applied sciences

MDPI

Article

Algorithm and Implementation of Distributed ESN Using Spark Framework and Parallel PSO

Kehe Wu, Yayun Zhu *, Quan Li and Guolong Han

School of Control and Computer Engineering, North China Electric Power University, Beijing 102206, China; wkh@ncepu.edu.cn (K.W.); power_Lee@sohu.com (Q.L.); wojiaohgl@126.com (G.H.)

* Correspondence: zyyaaa0702@ncepu.edu.cn; Tel.: +86-10-6177-2791

Academic Editor: Christos Bouras

Received: 10 February 2017; Accepted: 30 March 2017; Published: 2 April 2017

Abstract: The echo state network (ESN) employs a huge reservoir with sparsely and randomly connected internal nodes and only trains the output weights, which avoids the suboptimal problem, exploding and vanishing gradients, high complexity and other disadvantages faced by traditional recurrent neural network (RNN) training. In light of the outstanding adaption to nonlinear dynamical systems, ESN has been applied into a wide range of applications. However, in the era of Big Data, with an enormous amount of data being generated continuously every day, the data are often distributed and stored in real applications, and thus the centralized ESN training process is prone to being technologically unsuitable. In order to achieve the requirement of Big Data applications in the real world, in this study we propose an algorithm and its implementation for distributed ESN training. The mentioned algorithm is based on the parallel particle swarm optimization (P-PSO) technique and the implementation uses Spark, a famous large-scale data processing framework. Four extremely large-scale datasets, including artificial benchmarks, real-world data and image data, are adopted to verify our framework on a stretchable platform. Experimental results indicate that the proposed work is accurate in the era of Big Data, regarding speed, accuracy and generalization capabilities.

Keywords: Echo State Network; distributed algorithm; spark framework; Parallel Particle Swarm Optimization; machine learning; Big Data

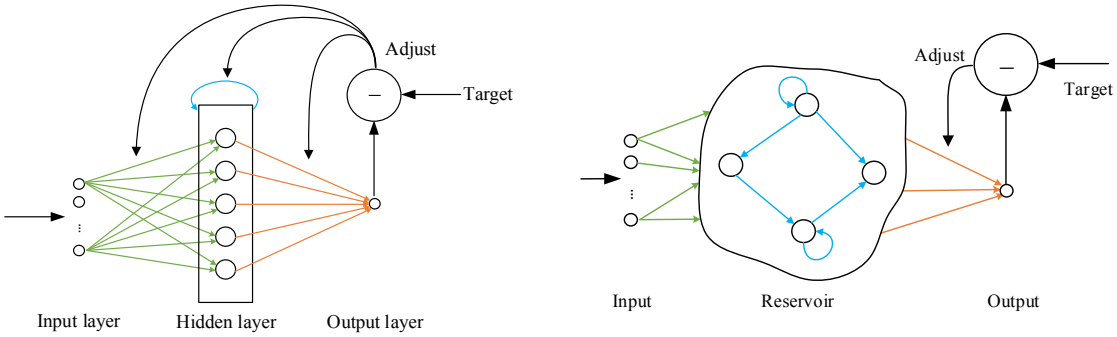
1. Introduction

As a mimic of the human brain, neural-like architectures have been proposed to learn, memorize, steer and game in a complex world [1]; one of the most famous and widely used structures is the feed-forward neural network [2–4]. In recent years, convolutional neural networks (CNNs) [5] have led to impressive results due to their ability to automatically learn complex feature representations using convolutional layers. CNNs exploit translational invariance within their structures by extracting features through receptive fields and learning with weight sharing, becoming the state-of-the-art approach in various image recognition and computer vision tasks.

Another improvement and a more powerful architecture is the recurrent neural network (RNN), which holds at least one cyclic pathway of feedback connections (Figure 1a). RNN mathematically implements dynamical systems and can in theory approximate (universal approximation property) an arbitrary nonlinear dynamical system with arbitrary precision by constantly adjusting the weights of the connections [1]. However, since relational weight-update algorithms are mostly based on gradient-descend, they unavoidably lead to suboptimal solutions [6]. Despite this disadvantage, the difficult of training RNNs also includes slowly convergence, high complexity [6,7] and the most well-known problem: the exploding and vanishing gradients [8].

Recently, a new scheme known as reservoir computing (RC), which greatly helped the practical application of RNNs and exceeded traditional fully trained RNNs in many fields [1], has been proposed

to solve these problems. The echo state network (ESN, Figure 1b) [9] is the most popular example of RC. When an input sequence is fed into the reservoir, a complex non-linear state compute is aroused and then processed by a simple readout procedure to produce the output. By generating and training RNNs without changing the transient dynamics of the recurrent network, only updating weights of connections from the reservoir to the output, RC avoids the conundrums of gradient-based algorithms such as slow and difficult progress. These advantages allow ESN to be applied in numerous domains, including pattern classification [10,11], time-series prediction [12–14], intrusion detection [15], etc.



(**a**) RNN (**b**) ESN

Figure 1. Schematic diagram of RNN and ESN. (**a**) In traditional RNN training, all connection weights including input-to-hidden (green arrows), hidden-internal (blue arrows), and hidden-to-output (orange arrows) weights are adjusted; (**b**) In ESN, only the reservoir-to-output (orange arrows) weights are adapted.

With the rapid development of wireless transmission technology, sensor technology, network communication technology and the speedy growth of smart mobile devices, large amounts of data have been accumulated in almost every aspect of our lives. Moreover, the volume of data is growing rapidly with increasingly complex structures and forms [16]. The Big Data era which is characterized by *volume*, *velocity and variety* has arrived [17], and it leads to the training samples' explosive growth in the machine learning (ML) field, including RNN/ESN in this paper.

In most existing frameworks, the learning and training of RNN/ESN are processed in a centralized way [6,8,12–14,18,19]. However, in the era of Big Data, centralized data storage becomes technologically unsuitable [20]; as a result, solutions relying on centralized training in ML could face challenges such as single point of failure, communication bottlenecks, training consistency, etc. Meanwhile, in multiple real-world applications, large-scale training data is not possible to store in a centralized location, but large amounts of it are dispersed throughout a cluster of interconnected nodes (e.g., wireless sensor networks, peer-to-peer networks) [21–23]. Thus, a decentralized training algorithm for RNN/ESN, at any rate, would be an invaluable tool.

In this paper, we propose a distributed training algorithm and platform for ESN. The optimization algorithm in our platform is the well-known particle swarm optimization (PSO) technique [24], which is good at solving optimization problems in dynamic environments. Further, a parallel PSO (P-PSO) algorithm is presented for distributed ESN. We chose Apache Spark [25] as a training framework, as Spark is an efficient and stretchable cloud platform that is suitable for data mining and machine learning. In the Spark framework, data are cached in memory and iterations for the same data come directly from memory [26]. Upon a stretchable experimental platform, the training time, training error and number of iterations are recorded under different data sizes and platform scales. The results indicate that the distributed ESN trained on Spark performs favorably regarding computing speed, accuracy, and other capabilities.

The contributions of this paper are:

(1) proposing a parallel optimization algorithm based on PSO to handle the distributed training of the ESN,
(2) constructing a distributed and stretchable platform based on Spark to implement the proposed algorithm,
(3) verifying the proposed platform by extremely large-scale datasets that include artificial benchmarks, real-world data and image data.

The rest of this paper is organized as follows. In Section 2, the related works are analyzed, including improvements in ESN, decentralized algorithms, distributed frameworks and research on CNN. The details of the proposed platform are given in Section 3, including algorithms and the calculation framework. Experimental setup and analyses are given in Section 4. Finally, the discussion and conclusion are given in Section 5.

2. Related Work

2.1. Improvements in ESN

Several improvements have been proposed since ESN was brought out by Jaeger [9]. Some adjustments done on the ESN itself include: adding white noise into internal neurons [27], introducing leaky integrator neurons [28], performing ridge regression instead of linear regression [29], clustering the internal neurons [1], etc. Meanwhile, several attempts have been made to combine ESN with other algorithms, such as decision tree [15], principal component analysis (PCA) [14], and the Bayesian method [12], and a hierarchical architecture of ESN was brought forward [30]. These improvements have been applied into numerous scenarios and achieve good results. In this paper, inspired by other achievements, adding white noise and performing ridge regression are adopted for distributed ESN training in order to increase robustness.

2.2. Decentralized Training Algorithm

Several works on the notion of decentralized training algorithms have been explored over the last few years. Vanneschi et al. [31] compared four parallel and distributed PSOs which inspires our work. Zhang et al. [32] improved the PSO and deployed it into distributed spatial precoding. Sheng et al. [33] used the standard PSO and map-reduce (MR) framework for prediction intervals (PIs) construction, which does not implement the parallelization of PSO. For linear regression, Mateos et al. [34] developed the distributed sparse linear regression algorithm to estimate the regression coefficients via Lasso when the training data are distributed across different agents, in which the alternating direction method of multipliers (ADMM) is adopted. Meanwhile, Boyd et al. [35] and Cevher et al. [36] argued that the ADMM is well suited to distributed convex optimization and in particular to large-scale problems, i.e., Big Data. Besides, Chu et al. [37] presented a ADMM-based distributed algorithm for fitting generalized additive models. For feed-forward neural networks, Dean et al. [38] showed another method; they developed two algorithms for large-scale distributed training, Downpour SGD (stochastic gradient descent) and Sandblaster, which are L-BFGS–based (limited-memory Broyden–Fletcher–Goldfarb–Shanno–based) and used for deep learning. On the other hand, the consistent problem should be considered simultaneously in decentralized algorithms. The authors of References [22,39] believe the decentralized average consensus (DAC) method should be an acceptable choice for the consistent problem in decentralized training, and propose an algorithm based on the use of DAC, an extremely efficient procedure for computing global averages on a network which starts from local measurement vectors.

Recent works show that PSO holds the advantages of faster computing speed, good global search capability, adaptive capability, and the P-PSO is particularly suitable for large-scale mathematical optimization problems, and thus it is used in this paper.

2.3. Distributed Framework

Previous literature indicates that Matlab is the preferred framework to test and verify new proposed algorithms [22,23,34,35,39]. On the other hand, in the Big Data era, several new processing frameworks have been proposed. Map-Reduce [40]/Bigtable [41], which were put forward by Google, have been applied to run gradient-based optimization [42] and PSO-based optimization [33]. Map-Reduce is highly successful in applying large-scale data-intensive applications on common-server–established clusters. However, its model is built according to an acyclic data flow scheme. Actually, Spark [25] is a framework that supports iterations which need to reuse a working set of data across multiple parallels, while retaining the scalability and fault tolerance of Map-Reduce [43]. Therefore, Spark is suitable for P-PSO training, as it is multi-iteration and multiple-parallel. In fact, Spark is already widely used in intruder detection in heterogeneous wireless sensor networks (WSN) [44], hospital queuing recommendation [26], etc., and it is accepted in this work.

2.4. Researches on CNN

Traditional CNN usually includes two parts [5]. One is a feature extractor, which learns features from raw data automatically. The other is a trainable, fully connected multilayer perceptron (MLP), which performs classification based on the learned features from the previous part. Generally, the feature extractor is composed of multiple similar stages, and each stage is made up of three cascading layers: the filter layer, the activation layer and the pooling layer [45].

CNN is the state-of-the-art approach in image recognition and computer vision tasks, since images or spectral representations of speech have a strong 2D structure [46], which perfectly suits the convolutional processing in CNN. However, several CNN achievements on strong 1D structures (i.e., time-series) have been brought out. Cui et al. [47] and Zheng, Liu, Chen, Ge and Zhao [45] propose two different time-series classification methods using a multi-scale convolutional neural network (MCNN) and a multi-channel deep convolutional neural network (MCDCNN), respectively. Wang and Oates [48] encode time series data as different types of images, which enables the use of techniques from computer vision for classification. Other research fields such as time-series prediction [49,50] and sound recognition [51,52] are also being studied. In this paper, the performance between distributed ESN and CNN will be illustrated.

3. Algorithm and Platform

3.1. Basic Algorithms

First, the notation of this paper is described in Table 1.

Table 1. Notation of this paper.

Variable	Implication
$\mathbf{x}$	Input vector
N_i	Dimension of input vector
$\mathbf{h}$	Internal state of reservoir
N_r	Dimension of internal state
$\mathbf{y}$	Output vector
N_o	Dimension of output vector
$\mathbf{W}_i^r$	Weights of connections from input layer to reservoir
$\mathbf{W}_r^r$	Weights of recurrent reservoir
$\mathbf{W}_o^r$	Feedback weights from output layer to reservoir
$f_{res}(\cdot)$	Function to compute next internal state
$\mathbf{W}_i^o$	Weights of connections from input layer to output layer
$\mathbf{W}_r^o$	Weights of connections from reservoir to output layer
$f_{out}(\cdot)$	Function to compute output vector
$\rho(\cdot)$	Spectral radius (of $\mathbf{W}_r^r$)
P	Number of desired input-outputs pairs
$\mathbf{d}$	Desired output

Table 1. *Cont.*

Variable	Implication
$\mathbf{H}$	Combination of input and internal state
$\mathbf{w} = [\mathbf{W}_i^o, \mathbf{W}_r^o]^T$	Weights need to be calculated, include weights from both input layer and reservoir to output layer
L	Number of nodes
$f_{fit}(\cdot)$	The fitness function of a particle
$\mathbf{P}_i$	The optimal solution of the i-th particle
$\mathbf{P}_g$	The optimal solution of current moment
$\mathbf{P}_u$	The global optimal solution of whole population
c_1, c_2, c_3	Acceleration coefficient
r_1, r_2, r_3	Random numbers
λ_n, λ_p	Impact factors

A depiction of the typical ESN structure has been shown in Figure 1b, while Figure 2 shows more details about it. An ESN is divided into three parts, namely an input layer, a recurrent reservoir, and an output layer.

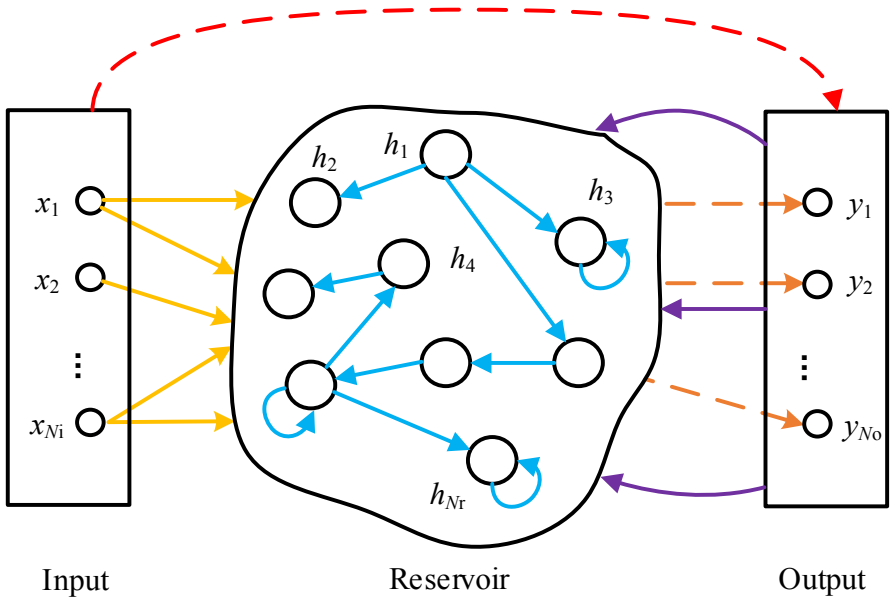


Figure 2. Detail structure of ESN. $\mathbf{W}_i^r$ in yellow line, $\mathbf{W}_r^r$ in blue line, $\mathbf{W}_o^r$ in purple line, $\mathbf{W}_i^o$ in red line, $\mathbf{W}_r^o$ in orange line. Also, trainable connections are shown with dashed lines and random connections are shown with solid lines.

The current output of the ESN could be computed through two phases:

(1) The input vector $\mathbf{x}[n] \in \mathbb{R}^{N_i}$ with N_i-dimensions is fed into the reservoir at first. Then the internal state $\mathbf{h}[n] \in \mathbb{R}^{N_r}$ of the reservoir is updated according to Equation (1):

$$\mathbf{h}[n] = f_{res}(\mathbf{W}_i^r \mathbf{x}[n] + \mathbf{W}_r^r \mathbf{h}[n-1] + \mathbf{W}_o^r \mathbf{y}[n-1]) \quad (1)$$

where $\mathbf{W}_i^r \in \mathbb{R}^{N_r \times N_i}$, $\mathbf{W}_r^r \in \mathbb{R}^{N_r \times N_r}$ and $\mathbf{W}_o^r \in \mathbb{R}^{N_r \times N_o}$ are weight matrices, which need to be generated randomly at the first training process, and remain changeless. It is worth noting that the matrix $\mathbf{W}_r^r$ should satisfied $\rho(\mathbf{W}_r^r) < 1$, where $\rho(\cdot)$ means the spectral radius operator. That is because the reservoir must meet the 'echo state property' (ESP) [9], according to the ESN theory. Once an ESN is ESP-satisfied, the effect of an input which has been fed into the reservoir could vanish in a finite number of iterations, and this ESN can approximate any non-linear filter with bounded memory to any given level of accuracy [53]. The $f_{res}(\cdot)$ in Equation (1) is a suitable non-linear function, and it typically uses a sigmoid shape; in this paper we set $f_{res}(\cdot) = \tanh(\cdot)$. Then $\mathbf{y}[n-1] \in \mathbb{R}^{N_o}$ is the

previous output of the ESN. To increase stability, as mentioned in Section 2.1, before computing the transformation $f_{res}(\cdot)$, we add a small uniform noise term to the state update.

(2) The ESN's current output is computed by:

$$\mathbf{y}[n] = f_{out}(\mathbf{W}_i^o \mathbf{x}[n] + \mathbf{W}_r^o \mathbf{h}[n]) \tag{2}$$

where $\mathbf{W}_i^o \in \mathbb{R}^{N_o \times N_i}$ and $\mathbf{W}_r^o \in \mathbb{R}^{N_o \times N_r}$ are objects to be optimized and determined by the training datasets, and $f_{out}(\cdot)$ in Equation (2) is an invertible non-linear function.

To optimize the weight matrices $\mathbf{W}_i^o$ and $\mathbf{W}_r^o$ in the output layer, suppose a sequence of P desired input-output pairs are given by:

$$(\mathbf{x}[1], \mathbf{d}[1]), \cdots, (\mathbf{x}[P], \mathbf{d}[P]) \tag{3}$$

At the first phase of training, after the P inputs are fed to the reservoir, a sequence of internal states $\mathbf{h}[1], \cdots, \mathbf{h}[P]$ is generated according to Equation (1). Since the outputs of the ESN are not available for feedback, we use the desired output $\mathbf{d}[1], \cdots, \mathbf{d}[P]$ to replace $\mathbf{y}$ in Equation (2) in the training process. The input and internal states are stacked in a matrix $\mathbf{H} \in \mathbb{R}^{P \times (N_i + N_r)}$ and the desired outputs in a vector $\mathbf{d} \in \mathbb{R}^{P \times N_o}$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{x}^T[1], \mathbf{h}^T[1] \\ \vdots \\ \mathbf{x}^T[P], \mathbf{h}^T[P] \end{bmatrix} \tag{4}$$

$$\mathbf{d} = \begin{bmatrix} f_{out}^{-1}(\mathbf{d}[1]) \\ \vdots \\ f_{out}^{-1}(\mathbf{d}[P] \end{bmatrix} \tag{5}$$

Based on this, the training issue of the weight matrices $\mathbf{W}_i^o$ and $\mathbf{W}_r^o$ turns out to be a standard linear regression, which can be solved easily, i.e., by solving the following regularized least-square problem:

$$\mathbf{w}^* = \underset{\mathbf{w} \in \mathbb{R}^{N_i + N_r}}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{H}\mathbf{w} - \mathbf{d}\|_2^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \tag{6}$$

where $\mathbf{w} = \left[\mathbf{W}_i^o, \mathbf{W}_r^o\right]^T$ and $\lambda \in \mathbb{R}^+$ is the regularization factor. Additionally, the first D rows from Equations (4) and (5) should be discarded when solving Equation (6), since they refer to a transient phase in the ESN's behavior, also denoted as the dropout elements [39].

3.2. Distributed Algorithm

In actual practices, the training dataset is often distributed, e.g., measurements collected by multiple sensors in a WSN (wireless sensor networks). Especially in the Big Data era, centralized data storage becomes technologically unsuitable. Under this circumstance, some reforms for distributed processing are necessary.

For a classic PSO, a particle i can be represented by a position vector $\mathbf{x}_i$ and a velocity vector $\mathbf{v}_i$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \cdots, x_{iD}]$, $\mathbf{v}_i = [v_{i1}, v_{i2}, \cdots, v_{iD}]$ and $i = 1, 2, \cdots, M$. M is the number of particles in a population and D is the number of decision variables. Each particle in the population should update its own velocity and position according to Equation (7) during the evolution.

$$\begin{cases} \mathbf{v}_i(t+1) = \omega \mathbf{v}_i(t) + c_1 r_1 (\mathbf{P}_i - \mathbf{x}_i(t)) + c_2 r_2 (\mathbf{P}_g - \mathbf{x}_i(t)) \\ \mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \end{cases} \tag{7}$$

where t is the iteration time, $\omega \geq 0$ is the inertia weight coefficient, $c_1, c_2 \geq 0$ is the acceleration coefficient, r_1 and r_2 are uniformly distributed random numbers in the closed interval $[0, 1]$, $\mathbf{P}_i$ is the local optimal solution of the i-th particle and $\mathbf{P}_g$ is the global optimal solution.

In this paper, a P-PSO algorithm is presented to solve the distributed training problem of the ESN. Since the training dataset S is dispersed among a cluster of L nodes, the idea of the P-PSO is to redevelop the classic PSO by introducing particles for every node, and forcing them to fly synchronously, and reaching a global optimal solution at convergence. Suppose that all nodes in the cluster have accepted a uniform choice of the fixed matrices $\mathbf{W}_i^r$, $\mathbf{W}_r^r$ and $\mathbf{W}_o^r$. Denoted by $\mathbf{H}_k$ and $\mathbf{d}_k$, the hidden matrices and output vectors can be precomputed at the k-th node according to Equations (4) and (5), with their own local dataset, and stored into memory, with respect to the training complexity, where $k = 1, 2, \cdots, L$. The same works with a classic PSO, where $\mathbf{x}_i$, $\mathbf{v}_i$ are the position and velocity of the i-th particle, respectively. The fitness function $f_{fit}(\cdot)$ of a particle $\mathbf{x}_i$ is defined as Equation (8):

$$\begin{aligned} f(\mathbf{x}_i^k) &= \tfrac{1}{2}\left(\|\mathbf{H}_k\mathbf{x}_i - \mathbf{d}_k\|_2^2\right) \\ f_{fit}(\mathbf{x}_i) &= \sum_{k=1}^{L} f(\mathbf{x}_i^k) \end{aligned} \tag{8}$$

where $f(\mathbf{x}_i^k)$ is the fitness value of particle $\mathbf{x}_i$ in the k-th node.

Besides, define $\mathbf{P}_i$ as the optimal solution of the i-th particle and $\mathbf{P}_g$ as the optimal solution of the current moment. As the center of the population, $\mathbf{P}_g$ is relatively variable, which makes the whole particle group more brisk, and a larger area will be explored. $\mathbf{P}_i$ and $\mathbf{P}_g$ are determined by Equations (9) and (10):

$$\mathbf{P}_i = \underset{\mathbf{x}_i(t)}{\operatorname{argmin}}\{f_{fit}(\mathbf{x}_i(t)) | t = 1, 2, \cdots, current\ iteration\} \tag{9}$$

$$\mathbf{P}_g = \underset{\mathbf{x}_i(t)}{\operatorname{argmin}}\{f_{fit}(\mathbf{x}_i(t)) | t = current\ iteration, i = 1, 2, \cdots, M\} \tag{10}$$

To avoid the local convergence $\mathbf{P}_u$, the global optimal solution of the whole population is necessary, which is defined as Equation (11):

$$\mathbf{P}_u = \underset{\mathbf{x}_i(t)}{\operatorname{argmin}}\{f_{fit}(\mathbf{x}_i(t)) | t = 1, 2, \cdots, current\ iteration, i = 1, 2, \cdots, M\} \tag{11}$$

In this paper, constriction factor χ is adopted to replace the ω in the classic PSO; χ is determined by Equation (12):

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \varphi = c_1 + c_2, \varphi > 4 \tag{12}$$

Particles could update the velocity according to Equation (13):

$$\mathbf{v}_i(t+1) = \chi[\mathbf{v}_i(t) + c_1 r_1(\mathbf{P}_i - \mathbf{x}_i(t)) + c_2 r_2(\mathbf{P}_g - \mathbf{x}_i(t))] + c_3 r_3(\mathbf{P}_u - \mathbf{x}_i(t)) \tag{13}$$

where $c_3 \geq 0$ is the acceleration coefficient of the whole particle population, r_3 is a random number in [0, 1]. Furthermore, in order to allow the particles to search for a larger range at the beginning of the iteration and to converge quickly at a later stage, two factors are introduced into this algorithm, λ_n and λ_p. Concretely, updating the equation for velocity and position, it is modified to:

$$\begin{cases} \mathbf{v}_i(t+1) = \tanh(\frac{\lambda_n}{t}) \cdot \chi[\mathbf{v}_i(t) + c_1 r_1(\mathbf{P}_i - \mathbf{x}_i(t)) + c_2 r_2(\mathbf{P}_g - \mathbf{x}_i(t))] + \tanh(\frac{t}{\lambda_p}) \cdot c_3 r_3(\mathbf{P}_u - \mathbf{x}_i(t)) \\ \mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \end{cases} \tag{14}$$

Obviously, in Equation (14), when the iteration begins, t is rather small, $\tanh(\frac{\lambda_n}{t})$ approaches 1 and $\tanh(\frac{t}{\lambda_p})$ approaches 0, which leads to a relatively large influence of $\mathbf{P}_g$, and $\mathbf{P}_u$ is lightweight, which benefits the exploration of the whole particle population. Later, when t becomes large, the opposite situation occurs, so that the particles accelerate convergence.

P-PSO can be stopped when the global optimal solution $\mathbf{P}_u$ does not change (the difference between two adjacent iterations is lower than the pre-specified threshold, Equation (15)), or a maximum number of iterations is reached.

$$\|\mathbf{P}_u(t) - \mathbf{P}_u(t-1)\|_2^2 < \varepsilon \tag{15}$$

Since $f(\mathbf{x}_i^k)$ in Equation (8) can be computed at every node locally, the parallelization is restricted to the computation of $\mathbf{P}_i$, $\mathbf{P}_g$ and $\mathbf{P}_u$, which are global variables and needed to broadcast. However, they are rather easy to compute by Equations (8)–(11), and thus the overall algorithm can be achieved in a purely decentralized fashion.

3.3. Spark-Based Platform

It is necessary to take a few words for Map-Reduce before introducing Spark. Map-Reduce is the basic open-source, scalable and fault-tolerant performance model for distributed processing. The core of its performance functionality is conducted through two main functions: (1) Map, which is responsible for loading the data into a powerful system and defining a set of keys, uses a key-value pair ($k1$, $v1$) as the input and generates intermediate key-value pairs [($k2$, $v2$)]; and (2) Reduce, which collects the built set of key-based data from the map function and distributes or reduces the whole implementation through a multi-core HPC platform, merges all the key-value pairs ($k2$, [$v2$]) with the same key into output key-value pairs [($k3$, $v3$)].

Map-Reduce has many drawbacks, especially in disk I/O processing and re-loading of data at each Map-Reduce processing function. Thus, it is very difficult to develop an appropriate iterative-computational algorithm for reliable and efficient predictive modeling within the Map-Reduce system. Distinct from the Map-Reduce model on the Hadoop platform, Spark is an in-memory computing solution to reduce disk I/O and also an efficient and stable solver of large-scale datasets with both iterative least squares and maximum likelihood optimizations. The intermediate results generated in the training process of Spark are stored in the memory system on the Spark framework as RDD (resilient distributed datasets) objects, which is the most important highlight in Spark. In the RDD programming paradigm, each RDD object supports two types of operations, *transformation* and *action*, concretely. Within the whole computation progress, *transformations* just modify RDDs, and only *actions* trigger task submission and then calculate the result. *Transformation* operations include several operations, such as *map*(), *flatMap*(), *filter*(), *mapPartitions*(), *union*(), and *join*(). Then, a new RDD object is produced from each *transformation* operation. *Action* operations include operations such as *reduce*(), *collect*(), *count*(), *saveAsHadoopFile*(), and *countBykey*(), which compute a result and call back to the driver program or save it to an external storage system [26].

Although the in-memory computations of Spark reduce the disk I/O, developing an efficient and stable machine learning algorithm on Spark is difficult and not straightforward. Therefore, the motivation in this research is to parallelize the PSO-based ESN algorithm on the Spark framework with the RDD programming model. In this work, the training process of the parallelized PSO-based ESN algorithm includes three stages, which are shown in Figure 3. Stage 1 is for data preprocessing and variable initialization, stage 2 trains the ESN model with several iterations, and then stage 3 calculates errors and saves the results. The L training datasets are trained in a parallel process, and both $\mathbf{W}_i^o$ and $\mathbf{W}_r^o$ are calculated at the same time.

In stage 1, the training data are loaded from HDFS to the memory and stored as a RDD object, namely $RDD_{Original\_data\_set}$, which is defined to save the original data. Then, the *sortByKey*() function is performed to make sure the original data is ordered by time series and evenly distributed to multiple nodes to avoid data skew. In every slave node k, data is split into two parts for training and testing, called RDD_{traink} and RDD_{testk}. Typically, 70% of data is used for training and the rest for testing. Next, we assign matrices $\mathbf{W}_i^r$, $\mathbf{W}_r^r$, $\mathbf{W}_o^r$ and broadcast them to every node, followed by initializing $\mathbf{x}_i(0)$, $\mathbf{v}_i(0)$, $\mathbf{P}_i$, $\mathbf{P}_g$ and $\mathbf{P}_u$.

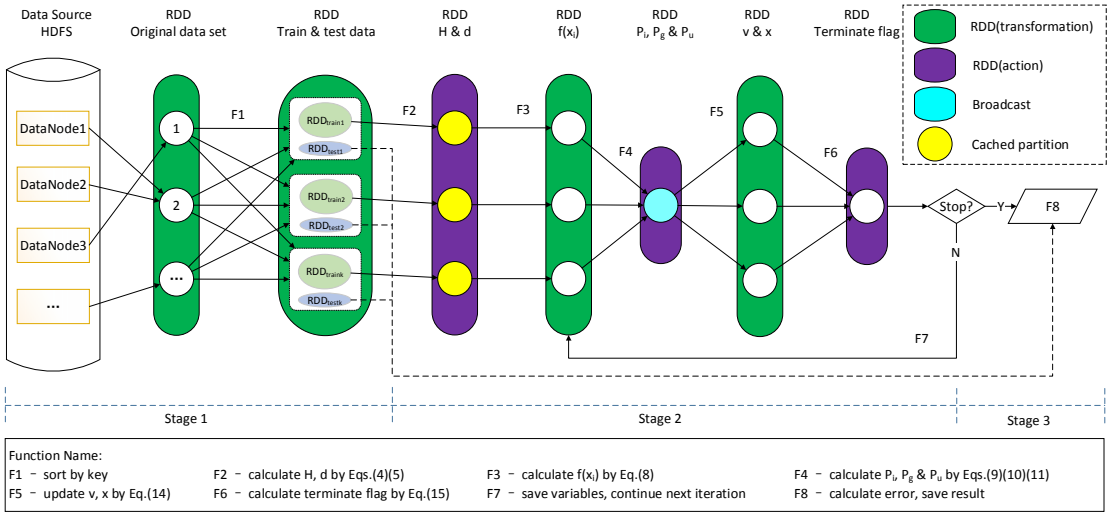


Figure 3. Training processing of parallelized PSO-based ESN algorithm.

In stage 2, with RDD_{traini} and RDD_{testi} for every slave node k, both $\mathbf{W}_i^o$ and $\mathbf{W}_r^o$ are calculated in a parallel process. The training algorithm at k-th node is described in Algorithm 1.

Algorithm 1 Training algorithm for PPSO-based ESN at k-th node

Algorithm Inputs: Training set RDD_{traink}, size of reservoir N_r, factors c_1, c_2, c_3, r_1, r_2, r_3, λ_n and λ_p, maximum iterations number T

Algorithm Output: Optimal output weights $\mathbf{w}^* = \left[\mathbf{W}_i^o, \mathbf{W}_r^o\right]^T$

Compute $\mathbf{H}_k$ and $\mathbf{d}_k$ from RDD_{traink}.
for n from 1 to T do
  Compute $f(\mathbf{x}_i^k)$ according to Equation (8).
  Compute $\mathbf{P}_i$, $\mathbf{P}_g$ and $\mathbf{P}_u$ according to Equations (9)–(11).
  Update $\mathbf{x}_i$ and $\mathbf{v}_i$ according to Equation (14).
  Check terminated criterion by Equation (15).
  if (Equation (15) is satisfied or n equal to T)
    then break;
    else save variables.
  end if
end for
return $\mathbf{P}_u$

In stage 3, we use optimal weight $\mathbf{P}_u$ to calculate the error of the RDD_{testk}, and then save it for future calculation and analysis.

4. Experimental Analysis

4.1. Setup

The platform and experiments were performed on a Spark framework, the network topology of which is shown in Figure 4. The platform contains one master node and several slave nodes, and they share the same software and hardware configuration, as shown in Table 2.

Table 2. Software and hardware configuration.

Hardware	CPU	Xeon E5-2609 v2, 2.5 GHz, 4 cores × 2
	Memory	16 GB
	Network	100 MB/s
Software	Operating System	CentOS-6.5
	Spark	1.6.1
	Hadoop	2.2.0
	ZooKeeper	3.4.5
	Java Development Kit	1.7.0_60

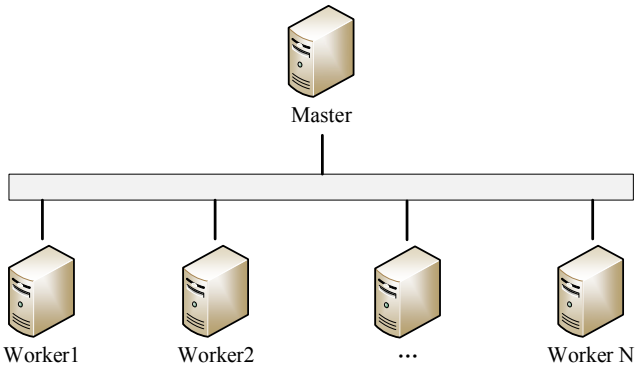


Figure 4. Network topology of our platform. As a stretchable platform, the number of workers is variable on demand.

Readers are referred to [25] for the details of constructing the Spark cluster. In this experiment, the monitoring interface of the Spark platform is shown in Figure 5. The platform has six workers altogether.

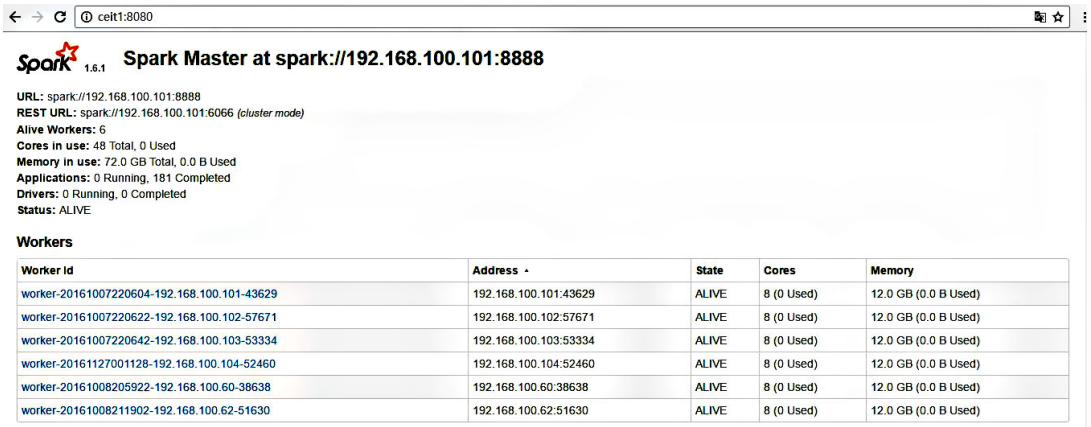


Figure 5. Monitoring interface of our platform.

4.2. Datasets

We test the proposed distributed ESN algorithm on four datasets, which are related to non-linear system identification, time-series prediction and image recognition. Three datasets of four are standard artificial benchmarks, and the other is sensor-monitoring data from the real world. A brief description of the four datasets is provided below.

(1) NARMA-10 dataset. It is a non-linear system identification application, where the input $x[n]$ is white noise distributed in the interval [0, 0.5], and the output $d[n]$ is computed from the following equation [54]:

$$d[n+1] = 0.3d[n] + 0.05d[n]\left[\sum_{i=0}^{9} d[n-i]\right] + 1.5d[n-9]d[n] + 0.1 \tag{16}$$

Then, the output is squashed into $[-1, +1]$ by the $\tanh(\cdot)$ transformation:

$$d[n+1] = \tanh(d[n+1] - \hat{d}) \tag{17}$$

where $\hat{d}$ is the empirical mean of the overall output.

(2) The second dataset is a three-dimensional time-series prediction task based on the Lorenz attractor, defined by the following differential equations [39]:

$$\begin{cases} x_1[n+1] = \sigma(x_2[n] - x_1[n]) \\ x_2[n+1] = x_1[n](\eta - x_3[n]) - x_2[n] \\ x_3[n+1] = x_1[n]x_2[n] - \zeta x_3[n] \end{cases} \tag{18}$$

where $\sigma = 10$, $\eta = 28$ and $\zeta = 8/3$ (the default settings). The task in Equation (18) is sampled every second, and the input vector is $\begin{bmatrix} x_1[n] & x_2[n] & x_3[n] \end{bmatrix}$, while the required output is a one-step-ahead prediction of the variable x_1, namely:

$$d[n] = x_1[n+1] \tag{19}$$

(3) The third dataset is real-time monitoring data of smart meters in an electric power system [55]. Concretely, it is real-world electric data from the circuits of one resident home, and it is sampled every second.

(4) Nowadays, unstructured data such as images and videos are accumulated rapidly in Big Data. It is necessary to inspect the performance of the proposed framework when the input is an image, namely the input is high-dimensional data (supposing each pixel in the image is regarded as a dimension). We choose the Mixed National Institute of Standards and Technology (MNIST) dataset [56], the popular database of handwritten digits, as the fourth dataset.

Additionally, as a standard practice in ESN implementations, we supplemented the original input with an additional constant unitary input to all four datasets [53].

4.3. Parameters

Several parameters must be determined for further analysis. A brief introduction on the parameters' determination is described in this section.

Firstly, we select N_r = 300 as the default reservoir's size, which is a typical set and is found to work well in all situations, and we set D = 100 to discard the first 100 elements on each node.

Secondly, we set $c_1 = c_2 = 1.49445$, $c_3 = 2$ and $\varphi = 4.1$, which could achieve a faster convergence for P-PSO.

Thirdly, since the first two datasets are artificial benchmarks and noiseless, we set small factors λ_n = 50 and λ_p = 100, and for the last two datasets, a relatively larger λ_n = 200 and λ_p = 300 are set since they are collected in the real world and may need more iterations to explore the optimal solution.

Finally, $\mathbf{W}_i^r$, $\mathbf{W}_r^r$ and $\mathbf{W}_o^r$ are randomly initialized with the uniform distribution , as stated in Section 3.1. It is worth noting that $\mathbf{W}_r^r$ is encouraged to be sparse, and typically only 25% of its connections are non-zero. Furthermore, $\rho(\mathbf{W}_r^r)$ should be less than one.

4.4. Analysis

In our experiment, we tested the performance of all the datasets under both different data scales and different cluster scales; every situation was repeated 10 times. Concretely, each validation dataset had a length of $\{1 \times 10^4, 3 \times 10^4, 1 \times 10^5, 3 \times 10^5, 1 \times 10^6, 3 \times 10^6, 1 \times 10^7\}$ (due to the limit of real collected data, the longest length for dataset 3 would be 6.7×10^6, and that for dataset 4 would be 7.0×10^4). The number of workers in this platform ranges from two to six. Actually, since each worker contains eight CPUs (shown in Table 2), the number of nodes L could range from 16 to 48.

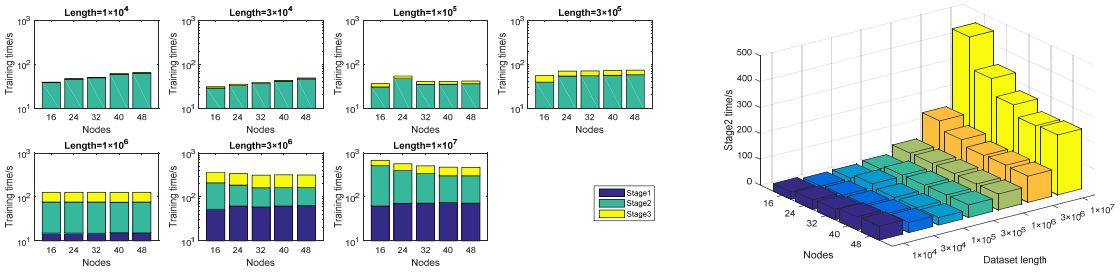
The training time, testing error and number of iterations are selected to evaluate the performance of the proposed algorithm and framework. We also evaluate the performance of the other three algorithms on these four datasets, including: (1) traditional PSO in centralized training mode; (2) the ADMM-based decentralized ESN training algorithm, which is implemented by Matlab [39]; (3) CNN, which uses a centralized training fashion. All centralized algorithms (PSO and CNN) are running on a single machine which shares the same software and hardware configuration with the mentioned cluster (in Table 2). Several essential configurations of the three algorithms are shown in Table 3.

Table 3. Software and hardware configuration.

Algorithm	Configuration
PSO	Initiate range of particle: [−0.5 0.5] Lower bounds: −1 Upper bounds: 1 Generations: 500 PopulationSize: 80
ADMM	Iterations of ADMM: 1000 Regularization parameter of ADMM (rho): 0.01 Relative tolerance value: 10^{-4} Absolute tolerance value: 10^{-4}
CNN	Network structuer: input—6 convolution (kernel 5)—2 sub sampling—12 convolution (kernel 5)—2 sub sampling—output

4.4.1. Training Time

We recorded the training time of the four datasets in different lengths and nodes. For convenience, the logarithmic scale was used in the total training time presentation. In the training process (see Figure 3), the iterations process is much more critical, so the time spent on this stage (stage 2) is illustrated strikingly. Figure 6 shows the training time in detail.



(**a1**) (**b1**)

Figure 6. *Cont.*

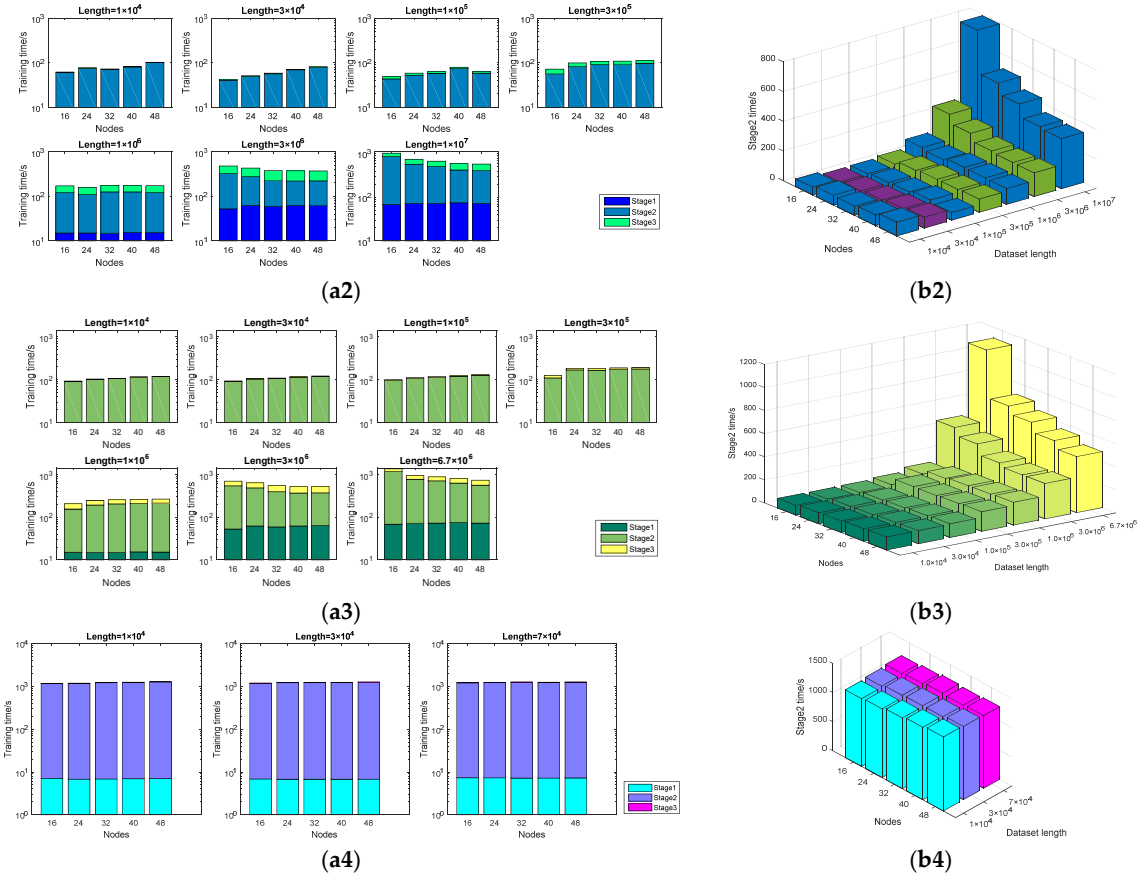


Figure 6. Training time of four datasets from top to bottom. (**a1–a4**) Training time of each dataset in different lengths and nodes, which is stacked by three stages. Logarithmic scale is used for convenience. (**b1–b4**) Time spent on iterations (stage 2).

Several points can be concluded from Figure 6:

1. For a fixed node number, observed by the length of the dataset, we will find the training time becomes a bit shorter at first (1×10^4~1×10^5), and then increases rapidly (>3×10^5) along with the larger size of the dataset.
2. Observed by the number of nodes, when the dataset is big (1×10^7), the training time decreases when the nodes become larger, and the training time is more stable; however, a relatively small dataset (1×10^4) shows the opposite trend. Figure 7 shows some details about this issue.
3. More time will be spent on reading data, especially when dataset is huge (>3×10^6).
4. In Stage 2 (iteration stage), in small datasets, with the increase of nodes, the time increases slowly. In large datasets, more nodes may save a lot of time. Nevertheless, the effect is gradually fading.

The relations of the training time between small and large datasets are interesting, but the reason for it is trivial. In a small dataset, i.e., 1×10^4, with 16 nodes, each node could get 625 records on average. However, it can only get 208 records when 48 nodes are tested. Even discarded elements are not taken into consideration, as the samples are still too few to train. This leads to a relatively longer time convergence. When the dataset is a bit larger, say 1×10^5, every node can get 2000~6000 records, and sufficient samples are provided. So the training time becomes shorter. When the dataset continues to be large, the calculation burden becomes gradually heavy, and more time would be spent, especially in the iteration stage. At this moment, more nodes may help significantly.

Table 4 shows the comparative results of the training time between different algorithms on four datasets, with a typical length of data size of (1×10^5). With all four datasets, distributed

training algorithms showed greater advantages than the centralized ones, shortening the training time efficiently. Especially in former three datasets, the P-PSO-based algorithm could save more than 90% of the training time. In dataset 4, since the dimension of the input data increases significantly, the huge amount of computation makes the training time of the distributed algorithm increase. Nevertheless, the proposed P-PSO~based ESN algorithm is the most efficient algorithm. Additionally, it should be noted that this result is based on a cluster (six nodes) and not on a single machine.

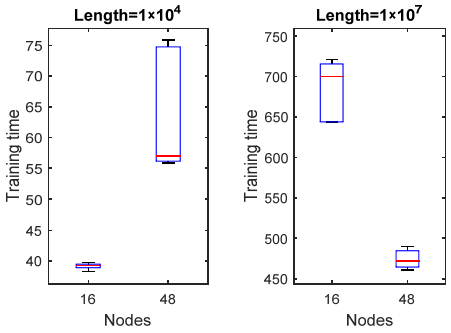


Figure 7. Training time distribution of dataset 1 in lengths of 1×10^4, 1×10^7 and in nodes of 16, 48. In small datasets, less nodes take less time, and the time distribution is stable. Large datasets show the opposite result.

Table 4. Comparative results of training time.

Dataset 1	#node	16	24	32	40	48
	PSO			39,226.026		
	CNN			24,321.529		
	ADMM	300.386	415.349	549.545	686.230	875.851
	P-PSO	36.085	54.094	40.524	40.690	41.611
	P-PSO (Stage 2)	23.180	41.115	27.709	27.851	28.807
Dataset 2	#node	16	24	32	40	48
	PSO			30,630.443		
	CNN			27,456.923		
	ADMM	411.361	528.697	668.287	836.234	1091.287
	P-PSO	49.181	58.554	63.622	79.493	64.106
	P-PSO (Stage 2)	36.277	45.575	50.807	66.654	51.302
Dataset 3	#node	16	24	32	40	48
	PSO			26,324.124		
	CNN			31,259.718		
	ADMM	266.858	343.332	457.197	669.109	706.769
	P-PSO	100.770	112.695	118.022	125.488	132.211
	P-PSO (Stage 2)	87.866	99.717	105.208	112.649	119.408
Dataset 4	#node	16	24	32	40	48
	PSO			9521.099		
	CNN			11,865.469		
	ADMM	2613.941	3692.024	4837.119	5567.593	6669.910
	P-PSO	1230.089	1248.736	1274.183	1259.708	1270.916
	P-PSO (Stage 2)	1207.184	1225.758	1251.368	1236.869	1248.113

4.4.2. Testing Error

In our work, we selected NRMSE (normalized root mean-squared error) to measure the accuracy of the proposed framework, which is computed by:

$$NRMSE = \sqrt{\frac{\sum_{i=1}^{K} (y_i - d_i)^2}{K\sigma_d}} \tag{20}$$

where K is the number of datasets (excluding dropout elements), y_i is the predicted values, d_i is the corresponding true values, σ_d is the variance of d.

As stated above, we tested each situation 10 times. Figure 8 shows the NRMSE of the former three datasets and the accuracy of the last dataset in box graphs.

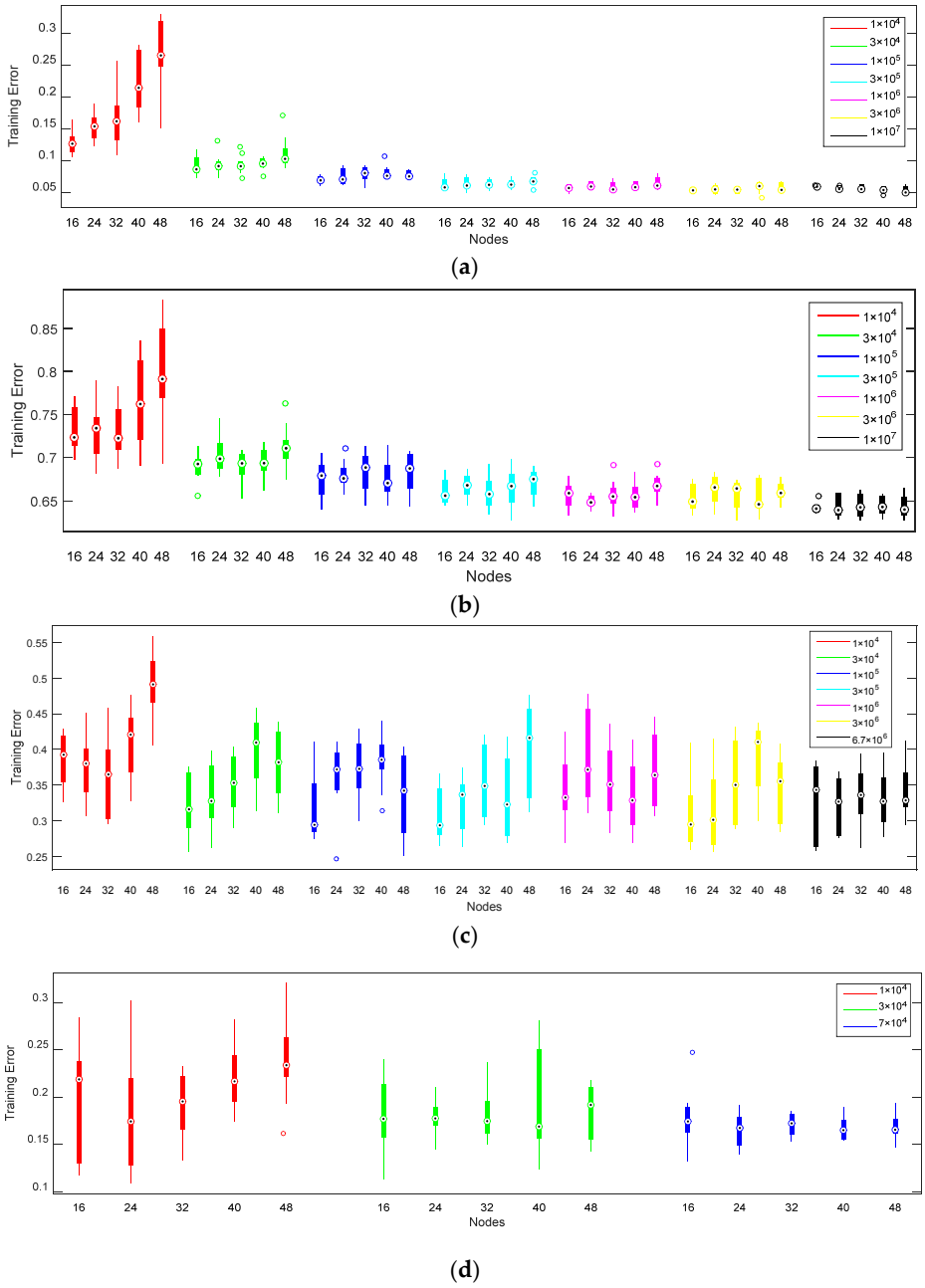


Figure 8. NRMSE of each dataset in different lengths and nodes. (**a**) NRMSE of dataset 1; (**b**) NRMSE of dataset 2; (**c**) NRMSE of dataset 3; (**d**) Accuracy of dataset 4.

When the dataset is small, the training error increases with the number of nodes. For large datasets, the training error is relatively stable. That is because different sample sizes may affect the training error. This is obviously the case with the length of 1×10^4.

With artificial datasets (Dataset 1 and 2), the training error of our framework is stable, while in real-world datasets (Dataset 3), the training error becomes flickering, which means that features in the real world are more difficult to obtain than artificial data.

Furthermore, the correlation coefficient is used to measure the accuracy of the proposed algorithm on former three datasets. Figure 9 illustrates the results in detail. All three datasets showed that the larger the data size, the higher the correlation coefficient, which indicates a better result. It also confirms the above conclusions from the NRMSE.

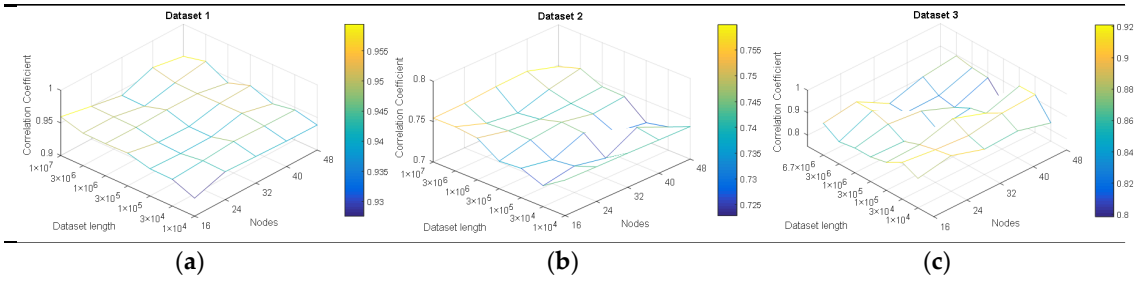


Figure 9. Correlation coefficient on former three datasets. (**a**) Dataset 1; (**b**) Dataset 2; (**c**) Dataset 3.

We compared the testing error between the traditional PSO, ADMM, CNN and P-PSO~based algorithms in this paper, with a typical length of data size of (1×10^5); the result is shown in Figure 10.

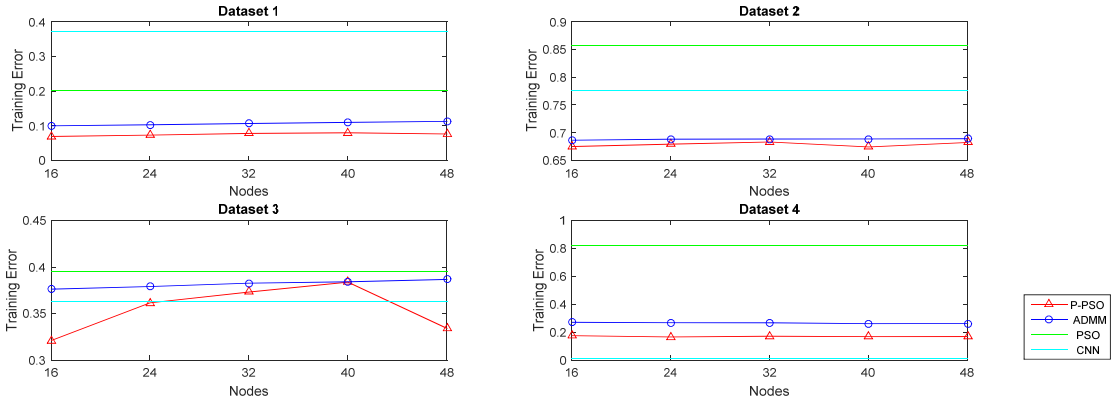


Figure 10. Training error comparison between traditional PSO, ADMM, CNN and P-PSO~based distributed ESN algorithms. A typical length of data size (1×10^5) is selected for the four datasets.

As shown in Figure 10, for artificial datasets (datasets 1 and 2), the P-PSO~based optimization algorithm is clearly outperforming the other three algorithms, and it even performs well in the real-world dataset (dataset 3). However, when images (high-dimension data) are treated, CNN shows an excellent score (1.2%). The comparison between the four algorithms indicates that CNN is good at two-dimensional image recognition, the P-PSO~based ESN algorithm has a better ability to handle time-series data, and ADMM is slightly inferior; the traditional PSO is the most unsteady algorithm.

4.4.3. Number of Iterations

The number of iterations is an important indicator of an algorithm. Figure 11 shows the number of iterations of the four datasets under different situations.

Compared with Figure 6, we can see that the iteration number of the small datasets is positively correlated to the training time. A small dataset needs more iterations. For large datasets, the change of the iteration number is relatively stable. Besides, Figure 11c shows more confusion than Figure 11a. Real-world data and image data need more iterations than other artificial data, which is because more noise and uncertainty are embodied in real-world data (e.g., dataset 3); in dataset 4, the dimensions that had to be optimized were much higher, thus leading to a relatively large iteration number.

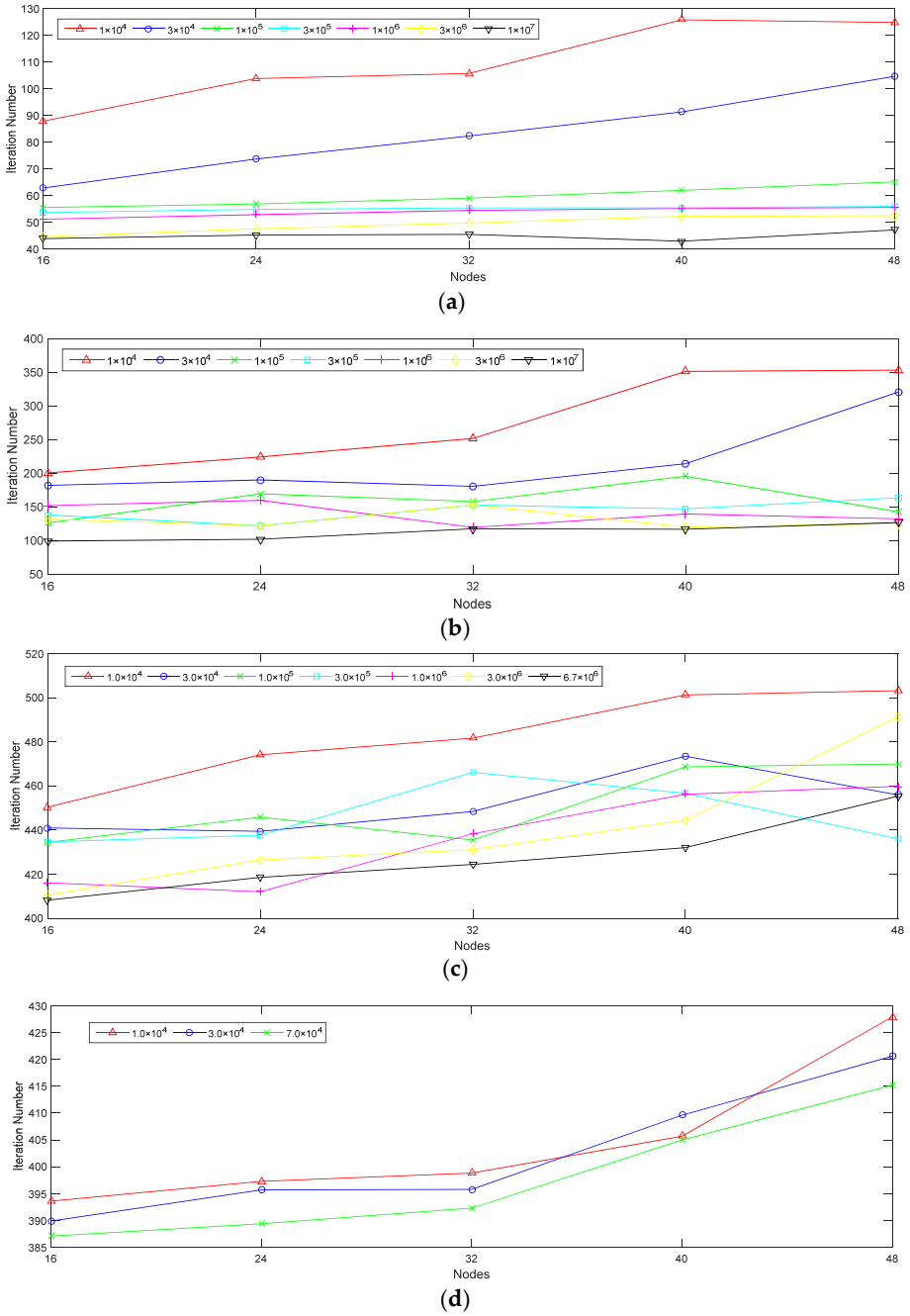


Figure 11. Number of iterations of each dataset in different lengths and nodes. (**a**) Number of iterations of dataset 1; (**b**) Number of iterations of dataset 2; (**c**) Number of iterations of dataset 3; (**d**) Number of iterations of dataset 4.

We chose datasets 1 and 4 to compare the iteration numbers between the four different algorithms, since dataset 1 represents time-series data and 4 represents image data. Figure 12 presents the comparison.

The most significant difference between dataset 1 and 4 is the calculation amount (for ADMM and CNN), and the search range (for PSO and P-PSO). When the dimension of the input data is small (e.g., dataset 1), P-PSO and ADMM can quickly converge, and the traditional PSO needs considerably more iterations. However, once a huge dimension of inputs is fed, the calculation amount/search range would increase sharply, and both P-PSO and ADMM need more iterations. Due to the special network architecture, CNN could converge along with a relatively small iteration number and obtain a

rather good achievement. Even though the traditional PSO uses the least iteration numbers, the testing error is quite high (82.2%, Figure 10). The reason is that traditional PSO is not able to handle the search task within such a high-dimensional space, and falls into suboptimal solutions easily.

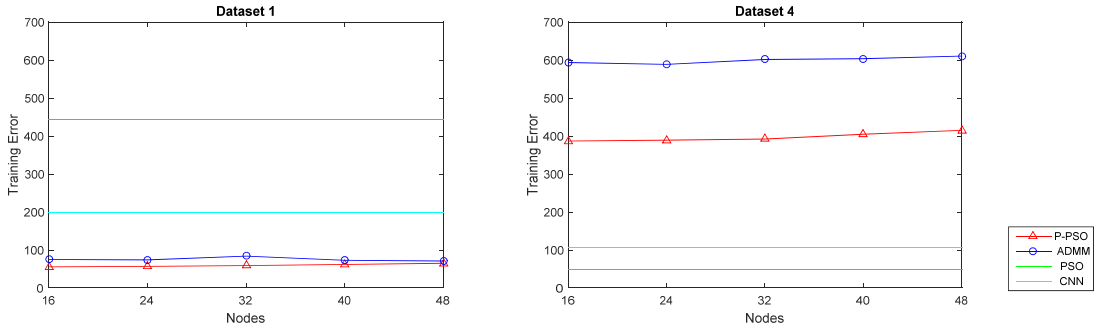


Figure 12. Training error comparison between traditional PSO, ADMM, CNN and P-PSO~based distributed ESN algorithms. A typical length of data size (1×10^5) is selected for the four datasets.

5. Discussion and Conclusions

This paper provided a distributed training algorithm for ESN, a type of RNN, and also a widely used algorithm of RC. The distributed optimization method is based on P-PSO, and was implemented on the Spark framework. For applications with huge amounts of data distributed and stored, the traditional centralized training mode may be ineffectual; our proposed platform, however, could handle these issues effortlessly.

We tested our algorithm and platform using artificial datasets, a real-world dataset and an image dataset, with several different volumes and different training nodes. (1) For small datasets ($<3 \times 10^4$), a relatively low number of nodes would obtain a good result, regardless of the training time or training error. (2) For large datasets (1×10^5~1×10^6), a big node number appears more advantages for the training time. For huge datasets ($>3 \times 10^6$), the biggest node number is best. However, the marginal efficiency of increasing nodes becomes lower. (3) In situations of 48 nodes, when the training dataset grows 10^3 times (from 1×10^4 to 1×10^7), the training time of our framework only increases 7.4 times. (4) Compared to traditional PSO, ADMM, CNN and P-PSO algorithms, the performance of our proposed algorithm was excellent in most cases, apart from the accuracy of image recognition being a bit lower than in CNN, due to its unique network architecture.

The test results show the good performance of this proposed algorithm and platform, while ensuring the training accuracy, and the training time spent is also very satisfactory, which allows this platform to be widely applied, especially for extremely large-scale datasets in the Big Data era.

Nevertheless, several issues are still worth investigating further. For real-world data, the stability of the training results is not as good as that for artificial datasets. The internal mechanism is still valuable to explore. Meanwhile, it was noticed that for huge datasets, the time spent on reading data (Stage 1) was obviously long. Besides, the ability to handle image data (or high-dimensional data) needs to be strengthened. We will focus on the above questions in future research.

Acknowledgments: This work is supported by the Fundamental Research Funds for Central Universities (No. 2015XS72).

Author Contributions: K.W. and Y.Z. conceived and designed the experiments; G.H. performed the experiments; Y.Z. and Q.L. analyzed the data; G.H. contributed analysis tools; Y.Z. wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Najibi, E.; Rostami, H. Scesn, spesn, swesn: Three recurrent neural echo state networks with clustered reservoirs for prediction of nonlinear and chaotic time series. *Appl. Intell.* **2015**, *43*, 460–472. [CrossRef]
2. Chen, D.; Burrell, P. On the optimal structure design of multilayer feedforward neural networks for pattern recognition. *Int. J. Pattern Recognit. Artif. Intell.* **2002**, *16*, 375–398. [CrossRef]
3. Van Der Smagt, P.P. Minimisation methods for training feedforward neural networks. *Neural Netw.* **1994**, *7*, 1–11. [CrossRef]
4. Wong, K.P. Artificial Intelligence and Neural Network Applications in Power Systems. In Proceedings of the 2nd International Conference on Advances in Power System Control, Operation & Management, Hong Kong, China, 7–10 December 1993; pp. 37–46.
5. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
6. Holzmann, G.; Hauser, H. Echo state networks with filter neurons and a delay&sum readout. *Neural Netw. Off. J. Int. Neural Netw. Soc.* **2010**, *23*, 244–256.
7. Luo, X.; Li, J.; Sun, Z.-Q. Review on echo state networks. *J. Univ. Sci. Technol. Beijing* **2012**, *34*, 217–222.
8. Pascanu, R.; Mikolov, T.; Bengio, Y. On the difficulty of training recurrent neural networks. In Proceedings of the 30th International Conference on Machine Learning (ICML 2013), Atlanta, GA, USA, 16–21 June 2013; International Machine Learning Society (IMLS): Atlanta, GA, USA, 2013; pp. 2347–2355.
9. Jaeger, H. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn* **2001**, *148*, 13.
10. Skowronski, M.D.; Harris, J.G. Minimum mean squared error time series classification using an echo state network prediction model. In Proceedings of the ISCAS 2006: 2006 IEEE International Symposium on Circuits and Systems, Kos, Greece, 21–24 May 2006; Institute of Electrical and Electronics Engineers Inc.: Kos, Greece, 2006; pp. 3153–3156.
11. Yusoff, M.-H.; Chrol-Cannon, J.; Jin, Y. Modeling neural plasticity in echo state networks for classification and regression. *Inf. Sci.* **2016**, *364–365*, 184–196. [CrossRef]
12. Li, D.; Min, H.; Wang, J. Chaotic time series prediction based on a novel robust echo state network. *IEEE Trans. Neural Netw. Learn. Syst.* **2012**, *23*, 787–799. [CrossRef] [PubMed]
13. Crisostomi, E.; Gallicchio, C.; Micheli, A.; Raugi, M.; Tucci, M. Prediction of the italian electricity price for smart grid applications. *Neurocomputing* **2015**, *170*, 286–295. [CrossRef]
14. Bianchi, F.M.; De Santis, E.; Rizzi, A.; Sadeghian, A. Short-term electric load forecasting using echo state networks and pca decomposition. *Access IEEE* **2015**, *3*, 1931–1943. [CrossRef]
15. Ding, H.Y.; Pei, W.J.; He, Z.Y. A multiple objective optimization based echo state network tree and application to intrusion detection. In Proceedings of the IEEE International Workshop on Vlsi Design and Video Technology, Suzhou, China, 28–30 May 2005; pp. 443–446.
16. Zhou, K.; Fu, C.; Yang, S. Big data driven smart energy management: From big data to big insights. *Renew. Sustain. Energy Rev.* **2016**, *56*, 215–225. [CrossRef]
17. Chen, J.; Chen, Y.; Du, X.; Li, C.; Lu, J.; Zhao, S.; Zhou, X. Big data challenge: A data management perspective. *Front. Comput. Sci.* **2013**, *7*, 157–164. [CrossRef]
18. Fagiani, M.; Squartini, S.; Gabrielli, L.; Spinsante, S.; Piazza, F. A review of datasets and load forecasting techniques for smart natural gas and water grids: Analysis and experiments. *Neurocomputing* **2015**, *170*, 448–465. [CrossRef]
19. Wang, T.-D.; Wu, X.; Fyfe, C. Factors important for good visualisation of time series. *Int. J. Comput. Sci. Eng.* **2016**, *12*, 17–28. [CrossRef]
20. Anagnostopoulos, I.; Zeadally, S.; Exposito, E. Handling big data: Research challenges and future directions. *J. Supercomput.* **2016**, *72*, 1494–1516. [CrossRef]
21. Predd, J.B.; Kulkarni, S.B.; Poor, H.V. Distributed learning in wireless sensor networks. *IEEE Signal Process. Mag.* **2006**, *23*, 56–69. [CrossRef]
22. Scardapane, S.; Fierimonte, R.; Wang, D.; Panella, M.; Uncini, A. Distributed music classification using random vector functional-link nets. In Proceedings of the International Joint Conference on Neural Networks (IJCNN 2015), Killarney, Ireland, 12–17 July 2015; Institute of Electrical and Electronics Engineers Inc.: Killarney, Ireland, 2015.

23. Vandoorne, K.; Dambre, J.; Verstraeten, D.; Schrauwen, B.; Bienstman, P. Parallel reservoir computing using optical amplifiers. *IEEE Trans. Neural Netw.* **2011**, *22*, 1469–1481. [CrossRef] [PubMed]
24. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; Volume 1944, pp. 1942–1948.
25. Apache Spark™. Available online: http://spark.apache.org/ (accessed on 10 February 2017).
26. Chen, J.; Li, K.; Tang, Z.; Bilal, K.; Li, K. A parallel patient treatment time prediction algorithm and its applications in hospital queuing-recommendation in a big data environment. *IEEE Access* **2016**, *4*, 1767–1783. [CrossRef]
27. Jaeger, H. *Short Term Memory in Echo State Networks*; GMD Report; Forschungszentrum Informationstechnik GmbH: Sankt Augustin, Germany, 2002.
28. Cius, M.L.S.; Dan, P.; Jaeger, H.; Siewert, U. *Time Warping Invariant Echo State Networks*; International University Bremen: Bremen, Germany, 2006.
29. Shi, Z.W.; Han, M. Ridge regression learning in esn for chaotic time series prediction. *Kongzhi Yu Juece Control Decis.* **2007**, *22*, 258–261, 267. (In Chinese).
30. Jaeger, H.; Jaeger, H. *Discovering Multiscale Dynamical Features with Hierarchical Echo State Networks*; Jacobs University Bremen: Bremen, Germany, 2007; Volume 35, pp. 277–284.
31. Vanneschi, L.; Codecasa, D.; Mauri, G. A comparative study of four parallel and distributed pso methods. *New Gener. Comput.* **2011**, *29*, 129–161. [CrossRef]
32. Zhang, C.; Wu, M.; Luan, L. An optimal pso distributed precoding algorithm in qrd-based multi-relay system. *Future Gener. Comput. Syst.* **2013**, *29*, 107–113. [CrossRef]
33. Sheng, C.; Zhao, J.; Wang, W. Map-reduce framework-based non-iterative granular echo state network for prediction intervals construction. *Neurocomputing* **2016**, *222*, 116–126. [CrossRef]
34. Mateos, G.; Bazerque, J.A.; Giannakis, G.B. Distributed sparse linear regression. *IEEE Trans. Signal Process.* **2010**, *58*, 5262–5276. [CrossRef]
35. Boyd, S.; Parikh, N.; Chu, E.; Peleato, B.; Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends® Mach. Learn.* **2011**, *3*, 1–122. [CrossRef]
36. Cevher, V.; Becker, S.; Schmidt, M. Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics. *IEEE Signal Process. Mag.* **2014**, *31*, 32–43. [CrossRef]
37. Chu, E.; Keshavarz, A.; Boyd, S. A distributed algorithm for fitting generalized additive models. *Optim. Eng.* **2013**, *14*, 213–224. [CrossRef]
38. Dean, J.; Corrado, G.S.; Monga, R.; Chen, K.; Devin, M.; Le, Q.V.; Mao, M.Z.; Ranzato, A.; Senior, A.; Tucker, P. Large scale distributed deep networks. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–8 December 2012; pp. 1232–1240.
39. Scardapane, S.; Wang, D.; Panella, M. A decentralized training algorithm for echo state networks in distributed big data applications. *Neural Netw.* **2016**, *78*, 65–74. [CrossRef] [PubMed]
40. Dean, J.; Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [CrossRef]
41. Chang, F.; Dean, J.; Ghemawat, S.; Hsieh, W.C.; Wallach, D.A.; Burrows, M.; Chandra, T.; Fikes, A.; Gruber, R.E. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.* **2008**, *26*, 205–218. [CrossRef]
42. Hall, K.B.; Gilpin, S.; Mann, G. Mapreduce/bigtable for distributed optimization. In Proceedings of the NIPS LCCC Workshop, Whistler, BC, Canada, 10–11 December 2010.
43. Dobre, C.; Xhafa, F. Parallel programming paradigms and frameworks in big data era. *Int. J. Parallel Program.* **2014**, *42*, 710–738. [CrossRef]
44. Mohapatra, S.K.; Sahoo, P.K.; Wu, S.-L. Big data analytic architecture for intruder detection in heterogeneous wireless sensor networks. *J. Netw. Comput. Appl.* **2016**, *66*, 236–249. [CrossRef]
45. Zheng, Y.; Liu, Q.; Chen, E.; Ge, Y.; Zhao, J.L. Time series classification using multi-channels deep convolutional neural networks. In *Web-Age Information Management, Proceedings of the 15th International Conference (WAIM 2014), Macau, China, 16–18 June 2014*; Li, F., Li, G., Hwang, S.-W., Yao, B., Zhang, Z., Eds.; Springer: Cham, Switzerland, 2014; pp. 298–310.
46. Lecun, Y.; Bengio, Y. *Convolutional Networks for Images, Speech, and Time Series*; MIT Press: Boston, MA, USA, 1998.

47. Cui, Z.; Chen, W.; Chen, Y. Multi-scale convolutional neural networks for time series classification. *arXiv*, 2016; arXiv:1603.06995.
48. Wang, Z.; Oates, T. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In Proceedings of the Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, TX, USA, 25–30 January 2015.
49. Dalto, M. Deep neural networks for time series prediction with applications in ultra-short-term wind forecasting. In Proceedings of the 2015 IEEE International Conference on Industrial Technology, Seville, Spain, 17–19 March 2015.
50. Gang, D.; Da, L.; Zhong, S. Time series prediction using convolution sum discrete process neural network. *Neural Netw. World* **2014**, *24*, 421–432. [CrossRef]
51. Aytar, Y.; Vondrick, C.; Torralba, A. Soundnet: Learning sound representations from unlabeled video. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–11 December 2016.
52. Oord, A.V.D.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A.; Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv*, 2016; arXiv:1609.03499.
53. Mantas, L.; Jaeger, H. Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.* **2009**, *3*, 127–149.
54. Jaeger, H. Adaptive nonlinear system identification with echo state networks. In Proceedings of the 16th Annual Neural Information Processing Systems Conference (NIPS 2002), Vancouver, BC, Canada, 9–14 December 2002; Neural Information Processing Systems Foundation: Vancouver, BC, Canada, 2003.
55. Smart* Data Set for Sustainability. Available online: http://traces.cs.umass.edu/index.php/Smart/Smart (accessed on 10 February 2017).
56. The Mnist Database. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 19 March 2017).