Comparative Study of Particle Swarm Optimization and Back-Propagation Artificial Neural Network Training Algorithms

Michael Young, Brock University
St. Catharines, Ontario
Email: my08tu@brocku.ca
Terrence Knox, Brock University
St. Catharines, Ontario
Email: tk08tu@brocku.ca

***Abstract*—Vanilla back-propagation and its variants are both a commonly used and effective way to training artificial neural networks. Vanilla back-propagation however has a tendency to have a long run time as it must do a forward pass through the network to determine the error and then propagate it backwards in order to alter the weights. In this paper we will use another algorithm, namely particle swarm optimization, to train the network on two datasets of varying complexity to determine its applicability as a neural network training algorithm.**

I. Introduction

Back-propagation is a commonly used technique to train artificial neural networks as it consistently provides exceptional results. In an effort to improve upon the structure of vanilla back-propagation many variants have been created, for example Delta-Bar-Delta and QuickProp, although many other variants also exist to alter the results in various ways. This paper will focus on the standard vanilla back-propagation, in which a forward pass through the network is done using the randomly initialized weights to determine the error by finding the difference of the expected and actual values. Using this error the weights are adjusted and the error is propagated backwards through the network to give each neuron its own contribution to the error, which is then used to alter more weights. This weight alteration signifies that the network is learning as it changes how it classifies supplied patterns.
Particle swarm optimization is a technique which uses a number of particles to work towards the minimization of error, which is the basis for using it as a neural network training algorithm. By creating a direct relationship between the position vector of a particle and the weights of the neural network, a harmonized structure can be created. Thus each weight of the network corresponds to one value of the particles position vector. We can then use the weights to perform the same forward pass as with standard back-propagation, however in this case use it to determine if a given pattern was classified properly. Since particle swarm optimization works towards the minimization of its error, we use the number of misclassified patterns as the fitness of a particle. As the number of misclassified patterns decreases, the network learns and classifies more patterns correctly. This hybridized structure is explained in greater detail in the 'Using Particle Swarm Optimization to Train Neural Network' section.

II. Problem

The objective of this paper is to investigate the potential for training an artificial neural network using an alternative method to the commonly preferred ones. In order to obtain a full understanding of the performance of this new training method, an analysis will be preformed on the performance of standard training methods and the newly proposed one. We will also investigate the behaviour on multiple data sets to determine if the proposed training method inflicts a change in the performance of the network compared to the normal method.

The performance of the network is broken down into three distinct categories. It is measured by the accuracy rating and error rating of the network and the computational time that is required for the training method to obtain optimal convergence levels. The accuracy rating will be a measure of how precise the network is in classifying a given number of training examples from the data set. The error rating is measured during training and is represented by the difference in the output of the network with the expected value of the training example, or in other words how close it was in classifying the example correctly. The computational time will be measured before training has started, and again after training has been successful. The difference in time will determine if one method is faster and computationally less expensive than the other.

III. Artificial Neural Networks

Artificial Neural Networks (ANN's) is a computational model that is inspired by biological neural networks, a common example being the human brain. By connecting series of neurons together, layers are constructed serving varying purposed. These layers are liked together through a series of connections that modify the amount of information that is passed through from one connection to another. Using this process, ANN's have become a powerful tool in manipulating non-linear data in order to determine if there

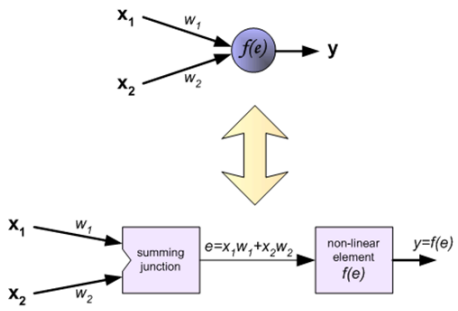

Fig. 1. Neuron Connection - The input values are modified by the weight values and summed. The total value is passed through an activation function and the output becomes an input value for the next layer.

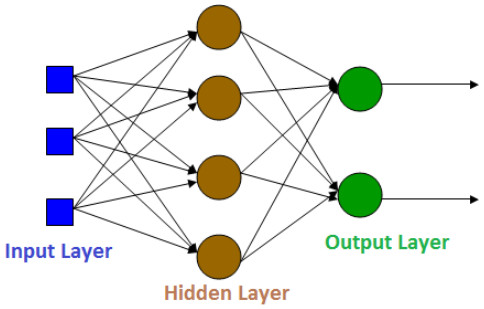

Fig. 2. Feed Forward Neural Network - Input layer is connected through a series of weighted connectioned to the hidden layer. If there are multiple hidden layers, the hidden layers connected to one another respectively. The last hidden layer is connected to the output layer.

exists any underlying relationship in the data set. It's been shown that this method of neural networks is a great tool for pattern matching. The overall object of a neural network is to be able to recognize patterns and classify certain data depending on the problem. The learning process of the network allows it to remember certain patterns and be able to classify new ones based on previous information, similar to the way we teach ourselves, for example when we learn about shapes. We are shown examples of shapes such as triangles, squares and circles and by repeatedly observation the characteristics of the shapes we are able to classify new shapes that are similar to the ones we learn.

Using this biological inspiration, it's possible to construct multiple types of networks that have different functionalities [**?**]. Of the most common is Feed Forward Neural Network [**?**]. A Feed Forward Neural Network(FFNN) is generally classified as a neural network with three different layer types. As shown in figure **??**, a typical FFNN consists of an input layer, a hidden layer and an output layer. The input layer servers as the first layer in the network and will be used to pass information through to the next layer. The number of neurons in the input layer is determined by the problem specific to the application of the network. The information is then passed through the input layer into the hidden layer and the values are modified based on the weight value for a given connection. Figure **??** shows the behaviour information passing through a neuron. If the input values are $[x_1, x_2, ..x_n]$ where n is the number of connections for the given neuron, the input of the neuron X is represented by the summation of the input values modified by the weights $[w_1, w_2, .., w_n]$ as shown in the following equation.

$$X = \sum_{i=0}^{n} w_i x_i \tag{1}$$

The calculated input is then passed into an activation, or similarly a squashing function so the values are placed within a specific range. There are a variety of different activation functions, one the most common ones being the sigmoid and hyperbolic functions as shown in equation **??** and **??** respectively.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3}$$

The hidden layer can actually have multiple layers, and in some applications the addition of multiple hidden layers increases overall performance of the network, such as convergence rate and accuracy rating.

One of the big challenges with neural networks is the learning process, how the network is supposed to learn based on training examples. A Feed Forward Neural Network is trained using a supervised training method where the outcome of the training examples are known before hand. The output of the network is compared to the expected outcome and the necessary changes are made to the weights in order to obtain the most accurate result. The most common learning method is the Back-Propagation method which is the method used in this paper.

A. *Training*

Training could be considered as the largest role in artificial neural networks. [**?**] There are various techniques on how to train neural networks. These training examples represent example cases of inputs that your network might see. The overall goal here is to show the network a number of sample cases, and tell the network how it should react (in other words, the values it should spit out in the output layer). It is the same thought process as teaching someone to solve a math problem. You can give the person a number of examples, and using these examples, the person will be able to approximate the solution of a new problem by seeing previous solutions. After training, the network should be able to properly classify new data that it has never seen before. The bonus to neural networks is that this new data could be noisy data, and it gets naturally filtered through the network and only the important information is used to make a decision.

A training example is represented by a vector with a list component and a target component. The list component represents a snapshot of a data example. Generally they are represented in the form:

$$Example = [x_1, x_2...x_n, target] \quad (4)$$

Where the component x_1,x_2...x_n is one input of a training example. Each x value is passed through it's corresponding input node in the network and the information is then modified by the weights and fed through the rest of the network. Once the information reaches the end (value of the output node) the value is compared with the target value of the training example. More explanation on how this is used to train the network is described in the section section.

There are multiple methods to train a system using the same data set, and in some cases certain methods have been known to out perform others [**?**] such as k-fold and holdout validation methods. K-fold can be advantageous over hold-out when the data set is small or not very diverse and complex. K-fold works by partitioning the data into k umber of folds and using each fold in turn as testing data using the left over partition for training. The trick with k-fold is knowing how many folds to use for the data set. This is sometimes dependent on the data itself and should be found empirically for accurate results.

The other method, and main focus in this paper is the hold-out method. The idea here is to split the data set into two separate groups - training set and testing set. Clearly, the training set will be used strictly for the training part of the neural network where as the testing set will be used to measure the performance of the network **after** is has completed training. The training set is then split up again to represent a combination of two subsets, training set and validation set. Like before, we use the training set normally to train the network, however this time we have a validation set. The validation set is only a small portion of the training set because it does not directly effect the overall training performance of the network, it's used more for ensuring the network has been properly trained. A problem can sometimes arise while training, known as overfitting. This means that the neural network not only has learned the training examples, it's actually memorized them and the accuracy of future classifications greatly declines. The validation set is used to deter the network from overfitting. After a certain number of training runs, the validation set is run on the network. Depending on the results from this validation run, the network can either continue training or stop if overfitting is happening. One of the biggest questions to ask is, how does one split up the original data set into two different ones? How much data should be keep for testing. How much for training? These values will have be found empirically and unfortunately are outside of this paper. Values that have been known to perform best on average were selected. More detail on the values in the analysis section.

B. Back-Propagation

One of the big challenges of neural networks is modifying the weights in such a way that they converge on values that effect the input enough, so that we could get our desired output by reducing the amount of errors in each. There are copious amounts of training methods for neural networds, such as Delta-Bar-Delta [**?**], QuickProp [**?**], and the standard Vanilla Back-propagation method [**?**]. For the purpose of this paper we will only be using Vanilla back-prop. Vanilla back-prop works similar to a gradient descent approach. After we feed the information forward through the network, we then start the back propagation process. In order to use back propagation, the activation function for each neuron has to be differentiable. In our case, the derivative of sigmoid function s(x) is:

$$\frac{\delta s(x)}{\delta x} = s(x)(1 - s(x)) \quad (5)$$

This value is multiplied with the current error of a neuron to get a new error representing the amount of weight delta, or in other words, the amount we need to change the weight in order to get a more accurate result.

Algorithm 1 Vanilla Back-Prop

Require: Initialize the weights in the network
  while all examples properly classified or stopping criteria met **do**
    for each example e in training set **do**
      $O \leftarrow$ neural-net-output(network, e)
      $T \leftarrow$ teacher output for e
      Calculate error $(T - O)$ at the output units
      Compute δw_h for all weights from hidden layer to output layer
      Compute δw_i for all weights from input layer to hidden layer
      Update weights in the network
    end for
  end while

The back-propagation algorithm, being a variant of gradient descent, can be geometrically represented by the error surface defined over the weight space. [**?**] At every iteration, the value of the weights are updated in the direction that minimizes the error using the following weight update process in hopes of finding the global minimum which represents the optimal weight values to achieve minimal error. The weight update is such:

$$w(t+1) = w(t) + \eta\delta\theta output \quad (6)$$

Where η is the specified learning rate of the network. δ represents the error for each layer and θ is the derivative of the activation function, most common value is output(1- output). With these values, the equation states that the weight value of the next iteration is equal to the old weight value plus this product representing the weight delta.

The Steepest Descent method works for basic examples, but if the error surface contains certain properties, this may lead to convergence problems. The reason is because of the direction and magnitude that is calculated during the iteration process. If the error surface is greatly sloped, then the magnitude that is calculated for the next step might be too large and actually leaps over the local minimum we are trying to find. Similarly the converse is true. If the slope is very low, almost a flat surface, the gradient descent method will only computer a certain large step size. If the surface is large enough, this could cause problems with convergence rate. The next algorithm we will look at addresses this issue.

IV. Particle Swarm Optimization

Particle Swarm Optimization (PSO) was originally introduced by Kennedy and Eberhart [**?**] in 1995 inspired by research in swarm behaviour by Reynolds [**?**] and Heppner and Grenader [**?**] by presenting simulations of birds flocking. PSO is designed to mimic swarm-like behaviour much like the way a flock of birds moves or a school of fish move together in order to achieve a common goal such as locating a food source. It was found that this method could be applied by PSO as a method for optimizing continuous non-linear functions. The ability to optimize non-linear functions creates a very powerful tools that can be applied to a wide range of problems.

The swarm, or sometimes called the population, consisted of a given number of entities, or particles. These particles work together in order to achieve an overall goal. The particle takes in to account it's position in search space based on some fitness value as well as the position of the particle with the best fitness value in the swarm. The fitness value is normally determined by the value of the function being optimized in the problem which is calculated by the particles position in the space. Using this methodology, the swarm will move as a unit based on minor changes to their position in the search space in order to achieve an the solution.
Each particle has a position vector in the search space and velocity vector to accelerate it through the space. If $x = [x_1, x_2, ..., x_n]$ is the position of the particle in n dimensions and $v = [v_1, v_2, ..., v_n]$ is the velocity of the particle, for every iteration of the algorithm, the velocity and position are updated in the following way:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1(p_{best} - x_i(t)) + c_2 r_2(g_{best} - x_i(t)) \tag{7}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{8}$$

In Equation **??** ω refers to an inertial constant. ω is applied to the previous velocity in an attempt to keep the particle travel in the same direction. c_1 and c_2 represent the cognitive and social components respectively. The cognitive component focuses on the particle's best position p_{best} and its current position and will influence the amount to change its position based on this value. The social component is similar however it uses the global best g_{best} particle in the swarm to guide its velocity. r_1 and r_2 are uniformly distributed vectors having values ranging between (0.0...1.0) exclusively. These random values are meant to help with the stochastic nature of algorithm by allowing the particle to explore more of the search space in hopes of finding better solutions.

The algorithm for PSO is as follows:

The algorithm can be read as follows - randomly initialize the swarm. For every particle in the swarm, calculated the fitness corresponding to that particle's position in search space then update the personal best solution. Find the particle with the best fitness out of the entire s and assign it to the global best. For every particle in the swarm, calculate the velocity using Equation **??** and update their position according to Equation **??**. These steps are repeating until the stopping criteria has been met. The stopping criteria is normally determined by a predefined number of iterations, or some minimum threshold value for the fitness to reach to be classified as converged.

Algorithm 2 Particle Swarm Optimization Algorithm

Require: Initialize the swarm randomly
  while stopping criteria not met or $t < t_{max}$ **do**
    for all particles p in swarm **do**
      evaluate p.fitness
      update p_{best}
    end for
    update g_{best}
    for all particles p in swarm **do**
      calculate new p.velocity
      update p.position
    end for
  end while

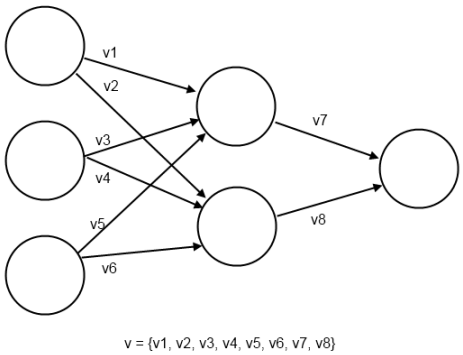

Fig. 3. Neural Network and Particle Weight vector - The weight values of each connection correspond to the particles position in search space.

V. Using Particle Swarm Optimization to Train Neural Network

The overall goal of back-propagation is to adjust the weights of the connections in the network in such a way that the system learns such that when given input examples it is able to properly classify each based on their characteristics. This means that there exists are optimal set of weights values such that for every training example, the inputs will converge on the necessary classifications. The purpose of this paper is to implement particle swarm optimization to find this optimal set of weight values. Other studies have been done to apply genetic algorithms to neural networks to construct Evolutionary Neural Network and this is where the inspiration for this paper originates [**?**].

The particle's position corresponds to the weight values of the network. If there are n weights in the network, then the magnitude of the position vector of the particle will be of size n. Figure **??** displays the relationship between the network weights and the position values of the particle.
The particle's fitness is measured by the amount of misclassified training examples. During training, the algorithm uses some of the data to validate the training process. It's in this step that the fitness of the particle is calculated. The

total number of misclassified examples becomes the particles position for that epoch. Clearly the objective is to minimize the amount of misclassified examples such that the network is deemed accurate enough, therefore the swarm will focus on minimizing this value. The particles will move together through the weight search space every time an example is passed through the network in hopes that the values will converge such that the misclassified cases are minimized. Initially one would expect the number of misclassification's to be large, and as the network trains, the number should reduce as it begins to properly classify more and more training examples.

In order to include particle swarm optimization with back-prop, some slight alterations had to be made to the algorithm. The algorithm is as follows:

Algorithm 3 SwarmProp

Require: Initialize the network and swarm
  while all examples properly classified or $t < t_{max}$ **do**
    for each example e in training set **do**
      for every particle p in swarm **do**
        assign p.position to network weights
        $O \leftarrow$ neural-net-output(network, e, p)
        $T \leftarrow$ teacher output for e
        Calculate error $E = (T - O)$ at the output units
        Calculate number of misclassified examples n
        assign p.fitness = n
      end for
      iterate swarm
    end for
  end while

As shown above, this algorithm is very similar to the back-prop algorithm described earlier. Instead of simply calculating the output for every training example, we now loop through the swarm and use each particles position as the weight values in the network. Now the forward pass is applied to the network and the corresponding error is calculated using the output. Next the number of misclassified examples is calculated by feeding through the validation data and recording the number of negative classifications it made. This value is used as the fitness value of the particle. Once the fitness is calculated for the entire swarm, an iteration is done in the PSO algorithm and the swarm moves one position in search space. This process repeats until a certain number of iterations or the network has properly converged.

VI. Data and Samples

The data used in this paper is divided into two different sets, a rather simple, small data set and a much large more complex data set. The reason behind choosing two different sets of data is to aid in the analysis of performance of implementing PSO as a training method. There may be some correlation between different data sets and the performance of the learning algorithm, more is described in the analysis section. The data used in training the neural network can be found in the Machine Learning Repository [**?**].

A. Wine Data

The wine data was chosen because of the simple nature of the data set. There are a total of 3 different classifications depending on the type of wine, class 1 has 40 samples, class 2 has 49 samples and class 3 consists of 33 examples, giving a total of 122 training examples that will be used to train the network.

Since some of the attributes in an example contain a value greater than 1 (some of them are much larger), a scale factor for each attribute is used in order to shrink the value ranging from 0 to 1 and also to normalize the data. In this experiment, we simply implement a linear scale of each attribute. It works as follows:

- We first determine the largest value contained for each attribute vector, store this value as the scale factor.
- When we are building the training examples from the instances, we divide each attribute value by its corresponding scale factor.

This ensures that all attributes are ranged from 0 to 1, as this is required by the sigmoid activation function.

An alternative scaling method is to use the actual sigmoid function. We input the original value for each attribute from the data set to the sigmoid function then use the output as our training data. The drawback for this method is that when the attributes value is high, the output of the sigmoid is close to 0, so some accuracy might be lost.

B. Hand-Written Digit Recognition Data

The second data set chosen is a hand-written digit recognition set. Supplied by the United States Postal Service, series of samples were taken of hand written digits and transformed into training examples for each corresponding digit 0 through 9. The complexity of this data set is significantly higher than the previous wine data. There are 700 training examples for each digit, resulting in a total of 7000 training examples for the network in total. There are 400 testing examples for each digit, 4000 in total. Each example represents an 8x8 pixel square of a written digit. The list component of the training example vector represents the grey-scale value of each pixel(a value between 0.0 to 1.0). The target is represented as the integer value defined by the 64 pixel greyscale values.

VII. Analysis

The analysis done in this paper focuses mainly on the implementation of particle swarm optimization as a learning method for neural networks. In order to have a full understanding of the performance, analysis was done using Vanilla Back-Propagation training method and the newly proposed Particle Swarm training method. The three characteristics that were considered in this paper for comparing the performance and accuracy of the two algorithms are the error rate, validation rate and computational time.

The error rate of the network is represented by the difference between the expected output of the network given by the

training example and the predicted value of the network. During training, the network is considered to be learning when there has been a noticeable decrease in error rate. From this, one would expect the error rate to be substantially lower at the end of training.
The validation rate represents the amount of properly classified training examples that are tested on the network during the training phase. Initially one would the expect the network to perform poorly and thus have a low validation rate. As the network learns and is trained on the examples, naturally the validation rate should increase as the error decreases indicating there has been an increased in proper classification.
The computational time is a reflection of the time it takes for the network to train. The following two tables show the parameter values that were used during experimentation.

Neural Network Parameters		
Digit Recog. Data	0.7	Learning Rate α
	70/30	Hold-out
	18	Number of Hidden Nodes
Wine Data	0.7	Learning Rate α
	70/30	Hold-out
	5	Number of Hidden Nodes

Particle Swarm Parameters		
Digit Recog. Data	100	Swarm Size
	0.729844	Inertia ω
	1.496180	Components c_1 , c_2
	$R[0..1]$	Random Values r_1 , r_2
Wine Data	50	Swarm Size
	0.729844	Inertia ω
	1.496180	Components c_1 , c_2
	$R[0..1]$	Random Values r_1 , r_2

A study was done in 2001 on the parameters of particle swarm optimization and the values of ω, c_1 and c_2 were chosen accordingly [**?**]. For all sets of data the holdout validation method was used with 70% of the data sample used as training data and the remaining 30% used for testing. Batch training was used so that the network has time to obverse all types of examples before changes to the weights are made. This allows for a more accurate fitness value for the particle swarm. The weight values of the network represented by the particle's position in search space were initially randomized between the values of -1.0 and 1.0 . These values serve as the bounds of the particle's position in the space.

VIII. Results

The following graphs are displayed in pairs in or to fully understand the effect PSO has on training compared to vanilla back-prop. Figure **??** and **??** represent the error rate of the network during training using the wine data. It's clear from the graphs that over time, as the number of epochs increase, the error rate decreases leading to more accurate results of the network.

Figures **??** and **??** show the validation rate on the wine data set. It's clear from both these graphs that the validation rate is increasing as the network is learning. This behaviour combined with the reduction in error rate strongly suggests that the network is learning the data and properly classifying the validation data.

Figure **??** and **??** shows the error rate of the network using the hand-written digit recognition data. Similar to the previous error rates for the wine data, we see the error rate gradually being reducing as training is taking place. It's worth noting that in Figure **??** the error of the network appears to reach a global minimum, then as training continues the error rate is becoming worse. This has to do with overfitting the network with training examples. When the network is overfit, it no longer becomes a problem of general classification, but now the network is memorizing each training example, which will make it harder to classify new examples later on that is has never seen before. It would be wise to stop training at this global minimum to obtain optimal network performance.

Figures **??** and **??** represent the validation rate of the network during training. It's clear from the graphs that both vanilla backprop and swarmProp training methods are increasing the rate at which the examples are being properly classified.

Figures **??** and **??** represent the fitness of the global best particle during training. It's clear from the graphs that the fitness is significantly declining, which indicates the swarm is properly reducing their fitness the more the network is training. This is another indication that the network is converging and learning properly.

Based on the graphs for both sets of data, it's observed that the convergence rate using swarmProp happens faster than vanilla back-prop. Referring to Figure **??** and **??** it appears as though back-prop takes approximately 100 epochs to converge where as particle swarm took approximately 35

epochs to reach its optimal error rate.

Computation Time		
Back-prop		
Data	Time	Iteration
Wine	00:00:00.1740071	100
Digit Recog	00:03:51.9793623	200
PSO		
Data	Time	Iteration
Wine	00:00:10.09155615	40
Digit Recog	01:26:00.14775350	50
Time Differences		
Data	Time	Δ Time
Wine	00:00:00.1740071 00:00:10.09155615	00:00:09.91754905
Digit Recog	00:03:51.9793623 01:26:00.14775350	01:22:48.16064000

The above table displays the total computation time it took for the network to be properly trained. There is a noticeable difference in the computation time between the two data sets, this reflects the diverse complexity of the wine data compared to the digit recognition data. With the incorporation of particle swarm, training time for the wine data was increased by a value of $00:00:09.91754905$, or 9.918 seconds. For the digit recognition data, training took $01:22:48.16064$ longer, equivalently 1 hour, 22 minutes and 48.161 seconds.

IX. Conclusion

Based on the results from the analysis, implementing particle swarm optimization as a training method for neural networks is possible and could be a feasible replacement as an alternative algorithm. One trade-off however is the increase in computational time. The network appeared to train in fewer number of epochs, but at the price of higher complexity. There are many ways to implement particle swarm with neural networks, and this paper simply addressed one method, it may be worth investigating alternative methods. There exists many different parameter settings when combining PSO with ANN however many were outside the scope of this paper and therefore were not fostered. Further research can be extended to determine the optimal parameter settings for both the particle swarm and neural network. There are many parameters in neural networks that can be studied, such as the learning rate, training process and topology. Similarly for particle swarm there exists a number of parameters, such as the size of the swarm which may help with convergence, as well as possible topology types that traverse information differently throughout the swarm.

References

[1] G. K. Jha, "Artificial neural networks and its applications."
[2] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," 1987.
[3] T. M. Mitchell, *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.
[4] F. Scott E, "An empirical study of learning speed in back-propagation networks," 1988.
[5] R. G. U. Aberdeen, "The back-propgation algorithm," Feb 2013.
[6] R. E. James Kennedy, "Particle swarm optimization," *Neural Networks, 1995. Proceedings., IEEE International Conference*, 1995.
[7] C. Reynolds, "Flocks, herds and schools: a distributed behavioral model," *Computer Graphics*, 1987.
[8] "A stochastic nonlinear model for coordinated bird flocks," 1990.
[9] L. D. David J. Montana, "Training feedforward neural networks using genetic algorithms," *Neural Networks, 1995. Proceedings., IEEE International Conference*, 1995.
[10] K. Bache and M. Lichman, "Uci machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml
[11] F. van den Bergh, "An analysis of particle swarm optimizers. phd thesis, university of pretoria, south africa," 2001.

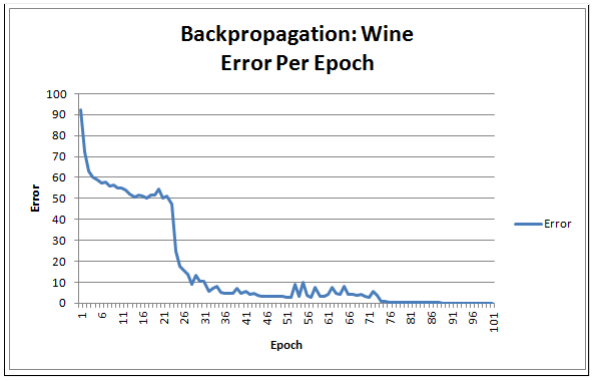

Fig. 4. Error rate per epoch for Vanilla BackProp on wine data set.

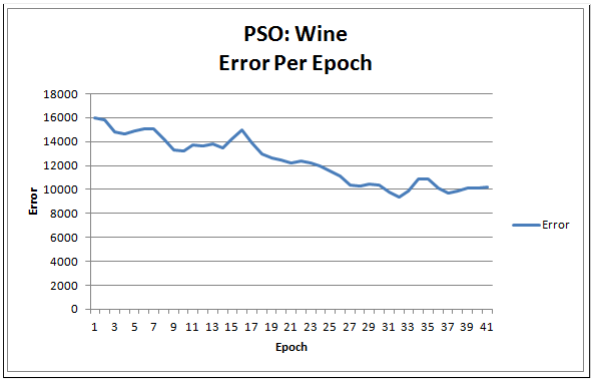

Fig. 5. Error rate per epoch for SwarmProp on wine data set.

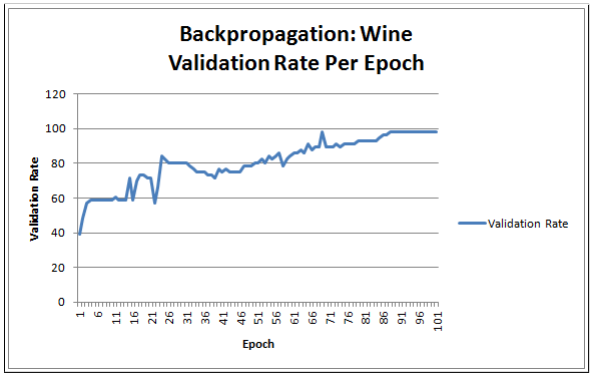

Fig. 6. Validation rate per epoch for Vanilla BackProp on wine data set.

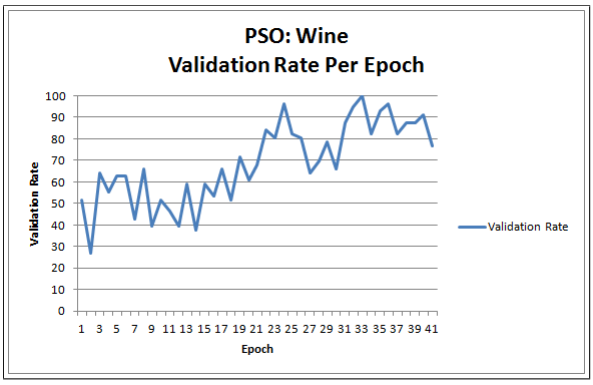

Fig. 7. Validation rate per epoch for SwarmProp on wine data set.

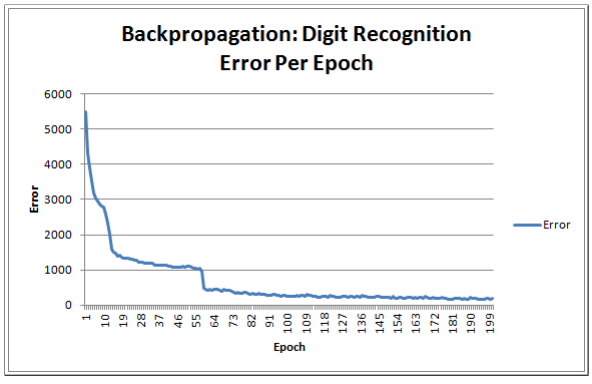

Fig. 8. Error rate per epoch for Vanilla BackProp on hand-written digit recognition data set.

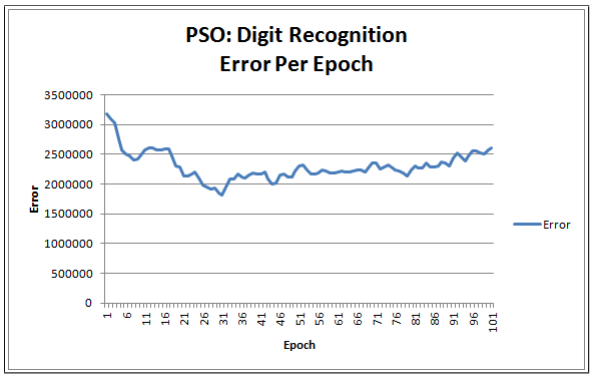

Fig. 9. Error rate per epoch for SwarmProp on hand-written digit recognition data set.

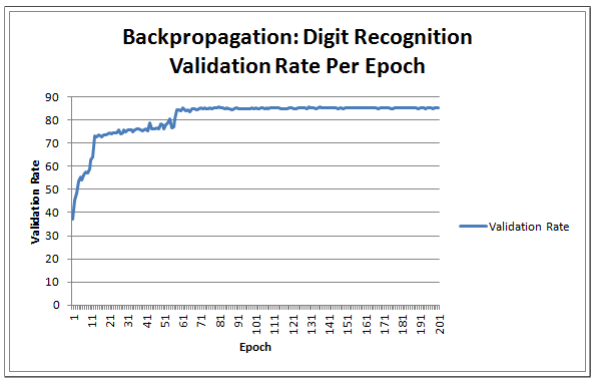

Fig. 10. Validation rate per epoch for Vanilla BackProp on hand-written digit recognition data set.

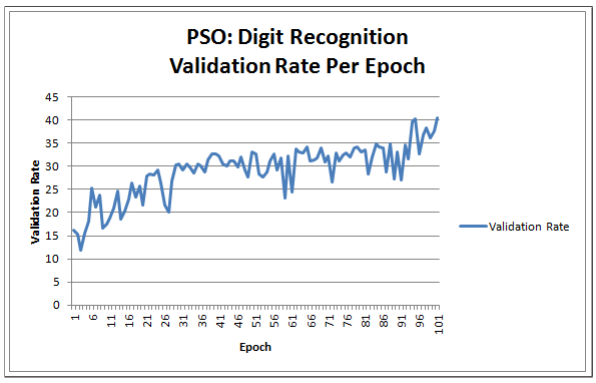

Fig. 11. Validation rate per epoch for SwarmProp on hand-written digit recognition data set.

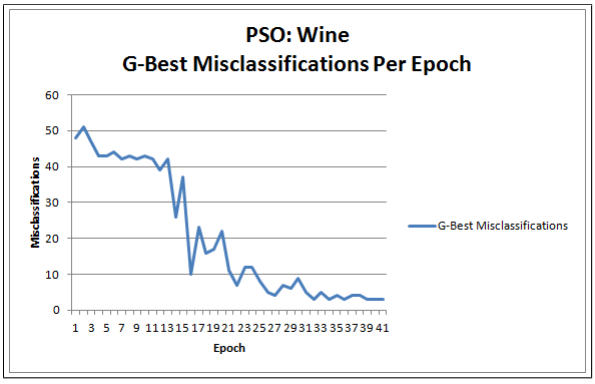

Fig. 12. Fitness value corresponding to the global best particle for wine data.

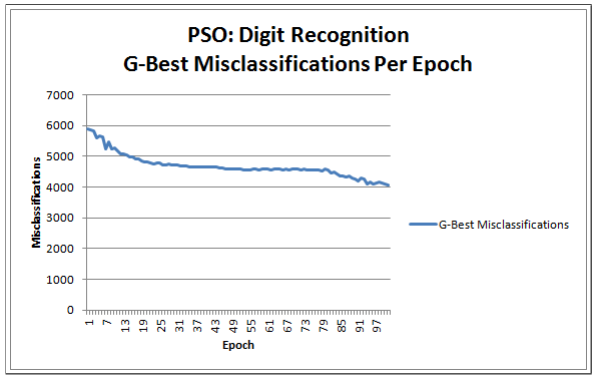

Fig. 13. Fitness value corresponding to the global best particle of the swarm for the digit recognition data set.