J. Mod. Power Syst. Clean Energy (2016) 4(3):414–426
DOI 10.1007/s40565-016-0209-4

# High-performance predictor for critical unstable generators based on scalable parallelized neural networks

**Youbo LIU[1], Yang LIU[1], Junyong LIU[1], Maozhen LI[2], Zhibo MA[3], Gareth TAYLOR[2]**

**Abstract** A high-performance predictor for critical unstable generators (CUGs) of power systems is presented in this paper. The predictor is driven by the MapReduce based parallelized neural networks. Specifically, a group of back propagation neural networks (BPNNs), fed by massive response trajectories data, are efficiently organized and concurrently trained in Hadoop to identify dynamic behavior of individual generator. Rather than simply classifying global stability of power systems, the presented approach is able to distinguish unstable generators accurately with a few cycles of synchronized trajectories after fault clearing, enabling more in-depth emergency awareness based on wide-area implementation. In addition, the technique is of rich scalability due to Hadoop framework, which can be deployed in the control centers as a high-performance computing infrastructure for real-time instability alert. Numerical examples are studied using NPCC 48-machines test system and a realistic power system of China.

**Keywords** Transient stability, Critical unstable generator (CUG), High-performance computing (HPC), MapReduce based parallel BPNN, Hadoop

CrossCheck date: 3 June 2016

Received: 30 November 2015 / Accepted: 25 April 2016 / Published online: 14 July 2016


✉ Yang LIU
yang.liu@scu.edu.cn

Youbo LIU
liuyoubo@scu.edu.cn

Junyong LIU
liujy@scu.edu.cn

Maozhen LI
maozhen.li@brunel.ac.uk

Zhibo MA
zhibo.ma@nationalgrid.com

Gareth TAYLOR
Gareth.taylor@brunel.ac.uk

[1] School of Electrical Engineering and Information, Sichuan University, Chengdu 610065, China

[2] Electronic and Computer Engineering, Brunel University, Uxbridge, Middx, London UB8 3PH, UK

[3] Senior Power System Engineer, National Grid, Bearwood road, Wokingham RG6 3DU, UK

## 1 Introduction

Transient stability has been widely regarded as one of the most concerned issues of modern power system. In the last two decades, a number of large blackouts occurred all over the world due to the loss of synchronization caused by cascading failures [1]. Insufficient online implementations and lack of timely emergency controls, such as load shedding, generator tripping and proactive islanding, are said to be the common causes of those accidents [2]. The increasing renewable energy integration brings dynamic security deterioration of power systems, which would lead to the operation risks [3]. However, the deployment of phasor measurement units (PMUs) provides a promising way to improve awareness ability of control centers for the disturbed operation scenarios. PMUs, the infrastructure of wide-area monitoring system (WAMS) of power system, is able to measure synchronized phasor data with much higher sampling frequency compared with supervisory control and data acquisition (SCADA) [4]. The measurement accuracy is also reported to be sufficiently satisfactory. Since PMUs can grasp the instant response of power

systems when faults occur, how to utilize the massive disturbed trajectories has been significantly investigated in the last decade.

As WAMS are now being deployed in quite a few power systems, PMU is playing an ever increasingly vital role in transient stability awareness [5]. A number of researches have been carried out to evaluate the transient stability by using PMU data. PMU trajectories based indicators are considered as the efficient estimators to understand dynamic features of power systems, especially during severe disturbances. For example, Alvarez et al proposed seven trajectory based indices which are suitable for fuzzy inference on real-time dynamic vulnerability [6]. A phasor data–based energy function indicator was developed in [7] aiming at monitoring the dynamic status of power transfer paths. A real-time transient stability assessment (TSA) method based on centre-of-inertia estimation from PMU records was reported in [8]. From voltage stability aspect, a coupled single-port model was applied to establish WAMS based assessment indicator [9]. Furthermore, Makarov et al. [10] presented a review on PMU-based TSA, offering a clear roadmap for further development.

Machine learning techniques have been widely applied for TSA. Most of the existing works are focused on the binary state prediction for global stability using clustering and classification. For example, support vector machine, decision tree and artificial neural network (ANN) are widely used to detect instability of power systems by using post-fault dynamic data during a few cycles [11–13]. Guo and Milanović presented a probabilistic framework to evaluate the accuracy of data mining tools applied for online prediction of transient stability [14], enabling the comprehensive analysis of performance of different implementations.

However, few machine learning techniques have considered the impact of the critical unstable generators (CUGs) of power systems. The majority of the researches have focused on the identification of the global system status due to the fact that a power system normally has hundreds of generators which generate massive volumes of data [15]. As a result, it has become a challenge for standalone machine learning techniques running on single computers to deal with stability assessment taking into account CUGs identification. For example, Passaro et al. [16] employed adaptive neural network to evaluate stability for each generator, admitting that standalone neural networks can hardly solve the problem in a reasonable time. For this purpose, applying advanced computing techniques to enable high-performance training and prediction associated with PMU measured data has become a necessity.

It is well known that neural network is highly adapted to classification tasks [17]. A number of researchers employed neural network to achieve high accuracy classifications in both academia and industrial fields. References [18, 19] figured out that BPNN encounters low efficiency issue due to large number of sum and sigmoid calculations. Some researchers focused on speeding up BPNN using cloud computing techniques. For example, Yuan et al. [20] implemented parallel BPNN using cloud computing technique. Ikram et al. [21] also employed cloud computing to parallelize BPNN in training phase. And also some researchers focused on solving the issue using MPI [22]. However, their ideas are all based on data separation, which does not consider the accuracy loss caused by the simple data separation. Therefore, to improve the efficiency of BPNN whilst maintains classification accuracy in predicting CUGs, this paper presents a MapReduce based parallel back propagation neural network(BPNN) algorithm. The algorithm firstly employs ensemble techniques [23] to complement the data information loss in data separation. And then the mappers in Hadoop clusters start training a number of sub-BPNNs. Finally, these sub-BPNNs can be employed to classify instances by fed with a few cycles of post-fault data and output final prediction results based on majority voting.

## 2 ANN-based CUGs prediction

### 2.1 Definition

CUGs are defined as the earliest group of generators rotor angles of which have a leading or lagging tendency compared with the rest units after fault clearing. The term of tendency refers to the given threshold of power angle difference between any pair of generators. Technically, CUGs are the most severely disturbed units that may lead to the ultimate loss of stability [24]. On the other hand, they are the potential control candidates for emergency tripping or correction action which is able to quickly diminish instability risk of power systems. The clustering-based method of identifying CUGs is detailed in the following section. Fig. 1 illustrates a few examples of CUGs in terms of rotor angle trajectories.

The unstable generators is belonged to CUGs, because their leading (or lagging) rotor angle against other units must exceed the given threshold which is usually set to be equal or little smaller than the wide-accepted instability criterion. For example, Fig. 1a and Fig. 1b illustrate rotor angle trajectories of CUGs which also contain all the unstable generators. It is a similar situation in Fig. 1d. In this situation, all the generators are determined as unstable ones at the end of observation time window, 150 cycles. But before that, none of generators reaches the CUG threshold criterion. Therefore, the strict two-cluster instability pattern corresponds to the situation that all the generators are CUGs, such as the case of Fig. 1d. However,

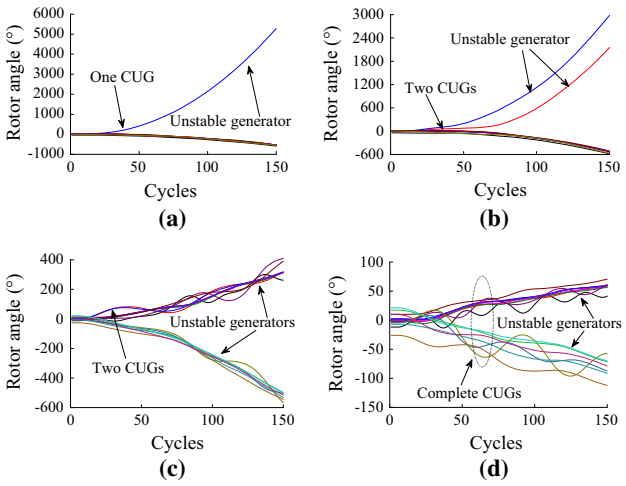



**Fig. 1** Illustration of CUGs

unlike Fig. 1a, Fig. 1b and Fig. 1d, Fig. 1c offers the different pattern in which CUGs only are part of unstable units. Although belongs to the leading cluster, ahead of other leading generators, the two generators indicated in Fig. 1c meet CUGs identification criterion at the very beginning of time windows. These two units are considered to be the most effective objects for the further control strategy.

The primary aim of this study is to enable fast CUGs prediction by means of large-scale parallelized BPNNs learning method, providing more in-depth information for situational awareness of power systems transient stability.

### 2.2 Clusterwise CUGs identification

It is not difficult to distinguish unstable generators from the rest stable ones through the trajectory plot of rotor angle in a few seconds, as shown in Fig. 1a, and Fig. 1b. However, due to the lack of commonly used confirmation criteria for CUG, a method which is able to identify CUGs based on *k*-means clustering algorithm is presented in the paper. For each fault scenario, CUGs can be confirmed by means of following procedure.

Step 1: Collect rotor angle trajectory of each generator in a few seconds.

Step 2: Calculate rotor angle difference of any two generators *i* and *j* cycle by cycle from the very beginning of post-fault point according to:

$$\Delta\delta_{ij}(t) = \delta_i(t) - \delta_j(t) \quad \forall i,j \tag{1}$$

where $\Delta\delta_{ij}(t)$ is the angle difference between $i$ and $j$ at cycle point $t$ after fault. If $\Delta\delta_{ij}(t)$ exceeds the given threshold, e.g. 170°, where the power system is critically unstable, then record the time point $t$. Otherwise, the procedure is terminated for the next round.

Step 3: Extract a specific power angle trajectory $\delta_i(t+\Delta t)$ of every individual generator for further analysis. Here $\Delta t$ refers to the CUG validation interval. If $\Delta t$ is selected to be a relatively large value, like 3 s, it is hardly possible to distinguish CUGs from subsequently potential unstable generators. Empirically, $\Delta t$ is preferably set to be 50 cycles, i.e. 1 s.

Step 4: Classify all the $\delta_i(t+\Delta t)$ trajectories into two groups by means of widely-used *k*-means clustering algorithm, followed by calculating center of inertia (COI) against classified rotors of each individual group $A$ and $B$ respectively using:

$$\delta_{\mathrm{COI}}^k = \frac{1}{M_T^k}\sum_{i=1}^{N_k} M_i^k \delta_i^k, \quad M_T^k = \sum_{i=1}^{N_k} M_i^k,\ k \in \{A,B\} \tag{2}$$

where $\delta_i^k$ and $M_i^k$ are rotor angle and inertia constant of generator $i$ which belongs to group $k$, $N_k$ refers to the number of generators in group $k$. It is worth noting that since the number of clusters is confirmed according to first swing situation, the replicate *k*-means, detailed in Appendix A, is employed in order to overcome the drawback of randomly selecting initial centroids.

### 2.3 Features selection

According to the previous works, a variety of dynamic parameters can be selected to compose features vector for the training procedure of particular machine learning algorithms. There exist two basic types of feature selection, i.e. time-series synchronized data such as a few cycles of voltage trajectory [25] and dynamic performance indices such as kinetic energy indicator of rotors [11].

In order to avoid information loss, the features in this study are confirmed to be the straightforward trajectory data after fault clearing including voltage amplitude, rotor angle and rotor speed of each individual generator. In addition, the maximal kinetic energy, a widely-used indicator highly related to disturbance severity of single generator, is also considered as one of the features. The features vector of generator $n$ for one sample with time interval $\Delta T$ is denoted as:

$$F_n(\Delta T) = \{\boldsymbol{V}(T), \boldsymbol{\delta}_{\mathrm{COI}}(T), \boldsymbol{\omega}_{\mathrm{COI}}(T), KE_{\mathrm{COI}}^n\} \tag{3}$$

where $\boldsymbol{V}(T)$, $\boldsymbol{\delta}_{\mathrm{COI}}(T)$ and $\boldsymbol{\omega}_{\mathrm{COI}}(T)$ represent time-series data of voltage amplitude, rotor angle and speed during the time window $T$, respectively. Let $t_c$ denote the exact time when fault clearing accomplishes, $T$ is acquired from $t_c$ to $t_c+\Delta T$. The symbol $KE_{\mathrm{COI}}^n$ refers to the kinetic energy at the instant of one cycle after fault clearing. It is noted that except voltage trajectory array, the time-dependent states such as δ and ω would be mapped to COI coordinate so as to consider mechanical effects of generators. The COI

coordinate transformation and kinetic energy calculation are detailed in [24]. The COI coordinate is a widely accepted method of considering global pattern of system stability including the impacts of all the generators. The COI coordinate is applied in training and prediction stage, ensuring generalization ability of ANNs for a specific generator with the inclusion of global impacts of stability pattern. In addition, it is noted that the shape of transient voltage recovery which reflects disturbance severity of a generator closely relates to the stability evaluation of this generator. The disturbed voltage trajectory is also reported as a well-performing attribute for machine learning based transient stability prediction of power systems [25].

Features dimension significantly affects ANN training performance as well as generalization ability. Large size dimension is prone to over-fitting while the short one probably leads to inadvertent information loss. However, it is difficult to confirm the preferable dimension of feature candidates before validating fitting performance. In this regard, the parameters vectors with different dimension are tested by training a set of parallel ANNs in order to determine the optimal dimension for the input features. According to the vector defined by (3), the length of input features, $d(\Delta T)$, is of linear dependence in terms of $\Delta T$, shown as:

$$d(\Delta T) = 3 \times \mathrm{num}(\Delta T) + 1 \tag{4}$$

where the symbol num($\Delta T$) refers to the number of cycles existing during $\Delta T$ interval. The triple means adding the length of vector containing $\boldsymbol{V}(T)$, $\boldsymbol{\delta}_{\mathrm{COI}}(T)$ and $\boldsymbol{\omega}_{\mathrm{COI}}(T)$ shown in (3). The kinetic energy indicator $KE^n_{\mathrm{COI}}$ is selected as the last feature, adding one more factor into the final vector of attributes. Taking voltage trajectory of generator bus as an example, the principle of determining preferable features dimension is detailed as follows.

As shown in Fig. 2, the time-series voltage amplitude after fault clearing during $\Delta T$ is taken as a sub-array of features, i.e. $\boldsymbol{V}(T)$. If $\Delta T$ equals 0.4 s, for example, the number of cycles during $\Delta T$ is 20 according to PMU data acquisition frequency. In this case, the total dimension

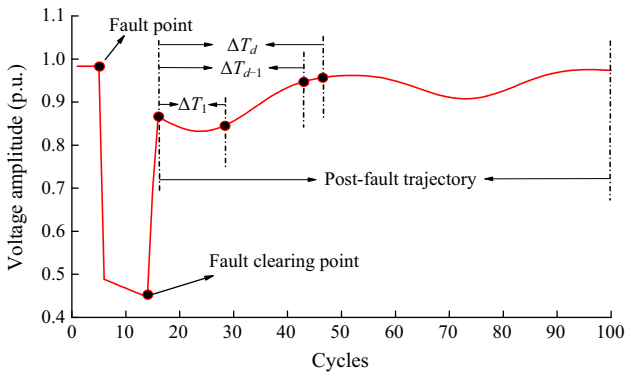



**Fig. 2** Illustration of confirming features dimension

given by (4) is 61. Aiming to determine the most effective features dimension for training regarding the trade-off between accuracy and computing cost, a number of ANNs fed by diverse dimension features associated with different $\Delta T$ are trained simultaneously. Specifically, $\Delta T$ is taken from 0.08 s to 0.56 s with step of 0.04 s, respectively corresponding to the intervals of 4 to 28 cycles using 2 cycles step-size after fault clearing. That means, for a single generator $n$, a set of ANNs denoted as:

$$ANN_1^n \quad ANN_2^n \quad ANN_3^n \ldots ANN_{13}^n \tag{5}$$

are involved in the same training scheme which is challenged by the extremely computing-intensive machine learning tasks. However, training ANNs tentatively with all the potential input dimension is a straightforward way to ascertain the most generalized structure for the ANN trained for each generator.

At the output side of ANN, on the other hand, learning target against each data sample is given to be a binary variable reflecting status of an individual generator. If a particular generator is identified to be critically unstable by clusterwise method, the target status is tagged as 1, otherwise, it equals to 0.

### 2.4 Data samples production

A time-domain simulation based program is designed to automatically produce massive data samples associated with determined features. Specifically, the data generation procedure is applied by the following steps.

Step 1: Scale the initial load level of base case through multiplying active power $P_{Li0}$ on each bus by a random stress factor $\alpha$, while update reactive power $Q_{Li}$ using the constant power factor shown as:

$$P_{Li} = P_{Li0}(1+\alpha) \quad i \in \{\mathrm{PQ}\} \tag{6}$$

$$Q_{Li} = P_{Li} \cdot \tan\left\{\cos^{-1}\left(P_{Li0} \Big/ \sqrt{P_{Li0}^2 + Q_{Li0}^2}\right)\right\} \tag{7}$$

where $\alpha$ ranges from $-0.25$ to 0.6. All generators, except the reference one, offset the load variation proportional to their base generations. The updated output of generator $n$ can be calculated using:

$$P_{Gn} = P_{Gn0}\left(\frac{1 + \sum\limits_{i\in\{\mathrm{PQ}\}} \alpha P_{Li0}}{\sum\limits_{n\neq \mathrm{ref}} P_{Gn0}}\right) \quad n \in \{\mathrm{PV}\} \tag{8}$$

Step 2: Execute time-domain simulation for a severe fault randomly selected from the pre-defined contingency list. The fault clearing time is applied as a random number between 0.15 s and 0.35 s. The longer clearing time implies the higher possibility of instability of power system.

Step 3: Organize features sample for each generator by using the simulated trajectories according to (3). It is noted that for any generator, a set of data samples with different feature dimensions due to the different $\Delta T$ selection (see Fig. 1) are produced at the same time for further parallel training, providing a way to determine the ANN structure with highest generalization ability for each individual generator.

Step 4: Identify CUGs based on the simulation results according to the proposed clusterwise method, enabling the learning target for each data sample.

Step 5: Save all the samples for the current round and repeat the complete procedure from the first step until the given total number of iterations is achieved.

In order to enhance generalization level of data samples, multiple faults are simulated in data producing procedure to consider $N - k$ ($k \leq 3$) scenarios.

## 3 Methodology of MapReduce based parallel BPNN

### 3.1 Issue of data volume

As discussed previously, BPNN has been widely used in quite a lot fields due to its stable performance and remarkable classification accuracy [26]. Therefore, we also employ such type of ANN to create online predictors to classify CUGs rapidly fed by PMUs measured data.

However, in this study, the time-series statuses of all generators produce an extremely large volume of data for ANN learning, which is completely different from the traditional applications. Traditionally, BPNN is frequently applied to train hundreds or a few thousands of instances in order to identify global instability. Thus, the algorithm generates less overhead. Currently because of the sharp increase of data volume, BPNN is forced to deal with massive scale data.

The volume of data is a specific term generally referring to the computable size and scale of a massive amount of data. Fig. 3 illustrates the quantification on the data volume of input instances fed to BPNN training for predicting CUGs.

In Fig. 3, $N$ represents the total length of selected features, $M$ is the number of available samples, and $K$ refers to the sum of generators. Therefore, the cubic data volume can be estimated by

$$\text{Data Volume} = Byte(M \times N \times K) \tag{9}$$

The data volume of fed instances highly affects training efficiency. For processing one instance, overhead occurs in both training and classification phases due to a large number of sum and sigmoid calculations existing in the network. Additionally, in training phase, BPNN has to execute back propagation to tune all the parameters, which generates overhead. At last each instance is not only trained once but a number of times. The training loops also generate overhead. Therefore, the standalone BPNN will meet a critical bottleneck for processing large volume of data in terms of efficiency. Our data intensive task will deteriorate the performance of CUGs prediction. This motivates us to parallelize BPNN using MapReduce [27].

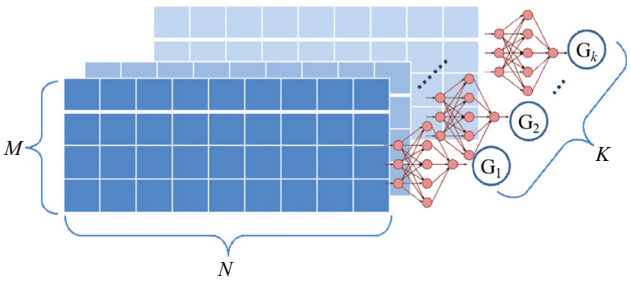



**Fig. 3** Illustration of the volume of data

### 3.2 Parallelization of BPNN

#### *3.2.1 Standalone BPNN*

BPNN is a widely used machine learning technique for classification due to its remarkable function approximation ability. It normally employs only feed forward to output final classification result for each input instance according to the trained weights and biases in training phase. In feed forward, the definitions of related variables are listed in Table 1.

The number of inputs in input layer is decided by $n$, and the number of outputs in output layer is decided by the length of the encoded classifications. Therefore, $I_j$ can be represented by:

$$I_j = \sum_i w_{ij} o_{lj} + \theta_j \tag{10}$$

In typical BPNN, the non-linear equation is frequently using sigmoid, therefore the output of the $j^{\text{th}}$ neuron from the current layer to next layer can be represented by:

**Table 1** Definition of variables used in feed forward stage

| Variable symbol | Definition |
|---|---|
| $n$ | Length of one input instance |
| $w_{ij}$ | Weight from $i^{\text{th}}$ neuron to $j^{\text{th}}$ neuron |
| $\theta_j$ | Bias for varying the activity of the $j^{\text{th}}$ neuron |
| $o_{lj}$ | Output of the $j^{\text{th}}$ neuron from last layer |
| $o_{cj}$ | Output of the $j^{\text{th}}$ neuron of the current layer |
| $I_j$ | Input of the $j^{\text{th}}$ neuron in hidden and output layers |

$$o_{cj} = \frac{1}{1 + e^{-I_j}} \tag{11}$$

Output layer finally outputs its $o_{cj}$ which indicates the classification result, and then feed forward completes. Following, back propagation starts. In back propagation, the related variables are defined in Table 2.

Therefore, $Err_j$ in output layer is expressed by:

$$Err_j = o_j(1 - o_j)(t_j - o_j) \tag{12}$$

while $Err_j$ in hidden layers can be represented by:

$$Err_j = o_j(1 - o_j) \sum_k Err_k w_{kj} \tag{13}$$

As a result, the weight $w_{ij}$ and bias $\theta_j$ can be tuned using:

$$w_{ij} = w_{ij} + Err_j o_j \tag{14}$$

$$\theta_j = \theta_j + Err_j \tag{15}$$

BPNN terminates its training procedure based on two conditions. The first one is that if the loop reaches a certain number, the algorithm terminates. The second condition is that if the error reaches a given threshold according to (16) for the single output and (17) for the multi-outputs:

$$\min(E[e^2]) = \min(E[(t - o)^2]) \tag{16}$$

$$\min\left(E\left[e^{\mathrm{T}}e\right]\right) = \min\left(E\left[(t - o)^{\mathrm{T}}(t - o)\right]\right) \tag{17}$$

### 3.2.2 MapReduce and Hadoop framework

MapReduce is a remarkable distributed computing model, offering two main operations named as Map and Reduce. Map function is responsible for data processing and computation. Reduce function, however operates the collecting and outputting operations. Specifically, the inputs and outputs for Map and Reduce are controlled by key-value pairs. Map processes each input *key-value* pair {K1, V1} and outputs intermediate output {K2, V2}. Reduce collects the output pairs with the same keys and executes merging, shuffling operations. And then Reduce outputs the final results {V2}.

Among a number of MapReduce implementations [28, 29], Hadoop framework [30] is the most famous one. Specifically, the nodes in a Hadoop cluster contribute their resources including processors, memory, hard disks and network adaptors to form hadoop distributed file system (HDFS) which is not only aiming at storing data but also the basic infrastructure of Hadoop. The nodes are categorized into one NameNode managing the metadata of cluster and several DataNodes executing computations. The implementations of Map and Reduce functions in Hadoop are named as mapper and reducer which are located in DataNodes. Based on the design of HDFS, Hadoop supplies remarkable scalability, fault tolerance, load balancing, heterogeneous environment support, and efficiency in dealing with large volume of data.

**Table 2** Definition of variables used in back propagation

| Variable symbol | Definition |
|---|---|
| $Err_j$ | Error-sensitivity of certain layer |
| $t_j$ | Desirable output of neuron $j$ in the output layer |
| $Err_k$ | Error-sensitivity of one neuron in the last layer |
| $w_{kj}$ | Corresponding weight of $Err_k$ |

### 3.2.3 Ensemble technique

The presented parallelization of BPNN is based on data separation. The main idea is to divide the training data set into a number of sub-sets. Each sub-set is input into a sub-BPNN maintained by one mapper in the Hadoop cluster. As a result, each sub-BPNN is only trained by a part of original data set so as to improve the training efficiency. However, the merely simple data separation encounters a problem that partial training data results in insufficiently trained NN, which may lose accuracy in classification. Therefore, this paper employs ensemble technique including bootstrapping and majority voting. The solution of insufficient training for sub-BPNN based on bootstrapping and majority voting [23] is further detailed in Appendix B.

### 3.2.4 Algorithm design

As long as each bootstrapped sub-set is generated by using bootstrapping, each instance in one sub-set is defined in the format of $<instance_k, target_k, type>$, where

$instance_k$ represents the bootstrapped instance, which is the input of neural network; $target_k$ represents the desirable output if $instance_k$ is a training instance; *type* field has two values: "*train*" and "*test*", which labels the type of $instance_k$. Therefore the sub-BPNN in a mapper is aware of the instance type moreover executing training or classification operations.

Afterwards each individual mapper in the Hadoop cluster constructs one BPNN and initializes weights and biases with random values between -1 and 1 for its neurons. And then the mapper inputs one instance in the form of $<instance_k, target_k, type>$ from one input sub-set of the mapper.

The mapper parses the data and retrieves the type of the instance. If the type value is "*train*", the instance is fed into the input layer. Secondly, each neuron in different layers computes its output until the output layer generates an output which indicates the completion of feed forward process. And then the mapper starts a back propagation process and updates weights and biases for neurons. If one

input instance is labeled as "*test*", all the mappers start to classify the instances. In this case, each mapper generates an intermediate output in the form of $< instance_k, o_{jm}>$ where $instance_k$ is the key and $o_{jm}$ represents the classification result of the $m^{th}$ mapper.

Finally, a reducer collects the intermediate outputs of all the mappers. As all the outputs have the same key $instance_k$, the reducer merges these outputs into one set, in which the reducer executes majority voting and outputs the finally voted result of $instance_k$ into HDFS in the form of $<instance_k, r_k>$ where $r_k$ represents the voted classification result of $instance_k$. Fig. 4 and Fig. 5 illustrate the algorithm architecture for training and classification procedure of MapReduce based BPNN respectively.

Based on the majority voting, a number of sub-BPNNs (weak classifiers) can form a strong classifier. Therefore, although each sub-BPNN is only trained by a portion of the original training data which may lead to the wrongly classification, the final voted classification result of a number of sub-BPNNs has a higher chance to be correct with higher efficiency for dealing with a large volume of data.

### 3.3 Implementation framework

The presented technique is able to be implemented to WAMS application platform, enabling online prediction of transient stability for each generator after fault clearing. Compared with the conventional methods only predicting global stability [16], the proposed approach could provide more in-depth information for the potential emergency control which aims to resynchronize the disturbed power system as fast as possible. The implementation framework is simply shown in Fig. 6.

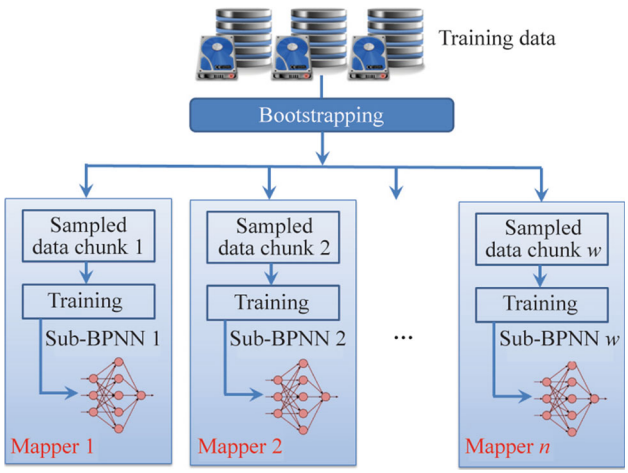



**Fig. 4** Algorithm architecture of parallelized BPNN training

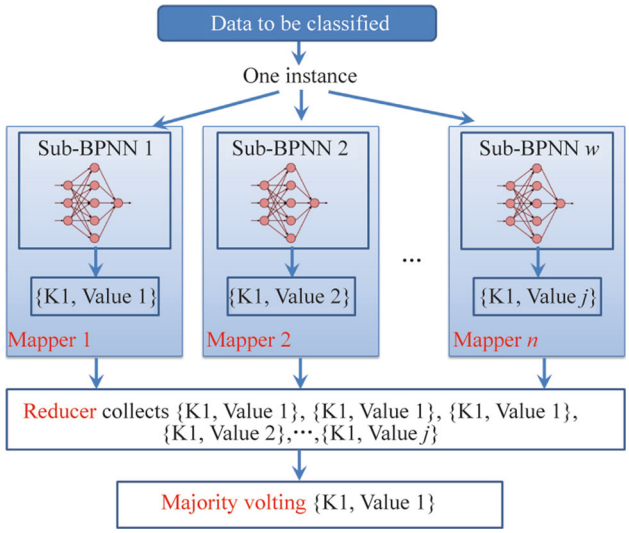



**Fig. 5** Algorithm architecture of parallelized BPNN classification

As illustrated in Fig. 6, a distributed simulator for fault response of power systems according to previous study [31] serves to generate massive data samples based on the given operation point in a parallel computing environment. Although it is an off-line procedure from the traditional concept, however, due to the fast variability of operation point including unexpected topology change, updating training samples ensures generalization of data. Therefore, the simulated fault scenarios which reflect various stability patterns of all the generators are collected accumulatively to update sample database in the proposed framework, resulting in an intensive computation burden of the standalone ANN training.

In the previous literature, the well-trained ANN is used as stability predictor which normally does not need update. However, the generalization of ANN could be enhanced by means of re-training the updated transient samples. The presented technique, shown in the dotted frame, enables high-performance computing framework and algorithm of large-scale parallelized BPNNs training. The training

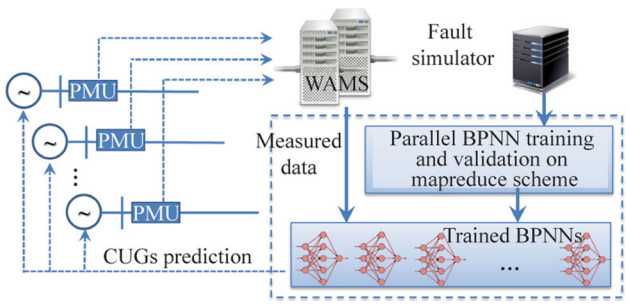



**Fig. 6** Implementation framework built on the presented technique

efficiency highly depends upon the number of DataNodes, which operates in a similar way of distributed samples generation [31].

Theoretically, in order to enable industrial application, the time consumption of samples update and thousands of BPNNs re-training could be reduced to an acceptable level, few minutes for example, by configuring sufficient computing resources based on the scalable Hadoop framework. The trained parallel BPNNs for each generator are stored in the distributed environment. When a fault occurs in power system, the PMUs installed on generator busbars capture the disturbance signal in a few cycles, providing the trajectories-based features which are imported into the well-trained parallel CUGs predictors that can avoid time-consuming series ANNs prediction. As a result, CUGs information, the most leading or lagging generators, will be sent back to the WAMS application, contributing to further emergency or correction control.

## 4 Case study

### 4.1 Test systems and training data

NPCC 48-machines test system and a provincial power system in the southwest of China [31] are used to validate the proposed technique. Details of test systems are shown in Table 3.

In order to obtain BPNN-based CUGs predictor with high generalization ability, rich data samples are required, preferably including all the potential operation conditions and fault modes. Therefore, massive samples are generated for both test systems. In this work, a distributed random fault simulator has been developed to generate massive samples [31]. Random fault refers to stochastic three-phase short circuits on any transmission lines. In addition, fault clearing time is randomly set between 0.1 s to 0.35 s. The samples generation procedure is in the following steps.

Step 1: Load base case, if the initial outage exists, trip the component and calculate power flow.

Step 2: Change $P$ and $Q$ on each bus by multiply a random number in the range of [0.8, 1.4] to simulate load level, allocate unbalance power to all generators in proportion to their base generation.

Step 3: Implement three-phase fault on a randomly selected component at time $T_f$, clear fault at $T_f+\mu$, where $\mu$ is a random decimal in [0.1, 0.35].

**Table 3** Details of test system

| Test system | Bus | Branch | Generator | Base load (GW) |
|---|---|---|---|---|
| System I | 140 | 233 | 48 | 28.03 |
| System II | 762 | 889 | 146 | 13.56 |

**Table 4** Details of simulated samples

| Test system | Fault modes | | | Stable | Unstable |
|---|---|---|---|---|---|
| | $N$-1 | $N$-2 | $N$-3 | | |
| System I | 8450 | 4600 | 2950 | 11,426 | 4574 |
| System II | 14,252 | 6223 | 3525 | 17,208 | 6792 |

**Table 5** Volume of data

| Test system | Total volume (GB) | $V_{max}$ (MB) | $V_{min}$ (MB) |
|---|---|---|---|
| System I | 5.188 | 14.67 | 2.77 |
| System II | 20.042 | 21.78 | 3.94 |

Step 4: Perform time-domain simulation for above randomly configured operation and fault scenario, collect output trajectories to calculate features.

The results of data samples production based on the fault simulator are detailed in Table 4. In addition, the volume of data which are fed to train and validate parallel BPNNs in this study is shown in Table 5 according to (9).

In Table 5, $V_{max}$ and $V_{min}$ represent the maximal and minimal block of data samples for individual generator in test systems. Table 4 and Table 5 indicate that the massive samples production as well as $\Delta T$-dependent features definition for generators according to (5) results in a huge volume of data. However, a standalone BPNN inputs instances one by one leading to both sizable IO overhead and considerable computational overhead. As a result, the most reasonable way of identifying the CUGs in such a great volume of data is to apply the presented parallel BPNN which parallelizes both training and classification phase. Furthermore, it is the most feasible approach to enable WAMS application integrated with CUGs prediction according to the architecture in Fig. 6.

### 4.2 Computing cluster configuration

In order to evaluate the performance of MapReduce based parallel BPNN enabling predicting CUGs after fault clearing, a practical Hadoop cluster was built up. The cluster contains ten nodes, nine of which are Datanodes and the rest one is Namenode. Table 6 shows the configuration of the cluster.

**Table 6** Configuration of computing cluster

| | CPU (GHz) | Memory (G) | SSD (G) | Operating system |
|---|---|---|---|---|
| Namenode | Core i7@3 | 8 | 750 | Fedora |
| Datanode | Core i7@3.8 | 32 | 250 | Fedora |

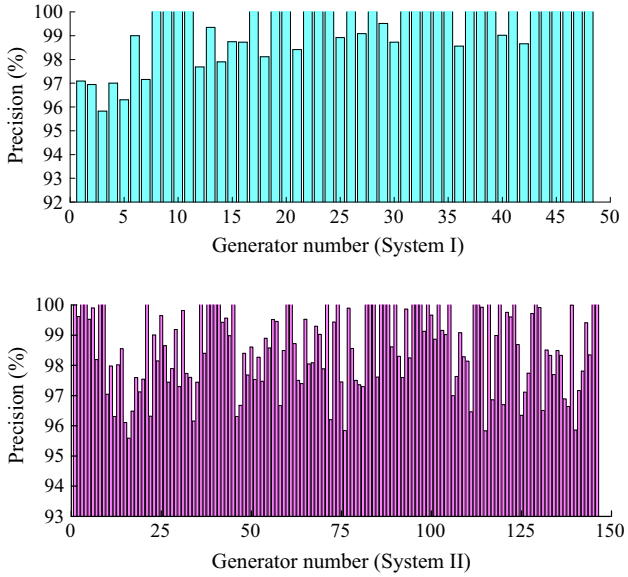



**Fig. 7** Precision of predicted CUGs

## 4.3 Evaluation of MapReduce based parallel BPNN

### *4.3.1 Precision of CUGs prediction*

In this evaluation, we tested the algorithm precision of the generator status prediction. In terms of precision, when the number of training instance is large, the presented MapReduce based parallel BPNN has the same precision compared to that of standalone BPNN. Therefore, the following only lists the precision of the parallel BPNN without comparison with a standalone BPNN algorithm. Fig. 7 illustrates the precision of CUG identification for two test systems.

The figure recording the CUGs predicting precision of test systems indicates that the parallel BPNN is of satisfactorily high precision in identifying the generators transient status during the post-fault period of power systems. The average precisions for all generators of two test systems are 99.18% and 98.57% respectively.

However, it is worth noting that features dimension which depends upon the determination of $\Delta T$ affects CUGs prediction precision. In this study, we choose BPNNs possessing the highest average precision to be the CUGs predictor, even though it is not the best $\Delta T$ choice for all the generators. Table 7 gives the details.

Table 7 indicates that when the value of $\Delta T$ equal to 5 cycles and 8 cycles respectively, two test systems get the BPNNs predictor with highest average precision. However, the particular machines, 3$^{rd}$ generator in system I and 16$^{th}$ generator in system II for example, could only obtain their most precise NN-based predictors by setting longer $\Delta T$.

### *4.3.2 New samples validation*

In order to test the generalization ability of parallelized ANNs based CUGs predictor, thousands of new simulated samples of two test systems are generated and fed into the trained ANNs to assess their prediction precision. The additional samples information is listed in Table 8.

Aiming to validate the comprehensive performance of MapReduce based CUGs predictor with new data cases, the numerical studies are investigated by retraining the BPNNs using the initial data samples introduced in Table 4 with different $\Delta T$ setting. Then the well-trained BPNNs are applied to predict CUGs using the data of new cases as input with the corresponding $\Delta T$ setting. The test results of precision statistics are provided in Table 9.

The validation results listed in Table 9 indicate that the parallelized BPNNs based on the presented technique have a well-performance of generalization ability for the new data cases which are not included in the training data set.

### *4.3.3 Features dimension impact analysis*

As discussed above, the features dimension determined by $\Delta T$ according to (3) is highly related to the prediction precision. However, it is hardly to understand in advance that which $\Delta T$ choice is of the highest precision. Owing to MapReduce and HDFS technique, the parallelized BPNNs with thirteen types of features dimension for every single generator are trained and validated at the same time.

In this test, we focus on the comparison of algorithm precision with increasing dimension of the instance. Fig. 8 shows the variation of CUGs prediction precision against the change of $\Delta T$.

**Table 7** Precision details of test systems

| Test system | $\Delta T$ (s) | Average precision (%) | Generator with lowest precision | | |
|---|---|---|---|---|---|
| | | | Generator No. | Precision (%) | $\Delta T_{best}$ (s) |
| System I | 0.10 | 99.18 | 3$^{rd}$ | 95.82 | 0.20 |
| System II | 0.16 | 98.57 | 16$^{th}$ | 95.59 | 0.22 |

**Table 8** Details of new cases for prediction precision test

| Test system | Fault modes | | | Stable | Unstable |
|---|---|---|---|---|---|
| | *N*-1 | *N*-2 | *N*-3 | | |
| System I | 1210 | 820 | 384 | 1722 | 692 |
| System II | 2290 | 1315 | 945 | 3295 | 1255 |

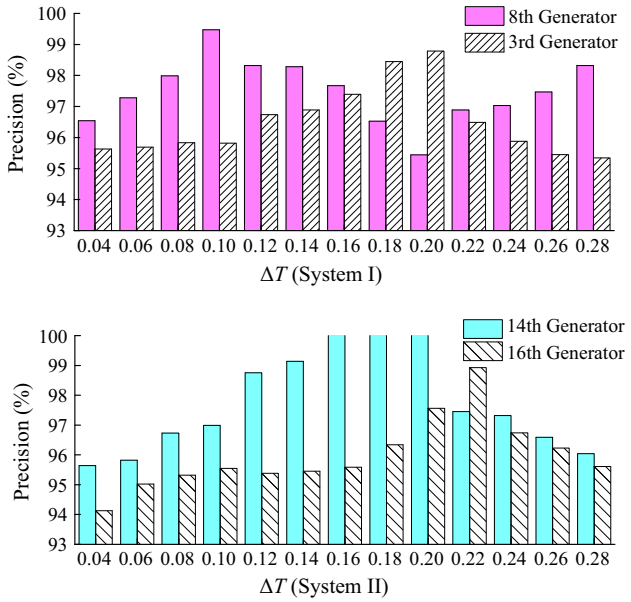



**Fig. 8** Variation of prediction precision against the change of $\Delta T$

It can be observed that, along with an increasing number of elements involved in an instance, the final precision of BPNNs trained for single generator changes in a nonlinear way. It is assumed that over-fitting largely affects training procedure. Moreover, due to the major part of generators obtain their most accurate BPNNs when $\Delta T$ equal 0.1 s and 0.16 s respectively in System I and System II, the average precisions are highest by selecting these $\Delta T$ for all generators. Fig. 9 illustrates statistical distribution of BPNNs with highest precision in term of $\Delta T$.

### 4.3.4 *Validation of ensemble training*

Figure 10 indicates that the presented ensemble based neural network algorithm outperforms the standalone neural network in terms of precision.

The figure shows that, when the number of training instance is small, the precision of the ensemble training strategy in the presented parallel BPNN outperforms that of a standalone BPNN algorithm. The figure also tells that the precision of ensemble training strategy increases stably without fluctuations.

### 4.3.5 *Algorithm efficiency*

In this test, we primarily focused on the evaluation of algorithm efficiency. We duplicate the training data from 1 MB to 1024 MB, with 16 mappers employed. The experimental result is shown in Fig. 11.

It can be observed that when the data size is small, the standalone BPNN outperforms the parallel BPNN due to the overhead of Hadoop framework. However, when the data size keeps increasing, the parallel BPNN can still execute efficiently. Contrarily, the standalone BPNN needs more time to execute the data processing.

According to the test result, the time consumption of training 1 GB data by standalone BPNN costs around 9000

**Table 9** Results of new sample cases test

| Test system | $\Delta T$ | Average precision (%) | Generator with lowest precision | |
|---|---|---|---|---|
| | | | Generator no. | Precision (%) |
| System I | 0.06 | 97.24 | 3rd | 94.82 |
| | 0.08 | 98.09 | 5th | 95.17 |
| | 0.10 | 98.32 | 12th | 95.98 |
| | 0.12 | 99.07 | 3rd | 96.72 |
| System II | 0.12 | 97.21 | 16th | 94.51 |
| | 0.14 | 98.37 | 76th | 95.66 |
| | 0.16 | 98.82 | 16th | 95.37 |
| | 0.18 | 98.66 | 35th | 95.29 |

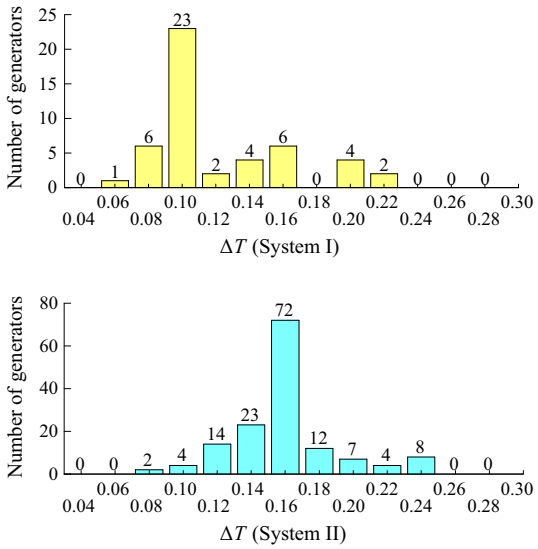



**Fig. 9** Distribution of BPNNs with highest precision

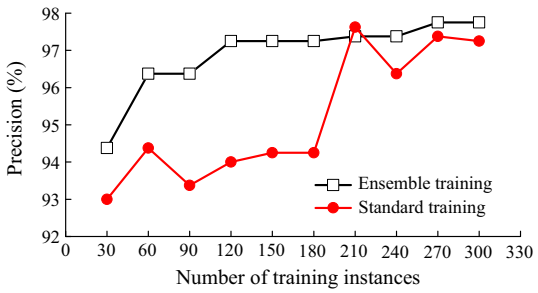



**Fig. 10** Precision of less number of training instances

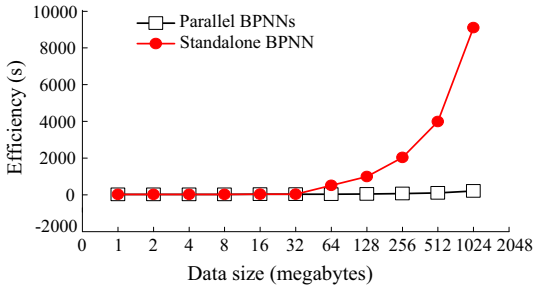



**Fig. 11** Distribution of BPNNs with highest precision

**Table 10** Efficiency comparison

| | Standalone BPNN (h) | MapReduce based Parallel BPNNs | | |
|---|---|---|---|---|
| | | 2 Datanode (s) | 5 Datanode (s) | 9 Datanode (s) |
| System I | 6.5 | 2482 | 1148 | 682 |
| System II | 23.4 | 9233 | 3827 | 2083 |

s. It is obviously not acceptable for continually updated samples production application, such as the technique architecture presented in Fig. 6. In order to investigate efficiency of the presented CUGs prediction algorithm, the comprehensive comparison in term of training time consumption is performed in Table 10.

The efficiency comparison indicates that along with the computing resource increasing, the algorithm processing time is largely reduced without accuracy loss. If more Datanodes are configured, the efficiency will be further improved so as to accomplish training procedure in a few minutes even a few seconds. That enables the on-line training for the huge bulk of updated samples.

## 5 Conclusion

This paper presents a high-performance CUGs prediction approach using MapReduce based parallel BPNN. Our work in the first place employs time domain simulation to generate massive disturbed scenarios using the published fault simulator. Secondly we propose features selection principles to produce feature vector which represents the system status with reasonable data dimension. Thirdly for overcoming the disadvantages of standalone application, MapReduce based BPNNs technique is developed aiming to facilitate simultaneous training for every single generator. The presented methodology employs ensemble technique and data separation to enhance the training efficiency, whilst it uses data separation in enabling scalable integration of computing resource as well as large classification efficiency improvement. The experiment results show that the presented technique is able to predict CUGs especially in large-scale data with high accuracy and efficiency, providing a way to in-depth awareness of stability based on WAMS architecture.

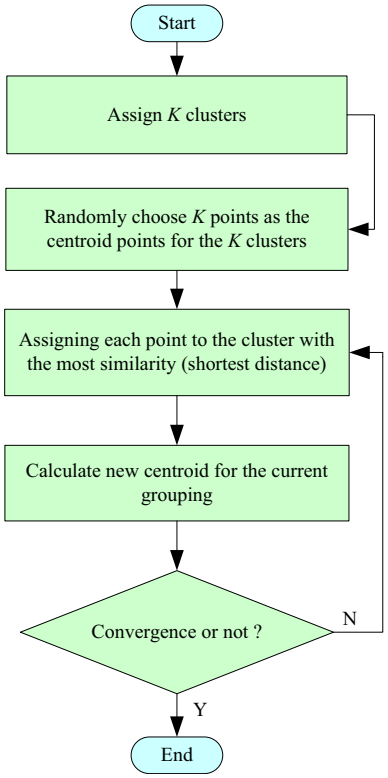


## Appendix A

Generally k-means algorithm tries to minimize the within-cluster sum of squares, that is, to minimize

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x_j \in S_i} D^2\left(x_j, \mu_i\right)$$

where $k$ is the number of cluster, $S_i$ is the set of the $i^{\text{th}}$ cluster (or elements in the $i^{\text{th}}$ cluster), $\mu_i$ is the mean of the points in $i^{\text{th}}$ cluster and $D^2(x_j, \mu_i)$ is the distance between point $x_j$ and $\mu_i$.




The above flow chart is the primary procedure of $k$-means algorithm. The third and fourth step can be further expressed as following mathematic equations respectively

$$S_i^{(t)} = \left\{ x_p : D\left(x_p, \mu_i^{(t)}\right) \le D\left(x_p, \mu_j^{(t)}\right) \quad \forall 1 \le j \le k \right\}$$

$$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$$

where $(t)$ means the $t^{\text{th}}$ iteration and $|S_i^{(t)}|$ denotes the number of elements in $i$th set ($S_i$). Each point $x_p$ is assigned to exactly one set $S_i$.

The "replicate" indicates the number of times to repeat the clustering, each with a new set of initial cluster centroid positions. $k$-means returns the solution with the lowest value for sum of distance. The times to repeat the clustering depends on the distinction in clustering data. The choice of the initial clustering center has a large effect on the clustering results. It may lead bad results if the initial clustering center is not appropriate. In this case, increase the times to repeat the clustering with different set of initial cluster centroid positions can improve the accuracy of clustering results.

## Appendix B

To solve the issue of insufficient training for sub-BPNN, the solution is to re-sample the training dataset based on bootstrapping and then execute majority voting. The operations are able to reduce the misclassification errors and increase the classification accuracy. Balanced bootstrapping ensures that each training instance equally appears in the bootstrap samples. However, it might not be always the case that each bootstrapping sample contains all the training instances. The method to create the balanced bootstrap samples is to construct a number of instances $X_1$, $X_2$, $X_3$,…, $X_n$ repeating $B$ times so that a sequence of $Y_1$, $Y_2$, $Y_3$,…,$Y_{Bn}$ is achieved. A random permutation $p$ of the integers from 1 to $B_n$ is taken. Therefore the first bootstrapping sample can be created from $Y_p(1)$, $Y_p(2)$, $Y_p(3)$,…, $Y_p(n)$, moreover the second bootstrapping sample from $Y_p(n+1)$, $Y_p(n+2)$, $Y_p(n+3)$,…, $Y_p(2n)$ and so on, until $Y_p((B-1)n+1)$, $Y_p((B-1)n+2)$, $Y_p((B\text{-}1)n+3)$,…, $Y_p(Bn)$ is the $B^{\text{th}}$ bootstrapping sample. The majority voting is a commonly used combination technique. The ensemble classifier predicts a class for a testing instance which is predicted by the majority of base classifiers.

## References

[1] Baldick R, Chowdhury B, Dobson I et al (2009) Vulnerability assessment for cascading failures in electric power systems. In: Proceedings of the 2009 power systems conference and exposition (PSCE'09), Seattle, WA, 15–18 Mar 2009, 9 pp
[2] Vaiman M, Bell K, Chen Y et al (2012) Risk assessment of cascading outages: methodologies and challenges. IEEE Trans Power Syst 27(2):631–641
[3] Miao L, Fang JK, Wen JY et al (2013) Transient stability risk assessment of power systems incorporating wind farms. J Mod Power Syst Clean Energy 1(2):134–141. doi:10.1007/s40565-013-0022-2
[4] De La Ree J, Centeno V, Thorp JS et al (2010) Synchronized phasor measurement applications in power systems. IEEE Trans Smart Grid 1(1):20–27
[5] Makram EB, Vutsinas MC, Girgis AA et al (2012) Contingency analysis using synchrophasor measurements. Electr Power Syst Res 88(1):64–68
[6] Alvarez JMG, Mercado PE (2007) Online inference of the dynamic security level of power systems using fuzzy techniques. IEEE Trans Power Syst 22(2):717–726
[7] Chow JH, Chakrabortty A, Arcak M et al (2007) Synchronized phasor data based energy function analysis of dominant power transfer paths in large power systems. IEEE Trans Power Syst 22(2):727–734
[8] Cepeda JC, Rueda JL, Colomé DG et al (2014) Real-time transient stability assessment based on centre-of-inertia estimation from phasor measurement unit records. IET Gener Transm Distrib 8(8):1363–1376
[9] Liu JH, Chu CC (2014) Wide-area measurement-based voltage stability indicators by modified coupled single-port models. IEEE Trans Power Syst 29(2):756–764

[10] Makarov YV, Du PW, Lu S et al (2012) PMU-based wide-area security assessment: Concept, method, and implementation. IEEE Trans Smart Grid 3(3):1325–1332
[11] Gomez FR, Rajapakse AD, Annakkage UD et al (2011) Support vector machine-based algorithm for post-fault transient stability status prediction using synchronized measurements. IEEE Trans Power Syst 26(3):1474–1483
[12] Kamwa I, Samantaray SR, Joos G (2010) Development of rule-based classifiers for rapid stability assessment of wide-area post-disturbance records. In: Proceedings of the 2010 IEEE PES general meeting, Minneapolis, MN, 25–29 Jul 2010, 1 pp
[13] Xu Y, Dong ZY, Zhao JH et al (2012) A reliable intelligent system for real-time dynamic security assessment of power systems. IEEE Trans Power Syst 27(3):1253–1263
[14] Guo TY, Milanović JV (2014) Probabilistic framework for assessing the accuracy of data mining tool for online prediction of transient stability. IEEE Trans Power Syst 29(1):377–385
[15] Hashiesh F, Mostafa HE, Khatib AR et al (2012) An intelligent wide area synchrophasor based system for predicting and mitigating transient instabilities. IEEE Trans Smart Grid 3(2):645–652
[16] Al-Masri AN, Ab Kadir MZA, Hizam H et al (2013) A novel implementation for generator rotor angle stability prediction using an adaptive artificial neural network application for dynamic security assessment. IEEE Trans Power Syst 28(3):2516–2525
[17] Cui MJ, Ke DP, Gan D et al (2015) Statistical scenarios forecasting method for wind power ramp events using modified neural networks. J Mod Power Syst Clean Energy 3(3):371–380. doi:10.1007/s40565-015-0138-7
[18] Gu R, Shen FR, Huang YH (2013) A parallel computing platform for training large scale neural networks. In: Proceedings of the 2013 IEEE international conference on big data, Silicon Valley, CA, 6–9 Oct 2013, pp 376–384
[19] Rizwan M, Jamil M, Kothari DP (2012) Generalized neural network approach for global solar energy estimation in India. IEEE Trans Sustain Energy 3(3):576–584
[20] Yuan JW, Yu SC (2014) Privacy preserving back-propagation neural network learning made practical with cloud computing. IEEE Trans Parallel Distrib Syst 25(1):212–221
[21] Ikram AA, Ibrahim S, Sardaraz M et al (2013) Neural network based cloud computing platform for bioinformatics. In: Proceedings of the 2013 IEEE Long Island conference on systems applications and technology (LISAT'13), Farmingdale, NY, 3 May 2013, 6 pp
[22] Long LN, Gupta A (2008) Scalable massively parallel artificial neural networks. J Aerosp Comput Inf Commun 5(1):3–15
[23] Alham NK (2011) Parallelizing support vector machines for scalable image annotation. PhD Thesis. Brunel University, London
[24] Kundur P (2012) Power system stability and control, 3rd edn. McGraw-Hill, New York
[25] Rajapakse AD, Gomez F, Nanayakkara K et al (2010) Rotor angle instability prediction using post-disturbance voltage trajectories. IEEE Trans Power Syst 25(2):947–956
[26] Hagan MH, Demuth HB, Beale MH (1996) Neural network design. PWS Publishing Company, Boston
[27] Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
[28] He BS, Fang WB, Govindaraju NK et al (2008) Mars: a MapReduce framework on graphics processors. In: Proceedings of the 17th international conference on parallel architectures and compilation techniques (PACT'08), Toronto, 25–29 Oct 2008, pp 260–269
[29] Taura K, Kaneda K, Endo T et al (2003) Phoenix: a parallel programming model for accommodating dynamically joining/leaving resources. ACM SIGPLAN Notices 38(10):216–229
[30] Apache Hadoop. http://hadoop.apache.org/. Accessed 07 June 2016
[31] Liu YB, Liu Y, Liu JY et al (2014) A cloud computing framework for cascading failure simulation and analysis of large-scale transmission systems. In: Proceedings of the 2014 international conference on power system technology (POWERCON'14), Chengdu, 20–22 Oct 2014, pp 287–293

**Youbo LIU** received the PhD degree in electrical engineering from Sichuan University, China, in 2011. He is currently a lecturer with the School of Electrical Engineering and Information, Sichuan University, China. His research interests mainly include cascading failure modeling and analysis, machine learning application, and data-driven optimal operation of power systems.

**Yang LIU** is an associate professor with the School of Electrical Engineering and Information, Sichuan University, China. His research interests include distributed computing technologies, high-performance information retrieval, and big data analysis in power systems.

**Junyong LIU** received the PhD degree in electrical engineering from Brunel University, UK, in 1998. He is currently a Professor with the School of Electrical Engineering and Information, Sichuan University, China. His current research interests include power system planning, operation, stability, and big data application.

**Maozhen LI** is a professor with the School of Electronic and Computer Engineering, Brunel University, UK. His research interests include high-performance computing, big data analysis, information retrieval and distributed machine learning techniques.

**Zhibo MA** received his B.Sc. degree from the Sichuan University, China, in 2007, and the M.Sc. degree in power systems from the University of Bath, Bath, in 2008. He is pursuing PhD degree from University of Bath since 2012. He worked various positions in National grid UK since 2009 and currently he is a senior power system engineer in Electricity Control Center. His current research interests include smart grids, renewable energy, and system control.

**Gareth A. TAYLOR** received his B.Sc. degree from the University of London, London, UK, in 1987, and the M.Sc. and PhD degrees in power systems from the University of Greenwich, London, in 1992 and 1997, respectively. He was the National Grid UK. Post-Doctoral Scholar at Brunel University, London, from 2000 to 2003. He is currently a Professor and Director within the Brunel Institute of Power Systems, Brunel University London. His current research interests include smart grids, wide area monitoring of power systems, and network optimization.