Neural Network Mini-Batch Training: A Comparative Analysis Between Gradient-Based Optimization and Meta-Heuristic Optimization, in a Dynamic Context

[*] An analysis between swarm-based optimization algorithms, and stochastic gradient decent, as neural network optimizers.

JC De Witt
Stellenbosch University
(Department of Computer Science)
Stellenbosch, South Africa
21663904@sun.ac.za

***Abstract*—Nature is a source of inspiration for computational models that are applied widely across a host of modern industries. Among the most influential, is the artificial neural network. Once trained, the artificial neural network can be applied to nearly any optimization and classification context. There is therefore tremendous incentive to devise efficient training algorithms capable of producing these models. It has been shown that training an artificial neural network using the popular technique of mini-batching changes the nature of the search space, from static to dynamic. It may therefore be useful to apply an optimization technique designed for dynamic applications. In this paper, particle swarm optimization as well as a dynamically adapted version thereof, is used to train an artificial neural network. The optimization success is compared to that of traditional gradient decent, across a variety of recorded metrics. Evidence was obtained that suggests that swarm-based algorithms can yield results superior to that of stochastic gradient decent, under certain conditions.**

***Keywords*—Particle Swarm Optimization, Dynamic Environment, Artificial Neural Networks, Mini-Batch Training.**

I. Introduction

The advent of the Artificial Neural Network has proved to be a pivotal moment across nearly the entire spectrum of global industry. Machine learning and artificial intelligence applications using these algorithms at their core, influence nearly all aspects of contemporary society. Despite the wide window of application associated with optimized models, configuring the weight and bias values associated with a network is a computationally expensive task. There exists an immense incentive to uncover new and efficient techniques capable of training these models, or even a subset of models, in an coherent manner. Research has uncovered that the total computational time involved in the optimization of a network's weight and bias values, can be reduced if the data on which the model learns is split into batches of a more digestible size. Unfortunately, splitting the dataset into batched subsets changes the optimization environment, from static to dynamic. Research suggests that using a optimization technique designed for the dynamic context will prove do be an efficient training technique. In this study we wish to explore this hypothesis by implementing a dynamic adaptation to the traditional particle swarm optimization (PSO) meta-heuristic. The merits of this algorithm will be evaluated against the static PSO counterpart, and traditional stochastic gradient decent. Several performance evaluation metrics are used to guarantee a tenable comparison.

The remainder of this paper is organized as follows: section II and III supply an overview of artificial neural networks and traditional particle swarm optimization. Section IV presents the quantum adaptation to traditional particle swarm optimization, and provides a discussion on the dynamic nature associated with mini-batch training. Section V highlights the empirical procedure used to conduct this study. Section VI mentions the algorithmic implementation, from which the results are gathered. The Results are presented in section VII, and are subsequently summarized. Section VIII concludes the paper.

II. Neural Network Background

An artificial neural network (ANN) is a machine learning model inspired by the human brain in terms of function and structure. ANNs are composed of a connected, layered, series of artificial neuron (AN) perceptrons. The perceptron was proposed by McCulloch and Pitts [1] in 1943, and acts as a self contained linear regression model. The perceptron accepts a series of input signals and subsequently produces a single output signal, from which conclusions are inferred. In the context of ANNs, the output signal of a single AN proportionally acts as a member of the input signal corresponding to a ANs in a subsequent layer. The net input signal of an internal AN is calculated as [2]:

$$z = \sum_{i=0}^{n} x_i w_i - b \tag{1}$$

where the notation of this equation is defined as:

1) w_i is the weight connecting AN i of the previous layer, to the AN which is being evaluated.
2) x_i is the activation of the AN in the previous layer to which w_i connects.
3) b represents the bias of the AN which is being evaluated. Intuitively, the bias acts as an "activation threshold" which must be exceeded before the neuron fires.

The resulting propagation of signals throughout the perceptron layers is how the ANN evaluates a data point. For the comparisons conducted in this paper, we have implemented a standard *Feedforward* neural network (FFNN). A FFNN is comprised of three layers: an input layer, a hidden layer, and an output layer. A FFNN may contain multiple hidden layers, however it has been proved that a FFNN with a single hidden layer can approximate any continuous function, provided that the hidden layer contains enough ANs, and that a monotonically increasing differentiable activation function is being used. Subsequent to this finding, the FFNNs implemented in this study contain only one hidden layer, with an over-estimation on the number of ANs that this layer should contain.

A. Forward Propagation

Provided an input pattern $\mathbf{z}_p$, we determine the corresponding output pattern, $\mathbf{o}_p$, with a single forward pass through the network. For each output unit k, associated with output pattern $\mathbf{o}_p$, we have

$$\begin{aligned} o_{k,p} &= f_{o_k}(net_{o_{k,p}}) \\ &= f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}(net_{y_j,p})\right) \\ &= f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{I+1} v_{ji} z_{i,p}\right)\right) \end{aligned} \tag{2}$$

The notation of (2) is clarified:

1) f_{o_k} and f_{y_j} are the activation functions for output unit o_k and hidden unit y_j respectively.
2) w_{kj} is the weight of the edge connecting output unit o_k and hidden unit y_j.
3) $z_{i,p}$ is the value of input unit z_i corresponding to the input pattern $\mathbf{z}_p$.
4) For convenience in computation, the bias units referenced in (1) are packed into the $(I+1)$-th and $(J+1)$-th indicies of the input and hidden patterns, respectively.

After the forward propagation procedure, the index of the most activated unit in the output pattern determines the class to which the sample that produced input patter $\mathbf{z}_p$ is classified. Optimizing the values of the bias and weight values associated with each AN forms part of the network training process. Differentiating between favorable and poor configurations depends on the evaluation of a chosen cost function. The cost function used in this implementation is discussed in section IV.

III. Neural Network Training Background

Optimizing the neural network involves algorithmically determining a configuration of weight and bias values such that the units of the output vector match closely to those specified by the corresponding target vector. Assuming sum of squared error (SSE) is being used to evaluate cost, we wish to minimize

$$\varepsilon_p = \sum_{k=1}^{K} (t_{k,p} - o_{k,p})^2 \tag{3}$$

where K is the number of output units, and $t_{k,p}$ and $o_{k,p}$ are the target and actual output values of the k-th output unit, respectively. There are many techniques capable of determining desirable weight and bias configurations that minimize ε_p, though these methods do not guarantee an optimal configuration be found [3]. The selection of a suitable training technique is often times problem specific. There exist approaches which seek to estimate the directed derivative of the error surface at a point, and update the weight and bias values such that the model moves in the direction of reduced error. Gradient decent is such a method. Other methods focus on a macro-view of the error topology, using a memory of previously visited locations (and their respective fitness) to guide the movement towards suitable minima. Particle swarm optimization is such a method. The following subsections provide insight into the workings of these methods.

A. Gradient Decent

Although gradient descent (GD) is not the first training rule for ANNs, it is possibly the most popular method used. GD requires the definition of an error function, such as (3). The aim of GD is to determine weight and bias values that minimize ε. This is achieved by calculating the gradient of ε in weight space, and adjusting the weight vector along its negative gradient [2]. Given a single training pattern, weights are updated using

$$v_i(t) = v_i(t-1) + \Delta v_i(t) \tag{4}$$

with

$$\Delta v_i(t) = \eta(-\frac{\partial \varepsilon}{\partial v_i})$$

where

$$\frac{\partial \varepsilon}{\partial v_i} = -2(t_p - o_p)\frac{\partial f}{\partial net_p} z_{i,p}$$

The size of the step taken in the direction of the negative gradient is dictated through the η parameter, often labeled the *learning rate*. Determining a suitable value for η is not an exact science, often times being selected through a trial and error process. For the purposes of this study, $\eta = 0.1$ was empirically determined as suitable. Notice the $\frac{\partial f}{\partial net_p}$ term. The calculation of the partial derivative of our activation function f with respect to the net input pattern, net_p, presents a problem for all discontinuous activation functions, where this derivative is not defined. This activation function differentiability constraint represents a major restriction that other training methods, such as PSO, are not confined by.

The standard Inertia Weight Particle Swarm Optimization algorithm is discussed in the proceeding subsection.

B. Inertia Weight Particle Swarm Optimization

Particle swarm optimization (PSO), as established by Kennedy and Eberhart in 1995 [4], is a nature-inspired stochastic optimization algorithm. In PSO, a population of particles traverse a search space guided by both global and local topology. The position of a particle during this procedure represents a candidate solution to a predetermined optimization problem. The quality of a particular solution is assessed by means of an objective function evaluation. The movement characteristics of particles greatly depend on the values of key model parameters.

1) PSO Velocity and Position Update: Various PSO techniques have been developed since the initial contributions of Kennedy and Eberhart. This paper uses the inertia weight model of PSO, developed by Shi and Eberhart [5]. Intertia weight PSO establishes a proportional relationship between a particle's prior velocity and its successive velocity. According to this model, the equations to update the velocity and position of a particle are given by equations (3) and (4)

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1} \tag{5}$$

$$\mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + c_1\mathbf{r}_1(\mathbf{y}_i^t - \mathbf{x}_i^t) + c_2\mathbf{r}_2(\hat{\mathbf{y}}_i^t - \mathbf{x}_i^t) \tag{6}$$

Bold symbols in the above equations represent n dimensional vectors, where n corresponds to the dimensionality of the search domain. The syllabary of equations (3) and (4) are clarified below:

- $\mathbf{x}_i^t$ and $\mathbf{x}_i^{t+1}$ specify the position of the i^{th} particle during iteration t and $t+1$ respectively. The position of a particle represents a potential solution to the optimization problem.
- $\mathbf{v}_i^t$ and $\mathbf{v}_i^{t+1}$ denote the velocity of the i^{th} particle during iteration t and $t+1$ respectively. It is important to note that the velocity vector of a particle specifies both the direction this particle is moving, as well as the magnitude of the movement. The velocity of all particles are initialized to the zero vector at iteration one.
- $\mathbf{y}_i^t$ represents the position which yields the best objective function evaluation, that has been visited by particle i. This point is commonly referred to as a particle's personal best vector.
- $\hat{\mathbf{y}}_i^t$ denotes the position which yields the best objective function evaluation that any particle in the neighborhood has visited. In this paper, a ring topology was used as the particle neighborhood.
- $\mathbf{r}_1$ and $\mathbf{r}_2$ are n dimensional random vectors, where each element in the vector is chosen from a *uniform(0,1)* distribution.
- c_1 is a control coefficient for the term $\mathbf{r}_1(\mathbf{y}_i^t - \mathbf{x}_i^t)$. Collectively this term is referred to as the "cognitive component", which quantifies performance of the particles current position relative to that of the particles personal best. This interaction has aptly been coined "nostalgia" by Kennedy and Eberhart [4].
- c_2 is the control coefficient for the term $\mathbf{r}_2(\hat{\mathbf{y}}_i^t - \mathbf{x}_i^t)$. Collectively, this term is referred to as the "social component", which quantifies performance of the particle's current position relative to that of the particle's neighborhood best. This interaction has similarly been referred to as "envy" by Kennedy and Eberhart [4].
- w regulates the affect that the previous velocity has on the calculation of the velocity in the subsequent iteration. The velocity of the previous iteration is often regarded as the "inertia" or "momentum" of that particle, hence the name for this PSO model - Interia Weight PSO.

2) Particle Convergence Condition: If the control parameters w, c_1, c_2 are chosen to satisfy:

$$c_1 + c_2 < \frac{24(1-w^2)}{7-5w} \quad \text{for } w \in [-1,1] \tag{7}$$

it has been both theoretically and empirically demonstrated that particles will eventually reach equilibrium [6] [7] [8] [9]. Investigation has revealed that we can fine tune the behavioural characteristics of the swarm by configuring w, c_1 and c_2 to obtain a spectrum of trade-offs between exploration and exploitation, however in this paper we have opted to maintain a poli-stable configuration of $w = 0.7$, $c_1 = 1.4$ and $c_2 = 1.4$.

3) The PSO Algorithm: The Particle swarm optimization algorithm is provided in Algorithm 1.

Algorithm 1 Standard PSO Algorithm

```
Create and initialize an n_x-dimensional swarm, S;
repeat
    for each particle i=1,...,S.n_s do
        if f(S.x_i) < f(S.y_i) then
            S.y_i = S.x_i;
        end if
        if f(S.y_i) < f(S.ŷ_i) then
            S.ŷ_i = S.y_i;
        end if
    end for
    for each particle i=1,...,S.n_s do
        update the velocity;
        update the position;
    end for
until stopping condition is true
```

IV. Methodology and Implementation

In this section, details regarding the nature of the search environment being traversed, are disclosed. Thereafter, an adapted version of PSO is presented, which is designed to be better suited for such environments.

A. Stochastic Mini-batch Training

There are applications when the practicality of using traditional gradient decent as an ANN optimizer is computationally

unrealistic. For example, if the training dataset is large, it is expensive (and unnecessary) to back-propagate all the datapoints through the network to obtain the direction of steepest decent. Methodologically, the concept of mini-batch training involves back-propagating a subset of the training data through the network, and estimating the direction of steepest decent, based upon information gained from this propagation in isolation. Trivially, if a subset of the dataset is randomly selected, it may be that the estimated "downhill" direction differs from the true direction of steepest decent. The intention of using a subset of the dataset during each training iteration is to take more steps in the direction of *a* negative gradient in the same time-frame as a single step in the absolute negative gradient. Irregardless of the optimization method used, we are reducing the accuracy of our downhill estimation when exposing ourselves to only a subset of the dataset, but we are increasing our estimation frequency. This trade-off suggests there may exist an optimal batch size that perfectly balances estimation integrity, with estimation frequency. This paper implements its analysis across a variety of batch sizes for comparative robustness. Throughout the literature, an *epoch* is used to label a backwards propagation iteration with respect to a batch of training data. This paper adopts this convention.

B. Dynamic Search Environment

As discussed in a previous section, training a neural network is equivalent to the optimization problem of identifying a set of weight and bias values which result in a minimized objective function evaluation. Traditionally, an optimization problem is static. In this regard, the quality of a candidate solution does not depend on the moment in which the solution is evaluated. By contrast, a dynamic optimization problem is a problem in which the quality of a solution is time dependent. Solutions which were previously evaluated, and found to perform favorably, may perform poorly in subsequent evaluations. Similarly, solutions which under performed during previous evaluations, may be found to perform well during subsequent iterations [10]. When the error of a specific weight vector can change with time, such as batch selection during stochastic mini-batch training, the error surface becomes dynamic [11]. There are adaptations to be made to traditional static optimizers, to make them more suitable for optimization in a dynamic context. Quantum particle swarm optimization is an adapted form of traditional PSO which is more suited for dynamic optimization applications.

C. Quantum Particle Swarm Optimization

To make PSO more suitable for the dynamic environment, we initialize a proportion of the swarm population as quantum individuals, rather than traditional PSO particles. A quantum individual is one which randomly gets reinitialized to a position offset from the neighborhood best position, by a calculated radius. The initialization rule of a quantum individual is provided in algorithm 2.

This process increases swarm diversity, increasing the exploratory behaviour of the swarm. In the context of quantum particle swarm optimization (QPSO), we randomly reinitialize a particle within a ball centered at the global best individual, $\hat{\mathbf{x}}(t)$. The radius, and re-initialization probability are configurable model parameters, the configurations used in this study are disclosed in section V.

Algorithm 2 Initialization of Quantum Individuals

1: **for** each individual, $\mathbf{x}_i(t)$, to be re-initialized **do**
2: Generate a random vector, $\mathbf{r}_i \sim N(0,1)$
3: Compute the distance of $\mathbf{r}_i$ from the origin:
4: $\varepsilon(\mathbf{r}_i, 0) = \sqrt{\sum_{j=1}^{n_x} r_{ij}}$
5: Find the radius, $R \sim U(0, R_{max})$
6: $\mathbf{x}_i(t) = \hat{\mathbf{x}}(t) + \frac{R\mathbf{r}_i}{\varepsilon(\mathbf{r}_i,0)}$
7: **end for**

D. Regularization

Hornik *et al* [12] proved that a FFNN with a single hidden layer can approximate any continuous function, provided that the hidden layer contains enough ANs, and that a monotonically increasing differentiable activation function is being used. However, if we vastly overestimate the quantity of ANs needed to implement a favorable classification, our model may be prone to over fitting. That is, if we are careless, our model may opt to memorize training data. The result will be a network that does not generalize well, yielding poor performance on unseen test data. To curtail potential overfitting tendencies, we add a penalty to the objection function to be minimized. In this case, the objective function changes to

$$\varepsilon = \varepsilon_T + \lambda \varepsilon_C$$

where ε_T is the usual measure of data misfit, and ε_C is a penalty term, penalizing network complexity [2]. The constant λ controls the influence of the penalty term. With this adjusted objective function, the ANN now attempts to identify a locally optimal trade-off between data-mistfit, and network complexity. This paper implements *weight decay*, a popular regularization technique where λ_C is selected as:

$$\varepsilon_C = \frac{1}{2}\sum w_i^2$$

To demonstrate the influence of weight decay, the following original illustrations are shown:

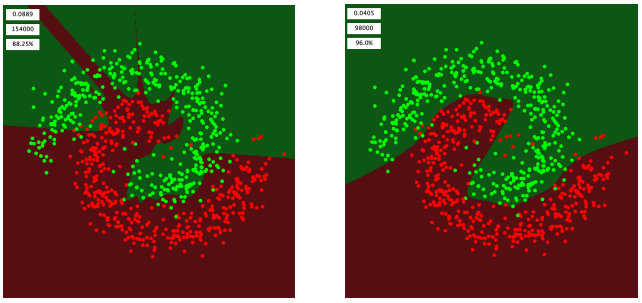

Fig. 1: An original visualization of the decision boundaries of an ANN using no regularization (left), and an ANN using weight decay with $\lambda = 0.01$ (right). The visualizations framework is implemented in Java, as part of this study. Find more here.

E. Comparative Scheme

For the purposes of this study, we wish to explore the optimization ability of stochastic gradient decent and particle swarm optimization in the context of training artificial neural networks, across different batch sizes. In light of the research [10] [11] suggesting the dynamic nature of ANN training using mini-batching, we also implement QPSO to record its performance in comparison to its static counterpart. We conduct the comparison over a variety of batch sizes, ranging from small, to large, over multiple independent trials. The overall performance, and variance in performance between trials is recorded.

V. Empirical Procedure

In this section the details and justifications regarding the implementation of a robust empirical comparison are presented. In the first subsection we present the relationship between the PSO search space, and its corresponding ANN configuration. Thereafter a mechanism included to balance the *per-iteration* computation of PSO and stochastic gradient decent is discussed. The selected activation function is mentioned next. Finally, comparative details are disclosed. Details of interest include: the iteration threshold, batch sizes, model parameters, and comparison metrics.

A. Connecting a PSO Solution to a Neural Network Configuration

In this implementation, an initialized ANN can be decomposed into a single vector containing its weight and bias values. The size of this vector corresponds to the dimension of the search space that the particles in the swarm traverse. At each iteration, a particle from the swarm initializes a neural network using the weight and bias configurations associated with the particle's position. Thereafter, the cost of the candidate configuration is determined with respect to a validation set, selected as a subset of the training data. The validation set remains consistent throughout the independent trial. The split ratio selected for the comparisons implemented in this study, are 70% training data, and 30% validation data. Once the iteration threshold is met, the total classification accuracy is determined on a test set which has never been exposed to the ANN, until this point.

B. Computational Effort Balancing Between PSO and SGD

During the PSO position update rule, a neural network is created and evaluated for each particle in the swarm. The implication is that PSO is given an unfair advantage in terms of computational privilege. To re-balance the algorithms, an array of ANNs is initialized at the start of each trial. The length of the array corresponds to the selected swarm size, which was configured to 30 particles for the purposes of this study. During each iteration, all of the 30 ANNs are back-propagated using stochastic gradient decent. At the end of the trial, the performance of the most fit network is recorded. This network is thereafter compared to the global best solution identified by PSO.

C. The Selected Activation Function

In this study a variety of activation functions were considered. For comparative simplicity, it was decided that a single activation function should be used throughout the FFNN. Through a trial-and-error process it was determined that the best performing activation function subject to this constraint was the *sigmoid* activation function, with unit slope and range on the interval $[0, 1]$.

$$f(net) = \frac{1}{1 + e^{-net}} \tag{8}$$

D. Comparison Fields and Metrics

To evaluate the success of a candidate artificial neural network configuration, we track three performance metrics across each independent trial. The metrics of interest include:

- Mean squared error (MSE) per epoch.
- Classification accuracy at the end of each epoch on a hidden evaluation dataset.
- Variance in MSE between independent trials.

An optimal configuration minimizes MSE and its respective inter-trial variance, while simultaneously maximizing the classification accuracy with respect to the unseen evaluation set. The simulation procedure is conducted over 20 independent trials of $10,000$ epochs each. The batch sizes considered in this study are

Batch Sizes				
10	50	100	200	500

For each batch size, a series of 20 independent trials are conducted using PSO, QPSO, and SGD as ANN training algorithms. The mentioned evaluation metrics are subsequently recorded for further evaluation.

E. Network Architecture

Due to the use of weight decay as a regularization technique, we opt to overestimate the number of ANs in the network's hidden layer. In this regard, we have decided to use a hidden layer comprised of 20 artificial neuron perceptrons. For simple classification problems, we rely on the implemented regularization technique to curtail over-fitting.

VI. Algorithmic Implementation

The results presented in this study are independently obtained through an original implementation using Java as the computational workhorse. The visualizations were also manufactured as a part of this study, through the use of a canvasing framework, written in Java. The visualization library written for the purpose of this paper is used to illustrate the geometric change of a model's decision boundary throughout the training procedure. After the results are obtained, the data is written to output files which can be further processed by a Python script written for the purposes of this study. All plots are generated using Python's *Matplotlib.pyplot* library. Please contact the writer to request a copy of the data, or code, used in this study.

A. Dataset Construction

The datasets used for classification were created for the purposes of this study. Three datasets were created, each ranging in difficulty. The most simple dataset is a 3-class dataset which can be linearly separated. Thereafter is a 5-class dataset where datapoints are grouped by class, and plotted in a pinwheel-esque fashion. The pinwheel set is fairly successfully linearly separable, though it does require some non-linearity. Finally, a 2-class dataset where points are follow a swirl distribution, poses the most significant challenge to the optimizers. This set requires the identification of a highly convex decision boundary. The datasets are illustrated in Figure 2

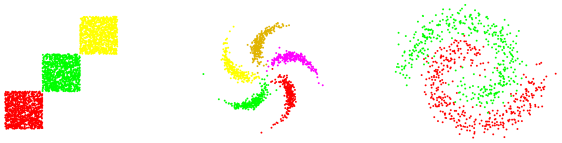

Fig. 2: Visual representation of the created datasets.

VII. Research Results

In this section, the empirical results and observations regarding the comparison between the neural network training algorithms are presented to the reader.

A. Sensitivity to Batch Size

In the context of mini-batching, it is desirable that the identified neural network configuration be reasonably stable between independent training trials. That is to say, we should be confident that the obtained output network is both capable in terms of classification, and stable in terms of repeatability. The simulation results suggest that both particle swarm algorithms are highly sensitive to the selected mini-batch size.

It is observed that both PSO and QPSO are especially sensitive to small mini-batch sizes. Figure 3 illustrates the degree to which a poorly selected batch size can adversely effect training success. Notice that for small batch sizes there is large variance between independent trials, as well as extremely high final mean square error. The figure suggests that the

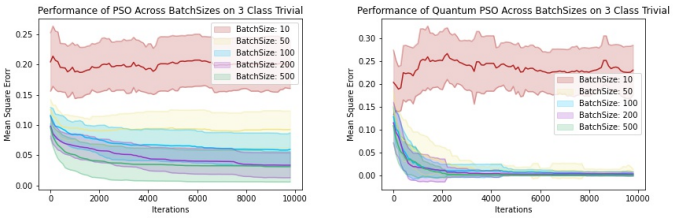

Fig. 3: PSO displaying high dependency on the selected batch size. This behaviour is displayed across all test-sets, for both PSO and QPSO.

computational effort invested into training the network is wasted if the batch size is chosen too modestly. For a batch size of 10, there is no evidence suggesting that the model is improved throughout the training procedure. A similar sensitivity was observed with regards to classification accuracy on the unseen test set. To present this result, the distribution of final classification accuracy with regards to each independent trial is shown.

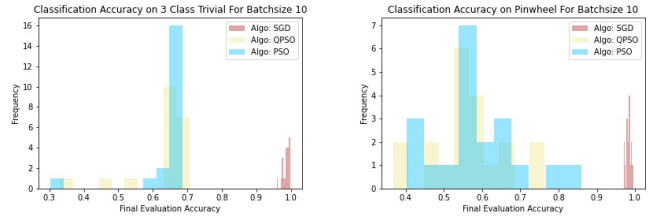

Fig. 4: The classification accuracy distribution of each algorithm when a batch size of 10 is used. Notice the resistance that SGD displays with regards to modest batch size choices. This tendency was exhibited for all three test sets used in this study.

Figure 4 illustrates the vulnerability that the swarm based algorithms display when implemented using a poor choice of mini-batch size. Both PSO and QPSO seem to be approximately equally susceptible to this shortcoming. Notice the degree to which SGD resists the influence of an incorrect mini-batch configuration, maintaining both a high-accuracy and low-variance distribution, across all the datasets used in this study. Interestingly, the compliment of this observation was also noticed. When a large batch size was used, swarm based algorithms seemed to outperform SGD by a considerable margin.

Figure 5 suggests that if the batch size is chosen to benefit swarm-based algorithms, they outperform traditional SGD. Furthermore, it is evident that QPSO identified both the most desirable network configuration, and it did so with the highest stability. That is, the variance of the identified configuration was low, between independent trials. This observation suggests that it may in fact be worthwhile to implement a optimization algorithm suited for the implicitly dynamic environment resulting from mini-batch training. Once more, this observation is reflected in the final accuracy distribution.

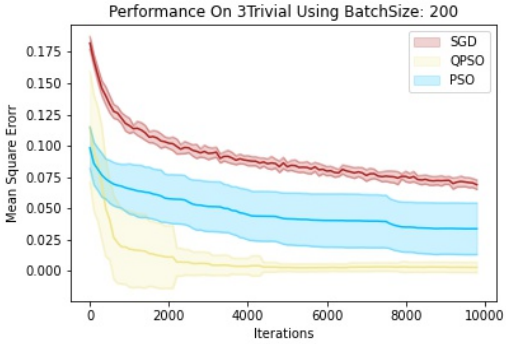

Fig. 5: When the mini-batch size was chosen sufficiently large, swarm based optimization algorithms performed better than SGD by a considerable margin.

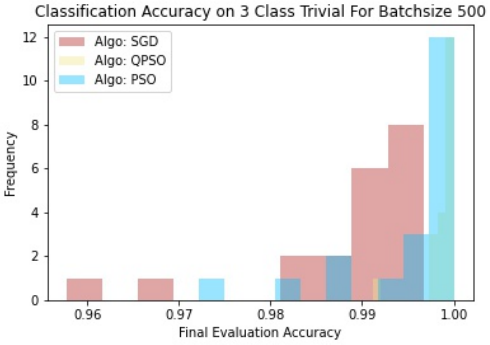

Fig. 6: For very large mini-batch sizes, swarm based algorithms outperform traditional gradient decent in terms of final classification accuracy on the unseen test set.

The accuracy distribution illustrated in Figure 6 affirm the findings of Figure 5. For large batch sizes, swarm based algorithms outperform traditional gradient decent in terms of final classification accuracy as well. Once again, the dynamic adaptation to traditional PSO yielded the most favorable result, displaying the lowest variance distribution, with highest mean inter-trial accuracy.

Although SGD did not perform the best in all scenarios, the resistance it displayed to changes in mini-batch size is notable. The final classification accuracy was impressive regardless of the batch size used, as seen in Figure 7. This resilience may make SGD an attractive training algorithm for applications where robustness is essential, or computational effort cannot be dedicated towards suitable tuning. This observation is further reinforced by Figure 8.

It is observed that lower batch sizes are associated with more stochasticity in the training procedure, but not nearly to the same degree of sensitivity as the competitor algorithms. With regards to PSO and QPSO, Figure 3 clearly shows that for low batch sizes, training the model has marginal effect. Although SGD does show increased variance for low batch sizes, there is still a clear trend of decreasing mean squared error throughout the training process.

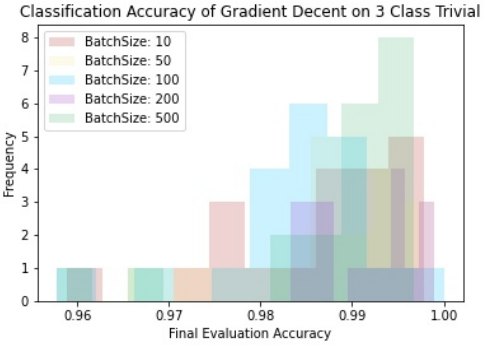

Fig. 7: SGD displayed an impressive resistance to changes in mini-batch size. Regardless of the selected batch size, the final classification accuracy is higher than 98% on average.

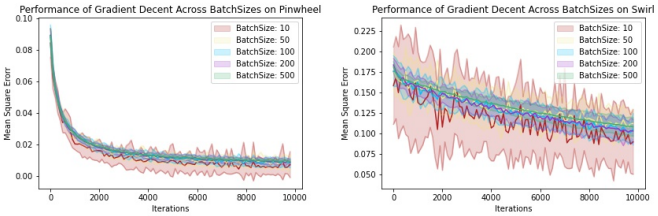

Fig. 8: SGD displayed a low error, high stability, tendency regardless of the batch size used during training.

B. *Classification Ability*

It was observed that swarm based algorithms struggle to correctly identify highly nonlinear decision boundaries. In the context of the swirl dataset used in this study, both PSO and QPSO did not perform as well as traditional SGD. Both swarm based algorithms found an acceptable linear classification which yielded approximately 75% test accuracy. The boundaries identified by each algorithm are presented in Figure 9

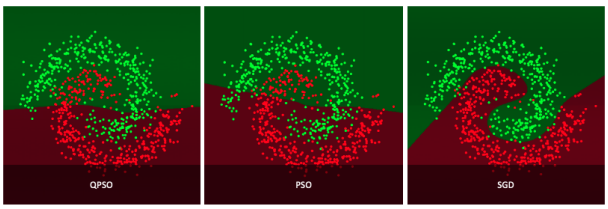

Fig. 9: The obtained classification boundaries for QPSO, PSO, and SGD reading from left to right.

Notice that the decision boundaries identified by the swarm based algorithms are fairly linear in nature. This trend was consistent throughout independent simulations. This observation is further affirmed by Figure 10

The continuous identification of linear classification boundaries by swarm-based algorithms on the swirl dataset led to unchanging mean squared error throughout the majority of the independent trials. The optimization space likely contains

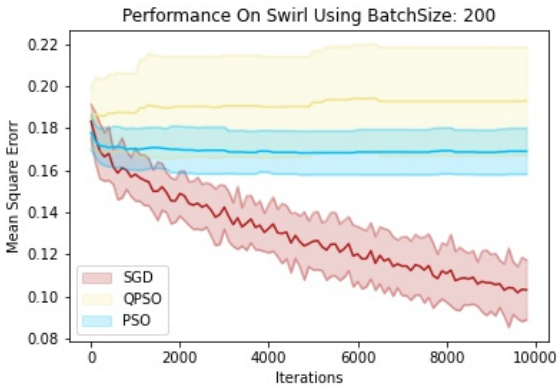

Fig. 10: Performance of training algorithms on the swirl dataset. Notice that the swarm-based algorithms seem to stagnate.

a high-volume local minimum in this region, that captures the entire swarm, at a width so large that the quantum particles cannot tunnel towards other desirable minima. Further investigation can be employed to make QPSO more suitable for this dataset. Overall there merit towards using swarm based algorithms in the context of neural network optimization, although this study concludes that they perform best when the classification boundary is somewhat linear in nature (Pinwheel and 3ClassTrivial datasets). Swarm based algorithms did not perform well when the classification boundary is highly non-linear (Swirl), likely a result of PSO's dimensionality curse. More investigation can be conducted in the future, to study this behaviour further. PSO can potentially be adapted to be more suitable for this application.

C. Further Classification Boundary Visualization

If the reader wishes to further explore the classification boundaries identified by these algorithms across a several more datasets created for the purpose of this study, consider visiting the following link:

Press Here To Explore More Classification Boundary Visualization

The visualizations omitted are beyond the scope of this study, though the behaviour of swarm based algorithms in the context of neural network optimization can be further inspected if the reader chooses.

VIII. Conclusion

The intention of this paper was to investigate the practicality of using swarm-based algorithms in the context of mini-batch neural network training. It has been shown that the optimization environment associated with models using mini-batch training is dynamic in nature. Evidence was obtained that suggests using an adapted particle swarm optimization algorithm (PSO), quantum PSO, yields superior performance to its static counterpart. It was concluded that traditional gradient decent is more robust to mini-batch sizes, and is the overall most stable algorithm. There were however instances where quantum particle swarm optimization (QPSO) outperformed both standard inertia weight PSO and stochastic gradient decent SGD. The algorithm of choice therefore largely depends on the application, and the computational effort which can be dedicated towards tuning the selected model. Swarm-based algorithms were shown to perform well for classification problems with fairly linear classification boundaries.

REFERENCES

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[2] A. P. Engelbrecht, *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[3] A. Ghaffari, H. Abdollahi, M. Khoshayand, I. S. Bozchalooi, A. Dadgar, and M. Rafiee-Tehrani, "Performance comparison of neural network training algorithms in modeling of bimodal drug delivery," *International journal of pharmaceutics*, vol. 327, no. 1-2, pp. 126–138, 2006.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[5] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*. IEEE, 1998, pp. 69–73.

[6] C. W. Cleghorn and A. P. Engelbrecht, "Particle swarm convergence: An empirical investigation," in *2014 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2014, pp. 2524–2530.

[7] R. Poli, "Mean and variance of the sampling distribution of particle swarm optimizers during stagnation," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 4, pp. 712–721, 2009.

[8] M. Xu and J. Gu, "Parameter selection for particle swarm optimization based on stochastic multi-objective optimization," in *2015 Chinese automation congress (CAC)*. IEEE, 2015, pp. 2074–2079.

[9] Q. Liu, "Order-2 stability analysis of particle swarm optimization," *Evolutionary computation*, vol. 23, no. 2, pp. 187–216, 2015.

[10] C. Dennis, A. Engelbrecht, and B. M. Ombuki-Berman, "An analysis of the impact of subsampling on the neural network error surface," *Neurocomputing*, vol. 466, pp. 252–264, 2021.

[11] A. S. Rakitianskaia and A. P. Engelbrecht, "Training feedforward neural networks with dynamic particle swarm optimisation," *Swarm Intelligence*, vol. 6, no. 3, pp. 233–270, 2012.

[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.