# A Variation of Particle Swarm Optimization for Training of Artificial Neural Networks

Masood Zamani and Alireza Sadeghian
*Ryerson University*
*Canada*

## 1. Introduction

Particle swarm optimization (PSO) is a stochastic global optimization method (Eberhart & Kennedy, 1995) that belongs to the family of Swarm Intelligence and Artificial Life. Similar to artificial neural networks (ANN) and genetic algorithms (GA) which are the simplified models of the neural system and the natural selection of the evolutionary theory, PSO is a simplified model of psychological principles and social behaviors (Reynolds, 1987). PSO is based on the principles that flocks of birds,schools of fish, or swarm of bees searches for food sources where at the beginning the perfect location is not known. However, they eventually reach the best location of food source by means of communicating with each other.

PSO is also conceptually compared to evolutionary computation methods such as GA (Eberhart & Shi, 1998). The uniqueness of PSO is the dynamic interaction among the particles.The optimization method starts with randomly generated particles (the population) in a defined domain called the search space. Particle locations are updated in each generation (iteration) to explore the search space for an optimum solution. In PSO, particle positions and velocities are updated based on cooperation and competition. Each particle finds its next position in the search space according to its own search experience and the best experience of the particles located in its local group, neighborhood and the entire population. Neighbors are the particles located within a pre-defined distance of a specific particle.

In this article, we propose a method to update the velocities and positions of particles when the maximum search space boundary and velocity are reached. The efficiency of the proposed particle swarm optimization method is investigated through the training of feed-forward artificial neural networks used for classification. The experiments show the particle swarm optimization lends itself very well to training of neural networks and is also highly competitive with the other methods of training feed-forward ANNs. We have conducted four classification experiments using feed-forward ANNs with PSO based training. The data sets used in the experiments are from the UCI repository (Asuncion & Newman, 2007) commonly used in the literature.

## 2. Related works

The most widely used method of training for feed-forward ANNs is back-propagation (BP) algorithm (Hecht-Nelso R., 1989). Feed-forward ANNs are commonly used for function approximation and pattern classifications. Back-propagation algorithm and its variations such as QuickProp (Fahlman, 1998) and RProp (Riedmiller and Braun, 1993) are likely to reach local minima especially in case that the error surface is rugged. In addition,the efficiency of BP methods depends on the selection of appropriate learning parameters. The other training methods for feed-forward ANNs include those that are based on evolutionary computation and heuristic principles such as Genetic Algorithm(GA), and PSO.

Although, Genetic Algorithm (Mitchell M., 1988) is a suitable choice for the trainnig due to its exploration and exploitaion properties and solves the gradient-based drawbacks, however it sufferes from the mutation problem leading to premature convergences and needs more time to converge to an optimum solution comparing to the particle swarm optimization. As we discuss about the properties of PSO later, it has been shown that PSO is a better evolutionary candidate for optimization (Eberhart, and Shi, 1998). The PSO algorithm posseses imprtant chractristcs such as memory and costructive cooperation among the individuals that can prevent mutation problem exitst in GA. Different variations of PSO have been applied to train the feed-forward ANNs for non-linear function approximation and classification problems. The training of neural networks is achieved basically in two ways:

1- Adjusting the connection weights when the ANN structure is predefined such as the number of hidden layers, the number of neurons and their connections, and activation function parameters.
2- Evolving a ANN structure which is not predefined and adjusting the weights simultenously.

Training of fixed structure ANN has been experimented by basic PSO method (Mendes et al., 2002). In this study, it has been shown PSO's performance is competitive to BP methods and especially in some problems where the number of local minima is high.

Also, the variant of PSO with minimum velocity constraint was proposed and tested for function approximations using feed-forward ANNs (Xiaorong et al., 2007). Applying the velocity constrains reduces the premature convergences and alleviates the effect of dimentionality increase. This is done by guiding the particle in the search space by limiting the maximum moving distance in each iteration. Thus, it prevents the particle to go out of the bound (search space) or to stop when the velocity increases or decreases. The very effective modifications focusing on optimizing the update equations of PSO were made in (Russ et al., 2000), (Kennedy, 2000). These modifications are adding the inertia weight and improving the PSO performance with cluster analysis. Using cluster analysis methods, the update equations are modified in a fashion that particle attempt to merge to the center of their cluster instead of merging to the global best location. This approach improves the performaces in some classes of problems. In (Angelin,1999), a selection mechanism was proposed for PSO similat to that already used in genetic algorithm to improve the quality of the particles in a swarm. Another modified PSO is the cooperative learning proposed in (Van den Bergh & Engelbrecht, 2004). The application of this method to neural network training has yield promising results. In this approach, input vectors are distributed into several sub-vectors which are optimized in their own swarms cooperatively. Performace improvment in this case are due to splitting the main vector into several sub-vector that in turn results in better credit assignments and reduces the chance to omit a possible good solution for a certain componet in the vector.

Training of ANNs by Multi-Phase PSO (MPPSO) is another variation which evolves simeltiously multiple groups of particles that change the direction of search in different phases of the algorithm (Al-kazemi & Mohan, 2002). Each particle in this method is in a specific group and phase at a given time. MPPSO boosts the wider exploration of the search space, increases pupolation diversity and prevents premature convergences. Furthermore, MPPSO has different update equations comparing to the basic PSO and permits changes to the locations of the particle that only lead to some improvements. PSO also has been used as a means to evolve ANN architectures (Chunkai et al., 2000). In this study, the network structure is adaptively adjusted and the PSO algorithm is applied to evoles the nodes of the neural network with specific generated structure. The techniques such as the combination of partial training and evolving added nodes are employed to generate the desired architecture and then PSO is used to evolve the nodes of the pre-defined structure. Hybrid of genetic algorithm and particle swarm optimization (HGAPSO) is another modified PSO that was employed to design recurrent neural networks [Juang, 2004]. In HGAPSO method, the individuals of the next generation are created not only by crossover and mutation operators but also by PSO. The upper-half of the best-performing individuals in a population are ehanced using PSO and the other half is generated by applying the crossover and mutations. Unlike GA, HGAPSO removes the restrictions of evolving the individuals within the same generation. In this article, the proposed method is another variation of particle swarm optimization for fixed structure ANNs where only weights are adjusted.

## 3. Particle Swarm Optimization

The Particle Swarm Optimization algorithm is represented by the evolution of a population in the form of an $n$-dimensional vector $x$=($x_1$,…,$x_n$) ,$i$=1,..,$n$. These particles represent an approximation of the desired solution, and the number of dimensions depends on a given problem. Each particle has a memory $p^i$ , $i$=1,..$m$ where $m$ is the number of particles) which keeps the best location that $i$th particle has found since its search started. Furthermore, every flying particle has a velocity $v^i(t)$ that shows its direction and speed at the time instance $t$. In each iteration, particle locations and velocities are updated according to equations (1), and (2). The global best location, $p^g$, found by any particle, and local best location, $p^L{}_i$, found by neighbors of the $i$th particle, are the two elements of shared information in the entire population. To evaluate each particle's performance, a fitness function is defined. There are two types of PSO, global and local (Bergh & Engelbrecht, 2002). The local version of PSO that is proven experimentally to be able to find the global optimum is shown by equation

(3). This method is computationally extensive since for each particle a neighborhood of size *k* is identified in each iteration as shown in Figure 1.

$$v^i(t+1)=w.v^i(t)+c_1.r_1(p^i-x^i(t))+c_2.r_2(p^g-x^i(t)) \quad , \quad i=1,\ldots,m \tag{1}$$
$$x^i(t+1)=x^i(t)+v^i(t+1) \tag{2}$$

In the local version of PSO, the equation (1) is changed to (3).

$$v^i(t+1)=w.\ v^i(t)+c_1.r_1(p^i-x^i(t))+c_2.r_2(p^g-x^i(t))+c_3.r_3(p^{Li}-x^i(t)) \quad , \quad i=1,\ldots,m \tag{3}$$

The initial values of positions and velocities are calculated for each particle by the equations (4), and (5).

$$v^i(0)=v_{min}+rand(v_{max}-v_{min})\ , \quad i=1,..,m \tag{4}$$
$$x^i(0)=x_{min}+rand(x_{max}-x_{min})\ , \quad i=1,..,m \tag{5}$$

In equation (1), $r_1, r_2$ are two random vectors with values ranging from zero to one. The inertia $w$ is a predefined positive value that is decreased in each iteration to slow down the speed of particles which are closing gradually to the global best particle (Shi & Eberhart, 1998). As a result, this parameter gives more chance to particles to explore the search space and bound the increase of velocity. The expression $c_1.r_1(p^i-x^i(t))$ in equation (1) is the particle memory influence which indicates the scale that a particle relies on its own best past experience. Also, the expression $c_2.r_2(p^g-x^i(t))$ in equation (1) is swarm influence indicating the degree that a particle follows the best experience of the entire population or the local group which is shown the expression $c_3.r_3(p^{Li}-x^i(t))$ in equation (3). The three confidence measures which are self-confidence, swarm and local-group confidences are denoted by $c_1$, $c_2$ and $c_3$ respectively. These are positive constant values ranging from 1.5 to 2.5. The inertia value is usually chosen from 0.4 to 1.4 (Shi & Eberhart, 1998). Schematics for the equation (1), (2) are shown in Figure 2.

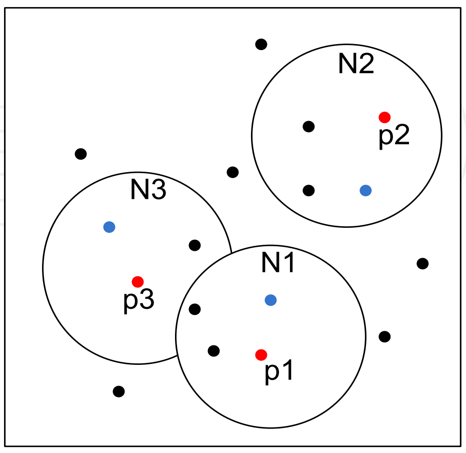

Fig. 1. Neighborhoods of size 4, $N_i$ standing for neighborhood and $P_i$ for particles

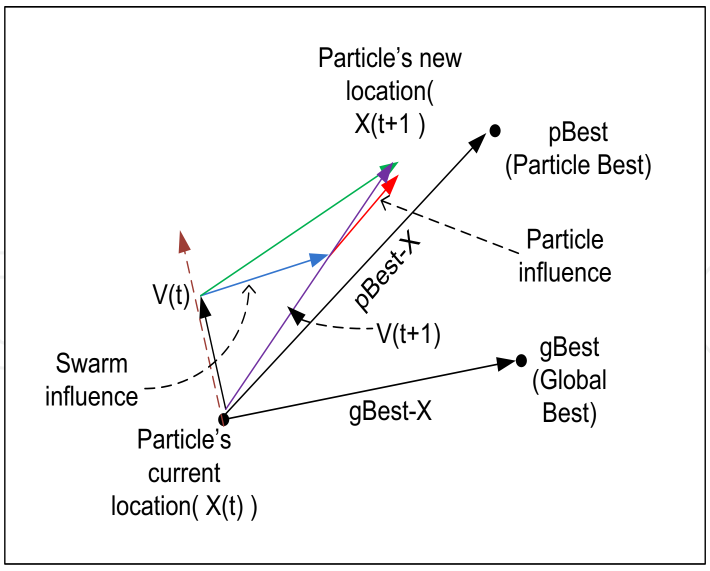

Fig. 2. The equation (1), (2) in the form of vector representation

The steps of PSO algorithm can be defined as the following:

1- Representing the primary solution of a given problem in the form of n-dimensional vectors.
2- Defining a boundary for the search space and maximum velocity.
3- Defining a fitness function to evaluate the quality of each particle.
4- Generating a population consist of m particles represented in the form of n-dimensional vectors and locating them randomly in the search space.
5- Updating the position and velocities of the particle by the equation (1),(2).
6- Evaluating the fitness of each particle
7- Updating the best experience or memory of each particle and the global best particle.
8- Repeating from the step 5 until the desired solution is achieved.

PSO is comparable to Evolutionary Computation (EC) methods namely genetic Algorithm (GA) since it is a stochastic population based method. In PSO the information is shared among the population by global best and local best particles whereas in GA the crossover operator performs the same function. The random vectors $r_1$, $r_2$ in the equation (1) also are similar to the mutation operator in the GA. However, unlike GA that discards the individuals with lower fitness, in PSO all particles are kept and transferred to the next generation. In addition, each particle has memory, whereas in GA the best individual is kept in each generation.

## 4. Methodology

Training of neural networks can generally be considered as modification of the randomly generated weights of pre-defined ANNs that comprise of a certain number of inputs, outputs, and neurons in different hidden layers. The weights are changed until the difference between the actual ANNs outputs corresponding to input (samples or training set) and the desired output reaches a certain error. Therefore, the training of ANNs can be

interpreted as constructing a model (function) and optimizing it in an n-dimensional space. That is, we attempt to optimize the empirical error by training in the search space.
This optimization problem can be solved by PSO, a stochastic global optimization method suitable for non-linear function optimization. We consider the entire ANN as a particle in a *D*- dimensional space. In other words, for instance, if an ANN has *x* inputs, *y* outputs and $n_1$, $n_2$, $n_3$ neurons in its three hidden layers respectively, then the number of dimensions(weights) for the particles is $d=x\times n_1+n_1\times n_2+ n_2\times n_{3,}+ n_3\times y$.
Using this method, we create a population of particles that represent the weights of different an ANN. The other component of this optimization to be defined is the fitness function. A fitness function definition depends on the problem we aim to solve. In our experiment for classification, a fitness function is defined as feeding the entire training set to the ANN (one epoch) and adding up the number of correct classifications. To interpret the fitness values of particles, we assume that a particle with higher fitness value has less misclassification rates since the ANN weights have been represented as the dimensions of the particles.
In this variation of PSO the directions of flying particles are recorded at the beginning as positive or negative. When a particle reaches the maximum value of search space boundary, the particle position is reset to a random coordinate approximately in the middle of search space and its flying direction at this time will be changed to the opposite of that particle's original direction. In addition, if a particle's velocity reaches its maximum value and its coordinate is still within the search space, the particle's velocity is set to minimum value and its direction again is changed to the opposite direction. This updating method enables particles to explore the search space more thoroughly by experiencing broader ranges of possible values for both speed and location.

## 5. Results and discussion

We conducted four classification experiments using feed-forward ANNs with PSO based training. The two main parts of these experiments, PSO and the training of ANNs have been implemented in C++. The data sets were chosen from the UCI repository (Asuncion & Newman, 2007). The four data sets are Iris, Wine, SPECT heart and Ionosphere. The parameters values of ANNs and PSO used in the experiments are shown in Table 1. The information of attributes, training and validation data sets and target attributes (classes) is shown in Table 2. Since we are dealing with the two parameterized methods, PSO and ANNs, it is not feasible to set identical values for the parameters in every experiment because the type of problem that is solved greatly influences the values chosen for the parameters.

| PSO parameters | | | | |
|---|---|---|---|---|
| Dataset name | Iris | Wine | SPECT Heart | Ionosphere |
| Confidence factor ($c_1$, $c_2$) | 1.7,1.7 | 1.7,1.7 | 1.7,1.7 | 1.7,1.7 |
| Inertia(w) | 0 | 0.9 | 0.9 | 0.9 |
| maximum speed step (vmax) | 1.0 | 1.2 | 1.2 | 1.2 |
| Maximum value for each dimension (max-dim ) | 3.0 | 4.0 | 4.0 | 4.0 |
| Iteration(T) | 5000 | 1000 | 10000 | 1000 |
| Particles(p) | 20 | 30 | 20 | 30 |
| MLP parameters | | | | |
| Hidden layers | 2 | 2 | 2 | 2 |
| number of neurons in input, hidden and output layers respectively | 4,40,28,1 | 13,15,10,1 | 22,30,5,1 | 34,15,10,1 |

Table 1. The parameters of PSO and Neural Networks used in the experiments

| Dataset | Number of attributes /classes | Number of records in training/validation datasets | Number of correct classification | Accuracy MLP trained by PSO | Accuracy other methods |
|---|---|---|---|---|---|
| Iris | 4/3 | 150/15 | 148 | **98.66%** | 85% - 97.77% |
| Wine | 13/3 | 178/18 | 177 | **99.44%** | 96.1%- 99.4% |
| SPECT Heart | 22/2 | 80/187 | 172 | **91.97%** | 80% - 90.7% |
| Ionosphere | 34/2 | 351/35 | 328 | 93.42% | 90.7% - 96.7% |

Table 2. The specifications of the data sets and the result of experiments

With the exception of the SPECT heart data set that contains a validation data set of 187 records, the rest of the data sets do not have any validation data sets. Therefore, we applied *k*-fold cross-validation method (*k*=10) for those data sets (Haykin, 1994). In *k*-fold cross-validation a data set is divided into *k* equal partitions where k-1 partitions are used as the training set and the remaining partition is used as a validation data set. The procedure is repeated *k* times with different partitions being used as the validation set each time, and the sum squared errors in *k* validations is considered as the final error for the model. The results of 5 trainings from the total 10 trainings on the four data sets, Iris, Wine, SPECT heart and Ionosphere have been shown in the Tables 3-7. To avoid showing several graphs of the trainings, we have chosen only one graph of the ten trainings in each experiment, however, the related data is completely represented in the Tables 3-7.

In addition, to avoid recording unnecessary data, we ignored the fitness values that were not changed and were repeated in the intervals of iterations. Therefore, fitness values were recorded only when they were improved. The empty entries in the tables below show that after certain iterations the fitness had never improved until the maximum iteration number was reached. The data represented in Tables 3, 4, 6 can be useful to evaluate how fast and efficiently ANNs are trained. Moreover, by comparing the last fitness value of the trainings, shown in the mentioned tables, to their corresponding validation results shown in Table 5, we observe that a satisfactory result in the training does not provide a good result in the validation result and vice versa.

| Training 5 | | Training 6 | | Training 7 | | Training 8 | | Training 9 | | Training 10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | FIT | ITE | FIT | ITE | FIT | ITE | FIT | ITE | FIT | ITE | FIT |
| 0 | 45 | 0 | 53 | 0 | 48 | 0 | 86 | 0 | 45 | 0 | 46 |
| 7 | 58 | 1 | 71 | 4 | 50 | 23 | 89 | 1 | 46 | 3 | 93 |
| 8 | 61 | 5 | 90 | 5 | 60 | 24 | 90 | 5 | 64 | 13 | 104 |
| 10 | 80 | 30 | 97 | 6 | 68 | 36 | 121 | 17 | 76 | 32 | 105 |
| 28 | 82 | 203 | 110 | 8 | 90 | 180 | 127 | 24 | 90 | 49 | 106 |
| 33 | 88 | 210 | 122 | 30 | 94 | 191 | 128 | 365 | 91 | 244 | 109 |
| 101 | 89 | 304 | 126 | 72 | 107 | 254 | 129 | 378 | 93 | 252 | 118 |
| 116 | 90 | 532 | 128 | 190 | 122 | 1707 | 130 | 452 | 94 | 466 | 121 |
| 368 | 91 | 540 | 129 | 210 | 124 | 4862 | 131 | 529 | 110 | 522 | 129 |
| 521 | 93 | 544 | 130 | 217 | 125 | 4963 | 132 | 754 | 120 | 853 | 132 |
| 604 | 104 | 573 | 131 | 584 | 126 | | | 872 | 127 | 922 | 133 |
| 745 | 116 | 1165 | 132 | 639 | 129 | | | 2012 | 128 | 1393 | 134 |
| 1066 | 126 | | | 735 | 130 | | | 3230 | 131 | | |
| 2034 | 128 | | | 2374 | 131 | | | | | | |
| 2035 | 129 | | | | | | | | | | |
| 2043 | 130 | | | | | | | | | | |
| 3343 | 131 | | | | | | | | | | |

Table 3. The last 5 training results on Iris data set - ITE denotes the number of iterations and FIT stands for fitness.

One of the challenges in this experiment is that changing the parameter of PSO influences the performance of ANNs and vice versa. For instance, increasing the number of particles demands more fitness evaluation of particles and changing the number of hidden layers and their neurons adds more dimensions to the particles. In addition, it also slows down the performances. Therefore, there need to be some tradeoffs. The higher dimension we set, the more iterations are required to obtain a given accuracy. On the other hand, increasing the number of iterations adds more computational load. Other difficulties arise when the number of samples and inputs of ANNs increase. In other word, increasing the number of records and attributes in a data set increases the training time.

| Training 1 | | Training 2 | | Training 3 | | Training 4 | | Training 5 | |
|---|---|---|---|---|---|---|---|---|---|
| ITE | FIT | ITE | FIT | ITE | FIT | ITE | FIT | ITE | FIT |
| 0 | 108 | 0 | 108 | 0 | 76 | 0 | 92 | 0 | 96 |
| 13 | 114 | 2 | 115 | 2 | 93 | 1 | 133 | 3 | 99 |
| 50 | 140 | 8 | 121 | 5 | 107 | 229 | 144 | 48 | 100 |
| 65 | 141 | 14 | 136 | 20 | 111 | | | 90 | 102 |
| 96 | 145 | 66 | 143 | 24 | 114 | | | 91 | 104 |
| 168 | 146 | 143 | 149 | 29 | 117 | | | 98 | 111 |
| 210 | 148 | | | 44 | 130 | | | 103 | 114 |
| 311 | 149 | | | 51 | 140 | | | 105 | 115 |
| 334 | 150 | | | 74 | 150 | | | 107 | 138 |
| 500 | 151 | | | 233 | 151 | | | 193 | 140 |
| 550 | 152 | | | | | | | 450 | 147 |
| 802 | 153 | | | | | | | | |
| 952 | 154 | | | | | | | | |
| 966 | 155 | | | | | | | | |

Table 4. The first 5 training results on Wine data set. - ITE denotes the number of iterations and FIT signifies the fitness

The other issue is the outputs chosen for the ANN. At the first glance, it might appear that the number of outputs should be equal to the number of classes. However, based on our experiment, we realized that this might not be the case necessarily. Therefore, we have chosen one output for ANN in all experiments and the activation function of the output neuron is set to a linear function.

| Validations | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Iris (Out of 15) | 14 | 15 | 15 | 14 | 15 | 15 | 15 | 15 | 15 | 15 |
| Wine (Out of 18) | 18 | 18 | 18 | 18 | 17 | 18 | 18 | 18 | 18 | 18 |
| Ionosphere (Out of 35) | 35 | 31 | 30 | 31 | 35 | 34 | 31 | 35 | 35 | 35 |

Table 5. Validation results (number of correct classification) on Iris, Wine and Ionosphere data sets according to 5 trainings shown in tables 3,4,6.

| Training 2 | | Training 3 | | Training 4 | | Training 5 | | Training 10 | |
|---|---|---|---|---|---|---|---|---|---|
| ITE | FIT | ITE | FIT | ITE | FIT | ITE | FIT | ITE | FIT |
| 0 | 244 | 0 | 234 | 0 | 225 | 0 | 225 | 0 | 219 |
| 2 | 254 | 4 | 240 | 1 | 245 | 1 | 234 | 7 | 232 |
| 5 | 263 | 8 | 241 | 4 | 248 | 2 | 244 | 10 | 242 |
| 70 | 264 | 9 | 250 | 17 | 251 | 6 | 248 | 13 | 244 |
| 72 | 266 | 25 | 251 | 39 | 258 | 8 | 254 | 19 | 256 |
| 86 | 270 | 27 | 260 | 46 | 274 | 49 | 270 | 45 | 257 |
| 87 | 273 | 50 | 273 | 72 | 275 | 335 | 272 | 52 | 262 |
| 115 | 274 | 53 | 274 | 75 | 278 | 371 | 275 | 63 | 264 |
| 124 | 279 | 68 | 276 | 79 | 279 | 415 | 277 | 65 | 265 |
| 158 | 280 | 78 | 277 | 83 | 281 | 556 | 278 | 71 | 266 |
| 180 | 283 | 83 | 278 | 700 | 287 | 662 | 279 | 85 | 269 |
| 181 | 284 | 84 | 280 | 802 | 290 | | | 119 | 270 |
| 280 | 286 | 202 | 281 | 990 | 291 | | | 122 | 271 |
| 311 | 287 | | | | | | | 183 | 272 |
| 318 | 288 | | | | | | | 211 | 274 |
| | | | | | | | | 217 | 276 |
| | | | | | | | | 224 | 278 |
| | | | | | | | | 226 | 280 |
| | | | | | | | | 286 | 281 |

Table 6. The training results of 2-5 and 10 on Ionosphere data set - ITE denotes the number of iterations and FIT signifies the fitness

By applying inertia parameter $w$ in equation (1) the optimum point is reached faster. This is evident from the results of training shown in Tables 3 and 4 where the inertia parameter was not used for the trainings with Iris data set.

| Iteration | 0 | 1 | 3 | 8 | 21 | 28 | 37 | 133 | 218 | 254 | 556 | 1014 | 2659 | 3634 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Fitness | 40 | 42 | 47 | 50 | 54 | 57 | 59 | 61 | 63 | 64 | 65 | 66 | 67 | 68 |

Table 7. The result of training on 80 records of the SPECT heat dataset. The validation result is 172 records out of 187 which is equal to ,15 misclassification.

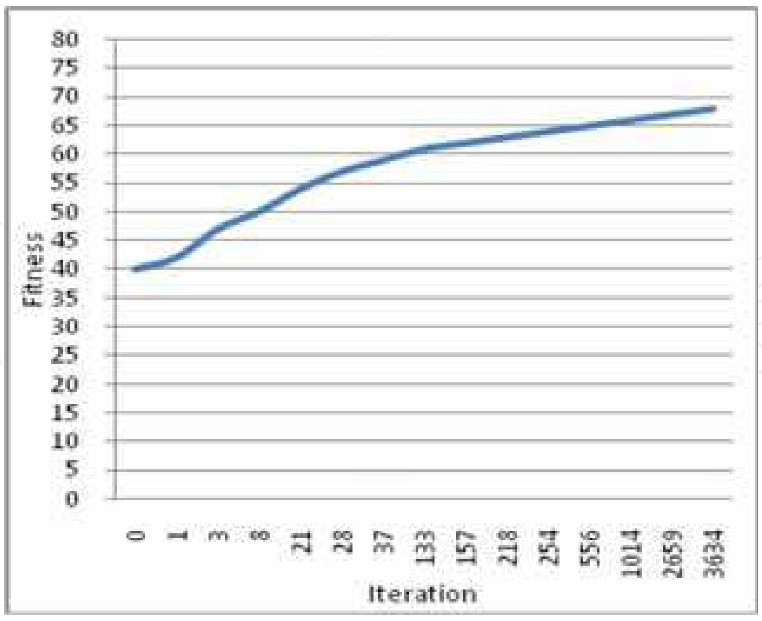

Fig. 3. Training on SPECT Heart

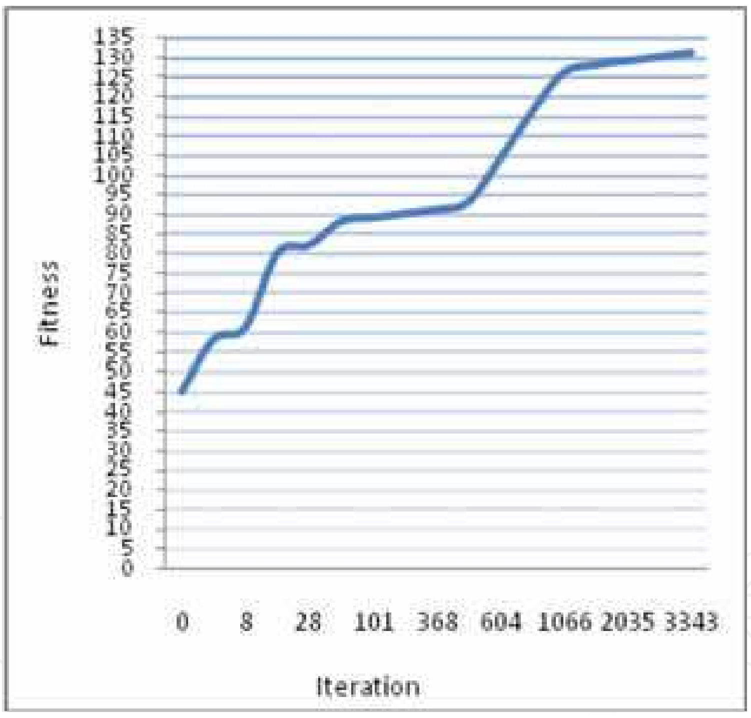

Fig. 4. Training on the 5th partition of Iris.

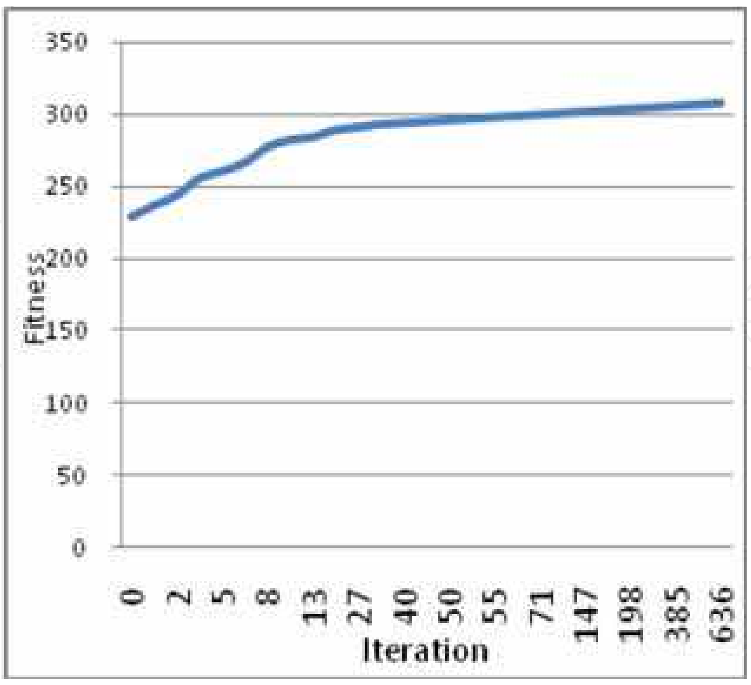

Fig. 5. Training on the 1st partition of Ionosphere

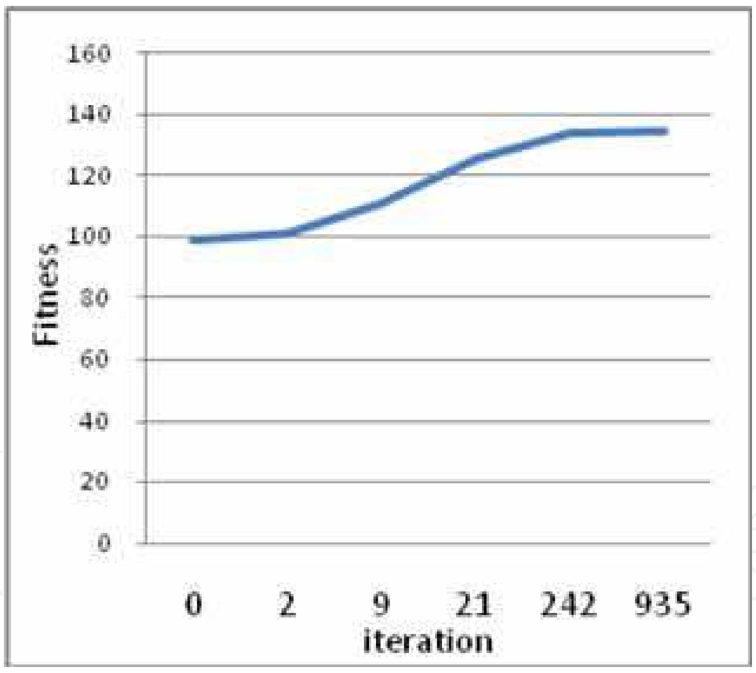

Fig. 6. Training on the 7th partition of Wine

## 6. Conclusion

PSO is a heuristic optimization method that performs well for different optimization problems. As with other optimization methods, it has not been proven that this method can always find the global optimum. However, it fulfills the exploitation and exploration of a search space. It is claimed that the local version of the method is able to find global optimum. In the simulation, we experimented with training of feed-forward ANNs and demonstrated both the efficiency of this method and its competitiveness with other ANNs training methods. The proposed approach outperformed some of the previous results in our experiments. Moreover, the application of this method to train ANNs is limited by the structure of given ANNs. This means that with high numbers of inputs and neurons in the hidden layers, particle dimensions are increased and consequently training time rises. However, based on the properties of this method, it is possible to decrease the training time by parallel implementation. The feasibility of parallel implementation is a very important advantage of this method over other common training methods. Lastly, this method can be employed for training of ANNs with different topologies such as recurrent neural network where the gradient-based training methods are not suitable choices.

## 7. References

Al-kazemi, B. & Mohan, CK (2002). Training feedforward neural networks using multi-phase particle swarm optimization, *Proceedings of the 9th International Conference on*, pp. 2615- 2619, 981-04-7524-1, USA, Nov. 2002, USA, IEEE, New York.

Angelin P. J.,(1999).Using selection to improve particle swarm optimization, *Proceedings of IJCNN'99*, 0-7803-4869-9, pp. 84-89, USA, July 1999, IEEE,Anchorage.

Asuncion A. & Newman D. J. (2007). UCI Machine Learning Repository, *Univ. of California, Irvine, School of Information and Computer Sciences*, 2007. [Online].available: http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html.

Bergh, F. & Engelbrecht, A. (2002). A New Locally Convergent Particle Swarm Optimizer, *Proceedings of the IEEE International Conference on Conference on Systems, Man and Cybernetics*, pp. 96-101, 0-7803-7437-1, Oct. 2002, IEEE.

Chunkai, Z., Yu, L. & Huihe, S. (2000). A new evolved artificial neural network and its application, *Proceedings of the 3rd World Congress on*, pp. 1065-1068, 0-7803-5995-X, June 2000, China, IEEE, Hefei.

Eberhart, R. C. & Kennedy, J. (1995). A new optimizer using particle swarm theory, *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39-43, 0-7803-2676-8, Japan, Oct 1995, IEEE, Nagoya.

Eberhart, R. C. & Shi, Y. (1998). Comparison between genetic algorithms and particle swarm optimization, *Proceedings of the 7th International Conference on Evolutionary Programming VII*, PP. 611-616, 3-540-64891-7, UK, 1998, Springer-Verlag, London.

Fahlman, S. E. (1988). Faster-learning variations on back-propagation: an empirical study. *Proc. 1988 Connectionist Models Summer School*. D. S. Touretzky, G. E. Hinton and T. J. Sejnowski, (eds), Morgan Kaufmann, San Mateo, CA, 1988, pp. 38-51.

Haykin S. (1994). *Neural networks, a comprehensive foundation*, Prentice Hall PTR , NJ, USA.

Hecht-Nelson, R. (1989). Theory of the backpropagation neural network. *Proceedings of International Joint Conference on Neural Netowrk*, vol. 1, pp. 593-605, Jun. 1989.

Juang, C. F. (2004). A hybrid of genetic algorithm and particle swarm optimization for recurrent network design, *Systems, Man, and Cybernetics*, vol. 34, no. 2, pp. 997-1006, April 2004, IEEE.

Kennedy, J. (2000). Stereotyping: Improving particle swarm optimization performance with cluster analysis, *Proceedings of the 2000 Congress on Evolutionary Computing*, pp. 1507-1512, USA, July 2000, IEEE, La Jolla.

Mendes, R., Cortez, P., Rocha, M. & Neves, J.,(2002). Particle swarms for feedforward neural network training, *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, pp. 1895-1899, 0-7803-7278-6, USA, May 2002, IEEE, Honolulu.

Mitchell, M., *Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA (1996).

Riedmiller M. & Braun H., "A direct adaptive method for faster backpropagation learning: The rprop algorithm," in *Proc. IEEE Int. Conf. Neural Networks*, San Francisco, CA, Apr. 1993.

Pu X., Z. Fang Z. & Liu Y. (2007). Multilayer perceptron networks training using particle swarm optimization with minimum velocity, *Proceedings of the 4th international symposium on Neural Networks: Advances in Neural Networks*, pp.237-245, 978-3-540-72394-3, China, 2007, Springer Berlin/Heidelberg, Nanjing.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model, *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 25-34, USA, July 1987, ACM, New York.

Russ C., & Eberhart, Y. Shi. (2000). Comparing inertia weights and construction factors in particle swarm optimization, *Proceedings of the 2000 Congress on Evolutionary Computing*, pp. 84-88, USA, July 2000, IEEE, La Jolla.

a) Shi, Y. & Eberhart, R. C. (1998). Parameter selection in particle swarm optimization, *Proceedings of the 7th International Conference on Evolutionary Programming VII*, pp. 591-600, 1998, UK, Springer-Verlag, London.

b) Shi, Y. & Eberhart, R. C. (1998). A modified particle swarm optimizer, *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp.69-73, May 1998, USA, IEEE, Anchorage.

Van den Bergh, F. & Engelbrecht, AP (2004). A Cooperative approach to particle swarm optimization, Evolutionary Computation, vol.8, No.3, (June 2004)(225-239), 1089-778X.

ISBN 978-953-7619-28-2
Hard cover, 348 pages
**Publisher** InTech
**Published online** 01, February, 2010
**Published in print edition** February, 2010