International Journal of Innovative
Computing, Information and Control
Volume **13**, Number **4**, August **2017**

 ISSN 1349-4198
pp. **1205–1219**

A PROPOSED PSOSEED2 ALGORITHM FOR TRAINING HARDWARE-BASED AND SOFTWARE-BASED NEURAL NETWORKS

Tuan Linh Dang[1], Thang Cao[2] and Yukinobu Hoshino[1]

[1]School of Systems Engineering
Kochi University of Technology
Tosayamada, Kami City, Kochi 782-8502, Japan
188001d@gs.kochi-tech.ac.jp; hoshino.yukinobu@kochi-tech.ac.jp

[2]Department of Information Physics and Computing
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
cao@hal.ipc.i.u-tokyo.ac.jp

Received February 2017; revised May 2017

Abstract. *Avoiding premature convergence while keeping performance is a challenge in training neural network (NN), especially in a case of large NNs or a large number of training data. This paper proposes an improved particle swarm optimization (PSO) algorithm called the PSOseed2 algorithm for training NN. The PSOseed2 algorithm solves the premature convergence of the standard PSO (SPSO) algorithm by slightly modifying the velocity update function without adding many computational tasks to the SPSO algorithm. To confirm the efficiency of the PSOseed2 algorithm on different NN architectures, we evaluated this algorithm on field-programmable gate array (FPGA)-based NN and software-based NN and trained these NNs with four different PSO algorithms that are SPSO, PSOseed, PSOseed2, and dissipative PSO. Experimental results with four different datasets confirmed that the NNs trained by the proposed PSOseed2 algorithm had better recognition rates and lower global learning errors than the NN trained by three other PSO algorithms.*

Keywords: Particle swarm optimization, Premature convergence, FPGA-based neural network, Software-based neural network

1. **Introduction.** Nowadays, particle swarm optimization (PSO) algorithm has become an attractive target for research. In the PSO algorithm, a movement of each particle in the swarm is based on the knowledge of this particle and the knowledge of the whole population. Each individual in the swarm tends to move to a particle which obtains the best position in the population [1, 2].

The PSO algorithm is a popular method for the training of a neural network (NN). In this situation, parameters concerning weights and biases of the NN are determined by the PSO algorithm during the training phase. The NN trained by the PSO algorithm (NN-PSO) has been used in previous studies [3, 4, 5].

However, the standard PSO (SPSO) algorithm may stick to a local minimum. In this case, the training phase of the NN will stop. The NN cannot be trained, and the recognition rate will be low. Several researchers have focused on the problem of the premature convergence of the SPSO algorithm. The attractive and repulsive PSO (ARPSO) adds the repulsion phase to the SPSO algorithm [6]. In each iteration, the diversity of the swarm is calculated to determine whether the repulsive function is conducted or not. The opposition-based PSO (OPSO) algorithm is also introduced to improve the SPSO algorithm [7, 8]. In OPSO algorithm, the opposition-based particles are calculated. If

these opposition-based particles have higher fitness values than the original particles, the opposition-based particles will replace the original particles. Another algorithm called particle swarm optimization with spatial particle extension (SEPSO) uses radius r to detect the collision of the particles [9]. This mechanism prevents the situation where one particle is trapped in the local minimum, other particles are also attracted to this local minimum, and the algorithm will be stopped. The fully informed PSO algorithm determines not only the best position of one particle but also the best positions of the neighborhood of this particle [10]. These positions are used to calculate the new velocities of the particle. The knowledge-based cooperative particle swarm optimization (KCPSO) uses a multi-swarm mechanism to maintain the swarm diversity [11, 12]. Hence, the premature convergence could be alleviated. Several other researchers have used mutation operations such as Gaussian mutation [13, 14], and Cauchy mutation [15, 16]. The combination of SPSO algorithm and genetic algorithm (GA) is also used to avoid the local minimum [17, 18]. The GA operators such as the crossover and the mutation are added to the SPSO algorithm. These improved versions of the SPSO algorithm may add many compute-intensive tasks to the SPSO.

This paper proposes a novel PSO algorithm called PSOseed2 algorithm which slightly changes the velocity update function without adding many computational tasks to the SPSO algorithm. The PSOseed2 algorithm is based on our PSOseed algorithm which uses seed factors to pull the particles out of the local minimum [19]. In our experiments, the proposed PSOseed2 algorithm was also compared with the dissipative particle swarm optimization (DPSO) algorithm [20]. The DPSO algorithm is an improved version of the SPSO algorithm which also keeps the particles out of the premature convergence without adding many computational tasks to the SPSO algorithm.

In recent years, field-programmable gate array (FPGA) has become an emerging research topic. An FPGA-based program may obtain a higher operating speed than the conventional software-based program [21, 22, 23]. Due to the limitation of logic elements and memory bits, FPGA-based programs are suitable for small scaled applications. In our research, a small-size FPGA-based NN is trained with four different PSO algorithms, namely SPSO, DPSO, PSOseed, and PSOseed2. For experiments with large NNs, the software-based NN is employed.

The main contribution of this paper is to propose an improved version of the PSOseed algorithm, the PSOseed2 algorithm, which could obtain high recognition rates and low learning errors in the training of the NNs without adding many computational tasks to the SPSO algorithm. The hardware-based NN and the software-based NN are used to evaluate the PSOseed2 algorithm.

The rest of this paper is presented as follows. Section 2 describes the encoding strategy for the NN trained by the PSO algorithms. Three different PSO algorithms (standard PSO, dissipative PSO, and PSOseed) are also presented in this section. Section 3 introduces our proposed PSOseed2 algorithm. Section 4 details two approaches for the implementation of the NN-PSO system. One is the software-based NN, and the other one is the hardware-based NN. Section 5 introduces our experiments. Section 6 concludes our paper and also gives several possible avenues for the future research.

2. **Related Work.**

2.1. **NN and encoding strategy.**

In our research, the NN has three layers which are one input layer, one hidden layer, and one output layer. Each layer consists of several nodes. In the NN, the output of one layer becomes the input of the next layer through an activation function. The operation of one node with two inputs of the NN is shown in

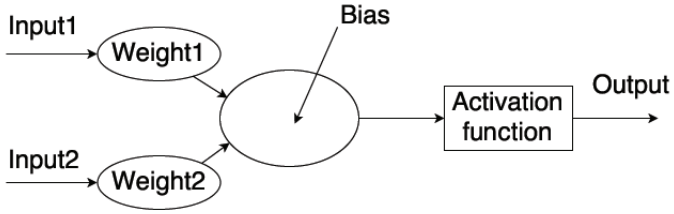


FIGURE 1. Operation of one node

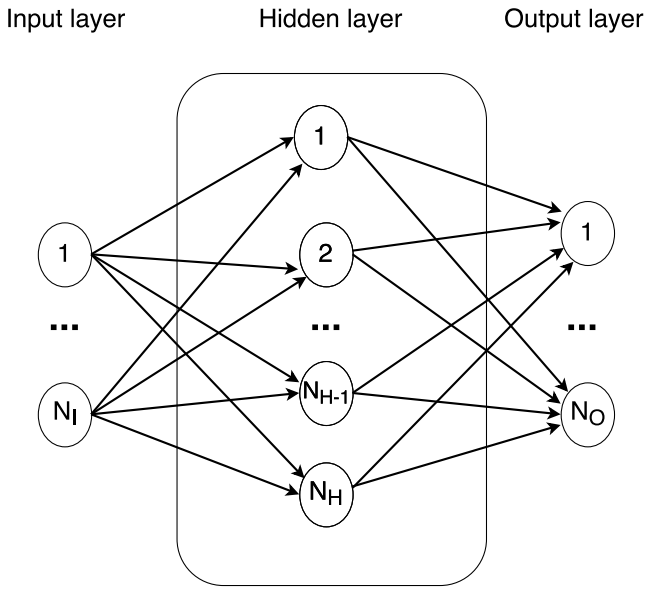


FIGURE 2. Operation of the neural network

Figure 1, and the operation of the NN is illustrated in Figure 2. In Figure 1, the result calculated before the activation function is expressed in Equation (1). This study uses the Sigmoid function as the activation function which is presented in Equation (2).

$$x = Bias + Weight1 \times Input1 + Weight2 \times Input2 \tag{1}$$

where $Bias$ is the bias value, $Input1$ and $Input2$ are the input signals, $Weight1$ and $Weight2$ are the weights, and x is used in the activation function.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Our research focuses only on the feedforward neural network because this type of NN does not contain any loop and may not require many resources of the FPGA device. The number of weights and biases of the NN is calculated as Equation (3).

$$D = (N_I + 1) \times N_H + (N_H + 1) \times N_O \tag{3}$$

where D is the number of weights and biases, $(N_I + 1) \times N_H$ indicates $N_I \times N_H$ weights and N_H biases in the hidden layer. In a similar way, $(N_H + 1) \times N_O$ indicates $N_H \times N_O$ weights and N_O biases in the output layer.

The position $\overrightarrow{x_p}$ of particle p could be presented as one vector as shown in Figure 3. If the PSO training has P particles, P vectors will be used during the training phase of the NN-PSO system. Each vector is considered as one particle which has D parameters. As presented in Equation (3), these D parameters are the number of weights and biases of

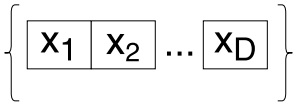


FIGURE 3. Encoding vector for one particle of the NN-PSO system

the NN. These parameters are determined during the PSO training. The PSO algorithms for the training of the NN will be described in the following sections of this paper.

2.2. **Standard PSO (SPSO) algorithm.** The PSO algorithm is based on social behaviors, for example, the bird flocking [1, 2]. At any given time t, the position of a particle p is $\vec{x_p}$. The new velocity $\vec{v_p}(t+1)$ of particle p at time $(t+1)$ can be updated based on Equation (4).

$$\begin{aligned}\vec{v_p}(t+1) = w \times \vec{v_p}(t) + c_1 \times r_1 \times \left(\overrightarrow{xPbest_p}(t) - \vec{x_p}(t)\right) \\ + c_2 \times r_2 \times \left(\overrightarrow{xGbest}(t) - \vec{x_p}(t)\right)\end{aligned} \tag{4}$$

where w is the inertia weight, r_1 and r_2 are the random numbers, c_1 is the cognitive coefficient, c_2 is the social coefficient, $\overrightarrow{xPbest_p}$ is the best personal position found by particle p, $\overrightarrow{xGbest}$ is the best position found by any particle in the swarm, and $\vec{v_p}(t)$ is the velocity of particle p at time t.

The new position $\vec{x_p}(t+1)$ at time $(t+1)$ is expressed in Equation (5).

$$\vec{x_p}(t+1) = \vec{x_p}(t) + \vec{v_p}(t+1) \tag{5}$$

The new fitness values at time $(t+1)$ are also calculated based on Equations (6) and (7).

$$Pbest_p(t+1) = \begin{cases} f(\vec{x_p}(t+1)) & \text{if } f(\vec{x_p}(t+1)) < Pbest_p(t) \\ Pbest_p(t) & \text{if } f(\vec{x_p}(t+1)) \geqslant Pbest_p(t) \end{cases} \tag{6}$$

$$Gbest(t+1) = \underset{p}{\operatorname{argmin}}\, Pbest_p(t+1) \tag{7}$$

where $Pbest_p(t+1)$ is the fitness value of position $\overrightarrow{xPbest_p}(t+1)$ at time $(t+1)$, $Gbest(t+1)$ is the fitness value of position $\overrightarrow{xGbest}(t+1)$ at time $(t+1)$, and $f(.)$ is the fitness function.

The PSO algorithm continues a new iteration to calculate the new velocity $\vec{v_p}(t+2)$, new position $\vec{x_p}(t+2)$, new fitness values $Pbest_p(t+2)$, $Gbest(t+2)$ at time $(t+2)$ until stopping criteria are met. The stopping criteria could be the number of iterations or the final fitness value $Gbest$.

2.3. **Dissipative PSO (DPSO) algorithm.** The standard PSO (SPSO) could be trapped by the local minimum. In this situation, the NN may not be trained. A well-known algorithm was created to overcome the premature convergence called dissipative PSO algorithm (DPSO) [20]. This is a simple algorithm which adds Equations (8) and (9) after calculating the new velocity $\vec{v_p}(t+1)$ and the new position $\vec{x_p}(t+1)$.

$$\text{If } (rand() < c_v) \text{ Then } \quad \vec{v_p}(t+1) = rand() \times \overrightarrow{v_{max}} \tag{8}$$

$$\text{If } (rand() < c_l) \text{ Then } \quad \vec{x_p}(t+1) = Rand(lo, up) \tag{9}$$

c_v and c_l are numbers in the range $[0, 1]$, $Rand(lo, up)$ is the random number between $[lo, up]$, $rand()$ is the random number in the range $[0, 1]$, and $\overrightarrow{v_{\max}}$ is the maximum velocity.

2.4. **PSOseed algorithm.** In our previous research, the PSOseed algorithm was introduced to solve the premature convergence [19]. In this PSOseed algorithm, the velocity update function was modified as can be seen in Equation (10).

$$\begin{aligned}\overrightarrow{v_p}(t+1) = w \times \overrightarrow{v_p}(t) + c_1 \times \left(\overrightarrow{xPbest_p}(t) - \overrightarrow{x_p}(t)\right) \\ + c_2 \times \left(\overrightarrow{xGbest}(t) - \overrightarrow{x_p}(t)\right) + c_3 \times r \times \left(\overrightarrow{xSeed_p} - \overrightarrow{x_p}(t)\right)\end{aligned} \tag{10}$$

where $\overrightarrow{xSeed_p}$ is the seed position of particle p, c_3 is the coefficient, and r is the random number.

In this algorithm, the seed positions of all particles are randomly generated in the initialization phase of the PSO training. At any given time t, the calculation of the new velocity at time $(t+1)$ is not only influenced by the current velocity $\overrightarrow{v_p}(t)$, current $\overrightarrow{xPbest_p}(t)$, current $\overrightarrow{xGbest(t)}$ but also affected by the seed position $\overrightarrow{xSeed_p}$ of this particle. This seed factor always pulls the particles to the seed positions; even the particles are at the local minimum. Using this mechanism, the particles may continue to search in other areas of the searching space.

3. **Proposed PSOseed2 Algorithms.** In the PSOseed algorithm, with the appearance of the seed factor, the particles may be kept out of the local minimum. However, the efficiency of the PSOseed algorithm highly depends on the seed positions. If the suitable seed factors are generated in the initial phase, the PSOseed could obtain a high recognition rate and a low *Gbest*. On the other hand, the performance of the PSOseed algorithm decreases significantly if the seed positions are improper.

To overcome the drawback of the seed positions of the PSOseed algorithm, the PSOseed2 algorithm which has a mechanism to control the seed positions is introduced. Similar to the PSOseed algorithm, the seed positions of the PSOseed2 algorithm are randomly generated in the initialization phase of the PSO training. In each iteration, the calculating of the new velocity uses the seed positions as presented in Equation (10). However, these seed positions are not fixed as can be seen in the PSOseed algorithm. If the calculated fitness value $Gbest(t)$ at time t is not lower than the current fitness value *Gbest* of the PSOseed algorithm, the seed factors will be reseeded. The mechanism of our PSOseed2 algorithm can be presented as follows.

- Generate the seed positions of all particles in the initial step.
- Execute the PSO calculation for all particles based on:
  - Equation (10) to calculate the new velocities.
  - Equation (5) to calculate the new positions.
  - Equation (6) to estimate the new fitness values *Pbest* and the corresponding positions $\overrightarrow{xPbest}$, for all particles.
  - Equation (7) to estimate the new fitness values *Gbest* and the corresponding positions $\overrightarrow{xGbest}$.
- Compare the calculated *Gbest* with the current *Gbest* of the PSOseed2 algorithm:
  - If $calculated\_Gbest \geq current\_Gbest$ then reseed all seed factors.
  - If $calculated\_Gbest < current\_Gbest$ then update $current\_Gbest = calculated\_Gbest$.
- Continue a new iteration until the stopping criteria are satisfied.

Figure 4 illustrates the reseed mechanism of the PSOseed2 algorithm which could be used to solve the problem of seed positions. If both particle p and its initial seed position are in the local minimum, the PSOseed algorithm will stop. On the other hand, the

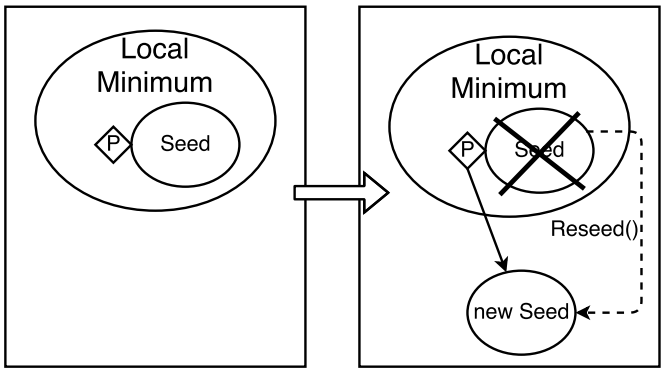


FIGURE 4. Operation of the PSOseed2 algorithm

PSOseed2 algorithm will activate the reseed mechanism that generates a new seed position of particle p. This new seed will pull particle p to its position.

4. **Implementation of the NN-PSO System.** The FPGA-based NN obtains a higher operating speed than the software-based NN. However, a large hardware-based NN which has many nodes requires many logic elements. This hardware-based NN may not fit in an FPGA device. Thus, this research employs the software-based NN for experiments with large NNs.

This paper presents two different approaches to evaluate the PSO algorithms. The first approach is the software-based NN, and the other approach is the FPGA-based NN.

4.1. **The software-based NN trained by the PSO algorithms.** In this design, the software-based NN is trained by four different PSO algorithms (SPSO, DPSO, PSOseed, and PSOseed2). The NN and the PSO algorithms were coded in the C language in Visual Studio 2013 [24].

Figure 5 presents the training phase of our implementation. The training data have two parts called input data and labeled data. The input data are sent to the particle blocks, and the labeled data are sent to the evaluation modules. Each particle is a D-dimensional vector that corresponds to D parameters (weights and biases) of one NN. The output data of the NN are fed to the evaluation modules. In these modules, the fitness values

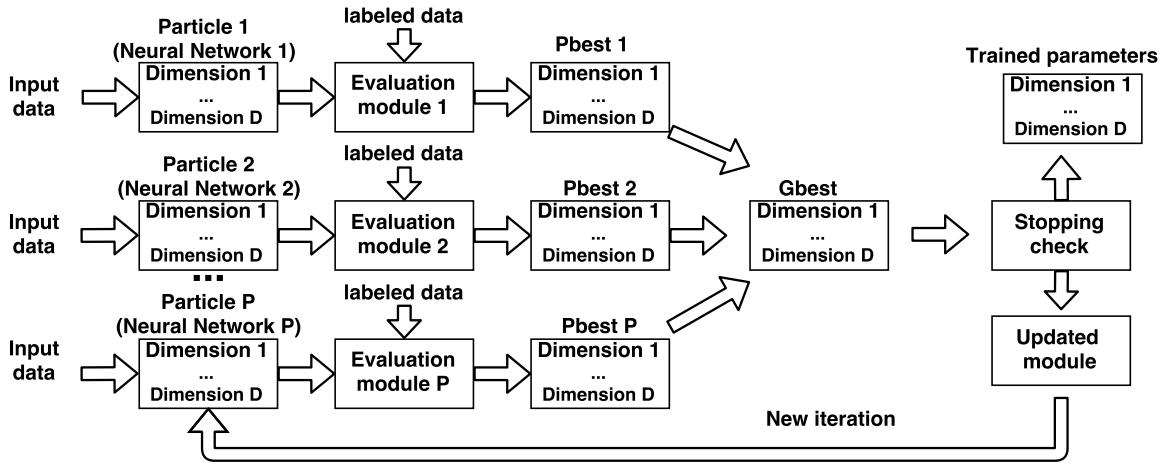


FIGURE 5. Training phase of the NN-PSO

are calculated by the mean square error function as seen in Equation (11).

$$fitness_i = \frac{1}{T}\sum_{j=1}^{T}(labeled\_data_j(k) - output\_data_{ij}(k))^2 \tag{11}$$

where $fitness_i$ is the fitness value of module i, T is the number of training samples, $labeled\_data_j(k)$ is the kth component of labeled data j ($1 \leq k \leq D$), and $output\_data_{ij}(k)$ is the kth component of output data j of particle i ($1 \leq k \leq D$).

The fitness values from evaluation modules are used to estimate the new $Pbest_p$ based on Equation (6) and the new $Gbest$ based on Equation (7).

The stopping-check module is used to check whether the stopping criteria are satisfied or not. If the stopping conditions are not met, the new velocities and positions of all particles are calculated in the updated module, and the new iteration will be conducted. The operation of the updated module depends on the PSO algorithm. For example, Equations (10), (5), and the reseed mechanism are used in the updated module when the PSOseed2 algorithm is used. On the other hand, the D trained parameters (weights and biases) of the NN are determined if the stopping criteria are met. These D trained parameters will be used in the testing phase.

4.2. **The FPGA-based NN trained by the PSO algorithms.** The FPGA-based NN may obtain a higher operating speed than the software-based NN. The operation of hardware-based NN trained by different PSO algorithms (SPSO, DPSO, PSOseed, and PSOseed2) was also investigated. Since an FPGA device has limited logic elements and limited memory bits, our FPGA-based NN has a smaller number of nodes in each layer than the software-based NN. In addition, the number of input nodes of our NN corresponds to the number of attributes in the dataset. For this reason, the datasets used in the FPGA-based NN experiments also have a smaller number of attributes than the datasets employed in the software-based NN experiments.

Figure 6 shows our design. The PSO algorithms are implemented in a hard processor system with an ARM processor. This study employs the Cyclone V hard processor system provided by Altera [25]. In the hard processor system, a Linux operating system is installed on the ARM processor. Our research uses Ubuntu as the operating system [26]. The Linux file systems are stored on an SD card. The DDR3 random-access memory (RAM) is divided into two different sections. The first section is reserved for the Ubuntu operating system. The second section is used for the data transmission. All weights, biases, input data sent to the NN, output data received from the NN are put in the second section.

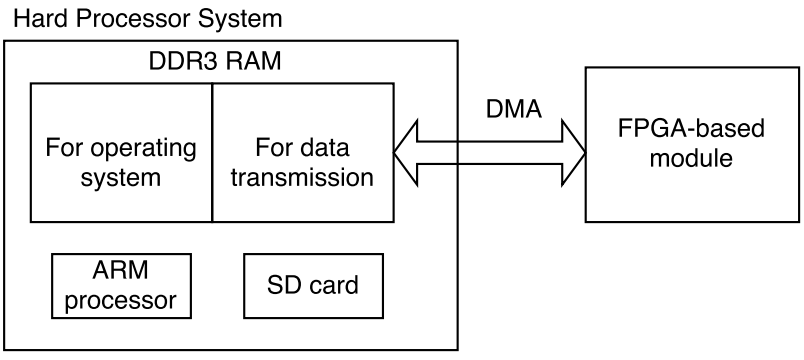


FIGURE 6. Hardware-based NN trained by PSO algorithms

Using the direct memory access (DMA) controller [27], the data from the transmission section may be sent to and be received from the FPGA-based module by DMA. The DMA controller is configured by the ARM processor with the address of the data transmission section and the address of the FPGA-based module.

An Avalon memory-mapped slave interface [28] is created in the FPGA-based module to do the data transmission with the DMA. The floating-point calculation is used in our design. The FPGA-based module has not only the hardware-based NN but also the floating-point (FP) calculations. Figure 7 presents the operation of the FPGA-based module. The details of the operation are as follows.

1) The operation of the FPGA-based NN is based on a finite state machine (FSM) which has two main states called *waiting_phase* and *NN_running_state*. In each clock cycle, the NN checks the status of the FP submodule by using $flag$ signal. If the $flag$ signal is ready ($flag = 1$), the FSM of the NN moves to *NN_running_state*. Otherwise, the FSM of the NN is put in the *waiting_state*. The *NN_running_state* of the FSM has several smaller states to calculate the output of each node in each layer of the NN based on Equations (1) and (2).
2) When the NN needs to use the floating-point calculations, the FPGA-based NN submodule sends *require_calculations* signal to the FP submodule. For example, if the addition is required, the NN will send *float_alu_mode_add* signal to the FP component.
3) The input data of the NN are forwarded to the FP submodule using *data* signal.
4) The NN rechecks the $flag$ signal of the NN to determine whether the processing of floating-point calculations is finished or not.
5) In next clock cycles
   (a) If the processing is finished ($flag = 1$), the NN receives the output data from the FP submodule. The FSM of the NN continues the operation in the *NN_running_state*. When the NN needs to use again the floating-point calculations, the algorithm returns to step 2.
   (b) If $flag = 0$, the FSM of the NN changes to the *waiting_phase*.

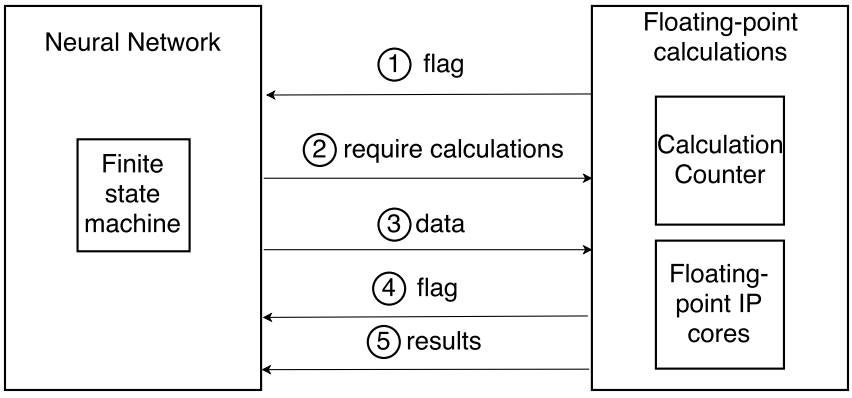


FIGURE 7. Operation of the FPGA-based module

The FP submodule implements the floating-point IP cores provided by Altera [29]. These IP cores are used in our research because these cores are already optimized for our FPGA device. In addition, the output latency of the arithmetic operation is also specified. These output latencies are used in the calculation-counter. If FP submodule receives *require_calculations* signal, the $flag$ signal is assigned value 0 ($flag = 0$), and the calculation-counter is set to a corresponding value. For example, the output

latency of the addition is seven clock cycles. When *require_calculations* signal is the $float\_alu\_mode\_add$, the calculation-counter is set to seven. This value is decreased one unit on each clock cycle. When the counter value equals zero or the addition calculation is finished, the $flag$ signal will have value 1 ($flag = 1$). When the counter value is greater than zero, the $flag$ signal still has value 0 ($flag = 0$).

5. **Experiments.** Four different PSO algorithms (SPSO, DPSO, PSOseed, and PSOseed2) were used to train two different types of NNs (software-based NN and FPGA-based NN).

Based on our experimental results, the parameters for the PSO algorithms which obtained high recognition rates and low learning errors in the training of the NNs could be $w = 0.9$, $c_1 = 0.5$, $c_2 = 0.5$, $c_3 = 0.5$, and the range of velocity was from -2 to 2. The parameters of the DPSO algorithm were similar to the previous research [20] ($c_v = 0.0$, $c_l = 0.001$). In our experiments, the number of particles and the number of iterations were varied to investigate different situations of the NN-PSO system.

Both FPGA-based NN and software-based NN trained by PSO algorithms were tested on different datasets to observe the operation of the NN trained by PSO algorithms in various scenarios. The datasets tested for each type of NN have different numbers of attributes and classes. In our programs, the number of attributes and classes of the datasets corresponds, respectively, to the number of input nodes and output nodes of the NN. If one dataset has a large number of attributes or classes, the NN needs to have a large number of input nodes or output nodes. This NN will not fit in an FPGA device because the FPGA-based NN has a drawback concerning the limited logic elements and the limited memory bits. Thus, the FPGA-based NN was used to test with the wine dataset and the Australian credit dataset which have smaller than fifteen attributes. The software-based NN which had more nodes in each layer than the hardware-based NN was trained with the spam dataset and the handwritten digit dataset that have more attributes than the datasets used in the FPGA-based NN experiments. Each dataset was randomly divided into two different subsets, and the cross-validation was conducted. The list of experiments is illustrated in Table 1.

Table 1. List of experiments

	Datasets
Software-based NN	Spambase dataset: 57 attributes, 2 classes, cross-validation data: 3601 samples in set 1, 1000 samples in set 2 (Section 5.1.1)
	Handwritten digits dataset: 64 attributes, 10 classes, cross-validation data: 1500 samples in set 1, 500 samples in set 2 (Section 5.1.2)
FPGA-based NN	Wine dataset: 13 attributes, 3 classes, cross-validation data: 120 samples in set 1, 58 samples in set 2 (Section 5.2.1)
	Australian credit dataset: 14 attributes, 2 classes, cross-validation data: 490 samples in set 1, 200 samples in set 2 (Section 5.2.2)

5.1. **Software-based NN with PSO algorithms.** The software-based NN experiments were conducted in Visual Studio 2013 [24].

5.1.1. *Spambase dataset.* The spambase dataset is the collection of 4601 emails, which are marked spam email or non-spam email [30]. Each email has 57 attributes. Most of the attributes are the percentage of particular words or characters. Our experiments used a three-layer neural network which had 57 input nodes corresponded to 57 attributes, 2 output nodes corresponded to 2 classes, and 10 hidden nodes. The spambase dataset was divided randomly into two different sets. Set 1 had 3601 samples and set 2 had 1000 samples. When set 1 was used as the training set, set 2 was considered as the testing set and vice versa.

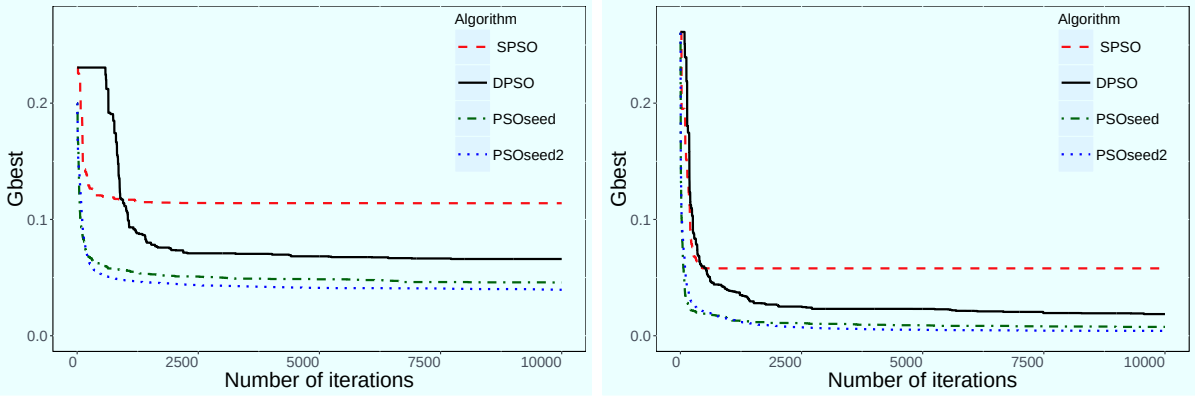
Table 2 and Figure 8(a) describe the experimental results when 3601 samples of set 1 were selected as the training data and 1000 samples of set 2 were used as the testing data. The parameters for the PSO algorithms were 20 particles, 10000 iterations. The *Gbest*, the global fitness value of the learning error, of the PSOseed2 algorithm decreased to the lowest value (0.0396). The PSOseed2 algorithm also obtained the higher recognition rate (95.40%) than other three algorithms.

Figure 8(b) presents the reduction of the *Gbest* when set 2 (1000 samples) was considered as the training data and set 1 (3601 samples) was chosen as the testing data. The number of particles and the number of iterations were kept at 20 particles and 10000 iterations. The PSOseed2 algorithm still produced better *Gbest* (final *Gbest* = 0.0042) than other three algorithms. As can be seen in Table 2, the recognition rate of the PSOseed2 algorithm was also the highest number (88.25%).

Experimental results showed that the PSOseed2 algorithm obtained the lowest fitness value *Gbest* and the highest recognition rate among four different algorithms not only

TABLE 2. Results with spambase dataset

	3601 training data, 1000 testing data				1000 training data, 3601 testing data			
Algorithm	SPSO	DPSO	PSOseed	PSOseed2	SPSO	DPSO	PSOseed	PSOseed2
Final *Gbest*	0.1140	0.0660	0.0458	0.0396	0.0580	0.0186	0.0076	0.0042
Recognition rate	87.40%	92.40%	93.60%	95.40%	80.34%	85.62%	85.87%	88.25%



(a) 3601 training data, 1000 testing data (b) 1000 training data, 3601 testing data

FIGURE 8. Reduction of *Gbest* with spambase dataset

with the large number of training samples (3601 samples) but also with the small number of training samples (1000 samples).

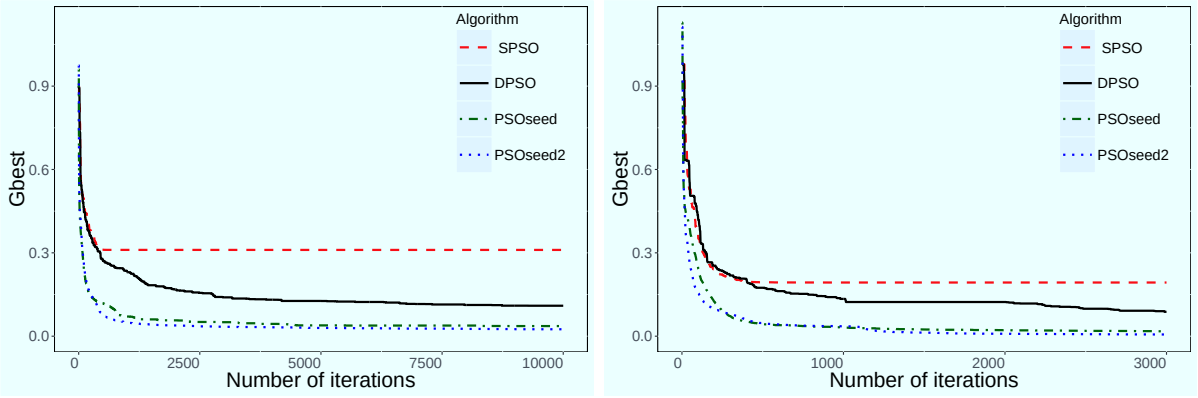
5.1.2. *Handwritten digits dataset.* The software-based NN trained by four PSO algorithms was also tested with another dataset called handwritten digits [30]. This database contains the processed handwritten digits from 0 to 9. Each processed digit is an input matrix of size 8×8 whose each element has a range from 0 to 16. Each sample in handwritten digits has 64 attributes and belongs to one of ten classes. With this dataset, 1500 samples were selected randomly as set 1, and other 500 samples were also chosen randomly as set 2.

In the first test, set 1 with 1500 samples was considered as the training data and set 2 with 500 samples was chosen as the testing data. The parameters of the NN were 64 input nodes (for 64 attributes), 10 output nodes (for 10 classes), and 20 hidden nodes. The number of particles was 85 particles. The number of iterations was 10000 iterations. The reduction of *Gbest* is presented in Figure 9(a). The final *Gbest* values and the final recognition rates after 10000 iterations are shown in Table 3. These results demonstrated the high accuracy of the software-based NN trained by the PSOseed2 algorithm concerning the recognition rates and the global learning errors (*Gbest*) among four different PSO algorithms.

To investigate the cross-validation, the 500-samples dataset was also selected as the training data. The NN in this experiment had 64 input nodes, 10 output nodes, and 15 hidden nodes. The PSO parameters were 100 particles and 3000 iterations. The experimental results are shown in Table 3 and Figure 9(b). The final *Gbest* of the PSOseed2 decreased to the lowest value 0.0062 among these four algorithms. The recognition rate of the PSOseed2 was also the highest number (85.60%).

TABLE 3. Results with handwritten digits dataset

	1500 training data, 500 testing data				500 training data, 1500 testing data			
Algorithm	SPSO	DPSO	PSOseed	PSOseed2	SPSO	DPSO	PSOseed	PSOseed2
Final *Gbest*	0.3105	0.1096	0.0364	0.0250	0.1931	0.0877	0.0177	0.0062
Recognition rate	62.20%	92.00%	96.20%	97.60%	40.27%	60.80%	84.93%	85.60%



(a) 1500 training data, 500 testing data

(b) 500 training data, 1500 testing data

FIGURE 9. Reduction of *Gbest* with handwritten digits dataset

5.2. **FPGA-based NN with PSO algorithms.** The PSO algorithms were also evaluated in the training of the FPGA-based NN. Our experiments used the FPGA-based NN which had a smaller number of nodes in each layer than the software-based NN on account of the limited logic elements and the limited memory bits of our FPGA device.

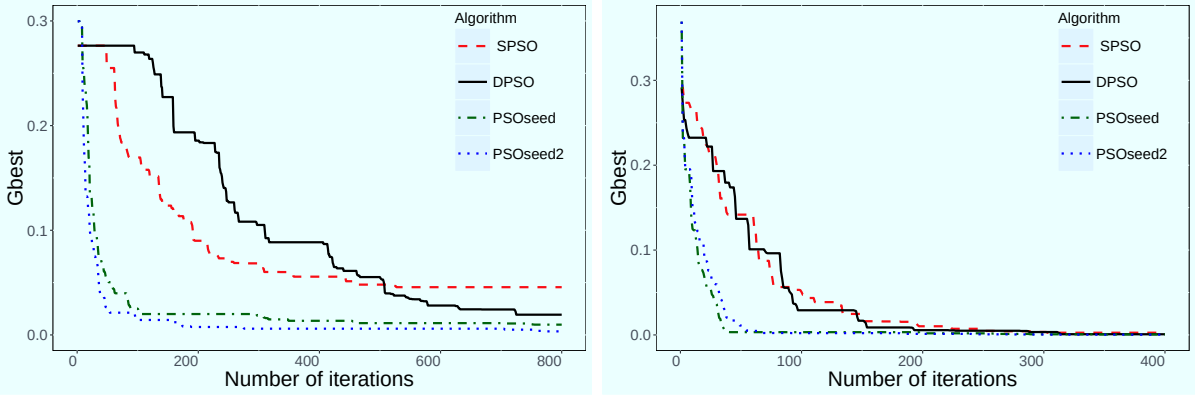
In our FPGA-based experiments, the FPGA-based NN was implemented in hardware by SystemVerilog programming language. The PSO algorithms were coded in the C programming language by the ARM processor. The target device in our experiments was the Cyclone V SoC provided by Altera [25].

5.2.1. *Wine dataset.* The wine dataset comes from a chemical analysis of Italian wines. This dataset has three different types of wines (three classes) and 178 samples [30]. Each sample in the dataset contains thirteen attributes. The parameters of the NN in the experiments with wine dataset were 13 input nodes, 3 output nodes, and 24 hidden nodes. The samples of this dataset were divided into two different sets to conduct the cross-validation. The first set had 120 data samples, and the second set had remaining 58 samples. The data in each set were chosen randomly.

Figure 10(a) shows the reduction of *Gbest*, the learning errors, of the FPGA-based NN trained by four different algorithms when the 120-samples dataset was used as the training data, and the 58-samples dataset was considered as the testing data. The configurations of the PSO algorithms were 15 particles and 800 iterations. Similar to experiments with the software-based NN, the *Gbest* of the PSOseed2 algorithm still decreased to the lowest value. Concerning the recognition rates of the NN, Table 4 gives information that at the minimum value of *Gbest* (0.0034), the PSOseed2 algorithm also obtained the highest percentage of correct recognition at 98.28%. On the other hand, the NN trained by the SPSO algorithm still had the lowest recognition rate at 89.66% and the highest *Gbest* at 0.0457 among four algorithms.

TABLE 4. Results with wine dataset

	120 training data, 58 testing data				58 training data, 120 testing data			
Algorithm	SPSO	DPSO	PSOseed	PSOseed2	SPSO	DPSO	PSOseed	PSOseed2
Final *Gbest*	0.0457	0.0193	0.0098	0.0034	0.0026	0.0008	0.0006	0.0003
Recognition rate	89.66%	94.83%	96.55%	98.28%	86.67%	90.00%	90.00%	93.33%



(a) 120 training data, 58 testing data (b) 58 training data, 120 testing data

FIGURE 10. Reduction of *Gbest* with wine dataset

Figure 10(b) and Table 4 show the reduction curves of *Gbest*, the final *Gbest* values, and the recognition rates when set 1 (58 samples) was used as the training data and set 2 (120 samples) was used as the testing data. The PSO parameters were 40 particles, 400 iterations. These experimental results confirmed the high accuracy concerning the recognition rates and the learning errors of the FPGA-based NN trained by the PSOseed2 algorithm.

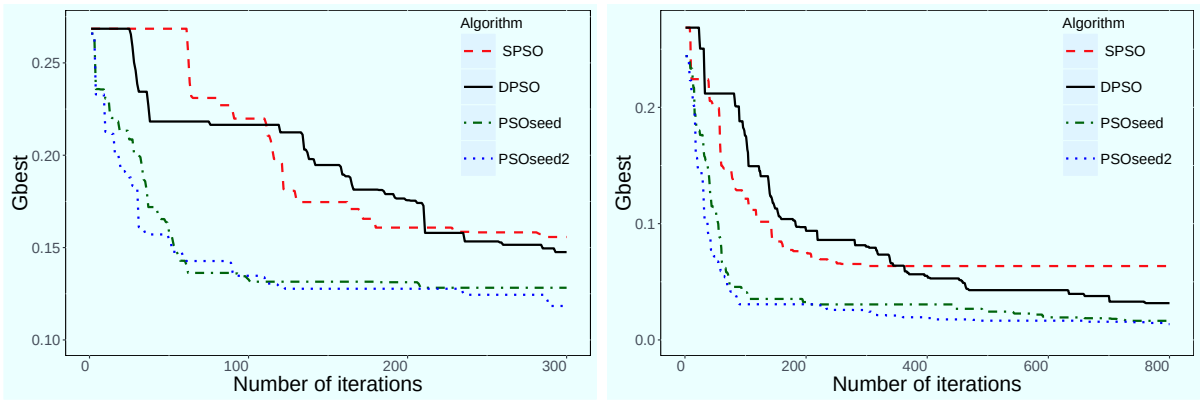
5.2.2. *Australian credit dataset.* The Australian credit approval dataset was also used to observe the operation of the FPGA-based NN trained by four PSO algorithms in a different situation. This database has 690 samples which consist of two classes called approval and rejection. Each sample which belongs to one of these two categories has fourteen attributes [30]. To conduct the experiments with this data, the NN had 14 nodes in the input layer, 24 nodes in the hidden layer, and 2 nodes in the output layer. Similar to the experiments with other datasets, this Australian credit dataset was also divided randomly into two different sets. The first set had 490 samples, and the second set consisted of 200 samples.

When set 1 was used as the training data, 200 samples of set 2 were considered as the testing data. The settings for the PSO algorithms were 10 particles, 300 iterations. Experimental results which contain the recognition rates and the learning errors *Gbest* are presented in Table 5. The reduction of the learning errors in each iteration is shown in Figure 11(a). These results showed that the NN trained by PSOseed2 algorithm achieved the highest accuracy when compared with the NN trained by the PSOseed algorithm, the NN trained by the DPSO algorithm, and NN trained by the SPSO algorithm.

To conduct the cross-validation, set 2 that had 200 samples was also used as the training data, and 490 samples of set 1 were also selected as the testing data. This experiment was conducted with 5 particles and 800 iterations as the parameters. The learning curves of the NN trained by PSO algorithms are illustrated in Figure 11(b). As seen in Table 5,

TABLE 5. Results with Australian credit dataset

	490 training data, 200 testing data				200 training data, 490 testing data			
Algorithm	SPSO	DPSO	PSOseed	PSOseed2	SPSO	DPSO	PSOseed	PSOseed2
Final *Gbest*	0.1558	0.1476	0.1282	0.1194	0.0635	0.0316	0.0164	0.0136
Recognition rate	90.00%	93.00%	94.50%	96.00%	80.82%	81.84%	82.65%	82.86%



(a) 490 training data, 200 testing data

(b) 200 training data, 490 testing data

FIGURE 11. Reduction of *Gbest* with Australian credit dataset

the final values of the learning errors were 0.0136 with the PSOseed2 algorithm, 0.0164 with the PSOseed algorithm, 0.0316 with the DPSO algorithm, and 0.0635 with SPSO algorithm. The NN trained by PSOseed2 algorithm also obtained the highest recognition rate at 82.86%. Experimental results confirmed that the PSOseed2 could be a suitable solution for the training of the FPGA-based NN even with the small number of training samples or the small number of particles.

6. **Conclusions.** This paper proposes an enhanced version of the SPSO algorithm called the PSOseed2 algorithm for training NN. The PSOseed2 algorithm slightly modifies the velocity update function with only slight impact on the performance of the SPSO algorithm in the training of the NN concerning the recognition rate and the learning error.

The operation of the PSOseed2 algorithm was compared with three other PSO algorithms (SPSO algorithm, DPSO algorithm, and PSOseed algorithm) on two different architectures. In the first architecture presented in Section 4.1 of this manuscript, four PSO algorithms were used to train the software-based NN. In the second architecture presented in Section 4.2 of this paper, the PSO algorithms were employed to train the hardware-based NN.

In our architectures, the number of input nodes of the NN corresponds to the number of attributes in the dataset. If the dataset has a large number of attributes, the NN needs to have a large number of input nodes. However, the FPGA-based NN that has a large number of nodes will not fit in an FPGA device because of the limited resources concerning the logic elements and the memory bits. Therefore, the FPGA-based NN was used to test with the datasets which have smaller than fifteen attributes. The software-based NN which has more nodes in each layer than the hardware-based NN was used to train with datasets that have more attributes than the datasets employed in the FPGA-based NN experiments.

Our experimental results demonstrated that the implementations of both FPGA-based NN and software-based NN trained by PSO algorithms were feasible. With the same conditions for the parameters, both the FPGA-based NN and the software-based NN trained by the PSOseed2 algorithm obtained higher recognition rates and lower fitness values (*Gbest*) than the NNs trained by other three PSO algorithms.

Our research used the Cyclone V as the FPGA device. This Cyclone V chip, which is the low system cost and performance, has limited resources concerning the logic elements and the memory bits. For this reason, the PSO algorithms were implemented in software. In future studies, if higher device families such as Arria or Stratix could be employed, the PSO algorithms could be implemented in hardware to increase the training speed. In this situation, the logic utilization of the PSOseed2 algorithm could be investigated. This paper focuses on the feedforward NN. Another possible avenue for the future research is to study the operation of the PSOseed2 algorithm in the training of other types of the NN.

REFERENCES

[1] J. Kennedy and R. Eberhart, Particle swarm optimization, *Proc. of the IEEE International Conference on Neural Networks*, vol.4, pp.1942-1948, 1995.

[2] R. Poli, J. Kennedy and T. Blackwell, Particle swarm optimization, *Swarm Intelligence*, vol.1, no.1, pp.33-57, 2007.

[3] Z. A. Bashir and M. E. El-Hawary, Applying wavelets to short-term load forecasting using PSO-based neural networks, *IEEE Trans. Power Systems*, vol.24, pp.20-27, 2009.

[4] K. W. Chau, Application of a PSO-based neural network in analysis of outcomes of construction claims, *Automation in Construction*, vol.16, no.5, pp.642-646, 2007.

[5] W. Z. Lu, H. Y. Fan, A. Y. T. Leung and J. C. K. Wong, Analysis of pollutant levels in central Hong Kong applying neural network method with particle swarm optimization, *Environmental Monitoring and Assessment*, vol.79, no.3, pp.217-230, 2002.
[6] J. Riget and J. S. Vesterstrom, *A Diversity-Guided Particle Swarm Optimizer – The ARPSO*, EVALife Technical Report, vol.2, pp.1-13, 2002.
[7] L. Han and X. He, A novel opposition-based particle swarm optimization for noisy problems, *Proc. of the 3rd International Conference on Natural Computation*, pp.624-629, 2007.
[8] H. Wang, H. Li, Y. Liu, C. Li and S. Zeng, Opposition-based particle swarm algorithm with Cauchy mutation, *Proc. of the IEEE Congress on Evolutionary Computation*, pp.4750-4756, 2007.
[9] T. Krink, J. S. Vesterstrom and J. Riget, Particle swarm optimisation with spatial particle extension, *Proc. of the IEEE Congress on Evolutionary Computation*, pp.1474-1479, 2002.
[10] R. Mendes, J. Kennedy and J. Neves, The fully informed particle swarm: Simpler, maybe better, *IEEE Trans. Evolutionary Computation*, vol.8, no.3, pp.204-210, 2004.
[11] J. Jie, J. Zeng, C. Han and Q. Wang, Knowledge-based cooperative particle swarm optimization, *Applied Mathematics and Computation*, vol.205, no.2, pp.861-873, 2008.
[12] N. J. Cheung, X. M. Ding and H. B. Shen, OptiFel: A convergent heterogeneous particle swarm optimization algorithm for Takagi-Sugeno fuzzy modeling, *IEEE Trans. Fuzzy Systems*, vol.22, no.4, pp.919-933, 2014.
[13] N. Higashi and H. Iba, Particle swarm optimization with Gaussian mutation, *Proc. of the IEEE Swarm Intelligence Symposium*, pp.72-79, 2003.
[14] Q. Wu, Power load forecasts based on hybrid PSO with Gaussian and adaptive mutation and Wv-SVM, *Expert Systems with Applications*, vol.37, no.1, pp.194-201, 2010.
[15] A. Stacey, M. Jancic and I. Grundy, Particle swarm optimization with mutation, *Proc. of the IEEE Congress on Evolutionary Computation*, vol.2, pp.1425-1430, 2003.
[16] C. Li, Y. Liu, A. Zhou, L. Kang and H. Wang, A fast particle swarm optimization algorithm with Cauchy mutation and natural selection strategy, *Advances in Computation and Intelligence, Lecture Notes in Computer Science*, vol.4683, pp.334-343, 2007.
[17] B. Yang, Y. Chen and Z. Zhao, A hybrid evolutionary algorithm by combination of PSO and GA for unconstrained and constrained optimization problems, *Proc. of the IEEE International Conference on Control and Automation*, pp.166-170, 2007.
[18] M. Lovbjerg, T. K. Rasmussen and T. Krink, Hybrid particle swarm optimiser with breeding and subpopulations, *Proc. of the Genetic and Evolutionary Computation Conference*, pp.469-476, 2001.
[19] T. L. Dang and Y. Hoshino, An-FPGA based classification system by using a neural network and an improved particle swarm optimization algorithm, *Proc. of the Joint the 8th International Conference on Soft Computing and Intelligent Systems and the 17th International Symposium on Advanced Intelligent Systems*, pp.97-102, 2016.
[20] X. F. Xie, W. J. Zhang and Z. L. Yang, Dissipative particle swarm optimization, *Proc. of the IEEE Congress on Evolutionary Computation*, pp.1456-1461, 2002.
[21] E. Monmasson and N. C. M. Cirstea, FPGA design methodology for industrial control systems – A review, *IEEE Trans. Industrial Electronics*, vol.54, no.4, pp.1824-1842, 2007.
[22] J. A. G. Pulido, M. A. V. Rodriguez, J. M. S. Perez, S. P. Mendes and V. Carreira, Accelerating floating-point fitness functions in evolutionary algorithms: A FPGA-CPU-GPU performance comparison, *Genetic Programming and Evolvable Machines*, vol.12, no.4, pp.403-427, 2011.
[23] S. Asano, T. Maruyama and Y. Yamaguchi, Performance comparison of FPGA, GPU and CPU in image processing, *Proc. of the International Conference on Field Programmable Logic and Applications*, pp.126-131, 2009.
[24] Microsoft, *Visual Studio*, https://www.visualstudio.com, 2017.
[25] Altera, *Cyclone V Device Handbook*, https://www.altera.com, 2016.
[26] Canonical, *Ubuntu*, https://www.ubuntu.com, 2017.
[27] Altera, *Embedded Peripherals IP User Guide*, https://www.altera.com, 2016.
[28] Altera, *Avalon Interface Specifications*, https://www.altera.com, 2015.
[29] Altera, *Floating-Point IP Cores User Guide*, https://www.altera.com, 2016.
[30] M. Lichman, *UCI Machine Learning Repository*, http://archive.ics.uci.edu/ml, 2013.