Improved Swarm-based Algorithms for Neural Network Training

by

Marshall Joseph

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

in

The Faculty of Mathematics and Sciences

Department of Computer Science

BROCK UNIVERSITY

October 26, 2021

Abstract

Swarm Intelligence (SI) algorithms have been used to train Artificial Neural Networks (ANNs) since the introduction of Particle Swarm Optimization (PSO) in 1995. Since then, hundreds of other SI algorithms have been developed and improved upon for a variety of uses. The purpose of this thesis is to train ANNs using three popularized SI algorithms and compare their performance against an improved variant version of each. The ANNs are trained for classification problems. SI algorithms are compared in terms of training accuracy, training loss, testing accuracy, testing loss, and overfitting. Observations conducted in this study point to a noticeable improvement of nearly all performance measures when using an improved variant version of each algorithm, with Genetic Artificial Bee Colony Optimization (GABC) and Variable Step Size Firefly Algorithm (VFA) having the best overall performance.

Acknowledgements

To my parents for their constant encouragement and support throughout all of my past, present, and future endeavours. To Prof. Ombuki-Berman for her supervision, guidance, and expertise in the field that made this research possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis is a comparative study of three swarm intelligence algorithms and three improved versions of these algorithms. The way the performance of each algorithm is being measured is by using them to train artificial neural networks to classify the information from six different data sets. The results that will be compared in this thesis are the accuracy, loss, and overfitting indicator that is generated by each neural network for each data set. The three algorithms studied in this thesis are Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Artificial Bee Colony Optimization (ABC). The variants of these algorithms are Time Variant Particle Swarm Optimization (TVPSO), Variable Step Size Firefly Algorithm (VFA), and Genetic Artificial Bee Colony Optimization (GABC).

1.1 Overview

In the field of Artificial Intelligence (AI), there exists a branch of powerful, population-based, metaheuristic optimization algorithms called Evolutionary Algorithms (EA). EAs are a subset of Evolutionary Computation (EC) which are inspired by the process of evolution found in nature and can be formulated by an evolutionary process. Another related technique to EAs are Swarm Intelligence (SI) algorithms such as Particle Swarm Optimization which represent the solutions to difficult optimization problems as the position of an agent in the population. These algorithms follow the evolutionary process by updating the position of each agent through use of a mathematical function that favours better performing agents and their positions, allowing them to improve and evolve over time.

Particle Swarm Optimization is a SI algorithm proposed by Kennedy and Eberhart in [12]. The algorithm is inspired by the flocking behaviour found in certain species in nature such as birds or schools of fish and was originally intended to simulate social behaviour [11]. Since its development, it has been simplified to be used for optimization problems instead. PSO is not a gradient descent algorithm, but otherwise known as a metaheuristic. This means that the most optimal solution to a problem is not always guaranteed to be found, however, PSO can find good solutions for many problems [12]. In the past, PSO has shown to lack some promising qualities when training ANNs, although performing better than the traditional backpropagation algorithm for most problems. It has been proved that PSO often generates ANNs that are overfit and oversaturated [19]. In this thesis, a variant of PSO called Time Variant PSO is introduced which was developed with the goal to improve the ability of generalization and self-learning of ANNs [17].

The Firefly Algorithm is a SI algorithm inspired by the flashing behaviour of fireflies [27]. The brightness of each firefly corresponds to the fitness of the function being optimized. This means that the fireflies that are performing best for the given problem are the ones who shine brightest and attract the fireflies who are less bright [27]. Movement occurs on an iterative basis

where all fireflies are attracted to any other firefly in the population. In the past, FA has shown to converge rather slowly when compared to its counterparts [22]. In this thesis, a variant of FA called Variable Step Size FA is introduced which was developed with the goal to converge faster with less error to produce optimum solutions [22].

The Artificial Bee Colony Optimization Algorithm is a SI algorithm inspired by the intelligent behaviour of the honey bee swarm [14]. The colony consists of three groups of bees: employed bees, onlooker bees, and scout bees [7]. Each employed bee is assigned a food source to evaluate the amount of nectar and dance on. In this case, the evaluation of nectar is equal to the fitness function for the given problem. The food sources with more nectar are more likely to influence onlooker bees to come visit them. They also choose a food source based on the dance of the bee [14]. Lastly, if a food source becomes abandoned, the corresponding employed bee will become a scout bee and go search for a new food source. In this thesis a variant of ABC called Genetic ABC is introduced which adds a genetic mutation component to the onlooker bee phase. This mutation is meant to avoid local optima and increase convergent rate [26].

In machine learning, the artificial neural network (ANN) is a powerful, versatile model that is able to perform a variety of difficult tasks efficiently and with high accuracy. A few of the many use-cases for ANNs include: speech recognition, function approximation, data processing, and directing robotics prostheses. An ANN is a collection of connected nodes referred to as artificial neurons, which are meant to simulate the firing of real neurons in the human brain. Several weights and bias values are also part of the ANNs which are optimized by the swarm intelligence algorithms. In this study, a three layer, fully connected, feed-forward neural network will be the chosen topology for all experiments. The three layers consist of an input layer which receives each attribute from the data set, a hidden layer which applies some nonlinear activation function to each of the inputs, and an output layer which also applies a nonlinear function to each of the hidden layer outputs, and works to generate an output that resembles the expected outcome as close as possible to the respective data set.

1.2 Objectives

This thesis explores how different swarm intelligence algorithms and their improved variants can be used to train artificial neural networks. The data sets used in this study are classification data sets. Classification refers to the AI's ability to recognize similarities and differences between input data and give the respective class or category that each datum belongs to. For instance, an ANN may receive information about an animal, such as its number of legs, its weight, height, average lifespan, and other attributes. Based on this information, the ANN will make a prediction of its class such as dog or cat. The goal of this study is to determine which methods of training produce the strongest ANN for each classification data set. Six data sets were chosen from the UCI machine learning repository. The data sets are: Iris, Glass, Sonar, Wine, Zoo, and Parkinsons. Each SI algorithm will be run several times for each data set and the best set of connection weights will have their results measured. 80% of each data set will be used to train the networks, then, the remaining 20% of the data set will be used for testing the networks. Results from both training and testing will be recorded and compared in the results section of this paper. The parameter settings for each experiment will come from [14] or from

other literature.

1.3 Contribution

The following items represent the novel contributions of this work:

- Three improved versions of SI algorithms designed for training artificial neural networks are implemented, trained, and tested against six classification data sets.
- Data from improved versions of SI algorithms are compared against data from vanilla versions of SI algorithms.
- Previous data in [14] is compared against data from this paper to determine similarities and differences across all experiments.
- For each SI algorithm and each data set, those that were observed to perform best will be mentioned.

It was found that most of the variants improved upon the results of their vanilla implementations. Specific performance measures and a comparison study will be conducted in Chapter 7.

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 provides much of the necessary background information used to follow this study. Swarm Intelligence (SI) algorithms are described in detail alongside Artificial Neural Networks (ANNs), the activation functions used in the ANNs are then explained. Lastly, measurements such as loss and overfitting are broken down.

Chapter 3 provides the background information describing the improved SI algorithm variants used in this study. SI variants are algorithms based on vanilla SI except modifications have been made that are meant to improve performance. The specific modifications will be outlined in this chapter.

Chapter 4 presents some examples of previous work of each SI algorithm and their application to neural network training. Much of the previous work comes from [14], however there are many other studies that are important to note and provide fundamental findings which give this research a purpose.

Chapter 5 describes each algorithm in the form of pseudo-code at the implementation level.

Chapter 6 outlines the parameters for each SI algorithm and ANN architecture used in each experiment.

Chapter 7 presents the results and other findings from each experiment. Results are broken down into training results, testing results, and overfitting results.

Chapter 8 offers conclusions and future work related to this study.

Appendix A lists additional experimental analysis not directly included in the results chapter.

Chapter 2

Background Information

2.1 Swarm Intelligence Algorithms

Swarm Intelligence (SI) algorithms are a subset of Evolutionary Algorithms (EA) which are inspired by the process of evolution found in nature and can be formulated by an evolutionary process. Agents of a particular swarm can be used as solutions to continuous or discrete optimization problems and rely on the movement or behaviour of other agents in the swarm to influence each other. Some major properties of SI algorithms include [14]:

- Each SI algorithm is composed of many similar individuals that belong to the same class or a small set of sub-classes.
- Agents of a swarm may only interact with each other through use of information exchange or by interacting with their environment.
- The swarm itself will behave in a self-organizing fashion.

Each of the six algorithms used to train ANNs in this thesis are categorized as SI algorithms.

2.1.1 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a SI algorithm proposed by Kennedy and Eberhart in [12]. The algorithm works by keeping track of a population of solutions which influence the movement of each other across the search space. The movement of a particle is calculated by a velocity Equation 2.1 and a position update Equation 2.2. Each particle keeps track of its current position, best position, current fitness value, best fitness value, and current velocity. The velocity of a particle represents the speed at which it is able to move across the search space to find new solutions. Its position represents a current solution to the problem being optimized. All particles in the swarm are ranked based on fitness. In maximization problems, particles with a higher fitness value will be ranked highest in the swarm and have the most influence over the other particles. In minimization problems, particles with a lower fitness value will be ranked highest in the swarm and have the most influence over the other particles. In this paper, a fully connected neighbourhood of particles is used. In a fully connected neighbourhood, all particles share their information to each other in order to satisfy the velocity and position update equations efficiently.

The velocity update function for each particle is calculated per datum per iteration using Equation 2.1:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1(y(t) - x_i(t)) + c_2 r_2(\hat{y}(t) - x_i(t)) \tag{2.1}$$

Where:

- t is the current iteration.
- $x_i(t)$ is the current position of the particle at dimension i.
- $v_i(t)$ is the current velocity of the particle at dimension i.
- $y_i(t)$ is the best position of the current particle at dimension i.
- $\hat{y}_i(t)$ is the global best position at dimension i.
- w is the inertial term which applies a portion of the previous velocity to the next velocity, typically in the range [0, 1].
- c_1 is the cognitive component, which influences the effect of the best found position of the current particle, typically in the range [0, 2].
- c_2 is the social component, which influences the effect of the global best found position, typically in the range [0, 2].
- r_1 & r_2 are randomly generated values in the range [0, 1].

The position of each particle is then updated using Equation 2.2:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.2}$$

Where each variable corresponds to the same description from Equation 2.1.
After each iteration, the result from the velocity function gets added to the current position of the particle. Thus, moving its position towards a combination of the local and global best found positions in the swarm.

2.1.2 Firefly Algorithm (FA)

Firefly Algorithm (FA) is a SI algorithm introduced by Yang and is based on the flashing characteristics of fireflies [27]. Flashing characteristics of fireflies can be summarized by the following three rules [28]:

- All fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex.
- Attractiveness is proportional to firefly brightness. For any couple of flashing fireflies, the less bright one will move towards the brighter one. The brightness decreases when the distance between fireflies is increased. The brightest firefly moves randomly, because there is no other bug to attract it.
- The brightness of a firefly is affected or determined by the landscape of the objective function to be optimized.

The brightness of each firefly is established by Equation 2.3:

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{2.3}$$

Where:

- β_0 is the firefly brightness value at r=0.
- γ is the light absorption coefficient.

Fireflies movement is based on the principles of attractiveness: when firefly j is more attractive than firefly i, the movement is determined by the following equation [2]:

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_j - x_i) + \alpha\epsilon_i \tag{2.4}$$

Where:

- x_i is the position of firefly i.
- x_j is the position of firefly j.
- α is a randomization term in the range [0, 1].
- ϵ is the scaling parameter found by subtracting the lower bound from the upper bound of the function being optimized.
- r_{ij} is the Cartesian distance found by Equation 2.5.

Equation 2.5 outlines how the Cartesian distance between two fireflies is obtained:

$$r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2} \tag{2.5}$$

Where d is the dimension of the firefly. In the proposed firefly algorithm, an α reduction function is used to reduce geometric progression found in Equation 2.6.

$$\alpha(t) = a_0 * 0.9^t \tag{2.6}$$

Where t is the current iteration. The parameter gamma characterizes the variation of the attractiveness, thus controlling the convergence and behaviour of the FA [14].

2.1.3 Artificial Bee Colony Optimization (ABC)

A more recent swarm intelligence algorithm is the Artificial Bee Colony (ABC) algorithm, originally proposed by Karaboga [8] and inspired by the foraging behaviour of honeybees [10]. In the ABC algorithm, a solution to the optimization problem is represented by the position of each food source. The amount of nectar at each food source corresponds to its fitness value. The colony consists of three groups of bees: employed bees, onlooker bees, and scout bees [7]. The employed bees are bees that belong to a food source and therefore represent

a solution vector to the given optimization problem. Each food source is randomly generated upon initialization using Equation 2.7:

$$x_{mi} = l_i + rand(0,1) * (u_i - l_i) \tag{2.7}$$

Where u_i is the upper bound of position i and l_i is the lower bound of position i. Even though employed bees have a food source, they still like to search locally for other potential food sources using the following Equation 2.8. The employed bee does this by producing a modification on the position in its memory depending on the local information and tests the nectar amount of the new source [9]. Neighbouring food sources are determined using Equation 2.8:

$$v_{mi} = x_{mi} + \phi_{mi}(x_{mi} - x_{ki}) \tag{2.8}$$

Where:

- x_{mi} is the current food source k at a randomly chosen parameter index i.
- x_{ki} is a randomly chosen food source k at a randomly chosen parameter index i.
- $k \in \{1, 2, \ldots, SN\}$ and $i \in \{1, 2, \ldots, D\}$ must be different from m.
- ϕ_{mi} is a randomly generate number between (l_i, u_i).

The following fitness function is used to determine the quality of each food source. If fitness < 0, it will be normalized to a positive value by using absolute value operator seen in Equation 2.9:

$$fit(x_m) \begin{cases} \frac{1}{1+y(x_m)}, & \text{if } y(x_m) \geq 0 \\ 1 + abs(y(x_m)), & \text{if } y(x_m) < 0 \end{cases} \tag{2.9}$$

The fitness of each employed bee will be calculated during the employed bee phase after applying Equation 2.8. The fitness values are then stored and used to determine probabilities of each bee being chosen during the onlooker phase using Equation 2.10:

$$p_m = \frac{fit_m(\overrightarrow{x_m})}{\sum_{m=1}^{SN} fit_m(\overrightarrow{x_m})} \tag{2.10}$$

Where:

- fit_i is the fitness value of the solution vector $\overrightarrow{x_m}$ and is proportional to the amount of nectar at the food source.
- SN is the number of food sources.

The scout bees have a limit equal to $n * dimensions$ which allows for control of the exploration process, while the onlooker bees and employed bees carry out the exploitation process. Each time a new food source is rejected due to poor fitness, the current bee increases its trial counter by 1. If the food source is accepted, the scout counter is set back to 0 and the original food source is replaced with the new food source. After all onlooker bees have finished their search for the current iteration, an onlooker bee becomes a scout bee if their trial counter $>$ limit. The scout bee will then abandon their food source entirely and find a new position to dance at using Equation 2.7.

2.2 Artificial Neural Networks

An artificial neural network (ANN) is a powerful, versatile machine learning model that is able to perform a variety of difficult tasks efficiently and with high accuracy. An ANN is made up of several interconnected neurons that receive information and give a result based on its programmed functionality. Figure 2.1 shows what a real neuron map looks like in a human

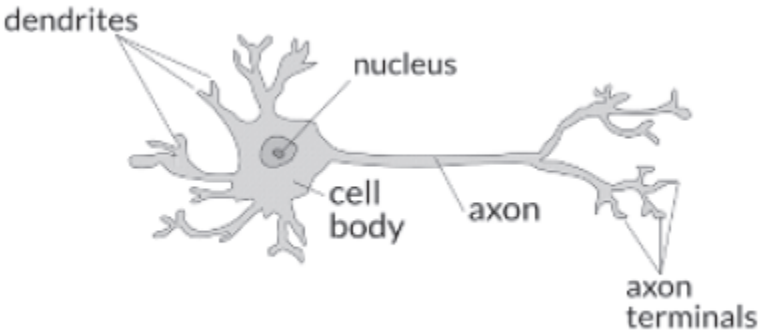

Figure 2.1: Neuron

brain. The input signals are accumulated in the cell body of the neuron, and if the accumulated signal exceeds a certain threshold, an output signal is generated which is pass on by the axon [16]. Figure 2.2 shows a model of an artificial neuron that works to simulate the process of a

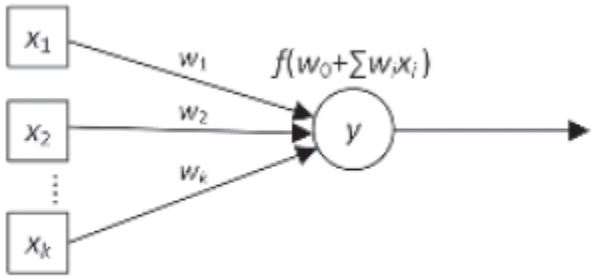

Figure 2.2: Artificial Neuron

real neuron as close as possible. The artificial neuron takes as input a vector $(x_1, x_2, \ldots, x_k)$ and then applies weights $(w_0, w_1, w_2, \ldots, w_k)$ to that input yielding a weighted sum [16]:

$$w_0 + \sum_{i=1}^{k} w_i x_i \tag{2.11}$$

Once the input has been summed, it is ran through an activation function and the final output is given. This function must be nonlinear to allow for nonlinear relationships to be modelled [14]. The activation function will be further discussed in Section 2.3. An artificial neural network

is built from one to many artificial neurons which can communicate information to each other. In a fully connected ANN, all input nodes communicate to all hidden nodes, and all hidden nodes communicate to all output nodes. The first layer of an ANN is the input layer, where

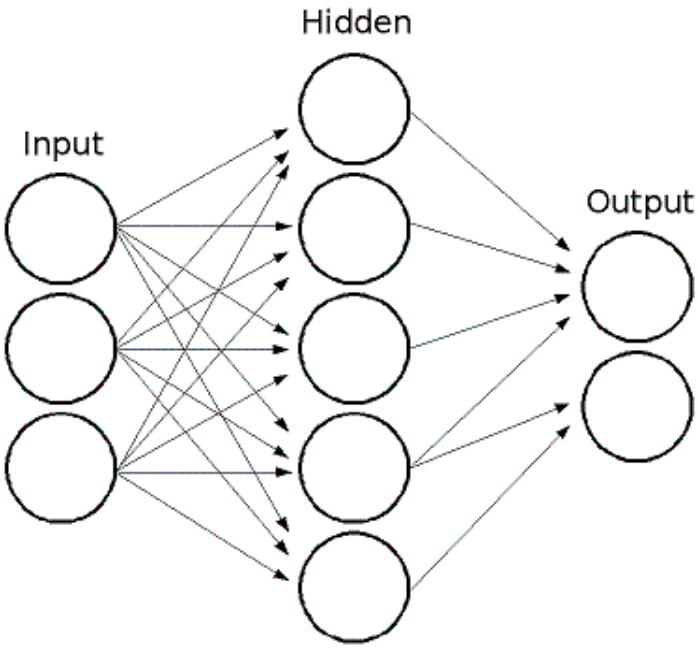

Figure 2.3: An Artificial Neural Network

each input node is associated with an attribute from the data set. For instance, the Iris data set contains entries with 4 attributes pertaining to sepal length in cm, sepal width in cm, petal length in cm, and petal width in cm. This means that an ANN for the Iris data set would need 4 input nodes; one for each attribute. After the attributes are read into the input layer, a set of hidden nodes are established. The number of hidden nodes can be difficult to determine, and sometimes becomes a problem in itself to find the best number of hidden nodes to use. In general, there are some rules that can be followed:

- The number of hidden nodes should be between the number of input nodes and the number of output nodes.
- The number of hidden nodes should be 2/3 of the number of input nodes, plus the number of output nodes.
- The number of hidden nodes should be less than two times the number of input nodes.

In this thesis, the number of hidden nodes used comes from literature where experiments have been run to determine the optimal number to use for each data set. The input from the hidden nodes is multiplied by its corresponding weight and fed into the hidden nodes. From there, the hidden nodes run an activation function on each of the inputs, and the output from the hidden layer is then passed to the output layer. The output layer is made up of several neurons that give a prediction amount based on confidence for classification data sets. For the Iris data set,

there are 3 types of Iris flowers. This means that the output layer will consist of 3 neurons. The output layer also runs an activation function similar to the hidden layer before giving its final prediction. This activation function could be the same activation function used in the hidden layer, or it could be different. In this paper, the experiments run will use two activation functions. At the hidden layer, the activation function used is called the hyperbolic tangent function (tanh). At the output layer, the activation function used is called softmax.

2.3 Activation Functions

Activation functions allow for a non-linear relationship to be modelled by the ANN [14]. The two activation functions used in this study are hyperbolic tangent and softmax, described below:

2.3.1 The Hyperbolic Tangent Function

The hyperbolic tangent function, also known as tanh, is written as:

$$f(net) = \frac{(e^{net} - e^{-net})}{(e^{net} + e^{-net})} \tag{2.12}$$

The function takes in a single input value and outputs a value between (-1, 1). Normalizing the values of the network between this range allows for outputs from each neuron to be restricted to a set value. If there was no restrictions on these values, they could grow exponentially high, creating computational issues in complex, deep neural network architectures. A graph of the hyperbolic tangent function can be found in Figure 2.4.

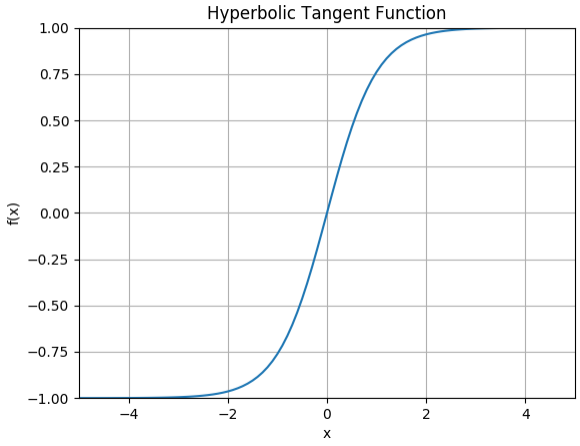

Figure 2.4: The Hyperbolic Tangent Function

2.3.2 The Softmax Function

The softmax function is written as:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.13}$$

Where:

- σ is the result from the softmax.
- $\vec{z}$ is the input vector
- e^{z_i} is the standard exponential function for the input vector
- e^{z_i} is the standard exponential function for the output vector
- K is the number of classes

The softmax function works by taking in an entire vector rather than a single value and applying an exponential normalization function over each value. The result of this is that a vector z consisting of K real numbers is transformed into a probability distribution proportional to each input value. Figure 2.5 shows an example of the softmax function being applied to the output layer of an ANN. In this figure, the output layer is using softmax as its activation function.

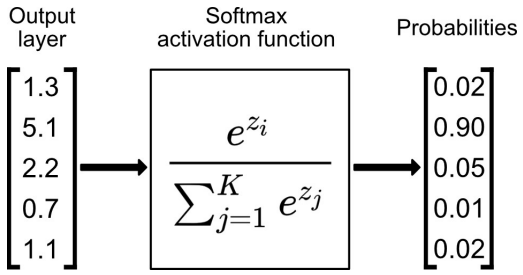

Figure 2.5: A Softmax Function Example

A vector z consisting of $K = 5$ real numbers is fed into the function and normalized into a probability distribution. The probability distribution is proportional to each value and adds up to 1. The number 1.3 becomes 0.02, the number 5.1 becomes 0.90, the number 2.2 becomes 0.05, the number 0.7 becomes 0.01, and the number 1.1 becomes 0.02. The highest input value from vector z has the highest probability value in the output vector. Therefore as the result of the ANN classification, value 2 from the probability vector would be the ANN's prediction.

2.4 Loss

The measure of loss is often described as the penalty an ANN incurs as a result of a bad prediction. If an ANN makes a correct guess, the loss for that guess is equal to 0. If an

ANN makes an incorrect guess, the loss for that guess is greater than 0 and proportional to the measure of loss used. In this paper, the measure of loss used is mean squared error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.14}$$

Where:

- MSE is the result from mean squared error.
- n is the number of data points.
- Y_i are the truth values.
- $\hat{Y}_i$ are the ANN's predicted values.

MSE is calculated after each pass during the training phase of the feed-forward ANN. For each classification data set, the goal of the algorithm is to minimize the loss during each pass. This is also known as the fitness function. ANNs with a lower MSE are the most fit ANNs. However, it is possible that some ANNs may become overfit.

2.5 Overfitting

The training of ANNs can be done in a variety of ways, with the traditional way being to partition the data set into three exclusive sets: training, validation, and testing [14]. A model that is well-fit will be able to accurately predict training data and perform well during validation. When it comes to testing, the ANNs prediction accuracy should be similar or better than its performance during training. In statistics, an overfit model is a statistical model that contains more parameters than can be justified by the data [5]. Similarly, a machine learning model that has been overfit has been taught to memorize the training data, rather than understand why it is making its predictions. This means that while an ANN may get perfect accuracy on the training data because it has seen it so many times, when the new testing data is presented, the accuracy will be very low in comparison. To lower the chances of overfitting happening, several methods have been introduced such as cross-validation or data pruning. In this study, the data is simply split into two sets, training data and testing data, and fed through the network. This means overfitting can occur quite easily. Since this is the case, a measure of indicating overfitting has been introduced.

2.5.1 Indication of Overfitting

As previously mentioned, overfitting in ANN training is the phenomenon in which the ANN has adequate performance on the training data set but poor performance on the testing data set or any other unseen data [14]. In [20], a generalization factor was developed to indicate overfitting behaviour of ANNs, seen in Equation 2.15:

$$pf = \frac{e_{test}}{e_{train}} \tag{2.15}$$

Where:

- e_{test} is the ANN loss on the testing data.
- e_{train} is the ANN loss on the training data.
- $pf > 1$ indicates overfitting may have occurred due to a generalization error greater than that of the training error.
- $pf \leq 1$ indicates overfitting is unlikely due to a generalization error being less than that of the training error.

In this thesis, pf will be used as an indication of potential overfitting across all six SI algorithms.

Chapter 3

Improved Swarm Intelligence Algorithms

Improved Swarm Intelligence algorithms are classified as:

- SI algorithms that are based on another algorithm such as PSO.
- Have at least one modified component of the original algorithm which is meant to improve the effectiveness.
- Have been used in academic research papers with proof of improved results.

The first improved SI algorithm that will be examined in this thesis is Time Variant Particle Swarm Optimization (TVPSO) which is a variant of Particle Swarm Optimization (PSO). The second improved SI algorithm that will be examined is Variable Step Firefly Algorithm (VFA) which is a variant of Firefly Algorithm (FA). The third and final improved SI algorithm that will be examined is Genetic Artificial Bee Colony Optimization (GABC) which is a variant of Artificial Bee Colony Algorithm (ABC).

3.1 Time Variant Particle Swarm Optimization (TVPSO)

The concept of time varying parameters has been used in earlier PSO research [15], [24]. In this thesis, the TVPSO algorithm implemented is based on [17]. TVPSO works by adaptively and dynamically altering both acceleration coefficient values and the inertia weight value with respect to iterations and fitness value. By doing so, the algorithm is able to explore the search space more efficiently [17]. The particle velocity update equation and position update equation remains the same from the PSO algorithm.

The inertia weight value works to balance the global exploration and local exploitation of each particle in the swarm. In TVPSO, an adaptive and dynamic inertia weight w is given by Equation 2.11:

$$w = \begin{cases} w_{min} + \frac{(w_{max} - w_{min})(f - f_{min})}{(f_{avg} - f_{min})}, & f \leq f_{avg} \\ w_{max}, & f > f_{avg} \end{cases} \tag{3.1}$$

Where:

- w_{min} is the minimum inertia weight.
- w_{max} is the maximum inertia weight.
- f is the current fitness value of the particle.
- f_{min} is the minimum fitness value of all particles.

- f_{avg} is the average fitness value of all particles.

By using Equation 2.11, w is varied depending on the fitness value of the particle, which provides a good way to maintain population diversity and sustain good convergence capacity [17].

$$c_1 = \frac{c_{1f} + (c_{1i} - c_{1f})(maxiter - iter)}{maxiter} \tag{3.2}$$

$$c_2 = \frac{c_{2f} + (c_{2i} - c_{2f})(maxiter - iter)}{maxiter} \tag{3.3}$$

Where:

- c_1 is the cognitive component, which influences the effect of the best found position of the current particle, typically in the range [0,2].
- c_2 is the social component, which influences the effect of the best found position of the current particle, typically in the range [0,2].
- c_{1i} is the initial value of c_1.
- c_{2i} is the initial value of c_2.
- c_{1f} is the final value of c_1.
- c_{2f} is the final value of c_2.

The time variant coefficients c_1 and c_2 are updated each iteration using Equation 2.12 and Equation 2.13. The value of c_1 starts as a large value and the value of c_2 starts as a small value. This allows the particles to move around the search space rather than converging quickly towards the current best particle. As time goes on, c_1 becomes smaller and c_2 becomes larger which allows the particles to converge to the global optima. This can result in a more effective optimization of particles due to a longer exploration process.

3.2 Variable Step Firefly Algorithm (VFA)

In the standard Firefly Algorithm (FA), the step size α is not a variable, but rather a constant decimal value. This means it cannot follow the search mechanism properly [22]. Since the step size α has a large influence on the convergence and exploration process, modification of α across iterations could be an improvement of current conditions. To account for this, a way of dynamically adjusting the step size α has been adopted as the following Equation 2.14:

$$\alpha(t) = \frac{0.4}{1 + e^{\frac{0.015*(t-maxgen)}{3}}} \tag{3.4}$$

Where:

- t is the number of iterations remaining.

- $maxgen$ is the total number of iterations.

By using a variable step size, the goal is to verify whether or not the VFA algorithm converges faster and performs better when compared to FA. The equations for firefly brightness and movement remain the same from FA to VFA.

3.3 Genetic Artificial Bee Colony Optimization (GABC)

The Genetic Artificial Bee Colony Algorithm (GABC) follows many of the same principles as ABC. The solution vector for each optimization problem is represented by the position of each food source. Alongside this, the amount of nectar at each food source corresponds to its fitness value. The colony consists of three groups of bees: employed bees, onlooker bees, and scout bees [7]. The employed bee phase functions the same way as the original ABC. For the onlookers phase, a change made in GABC is that in order to avoid local optima and increase convergence rate, a new method for searching for new food sources with genetic mutation is proposed in Equation 3.5 [26].

$$v_{mi} = \begin{cases} x_{best} + F_t(x_{pi} - x_{qi}), & \text{if } F_2 < 0.7 + 0.2\frac{t}{T} \\ x_{mi} + F_1(x_{mi} - x_{ki}), & \text{else} \end{cases} \tag{3.5}$$

Where:

- x_m is the original food source
- x_p is the first unique, diverse food source used for comparing.
- x_q is the second unique, diverse food source used for comparing.
- x_k is the third unique, diverse food source used for comparing.
- t is the current iteration.
- T is the maximum number of iterations.
- $F_1 = rand(-1, 1)$
- $F_2 = rand(0.6, 0.9)$
- $F_t = F_2\frac{\sqrt{T^2-t^2}}{T}$

During early iterations, the second condition of Equation 3.5 is met much more frequently due to the fact that F_2 is more likely to be greater than the threshold. This means that onlooker bees will continue searching for new food sources in the vicinity they are currently in. As time goes on, meeting the first condition of Equation 3.5 becomes easier as F_2 will begin getting smaller. This means that onlooker bees will begin searching for new food sources in a random direction from the current best food source.

Chapter 4

Swarm Intelligence Algorithms Previous Work

Swarm Intelligence (SI) algorithms have been used in a variety of ANN research since their development. In this thesis, many experiments are run based on the parameters of three best performing algorithms in [14]: Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Artificial Bee Colony Optimization (ABC). [14] found that when applied to regression data sets, PSO was the best performing algorithm in terms of testing loss. FA outperformed PSO in terms of testing loss on the classification data sets. ABC had an average rank across all experiments.

4.1 Particle Swarm Optimization

Before the development of Particle Swarm Optimization (PSO), the traditional method of ANN training was to use the backpropagation algorithm. In [6] it was found that PSO requires less iterations of training to achieve a similar level of error as the backpropagation algorithm [14]. [13] used PSO to train an ANN for medical diagnoses with a small sample size and a large amount of features. It was found that the backpropagation algorithm was preferred for this type of data where only a small number of samples are available and the number of features are very large. In [3], several ANN architectures were tested using PSO and the results obtained by this methodology were situated between the results presented by other well studied techniques such as genetic algorithms or simulated annealing. Lastly, [14] performed a comparison study between PSO and six other SI algorithms when training ANNs. They found that PSO was the algorithm that performed the best when applied to all data sets and ranked first overall in terms of testing loss. However, PSO had the worst ranking based on the overfitting indicator which falls in line with previous research in this area [14]. Several algorithms outperformed PSO in terms of accuracy such as FA.

4.2 Firefly Algorithm

The Firefly Algorithm (FA) was used in [21] to train an ANN for character recognition in Microsoft Paint. The result was that FA combined with backpropagation had a better performance and quicker convergence rate than any of the comparable algorithms. Alongside this, [2] used FA to train ANNs for classification problems such as XOR, 3-bit parity, and 4-bit Encoder-Decoder. The result from was that ABC performed the best, FA performed second best, and genetic algorithms came last. Finally, [14] performed a comparison study between FA and six

other SI algorithms when training ANNs. They found that FA surpassed PSO in testing accuracy when applied to classification data sets. FA also outperformed PSO in terms of testing loss on classification data sets.

4.3 Artificial Bee Colony Optimization

The Artificial Bee Colony (ABC) algorithm has been applied to the training of ANNs in previous works with mild levels of success [14]. The ABC algorithm was used to train an MLP ANN on oil spill detection in [18]. It was found that ABC can certainly be used to train an MLP ANN instead of backpropagation or Levenberg-Marquardt (LM) learning algorithms, however LM is the preferred algorithm for this task. In a 2012 study, [23] applied ANNs trained by ABC algorithm to predict the temperature of a volcano based on time-series data. The result was that the vanilla ABC trained ANN outperformed the ANN trained by backpropagation. Lastly, [14] performed a comparison study between ABC and six other SI algorithms when training ANNs. They found that ABC had mild levels of success across all experiments. ABC was surpassed by PSO and FA, but outperformed other algorithms such as bacterial foraging optimization algorithm (BFA) and ant colony optimization algorithm (ACO).

4.4 Time Variant Particle Swarm Optimization

The Time Variant Particle Swarm Optimization (TVPSO) algorithm is an improved variant of the PSO algorithm and has been used in [17] to train fuzzy ANNs for a speech recognition system. The study outlines experiments that compare vanilla PSO, TVPSO, and backpropagation against several vocabulary sets. It was found that in a vocabulary set of 10, TVPSO had an average recognition rate of 96.5, PSO had an average recognition rate of 96.0, and backpropagation had an average recognition rate of 95.0 [17]. TVPSO also performed better than PSO and backpropagation in every other set of vocabulary, including a set of 20, 30, 40, and 50. Vanilla PSO came a close second in each experiment, and backpropagation trained ANNs trailed in third place. In this study, TVPSO and PSO will be compared to see if TVPSO will continue to be more accurate than PSO for classification data sets.

4.5 Variable Step Size Firefly Algorithm

The Variable Step Size Firefly Algorithm (VFA) is an improved variant of FA and has been used in [22] to train MLP equalizers for channel equalization. The study outlines experiments that compare vanilla PSO, FA, and VFA in multiple channel functions. The results from this study show that the VFA algorithm trained equalizer provided the same BER at a reduced SNR which is a remarkable advantage in comparison to the two traditional algorithm based equalizers [22]. The VFA has also been proven in this study to converge quicker than the other two algorithms. In this thesis, FA and VFA will be compared to see if VFA will continue to be more efficient than FA for classification data sets.

4.6 Genetic Artificial Bee Colony Optimization

The Genetic Artificial Bee Colony Algorithm (GABC) is an improved variant of ABC and has been used in [26] for cooperative outage detection in improved RBF neural networks. The study outlines experiments that compare K-nearest neighbours (KNN) algorithm, standard RBF with back propagation (RBF) algorithm, and improved RBF with genetic artificial bee colony algorithm (GABC). The results from this study shows that KNN has a much lower detection accuracy than those of neural networks. Meanwhile, the genetic factor in GABC produces better performance as a result of better approximation to the global optimum and the application of artificial bee colony algorithm also obtains a quicker convergence [26]. Therefore, ideal performance in the dense network environment was obtained through use of GABC algorithm. In this paper, GABC and ABC will be compared to see if GABC improves on ABC in terms of accuracy, loss, and overfitting indicator for classification data sets.

Chapter 5

Algorithm Implementation

This section defines the implementation details of each of the six SI algorithms used to train ANNs. Each algorithm in this thesis was implemented from scratch using Java JDK Version 8. Each algorithm uses a population size of 50 and a max generation count of 1000 [14]. The fitness of each ANN is equal to the mean squared error of the output function after a single feed-forward pass. The feed-forward pass works by unflattening the position vector of the agent into a set of matrix weights and biases, and using them as the weights and biases of the ANN for multiplication. After the feed-forward pass, the SI algorithm flattens the weights back into a position vector and modifies it using its unique position update equation seen below:

5.1 Particle Swarm Optimization

The implementation of PSO trained ANN was based on [14]. Pseudo-code for the algorithm is detailed in Algorithm 1.

Algorithm 1 Particle Swarm Optimization

```
while Current_Generation < Max_Generation do
    for Particles in Population do
        Evaluate Particle Fitness
        if Particle_Fitness < Personal_Best_Fitness then
            Update Personal Best Fitness
            Update Personal Best Position
        end
        if Particle_Fitness < Global_Best_Fitness then
            Update Global Best Fitness
            Update Global Best Position
        end
    end
    for Particles in Population do
        Update Particle Velocity
        Update Particle Position
    end
    if Rnd_Double < Prob_Death then
        Kill Current Particle
        Initialize New Random Particle
    end
end
```

5.2 Firefly Algorithm

The implementation of FA trained ANN was based on [14]. Pseudo-code for the algorithm is detailed in Algorithm 2.

Algorithm 2 Firefly Algorithm

```
while Current_Generation < Max_Generation do
    for i = 1 : Population_Size do
        for j = 1 : i do
            if I_j > I_i then
                Modify Attractiveness According To Distance r
                Move Firefly i towards j
                Evaluate New Solution And Update Light Intensity I
            end
        end
    end
end
```

5.3 Artificial Bee Colony Optimization

The implementation of ABC trained ANN was based on [14]. Pseudo-code for the algorithm is detailed in Algorithm 3.

Algorithm 3 Artificial Bee Colony Optimization

```
while Current_Generation < Max_Generation do
    for i = 1 : Num_Food_Sources do
        Employed Bee Searches Locally For Food Source Using Equation 2.8
        if fit(Current_Food_Source) > fit(New_Food_Source) then
            Move To New Food Source
        end
    end
    for i = 1 : Num_Food_Sources do
        Onlooker Bee Probabilistically Chooses New Food Source Using Equation 2.10.
        if Rnd_Double < Probability then
            Move To New Food Source
        end
    end
    for i = 1 : Num_Food_Sources do
        if Bee_Trial > Trial_Limit then
            Scout Bee Moves To New Food Source Using Equation 2.7
        end
    end
end
```

5.4 Time Variant Particle Swarm Optimization

The implementation of TVPSO trained ANN was based on [24]. Pseudo-code for the algorithm is detailed in Algorithm 4.

Algorithm 4 Time Variant Particle Swarm Optimization

```
while Current_Generation < Max_Generation do
    for Particles in Population do
        Evaluate Particle Fitness
        Calculate Acceleration Coefficients According To Equations 3.2 and 3.3
        Calculate Inertia Weight According To Equation 3.1.
        if Particle_Fitness < Personal_Best_Fitness then
            Update Personal Best Fitness
            Update Personal Best Position
        end
        if Particle_Fitness < Global_Best_Fitness then
            Update Global Best Fitness
            Update Global Best Position
        end
    end
    for Particles in Population do
        Update Particle Velocity
        Update Particle Position
    end
    if Rnd_Double < Prob_Death then
        Kill Current Particle
        Initialize New Random Particle
    end
end
```

5.5 Variable Step Size Firefly Algorithm

The implementation of VFA trained ANN was based on [22]. Pseudo-code for the algorithm is detailed in Algorithm 5.

Algorithm 5 Variable Step Size Firefly Algorithm

```
while Current_Generation < Max_Generation do
    Determine The Step Size α
    for i = 1 : Population_Size do
        for j = 1 : i do
            if I_j > I_i then
                Modify Attractiveness According To Distance r
                Move Firefly i towards j
                Evaluate New Solution And Update Light Intensity I
            end
        end
    end
end
```

5.6 Genetic Artificial Bee Colony Optimization

The implementation of GABC trained ANN was based on [26]. Pseudo-code for the algorithm is detailed in Algorithm 6.

Algorithm 6 Genetic Artificial Bee Colony Optimization

```
while Current_Generation < Max_Generation do
    for i = 1 : Num_Food_Sources do
        Employed Bee Searches Locally For Food Source Using Equation 2.8
        if fit(Current_Food_Source) > fit(New_Food_Source) then
            Move To New Food Source
        end
    end
    for i = 1 : Num_Food_Sources do
        Onlooker Bee Probabilistically Chooses New Food Source Using Equation 3.5
        Onlooker Bee Moves To Genetically Modified Food Source
    end
    for i = 1 : Num_Food_Sources do
        if Bee_Trial > Trial_Limit then
            Scout Bee Moves To New Food Source Using Equation 2.7
        end
    end
end
```

Chapter 6

Experimental Setup

6.1 Neural Network Architectures

In this study, the ANN architecture used across all experiments was a 3-layer, feed-forward, ANN. The ANN consists of an input layer, a hidden layer, and an output layer. The input layer has a number of nodes equal to the number of attributes in the data set. The hidden layer has a number of nodes that comes from literature as referenced in Table 6.1. The output layer has a number of nodes equal to the number of classes in the data set.

Dataset	Number of Hidden Neurons
Iris [4]	4 [1]
Sonar [4]	30 [14]
Glass [4]	9 [1]
Parkinsons [4]	20 [14]
Wine [4]	10 [25]
Zoo [4]	10 [14]

Table 6.1: Neural Network Architectures

Many of the experimental parameters in this thesis were taken from [14] since the work from three of the algorithms needed to be replicated. If the number of hidden neurons could not be determined from other literature, [14] used a genetic algorithm. The genetic algorithm used in [14] was implemented using the Keras ANN framework. Keras is a machine learning framework designed for easy prototyping of ANN models written in Python [14]. Using this approach, the genetic algorithm would generate potential ANNs, train and test them, then genetic operators would be applied such as crossover, tournament selection, and mutation. The range of hidden nodes tested were set in increments of five, starting at 5 to a maximum of 500 [14].

6.2 Swarm Intelligence Parameters

The three vanilla SI algorithm parameters were taken from [14] where Bayesian Optimization (BO) was used to tune the parameters for each experiment. The remaining three improved SI algorithm parameters were taken from other literature. Tables for each SI algorithm parameter set can be seen below:

All parameters for the PSO algorithm with tanh and softmax activation function come from [14].

Dataset	W	c_1	c_2
Iris	0.49452	1.57499	1.98687
Sonar	0.69614	1.54019	1.92433
Glass	0.63589	1.37756	1.00825
Parkinsons	0.44225	1.42556	1.51641
Wine	0.90000	1.00000	2.00000
Zoo	0.69983	1.54484	1.87629

Table 6.2: PSO Parameters

Dataset	α	$Beta_0$	γ
Iris	0.99000	0.01000	17
Sonar	0.16235	0.16790	11
Glass	0.01000	0.01000	20
Parkinsons	0.40974	0.41740	25
Wine	0.49612	0.42815	20
Zoo	0.57417	0.99000	26

Table 6.3: FA Parameters

All parameters for the FA algorithm with tanh and softmax activation function come from [14].

Dataset	Employed Percentage
Iris	0.94317
Sonar	0.82849
Glass	0.57400
Parkinsons	0.66325
Wine	0.56693
Zoo	0.78172

Table 6.4: ABC Parameters

All parameters for the ABC algorithm with tanh and softmax activation function come from [14].

Dataset	w_{init}	w_{final}	$C1_{init}$	$C1_{final}$	$C2_{init}$	$C2_{final}$
Iris	0.4	0.9	1.5	0.5	1.8	2.5
Sonar	0.4	0.9	1.5	0.5	1.8	2.5
Glass	0.4	0.9	1.5	0.5	1.8	2.5
Parkinsons	0.4	0.9	1.5	0.5	1.8	2.5
Wine	0.4	0.9	1.5	0.5	1.8	2.5
Zoo	0.4	0.9	1.5	0.5	1.8	2.5

Table 6.5: TVPSO Parameters

All parameters for the TVPSO algorithm come from [24].

Dataset	α	$Beta_0$	γ
Iris	1.0	0.99	0.01
Sonar	1.0	0.99	0.01
Glass	1.0	0.99	0.01
Parkinsons	1.0	0.99	0.01
Wine	1.0	0.99	0.01
Zoo	1.0	0.99	0.01

Table 6.6: VFA Parameters

All parameters for the VFA algorithm come from [22].

Dataset	Employed Percentage
Iris	0.5
Sonar	0.5
Glass	0.5
Parkinsons	0.5
Wine	0.5
Zoo	0.5

Table 6.7: GABC Parameters

All parameters for the GABC algorithm come from [26].

Chapter 7

Results

This section of the study outlines the results found after running each experiment. Section 7.1 details the results based on training, Section 7.2 details the results based on testing, and Section 7.3 details the results based on the overfitting indicator described in Section 2.5. The ranking system used was to take a look at the results from each algorithm for each data set and determine which algorithm performed the best for the given measure. For training accuracy and testing accuracy, algorithms that had a higher accuracy scored best. For training loss, testing loss, and overfitting indicator, algorithms that had lower values scored best. The average rank for each algorithm for each data set is also described, where a higher rank means the algorithm performed better. The number of first place measures is also noted on the last row of each table. The six data sets used can be classified as either a small or medium data set. The small data sets contain less than 500 instances or less than 50 features. These data sets are Iris, Glass, Parkinson's, Wine, and Zoo. The medium data set contains more than 500 instances or more than 50 features, but less than 10,000 instances and less than 500 features. The medium data set is Sonar.

7.1 Training Results

The first measurement used in this thesis is training accuracy. Training accuracy was calculated as a percentage by dividing the number of correct guesses by the number of total guesses made by each algorithm. The overall results for training accuracy are outlined in Table 7.1. The algorithm that had the highest training accuracy was GABC, second place was ABC, third place was VFA, fourth place was FA, and last place was a tie between PSO and TVPSO.

Algorithm	**PSO**	**TVPSO**	**FA**	**VFA**	**ABC**	**GABC**
Iris	3	3	4	2	1	1
Sonar	6	5	4	3	2	1
Glass	5	4	6	3	2	1
Parkinsons	3	3	1	2	4	2
Wine	4	5	3	2	1	1
Zoo	3	4	2	2	3	1
Average Rank	4	4	3.333	2.333	2.167	1.167
Rank	5	5	4	3	2	1
First Frequency	0	0	1	0	2	5

Table 7.1: Training Accuracy Rankings

The second measurement used in this thesis is training loss. Training loss was calculated

using MSE as previously described in Section 2.4. The overall results for training loss are outlined in Table 7.2. The algorithm that had the best training loss was GABC, second place was ABC, third place was VFA, fourth place was FA, fifth place was TVPSO, and last place was PSO. These results similarly reflect the results of Table 7.1. Based on these rankings, TVPSO looks to outperform PSO in terms of the training set, however more analysis is needed to confirm this.

Algorithm	**PSO**	**TVPSO**	**FA**	**VFA**	**ABC**	**GABC**
Iris	6	4	5	3	2	1
Sonar	6	5	4	3	2	1
Glass	6	5	4	3	2	1
Parkinsons	5	6	1	3	4	2
Wine	5	6	4	3	1	2
Zoo	5	6	4	3	2	1
Average Rank	5.5	5.333	3.667	3	2.1667	1.333
Rank	6	5	4	3	2	1
First Frequency	0	0	1	0	1	4

Table 7.2: Training Loss Rankings

7.2 Testing Results

The third measurement used in this thesis is testing accuracy. Testing accuracy was calculated as a percentage by dividing the number of correct guesses by the number of total guesses made by each algorithm. The overall results for testing accuracy are outlined in Table 7.3. The algorithm that had the highest testing accuracy was GABC, second place was VFA, third place was ABC, fourth place was a tie between PSO and TVPSO, and last place was FA. The results for the testing data are similar to that of the training data but are not exactly the same. In this data, we see that GABC remains the best performing algorithm, however VFA outperforms ABC and takes over second place. FA is the worst performing algorithm in terms of overall testing accuracy. PSO and TVPSO are shown to both perform better than FA whereas FA outperformed both PSO and TVPSO in terms of training accuracy.

The fourth measurement used in this thesis is testing loss. Testing loss was calculated using MSE as previously described in Section 2.4. The overall results for testing loss are outlined in Table 7.4. The algorithm that had the best testing loss was ABC, second place was a tie between GABC, FA, and PSO, third place was FA, and last place was TVPSO. These results do not fall in line with the results from testing accuracy. This shows that some of the data may be scattered and the best performing ANNs with low testing loss could be outliers. GABC had the best performance in terms of testing accuracy, however ABC performed better than its variant in terms of testing loss.

Algorithm	**PSO**	**TVPSO**	**FA**	**VFA**	**ABC**	**GABC**
Iris	1	1	2	1	1	1
Sonar	1	2	3	2	4	4
Glass	5	4	4	3	2	1
Parkinsons	2	4	1	3	2	3
Wine	4	3	3	2	2	1
Zoo	3	2	5	3	4	1
Average Rank	2.667	2.667	3	2.333	2.5	1.833
Rank	4	4	5	2	3	1
First Frequency	2	1	1	1	1	4

Table 7.3: Testing Accuracy Rankings

Algorithm	**PSO**	**TVPSO**	**FA**	**VFA**	**ABC**	**GABC**
Iris	1	5	6	2	3	4
Sonar	1	4	3	2	6	5
Glass	5	6	4	3	1	2
Parkinsons	2	6	1	4	3	5
Wine	6	5	4	3	2	1
Zoo	4	3	6	5	1	2
Average Rank	3.167	4.833	4	3.167	2.667	3.167
Rank	2	4	3	2	1	2
First Frequency	2	0	1	0	2	1

Table 7.4: Testing Loss Rankings

7.3 Overfitting Results

The fifth and final measurement used in this thesis is the overfitting indicator. The overfitting indicator was calculated using the formula in Section 2.5. The overall results for overfitting indicator are outlined in Table 7.5. The algorithm that was least overfit was PSO, second place was TVPSO, third place was GABC, fourth place was ABC, fifth place was VFA, and last place is FA. These results go against the results found in [14] because PSO was found to be the most overfit algorithm. In this study, PSO was found to be the best performing algorithm in terms of overfitting indicator.

Algorithm	**PSO**	**TVPSO**	**FA**	**VFA**	**ABC**	**GABC**
Iris	1	4	6	2	3	5
Sonar	1	2	3	4	5	6
Glass	4	3	1	2	5	6
Parkinsons	2	6	1	5	3	4
Wine	4	3	5	2	6	1
Zoo	1	2	6	5	4	3
Average Rank	2.167	3.333	3.667	3.333	4.333	4.167
Rank	1	2	3	2	5	4
First Frequency	3	0	2	0	0	1

Table 7.5: Overfitting Indicator Rankings

Chapter 8

Conclusion

In this section, final remarks about each SI algorithm and their performance will be made. A comparison between each SI algorithm and their variant will also be mentioned to understand whether or not the variant is an improvement for each performance measure used in this study. To recap, six SI algorithms were studied to determine which performed best overall, and which variants were likely to improve on their vanilla counterparts. The algorithms chosen for this study were Particle Swarm Optimization (PSO) and a variant Time Variant Particle Swarm Optimization (TVPSO), Firefly Algorithm and a variant Variable Step Size Firefly Algorithm (VFA), and lastly Artificial Bee Colony Optimization (ABC) and a variant Genetic Artificial Bee Colony Optimization (GABC). The measures used to determine performance are training accuracy, training loss, testing accuracy, testing loss, and an overfitting indicator.

8.1 Performance of SI Algorithms

Results from each SI algorithm have been tallied against six classification data sets. Based on this data, the following conclusions can be made:

- In terms of training accuracy, the best performing algorithm was GABC which came first place in 5 of the 6 data sets and second place in 1 of the 6 data sets.
- Moreover, the worst performing algorithms were PSO and TVPSO which ranked the same in terms of training accuracy.
- GABC performed better than vanilla ABC, VFA performed better than FA when looking at the training accuracy results. Overall the variants do perform better when training ANNs across all 6 data sets.
- In terms of training loss, the best performing algorithm was GABC which also had the best performance in terms of training accuracy. ABC was a close second.
- Moreover, the worst performing algorithm was PSO which goes against the results found in previous studies. PSO had high accuracy overall but did not perform as well as the other algorithms. TVPSO performed better than its vanilla PSO implementation.
- Each of the three variants performed better than their vanilla counterparts in terms of training loss.
- In terms of testing accuracy, the best performing algorithm was GABC which came first place in terms of training accuracy as well. However vanilla ABC did not perform as well as VFA when it came to testing accuracy.

- The worst ranking algorithm in terms of testing accuracy was FA which does not reflect the results found when looking at the training accuracy. PSO and TVPSO tied for fourth place in terms of testing accuracy which shows an improvement from their rankings of training accuracy. Overall the Firefly variant showed the most improvement from its vanilla counterpart going from last place to second place.

- In terms of testing loss, the best performing algorithm was ABC. PSO, GABC, and VFA all tied for second place. This shows that each algorithm had similar testing loss results except for TVPSO which had poor performance in this category.

- PSO had more first place rankings in both a small data set and the medium data set. This would put PSO ahead of VFA and GABC in terms of testing loss. Overall each variant does not improve the performance of testing loss except for VFA.

- Lastly, the best performing algorithm in terms of overfitting was PSO which comes to a surprise as PSO is known to generate ANNs that are overfit. Second place was TVPSO and VFA which shows that the variant does not improve overfitting performance.

- The algorithm that generated the most overfit ANNs was ABC and GABC closely trailed. Overall, vanilla PSO performed better than its variant, but VFA and GABC performed better than the vanilla versions in this performance measure.

- To conclude, the overall best performing algorithms across all measures were Genetic Artificial Bee Colony Optimization (GABC) and Variable Step Size Firefly Algorithm (VFA). Both algorithms consistently improved upon their vanilla versions and had high training accuracy, high testing accuracy, and a low level of overfitting that occured in their ANNs.

8.2 Future Work

This thesis focused on the results from only three SI algorithms and a single variant from each. In the future, research could be made in comparing these algorithms against new, improved variants to see which variant performed the best. There are also many other SI algorithms that were not tested such as Ant Colony Optimization (ACO) or Bacterial Foraging Optimization (BFO) as studied in [14]. These algorithms could be studied and improved upon with variants to see if they perform better than their vanilla implementations. Another option for future work would be to include large data sets in the study. Since some algorithms perform better when there is a large number of instances or features in the data, this could narrow the reasons for better or worse performance down to the type of data being used. Furthermore, parameter tuning of each variant was not accomplished in this study, and could easily lead to higher performance. Lastly, future work could include comparison of training techniques such as holdout percentage, cross validation, or inclusion of other metrics such as the computational cost of each algorithm.

Bibliography

[1] Anna Bosman and Andries Engelbrecht. Measuring saturation in neural networks. 2015.

[2] Ivona Brajevic and Milan Tuba. Training feed-forward neural networks using firefly algorithm. 2013.

[3] Marcio Carvalho and Teresa B. Ludermir. Particle swarm optimization of neural network architectures and weights. In *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, pages 336–339, 2007.

[4] Dheeru Dua and Casey Graff. Uci machine learning repository. 2017.

[5] B. S. Everitt and A. Skrondal. The cambridge dictionary of statistics. 2010.

[6] V.G. Gudise and G.K. Venayagamoorthy. Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, pages 110–117, 2003.

[7] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. 2005.

[8] Dervis Karaboga and Bahriye Akay. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, pages 108–132, 2009.

[9] Dervis Karaboga, B. Basturk, and Celal Ozturk. Artificial bee colony (abc) optimization algorithm for training feed-forward neural networks. *Model. Decis. Artif. Intell.*, 3:318–319, 2007.

[10] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga. A comprehensive survey: Artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review*, 2012.

[11] J. Kennedy. The particle swarm: social adaptation of knowledge. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pages 303–308, 1997.

[12] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948, November 1995.

[13] Maciej A. Mazurowski, Piotr A. Habas, Jacek M. Zurada, Joseph Y. Lo, Jay A. Baker, and Georgia D. Tourassi. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural Networks*, 21(2-3):427–436, 2008.

[14] Reginald McLean. Swarm-based algorithms for neural network training. MSc thesis. Brock University, 2019.

[15] Senthil Arumugam Muthukumaraswamy, Machavaram Rao, and Alan Tan. A novel and effective particle swarm optimization like algorithm with extrapolation technique. *Appl. Soft Comput.*, 9:308–320, 2009.

[16] Richard E. Neapolitan and Xia Jiang. *Artificial Intelligence: With an Introduction to Machine Learning, Second Edition*. CRC Press, 2018.

[17] Aiping Ning, Xueying Zhang, and Hui Sun. A speech recognition system based on fuzzy neural network optimized by time variant pso. In *2010 International Conference on Computational Aspects of Social Networks*, pages 497–500, 2010.

[18] Coskun Ozkan, Celal Ozturk, Filiz Sunar, and Dervis Karaboga. The artificial bee colony algorithm in training artificial neural network for oil spill detection. *Neural Network World*, 21:473–492, 2011.

[19] A. Rakitianskaia and A. Engelbrecht. Measuring saturation in neural networks. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 1423–1430, December 2001.

[20] A. Röbel. Dynamic pattern selection: Effectively training backpropagation neural networks. In Maria Marinaro and Pietro G. Morasso, editors, *ICANN '94*, pages 643–646, London, 1994. Springer London.

[21] M. Sahoo, Janmenjoy Nayak, S. Mohapatra, B. Nayak, and Dr. H. Behera. *Character Recognition Using Firefly Based Back Propagation Neural Network*, volume 32, pages 151–164. 2015.

[22] A. Sarangi, S. Priyadarshini, and S. K. Sarangi. A mlp equalizer trained by variable step size firefly algorithm for channel equalization. In *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, pages 1–5, 2016.

[23] Habib Shah, Rozaida Ghazali, and Nazri Mohd Nawi. Hybrid ant bee colony algorithm for volcano temperature prediction. 2012.

[24] Praveen Kumar Tripathi, Sanghamitra Bandyopadhyay, and Sankar Kumar Pal. Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients. *Information Sciences*, 177(22):5033–5049, 2007.

[25] Andrich B. van Wyk and Andries P. Engelbrecht. Overfitting by pso trained feedforward neural networks. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[26] Y. Wang, P. Long, N. Liu, Z. Pan, and X. You. A cooperative outage detection approach using an improved rbf neural network with genetic abc algorithm. In *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, 2018.

[27] Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2008.

[28] Xin-She Yang. Firefly algorithms for multimodal optimization. 2010.

Appendix A

Experimental Analysis

	Best	Average	Best	Average
Training Accuracy	**PSO**	**PSO**	**TVPSO**	**TVPSO**
Iris	0.966667	0.849583	0.966667	0.8525
Sonar	0.73494	0.667169	0.746988	0.678916
Glass	0.461988	0.371637	0.555556	0.355556
Parkinsons	0.858974	0.815064	0.858974	0.824359
Wine	0.838028	0.784155	0.830986	0.778521
Zoo	0.703704	0.607407	0.679012	0.609877

Table A.1: Training Accuracy Results PSO & TVPSO

	Best	Average	Best	Average
Training Accuracy	**FA**	**FA**	**VFA**	**VFA**
Iris	0.958333	0.78875	0.983333	0.97
Sonar	0.76506	0.651205	0.777108	0.714759
Glass	0.45614	0.366082	0.573099	0.471053
Parkinsons	0.871795	0.836859	0.865385	0.834936
Wine	0.908451	0.843662	0.93662	0.911268
Zoo	0.814815	0.525926	0.814815	0.719136

Table A.2: Training Accuracy Results FA & VFA

	Best	Average	Best	Average
Training Accuracy	**ABC**	**ABC**	**GABC**	**GABC**
Iris	1	0.986667	1	0.987083
Sonar	0.861446	0.80994	0.86747	0.813855
Glass	0.596491	0.49386	0.637427	0.533626
Parkinsons	0.852564	0.821474	0.865385	0.826282
Wine	1	0.998944	1	0.999648
Zoo	0.703704	0.640123	0.950617	0.875926

Table A.3: Training Accuracy Results ABC & GABC

	Best	Average	Best	Average
Training Loss	**PSO**	**PSO**	**TVPSO**	**TVPSO**
Iris	0.078453	0.212446	0.06697	0.210475
Sonar	0.452436	0.531193	0.443592	0.538781
Glass	0.711972	0.773838	0.684695	0.779813
Parkinsons	0.269667	0.335798	0.281666	0.321489
Wine	0.293273	0.400484	0.319638	0.403711
Zoo	0.564505	0.722309	0.586645	0.722079

Table A.4: Training Loss Results PSO & TVPSO

	Best	Average	Best	Average
Training Loss	**FA**	**FA**	**VFA**	**VFA**
Iris	0.074997	0.277114	0.030529	0.056487
Sonar	0.373131	0.432791	0.363094	0.402609
Glass	0.6691	0.73542	0.617901	0.67726
Parkinsons	0.205474	0.255001	0.261505	0.285642
Wine	0.169757	0.268229	0.123166	0.165414
Zoo	0.364328	0.538252	0.35084	0.515166

Table A.5: Training Loss Results FA & VFA

	Best	Average	Best	Average
Training Loss	**ABC**	**ABC**	**GABC**	**GABC**
Iris	0.000887	0.025307	0.000764	0.02497
Sonar	0.281187	0.369483	0.273522	0.360725
Glass	0.582046	0.618146	0.539079	0.599474
Parkinsons	0.263173	0.299134	0.246151	0.297053
Wine	1.91E-07	0.002129	1.03E-06	0.001157
Zoo	0.153159	0.19261	0.096	0.205837

Table A.6: Training Loss Results ABC & GABC

	Best	Average	Best	Average
Testing Accuracy	**PSO**	**PSO**	**TVPSO**	**TVPSO**
Iris	1	0.82	1	0.831667
Sonar	0.833333	0.622619	0.809524	0.629762
Glass	0.44186	0.327907	0.465116	0.318605
Parkinsons	0.897436	0.79359	0.846154	0.771795
Wine	0.888889	0.744444	0.944444	0.751389
Zoo	0.85	0.5525	0.9	0.5675

Table A.7: Testing Accuracy Results PSO & TVPSO

	Best	Average	Best	Average
Testing Accuracy	**FA**	**FA**	**VFA**	**VFA**
Iris	0.966667	0.766667	1	0.94
Sonar	0.785714	0.629762	0.809524	0.708333
Glass	0.465116	0.338372	0.55814	0.446512
Parkinsons	0.948718	0.811538	0.871795	0.782051
Wine	0.944444	0.813889	0.972222	0.875
Zoo	0.7	0.4825	0.85	0.66

Table A.8: Testing Accuracy Results FA & VFA

	Best	Average	Best	Average
Testing Accuracy	**ABC**	**ABC**	**GABC**	**GABC**
Iris	1	0.968333	1	0.96
Sonar	0.761905	0.645238	0.761905	0.65
Glass	0.604651	0.47093	0.697674	0.480233
Parkinsons	0.897436	0.796154	0.871795	0.764103
Wine	0.972222	0.925	1	0.915278
Zoo	0.8	0.615	0.95	0.7875

Table A.9: Testing Accuracy Results ABC & GABC

	Best	Average	Best	Average
Testing Loss	**PSO**	**PSO**	**TVPSO**	**TVPSO**
Iris	3.04E-06	0.233558	0.007091	0.241167
Sonar	0.331872	0.632053	0.359915	0.617659
Glass	0.659916	0.810292	0.749834	0.821993
Parkinsons	0.188932	0.375192	0.257426	0.42344
Wine	0.186765	0.472904	0.125124	0.455778
Zoo	0.277924	0.809534	0.195361	0.807394

Table A.10: Testing Loss Results PSO & TVPSO

	Best	Average	Best	Average
Testing Loss	**FA**	**FA**	**VFA**	**VFA**
Iris	0.066655	0.308547	1.19E-05	0.11013
Sonar	0.355487	0.460132	0.349634	0.45336
Glass	0.642371	0.746256	0.556587	0.693727
Parkinsons	0.098736	0.314573	0.227532	0.381128
Wine	0.124342	0.310909	0.055321	0.23517
Zoo	0.428884	0.5892	0.289567	0.635332

Table A.11: Testing Loss Results FA & VFA

	Best	Average	Best	Average
Testing Loss	**ABC**	**ABC**	**GABC**	**GABC**
Iris	3.64E-05	0.061868	0.00055	0.066937
Sonar	0.461119	0.668945	0.440672	0.663506
Glass	0.551256	0.619532	0.555545	0.632793
Parkinsons	0.205062	0.369632	0.244128	0.424577
Wine	0.055008	0.136349	5.31E-08	0.156316
Zoo	0.084644	0.297159	0.110795	0.388608

Table A.12: Testing Loss Results ABC & GABC

	Best	Average	Best	Average
Overfitting Indicator	**PSO**	**PSO**	**TVPSO**	**TVPSO**
Iris	3.0333E-05	1.10623701	0.05268	1.163976
Sonar	0.63839023	1.18351577	0.704384	1.144985
Glass	0.90364544	1.04774512	0.901467	1.056769
Parkinsons	0.52147315	1.13267544	0.786078	1.337613
Wine	0.4495178	1.19655188	0.371669	1.115771
Zoo	0.319285	1.13318887	0.324359	1.121487

Table A.13: Overfitting Indicator Results PSO & TVPSO

	Best	Average	Best	Average
Overfitting Indicator	**FA**	**FA**	**VFA**	**VFA**
Iris	0.270373	1.190433	0.00017805	2.11292903
Sonar	0.837103	1.066807	0.87118434	1.13040047
Glass	0.930154	1.014835	0.81193564	1.02496843
Parkinsons	0.399916	1.260248	0.74140047	1.34817649
Wine	0.634281	1.183903	0.284892	1.45988576
Zoo	0.736255	1.112958	0.55394131	1.25824867

Table A.14: Overfitting Indicator Results FA & VFA

	Best	Average	Best	Average
Overfitting Indicator	**ABC**	**ABC**	**GABC**	**GABC**
Iris	0.002178	14.10919	0.023431	20.61093
Sonar	1.089501	1.833058	1.14174	1.879879
Glass	0.860789	1.004583	0.89005	1.060005
Parkinsons	0.671859	1.250297	0.720092	1.429726
Wine	10.54873	15149.14	0.000212	14671.18
Zoo	0.510952	1.560867	0.503451	2.02265

Table A.15: Overfitting Indicator Results ABC & GABC