Examining Metaheuristic Performance for Network Training

***Abstract*—This paper examines the use of metaheuristic search strategies for training a neural network in lieu of backpropagation. It conducts a comparative study between two metaheuristics—particle swarm optimization and genetic algorithms—and a gradient based technique using backpropagation. The suitability of metaheuristics in training a neural network is also discussed.**

Keywords—neural network, backpropagation, gradient descent, genetic algorithms, particle swarm optimization, metaheuristic, training, machine learning.

I. Introduction

Machine learning is a discipline of computer science which focuses on artificial intelligence algorithms which employ statistical models and algorithms which can be used to process some form of data—oftentimes a solution to a problem may employ machine learning as is the case with classifications or recognition problems.

Neural networks, as a branch under the umbrella study of machine learning, is one such method used to approach classification and recognition problems [7]. As a supervised learning model, a neural network can become trained to recognize and classify novel data.

These models are often trained using a technique which takes advantage of an error gradient [7]—the gradient between what is expected and what is found from the known information about input data and the output of the network. The most common approach here is using *backpropagation*. However, there are alternatives to this methodology.

Metaheuristics are "informed searches" [8] of a problem which strive to find solutions to a problem based on the continuous space in which solutions reside. How this relates to neural networks is that as a neural network is trained, its network weights are optimized in a way to minimize error: if you consider a neural network as simply a set of synapse weights, the search space of which is continuous, a metaheuristic can be employed to find optimal synapse weights without the use of an error gradient [3].

There is merit in comparing backpropagation to metaheuristic approaches to network training as the entire training architecture and process is distinctly different. Backpropagation relies upon differential calculus for implementation and has many pitfalls to use, such as exploding or vanishing gradients when contrastingly, metaheuristics are typically devoid of these shortcomings. To this end, there is some import on the study of metaheuristics for training neural networks as any improvement in efficiency or implementation is welcome.

II. Network Training

A neural network is a sequence of layers consisting of neurons [7]. Neurons contain connections between each other in a forward direction between layers with neurons in the input layer being connected to neurons in the first hidden layer and so on until the output layer is reached.

Data is fed into the network at the input layer and through some training mechanism, the network weights are adjusted such that they incrementally better understand the input data [7].

Pseudocode for a networking training procedure agnostic of training methodology is below:

```
procedure train-network() is:
  generate randomly uniform network weights
  for each training epoch do:
    training-method()
  endfor
endprocedure
```

The training mechanism is important insofar synapse weights can be adjusted based on calculus or some other method. For example, backpropagation considers the output received from a forward pass of data throughout the network versus an expected output to generate an error gradient. Metaheuristics may use other methods which are often bioinspired or less mathematically robust.

Rather than reinforcing learning by continual input and comparison between output, metaheuristics instead take control of synapse weights directly and perform a local-global search within the confines of the search space [3].

When considering training, many of these methods will suffice in finding a better solution than randomly uniform weights but the degree to which a better solution is found is subjective and based upon the algorithm used.

This paper intends to explore the differences between backpropagation and two metaheuristics—genetic algorithms and particle swarm optimization—by way of comparing convergence speed and training time; both of these performance measures can allow each training method to be analyzed in a different way with consideration for speed, accuracy, and scalability.

III. Training Algorithms

Three different training techniques were implemented: online stochastic gradient descent backpropagation, genetic algorithm, and particle swarm optimization. Their implementations are detailed within this section.

A. Backpropagation

Backpropagation takes advantage of the fact an error gradient exists when training the network [7]. Using differential calculus, backpropagation computes the gradient of the loss or error function with respect to the synapse weights of the network. Through continued training, attempts are made to find the global optima of this function.

Pseudocode for the backpropagation algorithm is as below:

```
procedure bp() is:
  for each training example t do:
    feedforward t through layers 0..n
    compute error e at layer n
    backpropagate e through layers n..0
    update weights w using e and net params
  endfor
endprocedure
```

In the feedforward stage, an activation function is needed to aggregate inputs from each synapse weight and in the backpropagation stage, the derivative of this function is also needed to propel the error backwards through the network.

B. Genetic Algorithm

Genetic algorithms take cues from evolutionary biology to evolve a solution to a problem [8]. Within the context of neural network training, the problem is synapse weight optimization. A population of candidate solutions (chromosomes) consisting of network weights (genes) is generated and portions of the population are discarded and then repopulated based on the best fit chromosomes.

Pseudocode for the genetic algorithm is as below:

```
procedure ga() is:
  sort population p by fitness
  elites := best from p
  tournament := random k from p
  tournament-winner := best from tournament
  mating-pool := elites + tournament-winner
  while p is not full do:
    parent-a := random from mating-pool
    parent-b := random from mating-pool
    children := crossover(parent-a, parent-b)
    mutate(children)
    add children to p
  endwhile
endprocedure
```

Chromosomes are some encoding of the network's weights: if a network's synapse weights were considered as a point in n-space, a chromosome is simply a position, with each weight corresponding to a different axis [6]. Adjacent genes are adjacent weights in the network (either sharing a neuron with the previous and/or next weight; or preceding or succeeding the first or last weight of a neuron).

The genetic algorithm implements two genetic operators: crossover and mutation. Crossover is the recombination of parent chromosomes to produce child chromosomes and comes in many flavors: here used is two-point crossover, where two pivot points are chosen within the length of the chromosome. Genes within this continuous segment come from one parent while genes outside that range come from the other parent.

Two-point crossover was empirically found to provide more fruitful results than one-point (where child chromosomes are some combination of a parent's "head" and the other parent's "tail") and uniform (where child chromosomes are a random combination of each parent).

Adjacent genes can be assumed to be at least partially correlated [5]: as each chromosome can be viewed as a position in n-space, all genes have equal impart to the overall chromosome's position. Two-point crossover breaks the sequence of correlation twice to introduce a likewise correlated sequence of genes. The effect on overall chromosome correlation is affected to a random degree, whereas with one-point or uniform crossover, the effect can be large or very large respectively.

Mutation is the process of perturbing a gene or genes within some random range: this is the exploratory factor of a genetic algorithm, where new genetic material is added to the population. While crossover recombines parents, it does not introduce diversity of genetic material, mutation does.

The mutation operator used is gaussian mutation around a single gene. Were a gene to be mutated, it is randomly perturbed with a range described by a normal distribution where the mean of the distribution is at the gene being mutated and the standard deviation is some value based on the fitness of the chromosome.

This operator perturbs a gene within a set of probabilities: depending on the fitness of the chromosome, a gene has a higher chance to be perturbed slightly and a lower change to be perturbed greatly. This mutation operator uses scheduling based on the mean squared error: for earlier mutations, there is a higher likelihood of a large perturbation which promotes early global exploration, when in later generations, this chance for this slims, allowing for more local exploration.

When sorting the population of chromosomes, the mean squared error is found for each possible set of network weights. As mean squared error is a function of each training example being fed through the network, the feedforward step as described when using backpropagation also applies here, including the use of an activation function. The difference, however, is there is no activation function derivative.

Previously mentioned was a mating pool being devised: this pool consists of potential parents for generating child chromosomes. The selection strategy used to populate the mating pool uses elitism and a tournament.

For elitism, the best chromosomes in a generation are brought into the next generation verbatim (to avoid regressing to a lesser fitness) but also enter the mating pool. This ensures children are partially comprised of the best fit chromosomes.

A tournament is made which selects k random chromosomes in the population and a winner is decided as the most fit chromosome in the tournament. This chromosome also enters the mating pool. This allows for less fit chromosomes to become parts of child chromosomes.

Crossover, mutation, elite and tournament proportions are all parameterized: there is no "one size fits all" approach; depending on the problem, these parameters are balanced to provide generally good if not great results.

Over time and using the genetic operators as previously defined, chromosomes tend to be better fit solutions to the problem of optimizing the network weights.

C. Particle Swarm Optimization

Particle swarm optimization, not dissimilar to genetic algorithms, use a bioinspired method to train the network.

Rather than evolution, however, it takes cues from the social behavior of flocking birds [8].

Within the context of neural network training, candidate solutions (particles) are a position in n-space [2]: each particle correlates to an encoding of the network synapse weight. Like genetic algorithms, each axis for a particle position is a set of network weights.

A collection (swarm) of particles is generated and based on the fitness of individual particles as well as the swarm, particles tend to drift towards a global optimum within the search space.

Pseudocode for the particle swarm optimization algorithm is as below

```
procedure pso() is:
  s.best := find-swarm-best()
  for each particle p in swarm s do:
    for each axis a of p do:
      wei := w * p.vel
      cog := c1 * r1 * (p.best[a] – p.pos[a])
      soc := c2 * r2 * (s.best[a] – p.pos[a])
      new-vel[a] := wei + cog + soc
      new-pos[a] := p.pos[a] + new-vel[a]
    endfor
    p.pos := new-pos
    p.vel := new-vel
    if fitness(p) < p.fit do:
      p.fit := fitness(p)
      p.best := p.pos
    endif
  endfor
endprocedure
```

Unlike genetic algorithms, particle swarm optimization has no operators on which to improve solution fitness: it is purely by the position of the particle and particle swarm parameters.

Particle swarm optimization requires three parameters [8]: inertial weight (ω), a cognitive coefficient (c_1), and a social coefficient (c_2).

Inertial weight considers how resistant to change a particle has in velocity: a higher inertial weight takes into consideration the existing velocity of the particle more than a new velocity, whereas a lower inertial weight will consider more of the new velocity in initializing a new position.

Often inertial weight is scheduled to increase or decrease based on the current epoch or some other criterion: instead, the implementation uses a constant inertial weight [4].

The cognitive and social coefficients consider how much a particle's personal best position reflects in the new position versus the swarm's best position. Without these coefficients, a particle would continue in one direction ad infinitum.

The mean squared error is again the loss function and is interfaced whenever a particle's fitness is evaluated. This function yet again constitutes a feedforward step over all training examples.

Over enough iterations, particles tend to congregate around a global optimum (mean squared error is reduced), thus being a better weight initialization for the network.

IV. Algorithm Parameters

Each network training method uses various parameters are previously described. This section will describe each parameter and network structure within the context of the data tested.

A. Data Sets

Four data sets are used to test the efficacy of each training method for optimizing network weights:

TABLE I. DATA SETS

Name	*Instances*	*Attributes*	*Classes*
Iris	151	4	3
Wheat Seeds	211	7	3
Wine	178	13	3
Breast Cancer	570	31	2

Table I. Data sets used for testing efficacy of training.

The intention of these sets is to examine training performance in different problem sizes. The dimensionality of the problem is related to the number of weights within the network which by extension is a function of the topology of the network.

B. Network Structure and Dimensionality

The network structure remains consistent between training methods as there is a desire to compare the efficiency of each training method on training a specific network. To this end, the number of hidden layers and hidden layer size remains constant.

From the network structure, dimensionality of the problem can be found. Each layer outside of the output layer has one bias neuron. The dimensionality can be found using the below:

$$Dim = \big(HL * (IL + 1)\big) + (OL * (HL + 1))$$

Below is a summary of network structures per data set as well as the problem dimensionality:

TABLE II. NETWORK STRUCTURES AND DIMENSIONS

Data	*IL Neurons*	*HL Neurons*	*OL Neurons*	*Dim.*
Iris	4	3	3	27
Wheat Seeds	7	5	3	58
Wine	13	6	3	105
Breast Cancer	31	8	2	274

Table II. Network structure and dimensionality per data set.

The input and output layers are dependent upon the data (input layer size is the number of attributes; output layer size is the number of classes) and not user defined. Considering this, these also remain constant.

C. Backpropagation Parameters

As previously mentioned, backpropagation requires two hyperparameters to learn: learning rate and momentum rate. These parameters control how much of a training example's output error should be considered when adjusting network weights and how much the previous error should reflect in the update change (the inertia from the previous weight update).

Both parameters depend on the problem being optimized and therefore there is no pair of parameters that works for every problem. For each network structure, parameters were adjusted to find a combination which allowed for speedy convergence:

TABLE III. BACKPROPAGATION HYPERPARAMETERS

Data	*Learning Rate*	*Momentum Rate*
Iris	0.100	0.001
Wheat Seeds	0.100	0.001
Wine	0.100	0.002
Breast Cancer	0.100	0.003

Table III. Learning and momentum rates per data set.

While a learning rate of 0.100 was found to perform well for every test, the momentum rate increased slightly for the two largest tests to offset the higher dimensional data. Due to the problem space being more complex, the system was more apt to fall into local optimum and a higher momentum rate was needed to compensate.

With regards to the activation function, the logistic sigmoid function was chosen:

$$f(z) = \frac{1}{1 + e^{-z}}$$

$$f'(z) = f(z)(1 - f(z))$$

This activation function is used as it converged quickly and in testing did not suffer from exploding or vanishing gradients or dead neurons.

D. Genetic Algorithm Parameters

The implementation of genetic algorithm training uses five parameters: crossover and mutation rates, elite and tournament proportions, and a base sigma. The population size used is 100.

Crossover and mutation rates consider how frequently each operator is used whereas elite and tournament proportions consider how much of the population is considered elite or as entrant into the tournament respectively.

Mutation perturbs a gene within a normal distribution with the mean as the gene being mutated and standard deviation as some value which decreases as the mean squared error of a chromosome decreases. To this end there is a base sigma as minimum standard deviation for the normal distribution.

Each genetic algorithm parameter is tabled below:

TABLE IV. GENETIC ALGORITHM PARAMETERS

Data	*CR*	*MR*	*EP*	*TP*	*Base Sigma*
Iris	0.90	0.03	0.05	0.03	0.5
Wheat Seeds	0.90	0.04	0.05	0.04	0.6
Wine	0.90	0.05	0.05	0.05	0.7
Breast Cancer	0.90	0.05	0.05	0.07	0.8

Table IV. Genetic parameters per data set.

Since calculating the mean squared error for each chromosome requires the use of a feedforward method, an activation function is needed. The leaky rectified linear unit is used:

$$f(z) = \max(z, 0.01 * z)$$

This activation function was chosen as it showed high performance in minimizing mean squared error and was chosen over logistic sigmoid function as many of the drawbacks to rectified linear unit activation functions (vanishing/exploding gradients) do not apply when using non-gradient based training techniques.

E. Particle Swarm Parameters

The particle swarm optimization algorithm is parameterized with four parameters: inertial weight, cognitive and social coefficients, and a boundary. The swarm size used is 100.

As previously mentioned, the inertial weight considers how much of a particle's prior velocity reflects in its new position and the coefficients consider how much of a particle or swarm's best position reflect in the new position.

An assumption is made that optimized network weights are relatively small in magnitude: a boundary is needed when particles become too far from the origin. If a particle's position lays outside of a bounded region around the origin, its fitness is not updated. This allows particles to generally remain within the bounded region as there is no better fit solution outside of this region.

The particle swarm parameters are tabled below:

TABLE V. PARTICLE SWARM PARAMETERS

Data	*ω*	*C_1*	*C_2*	*Boundary*
Iris	0.5	1.5	1.2	3
Wheat Seeds	0.6	1.3	1.1	5
Wine	0.3	1.6	1.4	7
Breast Cancer	0.4	1.4	1.1	5

Table V. Particle swarm parameters per data set.

Like with the genetic training algorithm, the leaky rectified linear unit activation function is again chosen as it reflected in great performance and the training technique does not suffer from reasons this activation function might be avoided in other networks.

V. EXPERIMENTAL METHODOLOGY

The experiments performed were measurements of the mean squared error over each training epochs as well as the average number of epochs to reach a termination condition. Lastly, the time it took to train also needs to be examined to consider questions of scalability.

A. Termination Condition and Error Function

The overall accuracy of the network can be quantified by using the classification error which is simply a measure of how many correct outputs were found using the known classification of the training and testing data.

However, classification error is not a viable fitness function upon which to train using our metaheuristics. Classification

error is related to mean squared error such that a lower mean squared error generally results in a reduced classification error, but a smaller classification error does not always mean a lower mean squared error.

Consider the below figure which is of a network's output layer and the output from a training example:

FIGURE I. CLASSIFICATION THRESHOLD

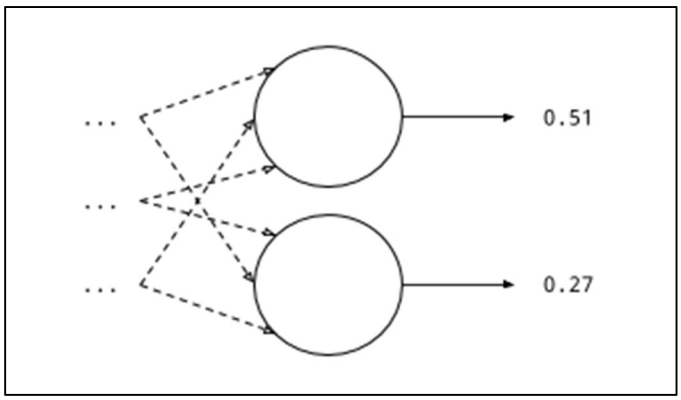


Figure I. An example of network outputs.

Note that the first output neuron—corresponding to one of the classes of the data—is *0.51*, whereas the second is *0.27*. Since the network will consider this example as belonging to the first class, since the first class had the highest neural response, the system is not particularly confident about this decision.

The network effectively outputs a set of *probabilities* that an instance belongs to a specific class, with the highest probability winning. For this example, this may allow classification error to be low, but the mean squared error may not: the system is not confident, and this reflects in the mean squared error.

To best train the network, instead the mean squared error is used as metaheuristic fitness function. Using this fitness function allows the network to be more confident in its classification decisions and rather than see neural outputs like *0.51*, it will be closer to *1.00*.

In simple terms, using classification error as fitness function will allow networks to emerge that optimize weights such that a correct classification is at least *0.51* for that neural output whereas mean squared error will optimize further and try to maximize that neural output.

The termination condition used will be reaching a mean squared error equal to or less than *0.10*. For all data sets, this was found to correlate to 95+% accuracy against training data and 90+% accuracy against testing data.

In situations where the termination condition was not reached, a maximum of one hundred epochs is used as it can be assumed if a network does not converge within one hundred epochs, it may not converge at all.

B. Holdout Ratio

In order to optimize a network to classify well on novel data, the input data needs to be split such that it is trained on only a portion of data. This is called the holdout ratio.

All input data uses the same holdout ratio of 70% training data and 30% testing data: this provides a big enough sample size to train upon but also to test how well the network can classify data it has not yet seen.

C. Test Structure

Every experiment is performed multiple times to gain statistical significance. Since each network training contains a stochastic element, no two runs are identical.

To this end, a sample of fifty runs is conducted and the best thirty runs are taken and average to produce graphs and tables. The best thirty runs are chosen as there are occasions where the network may perform poorly, and the test results should be indicative of a generally good result.

VI. RESULTS

Every test has a series of runs performed according to the previously defined test structure. Plots are then generated using the mean between tests.

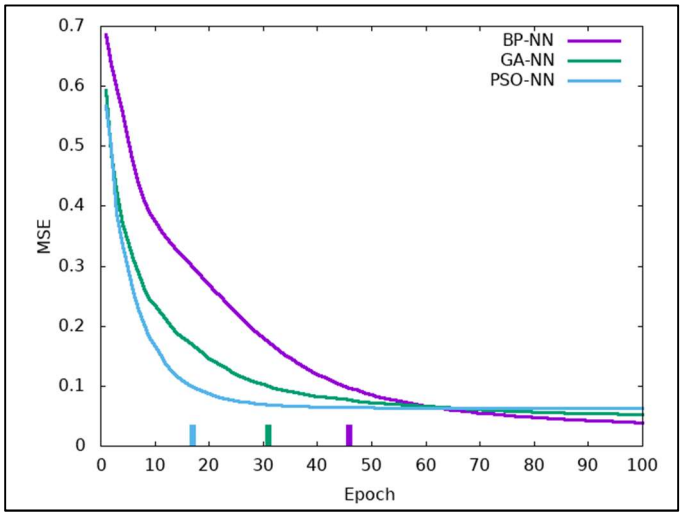
The comparison between each network is directly examined where "BP-NN" indicates performance using backpropagation, "GA-NN" for genetic algorithm training, and "PSO-NN" for particle swarm training.

Note for plots to follow, each curve indicates the mean squared error over epochs while a "tick" at the bottom of each plot denotes when the termination condition was met.

A. Iris

Using the Iris data set and network parameters as defined previously, a plot of mean squared error over time is generated to compare the difference between network training methods.

PLOT I. IRIS DATA TEST



Plot I. Network performances using Iris data.

A One-Way ANOVA test is performed to identify if there is any significance to these results. This test's null hypothesis is such that there is no significant difference between population means and therefore no clear stochastic dominance between samples.

In the context of these results, accepting the null hypothesis would suggest there is no difference between using one network training method over another; contrastingly, rejecting the null hypothesis would mean that one or more training methods outperforms the others.

After performing the ANOVA test, an F-score of 8.253 is found as well as a p-value less than 0.001, dramatically below the null hypothesis threshold of a p-value less than 0.05. This test concludes that there is statistical relevance to using one training method over another for this network.

The number of epochs it took to reach the termination condition is also examined:

TABLE VI. IRIS CONVERGENCE SPEED

BP-NN			*GA-NN*			*PSO-NN*		
Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
39	57	46	12	57	26	11	35	20

Table VI. Green denotes best, orange worst.

From the above, there is a consensus that both metaheuristic training methods outperform backpropagation with particle swarm optimization being the preferred training method.

As well, the training time can be examined:

TABLE VII. IRIS TRAINING SPEED

BP-NN	*GA-NN*	*PSO-NN*
1.14s	4.01s	1.52s

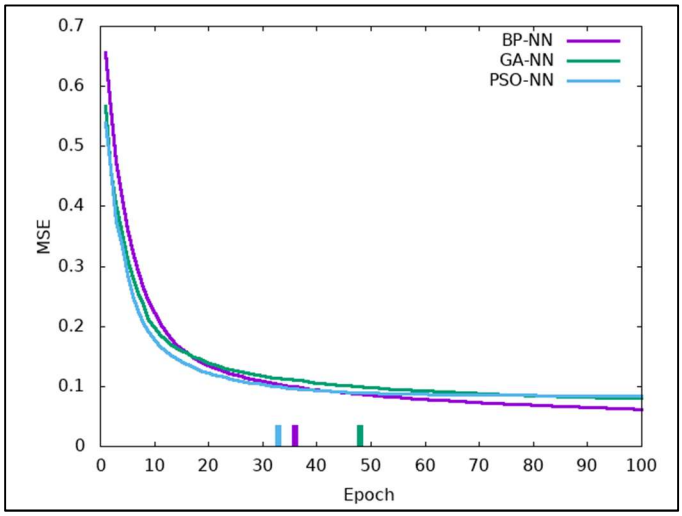
Table VII. Green denotes best, orange worst.

This table illustrates the average time it took to reach the termination condition in computational time. Here, backpropagation is much faster than the genetic algorithm but only slightly quicker than particle swarm.

B. Wheat Seeds

A similar experiment is performed using the Wheat Seeds data set. Again, a plot of mean squared error over time is generated:

PLOT II. WHEAT SEEDS DATA TEST



Plot II. Network performances using Wheat Seeds data.

The One-Way ANOVA test yields an F-score of 0.811 with a p-value of 0.446, significantly higher than the null hypothesis p-value of 0.05. This means there is no significance choosing between training methods.

The speed of convergence can be measured by examining how many epochs it took to reach the termination condition:

TABLE VIII. WHEAT SEEDS CONVERGENCE SPEED

BP-NN			*GA-NN*			*PSO-NN*		
Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
20	47	37	17	89	50	15	55	32

Table VIII. Green denotes best, orange worst.

From the above, it is found that particle swarm optimization outperforms both other methods and best-case performance for both metaheuristics outperforms backpropagation. However, due to the lack of statistical significance, it cannot be said that any one method dominates another.

The training time can also be examined:

TABLE IX. WHEAT SEEDS TRAINING SPEED

BP-NN	*GA-NN*	*PSO-NN*
1.67s	37.40s	5.90s

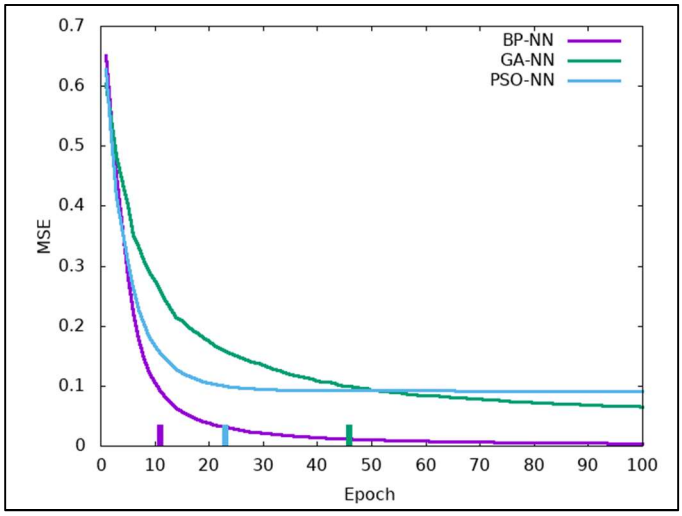
Table IX. Green denotes best, orange worst.

Analyzing the average time for a training method to reach the termination condition, backpropagation outperforms both other methods—in the case of genetic algorithms, backpropagation vastly outclasses.

C. Wine

The experiment is repeated using another different data set, the Wine set. Another plot of mean squared error over epochs is made:

PLOT III. WINE DATA TEST



Plot III. Network performances using Wine data.

Using a One-Way ANOVA test, an F-score of 27.934 is found alongside a p-value of less than 0.00001, showing strong significance. The null hypothesis is rejected: there is one training methodology that dominates the others.

Speed of convergence, as before, can be seen by looking at the number of epochs it took for a given training method to reach the termination condition:

TABLE X. WINE CONVERGENCE SPEED

BP-NN			*GA-NN*			*PSO-NN*		
Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
9	14	11	26	82	54	15	72	28

Table X. Green denotes best, orange worst.

At this point, the higher dimensional data begins to favor backpropagation as there is evidence backpropagation dominates the other methods. Genetic algorithms again show the worst performance.

Average training time can also be examined:

TABLE XI. WINE TRAINING SPEED

BP-NN	*GA-NN*	*PSO-NN*
2.03s	37.19s	11.28s

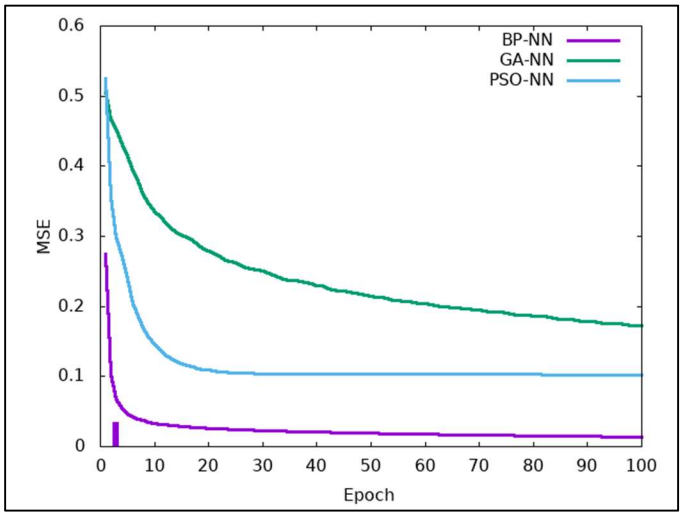
Table XI. Green denotes best, orange worst.

Interestingly, training using this data set showed a decrease in training time for genetic algorithms when compared to the lower dimensional Wheat Seeds data; however, both metaheuristics are much slower than backpropagation due to the higher dimensional data.

D. Breast Cancer

As before, the final experiment is conducted using another data set, the Breast Cancer data set. A plot of mean squared error over time is generated:

PLOT IV. BREAST CANCER DATA TEST



Plot IV. Network performances using Breast Cancer data.

Above it is shown that backpropagation converges very quickly—likely due to the larger number of training instances—while both metaheuristics failed to converge within 100 epochs.

If the plot isn't obvious in the best performer among training methods, another One-Way ANOVA test is performed to gauge significance of these results. A very large F-score of 351.882 is found with a p-value less than 0.00001, suggesting there is strong significance to these results.

Like before, data on speed of convergence is also collected:

TABLE XII. BREAST CANCER CONVERGENCE SPEED

BP-NN			*GA-NN*			*PSO-NN*		
Min	*Max*	*Avg*	*Min*	*Max*	*Avg*	*Min*	*Max*	*Avg*
2	2	2	DNC	DNC	DNC	16	DNC	DNC

Table XII. Green denotes best, orange worst.

Here a "DNC" denotes "did not converge", as in the training algorithm was unable to reach the termination condition within 100 epochs. Of note is that while particle swarm optimization can converge (and quite quickly), there were not enough successful runs of the sample of fifty runs to shift the average close enough.

For this performance measure, backpropagation vastly outclasses both other methods, always converging within two epochs.

Lastly, the training time can be examined:

TABLE XIII. BREAST CANCER TRAINING SPEED

BP-NN	*GA-NN*	*PSO-NN*
9.32s	485.63s	164.42s

Table XIII. Green denotes best, orange worst.

As expected with much higher dimensional data, backpropagation is heavily favored, with particle swarm optimization being 17 times slower and genetic algorithms being 49 times slower.

VII. RESULTS ANALYSIS

On interpreting these results, the rank of each can be examined within the context of convergence speed, mean convergence speed, and mean training speed.

TABLE XIV. TRAINING METHOD RANKING

Measure		*BP*	*GA*	*PSO*
Iris	Best Convergence Speed	3	2	1
	Mean Convergence Speed	3	2	1
	Training Time	1	3	2
Wheat Seeds	Best Convergence Speed	3	2	1
	Mean Convergence Speed	2	3	1
	Training Time	1	3	2
Wine	Best Convergence Speed	1	3	2
	Mean Convergence Speed	1	3	2
	Training Time	1	3	2
Breast Cancer	Best Convergence Speed	1	3	1
	Mean Convergence Speed	1	3	3
	Training Time	1	3	2
Average		1.58	2.75	1.67

Table XIV. Ranking each training technique per test measure.

Upon examining the collected data, backpropagation is the preferred method for network training given the data sets tested upon; however, particle swarm optimization does not fall far behind it in convergence speed and length of training.

Contrastingly, genetic algorithms were found wholly inefficient or impractical in every performance measure.

For smaller data sets, like Iris and Wheat Seeds, particle swarm optimization shows promise in training the network. However, these are smaller data sets and issues of scalability need to be considered. When looking at length of training, backpropagation is dominant for every problem size tested putting into question the practicality of using particle swarms to train larger networks.

There is one benefit to genetic algorithms for network training, however: examining each plot, particle swarm optimization does converge to some optimum but then stagnates for further epochs. Genetic algorithms, contrastingly, seem to improve even beyond 100 epochs. Without a termination condition of a limit to the number of epochs, it is expected that a genetic algorithm will continue to find a smaller mean squared error, at the very least beyond the point whereupon particle swarm or backpropagation might stagnate.

These results only illustrate that backpropagation is a good "go-to approach" which can almost assuredly find a global optimum given correct hyperparameters which is agnostic of problem size and dimensionality. The same cannot be said of either metaheuristic tested. Of import to note is that backpropagation is not uniformly superior to the other methods, it is just an "easy" approach which guarantees better results with more frequency and consistency.

Despite this, there is merit to study of metaheuristics as in several of the tests conducted in this report show favorable results in convergence speed and mean convergence speed. Closing the gap for practicality, however, is the lengthened training time. In other words, metaheuristic training may be best suited for smaller problem sizes where the time difference is nearly imperceptible.

VIII. Conclusion

There is merit to studying the use of bioinspired metaheuristics in training neural networks: many of the advantages include speedier convergence or ability to minimize the error/loss function in fewer steps. Were training time not a factor, any metaheuristic can successfully train a network given appropriate metaheuristic parameters [3].

Upon examining both genetic algorithms and particle swarm optimization as training strategies, the benefits are apparent for each for smaller data sets. As the dimensionality of the problem increases—and despite good parameter selection—performance does not compare to a gradient based approach.

Of special note is particle swarm optimization excels for smaller problem spaces, converging quicker by some order of magnitude. To compare to genetic algorithms, it is a very viable alternative to backpropagation. Genetic algorithms, however, do not seem practical or efficient for network training irrespective of problem size: by being vastly outclassed by particle swarm optimization and to a greater degree with backpropagation, the usefulness of genetic algorithms for this purpose is considered low.

The tests conducted herein do not show an "out of the box" performance measure and instead use parameters which were found by some experimentation. There is a limit, however, to expert decided parameter selection. A potential improvement to this study is use a metaheuristic to find metaheuristic parameters as performance can always be improved using different parameter choices; the limitless permutations of parameter and parameter pairings is infeasible to test with any practicality and an algorithm to find idealized parameters can improve performance even further.

References

[1] Gudise, V.G. & Venayagamoorthy, Ganesh. (2003). Comparison of Particle Swarm Optimization and Backpropagation as Training Algorithms for Neural Networks. Proceedings of the 2003 IEEE Svarm Intelligence Symposium. 110 - 117. 10.1109/SIS.2003.1202255.

[2] Zajmi, Leke & Ahmed, Falah.Y.H.Ahmed & Jaharadak, Adam Amril. (2018). Concepts, Methods, and Performances of Particle Swarm Optimization, Backpropagation, and Neural Networks. Applied Computational Intelligence and Soft Computing. 2018. 1-7. 10.1155/2018/9547212.

[3] Khan, Koffka & Ashok, Sahai. (2012). A comparison of BA, GA, PSO, BP and LM for training feed forward neural networks in e-learning context. International Journal of Intelligent Systems and Applications. 4. 10.5815/ijisa.2012.07.03.

[4] Rathore, Ankush & Sharma, Harish. (2017). Review on Inertia Weight Strategies for Particle Swarm Optimization. 10.1007/978-981-10-3325-4_9.

[5] Adeli, Hung, et. al, Machine Learning: Neural Networks, Genetic Algorithms, and Fuzzy Systems. 1995.

[6] Wu, Chen-Phon & Tseng, Ching-Shiow. (1995). Function approximation neural network with genetic training algorithms. 16. 373-381.

[7] T. M. Mitchell, Machine learning. New York: McGraw-Hill, 1997.

[8] S. Russell & P. Norvig, Artificial Intelligence: A Modern Approach, 3rd Edition. Prentice Hall, 2010.