*I.J. Intelligent Systems and Applications,* **2013, 01, 69-80**
Published Online December 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2013.01.07

# Optimizing Artificial Neural Networks using Cat Swarm Optimization Algorithm

**John Paul T. Yusiong**
Division of Natural Sciences and Mathematics, University of the Philippines Visayas Tacloban College, Magsaysay Boulevard, Tacloban City, Leyte, Philippines
jpyusiong@gmail.com

***Abstract***— An Artificial Neural Network (ANN) is an abstract representation of the biological nervous system which has the ability to solve many complex problems. The interesting attributes it exhibits makes an ANN capable of “learning”. ANN learning is achieved by training the neural network using a training algorithm. Aside from choosing a training algorithm to train ANNs, the ANN structure can also be optimized by applying certain pruning techniques to reduce network complexity. The Cat Swarm Optimization (CSO) algorithm, a swarm intelligence-based optimization algorithm mimics the behavior of cats, is used as the training algorithm and the Optimal Brain Damage (OBD) method as the pruning algorithm. This study suggests an approach to ANN training through the simultaneous optimization of the connection weights and ANN structure. Experiments performed on benchmark datasets taken from the UCI machine learning repository show that the proposed CSONN-OBD is an effective tool for training neural networks.

***Index Terms***— Artificial Neural Network, Neural Network Training, Neural Network Pruning, Optimal Brain Damage, Swarm Intelligence, Cat Swarm Optimization

## I. Introduction

An artificial neural network (ANN), also known as neural network (NN), is an abstract representation of the biological nervous system. It is composed of a collection of neurons that communicates with each other through the axons. An artificial neural network is an adaptive system that has interesting attributes like the ability to adapt, learn and generalize. An ANN is also highly accurate in classification and prediction of output because of its massively parallel processing, fault tolerance, self-organization and adaptive capability which enables it to solve many complex problems. Its ability to solve different problems is achieved by changing its network structure during the learning (training) process [1-2].

But, it was also pointed out that the determination of various ANN parameters like the number of hidden layers, number of neurons in the hidden layer, connection weights initialization etc. is not a straightforward process and finding the optimal configuration of ANNs is a very time consuming process [3]. Thus, designing an optimal ANN structure and choosing an effective ANN training algorithm for a given problem is an interesting research area.

Moreover, since the determination of various ANN parameters is not a straightforward process, various researches have been conducted with the purpose of finding the optimal configuration of ANNs. As a result, several algorithms have been proposed as training algorithms for ANNs and these include Genetic Algorithms (GA) [2], Ant Colony Optimization (ACO) [4], Artificial Bee Colony (ABC) [5] and Particle Swarm Optimization (PSO) [6]. These algorithms vary on how they can effectively optimize the artificial neural network with respect to the problem being solved.

Cat Swarm Optimization (CSO) is a more recent swarm intelligence-based optimization algorithm developed in 2006. It was developed to solve various problems by mimicking the behavior of cats. CSO has been proven to have a better performance in finding the global best solutions than other existing optimization algorithms [7-9].

In addition to choosing a training algorithm to train ANNs to carry out a certain task, the ANN structure can also be optimized by applying certain pruning techniques to reduce network complexity without drastically affecting its classification and prediction capabilities. A pruning technique presented in this paper is the Optimal Brain Damage (OBD) pruning algorithm [10]. This pruning technique was found to be computationally simpler and can produce relatively good ANNs.

Consequently, ANN researches can be classified into two categories: (1) training ANNs using a training algorithm and a non-OBD pruning technique to further improve the ANNs [11-13]; and (2) training ANNs using a training algorithm and an OBD pruning technique to further improve the ANNs [14-16].

In this paper, the CSO algorithm is proposed to be used as the training algorithm to train the ANNs with

OBD as its pruning method. The objective is to develop a CSO-based ANN optimizer that trains artificial neural networks to learn the input-output relationships of a given problem and then use the OBD pruning method to generate an optimal network structure. That is, the CSO-based ANN optimizer will generate an optimal set of connection weights and structure for a given problem.

The rest of the paper is organized as follows: Section II discusses the artificial neural networks and the NN training procedures while Section III describes the NN pruning procedures and the Optimal Brain Damage pruning method. In Section IV, the Cat Swarm Optimization algorithm is presented while Section V discusses the CSONN-OBD, which is the proposed algorithm. Experimental results and observations are presented in Section VI and conclusions are presented in Section VII.

## II. Artificial Neural Networks and NN Training

### 2.1 Artificial Neural Networks

Several researchers [3,17-18] conducted a thorough review of the various researches involving artificial neural networks and their work present an excellent starting point to get acquainted with researches on ANNs.

An Artificial Neural Network (ANN) is made up of simple processing units, called artificial neurons or nodes as shown in Figure 1, which mimics the biological nervous systems. The artificial neuron performs its task in two phases: computing for the weighted sum and using a certain kind of non-linear function. This approach allows the ANN to process the input data which represents the problem to be solved [2,5].

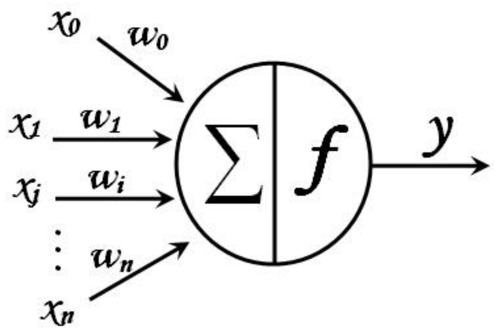


Fig. 1: The Artificial Neuron

In Figure 1, $x_j$ is the $j^{th}$ input to the neuron, $w_{ij}$ is the connection weight between the neuron and the input $x_j$, $x_0w_0$ is the threshold or bias neuron where $x_0=1$, the symbol, $\sum$, is the weighted sum of the input data usually written as *net*, *y* is the output of the neuron and *f* is the activation function which is usually a non-linear function [19].

Equation (1) describes the weighted sum of the input data, where $b_i=x_0w_0$ is the bias neuron, (2) describes the logarithmic sigmoid function (logistic Function), which is the activation function used in this paper and (3) describes the output of the neuron.

$$net = \sum_{j=1}^{n} w_{ij}\, x_j + b_i \tag{1}$$

$$f_i(net) = \frac{1}{1+e^{-net}} \tag{2}$$

$$y_i = f_i(net) \tag{3}$$

The most common structure of an artificial neural network used in many researches is shown in Figure 2. It is often referred to as Multilayer Perceptron (MLP), which is a feed-forward neural network. It is made up of several layers of processing units (neurons) and every neuron in each layer is connected to all neurons in the succeeding layers. All artificial neural networks have an input layer, $x_j$ and an output layer, $y_i$, but the number of hidden layers, $z_k$, may differ. Both $y_j$ and $z_k$ use (3) to compute for the output of the neuron.

However, it is has been shown that a three layer feed-forward neural network can approximate any non-linear function with arbitrary accuracy. Nonetheless, finding an optimal ANN structure and an optimal set of connection weights is still a difficult problem [2,5,12].

MLPs can receive inputs, process the data, and provide outputs. In MLPs the input data are processed within the individual neurons of the input layer and then the output values of these neurons are forwarded to the neurons in the hidden layer. The same process happens between the neurons of the hidden layer and the output layer. In every layer, each neuron receives inputs from the neurons in the previous layer and each of the inputs is multiplied by a different weight value. The weighted inputs are added as described in (1) and forwarded to an activation function which limits the output to a fixed range of values as shown in (2). The output of each neuron as described in (3) is then forwarded to all of the neurons in the next layer [19].

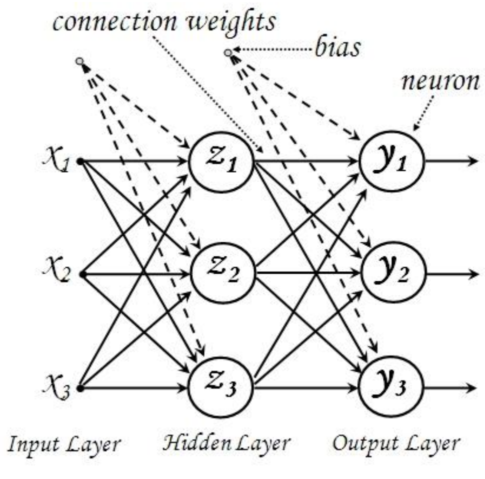


Fig. 2: The Multilayer Perceptron

The diagram in Figure 2 shows an artificial neural network, which is a three layer, fully-connected, feed-forward neural network. Fully-connected indicates that the output from each neuron in one layer is distributed to all of the neurons in the next layer while feed-forward signifies that the values only move from the input layer to the hidden layer and from the hidden layer to the output layer, that is, no values are fed back to the previous layers.

As mentioned in [4], artificial neural networks are widely used because they are effective in approximating real-valued, discrete-valued and vector-valued target functions. ANNs also have exceptional characteristics in machine learning and these capabilities make artificial neural networks a powerful tool for research problems that entail recognition, classification, and forecasting.

## 2.2 Neural Network Training

The goal of the neural network training procedure is to find the optimal set of connection weights that will cause the output from the artificial neural network to match the actual target values. To be able to find the set of connections weights, an algorithm is used. This algorithm should be able to adjust the connections weights of the ANN in order to obtain the desired output from the network given a specific set of inputs. The process of adjusting the weights is called learning or training. The primary objective of neural network training is to find a set of connection weights that minimizes the objective function [19]. In this paper the objective function is the mean-squared error (MSE) function as described in (4).

$$\begin{aligned} MSE &= \frac{1}{NM}\left(\sum_{i=1}^{N}\left(\sum_{k=1}^{M}\frac{1}{2}{e_k}^2\right)\right) \\ &= \frac{1}{NM}\left(\sum_{k=1}^{N}\left(\sum_{k=1}^{M}\frac{1}{2(t_k-y_k)}^2\right)\right) \end{aligned} \tag{4}$$

In (4), *N* is the total number of training examples in the training data, *M* is the number of output neurons in the output layer, $e_k$ represents the network error of a training example, $t_k$ is the desired output and $y_k$ is the actual output of the network.

Neural network training is an important task in supervised learning. In supervised learning the ANN is provided with the correct output for each of the training examples in the training data. So the aim is for the ANN to produce an output that is near the correct output. Figure 3 illustrates the supervised learning paradigm. That is why during training, a training data, made up of inputs (training examples) and their expected outputs, is presented to the ANN which is used to adjust the set of connection weights. The training data is fundamental for the ANNs as it provides the information that is essential in discovering the optimal set of connection weights. So, in this approach, connection weights are modified in response to the input/output patterns [20].

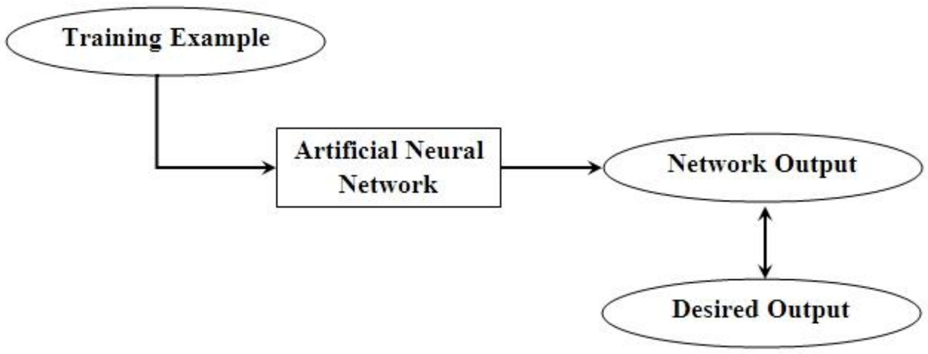


Fig. 3: The Supervised Learning Paradigm

During training, an input and the corresponding desired output is presented to the artificial neural network and the network error is computed. The network error is the difference between the desired output and the actual output of the network. The training algorithm adjusts the connection weights so as to minimize the mean-squared error function as described in (4).

In his paper, Yao [18] explained three general approaches to neural network training and these are the following:

1. finding a near-optimal set of connection weights for an NN with a fixed architecture for the task at hand;
2. finding a near-optimal NN architecture for the task at hand; and
3. finding both a near-optimal set of connection weights and a neural network architecture for the task at hand.

The first approach to neural network training is referred to as weight training in ANN and is typically formulated as a minimization of an error function, such as the mean-squared error. So, in this approach, a fixed

ANN structure is provided and the objective is finding a set of connection weights that minimizes the network's error on the training data. In effect, finding the set of connection weights of a network can be seen as an optimization process [18].

The second approach involves finding a near-optimal NN architecture. This approach emphasizes that the neural network's structure greatly influences its performance and its information processing capabilities. In this approach, constructive and destructive algorithms are used for automatic design of architectures and it takes into consideration the fact that if the network is too small it will not be able to form a good model of the problem while network that is too big may have very poor generalization ability [18,21]. Constructive algorithms are algorithms that start with simple architecture and gradually add new neurons during learning process while destructive algorithms start with an initially redundant network and simplify it during the learning process, and this process is often referred to as "pruning".

The third approach is a combination of the first two approaches which attempts to simultaneously evolve the NN structure and connection weights. It is very important to find the appropriate NN structure and the appropriate connection weights to ensure the best performance of the ANN [18]. In this paper, the aim is to obtain an optimal set of connection weights and subsequently apply a well-known destructive algorithm, the Optimal Brain Damage [10], to obtain an optimal NN structure.

## III. Neural Network Pruning and the Optimal Brain Damage Method

### 3.1 Neural Network Pruning

Designing optimal neural network architecture for a particular classification problem is an important issue. This is due to that fact that a network that is too big is more likely to overfit the training examples because of its excess information processing capability while a network that is too small may underfit the training data because of its limited information processing capability. Both of these problems lead to poor generalization capability on unseen examples, an undesirable aspect of ANN. Thus, it is necessary to design ANNs automatically in order to solve different problems efficiently [13,22].

However, the optimal network size is generally unknown and tedious experimentation is necessary to find it. Another approach to improving the generalization capability of ANNs is to train a network which is considered to be larger than necessary and prune the excess components [22].

Several approaches for optimizing neural network architectures have been proposed which include various pruning algorithms such as the Optimal Brain Damage algorithm [10]. A pruning algorithm starts with a relatively large ANN architecture, that is, an ANN with a large number of hidden neurons. In the pruning process, unnecessary hidden layers, neurons, and connection weights from a relatively large ANN are discarded. Generally, one connection weight or neuron is removed in each step of the pruning process [13].

The pruning process is defined as a network trimming procedure given a relatively large architecture which is basically a model reduction method with the goal of finding the optimal neural network architecture [11,23]. Pruning a network is based on estimating the sensitivity of the total error to the exclusion of each connection weight in the network, which is in essence estimating the importance of each connection weight or neuron in the network. In each step of the process, the connection weights or neurons which are insensitive to error changes are gradually removed which will eventually result to a network of smaller size that will likely have a better generalization capability [11].

Basically, finding for the optimal neural network architecture is a 4-step process and many pruning algorithms that have been proposed differ on how the 2nd and the 4th steps are implemented, and on the criterion being used to identify the significance of a connection weight or neuron. The 4-step process is presented as follows [23]:

1.Train an artificial neural network;

2. Identify the least significant connection weight or neuron;

3. Prune the least significant connection weight or neuron;

4. Re-train the artificial neural network and repeat steps 2 and 3.

In addition, Yang et al. [24] presented a similar framework for network pruning and is described as follows:

1. Set a large enough architecture for the NNs and train with any learning method, until the stopping criterion is met;

2. Compute the saliency of each element and eliminate the least important ones;

3. Retrain the pruned network. If the change of output between the original and pruned network is small enough, then go to step 2; otherwise stop and output the network architecture.

### 3.2 Neural Network Pruning using Optimal Brain Damage

As stated in [11], an important step for the discovery of knowledge from data is to find a method to optimize the structure of neural networks. A typical approach in the NN structure design is to simply utilize a fully-connected multilayer feed-forward neural network to solve problems but this approach limits the performance of the resulting neural networks. Thus, several neural network pruning techniques as presented in [1,5,11,13-16,22,24-25] have been proposed and developed to optimize the neural network structures for a given problem.

Among the different NN pruning techniques, it has been shown that very good results are obtained using saliency-based methods [14-15]. Saliency-based methods are also referred to as weight pruning algorithms, sensitivity-based approaches and destructive algorithms. This method attempts to balance between network complexity and training error.

The saliency-based methods analyze the sensitivity of the objective function to deletion of individual connection weights, that is, these methods evaluate the influence of each connection weight on the NN generalization error. In effect, these methods optimize the neural network structure by finding the contribution (saliency) of each connection weight or neuron in the network and pruning the connection weight or neuron which has the least effect on the objective function. This is because the connection weights with the smallest saliency are considered to be insignificant and therefore these connection weights can be deleted [11].

One of the well-known saliency-based methods whose effectiveness has been proven in different applications [14-16] is the Optimal Brain Damage (OBD) which was introduced by Le Cun et al. [10]. OBD is a method to establish the effect of each connection weight on the objective function.

Le Cun et al. [10] proposed the Optimal Brain Damage (OBD) method to approximate the measure of "saliency" of a connection weight by estimating the second derivative of the network output error with respect to that connection weight. The network complexity is also reduced by a large factor by constraining certain connection weights to be equal. The saliency of a connection weight is described as the change in the training error when the connection weight is removed and the remaining connection weights are retrained. In effect, the primary goal of OBD method is to reduce the complexity of the network by selectively deleting the connection weights with the goal of improving its generalization. The OBD method accepts a relatively large network, delete at least half of the connection weights and produce a network that performs as good as the network that is un-pruned [10].

In OBD, pruning is carried out iteratively on a trained network to a reasonable level, compute "saliencies", rank the connection weights according to saliency and then delete the connection weights with the smallest saliency, and resume training until a termination criterion is satisfied. OBD assumes that the error function is quadratic and that the Hessian is diagonal, that is, the saliencies are approximated by the second derivative of the objective function with respect to the connection weights [11].

The algorithm of the OBD method as described in [10,14-16] is presented as follows.

1.Choose an initial NN structure that is reasonably large.

2. Train the neural network using a training algorithm until a reasonable solution is obtained.

3. Compute the diagonal elements $h_{ii}$ of the Hessian matrix $H$.

4. Compute the saliency parameters $S_i$ for each of the connection weights as shown in (9).

5. Remove connection weights with the smallest saliency.

6. Re-train the NN with the new network structure until the termination criterion is satisfied.

As can be seen in the OBD algorithm, the initial steps are to choose a reasonably large neural network structure and train the network using a training algorithm to minimize the objective function. The typical objective function to minimize is the mean-squared error function as shown in (4). The NN training procedure attempts to minimize this function which is defined as the mean-squared error between the NN output and the real value (training examples) [10,14-16].

The next step is to remove connection weights which are considered insignificant because they have the smallest saliency. However, since the saliency coefficient cannot be determined directly from the objective function, it is necessary to approximate this function with Taylor series as presented in (5) [10,14-16].

$$\partial E = \sum_i g_i \partial w_i + \frac{1}{2}\sum_i H_{ii} \partial w_i^2 + \frac{1}{2}\sum_{i\neq j} H_{ij} \partial w_i \partial w_j + \cdots \quad (5)$$

Where:

$$g = \frac{\partial E}{\partial w} : Gradient\ vector \quad (6)$$

$$H = \frac{\partial^2 E}{\partial w} : Hessian\ matrix \quad (7)$$

The OBD method takes into account three approximations in order to determine the saliency of each connection weight [10,16]:

1. The neural network converged to a minimum of the error function, so $g = 0$ and the first term in (5) can be eliminated.

2. The objective function is quadratic so all terms of order greater than 2 can be neglected in (5).

3.The Hessian matrix $H$ is diagonal, so all terms $H_{ij} = 0$ (with $i \neq j$).

With the given approximations, (5) is reduced to (8):

$$\partial E = \frac{1}{2}\sum_i H_{ii}\partial w_i^2 \quad (8)$$

So, (9) is the equation that defines the saliency of each connection weight in the neural network, where $S_i$ is the saliency of $w_i$ and $H_{ii}$ is the $i^{th}$ diagonal entry of the Hessian matrix.

$$S_i \cong \partial E = \frac{1}{2} w_i^2 H_{ii} \quad (9)$$

The next step is to compute for the saliency of each connection weight, remove the connection weight with the smallest saliency while the final step is to re-train the network after the removal of this connection weight, and recalculate the saliency coefficients of each connection weights in the network.

Thus, the Optimal Brain Damage (OBD) method removes connection weights based on the second order derivatives of the objective function. The matrix of the second order derivatives is called Hessian. The full Hessian matrix can be computed using (10), and for a network with a single connection weight or for the diagonal entries of the Hessian matrix, (11) is used, where $\left(\frac{\partial z_i}{\partial w_i}\right)$ is the first derivative of the objective function with respect to weight $w_i$, $N$ is the number of examples, $M$ is the number of outputs, and $H_0$ is the identity matrix multiplied by a random number in the range of $10^{-8} \le \alpha \le 10^{-4}$ as shown in (12). $H_0$ is important to avoid any singularities especially in OBD [26,27].

$$H = \frac{\partial^2 E}{\partial w_i \partial w_j} = H_0 + \frac{1}{NM}\sum_{k=0}^{n}\left(-\frac{\partial z_i}{\partial w_i}\right)\left(-\frac{\partial z_j}{\partial w_j}\right) \quad (10)$$

$$H = \frac{\partial^2 \boldsymbol{E}}{\partial w_i \partial w_j} = H_0 + \frac{\mathbf{1}}{\boldsymbol{NM}}\sum_{\boldsymbol{k}=\mathbf{0}}^{\boldsymbol{n}}\left(-\frac{\partial z_i}{\partial w_i}\right)^{\mathbf{2}} \quad (11)$$

$$H_0 = \alpha I \quad (12)$$

## IV. Cat Swarm Optimization Algorithm: A Swarm Intelligence-based Algorithm

Swarm Intelligence (SI) is a novel artificial intelligence approach inspired by the swarming behaviors of groups of organisms such as ants, termites, bees, birds, fishes in foraging and sharing the information with each other. SI focuses on the collective intelligence of a decentralized system consisting of a group of organisms interacting with each other and their environment. So, by means of their collective intelligence swarms are able to effectively use their environment and resources. SI is also a mechanism that enables individuals to overcome their cognitive limitations and solve problems which are difficult for individuals to resolve alone. Swarm intelligence algorithms are essentially stochastic search and optimization techniques and were developed by simulating the intelligent behavior of these organisms. These algorithms are known to be efficient, adaptive, robust, and produce near optimal solutions and utilize implicit parallelism approaches [28].

One of the more recent optimization algorithm based on swarm intelligence is the Cat Swarm Optimization (CSO) algorithm. The CSO algorithm was developed based on the common behavior of cats. It has been found that cats spend most of their time resting and observing their environment rather that running after things as this leads to excessive use of energy resources. To reflect these two important behavioral characteristics of the cats, the algorithm is divided into two sub-modes and CSO refers to these behavioral characteristics as “seeking mode” and “tracing mode”, which represent two different procedures in the algorithm. Tracing mode models the behavior of the cats when running after a target while the seeking mode models the behavior of the cats when resting and observing their environment [7-8].

Furthermore, previous researches have shown that the CSO algorithm has a better performance in function minimization problems compared to the other similar optimization algorithms like Particle Swarm Optimization (PSO) and weighted-PSO [7-9].

### 4.1 Seeking Mode:Resting and Observing

The seeking mode of the CSO algorithm models the behavior of the cats during a period of resting but staying alert-observing its environment for its next move. The Seeking mode procedure has four essential factors and these are: Seeking Memory Pool (*SMP*); Seeking Range of the selected Dimension (*SRD*); Counts of Dimension to Change (*CDC*); and Self Position Consideration (*SPC*) as described by Chu et al. [7-8]. The seeking mode of the CSO algorithm can be described as follows:

**Step 1**: Make *j* copies of the present position of $cat_k$, where $j = SMP$. If the value of *SPC* is true, let $j = (SMP - 1)$, then retain the present position as one of the candidates.

**Step 2**: For each copy, according to *CDC*, randomly add or subtract *SRD* percentage to the present values and replace the old ones.

**Step 3:** Calculate the fitness values (*FS*) of all candidate points.

**Step 4:** If all *FS* are not exactly equal, calculate the selecting probability of each candidate point by (13), otherwise set all the selecting probability of each candidate point to 1.

**Step 5**: Randomly pick the point to move to from the candidate points, and replace the position of $cat_k$.

$$P_i = \frac{|FS_i - FS_b|}{FS_{max} - FS_{min}}, where\ 0 < i < j \quad (13)$$

If the goal of the fitness function is to find the minimum solution, $FS_b = FS_{max}$, otherwise $FS_b = FS_{min}$.

### 4.2 Tracing Mode: Running After a Target

The tracing mode of the CSO algorithm models the behavior of the cats when running after a target. Once a cat goes into tracing mode, it moves according to its own velocities for each dimension. The action of tracing mode according to Chu et al. [7-8] can be described as follows:

**Step 1**: Update the velocities for every dimension $(v_{k,d})$ according to (14).

**Step 2**: Check if the velocities are in the range of maximum velocity. In case the new velocity is over-range, it is set equal to the limit.

**Step 3**: Update the position of $cat_k$ according to (15).

$$v_{k,d} = v_{k,d} + r_1 * c_1 * \left(x_{best,d} - x_{k,d}\right),\ d = 1,2,\dots,M \quad (14)$$

where $x_{best,d}$ is the position of the cat, who has the best fitness value; $x_{k,d}$ is the position of $cat_k$; $c_1$ is a constant and $r_1$ is a random value in the range of [0,1].

$$x_{k,d} = x_{k,d} + v_{k,d} \quad (15)$$

### 4.3 CSO Movement = Seeking Mode + Tracing Mode

When applying the CSO algorithm to solve optimization problems, the initial step is to make a decision on the number of individuals or cats to use. Each cat in the population has the following attributes:

a) a position made up of *M* dimensions;

b) velocities for each dimension in the position;

c) a fitness value of the cat according to the fitness function; and

d) a flag to indicate whether the cat is in seeking mode or tracing mode.

The CSO algorithm keeps the best solution after each cycle and when the termination condition is satisfied, the final solution is the best position of one of the cats in the population. CSO has two sub-modes, namely seeking mode and tracing mode and the mixture ratio *MR* dictates the joining of seeking mode with tracing mode. To ensure that the cats spend most of their time resting and observing their environment, the *MR* is initialized with a small value.

The CSO algorithm can be described in 6 steps as presented in [7-8].

**Step 1**: Create *N* cats in the process.

**Step 2**: Randomly sprinkle the cats into the *M*-dimensional solution space and randomly give values, which are in-range of the maximum velocity, to the velocities of every cat. Then haphazardly pick number of cats and set them into tracing mode according to *MR*, and the others set into seeking mode.

**Step 3**: Evaluate the fitness value of each cat by applying the positions of cats into the fitness function, which represents the criteria of our goal, and keep the best cat into memory. Note that we only need to remember the position of the best cat $(x_{best})$ because it represents the best solution so far.

**Step 4**: Move the cats according to their *flags*, if $cat_k$ is in seeking mode, apply the cat to the seeking mode process, otherwise apply it to the tracing mode process.

**Step 5**: Re-pick number of cats and set them into tracing mode according to *MR*, then set the other cats into seeking mode.

**Step 6**: Check the termination condition, if satisfied, terminate the program, and otherwise repeat **Step 3** to **Step 5**.

## V. CSONN-OBD: A CSO-based ANN Optimizer with OBD Pruning Method

The proposed algorithm called CSONN-OBD is a swarm intelligence-based ANN optimizer to neural network training with the Cat Swarm Optimization algorithm [7-8] as the training algorithm and the Optimal Brain Damage technique [10] as the pruning method used to reduce network complexity. Also, the proposed algorithm employs the supervised learning approach in training neural networks. The training involves fully connected feed-forward neural networks where each neuron uses the logistic function as the activation function as described in (2). The CSONN-OBD will simultaneously determine the optimal set of connection weights and its corresponding network structure.

The CSO algorithm represents a cat as a vector and a population contains several cats. In this study, a cat represents a one-hidden layer fully-connected feed-forward neural network and the cat population consists of several one-hidden layer fully-connected feed-forward neural networks. Each cat is evaluated using the mean-squared error function as described in (4). The general framework of the CSONN-OBD is as follows:

**Step 1:** Build a one-hidden layer fully-connected feed-forward neural network.

**Step 2:** Train NNs using CSO and evaluate each NN (cat) using MSE until stopping criterion is satisfied

**Step 3:** Apply OBD on the best cat

**Step 4:** Re-train pruned NN

**Step 5:** If stopping criterion is not satisfied then go to Step 3, else go to Step 6.

**Step 6:** Output the best pruned NN

In CSO, a cat represents an artificial neural network as illustrated in Figure 4. The ANN is represented as a vector with dimension $D$ containing the connection weights as depicted in Figure 5. As presented in [29], the dimension of the vector representation of a single hidden layer fully-connected feed-forward neural network is determined using (16), where $I$ is the number of input neurons, $H$ is the number of hidden neurons, and $O$ is the number of output neurons.

$$D = (I + 1) * H + (H + 1) * O \qquad (16)$$

For the connection weights, these are initialized by assigning random values from a uniform distribution in the range of $\left[\frac{-1}{\sqrt{fan-in}}, \frac{1}{\sqrt{fan-in}}\right]$, where the value of *fan-in* is the number incoming connection weights of a given neuron [29].

The number of neurons in the input and output layers in a neural network are problem-specific whereas a trial-and-error approach is commonly used to decide on the number of neurons in the hidden layer, but a number of rules-of-thumb to obtain this value is presented in [25].

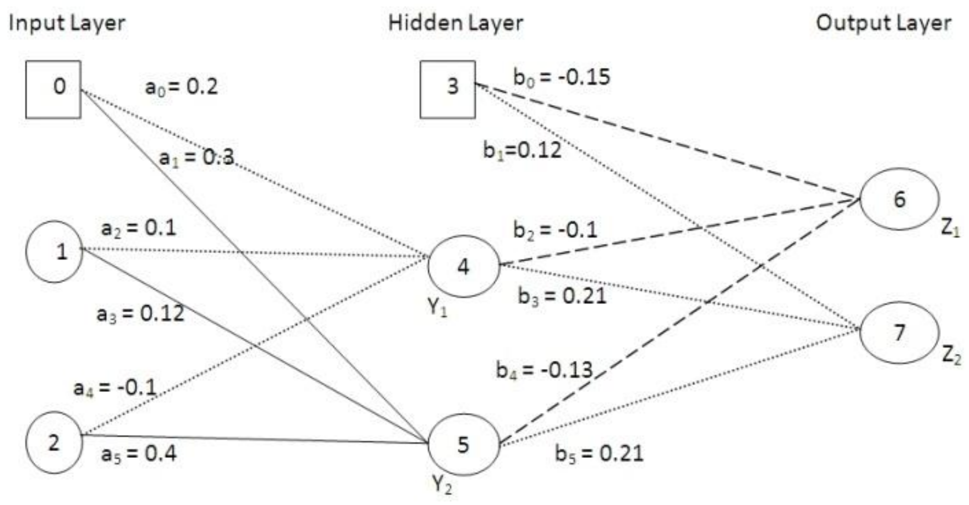
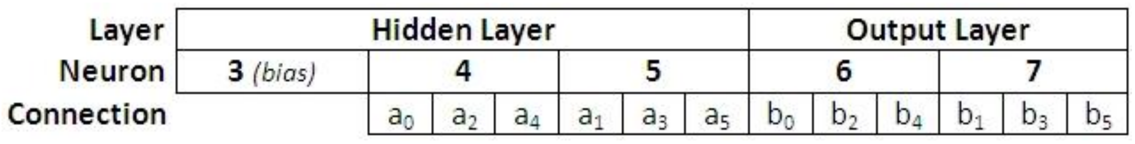


Fig. 4: A One-Hidden Layer Fully-Connected Feed-forward Neural Network



| Layer | Hidden Layer | | | | | | | Output Layer | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Neuron | 3 *(bias)* | 4 | | | 5 | | | 6 | | | 7 | | |
| Connection | | $a_0$ | $a_2$ | $a_4$ | $a_1$ | $a_3$ | $a_5$ | $b_0$ | $b_2$ | $b_4$ | $b_1$ | $b_3$ | $b_5$ |

Fig. 5: A One-Hidden Layer Fully-Connected Feed-forward Neural Network Representation

In this study, the number of neurons in the hidden layer is set to 10 which would be sufficient for the datasets selected in [30] and these will be used as benchmark datasets in the experiments.

The six datasets that will be used in the experiments are shown in Table 1. Each dataset is divided into two subsets, a training set and a test set. A training set is used during the training phase while the test set is used to evaluate the effectiveness the neural network on unseen examples.

Table 1: Description of the Datasets used as Benchmarks

| ***Domain*** | **Train** | **Test** | **Class** | **ATTRIBUTES** | |
|---|---|---|---|---|---|
| | | | | **Continuous** | **Discrete** |
| *Monks-1* | 124 | 432 | 2 | 0 | 6 |
| *Vote* | 300 | 135 | 2 | 0 | 16 |
| *Iris* | 100 | 50 | 3 | 4 | 0 |
| *Breast Cancer* | 457 | 226 | 2 | 9 | 0 |
| Heart | 180 | 90 | 2 | 6 | 7 |
| Thyroid | 3772 | 3428 | 3 | 6 | 15 |

The CSONN-OBD initializes the cat population with single hidden layer fully-connected feed-forward neural networks where each cat is represented as a vector of *D* dimension as described in (16).

At each iteration, each cat in the population is evaluated using the Mean-squared error function and after the maximum cycle is reached, the CSONN-OBD outputs the best cat representing the best neural network obtained by the CSO. The best NN found by the CSO is pruned using the Optimal Brain Damage technique to reduce its network complexity without affecting its classification and prediction capabilities. The best NN is retrained until a termination condition is satisfied. A detailed pseudo-code shows how the CSONN-OBD works and Figure 6 illustrates this procedure.

```
Read dataset
Determine the Number of Hidden Neurons and Maximum Cycles
Determine the Dimension of the Cat
    Initialize the Cats (ANNs)
While Maximum Cycle is not reached
Train ANNs
Evaluate each ANN using MSE
Output the Best Cat
Prune the Best Cat using OBD
Retrain the Best Cat and prune using OBD until the termination condition is satisfied.
Output the Pruned ANN
```

After the maximum cycle is reached the best cat is produced and this is the artificial neural network with the lowest MSE. The best cat is the best non-pruned artificial neural network. To produce a pruned ANN the best cat is subjected to the pruning process using the OBD pruning method. The termination condition for the pruning process is when the classification accuracy decreases. This means that as long as the classification accuracy of the pruned neural network is the same as the non-pruned neural network, the pruning procedure will be performed repeatedly.

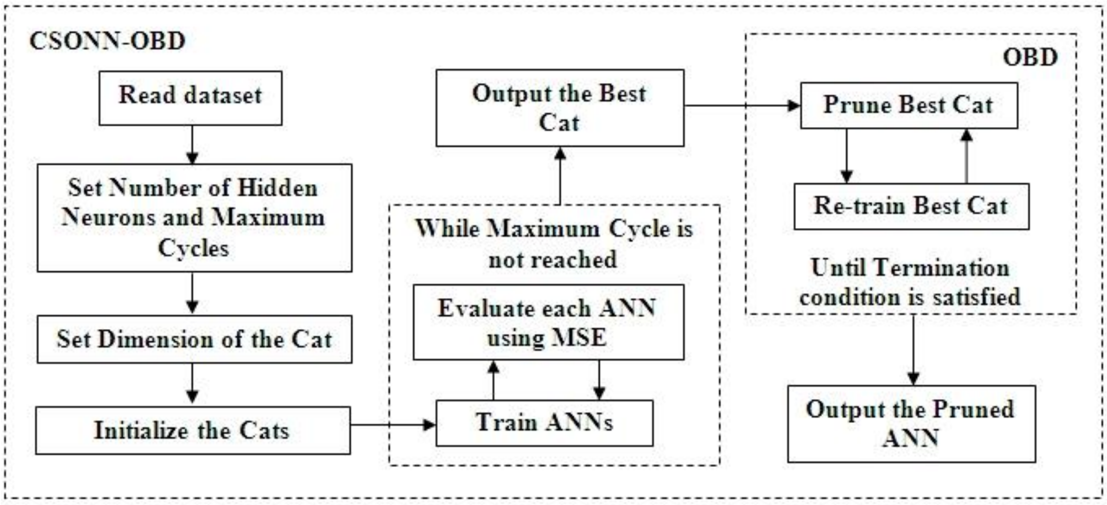

CSONN-OBD
Read dataset
Set Number of Hidden Neurons and Maximum Cycles
Set Dimension of the Cat
Initialize the Cats
While Maximum Cycle is not reached
Evaluate each ANN using MSE
Train ANNs
Output the Best Cat
OBD
Prune Best Cat
Re-train Best Cat
Until Termination condition is satisfied
Output the Pruned ANN

Fig. 6: The procedure for training neural networks with CSONN-OBD

The evaluation function used by CSONN-OBD to evaluate each cat is the Mean-squared error (MSE) function as described in (4). The MSE evaluates the neural network's performance by calculating the neural network's error on the training dataset. The lesser the error the better is the performance of the neural network. In effect, training neural network is a minimization problem since the objective is to reduce the network error given a dataset.

## VI. Experiment Results and Discussions

The aim of this research is to use the Cat Swarm Optimization algorithm with Optimal Brain Damage pruning technique to simultaneously optimize the connection weights and structure of artificial neural networks. The experiments were carried out with CSONN-OBD as the training algorithm.

To test the effectiveness of CSONN-OBD, six datasets as shown in Table 1 were used as benchmarks. For each dataset the experiments were repeated thirty (30) times to minimize the influence of random effects and to ensure that the results are statistically acceptable.

Each experiment uses a different randomly generated initial population. The result from each of the 30 independent runs of the CSONN-OBD algorithm were recorded and analyzed. Table 2 shows the parameters and their corresponding values.

Table 2: CSONN-OBD Parameters and their corresponding values

| Parameters | CSONN-OBD |
|---|---|
| Optimization Type | Minimization |
| Population Size | 100 |
| Objective Function | 1 |
| Constraints | 0 |
| Dimension of a Cat | (I + 1) * H + (H + 1) * O |
| Seeking Memory Pool (SMP) | 5 |
| Counts of Dimension to Change (CDC) | 0.8 |
| Seeking Range of the selected Dimension (SRD) | 0.2 |
| Mixture Ratio (MR) | 0.02 |
| Self Position Consideration (SPC) | True |

Table 3 shows the dimension of a cat and the maximum cycle used in the training based on the dataset used.

Table 3: Dimension of a Cat and the Maximum Cycle for each dataset used

| Dataset | Dimension | Maximum Generation |
|---|---|---|
| *Monks-1* | 81 | 500 |
| *Vote* | 181 | 300 |
| *Iris* | 83 | 500 |
| *Breast Cancer* | 111 | 300 |
| Heart | 151 | 500 |
| Thyroid | 253 | 500 |

The performance of a neural network is measured by how effective the neural network is in minimizing the mean-squared error (MSE) or the misclassification rate. Table 4 and Table 5 show a comparison on the average performance of the CSONN-OBD without pruning and CSONN-OBD with pruning, respectively while Table 6 shows a comparison on the average number of connections that were used. In the CSONN-OBD without pruning, the NN training was performed using CSONN but the neural networks were not pruned while in the CSONN-OBD with pruning, the NN training was performed using CSONN and then applied OBD to prune the neural networks.

Table 4: Average Mean-squared error on the training and test set

| Datasets | Training Set | | Test Set | |
|---|---|---|---|---|
| | Without Pruning | OBD | Without Pruning | OBD |
| Monks-1 | 2.48% | 2.47% | 3.68% | 3.62% |
| Vote | 1.64% | 1.65% | 1.77% | 1.80% |
| Iris | 0.91% | 0.90% | 1.72% | 1.71% |
| Breast Cancer | 0.86% | 0.87% | 1.61% | 1.61% |
| Heart | 7.33% | 7.40% | 6.44% | 6.52% |
| Thyroid | 0.52% | 0.51% | 0.56% | 0.55% |

Table 5: Average percentage of misclassification on the training and test sets

| Datasets | Training Sets | | Test Sets | |
|---|---|---|---|---|
| | Without Pruning | OBD | Without Pruning | OBD |
| Monks-1 | 6.13% | 5.73% | 9.52% | 9.32% |
| Vote | 3.62% | 3.51% | 4.30% | 4.40% |
| Iris | 3.60% | 3.57% | 6.20% | 6.27% |
| Breast Cancer | 1.95% | 1.87% | 3.79% | 3.83% |
| Heart | 18.87% | 18.61% | 15.96% | 15.85% |
| Thyroid | 3.80% | 3.77% | 4.23% | 4.18% |

Table 6: Average number of connection used

| Datasets | Pruning Method | | |
|---|---|---|---|
| | Without Pruning | OBD | |
| | | Used Connections | Percentage |
| Monks-1 | 81 | 42.7 | 52.72% |
| Vote | 181 | 86 | 47.51% |
| Iris | 83 | 52.8 | 63.61% |
| Breast Cancer | 111 | 44.67 | 40.24% |
| Heart | 151 | 62.53 | 41.41% |
| Thyroid | 253 | 168.87 | 66.75% |

Table 4 and Table 5 show a comparison between pruned networks and non-pruned networks with respect to their average MSE and average misclassification rate. In Table 4, it shows that the average MSE can be slightly higher for artificial neural networks that are pruned than the artificial neural networks that are not pruned. This is because with pruned artificial neural networks, fewer connections are used compared to artificial neural networks that are not pruned as shown in Table 6 and this can lead to a slight increase in the network error. The difference between the average MSE and average misclassification rate of pruned ANNs and non-pruned ANNs may be insignificant but pruned ANNs use lesser number of connections to achieve the same level of classification accuracy with non-pruned ANNs. The results show that the CSONN-OBD with pruning produces artificial neural networks that use less number of connections but are as effective as the artificial neural networks produced by CSONN-OBD without pruning.

The performance of the CSO-based ANN optimizer with OBD pruning method is compared to that of the existing algorithms which also optimize connections weights and NN structures concurrently. When presented with a completely new set of data, the capability to generalize is one of the most significant criteria to determine the effectiveness of artificial neural network learning. Table 7 compares the results obtained with that of MGNN [31] and NN-MOPSOCD [29] in terms of the error on the test set and the number of connections used. Table 7 shows that the CSONN-OBD is very effective in generating simple and accurate artificial neural networks with good generalization capability.

Table 7: Performance comparison between MGNN, NN-MOPSOCD and CSONN-OBD

| **Algorithms** | **MSE on Test Set** | | **Number of Connections** | |
|---|---|---|---|---|
| | Breast | Iris | Breast | Iris |
| MGNN-ep | 3.28% | 6.17% | 80.87 | 56.38 |
| MGNN-rank | 3.33% | 7.28% | 68.46 | 47.06 |
| MGNN-roul | 3.05% | 8.43% | 76.40 | 55.13 |
| NN-MOPSOCD | 1.68% | 4.58% | 48.13 | 66.02 |
| CSONN-OBD *without pruning* | **1.61%** | 1.72% | 111.00 | 83.00 |
| CSONN-OBD *with pruning* | **1.61%** | **1.71%** | **44.67** | **52.80** |

## VII. Conclusion

The CSONN-OBD algorithm, a CSO-based ANN optimizer with OBD pruning algorithm, was able to generate artificial neural networks with low training error and high classification accuracy given that it has a low misclassification rate. Thus, the cat swarm optimization algorithm is an effective training algorithm for artificial neural networks. As a training algorithm, the CSO was able to produce artificial neural networks that perform well using different datasets.

Using the Optimal Brain Damage pruning method, the CSO-based ANN optimizer was able to obtain artificial neural networks that are as effective as the artificial neural networks that were not pruned but pruned ANNs used fewer connections to achieve the same performance. With OBD, the CSONN-OBD algorithm was able to generate simpler neural networks but still with good generalization capability. As a result, the CSONN-OBD was able to generate an optimal set of connection weights and ANN structure for each of the dataset that was used in the experiments.

Thus, the CSO algorithm can be considered as an effective training algorithm for artificial neural networks. With a pruning method like the Optimal Brain Damage, the CSO-based ANN optimizer can produce artificial neural networks that use fewer connections but are still as effective as the artificial neural networks that are not pruned, that is, it is able to produce accurate and simple artificial neural network models. Also, the CSONN-OBD produced artificial neural networks with high classification accuracy.

## Acknowledgments

The author would like to thank the reviewers for their careful reading of this paper and for their helpful comments and suggestions. This work was supported by the University of the Philippines Visayas In-House Research Program under grant no. *SP10-06*.

## References

[1] Z. Huanping, L. Congying, Y. Xinfeng. Optimization research on Artificial Neural Network Model. Proceedings of the 2011 International Conference on Computer Science and Network Technology, (2011), pp. 1724-1727.

[2] H. Shi. Evolving Artificial Neural Networks Using GA and Momentum. Proceedings of the 2009 Second International Symposium on Electronic Commerce and Security, ISECS '09, (2009), (1): pp. 475-478.

[3] M. Paliwal. and U. Kumar. A. Neural Networks and Statistical Techniques: A Review of Applications. Expert Systems with Applications, (2009), 36(1), pp. 2-17.

[4] H. Shi and W. Li. Artificial Neural Networks with Ant Colony Optimization for Assessing Performance of Residential Buildings. Proceedings of the International Conference on Future BioMedical Information Engineering, FBIE 2009, (2009), pp. 379-382.

[5] B. A. Garro, H. Sossa and R. A. Vázquez. Artificial Neural Network Synthesis by means of Artificial Bee Colony (ABC) Algorithm. Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, (2011), pp. 331-338.

[6] Y. Wang, Z. Xia and Y. Huo. Neural Network Research Using Particle Swarm Optimization. Proceedings of the 2011 International Conference on Internet Computing and Information Services, ICICIS '11, (2011), pp. 407-410.

[7] S-C. Chu and P-W. Tsai. Computational Intelligence based on the Behavior of Cats. International Journal of Innovative Computing, Information and Control, (2007), 3(1), pp. 163-173.

[8] S-C Chu, P-W Tsai and J-S. Pan. Cat Swarm Optimization. Proceedings of the 9th Pacific Rim International Conference on Artificial Intelligence, LNAI 4099, Guilin, (2006), pp. 854-858.

[9] J.-C. Hwang, J.-C. Chen and J.-S. Pan. CSO and PSO to Solve Optimal Contract Capacity for High Tension Customers. Proceedings of 8th International Conference on Power Electronics and Drive Systems, PEDS-2009, (2009), pp. 76-81.

[10] Y. Le Cun, J. S. Denker and S. A. Solla. Optimal Brain Damage. Advances in Neural Information Processing Systems, Touretzky, DS (ed), Morgan Kaufmann, San Mateo, (1990), (2): pp. 598–605.

[11] M. Gethsiyal Augasta and T. Kathirvalavakumar. A Novel Pruning Algorithm for Optimizing

Feedforward Neural Network of Classification Problems. Neural Processing Letters, (2011), 34(3), pp. 241-258.

[12] L. Li and B. Niu. Designing Artificial Neural Networks Using MCPSO and BPSO, Proceedings of the 2008 International Conference on Computational Intelligence and Security, CIS 2008, (2008), pp. 176-179.

[13] J. Tu, Y. Zhan and F. Han. A Neural Network Pruning Method Optimized with PSO Algorithm. In Proceedings of the 2010 Second International Conference on Computer Modeling and Simulation, ICCMS '10, (2010), (3), pp. 257-259.

[14] T. Orlowska-Kowalska and M. Kaminski. Effectiveness of Saliency-Based Methods in Optimization of Neural State Estimators of the Drive System with Elastic Couplings. IEEE Transactions on Industrial Electronics, (2009), 56(10), pp. 4043-4051.

[15] T. Orlowska-Kowalska and M. Kaminski. Optimization of Neural State Estimators of the Two-mass System using OBD method. Proceedings of the IEE International Symposium on Industrial Electronics, ISIE 2008, (2008), pp. 461-466.

[16] I. Sansa, N. B, Mrabet and M. Bouzid Ben Khader. Effectiveness of the Saliency-Based Methods in Optimization of NN Structure for Induction Motor Fault Diagnosis. Proceedings of the 8th International Multi-Conference on Systems, Signals & Devices, (2011), pp.1-7.

[17] I. A. Basheer and M. Hajmeer. Artificial Neural Networks: Fundamentals, Computing, Design, and Application, Journal of Microbiological Methods, (2000), 43, pp. 3-31.

[18] X. Yao. Evolving Artificial Neural Networks. Proceedings of the IEEE, (1999), (87): 1423-1447.

[19] C. Ozturk and D. Karaboga. Hybrid Artificial Bee Colony Algorithm for Neural Network Training. Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2011, (2011), pp. 84-88.

[20] E. Alba and J. Chicano. Training Neural Networks with GA Hybrid Algorithms, K. Deb (ed.). Proceedings of GECCO ’04, Seattle, Washington, LNCS 3102, (2004), pp. 852-863.

[21] Y. Liu and X. Yao. A Population-Based Learning Algorithm Which Learns Both Architectures and Weights of Neural Networks. Chinese J. Advanced Software Res., (1996), 3(1), pp. 54-65.

[22] E. Cantu ́-Paz. Pruning Neural Networks with Distribution Estimation Algorithms. Proceedings of the 2003 International Conference on Genetic and Evolutionary computation, GECCO'03, (2003), 1: pp. 790-800.

[23] J. Sum and C-s. Leung. On the Error Sensitivity Measure for Pruning RBF networks. In Proceedings of the Second International Conference on Machine Learning and Cybernetics, (2003), pp. 1162-1167.

[24] J. Yang, A. Bouzerdoum and S. Phung. A Neural Network Pruning Approach based on Compressive Sampling. Proceedings of International Joint Conference on Neural Networks 2009, (2009), pp. 3428-3435.

[25] M. Shahin, M. Jaksa and H. Maier. Application of Neural Networks in Foundation Engineering. Theme paper to the International e-Conference on Modern Trends in Foundation Engineering: Geotechnical Challenges and Solutions, Theme No. 5: Numerical Modelling and Analysis, Chennai, India, (2004).

[26] B. Hassibi and D. G. Stork. Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. Advances in Neural Information Processing Systems 5, [NIPS Conference], Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, (1992), pp. 164-171.

[27] S. Samarasinghe. Neural Networks for Applied Sciences and Engineering, Auerbach Publications, Boston, MA, (2006).

[28] J. Ding, J. Shao, Y. Huang, L. Sheng, W. Fu and Y. Li. Swarm Intelligence Based Algorithms for Data Clustering. Proceedings of the 2011 International Conference on Computer Science and Network Technology, (2011), pp. 577-581.

[29] J. P. T. Yusiong and P. C. Naval, Jr. Training Neural Networks Using Multiobjective Particle Swarm Optimization, Lecture Notes in Computer Science, ICNC 2006, (2006), (1): pp. 879-888.

[30] D. Newman, S. Hettich, C. Blake and C. Merz. UCI Repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science, (1998).

[31] P. Palmes, T. Hayasaka and S. Usui. Mutation-based Genetic Neural Network. IEEE Transactions on Neural Networks, (2005), 16(3), pp. 587-600.

## Author’s Profile

**John Paul T. YUSIONG** received his B.S. degree in Computer Science *(cum laude)* from the University of the Philippines Visayas Tacloban College (UPVTC), Tacloban City, Leyte, Philippines in 2002 and his M.S. degree in Computer Science from the University of the Philippines Diliman (UPD), Diliman, Quezon City, Philippines in 2006. He has been teaching for ten years and he is currently an Assistant Professor in Computer Science at the University of the Philippines Visayas Tacloban College, Tacloban City, Leyte, Philippines. His research interests include Artificial Intelligence, Neural Networks and Optimization algorithms.