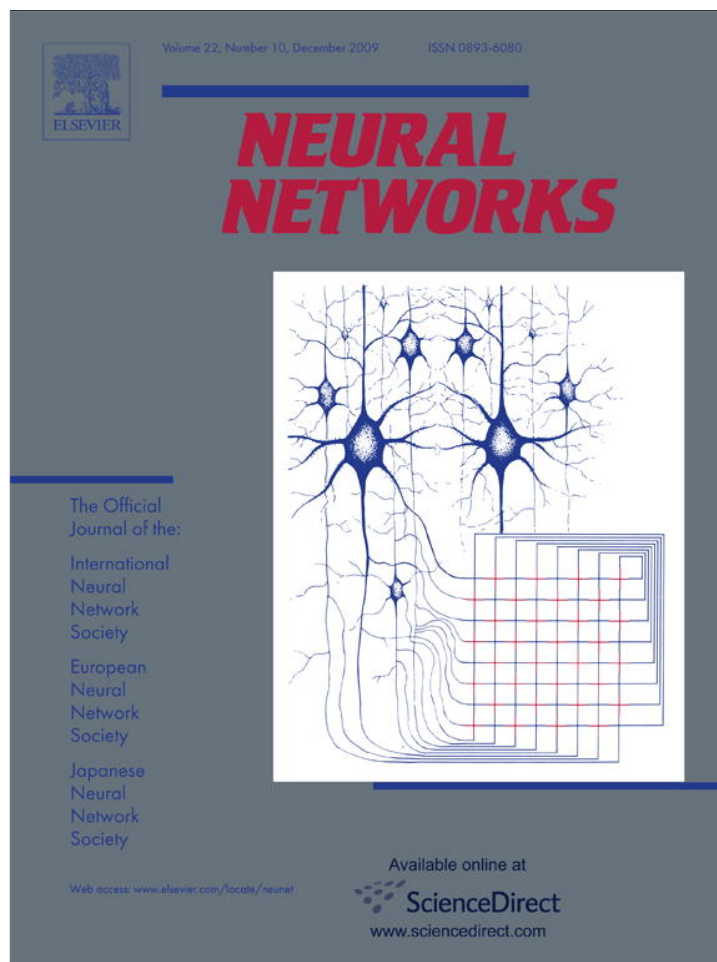


Provided for non-commercial research and education use.
Not for reproduction, distribution or commercial use.



This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

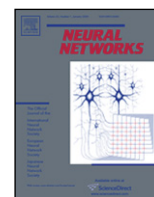
In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier's archiving and manuscript policies are encouraged to visit:

<http://www.elsevier.com/copyright>



Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Evolutionary artificial neural networks by multi-dimensional particle swarm optimization

Serkan Kiranyaz^{a,*}, Turker Ince^b, Alper Yildirim^c, Moncef Gabbouj^a

^a Tampere University of Technology, Tampere, Finland

^b Izmir University of Economics, Izmir, Turkey

^c TUBITAK, Ankara, Turkey

ARTICLE INFO

Article history:

Received 17 July 2008

Received in revised form 28 April 2009

Accepted 31 May 2009

Keywords:

Particle swarm optimization

Multi-dimensional search

Evolutionary artificial neural networks and multi-layer perceptrons

ABSTRACT

In this paper, we propose a novel technique for the automatic design of Artificial Neural Networks (ANNs) by evolving to the optimal network configuration(s) within an architecture space. It is entirely based on a multi-dimensional Particle Swarm Optimization (MD PSO) technique, which re-forms the native structure of swarm particles in such a way that they can make inter-dimensional passes with a dedicated dimensional PSO process. Therefore, in a multidimensional search space where the optimum dimension is unknown, swarm particles can seek both positional and dimensional optima. This eventually removes the necessity of setting a fixed dimension *a priori*, which is a common drawback for the family of swarm optimizers. With the proper encoding of the network configurations and parameters into particles, MD PSO can then seek the positional optimum in the error space and the dimensional optimum in the architecture space. The optimum dimension converged at the end of a MD PSO process corresponds to a unique ANN configuration where the network parameters (connections, weights and biases) can then be resolved from the positional optimum reached on that dimension. In addition to this, the proposed technique generates a ranked list of network configurations, from the best to the worst. This is indeed a crucial piece of information, indicating what potential configurations can be alternatives to the best one, and which configurations should not be used at all for a particular problem. In this study, the architecture space is defined over feed-forward, fully-connected ANNs so as to use the conventional techniques such as back-propagation and some other evolutionary methods in this field. The proposed technique is applied over the most challenging synthetic problems to test its optimality on evolving networks and over the benchmark problems to test its generalization capability as well as to make comparative evaluations with the several competing techniques. The experimental results show that the MD PSO evolves to optimum or near-optimum networks in general and has a superior generalization capability. Furthermore, the MD PSO naturally favors a low-dimension solution when it exhibits a competitive performance with a high dimension counterpart and such a native tendency eventually yields the evolution process to the compact network configurations in the architecture space rather than the complex ones, as long as the optimality prevails.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

The Artificial Neural Networks (ANNs) are known as the “universal approximators” and “computational models” with particular characteristics such as the ability to learn or adapt, to organize or to generalize data. Up-to-date designing a (near) optimal network architecture is made by a human expert and requires a tedious trial and error process. Especially automatic determination

* Corresponding address: Tampere University of Technology, Signal Processing Department, Tampere, Finland.

E-mail address: serkan@cs.tut.fi (S. Kiranyaz).

of the optimal number of hidden layers and hidden nodes in each (hidden) layer is the most critical task. For instance, an ANN with no or too few hidden nodes may not differentiate among complex patterns, instead leading to only a linear estimation of such – possibly non-linear – problem. In contrast, if the ANN has too many nodes/layers, it might be affected severely by the noise in data due to over-parameterization, which eventually leads to a poor generalization. On such complex networks proper training can also be highly time-consuming. The optimum number of hidden nodes/layers might depend on input/output vector sizes, training and test data sizes, more importantly the characteristics of the problem, e.g. its non-linearity, dynamic nature, etc.

Several researchers have attempted to design ANNs automatically with respect to a particular problem. The earlier attempts

fall into two broad categories: constructive and pruning algorithms (Burgess, 1994; Frean, 1990; LeCun, Denker, & Solla, 1990; Reed, 1993), where many deficiencies and limitations have been reported (Angeline, Sauders, & Pollack, 1994). Efforts have been then focused on evolutionary algorithms (EAs) (Back & Schwefel, 1993) particularly over Genetic Algorithm (GA) (Goldberg, 1989) and Evolutionary Programming (EP), (Fayyad, Shapire, Smyth, & Uthurusamy, 1996) for both training and evolving ANNs. Most GA based methods have also been found quite inadequate for evolving ANNs mainly due to two major problems: permutation problem and noisy fitness evaluation (Yao & Liu, 1997). Although EP-based methods (Angeline et al., 1994; Yao & Liu, 1997; Yao, 1999), might address such problems, they usually suffer from their high complexity and strict parameter dependence.

Conceptually speaking, Particle Swarm Optimization (PSO) (Esquivel & Coello Coello, 2003; Kennedy & Eberhart, 1995; Omran, Salman, & Engelbrecht, 2006), which has obvious ties with the EA family, lies somewhere in between GA and EP. Yet unlike GA, PSO has no complicated evolutionary operators such as *crossover*, *selection* and *mutation* and it is highly dependent on stochastic processes. PSO has been successfully applied for training feed-forward (Carvalho & Ludermir, 2007; Meissner, Schmucker, & Schneider, 2006; Yu, Xi, & Wang, 2007; Zhang & Shao, 2000) and recurrent ANNs (Salerno, 1997; Settles, Rodebaugh, & Soule, 2003) and several works on this field have shown that it can achieve a superior learning ability to the traditional Back-Propagation (BP) method in terms of accuracy and speed. Only few researchers have investigated the use of PSO for evolutionary design of ANNs or to be precise, the fully connected feed-forward ANNs, the multi-layer perceptrons (MLPs) only with a single hidden layer. In Carvalho and Ludermir (2007) and Zhang and Shao (2000) the PSO–PSO algorithm and its slightly modified variant, PSO–PSO: Weight Decay (PSO–PSO:WD) have been proposed. Both techniques use an inner PSO to train the weights and an outer one to determine the (optimal) number of hidden nodes. Both methods perform worse classification performance than EP and GA-based Evolutionary Neural Networks (ENNs) on three benchmark problems from *Proben1* dataset (Prechelt, 1994). Recently, Yu et al. proposed an improved PSO technique, the so-called IPSONet (Yu et al., 2007), which achieved a comparable performance in terms of average classification error rate over the same dataset in *Proben1*. All potential network architectures have been encoded into the particles of a single PSO operation, which evaluates their weights and architecture simultaneously. However, such a *all-in-one* encoding scheme makes the dimension of particles too high and thus the method can be applied to only single hidden layer MLPs with a limited number of (hidden) nodes (i.e. a maximum of 7 was used in Yu et al. (2007)). Furthermore, it turns out to be a hybrid technique, which uses GA operators such as *mutation* and *crossover* in order to alleviate the stagnation problem of PSO on such high dimensions.

The major drawback of many PSO variants including the basic method is that they can only be applied to a search space with fixed dimensions. However, in the field of ANNs as well as in many of the optimization problems (e.g. clustering, spatial segmentation, optimization of the dimensional functions, etc.), the optimum dimension where the optimum solution lies, is also unknown and should thus be determined within the PSO process. In order to address this problem, in this paper we first present a multi-dimensional PSO (MD PSO) technique, which negates the need of fixing the dimension of the solution space in advance. We then adapt MD PSO technique for designing (near-) optimal ANNs. In order to make comparative evaluations, the focus is particularly drawn on automatic design of the feed-forward ANNs and the search is carried out over all possible network configurations within the specified architecture space. Therefore, no assumption is made about the number of (hidden)

layers and in fact none of the network properties (e.g. feed-forward or not, differentiable activation function or not, etc.) is an inherent constraint of the proposed scheme. As long as the potential network configurations are transformed into a hash (dimension) table with a proper hash function where indices represent the solution space dimensions of the particles, the MD PSO can then seek both positional and dimensional optima in an interleaved PSO process. The optimum dimension found naturally corresponds to a distinct ANN architecture where the network parameters (connections, weights and biases) can be resolved from the positional optimum reached on that dimension. One important advantage of the proposed approach to other evolutionary methods is that at the end of a MD PSO evolution process, apart from the best solution achieved on the optimum dimension, 2nd, 3rd, etc. ranked solutions, which corresponds to the 2nd, 3rd, etc. best network configurations can also be obtained within the architecture space. This further indicates other potential ANN configurations that might be convenient to use for the particular problem encountered. For example the best solution might correspond to a highly complex network whereas an acceptable performance can also be achieved by a much simpler one, say in the 3rd rank with a slight performance loss. In addition, the worst architecture(s) will also be visible, indicating what ANN configurations should not be used at all for the same problem.

The rest of the paper is organized as follows. Section 2 surveys the related work on ENNs and PSO. MD PSO and the proposed application over the automatic ANN design are presented in detail in Section 3. Section 4 provides the experiments conducted over synthetic and real datasets and discusses the results. Finally, Section 5 concludes the paper and proposes topics for future research.

2. Related work

2.1. Evolutionary ANNs

After the introduction of simplified neurons by McCulloch and Pitts in 1943 (McCulloch & Pitts, 1943), ANNs have been applied widely to many application areas, most of which use feed-forward ANNs with the Back Propagation (BP) training algorithm. BP has the advantage of the directed search, in that weights are always updated in such a way that minimizes the error. However, there are several aspects, which make the algorithm not guaranteed to be universally useful. Most troublesome is its strict dependency to a learning rate parameter, which, if not set properly, can either lead to oscillation or an indefinitely long training time. Network paralysis might also occur, i.e. as the ANN trains, the weights tend to be quite large values and the training process can come to a virtual standstill. Furthermore, BP eventually slows down by an order of magnitude for every extra (hidden) layer added to ANN. Above all; the BP is just a gradient descent algorithm on the error space, which can be complex and contain many deceiving local minima (multi-modal). Therefore, the BP most likely gets trapped into a local minimum, making it entirely dependent on initial (weight) settings. There are many BP variants and extensions trying to address some or all of these problems, (Fahlman, 1988; Haykin, 1994; Sutton, 1986), yet all of them have one major drawback in common, that is, the ANN architecture has to be fixed in advance and the question of which specific ANN structure should be used for a particular problem still remains unanswered.

Many researchers proposed some elementary solutions for this. Let N_I , N_H and N_O be number of neurons in input, hidden and output layers. Jadid and Fairbairn (1996) proposed an upper bound on N_H such as $N_H \leq N_{TR}/(R + N_I + N_O)$ where N_{TR} is the number of training patterns. Masters (1994) suggested that ANN architecture should resemble a pyramid with $N_H \approx \sqrt{N_I + N_O}$. Hecht-Nielsen (1990) proved that $N_H \leq N_I + 1$ by using the Kolmogorov theorem.

Such boundaries may only give an idea about the architecture range that should be applied in general but many problems with high non-linearity and dynamic nature may require models that are beyond them. Therefore, they can only set some rules as a starting point for further analysis.

Designing the optimal network architecture can be thought as a search process within the architecture space containing all potential and feasible architectures. There are some efforts for a systematic way to achieve this such as the research on constructive and pruning algorithms, (Burgess, 1994; Frean, 1990; LeCun et al., 1990; Reed, 1993). The former methods initially assume a minimal ANN and insert nodes and links as warranted whilst the latter proceeds in the opposite way, i.e. assuming a large network, they prune off the superfluous components. However, Angeline et al. (1994) indicates that “Such structural hill climbing methods are susceptible to becoming trapped at structural local minima”. The reasoning behind this is clarified by Miller, Todd, and Hegde (1989), stating that the architecture space is non-differentiable, complex, deceptive and multi-modal. Therefore, those constructive and pruning algorithms are eventually faced with similar problems in the architecture space as the BP does in the error (weight) space. This makes evolutionary algorithms (EAs) (Back & Schwefel, 1993) such as Genetic Algorithm (GA) (Goldberg, 1989), Genetic Programming (GP) (Koza, 1992), Evolutionary Strategies (ES), (Back & Kursawe, 1995) and Evolutionary Programming (EP), (Fayyad et al., 1996), more promising candidates for both training and evolving the ANNs. The common point of all is that EAs are population based stochastic processes and they can avoid being trapped in a local optimum. Thus they can find the optimum solutions; however, this is never guaranteed. Furthermore, they have the advantage of being applicable to any type of ANN, feed-forward or not, with any activation function, differentiable or not.

GAs are a popular form of EAs that rely mainly on reproduction heuristics, *crossover* and random *mutations*. When used for training ANNs (with fixed architecture), many researchers (Bartlett & Downs, 1990; Hansen & Meservy, 1996; Miller et al., 1989; Prados, 1992; Porto, Fogel, & Fogel, 1995) reported that GAs can outperform BP in terms of both accuracy and speed, especially for large networks. However, as stated by Angeline et al. (1994) and Yao and Liu (1997), GAs do not suit well for evolving networks. For instance, the evolution process by GA suffers from the permutation problem (Bewley, McInerney, & Schraudolph, 1991), indicating that two identical ANNs may have different representations. This makes the evolution process quite inefficient in producing fit offspring. Most of the GAs use binary string representations of connection weights and architectures. This creates many problems, one of which is the representation precision of quantized weights. If weights are coarsely quantized, training might be infeasible since the required accuracy for a proper weight representation cannot be obtained. On the other hand, if too many bits are used (fine quantization), binary strings may be unfeasibly long, especially for large ANNs, and this makes the evolution process too slow or impractical. Another problem is the wide separation of the network components from the same or neighbor (hidden) nodes, in the binary string representation. Due to crossover operation, the interactions among them might be lost and hence the evolution speed is drastically reduced.

EP-based ENNs in Angeline et al. (1994) and Yao and Liu (1997) have been proposed to address the aforementioned problems of GAs and many others. The main distinction of EPs from GAs is that they do not use the problematic *crossover* operation, instead make the commitment to *mutation* as the sole operator for searching over the weight and architecture spaces. For instance the so-called EPNet, proposed in Yao and Liu (1997) uses 5 different mutations: hybrid training, node and connection deletions, node and connection additions. It starts with an initial population

of M random networks, partially trains each network for some epochs, selects the network with the best-rank performance as the parent network and if it is improved beyond some threshold, further training is performed to obtain an offspring network, which replaces its parent and the process continues until a desired performance criterion is achieved. Over several benchmark problems EPNet is shown to discover compact ANNs, which exhibit comparable performance with the other GA-based evolutionary techniques. However, it is neither shown nor proven that EPNet creates (near) optimal architectures; in that one potential drawback is the selection of the best network only with partial training since the winning network as a result of such initial (limited) BP training may not lead to the optimal network at the end, and hence this process may eliminate the crucial networks that have the potential to be the (near) optimum leading to minimum error if a proper training could have been performed. Another major drawback is its dependence on BP as the primary training method that has the aforementioned problems and severe limitations. Its complex structure can only be suitable or perhaps feasible for applications where the time is not a crucial factor and the training problem is not too complex, as stated in Yao and Liu (1997), “However, EPNet might take a long time to find a solution to a large parity problem. Some of the runs did not finish within the user-specified maximum number of generations”. Finally, as a hybrid algorithm it uses around 15 user-defined parameters/thresholds some of which are set with respect to the problem. This obviously creates a limitation in a generic and modular application domain.

2.2. The basic PSO algorithm

The particle swarm optimization (PSO) was introduced by Kennedy and Eberhart (1995) as a population based stochastic search and optimization process. It is originated from the computer simulation of the individuals (particles or living organisms) in a bird flock or fish school (Wilson, 1975), which basically show a natural behavior when they search for some target (e.g. food). The goal is, therefore, to converge to the global optimum of some multi-dimensional and possibly nonlinear function or system. In principle, PSO follows the same path as the other evolutionary algorithms (EAs) mentioned earlier. In a PSO process, a swarm of particles (or agents), each of which represents a potential solution to an optimization problem, navigate through the search space. The particles are initially distributed randomly over the search space and the goal is to converge to the global optimum of a function or a system. Each particle keeps track of its position in the search space and its best solution so far achieved. This is the personal best value (the so-called *pbest* in Kennedy and Eberhart (1995)) and the PSO process also keeps track of the global best solution so far achieved by the swarm with its particle index (the so called *gbest* in Kennedy and Eberhart (1995)). So during their journey with discrete time iterations, the velocity of each agent in the next iteration is computed by the best position of the swarm (personal best position of the particle *gbest* as the *social* component), the best personal position of the particle (*pbest* as the *cognitive* component), and its current velocity (the *memory* term). Both *social* and *cognitive* components contribute randomly to the position of the agent in the next iteration. As a stochastic search algorithm in multi-dimensional (MD) search space, PSO shows some major problems similar to the aforementioned EAs. A crucial problem for PSO is that parameter variations may result in large performance shifts (Lovberg & Krink, 2002). The second one is due to the direct link of information flow between particles and *gbest*, which then “guides” the rest of the swarm and thus resulting in the creation of similar particles with a loss of diversity. Hence this phenomenon increases the probability of being trapped in local optima (Riget & Vesterstrom, 2002) and

it is the main source of premature convergence problem especially when the search space is in high dimensions (Van den Bergh, 2002) and the problem to be optimized is multi-modal (Riget & Vesterstrom, 2002). Another reason for the premature convergence is that particles are flown through a single point which is (randomly) determined by *gbest* and *pbest* positions and this point is not even guaranteed to be a local optimum (Van den Bergh & Engelbrecht, 2002). Various modifications and PSO variants have been proposed in order to address this problem such as Abraham, Das, and Roy (2007), Christopher and Seppi (2004), Clerc (1999), Eberhart, Simpson, and Dobbins (1996), Higashi and Iba (2003), Kaewkamnerdpong and Bentley (2005), Lovberg (2002), Lovberg and Krink (2002), Peng, Reynolds, and Brewster (2003), Peram, Veeramachaneni, and Mohan (2003), Ratnaweera, Halgamuge, and Watson (2003), Ratnaweera, Halgamuge, and Watson (2002), Riget and Vesterstrom (2002), Richards and Ventura (2003), Shi and Eberhart (1998), Shi and Eberhart (2001), Van den Bergh and Engelbrecht (2002), Xie, Zhang, and Yang (2002a), Xie, Zhang, and Yang (2002b), Xie, Zhang, and Yang (2002c) and Zhang and Xie (2003).

In the basic PSO method, (*bPSO*), a swarm of particles flies through an N -dimensional search space where each particle represents a potential solution to the optimization problem. Each particle a in the swarm, $\xi = \{x_1, \dots, x_a, \dots, x_S\}$, is represented by the following characteristics:

$x_{a,j}(t)$: j th dimensional component of the position of particle a , at time t

$v_{a,j}(t)$: j th dimensional component of the velocity of particle a , at time t

$y_{a,j}(t)$: j th dimensional component of the personal best (*pbest*) position of particle a , at time t

$\hat{y}_j(t)$: j th dimensional component of the global best position of swarm, at time t .

Let f denote the fitness function to be optimized. Without loss of generality assume that the objective is to find the minimum of f in N dimensional space. Then the personal best of particle a can be updated in iteration $t + 1$ as,

$$y_{a,j}(t + 1) = \begin{cases} y_{a,j}(t) & \text{if } f(x_a(t + 1)) > f(y_a(t)) \\ x_{a,j}^{(t+1)} & \text{else} \end{cases} \\ j = 1, 2, \dots, N. \quad (1)$$

Since *gbest* is the index of the global best (GB) particle, then $\hat{y}(t) = y_{gbest}(t) = \min(y_1(t), \dots, y_S(t))$. Then for each iteration in a PSO process, positional updates are performed for each dimension component, $j \in \{1, N\}$ and for each particle, $a \in \{1, S\}$, as follows:

$$v_{a,j}(t + 1) = w(t)v_{a,j}(t) + c_1r_{1,j}(t) (y_{a,j}(t) - x_{a,j}(t)) \\ + c_2r_{2,j}(t) (\hat{y}_j(t) - x_{a,j}(t)) \quad (2) \\ x_{a,j}(t + 1) = x_{a,j}(t) + v_{a,j}(t + 1)$$

where w is the inertia weight and c_1, c_2 are the acceleration constants which are initially set to 2, (Shi & Eberhart, 1998). $r_{1,j} \sim U(0, 1)$ and $r_{2,j} \sim U(0, 1)$ are random variables with uniform distribution. Recall from the earlier discussion that the first term in the summation is the *memory* term, which represents the role of the previous velocity over the current velocity, the second term is the *cognitive* component, which represents the particle's own experience and the third term is the *social* component through which the particle is "guided" by the *gbest* particle towards the GB solution so far obtained. Although the use of inertia weight, w , was later added by Shi and Eberhart (1998), into the velocity update equation, it is widely accepted as the basic form of PSO algorithm. A larger value of w favors exploration while a small inertia weight favors exploitation. As originally introduced, w is often linearly decreased from a high value (e.g. 0.8) to a low value (e.g. 0.2) during iterations of a PSO run, which updates the positions

of the particles using Eq. (2). Depending on the problem to be optimized, PSO iterations can be repeated until a specified number of iterations, say *IterNo*, is exceeded, velocity updates become zero, or the desired fitness score is achieved (i.e. $f < \epsilon_C$). Accordingly the general pseudo-code of the *bPSO* can be given as follows:

bPSO (termination criteria: {*IterNo*, ϵ_C , ...}, V_{max})

1. For $\forall a \in \{1, S\}$ do:
 - 1.1. Randomize $x_a(0), v_a(0)$
 - 1.2. Let $y_a(0) = x_a(0)$
2. End For.
3. For $\forall t \in \{1, IterNo\}$ do:
 - 3.1. For $\forall a \in \{1, S\}$ do:
 - 3.1.1. Compute $y_a(t)$ using Eq. (1)
 - 3.1.2. If ($f(y_a(t)) < f(\hat{y}(t - 1))$) then $gbest = a$
 - 3.2. End For.
 - 3.3 If any of *termination criteria* is met, then **Stop**.
 - 3.4 For $\forall a \in \{1, S\}$ do:
 - 3.4.1. For $\forall j \in \{1, N\}$ do:
 - 3.4.1.1. Compute $v_{a,j}(t)$ using Eq. (2)
 - 3.4.1.2. If ($|v_{a,j}(t)| > V_{max}$) then clamp it to $|v_{a,j}(t)| = V_{max}$
 - 3.4.1.3. Compute $x_{a,j}(t)$ using Eq. (2).
 - 3.4.2. End For.
 - 3.5. End For.
4. End For.

Velocity clamping to the user-defined maximum velocity range V_{max} (and $-V_{max}$ for the minimum) as in step 3.4.1.2 is one of the earliest attempts to avoid premature convergence (Eberhart et al., 1996). Furthermore, this is the *bPSO* algorithm where the particle *gbest* is determined within the entire swarm. Another major topological approach, the so-called *lbest*, also exists where the swarm is divided into overlapping neighborhoods of particles and instead of defining *gbest* and $\hat{y}(t) = y_{gbest}(t)$ over the entire swarm, for a particular neighborhood N_i , the (local) best particle is referred as *lbest* with the position $\hat{y}_i(t) = y_{lbest}(t)$. Neighbors can be defined with respect to particle indices (i.e. $i \in \{j-l, j+l\}$) or by using some other topological forms (Suganthan, 1999). It is obvious that *gbest* is a special case of *lbest* scheme where the neighborhood is defined as the entire swarm. The *lbest* approach is one of the earlier attempts, which usually improves the diversity; however, it is slower than the *gbest* approach.

3. The MD PSO technique for automatic ANN design

In this section, we shall first introduce the MD PSO technique, which exhibits a substantial improvement over *bPSO* via inter-dimensional navigation. Its use for evolving the feed-forward ANNs will be detailed next.

3.1. MD PSO algorithm

Instead of operating at a fixed dimension N , the MD PSO algorithm is designed to seek both positional and dimensional optima within a dimension range, ($D_{min} \leq N \leq D_{max}$). In order to accomplish this, each particle has two sets of components, each of which has been subjected to two independent and consecutive processes. The first one is a regular positional PSO, i.e. the traditional velocity updates and due positional shifts in N dimensional search (solution) space. The second one is a dimensional PSO, which allows the particle to navigate through dimensions. Accordingly, each particle keeps track of its last position, velocity and personal best position (*pbest*) in a particular dimension so that when it re-visits that the same dimension at

a later time, it can perform its regular “positional” flight using this information. The dimensional PSO process of each particle may then move the particle to another dimension where it will remember its positional status and keep “flying” within the positional PSO process in this dimension, and so on. The swarm, on the other hand, keeps track of the *gbest* particles in all dimensions, each of which respectively indicates the best (global) position so far achieved and can thus be used in the regular velocity update equation for that dimension. Similarly the dimensional PSO process of each particle uses its personal best dimension in which the personal best fitness score has so far been achieved. Finally, the swarm keeps track of the global best dimension, *dbest*, among all the personal best dimensions. The *gbest* particle in *dbest* dimension represents the optimum solution and dimension, respectively.

In a MD PSO process at time (iteration) t , each particle a in the swarm, $\xi = \{x_1, \dots, x_a, \dots, x_S\}$, is represented by the following characteristics:

- $xx_{a,j}^{xd_a(t)}$ (t): j th component (dimension) of the position of particle a , in dimension $xd_a(t)$
- $vx_{a,j}^{xd_a(t)}$ (t): j th component (dimension) of the velocity of particle a , in dimension $xd_a(t)$
- $xy_{a,j}^{xd_a(t)}$ (t): j th component (dimension) of the personal best (*pbest*) position of particle a , in dimension $xd_a(t)$
- $gbest(d)$: Global best particle index in dimension d
- xy_j^{dbest} (t): j th component (dimension) of the global best position of swarm, in dimension d
- $xd_a(t)$: Dimension component of particle a
- $vd_a(t)$: Velocity component of dimension of particle a
- $\tilde{xd}_a(t)$: Personal best dimension component of particle a .

Let f denote the dimensional fitness function that is to be optimized within a certain dimension range, $\{D_{min}, D_{max}\}$. Without loss of generality assume that the objective is to find the minimum (position) of f at the optimum dimension within a multi-dimensional search space. Assume that the particle a visits (back) the same dimension after T iterations (i.e. $xd_a(t) = xd_a(t + T)$), then the personal best position can be updated in iteration $t + T$ as follows,

$$xy_{a,j}^{xd_a(t+T)}(t + T) = \begin{cases} xy_{a,j}^{xd_a(t)}(t) & \text{if } f(xx_{a,j}^{xd_a(t+T)}(t + T)) > f(xy_{a,j}^{xd_a(t)}(t)) \\ xx_{a,j}^{xd_a(t+T)}(t + T) & \text{else} \end{cases}$$

$$j = 1, 2, \dots, N. \quad (3)$$

Furthermore, the personal best dimension of particle a can be updated in iteration $t + 1$ as follows,

$$\tilde{xd}_a(t + 1) = \begin{cases} \tilde{xd}_a(t) & \text{if } f(xx_{a,j}^{xd_a(t+1)}(t + 1)) > f(xy_{a,j}^{\tilde{xd}_a(t)}(t)) \\ xd_a(t + 1) & \text{else} \end{cases} \quad (4)$$

Fig. 1 shows sample MD PSO and *bPSO* particles with indices a . *bPSO* particle that is at a (fixed) dimension, $N = 5$, contains only positional components whereas MD PSO particle contains both positional and dimensional components respectively. In the figure the dimension range for the MD PSO is given in between 2 and 10, therefore the particle contains 9 sets of positional components. In this example the current dimension the particle a resides is 2 ($xd_a(t) = 2$) whereas its personal best dimension is 3 ($\tilde{xd}_a(t) = 3$). Therefore, at time t a positional PSO update is first performed over the positional elements, $xx_a^2(t)$ and then the particle may move to another dimension with respect to the dimensional PSO. Recall that each positional element, $xx_{a,j}^2(t), j \in \{0, 1\}$, represents a potential solution in the data space of the problem.

Recall that $gbest(d)$ is the index of the global best particle at dimension d and let $S(d)$ be the total number of particles in dimension d , then $xy^{dbest}(t) = xy_{gbest(dbest)}^{dbest}(t) = \arg \min_{vi \in [1, S]} (f(xy_i^{dbest}(t)))$. For a particular iteration t , and for a particle $a \in \{1, S\}$, first the positional components are updated in its current dimension, $xd_a(t)$ and then the dimensional update is performed to determine its next ($t + 1^{st}$) dimension, $xd_a(t + 1)$. The positional update is performed for each dimension component, $j \in \{1, xd_a(t)\}$, as follows:

$$vx_{a,j}^{xd_a(t)}(t + 1) = w(t)vx_{a,j}^{xd_a(t)}(t) + c_1r_{1,j}(t)(xy_{a,j}^{xd_a(t)}(t) - vx_{a,j}^{xd_a(t)}(t)) + c_2r_{2,j}(t)(xy_j^{dbest}(t) - vx_{a,j}^{xd_a(t)}(t))$$

$$xx_{a,j}^{xd_a(t)}(t + 1) = xx_{a,j}^{xd_a(t)}(t) + C_{vx} [vx_{a,j}^{xd_a(t)}(t + 1), \{V_{min}, V_{max}\}]$$

$$xx_{a,j}^{xd_a(t)}(t + 1) \leftarrow C_{xx} [xx_{a,j}^{xd_a(t)}(t + 1), \{X_{min}, X_{max}\}]$$

where $C_{xx}[\dots] \equiv C_{vx}[\dots]$ are the clamping operators applied over each positional component, $xx_{a,j}^d$ and $vx_{a,j}^d$. $C_{xx}[\dots]$ may or may not be applied depending on the optimization problem but $C_{vx}[\dots]$ is basically needed to avoid exploding. Each operator can be applied in two different ways, such as,

$$C_{xx}[xx_{a,j}^d(t), \{X_{min}, X_{max}\}] = \begin{cases} xx_{a,j}^d(t) & \text{if } X_{min} \leq xx_{a,j}^d(t) \leq X_{max} \\ X_{min} & \text{if } xx_{a,j}^d(t) < X_{min} \\ X_{max} & \text{if } xx_{a,j}^d(t) > X_{max} \end{cases} \quad (a)$$

$$C_{xx}[xx_{a,j}^d(t), \{X_{min}, X_{max}\}] = \begin{cases} xx_{a,j}^d(t) & \text{if } X_{min} \leq xx_{a,j}^d(t) \leq X_{max} \\ U(X_{min}, X_{max}) & \text{else} \end{cases} \quad (b)$$

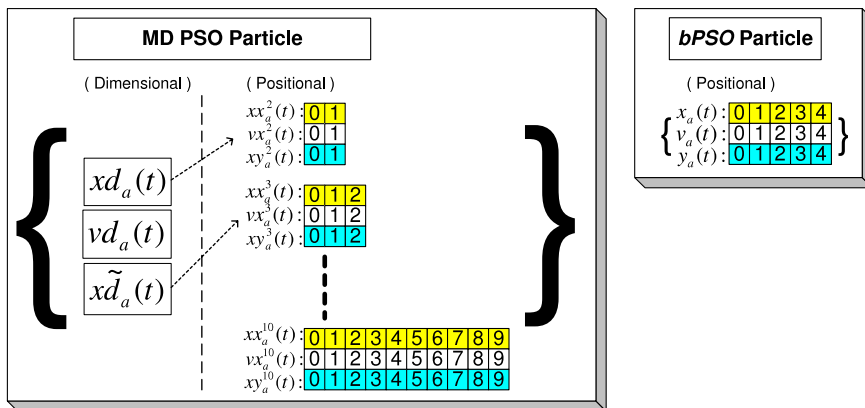


Fig. 1. Sample MD PSO (left) vs. *bPSO* (right) particle structures. For MD PSO $\{D_{min} = 2, D_{max} = 10\}$ and at time t , $xd_a(t) = 2$ and $\tilde{xd}_a(t) = 3$.

where the option (a) is a simple thresholding to the range limits and (b) re-initializes randomly the positional component in the j th dimension ($j < d$).

Note that the particle's new position, $xx_a^{xd_a(t)}(t + 1)$, will still be in the same dimension, $xd_a(t)$; however, the particle may jump to another dimension afterwards with the following dimensional update equations:

$$vd_a(t + 1) = \left\lfloor vd_a(t) + c_1 r_1(t) (x\tilde{d}_a(t) - xd_a(t)) + c_2 r_2(t) (dbest - xd_a(t)) \right\rfloor \quad (7)$$

$$xd_a(t + 1) = xd_a(t) + C_{vd} [vd_a(t + 1), \{VD_{min}, VD_{max}\}]$$

$$xd_a(t + 1) \leftarrow C_{xd} [xd_a(t + 1), \{D_{min}, D_{max}\}]$$

where $\lfloor \cdot \rfloor$ is the floor operator, $C_{xd}[\cdot, \cdot]$ and $C_{vd}[\cdot, \cdot]$ are the clamping operators applied over dimensional components, $xd_a(t)$ and $vd_a(t)$, respectively. As in (2), we employ the inertia weight for positional velocity update; however, we have witnessed no benefit of using it for dimensional PSO, and hence we left it out of (7) for the sake of simplicity. $C_{vd}[\cdot, \cdot]$ is similar to $C_{vx}[\cdot, \cdot]$, which is basically applied to avoid exploding and we use the basic thresholding for this, expressed as follows:

$$C_{vd}[vd_a(t), \{VD_{min}, VD_{max}\}] = \begin{cases} vd_a(t) & \text{if } VD_{min} \leq vd_a(t) \leq VD_{max} \\ VD_{min} & \text{if } vd_a(t) < VD_{min} \\ VD_{max} & \text{if } vd_a(t) > VD_{max} \end{cases} \quad (8)$$

$C_{xd}[\cdot, \cdot]$, on the other hand, is a mandatory clamping operator, which keeps the dimensional jumps within the dimension range of the problem, $\{D_{min}, D_{max}\}$. Furthermore within $C_{xd}[\cdot, \cdot]$, an optional in-flow buffering mechanism can also be implemented. This can be a desired property, which allows only a sufficient number of particles in a certain dimension and thus avoids redundancy. Particularly $dbest$ and dimensions within the close proximity have a natural attraction and without such a buffering mechanism, the majority of swarm particles may be hosted within this local neighborhood and hence other dimensions might encounter a severe depletion. To prevent this, the buffering mechanism should control the in-flow of the particles (by the dimensional velocity updates) to a particular dimension. On some early *bPSO* implementations over problems with low (and fixed) dimensions, 15–20 particles were usually sufficient for a successful operation. However, on high dimensions this may not be so since more particles are usually needed as the dimension increases. Therefore, we empirically set the limit as to be proportional with the solution space dimension and not less than 15. At a time t , let $P_d(t)$ be the number of particles in dimension d . $C_{xd}[\cdot, \cdot]$ can then be expressed with the (optional) buffering mechanism as follows:

$$C_{xd}[xd_a(t), \{D_{min}, D_{max}\}] = \begin{cases} xd_a(t - 1) & \text{if } P_d(t) \geq \max(15, xd_a(t)) \\ xd_a(t - 1) & \text{if } xd_a(t) < D_{min} \\ xd_a(t - 1) & \text{if } xd_a(t) > D_{max} \\ xd_a(t) & \text{else} \end{cases} \quad (9)$$

In short, the clamping and buffering operator, $C_{xd}[\cdot, \cdot]$, allows a dimensional jump only if the target dimension is within dimensional range and has space for a newcomer. Accordingly, the general pseudo-code of the MD PSO method can be expressed as follows:

MD PSO (termination criteria: $\{IterNo, \varepsilon_C, \dots\}$)

1. For $\forall a \in \{1, S\}$ do:
 - 1.1. Randomize $xd_a(0)$, $vd_a(0)$
 - 1.2. Initialize $x\tilde{d}_a(0) = xd_a(0)$
 - 1.3. For $\forall d \in \{D_{min}, D_{max}\}$ do:
 - 1.3.1. Randomize $xx_a^d(0)$, $xv_a^d(0)$
 - 1.3.2. Initialize $xy_a^d(0) = xx_a^d(0)$
 - 1.4. End For.
2. End For.
3. For $\forall t \in \{1, IterNo\}$ do:
 - 3.1. For $\forall a \in \{1, S\}$ do:
 - 3.1.1. If $(f(xx_a^{xd_a(t)}(t)) < \min(f(xy_a^{xd_a(t)}(t - 1)), \min_{p \in S - \{a\}}(f(xx_p^{xd_a(t)}(t))))$ then do:
 - 3.1.1.1. $xy_a^{xd_a(t)}(t) = xx_a^{xd_a(t)}(t)$
 - 3.1.1.2. If $(f(xx_a^{xd_a(t)}(t)) < f(xy_{gbest(xd_a(t))}^{xd_a(t)}(t - 1)))$ then $gbest(xd_a(t)) = a$
 - 3.1.1.3. If $(f(xx_a^{xd_a(t)}(t)) < f(xy_a^{x\tilde{d}_a(t-1)}(t - 1)))$ then $x\tilde{d}_a(t) = xd_a(t)$
 - 3.1.1.4. If $(f(xx_a^{xd_a(t)}(t)) < f(x\hat{y}_{dbest}(t - 1)))$ then $dbest = xd_a(t)$
 - 3.1.2. End If.
 - 3.2. End For.
 - 3.3. If the *termination criteria* are met, then **Stop**.
 - 3.4. For $\forall a \in \{1, S\}$ do:
 - 3.4.1. For $\forall j \in \{1, xd_a(t)\}$ do:
 - 3.4.1.1. Compute $vx_{aj}^{xd_a(t)}(t)$ and $xx_{aj}^{xd_a(t)}(t)$ using Eq. (5)
 - 3.4.2. End For.
 - 3.4.3. Compute $vd_a(t)$ and $xd_a(t + 1)$ using (7).
 - 3.5. End For.
4. End For.

Once the MD PSO process terminates, the optimum solution will be $x\hat{y}^{dbest}$ at the optimum dimension, $dbest$, achieved by the particle with the index $gbest(dbest)$ and finally the best (fitness) score achieved will naturally be $f(x\hat{y}^{dbest})$.

3.2. MD PSO for evolving ANNs

As a stochastic search process in multi-dimensional search space, MD PSO seeks for (near-) optimal networks in an architecture space, which can be defined by any type of ANNs with any properties. All network configurations in the architecture space are enumerated into a (dimensional) hash table with a proper hash function, which basically ranks the networks with respect to their complexity, i.e. associates higher hash indices to networks with higher complexity. The MD PSO can then use each index as a unique dimension of the search space where particles can make inter-dimensional navigations to seek an optimum dimension ($dbest$) and the optimum solution on that dimension, $x\hat{y}^{dbest}$. As mentioned earlier, the former corresponds to the optimal architecture and the latter encapsulates the (optimum) network parameters (connections, weights and biases).

In this section, we apply the MD PSO technique for evolving fully-connected, feed-forward ANNs or the so-called MLPs. The reasoning behind this choice is that MLP is the most widely used in this field and it offers conventional methods such as BP for training. Furthermore, we can perform comparative evaluations against other PSO techniques such as IPSONet, PSO-PSO and PSO-PSO-WD, which automatically design MLPs—yet only with a single hidden layer. This might be a serious drawback especially for complex problems and the main reason for their inferior performance with respect to other evolutionary algorithms based on GA and EP. We make no such assumptions over network

Table 1

A sample architecture space for MLP configuration arrays $R_{\min} = \{9, 1, 1, 2\}$ and $R_{\max} = \{9, 8, 4, 2\}$.

Dim.	Configuration	Dim.	Configuration
1	9 × 2	22	9 × 5 × 2 × 2
2	9 × 1 × 2	23	9 × 6 × 2 × 2
3	9 × 2 × 2	24	9 × 7 × 2 × 2
4	9 × 3 × 2	25	9 × 8 × 2 × 2
5	9 × 4 × 2	26	9 × 1 × 3 × 2
6	9 × 5 × 2	27	9 × 2 × 3 × 2
7	9 × 6 × 2	28	9 × 3 × 3 × 2
8	9 × 7 × 2	29	9 × 4 × 3 × 2
9	9 × 8 × 2	30	9 × 5 × 3 × 2
10	9 × 1 × 1 × 2	31	9 × 6 × 3 × 2
11	9 × 2 × 1 × 2	32	9 × 7 × 3 × 2
12	9 × 3 × 1 × 2	33	9 × 8 × 3 × 2
13	9 × 4 × 1 × 2	34	9 × 1 × 4 × 2
14	9 × 5 × 1 × 2	35	9 × 2 × 4 × 2
15	9 × 6 × 1 × 2	36	9 × 3 × 4 × 2
16	9 × 7 × 1 × 2	37	9 × 4 × 4 × 2
17	9 × 8 × 1 × 2	38	9 × 5 × 4 × 2
18	9 × 1 × 2 × 2	39	9 × 6 × 4 × 2
19	9 × 2 × 2 × 2	40	9 × 7 × 4 × 2
20	9 × 3 × 2 × 2	41	9 × 8 × 4 × 2
21	9 × 4 × 2 × 2		

properties and the architecture space can be defined over a wide range of configurations, i.e. say from a single-layer perceptron (SLP) to complex MLPs with many hidden layers. Suppose for the sake of simplicity, a range is defined for the minimum and maximum number of layers, $\{L_{\min}, L_{\max}\}$ and the number of neurons for the hidden layer l , $\{N_{\min}^l, N_{\max}^l\}$. Without loss of generality, assume that the size of both input and output layers is determined by the problem and hence fixed. This yields that the architecture space can now be defined only by two range arrays, $R_{\min} = \{N_i, N_{\min}^1, \dots, N_{\min}^{L_{\max}-1}, N_o\}$ and $R_{\max} = \{N_i, N_{\max}^1, \dots, N_{\max}^{L_{\max}-1}, N_o\}$, one for minimum and the other for maximum number of neurons allowed for each layer of a MLP. The size of both arrays is naturally $L_{\max} + 1$ where corresponding entries define the range of the l th hidden layer for all those MLPs, which can have an l th hidden layer. The size of input and output layers, $\{N_i, N_o\}$, is fixed and same for all configurations in the architecture space within which any l -layer MLP can be defined providing that $L_{\min} \leq l \leq L_{\max}$. $L_{\min} \geq 1$ and L_{\max} can be set to any value meaningful for the problem encountered. The hash function then enumerates all potential MLP configurations into hash indices, starting from the simplest MLP with $L_{\min} - 1$ hidden layers, each of which has minimum number of neurons given in R_{\min} , to the most complex network with $L_{\max} - 1$ hidden layers, each of which has maximum number of neurons given in R_{\max} .

Take, for instance, the following range arrays, $R_{\min} = \{9, 1, 1, 2\}$ and $R_{\max} = \{9, 8, 4, 2\}$, which indicate that $L_{\min} = 3$. If $L_{\min} = 1$ then the hash function enumerates all MLP configurations in the architecture space as shown in Table 1. Note that in this example, the input and output layer sizes are 9 and 2, which are eventually fixed for all MLP configurations. The hash function associates the 1st dimension to the simplest possible architecture, i.e., a SLP with only input and output layers (9×2). From dimensions 2 to 9, all configurations belong to 2-layers MLP with a hidden layer size varying between 1 and 8 (as specified in the 2nd entries of R_{\min} and R_{\max}). Similarly, for dimensions 10 and up-forth, 3-layers MLPs are enumerated where 1st and 2nd hidden layer sizes are varied according to the corresponding entries in R_{\min} and R_{\max} . Finally, the most complex MLP with the maximum possible layer and neuron sizes is associated with the highest dimension, 41. Therefore, all 41 entries in the hash table span the architecture space with respect to the configuration complexity and this eventually determines the dimension range of the solution space as $D_{\min} = 1$ and $D_{\max} = 41$.

Let N_h^l be the number of hidden neurons in layer l of a MLP with input and output layer sizes N_i and N_o , respectively. The input neurons are merely fan-out units since no processing take place. Let F be the activation function applied over the weighted inputs plus a bias, as follows:

$$y_k^{p,l} = F(s_k^{p,l}) \quad \text{where } s_k^{p,l} = \sum_j w_{jk}^{l-1} y_j^{p,l-1} + \theta_k^l \quad (10)$$

where $y_k^{p,l}$ is the output of the k th neuron of the l th hidden/output layer when the pattern p is fed, w_{jk}^{l-1} is the weight from j th neuron in layer $l-1$ to k th neuron in layer l , and θ_k^l is the bias value of the k th neuron of the l th hidden/output layer, respectively. The training mean square error, MSE , is formulated as,

$$MSE = \frac{1}{2PN_o} \sum_{p \in T} \sum_{k=1}^{N_o} (t_k^p - y_k^{p,O})^2 \quad (11)$$

where t_k^p is the target (desired) output and $y_k^{p,O}$ is the actual output from the k th neuron in the output layer, $l = O$, for pattern p in the training set T with size P , respectively. At a time t , suppose that the particle a in the swarm, $\xi = \{x_1, \dots, x_a, \dots, x_S\}$, has the positional component formed as, $xx_a^{x_{d_a}(t)}(t) = \{\{w_{jk}^0\}, \{w_{jk}^1\}, \{\theta_k^1\}, \{w_{jk}^2\}, \{\theta_k^2\}, \dots, \{w_{jk}^{O-1}\}, \{\theta_k^{O-1}\}, \{\theta_k^O\}\}$ where $\{w_{jk}^l\}$ and $\{\theta_k^l\}$ represent the sets of weights and biases of the layer l . Note that the input layer ($l = 0$) contains only weights whereas the output layer ($l = O$) has only biases. By means of such a direct encoding scheme, the particle a represents all potential network parameters of the MLP architecture at the dimension (hash index) $x_{d_a}(t)$. As mentioned earlier, the dimension range, $D_{\min} \leq x_{d_a}(t) \leq D_{\max}$, where MD PSO particles can make inter-dimensional jumps, is determined by the architecture space defined. Apart from the regular limits such as (positional) velocity range, $\{V_{\min}, V_{\max}\}$, dimensional velocity range, $\{VD_{\min}, VD_{\max}\}$, the data space can also be limited with some practical range, i.e. $X_{\min} < xx_a^{x_{d_a}(t)}(t) < X_{\max}$. In short, only some meaningful boundaries should be defined in advance for a MD PSO process and on contrary to other GA-based methods, or EPNet, which uses several parameters, thresholds and some other (alien) techniques (e.g. Simulated Annealing, BP, etc.) in a complex process, it uses a minimal parameter set, i.e. a few default parameters that are common with b PSO. Setting MSE in (11) as the fitness function enables the MD PSO to perform evolutions of both network parameters and architectures within its native process.

4. Experimental results

In order to test and evaluate the proposed technique for evolving ANNs, experiments are performed over synthetic and real (benchmark) problem sets. The aim is to test the “optimality” of the networks found with the former set, and to test “generalization” ability whilst performing comparative evaluations against several popular techniques with the latter. In order to determine which network architectures are (near-) optimal for a given problem, we apply exhaustive BP training over every network configuration in the architecture space defined. As mentioned earlier, BP is nothing but a gradient descent algorithm and thus for a single run, it is susceptible to get trapped to the nearest local minimum. However, performing it several times with randomized initial parameters eventually increases the chance of converging to (a close vicinity of) the global minimum in the error space, and therefore, it is performed by significantly large amount of times (e.g. $K = 500$) so that for a particular architecture, the minimum error obtained among all trials, can then be assumed to be the (near) optimal score achieved with that network. Note that even though K is kept

quite high, there is still no guarantee of converging to the global optimum with BP; however, the idea is to obtain the “trend” of best performances achievable with every configuration under equal training conditions. In this way the optimality of the networks evolved by MD PSO can be justified.

Due to the reasoning given earlier, the architecture space is defined over MLPs (possibly including SLP) with any activation functions. The input and output layer sizes are determined by the problem. We use the learning parameter for BP as $\lambda = 0.002$ and iteration number is 10 000. We kept the default PSO parameters for a MD PSO with a swarm size, $S = 200$, and velocity ranges are empirically set as $V_{\max} = -V_{\min} = X_{\max}/2$, and $VD_{\max} = -VD_{\min} = D_{\max}/2$. Dimension range is determined by the architecture space defined and position range is set as $X_{\max} = -X_{\min} = 2$. Unless stated otherwise, these parameters are used in all experiments presented in this section. We use the most typical activation functions: *hyperbolic tangent* ($\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$) in the next sub-section or *sigmoid* ($\text{sigm}(x) = 1/(1 + e^{-x})$) in Section 4.2.

4.1. Synthetic problems

Many researchers, who developed EA-based methods for automatic training or evolving ANNs, test them over some synthetic problems, most of which however, are quite simple. For example in Gudise and Venayagamoorthy (2003), the *bPSO* training algorithm is tested against BP over function approximation problem of $y = 2x^2 + 1$. In another work, Zhang, Zhang, Lok, and Lyu (2007), a hybrid PSO–BP algorithm is tested over problems such as 3-bit parity and function approximation of $y = \sin(2x)e^{-x}$. Note that the dynamic nature of these functions is quite stationary and it is thus fairly easy to approximate them by MLPs. EPNNet in Yao and Liu (1997) has been tested over an N -bit parity problem where N is kept within $4 \leq N \leq 8$ (due to feasibility problems mentioned earlier). Similarly, on such low N -bit parity problems, even the simplest MLPs can provide as satisfactory solutions as any other. This eventually creates an ambiguity over the decision of “optimality” since no matter how limited the architecture space is defined; still many networks in it can achieve “near-optimum” solutions. For instance consider the architecture space defined with $R_{\min} = \{3, 1, 1, 1\}$ and $R_{\max} = \{3, 8, 4, 1\}$ for the 3-bit parity problem in Zhang et al. (2007), above 90% of configurations in such a limited architecture space can achieve an MSE less than 10^{-4} . So in order to show the optimality of the network configurations evolved by the MD PSO, we shall first of all, use this “limited” architecture space ($R^1: R_{\min}^1 = \{N_I, 1, 1, N_O\}$ and $R_{\max}^1 = \{N_I, 8, 4, N_O\}$) containing the simplest 1, 2 or 3 layer MLPs with $L_{\min}^1 = 1$ and $L_{\max}^1 = 3$. It contains 41 networks similar to configurations in Table 1 where $N_I = 9$ and $N_O = 2$. Furthermore, we have selected the following the most challenging set of problems that can be solved with as few as possible “optimal” configurations: function approximation of $y = \cos(x/2) \sin(8x)$, 10-bit parity and two-spiral. In all experiments in this section, we set $K = 500$ for exhaustive BP training and perform 100 MD PSO runs, each of which terminates at the end of 1000 epochs (iterations).

As shown in Fig. 2, the function $y = \cos(x/2) \sin(8x)$ has a highly dynamic nature within the interval $\{-\pi, \pi\}$. 100 samples are taken for training where x coordinate of each sample is fed as input and y is used as the desired (target) output, so all networks are formed with $N_I = N_O = 1$. At the end of each run, the best fitness score (minimum error) achieved, $f(x\hat{y}^{dbest})$, by the particle with the index $gbest(dbest)$ at the optimum dimension $dbest$ is stored. The histogram of $dbest$, which is a hash index indicating a particular network configuration in R^1 , eventually provides the crucial information about the (near-) optimal configuration(s).

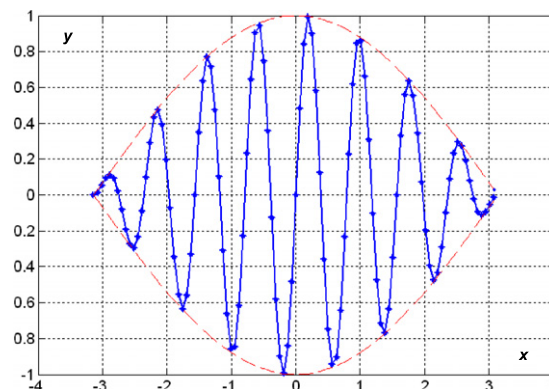


Fig. 2. The function $y = \cos(x/2) \sin(8x)$ plot in interval $\{-\pi, \pi\}$ with 100 samples.

Fig. 3 shows a *dbest* histogram and the error statistics plot from the exhaustive BP training for the function approximation problem. Note that BP can at best perform a linear approximation for most of the simple configurations (hash indices) resulting $MSE \approx 0.125$ and only some 3-layer MLPs (in the form of $1 \times M \times N \times 1$ where $M = 5, 6, 7, 8$ and $N = 2, 3, 4$) yield convergence to near-optima. Accordingly, from the *dbest* histogram it is straightforward to see that a high majority (97%) of the MD PSO runs converges to those near optima and moreover, the rest resulting with $dbest = 8$ and 9 (indicating 2-layer MLPs in $1 \times 7 \times 1$ and $1 \times 8 \times 1$ forms), also achieved a competitive performance with the optimal 3-layer MLPs. Therefore, these particular MD PSO runs are not indeed trapped to local minima; on contrary, the exhaustive BP training on these networks could not yield a solution. Furthermore, MD PSO evolution in this architecture space confirmed that 4 out of 41 configurations achieves the minimal MSEs (*mMSEs*), namely $1 \times 7 \times 4 \times 1$ ($mMSE(40) = 0.0204$), $1 \times 7 \times 2 \times 1$ ($mMSE(23) = 0.0207$), $1 \times 7 \times 3 \times 1$ ($mMSE(33) = 0.0224$), and $1 \times 8 \times 1$ ($mMSE(9) = 0.0292$). These are basically the four local peaks in the histogram indicating that majority of the runs evolves to these optimum networks whilst the rest converged to similar but near-optimum configurations, which performs only slightly worse (say only a few percent higher).

Fig. 4 presents fitness (MSE) and dimension (hash index) plots per iteration for two distinct MD PSO runs where red curves of both plots belong to the GB particle (the *gbest* particle at each iteration) and the corresponding plots of the blue curve are obtained from the particular *gbest* particles when the process terminates (i.e. *gbest* particles are 31 and 181 for the left and right plots, respectively). Recall that the main objective of the MD PSO is to find the true dimension where the global optimum exists and at this dimension, its internal process becomes identical with the *bPSO*. The optimum dimensions are 23 and 41 for the left and right plots. Note that in both experiments, several dimensions became *dbest* before they finally converge to the true optimum. Furthermore, it is interesting to observe the steep descent in MSE of both *gbest* particles (blue curve) after they converge to optimum dimensions.

Recall that an important advantage of the MD PSO process is its ability to provide not only the best network configuration but a ranked list of all possible network configurations in the architecture space, especially when MD PSO takes sufficient lifetime for evolution. Fig. 5 illustrates this fact with a typical experiment over the function approximation problem. The plot on the top is identical to the one in Fig. 3, showing the error statistics of the exhaustive BP training whereas the bottom one shows the minimum MSE (best fitness) achieved per dimension at the end of a MD PSO evolution process with 10 000 iterations (i.e. $mMSE(d) = f(x\hat{y}^d(10\ 000)) \forall d \in \{1, 41\}$). Note that

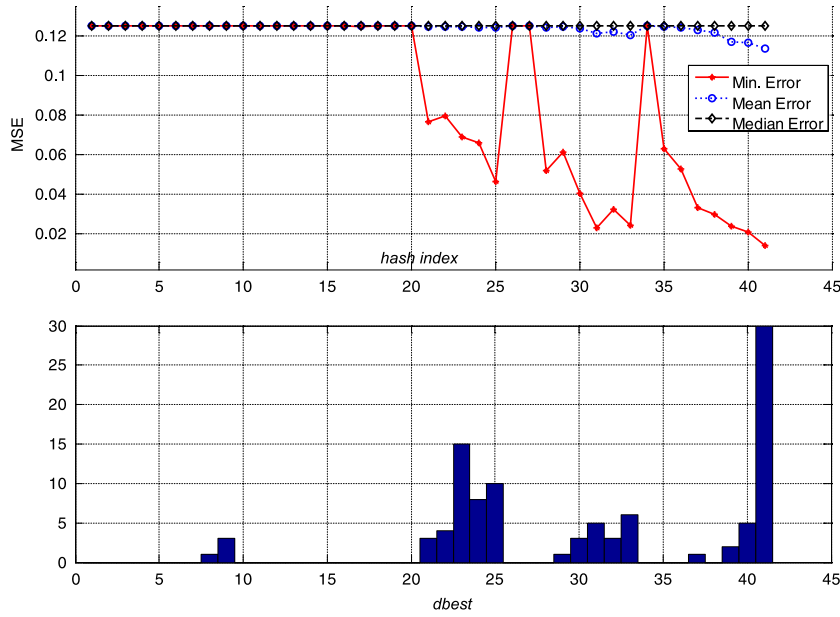


Fig. 3. Error statistics from exhaustive BP training (top) and dbest histogram from 100 MD PSO evolutions (bottom) for $y = \cos(x/2) \sin(8x)$ function approximation.

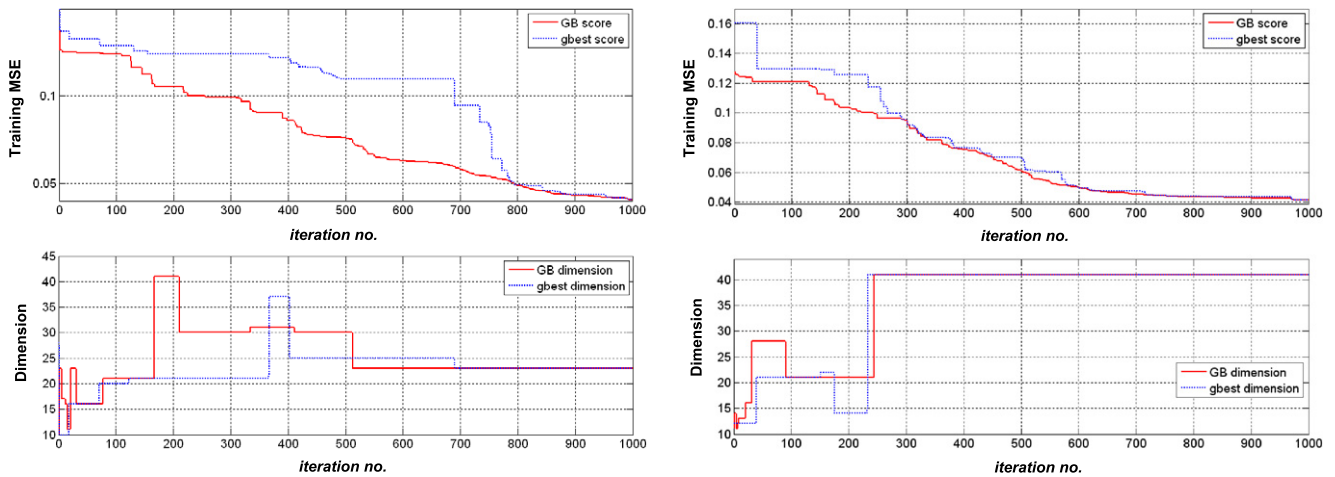


Fig. 4. Training MSE (top) and dimension (bottom) plots vs. iteration number for 17th (left) and 93rd (right) MD PSO runs.

both curves show a similar structure for $d > 18$ (e.g. note the peaks at $d = 19-20, 26-27, 34$); however, only the MD PSO manages to provide some near-optimal solutions for $d \leq 18$ whereas none of the BP runs managed to escape from local minima (the linear approximation). Most of the optimal and near-optimal configurations can be obtained from this single run, e.g. the ranked list is $\{dbest = 23, 24, 40, 33, 22, 39, 30, \dots\}$. Additionally, the peaks of this plot reveal the worst network configurations that should not be used for this problem, such as the ones with corresponding indices $d = 1, 2, 8, 9, 10, 11, 19, 26, 34$, etc. Note however that this can be a noisy evaluation since there is always the possibility that MD PSO process may also get trapped on a local minimum on these dimensions and/or not sufficient amount of particles visited this dimension due to their natural attraction towards $dbest$ (within MD PSO process), which might be too far away. This is the reason of the erroneous evaluation of dimensions 8 and 9, which should not be on that list.

Fig. 6 shows the $dbest$ histogram and the error statistics plot from the exhaustive BP training for 10-bit parity problem where $N_I = 10, N_O = 1$ used for all networks. In this problem, BP exhibits a better performance on the majority of the configurations, i.e. $4 \times 10^{-4} \leq mMSE(d) \leq 10^{-3}$ for $d = 7, 8, 9, 16, 23, 24, 25$, and

$\{29, 41\} - \{34\}$. The 100 MD PSO runs show that there are in fact two optimum dimensions: 30 and 41 (corresponding to 3-layer MLPs in $10 \times 5 \times 3 \times 1$ and $10 \times 8 \times 4 \times 1$ forms), which can achieve minimum MSEs, i.e. $mMSE(30) < 2 \times 10^{-5}$ and $mMSE(41) < 10^{-5}$. The majority of MD PSO runs, which is represented in the $dbest$ histogram in Fig. 6, achieved $MSE(dbest) < 8 \times 10^{-4}$ except the four runs (out of 100) with $MSE(dbest) > 4 \times 10^{-3}$ for $dbest = 4, 18$ and 19. These are the minority cases where MD PSO trapped to local minima but the rest evolved to (near-) optimum networks.

Two-spirals problem, which was proposed by Alexis Wieland (Zhou, Chen, & Chen, 2000), is highly non-linear and promises further interesting properties. For instance, the 2D data exhibits some temporal characteristics where the radius and angle of the spiral vary with time. The error space is highly multi-modal with many local minima, thus methods such as BP encounters severe problems on error reduction (Baum & Lang, 1991). The data set consists of 194 patterns (2D points), 97 samples in each of the two classes (spirals). It is now used as a benchmark for ANNs by many researchers. Lang and Witbrock (1988) reported that a near-optimum solution could not be obtained with standard BP algorithm over feed-forward ANNs. They tried a special network structure with short-cut links between layers. Similar conclusions

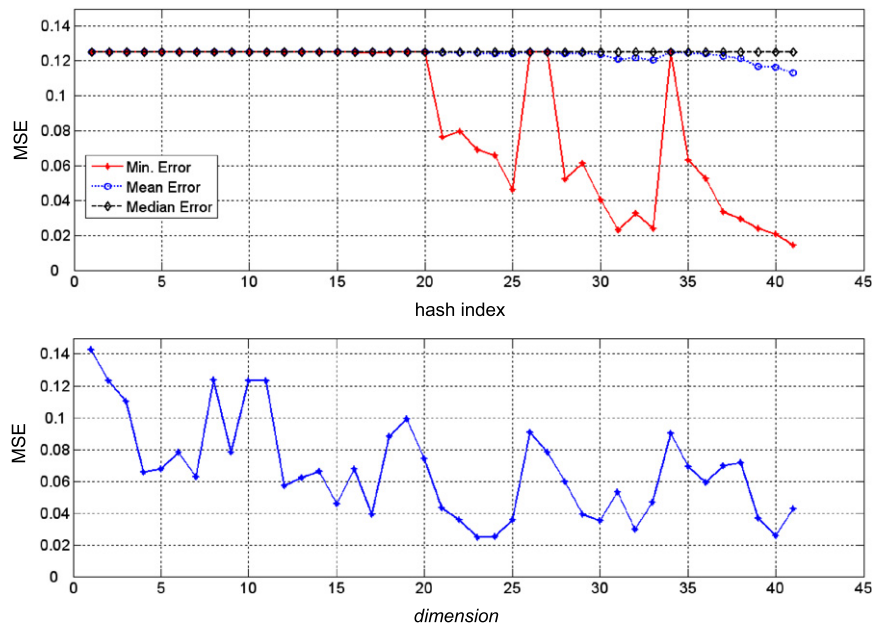


Fig. 5. MSE plots from exhaustive BP training (top) and a single run of MD PSO (bottom).

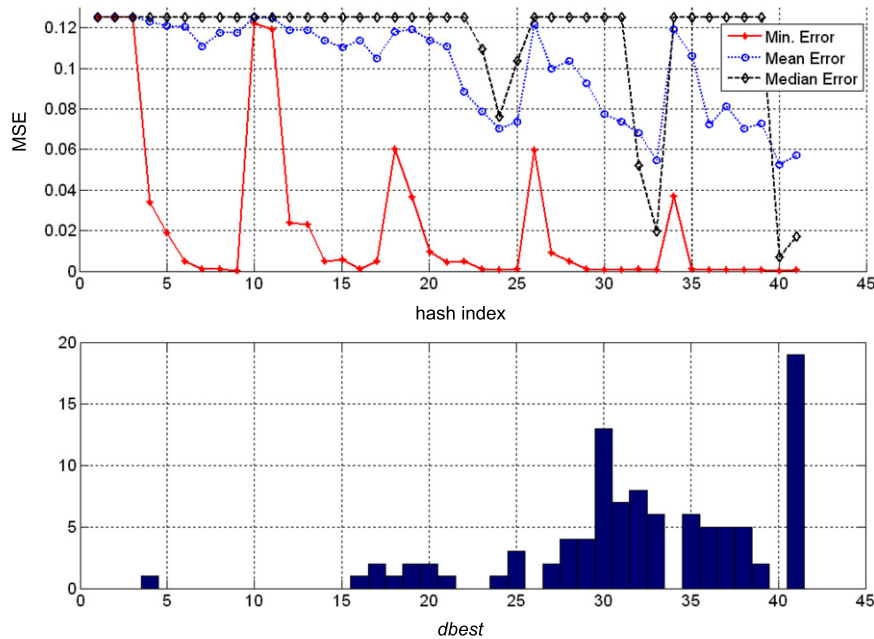


Fig. 6. Error statistics from exhaustive BP training (top) and dbest histogram from 100 MD PSO evolutions (bottom) for 10 bit parity problem.

are reported by Baum and Lang (1991) that the problem is unsolvable with 2-layers MLPs with $2 \times 50 \times 1$ configuration. This, without doubt, is one of the hardest problems in the field of ANNs. Fig. 7 shows *dbest* histogram and the error statistics plot from the exhaustive BP training for the *two-spirals* problem where $N_l = 2$, $N_o = 1$. It is obvious that none of the configurations yields a sufficiently low error value with BP training and particularly BP can at best perform a linear approximation for most of the configurations (hash indices) resulting in an $MSE \approx 0.49$ and only few 3-layer MLPs (with indices 32, 33, 38 and 41) are able to reduce MSE to 0.3. MD PSO also shows a similar performance with BP, achieving $0.349 \leq mMSE(dbest) \leq 0.371$ for $dbest = 25, 32, 33, 38$ and 41. These are obviously the best possible MLP configurations to which a high majority of MD PSO runs converged in R^1 .

4.2. Medical diagnosis problems from Proben1

In the previous section a set of synthetic problems that are among the hardest and the most complex in the ANN field, has been used in order to test the optimality of the MD PSO evolution process, i.e. to see whether or not the MD PSO can evolve to the few (near-) optimal configurations present in the limited architecture space, R^1 , which mostly contains shallow MLPs. In this section we shall test the generalization capability of the proposed method and perform comparative evaluations against the most promising, state-of-the-art evolutionary techniques, over a benchmark dataset, which is partitioned into three sets: training, validation and testing. There are several techniques (Hassoun, 1995) to use training and validation sets individually to prevent over-fitting and thus to improve the classification performance in

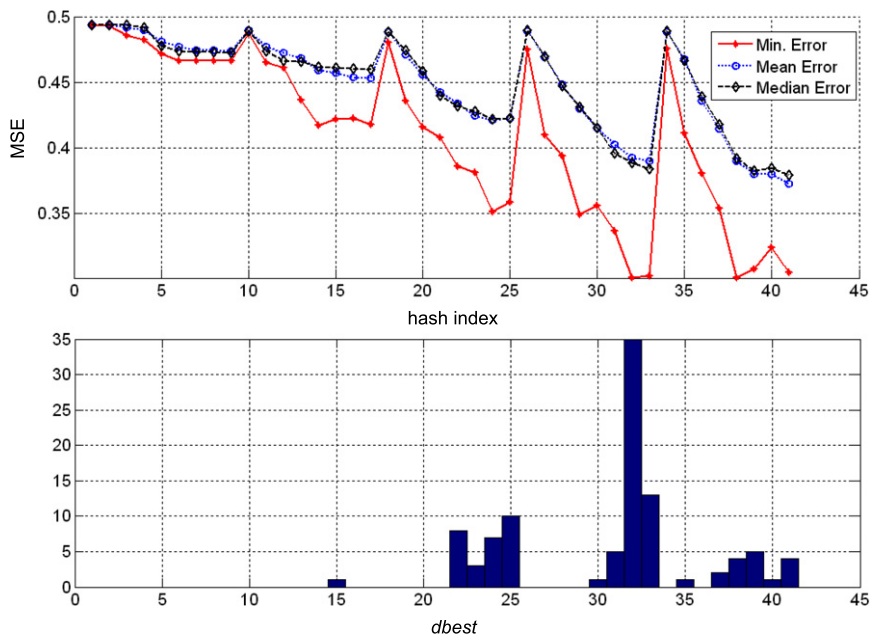


Fig. 7. Error (MSE) statistics from exhaustive BP training (top) and dbest histogram from 100 MD PSO evolutions (bottom) for the two-spirals problem.

the test data. However, the use of validation set should not be needed for EA based techniques since they search globally for a solution (Sexton & Dorsey, 2000). They are furthermore beyond the scope of this paper and hence not used in order to obtain an unbiased performance measure, although all competing methods presented in this article used them in some way to maximize their classification rate over the test data. We simply combine the validation and training sets to use for training.

From *Proben1* repository (Prechelt, 1994), we selected three benchmark classification problems, *breast cancer*, *heart disease* and *diabetes*, which were commonly used by the competing methods such as PSO-PSO (Zhang & Shao, 2000), PSO-PSO:WD (Carvalho & Ludermir, 2007), IPSONet (Yu et al., 2007), EPNet (Yao & Liu, 1997), GA (basic) (Sexton & Dorsey, 2000) and GA (Connection Matrix) (Cantu-Paz & Kamath, 2005). These are medical diagnosis problems, which mainly present the following attributes:

- All of them are real-world problems based on medical data from human patients.
- The input and output attributes are similar to those used by a medical doctor.
- Since medical examples are expensive to get, the training sets are quite limited.

We now briefly describe each classification problem as follows:

4.2.1. Breast cancer

The objective of this data set is to classify breast lumps as either benign or malignant according to microscopic examination of cells that are collected by needle aspiration. There are 699 exemplars of which 458 are benign and 241 are malignant and they are originally partitioned as 350 for training, 175 for validation and 174 for testing. The data set consists of 9 input and 2 output attributes. It is created at University of Wisconsin Madison by Dr. William Wolberg.

4.2.2. Diabetes

This data set is used to predict diabetes diagnosis among Pima Indians. All patients reported are females of at least 21 years old. There is a total of 768 exemplars of which 500 are classified as

diabetes negative and 268 as diabetes positive. The data set is originally partitioned as 384 for training, 192 for validation and 192 for testing. It consists of 8 input and 2 output attributes.

4.2.3. Heart disease

The initial data set consists of 920 exemplars with 35 input attributes, some of which are severely missing. Hence a second data set is composed using the cleanest part of the preceding set, which was created at Cleveland Clinic Foundation by Dr. Robert Detrano. The Cleveland data is called as “heartc” in *Proben1* repository and contains 303 exemplars but 6 of them still contain missing data and hence discarded. The rest is partitioned as 149 for training, 74 for validation and 74 for testing. There are 13 input and 2 output attributes. The purpose is to predict the presence of the heart disease according to input attributes.

The input attributes of all data sets are scaled to between 0 and 1 by a linear function. Their output attributes are encoded using a 1-of- c representation using c classes. The *winner-takes-all* methodology is applied so that the output of the highest activation designates the class. In overall, our experimental setup becomes identical to ones in the competing methods and thus fair comparative evaluations can now be made over the classification error rate of the test data. In all experiments in this section we mainly use R^1 that is specified by range arrays, $R_{\min}^1 = \{N_I, 1, 1, N_O\}$ and $R_{\max}^1 = \{N_I, 8, 4, N_O\}$ containing the simplest 1, 2 or 3 layers MLPs where N_I and N_O , are determined by the number of input and output attributes of the classification problem. In order to obtain statistical results, we perform 100 MD PSO runs, each of which use 250 particles ($S = 250$) and terminate at the end of 200 epochs ($E = 200$).

Before presenting the classification results over the test data, there are some crucial points worth mentioning. First of all, the aforementioned ambiguity over the decision of “optimality” is witnessed over the (training) data sets, *Diabetes* and particularly the *Breast Cancer*, as the majority of networks in R^1 can achieve similar performances. Fig. 8 demonstrates this fact by two error statistics plots from the exhaustive BP training ($K = 500$, $\lambda = 0.05$) with 5000 epochs. Note that most of the networks trained over both data sets result such *mMSEs* that are within a narrow

Table 2
Mean (μ) and standard deviation (σ) of classification error rates (%) over test data sets.

Algorithm	Data set					
	Breast cancer		Diabetes		Heart disease	
	μ	σ	μ	σ	μ	σ
Proposed (MD PSO)	0.39	0.31	20.55	1.22	19.53	1.71
PSO-PSO	4.83	3.25	24.36	3.57	19.89	2.15
PSO-PSO:WD	4.14	1.51	23.54	3.16	18.1	3.06
IPSONet	1.27	0.57	21.02	1.23	18.14*	3.42
EPNet	1.38	0.94	22.38	1.4	16.76*	2.03
GA (Sexton & Dorsey, 2000)	2.27	0.34	26.23	1.28	20.44	1.97
GA (Cantu-Paz & Kamath, 2005)	3.23	1.1	24.56	1.65	23.22	7.87
BP	3.01	1.2	29.62	2.2	24.89	1.71

Table 3
A sample architecture space with range arrays, $R^2_{\min} = \{13, 6, 6, 3, 2\}$ and $R^2_{\max} = \{13, 12, 10, 5, 2\}$.

Dim.	Configuration	Dim.	Configuration	Dim.	Configuration	Dim.	Configuration
1	13 × 2	9	13 × 6 × 6 × 2	17	13 × 7 × 7 × 2	141	13 × 12 × 9 × 5 × 2
2	13 × 6 × 2	10	13 × 7 × 6 × 2	18	13 × 8 × 7 × 2	142	13 × 6 × 10 × 5 × 2
3	13 × 7 × 2	11	13 × 8 × 6 × 2	19	13 × 9 × 7 × 2	143	13 × 7 × 10 × 5 × 2
4	13 × 8 × 2	12	13 × 9 × 6 × 2	144	13 × 8 × 10 × 5 × 2
5	13 × 9 × 2	13	13 × 10 × 6 × 2	145	13 × 9 × 10 × 5 × 2
6	13 × 10 × 2	14	13 × 11 × 6 × 2	138	13 × 9 × 9 × 5 × 2	146	13 × 10 × 10 × 5 × 2
7	13 × 11 × 2	15	13 × 12 × 6 × 2	139	13 × 10 × 9 × 5 × 2	147	13 × 11 × 10 × 5 × 2
8	13 × 12 × 2	16	13 × 6 × 7 × 2	140	13 × 11 × 9 × 5 × 2	148	13 × 12 × 10 × 5 × 2

band and it is rather difficult to distinguish or separate one from another.

Contrary to the two data sets, only four distinct sets of network configuration can achieve training *mMSEs* below 10^{-2} over the *Heart Disease* data set, as shown in Fig. 9. The corresponding indices (dimensions) to these four optimal networks are *dbest* = 9, 25, 32 and 41, where the MD PSO managed to evolve either to them or to neighbor (near-optimal) configurations. Note that the majority of MD PSO runs (>50%) evolve to the simplest MLPs with single hidden layer (i.e. from Table 1 *dbest* = 9 is for the MLP 13 × 8 × 2) although BP achieved slightly lower *mMSEs* over other three (near-) optimal configurations. The main reason is the fact that MD PSO or PSO in general performs better in low dimensions and recall that premature convergence problem might also occur when the search space is in high dimensions (Van den Bergh, 2002). Therefore, the MD PSO naturally favors a low-dimension solution when it exhibits a competitive performance with a high dimension counterpart. Such a natural tendency eventually yields the evolution process to compact network configurations in the architecture space rather than the complex ones, as long as the optimality prevails.

Table 2 presents the classification error rate statistics of the proposed and competing methods. An error rate in the table refers to the percentage of wrong classifications over the test data set of a benchmark problem. It is straightforward to see that the best classification performance is achieved with the proposed technique over the *Diabetes* and *Breast Cancer* data sets. Particularly on the latter set, roughly half of the MD PSO runs resulted in a zero (0%) error rate, meaning that perfect classification is achieved. The MD PSO exhibits a competitive performance on *Heart Disease* data set. However, note that both IPSONet and EPNet, showing slightly better performances, used a subset of this set (134 for training, 68 for validation and 68 for testing), excluding 27 entries overall. In Yao and Liu (1997) this is reasoned as, “27 of these were retained in case of dispute, leaving a final total of 270”. Therefore, it is not clear to make direct comparative evaluations between them and the proposed technique on this data set. Furthermore, note that the proposed technique in general achieved minimum standard deviations and thus promising a better robustness and stability over the evolutionary process. As a result MD PSO can provide a superior

generalization capability although its main purpose is evolution to the (near-) optimal network(s) over the training set and none of the aforementioned improvement methods is used whilst all of the competing techniques are.

4.3. Architecture space dependency and complexity analysis

First we shall investigate the effects of network architecture properties on the classification performance. To do so, a broader architecture space, R^2 , is defined with larger and deeper MLPs with the following range arrays, $R^2 : R^2_{\min} = \{N_I, 6, 6, 3, N_O\}$ and $R^2_{\max} = \{N_I, 12, 10, 5, N_O\}$ using $L_{\min} = 1, L_{\max} = 4$. The second architecture space has 148 MLP configurations as shown in Table 3 where $N_I = 13$ and $N_O = 2$. Table 4 presents the classification error rate statistics of the proposed technique applied to both architecture spaces, R^1 and R^2 for a direct comparison. From the table, it is obvious that classification performances with both R^1 and R^2 are quite similar. This is indeed an expected outcome since all classification problems require only the simplest MLPs with a single hidden layer for a (near-) optimal performance. Therefore, no significant performance gain is observed when deeper and more complex MLPs in R^2 are used for these problems and furthermore, the MD PSO's evolution process among those complex networks might be degraded by the limited number of iterations (*iterNo* = 200) used for training and high dimensionality/modality of the solution spaces encountered in R^2 .

During a MD PSO process, at each iteration and for each particle, first the network parameters are extracted from the particle and input vectors are (forward) propagated to compute the average MSE at the output layer. Therefore, it is not feasible to accomplish a precise computational complexity analysis of the MD PSO evolutionary process since this mainly depends on the networks that the particles converge and in a stochastic process such as PSO this cannot be determined. Furthermore, it also depends on the architecture space selected because the MD PSO can only search for the optimal network within the architecture space. This will give us a hint that the computational complexity also depends on the problem in hand because particles eventually tend to converge to those near-optimal networks after the initial period of the process. Yet we can count certain attributes which directly affect the complexity, such as the size of the training data set (T), swarm

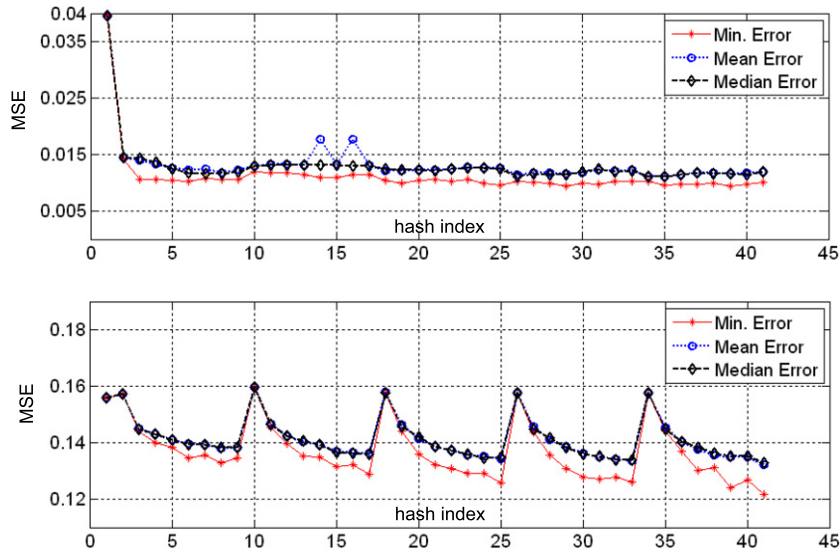


Fig. 8. Error statistics from exhaustive BP training over Breast Cancer (top) and Diabetes (bottom) data sets.

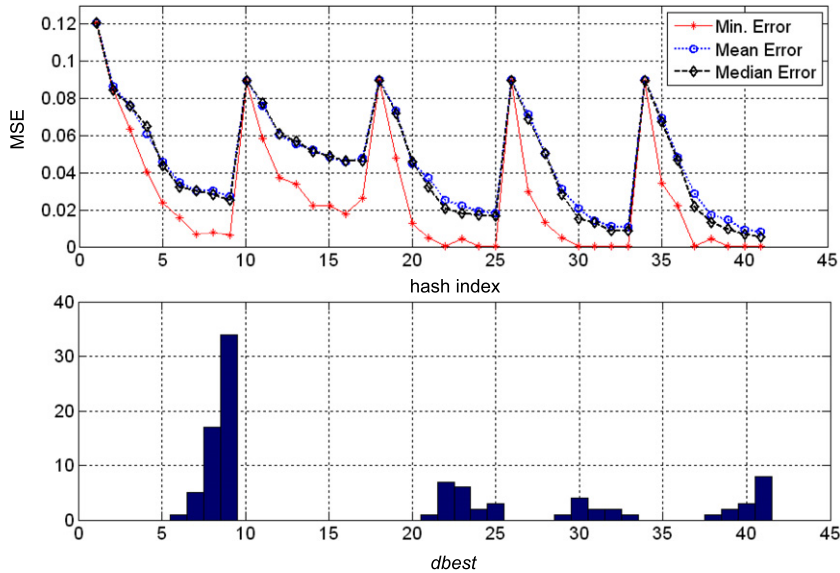


Fig. 9. Error statistics from exhaustive BP training (top) and dbest histogram from 100 MD PSO evolutions (bottom) over the Heart Disease data set.

size (S) and number of epochs to terminate the MD PSO process (E). Since the computational complexity is proportional with the total number of forward propagations performed, then it can be in the order of $O(SET\mu_t)$ where μ_t is an abstract time for the propagation and MSE computation over an average network in the architecture space. Due to the aforementioned reasons, μ_t cannot be determined a priori and the problem determines T , yet the computational complexity can still be controlled by S and E settings.

5. Conclusions

In this paper, we proposed a novel evolutionary method for ANNs by the MD PSO with the following innovative properties:

- The MD PSO technique promises a generic solution for the common drawback of the family of PSO methods such as *a priori* fixation of the search space dimension. It efficiently addresses this by defining a new particle formation and embedding

the ability of dimensional navigation into the core of the process. It basically allows particles to make inter-dimensional ‘passes’ with a dedicated PSO process whilst performing regular positional updates in every dimension they visit. Such flexibility negates the requirement of setting the dimension in advance since swarm particles can now converge to the global solution at the optimum dimension, in a simultaneous manner.

- With the proper adaptation of the native MD PSO process, the proposed method can evolve to the optimum network within an architecture space for a particular problem. Additionally, it provides a ranked list of all other potential configurations, indicating that any high rank configuration may be an alternative to the optimum one, yet some with low ranking, on the contrary, should not be used at all.
- The proposed method is generic and applicable to any type of ANNs in an architecture space with varying size and properties, as long as a proper hash function enumerates all configurations in the architecture space with respect to their complexity into

Table 4

Classification error rate (%) statistics of the MD PSO when applied to two architecture spaces.

Error rate statistics	Breast cancer		Diabetes		Heart disease	
	R^1	R^2	R^1	R^2	R^1	R^2
μ	0.39	0.51	20.55	20.34	19.53	20.21
σ	0.31	0.37	1.22	1.19	1.71	1.9

proper hash indices representing the dimensions of the solution space over which the MD PSO seeks for optimality.

- Most of the competing techniques presented in this paper are of a complex nature that follows a long, rather ambiguous pseudo-code with many conditional steps and requires several parameters to be fixed in advance. They also rely on some alien methods with known limitations and drawbacks. On the other hand, the proposed method, which entirely depends on an MD PSO, has a simple and unique structure for evolving ANNs without any significant parameter dependency or interference of any other method.
- Due to the MD PSO's native feature of having better and faster convergence to optimum solution in low dimensions, its evolution process in any architecture space naturally yields to compact networks rather than large and complex ones, as long as the optimality prevails.
- Finally, the proposed method achieves a superior generalization ability in comparison with others despite the fact that it uses none of the improvement techniques that they use. We do not apply any of the well-known PSO modifications or variants to improve the global search ability and hence have shown basically what a "baseline" implementation is capable of. Applying them is likely to improve the convergence rate to the optimum solution as well as the generalization capability, both of which shall be topics for further research studies.

Experimental results show that the proposed method achieves all the abovementioned properties and capabilities in an automatic way and as a result, it alleviates the need of human "expertise" and "knowledge" for designing a particular network; instead, such virtues may still be used in a flexible way to define only the size and perhaps some crucial properties of the architecture space. In this way further efficiency in terms of speed and accuracy over global search and evolution process can be achieved.

Current and planned future studies include: evolution of the initial architecture space by using the ranked list from the MD PSO evolution process in such a way that the worst configurations in the list can be removed and instead, new configurations similar to ones in the high ranks can be added in an iterative way. In this way there will be an ongoing upgrade of the architecture space with better and more promising networks. To increase the efficiency on the evolution process, developing an efficient technique, which improves MD PSO's convergence accuracy and speed, both positional and dimensional, is also considered.

References

- Abraham, A., Das, S., & Roy, S. (2007). Swarm intelligence algorithms for data clustering. In *Soft computing for knowledge discovery and data mining Part IV*, (pp. 279–313), October 25.
- Angeline, P. J., Saunders, G. M., & Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5, 54–65.
- Back, T., & Schwefel, H. P. (1993). An overview of evolutionary algorithm for parameter optimization. *Evolutionary Computation*, 1, 1–23.
- Back, T., & Kursawe, F. (1995). Evolutionary algorithms for fuzzy logic: A brief overview. In *Fuzzy logic and soft computing* (pp. 3–10). Singapore: World Scientific.
- Bartlett, P., & Downs, T. (1990). Training a neural network with a genetic algorithm. *Technical report*, Australia: Dep. Elect. Eng., Univ. Queensland.
- Baum, E., & Lang, K. (1991). Constructing hidden units using examples and queries. *Advances in Neural Information Processing Systems*, 3, 904–910.
- Belew, R. K., McInerney, J., & Schraudolph, N. N. (1991). Evolving networks: Using genetic algorithm with connectionist learning. *Technical report CS90-174 revised*. San Diego: Computer Sci. Eng. Dept., Univ. California.
- Burgess, N. (1994). A constructive algorithm that converges for real-valued input patterns. *International Journal of Neural Systems*, 5(1), 59–66.
- Christopher, K. M., & Seppi, K. D. (2004). The Kalman Swarm. A new approach to particle motion in swarm optimization. In *Proc. of the genetic and evolutionary computation conf.* (pp. 140–150).
- Cantu-Paz, E., & Kamath, C. (2005). An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 35, 915–927.
- Carvalho, M., & Ludermir, T. B. (2007). Particle swarm optimization of neural network architectures and weights. In *Proc. of the 7th int. conf. on hybrid intelligent systems* (pp. 336–339).
- Clerc, M. (1999). The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In *Proc. of the IEEE congress on evolutionary computation*. vol. 3 (pp. 1951–1957).
- Eberhart, R., Simpson, P., & Dobbins, R. (1996). *Computational intelligence. PC Tools*. Boston, MA, USA: Academic Press, Inc.
- Esquivel, S. C., & Coello Coello, C. A. (2003). On the use of particle swarm optimization with multimodal functions. *IEEE Transactions on Evolutionary Computation*, 2, 1130–1136.
- Fahlman, S. E. (1988). An empirical study of learning speed in backpropagation. *Technical report, CMU-CS-88-162*. Carnegie-Mellon University.
- Fayyad, U. M., Shapire, G. P., Smyth, P., & Uthurusamy, R. (1996). *Advances in knowledge discovery and data mining*. Cambridge, MA: MIT Press.
- Frean, M. (1990). The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2(2), 198–209.
- Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley (pp. 1–25).
- Gudise, V. G., & Venayagamoorthy, G. K. (2003). Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. *Swarm intelligence symposium, 2003. Proc. of the 2003 IEEE* (pp. 110–117).
- Hansen, J. V., & Meservy, R. D. (1996). Learning experiments with genetic optimization of a generalized regression neural network. *Decision Support Systems*, 18(3–4), 317–325.
- Hassoun, M. H. (1995). *Fundamentals of artificial neural networks*. Cambridge, MA: MIT Press.
- Haykin, S. (1994). *Neural networks: A comprehensive foundation*. New York: Macmillan.
- Hecht-Nielsen, R. (1990). *Neurocomputing*. Reading, MA: Addison-Wesley.
- Higashi, H., & Iba, H. (2003). Particle swarm optimization with Gaussian mutation. In *Proc. of the IEEE swarm intelligence symposium* (pp. 72–79).
- Jadid, M. N., & Fairbairn, D. R. (1996). Predicting moment-curvature parameters from experimental data. *Engineering Applications of Artificial Intelligence*, 9(3), 309–319.
- Kaewkamnerdpong, B., & Bentley, P. J. (2005). Perceptive particle swarm optimization: an investigation. In *Proc. of IEEE swarm intelligence symposium* (pp. 169–176).
- Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proc. of IEEE int. conf. on neural networks*. vol. 4 (pp. 1942–1948).
- Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.
- Lang, K. J., & Witbrock, M. J. (1988). Learning to tell two spirals apart. In *Proc. of the 1988 connectionist models summer school*.
- LeCun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. *Advances in Neural Information Processing Systems*, 2, 598–605.
- Lovberg, M. (2002). Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality. *M.Sc. thesis*. Denmark: Department of Computer Science, University of Aarhus.
- Lovberg, M., & Krink, T. (2002). Extending particle swarm optimisers with self-organized criticality. In *Proc. of the IEEE congress on evolutionary computation*. vol. 2 (pp. 1588–1593).
- Masters, T. (1994). *Practical neural network recipes in C++*. Boston, MA: Academic Press.
- McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 7, 115–133.
- Meissner, M., Schmuker, M., & Schneider, G. (2006). Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7, 125.
- Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *Proc. 3rd int. conf. genetic algorithms their applications* (pp. 379–384).
- Omran, M. G., Salman, A., & Engelbrecht, A. P. (2006). *Particle swarm optimization for pattern recognition and image processing*. Berlin: Springer.
- Peng, B., Reynolds, R. G., & Brewster, J. (2003). Cultural swarms. In *Proc. of the IEEE congress on evolutionary computation*. vol. 3 (pp. 1965–1971).
- Peram, T., Veeramachaneni, K., & Mohan, C. K. (2003). Fitness-distance-ratio based particle swarm optimization. In *Proc. of the IEEE swarm intelligence symposium*, (pp. 174–181). IEEE Press.
- Prados, D. L. (1992). Training multilayered neural networks by replacing the least fit hidden neurons. In *Proc. IEEE SOUTHEASTCON'92*. vol. 2 (pp. 634–637).
- Prechelt, L. (1994). Proben1—A set of neural network Benchmark problems and benchmark rules. *Technical report 21/94*, Germany: Fakultät für Informatik, Universität Karlsruhe.

- Porto, V. W., Fogel, D. B., & Fogel, L. J. (1995). Alternative neural network training methods. *IEEE Expert*, 10(March), 16–22.
- Ratnaweera, A. C., Halgamuge, S. K., & Watson, H. C. (2002). Particle swarm optimiser with time varying acceleration coefficients. In *Proc. of the int. conf. on soft computing and intelligent systems* (pp. 240–255).
- Ratnaweera, A. C., Halgamuge, S. K., & Watson, H. C. (2003). Particle swarm optimization with self-adaptive acceleration coefficients. In *Proc. of the first int. conf. on fuzzy systems and knowledge discovery* (pp. 264–268).
- Riget, J., & Vesterstrom, J. S. (2002). A diversity-guided particle swarm optimizer—The ARPSO. *Technical report*. Department of Computer Science, University of Aarhus.
- Reed, R. (1993). Pruning algorithms—A survey. *IEEE Transactions on Neural Networks*, 4(5), 740–747.
- Richards, M., & Ventura, D. (2003). Dynamic sociometry in particle swarm optimization. In *Proc. of the sixth int. conf. on computational intelligence and natural computing* (pp. 1557–1560).
- Salerno, J. (1997). Using the particle swarm optimization technique to train a recurrent neural model. *IEEE International Conference on Tools with Artificial Intelligence*, 45–49.
- Settles, M., Rodebaugh, B., & Soule, T. (2003). Comparison of genetic algorithm and particle swarm optimizer when evolving a recurrent neural network. *Lecture notes in computer science (LNCS): Vol. 2723. Proc. of the genetic and evolutionary computation conference 2003* (pp. 151–152).
- Sexton, R. S., & Dorsey, R. E. (2000). Reliable classification using neural networks: A genetic algorithm and back propagation comparison. *Decision Support Systems*, 30, 11–22.
- Shi, Y., & Eberhart, R. C. (1998). A modified particle swarm optimizer. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 69–73).
- Shi, Y., & Eberhart, R. C. (2001). Fuzzy adaptive particle swarm optimization. In *Proc. of the IEEE congress on evolutionary computation: Vol. 1* (pp. 101–106). IEEE Press.
- Suganthan, P. N. (1999). Particle swarm optimizer with neighborhood operator. In *Proc. of the IEEE congress on evolutionary computation* (pp. 1958–1962). IEEE Press.
- Sutton, R. S. (1986). Two problems with backpropagation and other steepest-descent learning procedures for networks. In *Proc. 8th annual conf. cognitive science society* (pp. 823–831).
- Van den Bergh, F. (2002). An analysis of particle swarm optimizers. Ph.D. thesis. Pretoria, South Africa: Department of Computer Science, University of Pretoria.
- Van den Bergh, F., & Engelbrecht, A. P. (2002). A new locally convergent particle swarm optimizer. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 96–101.
- Wilson, E. O. (1975). *Sociobiology: The new synthesis*. Cambridge, MA: Belknap Press.
- Xie, X., Zhang, W., & Yang, Z. (2002a). A dissipative particle swarm optimization. In *Proc. of the IEEE congress on evolutionary computation*. vol. 2 (pp. 1456–1461).
- Xie, X., Zhang, W., & Yang, Z. (2002b). Adaptive particle swarm optimization on individual level. In *Proc. of the sixth int. conference on signal processing*. vol. 2 (pp. 1215–1218).
- Xie, X., Zhang, W., & Yang, Z. (2002c). Hybrid particle swarm optimizer with mass extinction. In *Proc. of the int. conference on communication, circuits and systems*. vol. 2 (pp. 1170–1173).
- Yao, X., & Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3), 694–713.
- Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9), 1423–1447.
- Yu, J., Xi, L., & Wang, S. (2007). An improved particle swarm optimization for evolving feedforward artificial neural networks. *Neural Processing Letters*, 26(3), 217–231.
- Zhang, W.-J., & Xie, X.-F. (2003). DEPSO: Hybrid particle swarm with differential evolution operator. *Proceedings of the IEEE International Conference on System, Man, and Cybernetics*, 4, 3816–3821.
- Zhang, C., & Shao, H. (2000). An ANN's evolved by a new evolutionary system and its application. In *Proc. of the 39th IEEE conf. on decision and control*. vol. 4 (pp. 3562–3563).
- Zhang, J. R., Zhang, J., Lok, T. M., & Lyu, M. R. (2007). A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training. *Applied Mathematics and Computation* 185, 1026–1037.
- Zhou, Z. H., Chen, S. F., & Chen, Z. Q. (2000). FANNC: A fast adaptive neural network classifier. *Knowledge and Information Systems*, 115–129.