Metaheuristic Training Neural Networks

April 20th, 2021

Abstract

This paper explores the use of metaheuristics in training a feed-forward artificial neural network, examining each algorithm and their performances in training a neural network versus backpropagation. Results show promise for the use of certain metaheuristics although the go-to approach of backpropagation is still superior in many ways.

1 Introduction

Neural networks are a bio-inspired tool under the broader class of machine learning algorithms. It consists of a sequence of layers, themselves containing nodes—called neurons. The biology analogy is the brain where this technique takes its namesake.

These can be used for many tasks, although the classical example is that of classification: using a neural network to train a model which can then be used as a classifier for novel data.

One of the most popular approaches to training a neural network is using backpropagation (**BP**), which is a gradient-based training mechanism which relies on the existence of an error gradient between a target output value and the actual output value.

However, BP has several caveats: the problem it aims to solve must be linearly differentiable as it relies on differential calculus, the possibility of exploding or vanishing gradients is non-trivial, and while mathematically robust, is conceptually abstract.

Oftentimes neural networks are referred to as black-boxes, as while the inputs and outputs are known, the inner workings of training can be obtuse. Thus while BP is an adequate training mechanism, alternatives do exist which can be simpler to conceptualize.

2 Metaheuristics

Metaheuristics are a type of local-global search that is more informed than a random search. If a random search looked for solutions stochastically, metaheuristics instead consider variables like the search space and candidate solutions to improve solutions to ideally reach a more fit solution [1].

These search algorithms are often population based, where many solutions exist and the search is propelled by characteristics of the solutions—as a whole or individually. As well, metaheuristics are oftentimes accompanied by metaphor, commonly nature inspired, which governs the rules by which a solution can improve.

Typically metaheuristics are used for optimization problems, where an objective function has an unknown minimum or maximum. Metaheuristics can be deployed to optimize this function to find a global optima.

One such application is for neural network training, where the search space is known and is continuous, but the dimensionality of the problem is often large and therefore a guided search is needed to optimize network weights.

As previously mentioned, BP is a common method to train a neural network but there is merit to the study of metaheuristics for this application as well. Metaheuristics offer a simpler alternative, not only in implementation but also simpler conceptually thanks to the metaphors ascribed to them.

2.1 Genetic Algorithm

Genetic algorithm (**GA**) takes cues from evolutionary biology to evolve solutions to a problem based on both population and individual agent traits [3].

In GA, a population of solutions is generated at random, encoded as chromosomes which consist of genes. Every generation, a set of chromosomes are chosen through some selection strategy, and then two genetic operators are applied to create a new population: *crossover* and *mutation*.

The selection strategy is used to create a mating pool, as the metaphor is related to evolutionary biology. Two selection strategies are elitism (where the best fit chromosomes are transferred to the mating pool and new population verbatim) and tournaments (where a random subset of the population is compared and the most fit joins the mating pool).

From the mating pool, two parent chromosomes are chosen and from them the genetic operators are applied to generate two child chromosomes, filling the new population.

Crossover is the process of recombination between parent chromosomes, one such flavor is two-point crossover, where a contiguous sequence of genes is one parent is swapped with the same sequence in the other.

$$\begin{aligned} \text{parent}_1 &= \{x_1, x_2, x_3, x_4, x_5\} \\ \text{parent}_2 &= \{y_1, y_2, y_3, y_4, y_5\} \\ \text{child}_1 &= \{x_1, x_2, y_3, y_4, x_5\} \\ \text{child}_2 &= \{y_1, y_2, x_3, x_4, y_5\} \end{aligned}$$

A contiguous sequence of genes from $i = [3, 4]$ was swapped between both parent chromosomes and they are returned as child chromosomes. Crossover allows novel chromosomes to emerge and allow for exploration of the search space early and more local exploitation later.

To contrast, mutation is the process of randomly changing a chromosome's genes. For example, a Gaussian mutation operator will perturb a gene and set it to a random value around the gene as mean and some standard deviation. Mutation is an exploratory factor since novel genes enter chromosomes which otherwise might not exist in the population.

GA is parameterized with four parameters: *crossover rate*, which denotes how frequently crossover is to occur, *mutation rate*, which is the same for mutation, and then *elite proportion* and *tournament proportion*, each denoting the percent of the population chosen for elitism and tournaments respectively.

Due to elitism, solutions in the population cannot regress and instead either stagnate or continue to improve.

2.2 Particle Swarm Optimization

Particle swarm optimization (**PSO**) is inspired by the flocking psychology of birds or other swarming animals [5].

For PSO, a population of solutions is generated in the form of particles. Commonly this is referred to as a swarm. Particles are some encoding of a position and velocity which is used for particle movement: for each generation, the particles move according to arithmetic rules which factor in the swarm best fitness and a particle's historic best fitness.

Three components are needed for updating position and velocity of a particle:

$$\begin{aligned} w &= \omega v_i(t) \\ c &= c_1 r_1 (y_i(t) - x_i(t)) \\ s &= c_2 r_2 (\hat{y}_i(t) - x_i(t)) \end{aligned}$$

In order, these are an inertial component, cognitive component, and social component. v_i is the velocity of the particle in the ith axis, y and $\hat{y}$ are the particle and swarm best positions, and x is the particle's current position. r_1, r_2 are stochastic multiplicands.

PSO is taken with three parameters: ω, or the *inertial weight*, which determines the resistance to change in velocity of a particle, c_1, or the *cognitive coeffieicne*, dictates how much import a particle's historic best position influences its next position, and c_2, or the *social coefficient*, dictates how much import is instead placed on the swarm best position in updating a particle's position.

Particles are then moved using the values for w, c, and s:

$$\begin{aligned} v_i(t+1) &= w + c + s \\ p_i(t+1) &= p_i(t) + v_i(t+1) \end{aligned}$$

Every generation, every particle is moved according to these rules, and over time, particles seek out the global optima. A larger cognitive coefficient allows for more local exploitation whereas a larger social coefficient allows for more exploration of the search space.

Since particles tend to move closer to the swarm best position, and provided adequate parameters are chosen, particles seldom regress to a worse fitness. Through movement, the swarm tends to a better solution over time.

2.3 Differential Evolution

Not too dissimilar to GA, differential evolution (**DE**) is another evolution-based meta-heuristic but instead of inspired by biology, evolves solutions based on arithmetic and combining solutions according to some mathematical rules [6].

For DE, a population of solutions is generated randomly and are moved through the search space using arithmetic rules which allow combination of solutions. DE uses two parameters: *crossover rate*, which determines the frequency in which solutions are combined, and *differential weight*, which is a multiplicand determining how much of the difference between two crossovered solutions is imparted to the new solution.

While GA considers two parent chromosomes to generate two child chromosomes, DE contrastingly considers three "parent" solutions, combined in some ratio, to form a single new "child" solution. DE does not offer similar selection strategies like elitism as in GA, and instead a new population is guaranteed to be unique between generations.

Solutions evolve like so: for every solution x in the population, three other solutions are chosen, a, b, c such that x, a, b, c are distinct entities from each other. A solution y is generated according to the below:

$$y_i = \begin{cases} a_i + F \times (b_i - c_i) & \text{if } r_i < CR \text{ or } i = R \\ x_i & \text{otherwise} \end{cases}$$

y is generated in sequence of axes $i = 1, 2, \ldots, n$. For every solution that is created this way, a random index $R \in \{1, 2, \ldots, n\}$ is chosen which signifies an axis which is always updated, even if crossover is not to occur. For this equation, CR is the crossover rate and F is the differential weight.

Once y is found, it is compared to x, and if it has a better fitness, then x is replaced with y in the population and the process continues for other agents.

This type of evolution is not strictly bio-inspired and instead is more mathematical in nature. Crossover for DE allows the metaheuristic to have an exploratory factor, where the search space is explored in large steps.

DE still exhibits an exploitative factor, however, as when the population converges to an optima, all solutions within the population more closer resemble each other, meaning exploration is minute around agents.

Over time, the best fitness of solutions in the population is improved until convergence (either to a global optimum or local optima). Every generation is assured to either remain the same or improve since a fitness comparison is the criteria for a solution being replaced. In other words, solutions will never regress to a worse fitness.

2.4 Bat Algorithm

Bat algorithm (**BA**) is similar to other metaheuristics in that it is a swarm-based approach to improving solutions. Solutions within the search space are updated similar to how swarming microbats descend onto prey or food [7].

Like with the other metaheuristics, BA creates a population of solutions—here called bats—as encodings of positions and velocities in n-space. Their position is updated according to some rules, not unlike PSO.

BA considers four parameters: *loudness*, which denotes how close to prey (optima) they are, *pulse rate*, which will determine how quickly a bat converges to prey, then values α and γ which is a loudness decreasing factor and pulse rate increasing factor respectively. As bats get closer to prey, their loudness will decay, but will converge quicker, thus their pulse rates must increase.

Bats move according to these rules:

$$v_i(t+1) = v_i(t) + f_i(p_i(t) - p_*)$$
$$p_i(t+1) = p_i(t) + v_i(t+1)$$

Where v_i is the velocity of the bat at position p_i. p_* is the best position of bats in the population, and f_i is some frequency. Frequency is determined on a per-bat basis as $f_i = U(f_{min}, f_{max})$, where f_{min} and f_{max} are the minimum and maximum frequencies possible. U being picking uniformly random from within this range.

If a bat is sufficiently close to prey, it will perform a random walk around the current position. Since loudness will decrease the closer a bat gets to its prey, a random walk is performed if $U(0, 1) \geq A_i$, where A_i is the loudness of the bat. This chance increases the closer a bat gets to its target.

Bat loudness will decay at a rate of $A_i(t+1) = \alpha A_i(t)$ and its pulse rate will increase according to $r_i(t+1) = r_i^0(1 - \exp(-\gamma t))$, where t is a value denoting which generation is current.

With its general traversal of bats, BA exhibits an explorative factor, and with a random walk to fine tune position if close enough to target, an exploitative factor as well. Since bats update their position to only better solutions, it is said BA cannot regress to a worse fitness over time, much like the other metaheuristics mentioned previously.

3 Network Training

The layers between neural network layers are composed of synaptic weights: these weights are mathematical links between neurons and signify how strong a signal between neurons is. In the simplest sense, a synaptic weight is some multiplicand between neurons, altering the signal between them: a higher weight amplifies the signal while a lower weight weakens.

Training patterns are inputted into the network at the first layer, travel to subsequent layers through a *feedforward* process, then when in the output layer, the error is calculated as a function of target and actual values. For BP, the error signal is backpropagated through the network in reverse, adjusting the synaptic weights as needed so the model more closely resembles the training pattern.

Given enough training patterns and enough repetitions of the process, the network model is able to classify not only previous data fed into it, but also novel data from the same data set.

You can abstract the process of training a neural network to an optimization problem: finding a network model A such that $E(A) < E(B)$, where E is some error function and B is any other network model. The goal is to minimize the error of the network. While BP can achieve this, metaheuristics can as well.

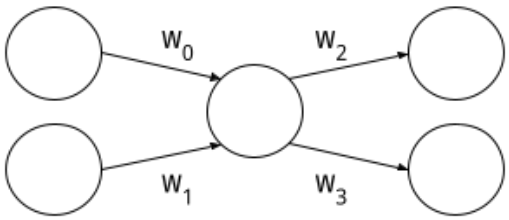

Figure 1: Connections between layers can be enumerated as a set of weights.

In the previous figure, a simple 2-1-2 network is shown and each synaptic weight connecting each layer is enumerated. For this network, which presents a 4-dimensional problem space, it has four weights.

Since metaheuristics explore the search space, it's useful to think of solutions as a position in n-space, where n is the dimensionality of the problem. For this example network, solutions update themselves in a 4-dimensional space.

Metaheuristics can take literal sense of this concept—bat algorithm having bat agents with a specific position, as example—or can be more abstract or have no close conceptual similarity to an agent "moving". Nevertheless, they all function in an identical fashion.

Since synaptic weights are enumerable in this way, it is simple to decode a network as instead a position in n-space. For this reason, a network can be be represented as a solution for a metaheuristic.

To use an example, a GA chromosome can be created as some decoding of the network synaptic weights. A network of weights $[w_0, w_1, ..., w_n]$ can be taken as a chromosome of genes $[x_0, x_1, ..., x_n]$, where $x_i = w_i$ for $i = 0, 1, \ldots, n$. Likewise, this decoding is reversible: for this example, a chromosome can be reinitialized or encoded as a network again.

This translation from solution to network and vice-versa is integral to the interconnectivity between a network and its metaheuristic training mechanism, as a metaheuristic cannot optimize a network weights directly.

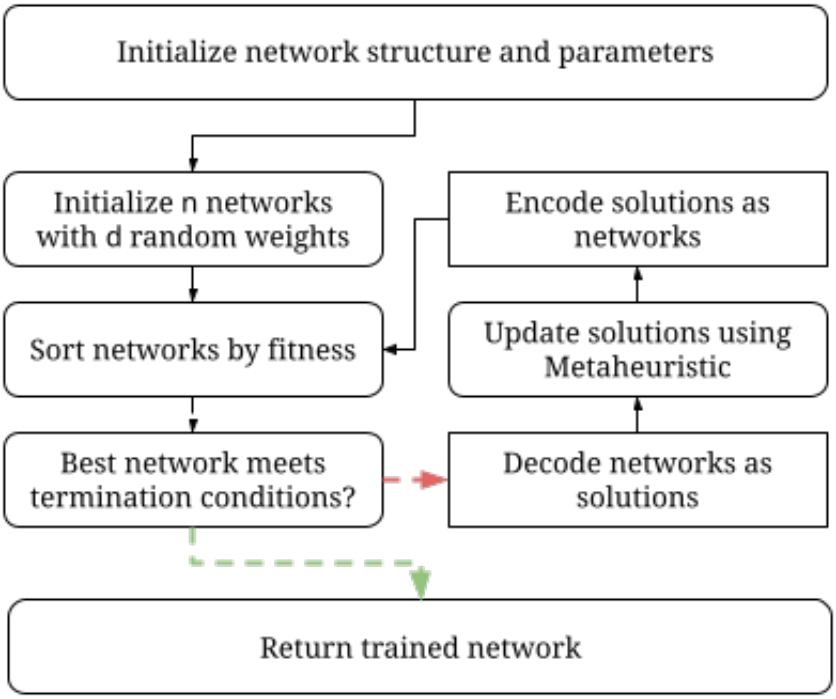

Figure 2: Metaheuristic adapted for training a neural network.

In the previous figure, it is shown how a metaheuristic is integrated within a neural network's training process. For this figure, d is the dimensionality of the network—how many synaptic weights exist in the network—and n is the number of agents for use with the metaheuristic.

Since the metaheuristics tested are population based, this method of training is analogous to training multiple networks laterally at once. However, many of the performance measures are only performed on the best solution of the population.

4 Experimental Setup

To evaluate the efficacy of metaheuristics in training a neural network, a standard feed-forward network was implemented alongside each training mechanism: backpropagation (**BP-NN**), genetic algorithm (**GA-NN**), particle swarm optimization (**PSO-NN**), differential evolution (**DE-NN**), and bat algorithm (**BA-NN**).

Further, a testing suite was devised to train a model using each training mechanism and collect the results in the form of error over time curves. The error function used is mean squared error (**MSE**).

This suite performs each training 100 times for each data set and then discards the worst 20% of runs, as the mean of runs should be indicative of generally good results. A poor training attempt would normally be discarded and here is no different. Runs are then aggregated and the mean is found and plotted.

Training ceases when one of two conditions are met: either the MSE of the network has reached a value of 0.1 or below, or the maximum number of epochs has elapsed. In the case where convergence did not occur without the defined number of epochs, that training technique is determined to have not converged.

Consideration has to be given for network and training parameters which vary by data set. Additionally, the data sets themselves need to be examined.

4.1 Data

Six data sets were used: *Iris* [2] for classification of species of iris flowers, *Penguins* [4] for classification of penguin species, *Wheat Seeds* [2] for classification of wheat varieties, *Wine* [2] for classification of wine varieties, *Breast Cancer* [2] for classification of breast cancer diagnosis, and *Ionosphere Radar* [2]. These data sets were chosen as they increase in complexity at a generally consistent rate, allowing for scalability of metaheuristic training to be considered.

Rows in the data set are comprised of input patterns which are separated into attributes and classifications. The network structured initialized for a given data set is a consequence of these variables: the input layer contains a number of neurons equal to the number of attributes of the data, and the output layer contains a number of neurons equal to the number of distinct possible classifications of the data.

Data Set	**Instances**	**Attributes**	**Classes**
Iris	151	4	3
Penguins	333	6	3
Wheat Seeds	211	7	3
Wine	178	13	3
Breast Cancer	570	31	2
Ionosphere Radar	352	34	2

Table 1: Data sets used for results collection.

While the size of input and output layers of the network is decided by the data, the hidden layer size is arbitrary but still chosen through some experimentation. A smaller hidden layer will train faster, but too small will result in poor results. To this end, smaller hidden layer sizes were preferred if it did not negative impact performance.

Data Set	**IL Size**	**HL Size**	**OL Size**	**Dimensionality**
Iris	4	3	3	27
Penguins	6	4	3	43
Wheat Seeds	7	5	3	58
Wine	13	6	3	105
Breast Cancer	31	8	2	274
Ionosphere Radar	34	10	2	372

Table 2: Dimensionality of each problem.

In the previous table, the dimensionality of each classification problem is found as a function of network topology. Declaratively, the dimensionality of a network problem is found using the below equation:

$$d = (HL \times (IL + 1)) + (OL \times (HL + 1))$$

Dimensionality is related to how "difficult" the classification problem is: while data can be intrinsically complex, dimensionality gives a general idea of how complicated it will be to train a network using that data.

4.2 Parameters

Each network is initialized with a few parameters, some of which are specific to each metaheuristic used. Many parameters remain consistent between training methods, but many parameters are initialized on a per-problem basis.

4.2.1 Consistent Parameters

Parameters related to network topology and the training process are consistent between data sets and metaheuristics. Three parameters, the number of epochs, the holdout ratio, and the initial weight range are fixed for all problems.

Epochs	**Holdout Ratio**	**Initial Weight Range**
100	0.70	[-0.50, 0.50]

Table 3: Constant network parameters.

The three constant network parameters are tabulated. Epochs is the amount of training iterations performed and is one of the termination conditions for training, the holdout ratio is the proportion of input data to be used for training the network versus testing the network, and the initial weight range is the range between minimum and maximum values for weight initialization.

Likewise, hidden layer size is a constant, but only between training methods: this parameter varies only by data set used.

Data Set	**Hidden Layer Size**
Iris	3
Penguins	4
Wheat Seeds	5
Wine	6
Breast Cancer	8
Ionosphere Radar	10

Table 4: Hidden layer size per data set.

The hidden layer size is tabulated per data set. Since the difference between BP and metaheuristics is being examined, this needs to remain equal as all training methods are effectively training identical networks.

4.2.2 BP-NN

BP relies on two parameters: learning rate, or the amount of correction a synaptic weight should take from the training instance, and momentum rate, which considers some portion of the prior training instance when updating using the current training instance.

Data Set	LR	MR
Iris	0.100	0.001
Penguins	0.100	0.001
Wheat Seeds	0.100	0.001
Wine	0.100	0.002
Breast Cancer	0.100	0.003
Ionosphere Radar	0.100	0.002

Table 5: Backpropagation parameters.

BP-NN parameters are tabulated per data set, where LR, MR are learning rate and momentum rate respectively.

4.2.3 GA-NN

For the first metaheuristic, GA considers six parameters for training. They are the population size (how many chromosomes), crossover rate, mutation rate, elite proportion, tournament proportion, and a base mutation value.

Data Set	Population	CR	MR	E_p	T_p	Base
Iris	100	0.90	0.03	0.05	0.03	0.5
Penguins	100	0.90	0.03	0.05	0.03	0.5
Wheat Seeds	100	0.90	0.04	0.05	0.04	0.6
Wine	100	0.90	0.05	0.05	0.05	0.7
Breast Cancer	100	0.90	0.05	0.05	0.07	0.8
Ionosphere Radar	100	0.90	0.06	0.05	0.09	0.9

Table 6: Genetic algorithm parameters.

GA-NN parameters are tabulated per data set, where CR, MR, E_p, T_p are the crossover rates, mutation rates, elite proportions, and tournament proportions respectively. Base is a value for use in the Gaussian mutation operator where a standard deviation is needed to take values from a Gaussian distribution.

4.2.4 PSO-NN

PSO has five different parameters controlling its movement: population size, inertial weight, cognitive and social coefficients, and a boundary variable.

Data Set	Population	ω	c_1	c_2	Boundary
Iris	100	0.5	1.5	1.2	3
Penguins	100	0.5	1.4	1.3	4
Wheat Seeds	100	0.6	1.3	1.1	5
Wine	100	0.3	1.6	1.4	7
Breast Cancer	100	0.4	1.4	1.1	7
Ionosphere Radar	100	0.3	1.3	1.3	9

Table 7: Particle swarm optimization parameters.

These parameters are tabulated per data set, where ω is the inertial weight and c_1, c_2 are the cognitive and social coefficients respectively. Boundary is the edge of the search space: were a particle to go outside the bounds, that axis is randomized to be within the boundary.

4.2.5 DE-NN

DE only has three parameters: population size, crossover rate, and differential weight.

Data Set	Population	CR	F
Iris	50	0.90	0.25
Penguins	50	0.90	0.35
Wheat Seeds	50	0.90	0.25
Wine	50	0.90	0.20
Breast Cancer	50	0.90	0.15
Ionosphere Radar	50	0.90	0.10

Table 8: Differential evolution parameters.

Above are the DE-NN parameters tabulated, where CR is the crossover rate and F is the differential weight.

4.2.6 BA-NN

Lastly, BA has six parameters: population size, frequency minimum and maximum, a boundary, a loudness decreasing factor, and a pulse rate increasing factor. Like with PSO-NN, the boundary is used to keep bats within the search space.

Data Set	Population	F_{min}	F_{max}	Boundary	α	γ
Iris	100	0	2	3	0.90	0.90
Penguins	100	0	2	4	0.90	0.90
Wheat Seeds	100	0	2	5	0.90	0.90
Wine	100	0	2	7	0.90	0.90
Breast Cancer	100	0	2	7	0.90	0.90
Ionosphere Radar	100	0	2	9	0.90	0.90

Table 9: Bat algorithm parameters.

Here tabulated, the BA-NN parameters F_{min}, F_{max} are the frequency minimum and maximum values and α, γ are the loudness decreasing factor and pulse rate increasing factor respectively.

5 Experimental Results

Every network is trained according to the previously defined test structures. Plots are then generated allowing a visual aid to compare. Two statistical tests are also performed on the data: an ANOVA test and Tukey's HSD test.

The ANOVA test is to determine if multiple samples come from the same distribution; in other words, the ANOVA test is used for testing purposes to identify if all training techniques are the same or there is non-trivial differences between them. An ANOVA test produces two statistics that are relevant: an F-score, where an $F \geq 1.0$ denotes

confidence in the test, and a p-value, where a $p < 0.05$ suggests there is significance in choosing some training techniques over the others.

Tukey's HSD test, if an ANOVA test shows significance, will then identify where outliers exist by doing pairwise comparison of means and variances between samples. If better training methods exist, they will become evident more directly using this test. This test can identify if pairs of samples come from the same or different distributions.

Plots show a curve for each training method, where each curve is the MSE over time. At the bottom of each plot is a tickmark which denotes at what epoch, on average, the training method reached a termination condition.

5.1 Iris

Using the Iris data set and network parameters as defined previously for each training method, the below plot is found:

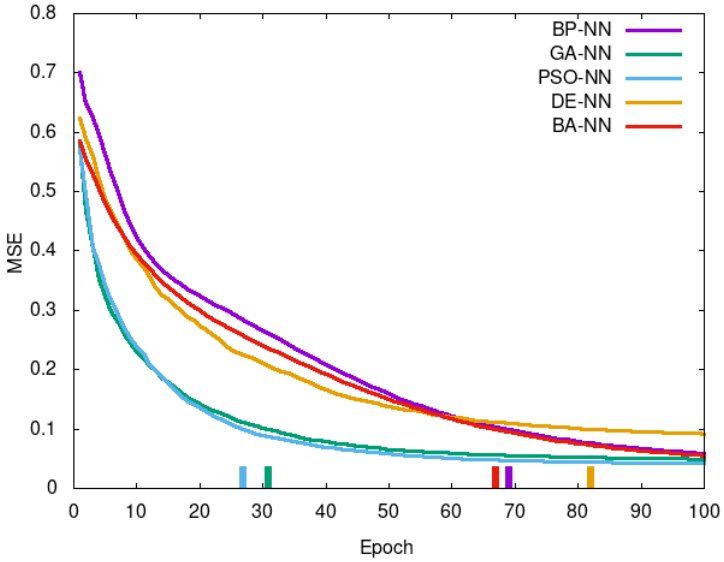

Figure 3: Iris test results for metaheuristics and backpropagation.

Visually, two training techniques (PSO-NN and GA-NN) converged to a termination condition much faster than the other three.

To determine if this difference is of statistic importance, an ANOVA test is run using each curve as a sample. This test yields an $F = 17.65$ and a $p < 0.001$ which suggests there is high confidence that there are some training methods that outperform the others.

Tukey's HSD test is then performed showing no significance between GA-NN and PSO-NN. Likewise it shows no significance between BP-NN, DE-NN, and BA-NN. However, samples between each grouping does show significance. Since Tukey's HSD test showed significance between two groups, it can be concluded that GA-NN and PSO-NN outperform the other training methods.

	BP-NN	**GA-NN**	**PSO-NN**	**DE-NN**	**BA-NN**
Min	51	9	10	36	31
Avg	70	31	27	82	67

Table 10: Minimum and average time to terminate for Iris data set.

This is the minimum and mean number of epochs it took to reach a termination condition, green denoting best, and red worst. Here, BP-NN was the slowest network to converge, but on average, DE-NN was the slowest to converge. Agreeing with our results from before, both GA-NN and PSO-NN outperform the others significantly, having the fastest training at once and on average.

5.2 Penguins

The Penguins data set is then trained upon using network parameters as defined previously for each training method, producing the below plot:

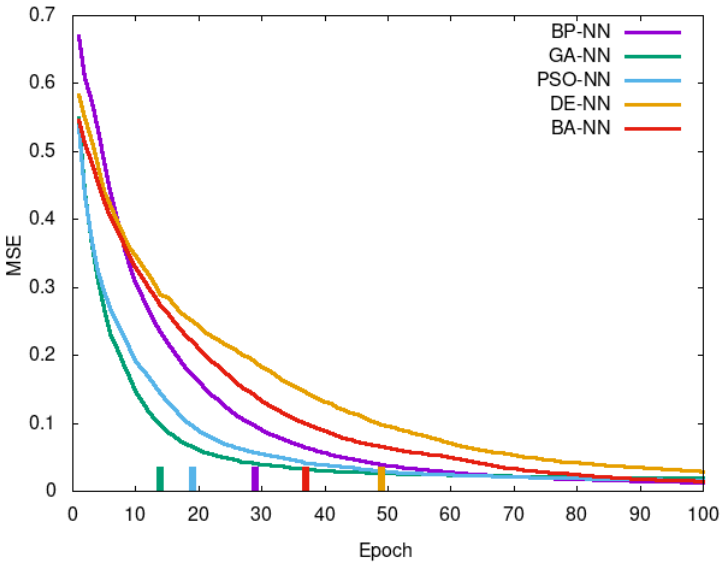

Figure 4: Penguins test results for metaheuristics and backpropagation.

It is difficult to discern visually whether curves come from the same distribution, so again an ANOVA test is performed. This test yields an $F = 9.0332$ and $p < 0.001$ which suggests there is confidence in curves not coming from the same population: outliers exist between training methods.

To identify them, Tukey's HSD test is performed, showing significance only between DE-NN and both GA-NN and PSO-NN. Here DE-NN is an outlier as the other samples show no significant difference.

From these tests, it is shown there is no significance in choosing between GA-NN, PSO-NN, BP-NN, or BA-NN; however, DE-NN is significantly slower than the other four.

	BP-NN	**GA-NN**	**PSO-NN**	**DE-NN**	**BA-NN**
Min	20	7	6	26	18
Avg	29	14	19	49	37

Table 11: Minimum and average time to terminate for Penguins data set.

In the previous table, as before the minimum and average number of epochs to reach a termination condition is given for each network type. As well, as before both GA-NN and PSO-NN are the fastest to train either as a single training or on average. DE-NN as found using the tests before was the slowest in every regard.

5.3 Wheat Seeds

Then, the Wheat Seeds data set is trained upon to attempt to make a model at classifying the data. Using the defined network parameters, the below plot is found:

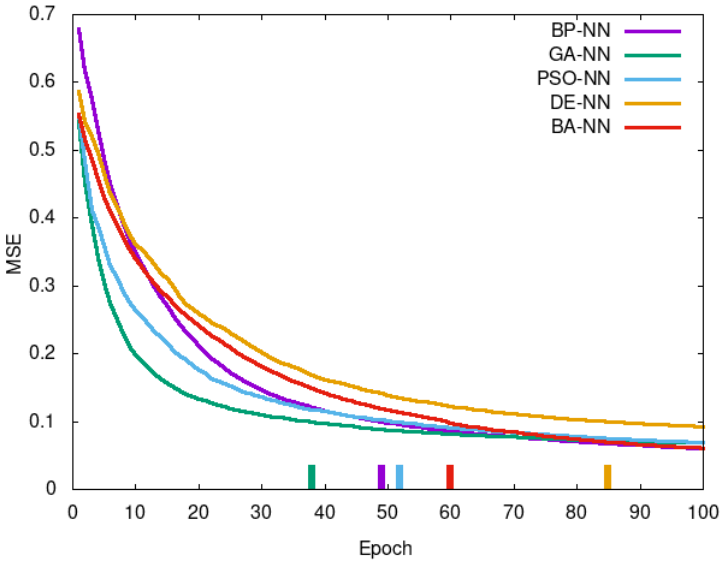

Figure 5: Wheat Seeds test results for metaheuristics and backpropagation.

To discover whether the curves are different to any meaningful degree, an ANOVA test is performed. This test yields an $F = 5.847$ with a $p < 0.001$, showing confidence that there is strong stochastic dominance between one or more samples.

Using Tukey's HSD test, each sample is examined and it is found that all samples—except DE-NN—come from the same distribution. This suggests there is no meaningful difference in choosing BP-NN, GA-NN, PSO-NN, or BP-NN. The only outlier is DE-NN.

While the ANOVA test showed significance, the significance is removed were DE-NN not a candidate sample for the tests, as pairwise difference of means between the other samples is insignificant.

	BP-NN	**GA-NN**	**PSO-NN**	**DE-NN**	**BA-NN**
Min	29	15	16	53	27
Avg	49	38	52	85	60

Table 12: Minimum and average time to terminate for Wheat Seeds data set.

Upon looking at the table above, this is the first test where GA-NN was fastest to converge in single-run training and on average. For the smaller problems, GA-NN was best in one or the other but it is interesting to see it outperform all other methods tested. However, since there is no significance in choosing it over the others, these results are inconclusive.

5.4 Wine

The Wine data set is trained upon using the previously defined network parameters. The below plot is made using those results:

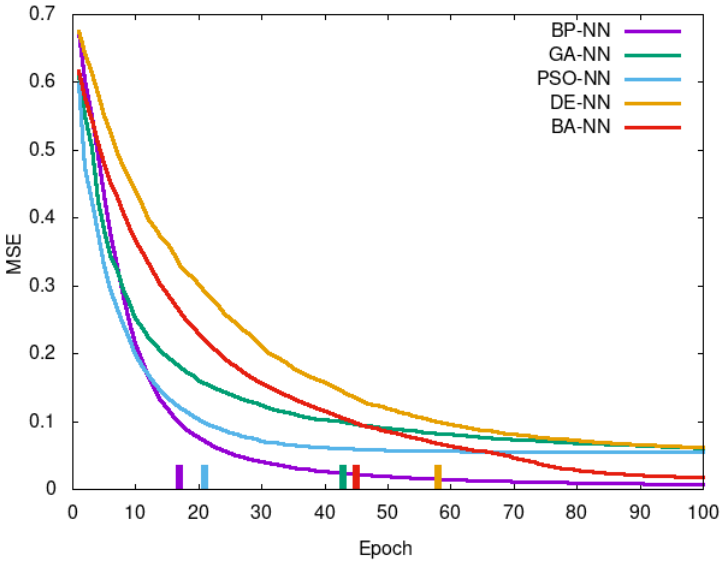

Figure 6: Wine test results for metaheuristics and backpropagation.

It is hard to determine which samples come from the same distribution, so an ANOVA test is performed yielding an $F = 12.081$, suggesting strong confidence in the test, and a $p < 0.001$ meaning strong significance in choosing some training methods over others.

To determine this, Tukey's HSD test is performed and the results are examined. The test suggests the curves come from three distributions: BP-NN and PSO-NN belong to the same distribution with strong confidence; GA-NN and BA-NN likewise belong to the same distribution; DE-NN is an outlier to each.

From this test, it can be surmised there is strong favor to choosing BP-NN or PSO-NN over other training methods.

	BP-NN	**GA-NN**	**PSO-NN**	**DE-NN**	**BA-NN**
Min	14	19	13	37	19
Avg	17	43	21	58	45

Table 13: Minimum and average time to terminate for Wine data set.

As before, a table is generated of the minimum and average number of epochs it took for each training method to converge.

Since BP-NN and PSO-NN have the smallest mean and minimum number of epochs to converge and they belong to the same distribution, either training method is preferable to the others. Of methods to choose, however, DE-NN is the outlier with the worst performance.

Interestingly, up to this point, GA-NN has been fastest to converge in single runs, on average, or both. This is the first problem where it is neither.

5.5 Breast Cancer

The Breast Cancer data set is then trained upon which yields the below plot using the defined network parameters for each training method:

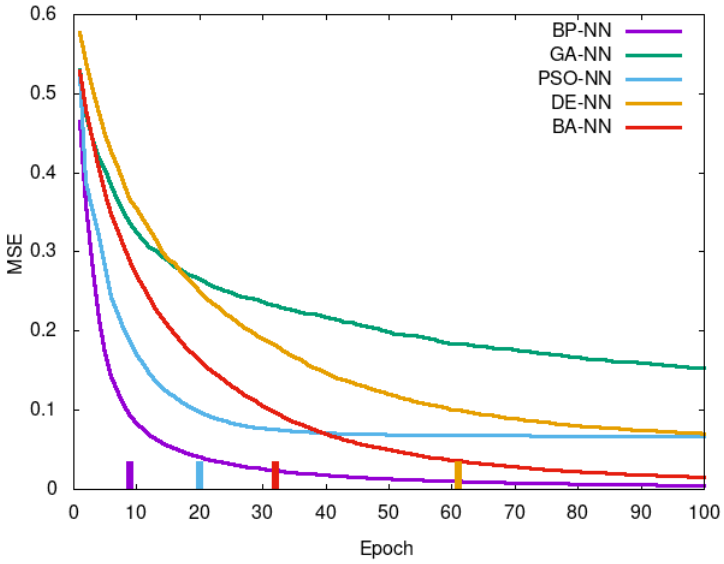

Figure 7: Breast Cancer test results for metaheuristics and backpropagation.

As with the Penguins data set, it is difficult to discern how related or unrelated each curve is other than GA-NN not converging.

To understand the plot better, an ANOVA test is performed yielding an $F = 61.494$ and $p < 0.001$ suggesting strong significance in choosing one or some training methods over others.

Tukey's HSD test is again performed to determine which samples come from the same distribution. The test reveals strong favor in choosing BP-NN over other methods. There is no significance in choosing PSO-NN over BA-NN or vice-versa, but there is slight significance in choosing BP-NN other either.

	BP-NN	**GA-NN**	**PSO-NN**	**DE-NN**	**BA-NN**
Min	6	52	11	37	16
Avg	9	DNC	20	61	32

Table 14: Minimum and average time to terminate for Breast Cancer set.

In the table above, performance is measured for each training method by way of the mean and minimum amount of epochs to reach termination condition. A "DNC" here, under GA-NN, means the training method did not converge.

As suggested by Tukey's HSD test, BP-NN has the speed advantage over the other training methods for the Breast Cancer classification problem.

5.6 Ionosphere Radar

The final data set, Ionosphere Radar, is test and plotted according to the previously defined network parameters:

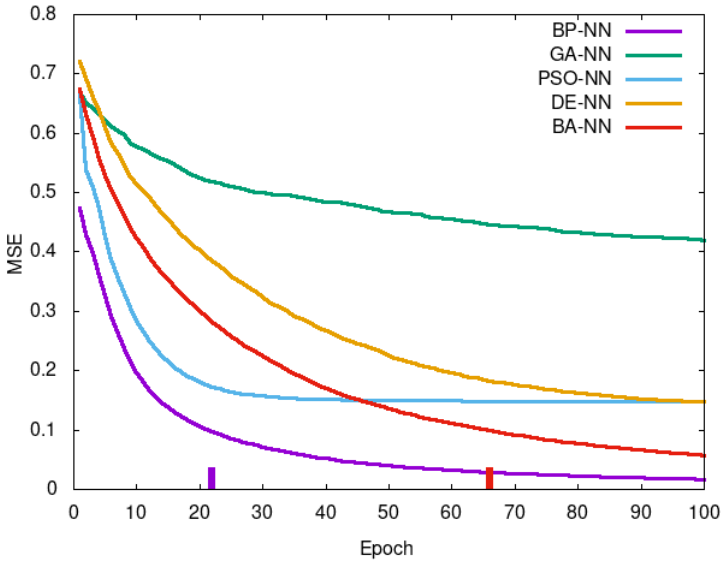

Figure 8: Ionosphere Radar test results for metaheuristics and backpropagation.

Here, only two training techniques converged within the allotted number of training iterations: BP-NN and BA-NN.

Visually, it is clear there is difference to the two methods, but to verify, an ANOVA test is performed, producing an $F = 185.7$, a very high F-score, and a $p < 0.001$, concluding that there is a large degree of significance between BP-NN and BA-NN.

Tukey's HSD test shows pairwise significance between samples as significant for all samples. What can be deduced from this is that between the two methods that converged, BP-NN is considerably faster.

	BP-NN	**GA-NN**	**PSO-NN**	**DE-NN**	**BA-NN**
Min	9	DNC	24	84	23
Avg	22	DNC	DNC	DNC	66

Table 15: Minimum and average time to terminate for Ionosphere Radar set.

Finally, a table is constructed showing the minimum and average number of epochs to reach a termination condition. As before, a "DNC" signifies the training method did not converge.

While PSO-NN and DE-NN could converge, on average they were unable to. Notable is GA-NN was unable to converge during any training runs. As alluded to from the ANOVA and Tukey HSD tests, BP-NN is the fastest training method for this problem.

6 Results Summary

On interpreting the results, a few things can be found for the specific data set classification problems used:

- For Iris, there is significance in choosing PSO-NN and GA-NN over the other methods. Notably, BP-NN is the worst performing training technique for this problem.
- For Penguins, there is significance in choosing any training method over DE-NN.
- For Wheat Seeds, there is significance in choosing any training method other than DE-NN, however the test is inconclusive on which of the remaining four training methods is superior.
- For Wine, there is significance in choosing BP-NN or PSO-NN over other training methods.
- For Breast Cancer, there is significance in choosing BP-NN over other training methods. Of note is for this test, some metaheuristics began to fail to converge.
- For Ionosphere Radar, there is significance in choosing BP-NN over other training methods. Notably, only BP-NN and BA-NN were consistently able to converge.

For the six tests performed, only for Iris were some metaheuristics able to outperform BP-NN. For Penguins, Wheat Seeds, and Wine problems, some metaheuristics were able to perform generally as well as BP-NN. Lastly, for Breast Cancer and Ionosphere Radar tests, BP-NN was the clear winner.

These results suggest metaheuristics may be appropriate for smaller problem sizes, where BP-NN is more suited for larger problems. The scalability of each training method can be verified by plotting convergence speed over dimensionality.

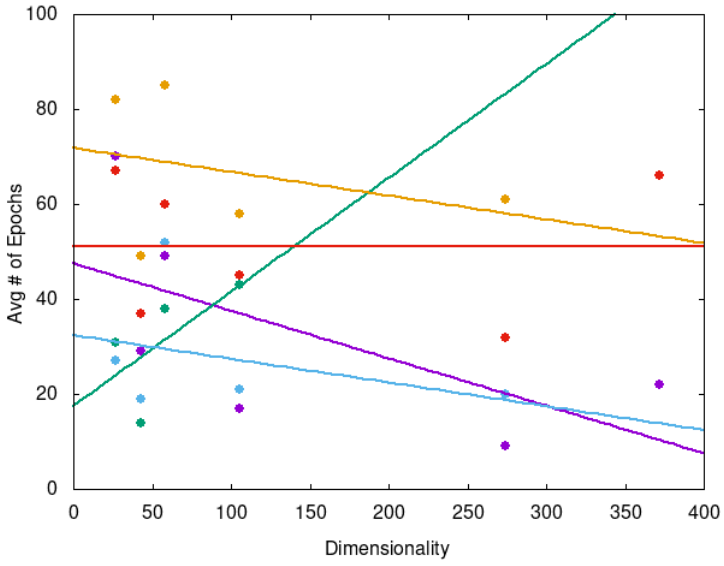

Figure 9: Scalability of training methods per dimension.

Each curve is colored for a specific training method: BP-NN is purpose, GA-NN is green, PSO-NN is blue, DE-NN is orange, and BA-NN is red.

This graph shows the scalability of each training method applied to problems of differing dimensionality. The slope of each line denotes how scalable each method is, with a positive slope meaning the method does not scale well with problem size, and a negative slope meaning it does scale.

From this, GA-NN is shown to scale poorly with problem size, yet is still the best performer for small problems. BP-NN, PSO-NN, and DE-NN scale well with problem size; however, DE-NN is the worst performer in most tests, and while it is as scalable as PSO-NN, the difference is consistent between problem sizes. BA-NN is shown to perform roughly equally agnostic of problem size. Likewise, it was the only metaheuristic to converge consistently for the largest test.

This verifies the idea that metaheuristics can outperform BP-NN for certain problem sizes, and while they can scale to higher dimensional problems, BP-NN is quick to overtake them in convergence speed.

7 Conclusion

There is merit to studying the use of bio-inspired metaheuristics in training neural networks: many of the advantages include speedier convergence or ability to minimize the error of a network in fewer steps—provided appropriate parameters and if the problem is within a certain size.

Overall, metaheuristics show promise applied to this problem. While a gradient-based approach like backpropagation is the go-to approach, a metaheuristic can be simpler to conceptualize for much of the same performance for some problems. As well, training a bio-inspired network using a bio-inspired metaheuristic seems appropriate.

References

[1] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: An International Journal*, 8(2):239–287, June 2009.

[2] D. Dua and C. Graff. UCI machine learning repository, 2017.

[3] D. Goldberg, G. David Edward, D. Goldberg, and V. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Artificial Intelligence. Addison-Wesley Publishing Company, 1989.

[4] A. M. Horst, A. P. Hill, and K. B. Gorman. *palmerpenguins: Palmer Archipelago (Antarctica) penguin data*, 2020. R package version 0.1.0.

[5] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[6] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, Dec. 1997.

[7] X.-S. Yang. *A New Metaheuristic Bat-Inspired Algorithm*, pages 65–74. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.