, Yu D, Deng L, Acero A (2012) Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Trans Audio Speech Lang Process 20(1):30–42

Dan CC, Meier U, Gambardella LM, Schmidhuber J (2010) Deep big simple neural nets excel on handwritten digit recognition. Corr 22(12):3207–3220

Gong T (2021) Deep belief network-based multifeature fusion music classification algorithm and simulation. Complexity 2021, Article ID 8861896, 2021. https://doi.org/10.1155/2021/8861896

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: The IEEE conference on computer vision and pattern recognition (CVPR)

Hinton G, Deng L, Yu D, Dahl GE, Mohamed A, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Saiainath TN (2012) Deep neural networks for acoustic modeling in speech recognition. IEEE Signal Process Mag 29(6):82–97

Huqqani AA, Schikuta E, Mann E (2014) Parallelized neural networks as a service. In: Proceedings of the international joint conference on neural networks (IJCNN '14), pp 2282–2289

Le QV, Ngiam J, Coates A, Lahiri A, Prochnow B, Ng AY (2011) On optimization methods for deep learning. In: International conference on machine learning, pp 67–05

Livingstone SR, Russo FA (2018) The ryerson audio-visual d/b of emotional speech and song (RAVDESS): a dynamic, multimodal set of facial and vocal expressions in North American English. PLoS ONE 13(5):1

Long LN, Gupta A (2008) Scalable massively parallel artificial neural networks. J Aerosp Comput Inf Commun 5(1):3–15

M.H. Hagan, H.B. Demuth, M.H. Beale (1996) Neural Network Design, PWS Publishing

Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, FreemanJ TsaiD, AmdeM OwenS, XinD XinR, Franklin MJ, Zadeh R, Zaharia M, Talwalkar A (2016) MLlib: machine learning in Apache Spark. J Mach Learn Res 17(1):1235–1241

Message Passing Interface (2015) http://www.mcs.anl.gov/research/projects/mpi/

Mohamed A, Dahl G, Hinton G (2009) Deep belief networks for phone recognition. In: Nips workshop on deep learning for

speech recognition and related applications, Vancouver, Canada, vol 1, p 39

Networked European Software and Services Initiative (NESSI) (2012) Big data, a new world of opportunities. Networked European Software and Services Initiative (NESSI) White Paper, 2012, http://www.nessi-europe.com/Files/Private/NESSI White-Paper BigData.pdf

Oh KS, Jung K (2004) GPU implementation of neural networks. Pattern Recogn 37(6):1311–1314

Ouyang W, Zeng X, Wang X, Qiu S, Luo P, Tian Y, Li H, Yang S, Wang Z, Li H, Wang K, Yan J, Loy CC, Tang X (2017) DeepID-Net: object detection with deformable part based convolution neural networks. IEEE Trans Pattern Anal Mach Intell 39(7):1320–1334

R. Gu, F. Shen, Y. Huang (2013) Aparallel computing platform for training large scale neural networks. In: Proceedings of the IEEE International Conference on Big Data, pp 376–384

Ren S, He K, Girshick R, Sun J (2017) Faster R-CNN: towards real time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 39(6):1137–1149

Senger SS, Mukhopadhyay S (2019) Moving object detection using statistical background subtraction in wavelet compressed domain. In: Multimedia tools and applications. https://doi.org/10.1007/s11042-019-08506-z

Shi G, Zhang J, Zhand C, Hu J (2020) A distributed parallel training method of deep belief networks. Soft Comput. https://doi.org/10.1007/s00500-020-04754-6.

Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556

Szegedy C, Liu W, Jia Y, Sermanet P (2015) Going deeper with convolutions. In: IEEE conference on computer vision and pattern recognition, pp 1–9

V. Kumar, A. Grama, A. Gupta, G. Karypis (2002) Introduction to Parallel Computing, Benjamin Cummings/Addison Wesley, San Francisco, Calif, USA

Wei J, He J, Chen K, Zhou Y, Tang Z (2017) Collaborative filtering and deep learning based recommendation system for cold start items. Expert Syst Appl 69:29–39

Y. Liu, J. Yang, Y. Huang, L. Xu, S. Li, M. Qi (2015) Map reduced based parallel neural networks in enabling large scale machine learning. Comput Intell Neuro Sci

Zhao L et al (2018) Parallel computing method of deep belief networks and its application to traffic flow prediction. Knowl-Based Syst. https://doi.org/10.1016/j.knosys.2018.10.025

Zikopoulos PC, Eaton C, deRoos D, Deutsch T, Lapis G (2012) Understanding Big Data. McGraw-Hill, Analytics for Enterprise Class Hadoop and Streaming Data