ANN Models Optimized using Swarm Intelligence Algorithms

N. KAYARVIZHY[1], S. KANMANI[2], R.V. UTHARIARAJ[3]
[1]Assistant Professor, Department of Computer Science and Engineering
AMC Engineering College
12th K.M., Bannerghatta Road, Bangalore – 560083
[2]Professor, Department of Information Technology
Pondicherry Engineering College
Puducherry – 605014
[3]Professor and Director, Ramanujam Computing Centre,
Anna University, Chennai – 25, Tamil Nadu
INDIA
[1]kayarvizhy@gmail.com, [2]kanmani@pec.edu, [3]rhymend@annauniv.edu

Abstract: - Artificial Neural Network (ANN) has found widespread application in the field of classification. Many domains have benefited with the use of ANN based models over traditional statistical models for their classification and prediction needs. Many techniques have been proposed to arrive at optimal values for parameters of the ANN model to improve its prediction accuracy. This paper compares the improvement in prediction accuracy of ANN when it is trained using warm intelligence algorithms. Swarm intelligence algorithms are inspired by the natural social behaviour of a group of biological organisms. Models have been formulated for evaluating the various ANN-Swarm Intelligence combinations. Fault prediction in Object oriented systems through the use of OO metrics has been considered as the objective function for the models. The swarm intelligence algorithms considered in this paper are Particle Swarm Optimization, Ant Colony Optimization, Artificial Bee Colony Optimization and Firefly. The object oriented metrics and fault information for the analysis have been taken from NASA public dataset. The models are compared for their convergence speed and improvement in prediction accuracy over traditional ANN models. The results indicate that Swarm Intelligence Algorithms bring improvement over ANN models trained with gradient descent.

Key-Words: - Artificial Neural Network, Swarm Intelligence, Particle Swarm Optimization, Ant Colony Optimization, Artificial Bee Colony Optimization, Firefly

1 Introduction

Artificial Neural Network (ANN) is a mathematical model inspired by the biological neural networks. ANN is a non-linear mapping model and has been successfully applied in many domains like bankruptcy prediction [6], [34], [57], handwriting recognition [20], [31], product inspection [50], [35] and in fault detection [8], [22]. ANN is also a widely accepted choice of fault prediction model [2], [27], [30]. ANN is adaptive as it can change its structure based on the information that flows through the network. This adaptability is achieved by training the ANN with known data set. In an ANN based prediction model, prediction accuracy can be improved by finding optimal parameter values for the model.

Optimal values can be found for ANN parameters like number of input neurons, hidden layers, hidden neurons, activation function and weight values. Since weight values are the key to a well trained ANN, finding the optimal weight values of ANN has been considered in many research studies. ANN can be trained using many techniques. Gradient Descent (GD) algorithm is a widely used ANN training technique due to its inherent simplicity and ease of implementation. In GD, an error function computes the difference between the observed and predicted values and hill climbing or descent is used to find the weight values which reduce the error. Studies have confirmed that gradient descent is prone to the following problems – it may get stuck at a local optimum [28] and it may take a very long time to converge [47].

Table 1 List of Swarm Intelligence Algorithms

Algorithm	Author	Year
Particle Swarm Optimization	Kennedy & Eberhart	1995
Ant Colony Optimization	Dorigo & Di Caro	1999
Artificial Bee Colony	Karaboga & Basturk	2007
Artificial Cooperative Search	Pinar Civicioglu	2012
Artificial Immune Systems	Dasgupta	1999
Bat Algorithm	Xin-She Yang	2010
Charged System Search	Kaveh & Taletahari	2010
Cuckoo search	Xin-She Yang	2009
Differential search algorithm	Pinar Civicioglu	2012
Firefly algorithm	Xin-She Yang	2009
Glowworm swarm optimization	Krishnanand & Ghose	2005
Gravitational search algorithm	Saryazdi et al.	2009
Intelligent water drops	Shah-Hosseini	2007
Krill herd algorithm	Gandomi & Alavi	2012
Magnetic optimization algorithm	Tayarani	2008
Multi-swarm optimization	Blackwell & Branke	2004
River formation dynamics	Rubio et. al.	2007
Self-propelled particles	Vicsek et al.	1997
Stochastic diffusion search	Bishop	1992
Group Search Optimizer	He et. al	2006
Fish Swarm Algorithm	Li et. al	2002
Bacterial Foraging Optimization	Passino	2002
Shuffled Frog Leaping Algorithm	Muzzaffar & Kevin	2003

This led researchers to focus on the ANN training mechanism. Swarm intelligence algorithms are a set of generic, population based, meta-heuristic optimization algorithms. They are inspired by the natural process of social-behavior among a group of organisms. Swarm algorithms are known to have good exploration and exploitation capabilities in solution space compared to traditional algorithms. This helps them to avoid getting caught in a local

optimal solution. Swarm intelligence algorithms have been adopted for use in various optimization problem domains. A number of swarm intelligence algorithms have been proposed. Table 1 gives details on some of the swarm intelligence algorithms that have been proposed. We consider four popular swarm intelligence algorithms for our study – Particle Swarm Optimization (PSO), Firefly, Artificial Bee Colony (ABC), Ant Colony Optimization (ACO).

2 Related Work

ANN has been used in a wide variety of applications like fault detection in Analog Circuits [62], classification of electrical disturbances [45], classifying structured patters [58], classification in time varying environment [52], electro-cardiogram analysis [48], [25]. ANN has also been successfully used in the food industry [14], power transmission [3] and noisy data classification [59]. Many research studies have focused on ANN Training. Traditionally, feed forward ANN has been trained with gradient descent using back propagation [51]. Improvements on the back propagation techniques like scale gradient conjugate back propagation [44], conjugate gradient back propagation with Polak-Riebre updates , conjugate gradient back propagation with Fletcher-Reeves updates , one secant back propagation [64], resilient back-propagation [40] were attempted. Other attempts include Marquardt algorithm [21], [56] enhanced heuristic training [11], [53] accelerated convergence [36], [43], [24], recursive training [4] and adaptive algorithms [32], [49].

ANNs have been trained using Evolutionary algorithms. Evolutionary algorithms are a subset of evolutionary computation in Artificial Intelligence. They are a general population based meta-heuristic optimization algorithms. Evolutionary algorithms are inspired by the biological process of evolution, reproduction, mutation, recombination and selection. Genetic Algorithms (GA) are one of the most popular evolutionary algorithms. Ganatra, et al. [19] investigated the application of genetic algorithm in training an artificial neural network. They conclude that ANN-GA has increased convergence speed and the local optima problem is overcome compared to ANN-BP. Feng, et. al. [17] used ANN-GABP for estimating cost of software projects. Compared with the general ANN-BP model, the ANN-GABP model has lower forecast errors in lesser iterations. They conclude that GABP model is appropriate for construction cost estimation. Alba and Chicano [5] have trained ANN with hybrid GA. Other evolutionary algorithms like Differential Evolution algorithms have been applied to ANN with good results [23], [55].

Swarm Intelligence algorithms have been applied to train ANN [69], [70], [71]. Particle Swarm Optimization is a heuristic algorithm proposed by Kennedy [29]. Bashir and El-Hawary [9] have employed the PSO technique to train ANN for load forecasting. Yogi et. al [63] have applied PSO to train a functional link ANN and noted that it improves the classification capability compared to traditional neural networks. Zhang and Wu [65] have used an adaptive chaotic PSO to train ANN for crop classification and have confirmed the superiority of ANN-PSO to various BP techniques. Other ANN-PSO studies also confirm an improvement over traditional ANN models [7], [54], [37]. Firefly algorithm [61], is a popular swarm intelligence algorithm. Nandy, et al. [46] has trained ANN for dynamic systems using Firefly and concluded that Firefly trained ANN takes lesser iterations for convergence compared to traditional ANN based on BP. Ant colony optimization [15] is a probabilistic technique in the swarm intelligence family. Kumar et al. [33] trained ANN using Ant colony optimization and applied it to biometric fusion. They found that it contributed significantly. Chen, et al. [12] constructed an ANN based on Ant colony optimization. They used it for optimizing discounted cash flows in project scheduling. McMullen [42] applied ANN-Ant colony for JIT sequencing problem. Artificial Bee Colony algorithm is another swarm intelligence algorithm that has been successfully used to train ANN. It has been used for software defect prevention by Farshidpour and Keynia [16]. They benchmarked the performance of ANN-ABC against ANN trained with the standard BP. Their experimental result showed that the performance of ANN-ABC was better than ANN-BP. Zhang et. al. [66] constructed a scaled chaotic artificial bee colony (SCABC). They used SCABC as training algorithm for an ANN and compared it with traditional training methods like BP, momentum BP, GA, elite GA with migration, simulated annealing. The used it for magnetic resonance brain image classification. They concluded that SCABC performs better than these algorithms both in lesser mean square error and better classification accuracy. ANN-ABC has been used in other classification problems [41].

There is a need for comparing various swarm intelligence algorithms in their ability to train ANN for predicting faults in OO systems. We also felt a need for testing different set of algorithm parameters to arrive at the optimal set of parameters giving the best prediction accuracy. This motivated

us to build an ANN model for predicting object oriented software quality using OO metrics and train it using different swarm intelligence algorithms.

3 Metrics

A key element of any engineering process is measurement. Measurement can be used to better understand the attributes of the models that are created and used to assess the quality of engineered products or system built. Measure of internal product attributes provides a real-time indication of the efficacy of the requirements, design, code and test cases and the overall quality of the software to be built [18].

Software quality of systems depends on the internal attributes of the software like size, coupling, and cohesion. These internal attributes can be measured and assigned a value - software metrics. Software quality is reflected only through its external attributes like reliability, usability and efficiency. These external attributes are referred to as the quality indicators. Quality prediction models are functions that map the internal attributes to the external attributes. By predicting the quality indicators (external) using the software metrics (internal), it becomes easy to control the quality of the product. During the development of a software product the internal attributes are measured and fed to the model. Based on the prediction made by the model for the quality indicator, necessary corrective and preventive actions can be taken. This ensures that quality is managed even during the design and development stage of a software product.

Object oriented systems continue to share a major portion of software development and customer base for these systems is on the rise. This is because there are huge incentives in taking the object oriented approach. Object oriented paradigm has features like encapsulation, data abstraction, inheritance, polymorphism etc. to model complex OO systems. A lot of time, money and effort are spent in ensuring the quality of these systems. Chidamber & Kemerer [13] proposed six measures which are the most widely used design measures for object oriented systems focusing on class and class hierarchy. The metrics in the CK Metric suite are Lack of Cohesion of Methods (LCOM), Coupling Between Objects (CBO), Depth of Inheritance Tree (DIT), Number of Children (NOC), Weighted Methods per Class (WMC) and Response for a Class (RFC). After the inception of the CK metric suite, many have introduced new class level design metrics. Li and Henry [38] introduced a set of measure for the maintenance of OO system and are validated by regression analysis. Lorenz and Kidd [39] categorized class measures into four broad categories as size, inheritance, internals and externals. Abreu et. al. [1] proposed a set of metrics, which can be applied at system level. Basili et. al. [10] introduced a set of coupling measures for C++. Xenos, et al. [60] surveyed the existing OO measures and evaluated them. Thus there are many metrics suites proposed in the literature. The general conclusion is that these metrics are important indicators of external quality factors.

Design level metrics play an important role since a bug is less costly to fix when caught in the design phase and is not allowed to propagate further. We have chosen the design level metrics proposed by Chidamber and Kemerer [13].

4 Artificial Neural Network

Artificial Neural Network (ANN) is a simplified model of the human nervous system. It is composed of many artificial neurons that co-operate to perform the desired functionality. ANN is an approximation function mapping inputs to outputs. The ability to learn and adapt to the data set makes ANN applicable in a variety of fields. The output is based on the 'n' inputs values (x_i) and weights (w_i) [67]. It is given by Eq. (1)

$$Output = \sum_{i=1}^{n} x_i w_i \qquad (1)$$

The example shows a sigmoidal activation function in Eq. (2), where y is the slope parameter and 'x' is the input,

$$\emptyset(x) = \frac{1}{1 - e^{-yx}} \qquad (2)$$

The artificial neurons can be combined to form an artificial neural network. The neurons in input layer map the various inputs values. One or more hidden layers with neurons map the input neurons to the output neurons. The number of output neurons depends on the number of output variables that we plan to map. Weights exist between input-hidden layers and hidden-output layers.

ANN exhibits some remarkable properties like adaptability, learning by examples and generalization which makes it an ideal candidate for pattern classification problems. Fault prediction is a subset of classification problem where the fault prone modules need to be identified and tagged. In the case of object oriented systems the lowest level of abstraction is a class. The prediction models like ANN identify classes that could be faulty and tag them. The development team can then work on the

tagged classes and design them better. The first step in the creation of a good fault prediction model based on ANN, involves providing known information with which the model can be trained. In our case we need to have class level metrics data along with the fault details. The ANN model is trained using this information. Once trained the ANN model is ready to be used on new data set where only the metrics are known and fault has to be predicted. The ANN model is used to predict whether the classes in the new data set are likely to be faulty or not. A well trained ANN has a higher probability of predicting fault-prone classes. The probability is captured by the prediction accuracy which is the percentage of correct classifications compared to the overall classification. The prediction accuracy of an ANN fault prediction model depends on various factors like model parameters, dataset, domain etc

4.1 ANN Parameter Optimization

ANN is a complex model and its prediction accuracy can be improved by optimizing its parameters. The parameters that can be optimized in an ANN can be grouped under the following categories

1. Architecture
    a. Number of input neurons
    b. Number of hidden layers and hidden neurons
    c. Number of output neurons
2. Training
    a. Weights
    b. Training algorithm
    c. Training epochs
3. Transfer function
4. Data
    a. Selection
    b. Pre-processing
    c. Quantity and quality

5 Swarm Intelligence Algorithms

A swarm is defined as a large number of insects or other organisms. Swarm behavior is a collective behavior exhibited by the swarm with the ability to communicate directly or indirectly with each other. The swarm collectively carries out a distributed problem solving through the swarm behavior. Swarm intelligence [29] is the discipline that deals with natural and artificial systems composed of many individuals that coordinate using decentralization and self organization. The individual organisms in the swarm follow very simple rules with no central control. Even random communication between the organisms in the swarm eventually results in an intelligent overall behavior. The organisms might not be even aware of this global intelligence. Many swarm intelligence exists in nature like termite colonies, ant colonies, bird flocking, bee colonies, schools of fish etc.

There are certain common traits in all swarm intelligence algorithms. The swarm is composed of many individual organisms. The organisms are homogeneous and communication between the organisms is based on set rules. The organisms have limited intelligence as individuals but they can carry out simple tasks. Swarm intelligence algorithms based on social behavior of biological organisms is analyzed further in this section.

5.1 Particle Swarm Optimization

PSO employs social learning concept to problem solving. Birds flocking together generally exchange valuable information on the location of the food. When a bird learns of a promising location, its experience grows about the surrounding. This is hugely enhanced when the birds share the information with one another, boosting the swarm's intelligence. This helps other birds to converge on the most promising food location. PSO is widely applied in many research areas and real world applications as a powerful optimization technique. Simulating the natural behavior, the PSO algorithm has a set of particles that fly around an n-dimensional problem space in search of an optimal solution. To start with, the particles are distributed randomly in the solution space.

Each particle P in the swarm S is represented as {X, V} where X = {x1, x2, x3… xn} represents the position of the particle and V = {v1, v2, v3… vn} represents the velocity of the particle. In every iteration, the particles learn from each other and update their knowledge regarding the whereabouts of a good solution. Each particle keeps track of its best solution with its corresponding position in pbest and the swarm's best position is tracked in gbest. Each particle will have the influence of its current direction, the influence of its memory (pbest) and the influence of the swarm's intelligence (gbest).

The particles update their velocity and position based on the formula given in Eq. (3) [29]. Here 'i' represents the particle number, 'd' represents the dimension, 'V' is the velocity, 'p' is the pbest, 'g' represents gbest, 'w' is the inertia weight, c1 and c2 are the constants for controlling the influence of pi and g respectively, 'x' is the current position and 'rp' and 'rg' are random numbers between 0 and 1.

$$V_{id} = w * V_{id} + c_1 r_p (p_{id} - x_{id}) + c_2 r_g (g_d - x_{id}) \quad (3)$$

PSO is simple to implement with less number of parameters to adjust. PSO has been used successfully in function optimization, neural network training and many more fields requiring optimization.

Pseudo code of PSO algorithm

g_{best} = *NULL*

for *i=1 to total_particles*

particle.pos = rand(pos_{lower}*,* pos_{upper}*)*

$particle_{best}.pos = x_i$

$particle_{best}.obj$ = *objfunc (*x_i*)*

if *(*$particle_{best}.obj < g_{best}.obj$*)*

$g_{best} = particle_{best}$

end if

particle.vel = rand($velocity_{lower}$*,* $velocity_{upper}$*)*

end for

do

for *i=1 to total_particles*

for *d =1 to total_dimension*

pick random numbers : r_p*,* r_g *between 0 and 1*

$particle_{id}.vel = w * particle_{id}.vel + c_1 r_p (particle_{best}.pos - particle_{id}.pos) + c_2 r_g (g_{best}.pos - particle_{id}.pos)$

$particle_{id}.pos = particle_{id}.pos + particle_{id}.vel$

$particle_{id}.obj = objfunc(particle_{id}.pos)$

if *(*$particle_{id}.obj < particle_{best}.obj$*)*

$particle_{best} = particle_{id}$

if *(*$particle_{best}.obj < g_{best}.obj$*)*

$g_{best} = particle_{best}$

end if

end if

end for

end for

while *(*$g_{best}.obj > obj_{desired}$ *OR iteration <* $iteration_{max}$*)*

5.2. Ant Colony Optimization

ACO is among the most successful swarm based. It is well suited for discrete optimization problems and is a probabilistic method for solving computational problems. ACO is inspired by the behavior of ants in finding paths from their colony to food sources.

Biological ants, in the real world, randomly search for food sources around their colony. But once they find a food source, they return to their colony leaving behind pheromone trails. When other ants sense the trail they tend to follow the trail instead of taking a random approach. They also leave behind pheromones and thus reinforce the trail. Pheromone is a chemical that evaporates after a period of time and thus reducing in its ability to attract ants. If the path is longer between the food source and colony the pheromones in that path have a greater chance of evaporating compared to a shorter path. The shorter path will get reinforcements frequently and hence sustain its pheromone concentration.

The ACO algorithm will search for an 'n' dimensional solution in the search space, X = {x1, x2, x3… xn}. At every step of the ACO algorithm there is a selection process for the next node, xi. A probability function based on the current pheromones levels is used. The desirability of having nodes xi, xj in the solution is given by Dij. If the remaining nodes to be processed are in a set S (xi) then the probability of each of those nodes in the set is given in Eq. (4) [15].

$$P_{ij} = \frac{D_{ij}}{\sum_{i,j=0}^{n} x_j \in N(x_i) D_{ij}} \quad (4)$$

The pheromone evaporation helps the ACO algorithm to avoid converging on a local optimal solution. In the absence of pheromone evaporation the first random path would have been reinforced and considered as the final solution. This would have resulted in the algorithm not doing any search

on the solution space and considering the first solution as final solution. But with pheromone evaporation this problem is solved and when an ant finds a shorter path from the colony to a food source, other ants are more likely to follow the new shorter path. This continues and the pheromone distribution towards shorter paths eventually leaves all the ants following the single shortest path

Pseudo code of ACO algorithm

init (ants)

init trail T_i

best trail $T_b = T_i$

$obj_{best} = objfunc(T_b)$

initialize pheromone on the trail

iteration = 0

do

new trail T_n = ***function****(ants_old, pheromone, random)*

for *i=1 to total edges*

$pheromone\_decrease = P_{df} * pheromone\_old$

$pheromone\_increase = i\ \varepsilon\ T_n$

pheromone_new = pheromone_decrease + pheromone_increase

end for

obj = $objfunc(T_n)$

if $obj < obj_{best}$

$obj_{best} = obj$

end if

while $(obj_{best} > obj_{desired}$ *OR iteration* $< iteration_{max})$

5.3. Artificial Bee Colony Optimization

The swarm intelligence algorithms based on bees are formulated on their foraging behavior. It includes artificial bee colony (ABC), the virtual Bee algorithm, the bee colony optimization algorithm and Bee hive algorithm. In all bee swarm optimization algorithms, an individual bee exhibits a simple set of behaviors. The group shows a complex overall behavior with useful properties such as scalability and adaptability.

In ABC algorithm, the colony of artificial bees can be grouped into three types: employed bees, onlookers and scouts. Each food source discovered so far is tracked by an employed bee. Hence, the number of food sources is equal to employed bees. The food source is the solution candidate and is coded with an 'n' dimensional vector X = {x_1, x_2, x_3… x_n}. The location of food sources map to the search space of the problem and the best food source is the most qualified optimized solution for the problem. An employed bee computes a modified position from her memorized food location depending on the local information and tests the quality of food on the new source (new solution). If the new location is a better food source compared to the previous one, the bee memorizes the new position and forgets the old one. After all employed bees complete the search process they share new food source information and their position information with the onlooker bees on the dance area. Onlooker bees wait on the dance area and select food sources based on the dance performed by the employed bees. The onlooker bees choose the solution based on the Eq. (5) below [68]

$$P_i = \frac{ObjFunc_i}{\sum_{i=1}^{FS} ObjFunc_i} \qquad (5)$$

Onlooker bees visit the food source that they select and identify a nearby modified source. They evaluate and choose between the original and new source. To get a nearby food source Eq. (6) is used, where Rand is a random number between -1 and 1

$$Nearby_{ij} = x_{ij} + Rand(x_{ij} - x_{kj}) \qquad (6)$$

The employed bees whose sources were abandoned become scouts and go in search of new food sources. The scout discovers a new food source by employing Eq. (7), where Rand is a random number between 0 and 1

$$New_{ij} = x_{jmin} + Rand(x_{jmax} - x_{jmin}) \qquad (7)$$

The algorithm avoids getting into local optimum by having the scouts perform a random global search for new food sources.

Pseudo code of ABC algorithm

init $total_{bees}$

init $bees_{employed}$, $bees_{onlooker}$, *scout_limit*

totalobj = 0

for *i=1 to* $bees_{employed}$

bee_i *= random (for each dimension)*

end for

do

for *i=1 to* $bees_{employed}$

dim = random(1,dimensions)

$beemodified_i$ *= random (*bee_i*,dim)*

$obj_{modified}$ = ***function****(*$beemodified_i$*)*

obj_i = ***function****(*bee_i*)*

if *(*$obj_{modified}$ > obj_i*)*

obj_i = $obj_{modified}$

bee_i = $beemodified_i$

end if

totalobj = totalobj+obj_i

end for

for *i=1 to* $bees_{employed}$

$prob_i$ *= 0.9 * obj_i*/totalobj + 0.1*

end for

for *i=1 to* $bees_{onlooker}$

bee_{chosen} *= choose employed bee based on* $prob_i$

dim = random(1,dimensions)

$chosenmodified_i$ *= random (*bee_{chosen}*,dim)*

$obj_{modified}$ = ***function****(*$chosenmodified_i$*)*

obj_i = ***function****(*bee_{chosen}*)*

if *(*$obj_{modified}$ > obj_i*)*

bee_{chosen} = $chosenmodified_i$

end if

end for

for *i=1 to* $bees_{employed}$

if *not chosen for limit times*

bee_i *= random (for each dimension)*

end if

end for

while *(*obj_{best} > $obj_{desired}$ *OR iteration <* $iteration_{max}$*)*

5.4. Firefly algorithm

The Firefly algorithm (FA) is a meta-heuristic algorithm that is inspired by the flashing behavior of fireflies. It can be used for constrained optimization tasks. The flashing behavior of fireflies occurs due to the bioluminescence phenomenon. Fireflies can control their flashing behavior based on external stimulus. They use it to attract other fireflies or prey. In a Firefly algorithm, a population of fireflies is considered. These fireflies get attracted to each other based on the intensity of light that they emit. The firefly would get attracted towards a firefly that has highest luminescence. The solution space is mapped on to the fireflies and the quality of solution of each firefly is directly proportional to the level of intensity of its flashing. Thus fireflies with better solutions attract its partners (regardless of their sex), which makes the search space exploration efficient. The candidate solutions mapped as fireflies are denoted by X = {x_1, x_2, x_3… x_n}. The fireflies 'i' and 'j' move towards each other based on the Eqs. (8)-(10) below where $Dist_{ij}$ is the distance between the two fireflies, β, γ are algorithm optimization parameters.

$$Dist_{ij} = \sqrt{\sum_{i,j,k=1}^{n} \left(x_{ik} - x_{jk}\right)^2} \quad (8)$$

$$\beta = \beta_0 e^{-\gamma Dist_{ij}} \quad (9)$$

$$x_{ik} = (1 - \beta)x_{ik} + \beta x_{ik} + Rand \qquad (10)$$

Three basic rules are followed during the movement of firefly towards each other.

- All fireflies are unisex and hence they would move towards the brighter firefly regardless of the sex.
- The degree of attractiveness of a firefly to another firefly is directly proportional to its brightness. So the less bright firefly would move towards the brighter firefly. Distance is inversely proportional to brightness. As the distance increases between two fireflies the perceived brightness decreases between them. This is because of the fact that the air absorbs light. If there is no brighter firefly than a particular one it will then move randomly.
- The objective function of a problem domain determines the brightness or light intensity of a fire fly.

Pseudo code of Firefly algorithm

for *i=1 to* $num_{fireflies}$

$firefly_i$ *= random(for all dimensions k)*

I_i = ***function****(*$firefly_{ik}$*)*

end for

iteration = 0

do

for *i = 1 to n*

for *j=1 to n*

for *k = 1 to dimensions*

distance = $firefly_{ik} - firefly_{jk}$ ***

$firefly_{ik} - firefly_{jk}$

end for

distance **= *sqrt(****distance****)***

if (*$I_i > I_j$****)***

for *k = 1 to dimensions*

*T = alpha * (distance – 0.5) **

(upper – lower)

$firefly_{ik} - firefly_{ik}$ ** (1 – beta) +*

$firefly_{ik}$ ** beta + T*

end for

end if

update Ii = ***function****(*$firefly_i$*)*

end for

end for

rank fireflies and find current best

while *(*obj_{best} *>* $obj_{desired}$ *OR iteration <* $iteration_{max}$*)*

6 ANN Training using Swarm Intelligence Algorithms

Adapting the swarm intelligence algorithm to train the ANN involves the following generic steps irrespective of which Swarm Intelligence Algorithm is considered. Since the weights of the ANN need to be optimized they need to be tracked as the fundamental entity depending on the Swarm Intelligence algorithm. The problem space contains the combinations of all possible weight values for the ANN. This search space is of n-dimensions where n is the total number of weights that needs to be optimized. The Swarm Intelligence Algorithm is applied and objective function is based on the prediction accuracy of ANN. The weights are mapped to the required entity of the algorithm, like a particle's position in PSO, the position of firefly or the trails that ants need to take in ACO. While evaluating the fitness in Swarm Intelligence, the weights are assigned to the ANN and its prediction accuracy is found. If the fitness is the best so far it will be taken as the best combination of weights and hence the best solution so far according to the Swarm Intelligence Algorithm. The steps for a ANN optimized using Swarm Intelligence is given below and summarized below in Fig. 1.

RFC	Response For a Class
LCOM	Lack of Cohesion in Methods
LCOM3	Lack of Cohesion in Methods
CA	Afferent Coupling
CE	Efferent Coupling
NPM	Number of Public Methods
DAM	Data Access Metric
MOA	Measure of Aggregation
MFA	Measure of Functional Abstraction
CAM	Cohesion Among Methods
IC	Inheritance Coupling
CBM	Coupling Between Methods
AMC	Average Method Complexity
CC	McCabe's Cyclomatic Complexity
LOC	Lines of Code

7.2. Comparison of ANN-GD with ANN-Swarm Intelligence Algorithms

Five ANN models were used in this study. ANN architecture of all the models had one input layer with 20 neurons mapping each of the metric in the data set. We opted for one hidden layer with 50 hidden neurons after testing a wide variety of values. We had one output layer with a single neuron to map the prediction accuracy of the model. While this particular ANN architecture cannot be considered as optimal, it is still sufficient to compare the accuracy and performance of the various ANN models. The parameters of swarm intelligence algorithms were arrived at after experimenting with different values taken from literature survey. The models were implemented in Visual C++ IDE as integrated software. The software picks different comma separated value (CSV) data files containing metrics and bug data from a folder and executes the models with this data. It computes the accuracy and time taken by each algorithm. The software was run on a Dell Inspiron 1525 laptop with a dual core processor running at 1.2 GHz, 2 GB of RAM and using Windows 8 operating system.

The comparison of the models was based on prediction accuracy – percentage of correctly mapped bug data and time taken in seconds to converge at the solution.

7.2.1. Training using back propagation

Gradient descent is the most widely used back propagation algorithm for training the ANN. It is simple to implement but has disadvantages of getting caught in a local optima. It also takes a longer duration to converge on the optimal solution. The learning rate chosen was 0.001. Momentum was set at 0.9 and 1000 epochs were performed on the input data sets. The activation function was a simple sigmoidal function

7.2.2. Training using ANN-PSO algorithm

Adapting the PSO algorithm to train the ANN involves the following steps. Since the weights of the ANN need to be optimized, they need to be tracked as the position of the particles in the PSO algorithm. The problem space contains the combinations of all possible weight values for the ANN. This search space is of n-dimensions where n is the total number of weights that needs to be optimized. Each particle has a position vector and a velocity vector of n-dimensions. The PSO particles fly around this search space and come up with the optimal set of weights. While evaluating the fitness of a particle in PSO, the weights are assigned to the ANN and its prediction accuracy is found. This provides the fitness for the particle. If the fitness is the best so far for the particle it will be taken as its personal best and if it is the best so far for the swarm, it would be considered as global best. The global best position after a desired number of iterations yield the optimized weights for the ANN.
The following PSO parameters were used. Twenty particles were used for 100 iterations. The inertia weight was set at 0.529. Both local (c1) and global weights (c2) were set to 1.4944

7.2.3. Training using ANN-Firefly algorithm

In the ANN-Firefly algorithm we mapped the weights as the position of the fireflies. The flashing behaviour of the firefly indicates the prediction accuracy of the ANN using the position of the firefly as training weights. Thus an ANN with higher prediction becomes the most attractive firefly and other fireflies converge towards it. The parameters of an ANN-Firefly algorithm were taken after analyzing similar studies in the literature. The number of fireflies was taken as 20 and the iterations were kept at 100. Alpha, beta and gamma were taken as 0.5, 0.2 and 1.0.

Table 3 Statistical highlights of the data

Metric	Min	25Q	Mean	Median	75Q	Max	SD
WMC	0	3	9.55	6	12	252	12.90
DIT	0	1	1.88	1	3	6	1.23
NOC	0	0	0.47	0	0	39	2.36
CBO	0	3	9.75	6	11	448	18.13
RFC	0	6	23.83	15	30	511	30.22
LCOM	0	0	94.11	4	33	29258	654.13
CA	0	1	4.72	1	4	446	17.19
CE	0	0	4.68	3	7	76	6.26
NPM	0	2	7.84	4	9	231	11.54
LCOM3	0	0.58	1.10	0.85	2	2	0.70
LOC	0	21	167.69	63	171	7956	355.01
DAM	0	0	0.61	1	1	1	0.48
MOA	0	0	0.74	0	1	24	1.50
MFA	0	0	0.36	0	0.82	1	0.41
CAM	0	0.30	0.49	0.44	0.67	1	0.26
IC	0	0	0.35	0	1	4	0.58
CBM	0	0	0.66	0	1	21	1.76
AMC	0	4.5	14.14	8.86	16.33	894.5	24.88
MAX_CC	0	1	2.68	1	3	95	4.13
AVG_CC	0	0.67	1.02	0.96	1.14	10	0.73
FAULTS	0	0	0.39	0	0	28	1.36

7.2.4. Training using ANN-ABC algorithm

In ANN-ABC the weights of ANN are mapped to the food sources of ABC. The fitness values of the food source are measured with an objective function. The objective function provides the prediction accuracy through ANN. A colony size of 40 was considered for the experiment out of which 20 were employed bees tracking as many food sources. The improve limit was kept at 100. So, any employed bee, whose food source could not be improved after 100 trials, will become a scout and search for a new food source to track. The foraging cycle was kept at 100.

7.2.5. Training using ANN-ACO algorithm

In ANN-ACO the ant trail maps the weights of ANN. The fitness of the trail is evaluated based on the performance of ANN. The pheromone effect helps the ANN-ACO from getting caught in local optima. The algorithm takes 100 ants with 200 stops for each ant. The pheromone evaporation time was fixed at 20 iterations.

7.2.6. Results

The comparison results of ANN-GD and ANN-Swarm intelligence algorithms have been listed in Table 4 and Table 5.

Table 4 Prediction Accuracy

Project	ANN-GD	ANN-PSO	ANN-Firefly	ANN-ABC	ANN-ACO
Arc	67.521	87.606	88.462	88.462	85.043
Camel 1.0	94.395	96.165	95.575	96.165	96.165
Camel 1.2	19.079	64.474	43.750	52.467	37.665
Camel 1.4	33.028	83.372	46.560	77.523	74.427
Camel 1.6	26.218	80.518	20.725	53.264	34.715
Intercafe	81.482	85.185	81.482	85.185	85.185
Tomcat	68.532	91.026	86.131	88.811	88.695

Table 4 indicates the prediction accuracy of the various algorithms for different input and Table 5 lists the time taken. ANN-GD has an average fault prediction accuracy of 55.75% at an average of 21.038 seconds per run. All the four models trained using swarm intelligence algorithms perform better than ANN-GD. ANN-Firefly has an average fault prediction improvement percentage of 18.559% over ANN-GD and the average time taken is 22.796 seconds. ANN-ACO has fault prediction accuracy better than ANN-GD by 28.606% and the average time taken is 5.568 seconds. ANN-ABC has an improvement of 38.852% with average time of 56.339.

Table 5 Time Taken (in seconds)

Project	ANN-GD	ANN-PSO	ANN-Firefly	ANN-ABC	ANN-ACO
Arc	6.133	1.793	1.693	0.971	4.998
Camel 1.0	9.562	2.212	1.133	1.579	7.181
Camel 1.2	4.446	5.052	54.609	138.24	3.446
Camel 1.4	16.875	11.069	46.198	157.49	5.874
Camel 1.6	53.591	14.167	47.163	88.857	13.206
Intercafe	0.002	0.033	0.355	1.396	1.109
Tomcat	56.661	2.319	8.421	5.84	3.167

The ANN-PSO provides the best fault prediction accuracy average of 84.04. This is a 50.75% improvement over ANN-GD. The time taken by ANN-PSO is also the least among the models considered, at an average of 5.235 seconds

7.3. Comparison with parameters used by existing ANN-Swarm Intelligence systems

In the second study we compared ANN-PSO and ANN-ABC with parameters used in existing systems. The motivation behind the study was to understand the impact of modifying the algorithm parameters based on an existing study. For ANN-PSO a study by Ardil & Sandhu [7] was considered. They had employed ANN-PSO for modelling severity of faults in software systems. Parameters from this system is denoted as ANN-PSORef in the Table 6 below, while the parameters used in the first study are retained as ANN-PSO.

Table 6 ANN-PSO parameters

Parameter	ANN-PSO	$ANN\text{-}PSO_{Ref}$
Number of Particles	20	25
Number of iteration	100	2000
Inertia Weight	0.529	0.9
Local weight (c1)	1.4944	2

Similar to ANN-PSO, we considered an existing system for ANN-ABC. The parameters were taken from an existing ANN-ABC study by Jin and Shu [26] and named as ANN-ABCRef. The reference system has more number of bees in the colony and hence more food sources. The foraging cycle or algorithm iterations in the reference system is 2000 compared to the 100 that we had for our ANN-ABC system.

Table 7 ANN-ABC parameters

Parameter	ANN-ABC	$ANN\text{-}ABC_{Ref}$
Colony size	40	50
No of food sources	20	25
Improve limit	100	100
Foraging cycle	100	2000

This would help us to compare the prediction accuracy and time taken between these two sets of values. The parameters are listed in Table 7

7.3.1. Results

The parameters for the ANN-PSO and ANN-ABC models given in the section above mainly differs in the size of the colony and the iterations. ANN-PSORef has 5 more particles and iterations increase from 100 to 2000 compared to ANN-PSO. Similarly ANN-ABCRef has 10 additional bees and the foraging cycle increases from 100 to 2000. Table 8 indicates the prediction accuracy and time taken for the ANN-PSO and ANN-PSORef models. Table 9 indicates the prediction accuracy and time taken in seconds for ANN-ABC and ANN-ABCRef models.

It can be seen from the results that in most cases the prediction accuracy of ANN-PSORef and ANN-ABCRef increases marginally or stays the same compared to ANN-PSO and ANN-ABC respectively. Minor improvments in prediction accuracy comes at a huge cost in time taken. The average prediction accuracy of ANN-PSORef increases by 0.312% compared to ANN-PSO. In case of ANN-ABCRef it is an increase of 8.763%. However the the time taken increases by an average of 16 seconds for ANN-PSORef and by 1300 seconds for ANN-ABCRef. In the dataset Camel 1.2 ANN-ABCRef increases from 52.467 to 64.638, an increase of 23.19%. For the same dataset, the time taken increases by 2268.158%. This is because, in general, swarm intelligence algorithm converge faster to a candidate solution. Further increase in the number of interations or organisms gives only minor improvements. Even the ANN- PSORef and ANN-ABCRef models converge in 100 iterations to the same level of accuracy as ANN-PSO and ANN-ABC modes. In further iterations it keeps trying to improve the accuracy. The time taken is logged only if the accuracy improves. Since the iterations are huge minor improvements are achieved but after significant number of iterations and hence the time taken increases drastically. So the parameters which are considered for the study based on literature review holds well and can be considered appropriate for the data sets chosen. They provide a good balance between prediction accuracy and time taken.

Table 8 Comparison of ANN-PSO and ANN-PSO_{Ref} models

Project	ANN-PSO	ANN-PSO_{Ref}	ANN-PSO	ANN-PSO_{Ref}
Arc	87.606	88.462	1.793	10.150
Camel 1.0	96.165	96.165	2.212	2.587
Camel 1.2	64.474	65.132	5.052	63.122
Camel 1.4	83.372	83.372	11.069	27.153
Camel 1.6	80.518	80.725	14.167	31.655
Intercafe	85.185	85.185	0.033	0.303
Tomcat	91.026	91.142	2.319	17.603

Table 9 Comparison of ANN-ABC and ANN-ABC_{Ref} models

Project	ANN-PSO	ANN-PSO_{Ref}	ANN-ABC	ANN-ABC_{Ref}
Arc	88.462	88.462	0.971	1.186
Camel 1.0	96.165	96.165	1.579	2.429
Camel 1.2	52.467	64.638	138.24	3273.742
Camel 1.4	77.523	83.372	157.49	413.754
Camel 1.6	53.264	80.518	88.857	654.411
Intercafe	85.185	85.185	1.396	2.864
Tomcat	88.811	91.026	5.84	5142.351

8 Limitations and Future Work

A subset of the available swarm intelligence algorithms has been considered for this study. Including other swarm intelligence algorithms could yield precise fault prediction or faster convergence. This study also focuses on weight optimization of ANN. There are other parameters of ANN that can be optimized using swarm intelligence algorithms. Even combinations of swarm intelligence algorithm can be attempted according to the parameters chosen for optimization. The projects considered for this study have been taken from NASA public dataset. Future work could include data from open source projects for a fair comparison.

9 Summary and Concluding remarks

In this paper we have investigated the influence of Swarm Intelligence Algorithms in improving the prediction accuracy of Artificial Neural Network based fault prediction models on NASA public datasets. Our analysis confirms that Swarm Intelligence Algorithms considerably improve the fault prediction capability of ANNs. We have considered four swarm intelligence algorithms for the comparison – particle swarm optimization, fire fly algorithm, artificial bee colony optimization and ant colony optimization. We also compared the various Swarm Intelligence Algorithms for the time taken to converge on the solution. We find that the Particle Swarm Optimization technique has better

prediction accuracy and efficiency in time taken to arrive at the solution compared to the other swarm intelligence algorithms considered. We also compared the algorithm parameters arrived at from literature survey with parameters taken from existing study on ANN-PSO and ANN-ABC models. We find that there is a marginal improvement with the parameters from existing study at huge cost in time taken for execution. This confirms that the parameters taken after rigorous literature review is appropriate.

References:

[1] Abreu, F Brito, Miguel Goulao, and Rita Esteves. "Toward the design quality evaluation of object-oriented software systems." Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA. 1995. 44-57.

[2] Aggarwal, KK, Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra. "Application of Artificial Neural Network for Predicting Maintainability using Object-Oriented Metrics." Transactions on Engineering, Computing and Technology (Citeseer) 15 (2006): 285-289.

[3] Aggarwal, RK, QY Xuan, RW Dunn, AT Johns, and A Bennett. "A novel fault classification technique for double-circuit lines based on a combined unsupervised/supervised neural network." Power Delivery, IEEE Transactions on (IEEE) 14, no. 4 (1999): 1250-1256.

[4] Aladjem, Mayer. "Recursive training of neural networks for classification." Neural Networks, IEEE Transactions on (IEEE) 11, no. 2 (2000): 496-503.

[5] Alba, Enrique, and J Francisco Chicano. "Training neural networks with GA hybrid algorithms." Genetic and Evolutionary Computation--GECCO 2004. 2004. 852-863.

[6] Altman, Edward I, Giancarlo Marco, and Franco Varetto. "Corporate distress diagnosis: Comparisons using linear discriminant analysis and neural networks (the Italian experience)." Journal of Banking \& Finance (Elsevier) 18, no. 3 (1994): 505-529.

[7] Ardil, Ebru, and Parvinder S Sandhu. "A soft computing approach for modeling of severity of faults in software systems." Int. J. Phys. Sci 5, no. 2 (2010): 74-85.

[8] Bartlett, Eric B, and Robert E Uhrig. "Nuclear power plant status diagnostics using artificial neural networks." Tech. rep., Tennessee Univ., Knoxville, TN (United States). Dept. of Nuclear Engineering, 1991.

[9] Bashir, ZA, and ME El-Hawary. "Applying wavelets to short-term load forecasting using PSO-based neural networks." Power Systems, IEEE Transactions on (IEEE) 24, no. 1 (2009): 20-27.

[10] Basili, Victor R, Lionel C. Briand, and Walcelio L Melo. "A validation of object-oriented design metrics as quality indicators." Software Engineering, IEEE Transactions on (IEEE) 22, no. 10 (1996): 751-761.

[11] Bello, Martin G. "Enhanced training algorithms, and integrated training/architecture selection for multilayer perceptron networks." Neural Networks, IEEE Transactions on (IEEE) 3, no. 6 (1992): 864-875.

[12] Chen, Wei-Neng, Jun Zhang, HS-H Chung, Rui-Zhang Huang, and Ou Liu. "Optimizing discounted cash flows in project scheduling—An ant colony optimization approach." Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on (IEEE) 40, no. 1 (2010): 64-77.

[13] Chidamber, Shyam R, and Chris F Kemerer. "A metrics suite for object oriented design." Software Engineering, IEEE Transactions on (IEEE) 20, no. 6 (1994): 476-493.

[14] Di Natale, Corrado, et al. "Electronic nose and electronic tongue integration for improved classification of clinical and food samples." Sensors and Actuators B: Chemical (Elsevier) 64, no. 1 (2000): 15-21.

[15] Dorigo, Marco, Vittorio Maniezzo, and Alberto Colorni. "Ant system: optimization by a colony of cooperating agents." Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on (IEEE) 26, no. 1 (1996): 29-41.

[16] Farshidpour, Solmaz, and Farshid Keynia. "Using Artificial Bee Colony Algorithm for MLP Training on Software Defect Prediction." 2012.

[17] Feng, W, W Zhu, and Y Zhou. "The Application of Genetic Algorithm and Neural Network in Construction Cost Estimate." Proc. Third International Symposium on Electronic Commerce and Security Workshops (ISECS'10). 2010. 29-31.

[18] Fenton, Norman E, and Shari Lawrence Pfleeger. Software metrics: a rigorous and practical approach. PWS Publishing Co., 1998.

[19] Ganatra, Amit, YP Kosta, Gaurang Panchal, and Chintan Gajjar. "Initial Classification Through Back Propagation In a Neural Network Following Optimization Through GA to Evaluate the Fitness of an Algorithm." 2011.

[20] Guyon, I. "Applications of neural networks to character recognition." International Journal of Pattern Recognition and Artificial Intelligence (World Scientific) 5, no. 01n02 (1991): 353-382.

[21] Hagan, Martin T, and Mohammad B Menhaj. "Training feedforward networks with the Marquardt algorithm." Neural Networks, IEEE Transactions on (IEEE) 5, no. 6 (1994): 989-993.

[22] Hoskins, JC, KM Kaliyur, and DM Himmelblau. "Incipient fault detection and diagnosis using artificial neural networks." Neural Networks, 1990., 1990 IJCNN International Joint Conference on. 1990. 81-86.

[23] Ilonen, Jarmo, Joni-Kristian Kamarainen, and Jouni Lampinen. "Differential evolution training algorithm for feed-forward neural networks." Neural Processing Letters (Springer) 17, no. 1 (2003): 93-105.

[24] Jacobs, Robert A. "Increased rates of convergence through learning rate adaptation." Neural networks (Elsevier) 1, no. 4 (1988): 295-307.

[25] Jiang, Wei, and G Seong Kong. "Block-based neural networks for personalized ECG signal classification." Neural Networks, IEEE Transactions on (IEEE) 18, no. 6 (2007): 1750-1761.

[26] Jin, Feihu, and Guang Shu. "Back Propagation Neural Network Based on Artificial Bee Colony Algorithm." 2012.

[27] Kanmani, S, V Rhymend Uthariaraj, V Sankaranarayanan, and P Thambidurai. "Object-oriented software fault prediction using neural networks." Information and Software Technology (Elsevier) 49, no. 5 (2007): 483-492.

[28] Kartalopoulos, Stamatios V, and Stamatios V Kartakapoulos. Understanding neural networks and fuzzy logic: basic concepts and applications. Wiley-IEEE Press, 1997.

[29] Kennedy, James. "Particle swarm optimization." In Encyclopedia of Machine Learning, 760-766. Springer, 2010.

[30] Khoshgoftaar, Taghi M, Edward B Allen, John P Hudepohl, and Stephen J Aud. "Application of neural networks to software quality modeling of a very large telecommunications system." Neural Networks, IEEE Transactions on (IEEE) 8, no. 4 (1997): 902-909.

[31] Knerr, Stefan, Leon Personnaz, and Gerard Dreyfus. "Handwritten digit recognition by neural networks with single-layer training." Neural Networks, IEEE Transactions on (IEEE) 3, no. 6 (1992): 962-968.

[32] Kollias, Stefanos, and Dimitris Anastassiou. "An adaptive least squares algorithm for the efficient training of artificial neural networks." Circuits and Systems, IEEE Transactions on (IEEE) 36, no. 8 (1989): 1092-1101.

[33] Kumar, Amioy, Madasu Hanmandlu, Harsh Sanghvi, and HM Gupta. "Decision level biometric fusion using Ant Colony Optimization." Image Processing (ICIP), 2010 17th IEEE International Conference on. 2010. 3105-3108.

[34] Lacher, R Christopher, Pamela K Coats, Shanker C Sharma, and L Franklin Fant. "A neural network for classifying the financial health of a firm." European Journal of Operational Research (Elsevier) 85, no. 1 (1995): 53-65.

[35] Lampinen, Jouko, Seppo Smolander, and Markku Korhonen. "Wood surface inspection system based on generics visual features." International Conference on artificial neural networks ICANN. 1995. 9-13.

[36] Levin, Esther, and Michael Fleisher. "Accelerated learning in layered neural networks." Complex systems 2 (1988): 625-640.

[37] Li, Kewen, Jisong Kou, and Lina Gong. "Predicting software quality by optimized BP network based on PSO." Journal of Computers 6, no. 1 (2011): 122-129.

[38] Li, Wei, and Sallie Henry. "Object-oriented metrics that predict maintainability." Journal of systems and software (Elsevier) 23, no. 2 (1993): 111-122.

[39] Lorenz, Mark, and Jeff Kidd. Object-oriented software metrics: a practical guide. Prentice-Hall, Inc., 1994.

[40] Mastorocostas, PA. "Resilient back propagation learning algorithm for recurrent fuzzy neural networks." Electronics Letters (IET) 40, no. 1 (2004): 57-58.

[41] Mazwin Mohmad Hassim, Yana, and Rozaida Ghazali. "Training a Functional Link Neural Network Using an Artificial Bee Colony for Solving a Classification Problems." 2012.

[42] McMullen, Patrick R. "An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives." Artificial Intelligence in Engineering (Elsevier) 15, no. 3 (2001): 309-317.

[43] Miniani, AA, and Ronald D Williams. "Acceleration of back-propagation through learning rate and momentum adaptation."

Proceedings of International Joint Conference on Neural Networks. 1990. 676-679.

[44] Moller, Martin Fodslette. "A scaled conjugate gradient algorithm for fast supervised learning." Neural networks (Elsevier) 6, no. 4 (1993): 525-533.

[45] Monedero, Inigo, Carlos Leon, Jorge Ropero, Antonio Garcia, Jose Manuel Elena, and Juan C Montano. "Classification of electrical disturbances in real time using neural networks." Power Delivery, IEEE Transactions on (IEEE) 22, no. 3 (2007): 1288-1296.

[46] Nandy, Sudarshan, Partha Pratim Sarkar, Ajith Abraham, Manoj Karmakar, Achintya Das, and Diptarup Paul. "Agent based adaptive firefly back-propagation neural network training method for dynamic systems." Hybrid Intelligent Systems (HIS), 2012 12th International Conference on. 2012. 449-454.

[47] Narendra, Kumpati S, and Kannan Parthasarathy. "Identification and control of dynamical systems using neural networks." Neural Networks, IEEE Transactions on (IEEE) 1, no. 1 (1990): 4-27.

[48] Oien, GE, NA Bertelsen, T Eftestol, and JH Husoy. "ECG rhythm classification using artificial neural networks." Digital Signal Processing Workshop Proceedings, 1996., IEEE. 1996. 514-517.

[49] Park, Dong C, Mohamed A El-Sharkawi, Robert J Marks, and others. "An adaptively trained neural network." Neural Networks, IEEE Transactions on (IEEE) 2, no. 3 (1991): 334-345.

[50] Petsche, Thomas, Angelo Marcantonio, Christian Darken, Stephen Jose Hanson, Gary M Kuhn, and Iwan Santoso. "A neural network autoassociator for induction motor failure prediction." Advances in neural information processing systems (MORGAN KAUFMANN PUBLISHERS), 1996: 924-930.

[51] Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." Cognitive modeling 1 (2002): 213.

[52] Rutkowski, Leszek. "Adaptive probabilistic neural networks for pattern classification in time-varying environment." Neural Networks, IEEE Transactions on (IEEE) 15, no. 4 (2004): 811-827.

[53] Samad, Tareq. "Back-propagation improvements based on heuristic arguments." Proceedings of International Joint Conference on Neural Networks. 1990. 565-568.

[54] Sharma, Kanu, Navpreet Kaur, Sunil Khullar, and Harish Kundra. "Defect Prediction based on Quantitative and Qualitative Factors using PSO optimized Neural Network." 2012.

[55] Slowik, Adam, and Michal Bialko. "Training of artificial neural networks using differential evolution algorithm." Human System Interactions, 2008 Conference on. 2008. 60-65.

[56] Suratgar, Amir Abolfazl, Mohammad Bagher Tavakoli, and Abbas Hoseinabadi. "Modified Levenberg--Marquardt method for neural networks training." World Acad Sci Eng Technol (Citeseer) 6 (2005): 46-48.

[57] Tam, Kar Yan, and Melody Y Kiang. "Managerial applications of neural networks: the case of bank failure predictions." Management science (INFORMS) 38, no. 7 (1992): 926-947.

[58] Tsai, Hsien-Leing, and Shie-Jue Lee. "Entropy-based generation of supervised neural networks for classification of structured patterns." Neural Networks, IEEE Transactions on (IEEE) 15, no. 2 (2004): 283-297.

[59] Wu, G, and P Huang. "A Vectorization-Optimization-Method Based Type-2 Fuzzy Neural Network for Noisy Data Classification." (IEEE) 2013.

[60] Xenos, Michalis, D Stavrinoudis, K Zikouli, and D Christodoulakis. "Object-oriented metrics-a survey." Proceedings of the FESMA. 2000. 1-10.

[61] Yang, Xin-She. "Firefly algorithms for multimodal optimization." In Stochastic algorithms: foundations and applications, 169-178. Springer, 2009.

[62] Yang, Zheng Rong, Mark Zwolinski, Chris D Chalk, and Alan Christopher Williams. "Applying a robust heteroscedastic probabilistic neural network to analog fault detection and classification." Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on (IEEE) 19, no. 1 (2000): 142-151.

[63] Yogi, Sandhya, KR Subhashini, and JK Satapathy. "A PSO based Functional Link Artificial Neural Network training algorithm for equalization of digital communication channels." Industrial and Information Systems (ICIIS), 2010 International Conference on. 2010. 107-112.

[64] Zakaria, Zulhadi, NAM Isa, and SA Suandi. "A study on neural network training algorithm for multiface detection in static images." World Acad Sci Eng Technol 38 (2010): 170-173.

[65] Zhang, Yudong, and Lenan Wu. “Crop Classification by forward neural network with adaptive chaotic particle swarm optimization.” Sensors (Molecular Diversity Preservation International) 11, no. 5 (2011): 4721-4743.

[66] Zhang, Yudong, Lenan Wu, and Shuihua Wang. “Magnetic resonance brain image classification by an improved artificial bee colony algorithm.” Progress in Electromagnetics Research (EMW Publishing) 116 (2011): 65-79.

[67] Lippmann, Richard P. "An introduction to computing with neural nets." ASSP Magazine, IEEE 4.2 (1987): 4-22.

[68] Karaboga, Dervis. “An idea based on honey bee swarm for numerical optimization” vol. 200. Technical report, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[69] Argha Roy, Diptam Dutta and Kaustav Choudhury. “Training artificial neural network using particle swarm optimization algorithm.” International Journal of Advance Research in Computer Science and Software Engineering 3, no. 3, (2013): 430-434.

[70] Kawam, Ahmad AL, and Nashat Mansour. "Metaheuristic Optimization Algorithms for Training Artificial Neural Networks." International Journal od Computer and Information Tecnology–ISSN: 2279-0764.

[71] Yu, Jianbo, Lifeng Xi, and Shijin Wang. "An improved particle swarm optimization for evolving feedforward artificial neural networks." Neural Processing Letters 26.3 (2007): 217-231.