Dynamic Sampling in Training Artificial Neural Networks with Overlapping Swarm Intelligence

Shehzad Qureshi
Department of Computer Science
Johns Hopkins University
Elkridge, MD 21075, USA
Email: squresh6@jhu.edu

John W. Sheppard
Department of Computer Science
Montana State University
EPS 357, PO Box 173880
Bozeman, MT 59717-3880
Email: john.sheppard@montana.edu

***Abstract*—This paper describes an extension to overlapping swarm intelligence for training artificial neural networks. Overlapping swarm intelligence is an application of particle swarm optimization that divides the network into paths from input to output, with each path represented by a swarm. Previous versions of this algorithm showed success on training networks on a variety of datasets but the method suffers from an explosion in fitness evaluations due to the number of paths that need to be evaluated. We propose an extension to overlapping swarm intelligence to use asynchronous updates and dynamic subsets of swarms for each generation, and demonstrate that this method performs as well as basic overlapping swarm intelligence in terms of mean squared error and classification accuracy with fewer fitness evaluations.**

I. Introduction

Artificial neural networks have been used extensively for a wide variety of tasks including classification, regression and pattern recognition. Large corporations such as IBM, Google, Microsoft, Facebook and Yahoo all use deep networks, consisting of many layers of neurons in a wide variety of configurations to solve computationally complex tasks.

Training an artificial neural network is usually done via the gradient descent method or one of its variants but is known to suffer from getting stuck in local optima [1]. In addition to that, the back-propagation algorithm depends on the error function itself which may not be simple to derive. Additional layers in the network only add more non-linear complexities which makes training even harder and suffer from the vanishing and exploding gradient problem [2].

Evolutionary computation has been applied to train neural networks in place of gradient descent. Genetic algorithms and particle swarm optimization have both been used individually and combined as hybrids to train neural networks, often successfully [3]–[5]. More recently, Ganesan Pillai and Sheppard introduced a novel adaptation of particle swarm optimization called overlapping swarm intelligence [6]. Overlapping swarm intelligence decomposes the unique paths of a network from input to output into swarms to train the network. Inter-swarm communication is facilitated by credit assignment and a common vector of network weights. This concept is similar to the cooperative split particle swarm optimization, first introduced by Van den Bergh and Engelbrecht [7].

This paper presents an approach to improve on the work of overlapping swarm intelligence by sampling possible sub-swarms in the search space, thereby reducing the computational complexity of this method. Since the number of swarms are less than the actual paths of the network, dynamic sampling of swarms will be introduced into the algorithm that will train all edges within the network. Section 2 of this paper explores related work in training neural networks with particle swarm optimization, Section 3 gives a brief overview of artificial neural networks, specifically the feed-forward neural network, particle swarm optimization, and overlapping swarm intelligence. It also presents our extension to the algorithm. Section 4 details our experimental setup and in Section 5 we present our results. Finally, Section 6 and 7 discuss our observations and our conclusions, respectively.

II. Related Work

Gudise and Venayagamoorthy did a comparison of training neural networks using back-propagation and particle swarm optimization [8]. Their work involved presenting the network with a non-linear quadratic function and training it using both algorithms. They concluded that a particle swarm optimization algorithm with well-defined parameters clearly outperforms a conventional back-propagation algorithm in terms of global convergence speed, assuming sufficient training data.

Salerno did a similar comparison and obtained favorable results as well [4]. In his experiment, he attempted to solve the XOR problem with both back-propagation and particle swarm optimization and showed that particle swarm optimization performed considerably better using fewer training epochs. He repeated the experiment using a recurrent neural network for the XOR problem and achieved similar results. His algorithm, however, did not seem to work with parsing natural language phrases.

Tsou and MacNish achieved the same result as Salerno when attempting to train a recurrent neural network for natural language parsing. They concluded that the standard particle swarm optimization algorithm is insufficient for converging in highly convex solution spaces as it simply converges to a local minimum [9]. Their comparison with back-propagation led to the development of an adaptive particle swarm optimizer, which implemented a learning rate such as the one found in back-propagation. Their implementation involved modeling particle swarm optimization update equations similar to Newton's laws of motion and introducing an explicit time step which served as the learning rate. Their modification was met with success and their analysis mentioned that performance was improved by reducing the step size of each particle around the minimum.

Van den Bergh and Engelbrecht introduced the concept of cooperative learning in particle swarm optimization for training neural networks [7]. Their approach divides the swarm into sub-swarms, for which the solution vector is then split among the different swarms via credit assignment. Inter-swarm communication occurs when the solution vector achieves a new lower error; all swarms are then updated to reflect this. Various experiments were conducted using Plain, Esplit, Nsplit, and Lsplit network architectures that achieved various levels of success and limitations.

Haberman and Sheppard introduced overlapping swarms in order to create a more energy-efficient routing protocol for sensor networks [10]. Their algorithm implements a best-of-the-best approach, where each particle is both the centroid of a swarm and part of neighboring swarms. Their approach showed success in increasing the lifetime of sensor networks, and outperformed state-of-the-art energy-aware routing protocols.

Ganesan Pillai and Sheppard introduced overlapping swarm intelligence, which builds on Haberman and Sheppard's work [6]. Their approach reduces the network into unique paths with each path represented as a sub-swarm. Each particle maintains a local view of the network as a whole, which is a union of their representation of their path and the rest of the network. Therefore, local evaluations result in treating the rest of the network as constant and evaluating only their fitness relative to other particles within the swarm. Inter-swarm communication occurs when the particles communicate their localized search results to overlapping swarms which are used to define the common vector.

Ganesan Pillai and Sheppard tested their approach against an artificial neural network trained with particle swarm optimization, in which the weights of the various layers were combined into a single weight vector and trained with a single swarm. Their overlapping swarm intelligence approach outperformed the particle swarm optimization based artificial neural network in all four datasets tested. The also compared their approach against a standard back-propagation algorithm and achieved equivalent or better success for the same datasets, and performed better for their deep network test.

Our version of this algorithm will introduce sampling in order to reduce the number of fitness evaluations incurred. We will employ the same datasets and experimental set up in order to compare both algorithms. Our comparison will be focused on comparing only against standard overlapping swarm intelligence since Ganesan Pillai and Sheppard already showed equivalent or better success against other algorithms.

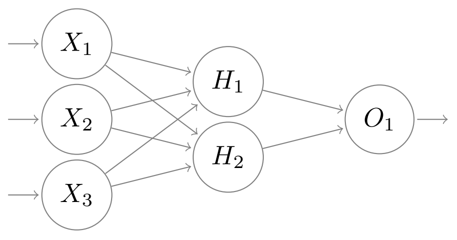

Fig. 1. A simple feed-forward neural network with three inputs, a hidden layer with two neurons, and a single output.

III. Background

A. Artificial Neural Networks

Artificial neural networks, inspired by a simplistic model of the brain, consist of a collection of neurons in layers that are used to perform a non-linear mapping from one layer to another. They have a wide variety of uses, including classification, pattern recognition, function approximation, optimization and much more.

A neuron is a single unit within a network that takes a number of input signals and their associated weights and transforms them via an activation function:

$$f(\mathbf{x}) \;=\; \mathbf{W}\cdot\mathbf{x}+\mathbf{b}$$

where $\mathbf{W}$ is a matrix representing the weights of the input edges, $\mathbf{x}$ is the input vector and $\mathbf{b}$ is the associated bias vector. The activation function is typically the logistic sigmoid, which gives an output in the domain $[0,1]$, or the hyperbolic tangent, which has a domain of $[-1,1]$.

Feed forward neural networks are networks with an input layer, a hidden layer and an output layer. While this network may have several hidden layers, it has been shown that a single hidden layer with enough neurons can approximate any continuous function [11], even though it is not very efficient. A simple feed-forward neural network is shown in Fig.1. This network has three inputs, a single hidden layer with two neurons, and a single output node. This network can be used for regression or binary classification problems.

Feed forward neural networks are usually trained using a gradient descent algorithm, typically the back-propagation algorithm, which consists of two phases; the forward pass is the calculation of the output values of each node given a set of inputs; and the backward propagation, which is the calculation of the gradient of error with respect to the expected output values and subsequent propagation of this backwards through the network and adjusting the weights as necessary, as defined as follows:

$$w_{ij}(t) \;=\; w_{ij}(t)+\alpha\cdot\Delta w_{ij}(t-1).$$

Here, α is a user-defined learning rate. Various forms of the algorithm exist, such as stochastic gradient descent, conjugate gradient descent and scaled conjugate gradient descent. The error value propagated back is typically a derivative of the error function, which can be computationally expensive depending on the application and error function.

B. Particle Swarm Optimization

Particle swarm optimization is a branch of swarm intelligence that is designed to mimic the social behavior of birds within a flock. Each individual of the swarm, known as a particle, is influenced by itself and the rest of the swarm, thereby driving the swarm as a whole towards its objective.

The basic particle swarm optimization algorithm maintains a collection of particles, where each particle represents a solution to the problem. Each particle is given an initial position in the solution space and is "flown" through the space via the velocity parameter. Each iteration in the algorithm updates the position and velocities of the particles, as follows:

$$x_{t+1} = x_t + v_{t+1}$$
$$v_{t+1} = \omega \cdot v_t + \phi_1 \cdot (y_t - x_t) + \phi_2 \cdot (\hat{y}_t - x_t)$$

where

$$\phi_1 = c_1 \cdot r_1$$
$$\phi_2 = c_2 \cdot r_2$$

The parameters ϕ_1 and ϕ_2 represent the coefficients of the cognitive and social components respectively, where c_1 and c_2 are user-defined parameters that control the components, and r_1 and r_2 are random vectors that drive the stochastic process. ω is the inertia weight, which is used to control the impact of the previous velocity. The cognitive component represents the individual particle's best position y_t at time t, whereas the social component represents the swarm's best position $\hat{y}_t$ at the same time. Each particle is evaluated against its objective, and the best particle is stored for that iteration. The coefficients for both the cognitive and social component help fly the swarm through the solution space. A larger cognitive component coefficient will favor more exploitative search whereas a larger social component coefficient will favor more exploratory search.

Particle swarm optimization has been shown to be an efficient alternative to back-propagation for feed-forward neural networks [12]. Each particle represents the weights of the network which are then evolved as the algorithm progresses. There have been many improvements to the basic algorithm such as the addition of the inertia weight, constriction coefficients, velocity clamping, etc.

C. Overlapping Swarm Intelligence (OSI)

Overlapping swarm intelligence is a novel adaptation of particle swarm optimization for training feed-forward neural networks that decomposes the network into unique paths from input to output [6]. The aim of the algorithm is to divide the problem into sub-problems, one for each path in the network. The algorithm learns each sub-problem and exploits the overlapping paths between the swarms.

Each path in the network is represented by a swarm. As such, the particles in the swarm cooperate to find a better solution in each generation. Each particle maintains a vector for the path it is on and constructs a local neural network using a global vector that represents the network as a whole. This global vector, gvn, facilitates inter-swarm communication. At each generation, the best weights from each path is used to construct this vector, which is then used in future generations.

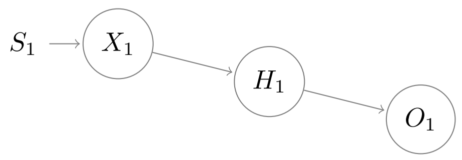

Fig. 2. Swarm S_1, showing the path from X_1 to H_1 to O_1.

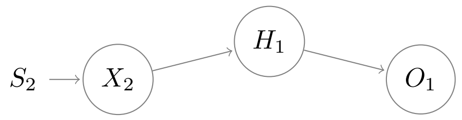

Fig. 3. Swarm S_2, showing the path from X_2 to H_1 to O_1.

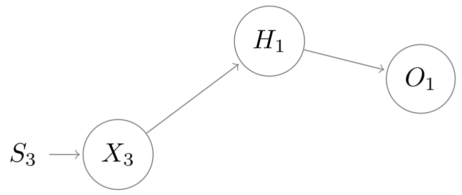

Fig. 4. Swarm S_3, showing the path from X_3 to H_1 to O_1.

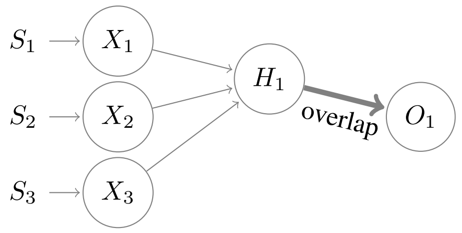

Fig. 5. Swarms S_1, S_2, and S_3, showing the overlap from H_1 to O_1. This overlap allows swarms to compete with other in order to obtain the best network weight.

As an example, we shall decompose the network from Fig.1. This network can be decomposed into the following six paths:

$$S_1 = X_1 \rightarrow H_1 \rightarrow O_1$$
$$S_2 = X_1 \rightarrow H_2 \rightarrow O_1$$
$$S_3 = X_2 \rightarrow H_1 \rightarrow O_1$$
$$S_4 = X_2 \rightarrow H_2 \rightarrow O_1$$
$$S_5 = X_3 \rightarrow H_1 \rightarrow O_1$$
$$S_6 = X_3 \rightarrow H_2 \rightarrow O_1$$

The decomposition of Swarms S_1, S_3, and S_5 is shown in Fig.2, Fig.3 and Fig.4 respectively, with the overlap shown in Fig.5. Overlap occurs between swarms $\{S_1, S_3, S_5\}$, and $\{S_2, S_4, S_6\}$, as shown above. Position and velocity updates are the same as in basic particle swarm optimization, with y_t representing each particle's best position, and $\hat{y}_t$ representing the swarm's best position.

For cooperative split particle swarm optimization, the num-

Algorithm 1 Dynamic Sampling in OSI

```
1: Initialize $gvn$
2: Initialize $swarms$
3: repeat
4:     Let $S_{sampled}$ be a sampled subset of $swarms$
5:     for $s_i \in S_{sampled}$ do
6:         for $p_j \in s_i$ do
7:             Construct local network $pnn_{ij}$
8:             Evaluate fitness $f(pnn_{ij})$
9:             if $f(pnn_{ij}) < f(p_{ij})$ then
10:                Update personal best for $p_{ij}$
11:            end if
12:            if $f(pnn_{ij}) < f(s_i)$ then
13:                Update swarm best for $s_i$
14:            end if
15:            Reconstruct global vector $gvn$
16:            Update position and velocity
17:        end for
18:    end for
19: until termination
```

ber of fitness evaluations, N_f, is

$$N_f = N_s \cdot N_p \cdot E.$$

where, N_s, N_p and E are the number of swarms, number of particles per swarm, and number of training epochs, respectively [7]. For overlapping swarm intelligence, the number of fitness evaluations is similar except that N_s is defined by the number of unique paths in the network. Thus the number of fitness evaluations explodes as the number of neurons and layers grow.

D. Dynamic Sampling of Swarms

We hypothesize that by sampling a subset of swarms in each generation we can achieve the same performance as OSI with fewer fitness evaluations. This is because not every swarm may provide a better weight candidate as compared to the previous generation; therefore, rather than iterate over every swarm, a different subset of swarms in every generation can accomplish the same goal.

Pseudo-code of the algorithm is presented in Algorithm 1. One important distinction between this and the previous implementation of OSI is that asynchronous updates are applied during each generation. That is, after each particle is evaluated and updated, the swarm's best weights are immediately propagated to other particles and swarms. Research has shown that incremental, asynchronous updates work better for local best problems, which is similar to overlapping swarms, and results in faster solutions [13].

The following sampling methods were explored:

1) RANDOM SAMPLING - swarm paths are randomly sampled from a discrete uniform distribution at each generation. For example, the first generation might sample swarms $\{S_1, S_3\}$ for a subset size of 2. Subsequent generations might sample $\{S_6, S_3\}$ and $\{S_4, S_5\}$. Recognizing that evaluating all paths every generation will not provide a better network, we expect that sampling a small fraction of the total paths will train a network to obtain approximately similar results.
2) ITERATIVE SAMPLING - swarm paths are sampled iteratively over the course of the algorithm. For example, the first generation will sample $\{S_1, S_2\}$ for a subset size of 2. Subsequent generations will sample $\{S_3, S_4\}$, $\{S_5, S_6\}$, and back to $\{S_1, S_2\}$. This sampling method is similar to random sampling but takes out the randomness and attempts to exploit an ordered approach to training the weights of a network.
3) WORST-FIRST SAMPLING - swarm paths are sampled inversely proportional to their performance. Each generation keeps a track of each swarm's performance metric, which is then used to select swarms for the next generation. Swarms with poor performance have a higher probability of being selected in the next generation. This method introduces a heuristic approach to swarm path selection. Since our algorithm trains specific paths in the network while keeping the rest of the network constant, we focus training on those paths that individually perform poorly with the expectation that incrementally improving poor paths in the network will improve the network as a whole.

IV. EXPERIMENTAL SETUP

Since we are comparing the performance of dynamic swarms to standard OSI with the aim of reducing fitness evaluations, we modified the original algorithm to keep track of fitness evaluations. Several experiments are then performed on two-layer and four-layer feed-forward neural networks. These experiments were based on a similar setup described in [6], and are further described in the following paragraphs.

For all experiments, the initial weights were set randomly in the range $[-3, 3]$ and the initial velocities were set randomly in the range $[-2, 2]$. During position and velocity updates, the values were also restricted to these domains. This was done to prevent position and velocity vectors from straying too far from the search space. The inertia weight ω was set to 0.729 and the values of c_1 and c_2 were set to 1.49445. r_1 and r_2 were sampled from a uniform distribution for each update.

Each experiment was run using 5×2 cross-validation, with 33% of the training data being used as a validation set to prevent over-fitting. Our performance metric is mean-squared error and our termination criteria is based on a linear regression of mean-squared error values obtained from the validation set based on a pre-defined window size. When the slope of the regression is positive, the training stops and mean-squared error and classification accuracy is calculated on the test data. No normalization or scaling of data was done on the datasets, which simplifies the training process.

The first experiment was conducted on the IRIS dataset [14]. This is a classification problem with 4 features and 3 classes, with one class linearly separable from the other two. We used a

4-input, 3-hidden, 3-output layer network for this problem, for a total of 48 paths. The second experiment was conducted on the IONOSPHERE dataset [14]. This is a classification problem with 34 features and 2 classes. We used a 34-input, 5-hidden, 2-output layer network for this problem, for a total of 352 paths.

The third experiment was conducted on the GLASS dataset [14]. This is a classification problem with 9 features and 6 classes and a highly skewed distribution. We used a 9-input, 6-hidden, 6-output layer network for this problem, for a total of 366 paths. The fourth experiment was conducted on a deep network using a generated dataset called 4-BIT. 1000 points of data were generated, each with 4 inputs, one for each bit. Each bit is treated as a 1 if its value is greater than 0.5, or 0 otherwise. Thus, the output value of each data point is a 1 if the number of 1s is even, or 0 otherwise. We used a 4-input, 4-hidden, 3-hidden, 2-hidden, 2-output layer network for this problem, for a total of 258 paths. All of the above networks include bias nodes and weights.

For our overlapping swarm intelligence implementation, we used $\{5, 10, 15, 20\}$ for the number of particles per swarm and $\{5, 10, 15, 20, 25, 30\}$ for the window sizes, which is a sliding window used for our linear regression termination criteria. For our dynamic swarm implementation, we used the same criteria for window sizes and particles per swarm, and additionally, we used $\{10\%, 15\%, 20\%, 25\%, 30\%, 35\%, 40\%\}$ for the size of the dynamic sample subset.

V. RESULTS

The results of our experiment are summarized in Tables 1 through 4. The columns represent the algorithms tested on each dataset. The rows represent the minimum mean-squared error and its corresponding classification accuracy, number of evaluations, number of swarms, window size, and the number of particles per swarm. Both mean-squared error and accuracy show 95% confidence intervals in parentheses. We also list the fraction of the number of evaluations in dynamic swarms relative to standard OSI, in parentheses, for comparison. The number of swarms used in dynamic swarms is similarly listed as a fraction in parentheses.

A difference of means test is performed between standard OSI and all dynamic sampling methods for mean-squared error and classification accuracy. Results for dynamic sampling are bold if they are statistically significantly different from standard OSI results ($p < 0.05$).

For the IRIS dataset results in Table 1, we observed that random, worst-first and iterative sampling methods performed similarly to the OSI counterpart in terms of mean-squared error, with random and worst-first sampling methods using less than half the number of fitness evaluations required. Performance was similar in terms of classification accuracy. Among the dynamic swarms, the worst-first sampling method performed the best with a mean-squared error score of 0.0367 and accuracy score of 0.9787.

For the IONOSPHERE dataset results in Table 2, we also observed that all dynamic swarms performed similarly in terms of mean-squared error to OSI with less than half the number fitness evaluations. Performance was also very similar in terms of classification accuracy. Among the dynamic swarms, the worst-first sampling method performed the best with a mean-squared error score of 0.0773 and accuracy score of 0.8986.

For the GLASS dataset results in Table 3, we observed that all dynamic algorithms were similar to OSI in terms of mean-squared error. Performance was similar for classification accuracy. Among the dynamic swarms, the random sampling method performed the best in terms of mean-squared error with a score of 0.0933 and the worst-first sampling method performed the best in terms of accuracy with a score of 0.5879. Both worst-first and iterative sampling methods had significantly fewer fitness evaluations than OSI, while random sampling used only 18% less fitness evaluations.

For the 4-BIT dataset results in Table 4, we observed that only the random sampling method performed significantly worse than OSI in terms of mean-squared error, although it used 77% fewer fitness evaluations. Both worst-first and iterative sampling methods performed comparably to OSI in terms of mean-squared error with 36% and 10% fewer fitness evaluations respectively. There was no significant difference in accuracy scores among all algorithms. Among the dynamic swarms, the iterative sampling method performed the best in terms of mean-squared error and accuracy with a score of 0.2354 and an accuracy score of 0.6148, respectively.

VI. DISCUSSION

The results of the IRIS dataset show that dynamic swarms perform similarly to OSI in terms of mean-squared error with less than half the number of fitness evaluations. Both random and worst-first sampling methods performed as well as OSI with less than half the number of fitness evaluations. The worst-first sampling method seems better in performance compared to the two sampling methods; although, the differences are not statistically significant. The iterative sampling method required considerably more fitness evaluations than other dynamic swarms but still used 27% less than OSI. We speculate that a larger window and smallar particles size per swarm, combined with the stochastic nature of the algorithm resulted in a higher number of fitness evaluations. Using 35% of the total swarms with a window and particle size of 25 and 15 respectively, the iterative sampling method obtained a mean-squared error score of 0.0379 and an accuracy score of 0.9760 using only 173,383 evaluations, which is 62% less than OSI.

The results of the IONOSPHERE dataset also show that all dynamic swarms perform similarly to OSI in terms mean-squared error. All swarms use at least half the number of fitness evaluations as compared to OSI. Here again, the worst-first sampling method appears to outperform all other methods with better mean-squared error and accuracy numbers, and fewer fitness evaluations; although, the differences are not statistically significant. It was also observed that this dataset required fewer fitness evaluations in general, when compared to the IRIS dataset. The best results indicate that smaller

TABLE I
IRIS DATASET RESULTS

	OSI	Random	Worst-First	Iterative
MSE	0.0363 ± 0.004	0.0374 ± 0.004	0.0367 ± 0.003	0.0371 ± 0.004
Accuracy	0.9720 ± 0.010	0.9747 ± 0.011	0.9787 ± 0.006	0.9707 ± 0.011
Evaluations	454,636.5	185,233.0 (40.7%)	164,277.7 (36.1%)	333,337.9 (73.3%)
Num Swarms	48	17 (35%)	19 (40%)	19 (40%)
Window	20	30	20	30
Particles	15	20	15	10

TABLE II
IONOSPHERE DATASET RESULTS

	OSI	Random	Worst-First	Iterative
MSE	0.0817 ± 0.011	0.0811 ± 0.010	0.0773 ± 0.013	0.0790 ± 0.008
Accuracy	0.8894 ± 0.014	0.8906 ± 0.010	0.8986 ± 0.018	0.8980 ± 0.012
Evaluations	155,247.7	76,404.7 (49.2%)	70,235.6 (45.2%)	75,553.7 (48.7%)
Num Swarms	352	123 (35%)	141 (40%)	123 (35%)
Window	10	15	10	20
Particles	15	15	15	10

TABLE III
GLASS DATASET RESULTS

	OSI	Random	Worst-First	Iterative
MSE	0.0916 ± 0.004	0.0933 ± 0.003	0.0938 ± 0.006	0.0950 ± 0.004
Accuracy	0.5953 ± 0.029	0.5785 ± 0.048	0.5879 ± 0.055	0.5570 ± 0.026
Evaluations	412,886.6	339,011.2 (82.1%)	182,250.6 (44.1%)	265,381.3 (64.3%)
Num Swarms	366	128 (35%)	146 (40%)	110 (30%)
Window	15	30	20	30
Particles	15	20	15	20

TABLE IV
4-BIT DATASET RESULTS

	OSI	Random	Worst-First	Iterative
MSE	0.2329 ± 0.005	$\mathbf{0.2418 \pm 0.007}$	0.2396 ± 0.009	0.2354 ± 0.005
Accuracy	0.6146 ± 0.020	0.5846 ± 0.027	0.5856 ± 0.040	0.6148 ± 0.026
Evaluations	1,081,641.8	248,945.8 (23.0%)	697,274.2 (64.5%)	977,914.3 (90.4%)
Num Swarms	258	77 (30%)	103 (40%)	103 (40%)
Window	30	30	30	30
Particles	20	20	20	20

window and particle sizes were more suited to this problem, leading to fewer fitness evaluations.

Our dynamic sampling methods also showed similar performance in the GLASS dataset experiment. All sampling methods achieved similar mean-squared error and accuracy scores with fewer fitness evaluations. The worst-first sampling method outperformed the others with considerably fewer fitness evaluations and a better accuracy score. It was observed that both random and iterative sampling methods achieved a better accuracy score with larger window and particle sizes, at the expense of fitness evaluations. Our exhaustive grid search results show that for a slightly higher mean-squared error score, both sampling methods can achieve similar accuracy scores for considerably fewer fitness evaluations.

Finally, all dynamic swarms, with the exception of the random sampling method, performed similarly to OSI in the 4-BIT dataset experiment. It was observed that all methods generally performed their best with larger window and particle sizes, and for dynamic swarms, larger samples of swarms. In general, dynamic swarms did better with a larger number of particles, swarms and bigger window. The random sampling method achieved a mean-squared error slightly larger and statistically significantly different from OSI, and we suspect that the swarms ended up at a local minimum and terminated early given the significantly fewer number of fitness evaluations. Other sampling methods were able to achieve better success with more fitness evaluations. The iterative sampling method required considerably more fitness evaluations than its dynamic swarm counterparts, but was able to achieve better scores with only a 10% reduction in evaluations. For a slightly higher mean-squared error, our results show a near 50% reduction in fitness evaluations and a slightly lower accuracy score.

The worst-first sampling method performed well compared to the other two sampling methods, which suggests that there is potential in a heuristic based approach to sampling the swarms. Although, when the algorithm was instrumented to record which swarm paths were selected, a histogram showed approximately the same frequency for all paths. While all paths were not selected equally, this might suggest that while some paths in the network are optimized slightly more than others, all other paths are equally as important. The iterative method also performed surprisingly well, although usually with more fitness evaluations. We speculate that iterating through all paths in an orderly fashion may lead to some generations not having any improvement at all, which may be why a heuristic based approach might work better.

VII. Conclusions

Overlapping swarm intelligence has been demonstrated to improve accuracy on certain datasets and on certain network architectures, but suffers from an explosion of fitness evaluations as the number of neurons and layers grow. We demonstrated that by applying asynchronous updates and using a smaller subset of the swarms per generation that we can achieve similar results with a significant reduction in fitness evaluations. We demonstrated various versions of dynamic swarms with good success on a variety of datasets and networks, including one deep network.

For future work, we plan on fine-tuning the dynamic selection of swarms for each generation. Specifically, we will explore adaptive selection of swarms where the size of the swarm may vary for each generation according to some heuristic. This has the benefit of broadening the search process when the algorithm perceives it to be necessary, and conversely, decreasing the number of swarms during rapid improvements in order to restrict the number of fitness evaluations. We will also explore other heuristic methods that guide the swarm selection process, such as considering competing swarms' scores and neighboring overlapping paths. Finally, we would like to extend this training process to other types of networks, such as recurrent neural networks and larger, deep belief networks.

Acknowledgments

We would like to thank Karthik Ganesan Pillai from Montana State University, and Stephyn Butcher from Johns Hopkins University for their help on early versions of this paper.

References

[1] M. Gori and A. Tesi, "On the problem of local minima in backpropagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, p. 76, 1992.

[2] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions On Neural Networks / A Publication Of The IEEE Neural Networks Council*, vol. 5, no. 2, p. 157, 1994.

[3] X. Cai, N. Zhang, G. K. Venayagamoorthy, and D. C. Wunsch, "Time series prediction with recurrent neural networks using a hybrid pso-ea algorithm," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol. 2, 2004, pp. 1647–1652.

[4] J. Salerno, "Using the particle swarm optimization technique to train a recurrent neural model," in *Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence*, 1997, pp. 45–49.

[5] J.-R. Zhang, J. Zhang, T.-M. Lok, and M. R. Lyu, "A hybrid particle swarm optimizationback-propagation algorithm for feedforward neural network training," *Applied Mathematics and Computation*, vol. 185, no. 2, pp. 1026–1037, 2007.

[6] K. G. Pillai and J. W. Sheppard, "Overlapping swarm intelligence for training artificial neural networks," in *IEEE Symposium on Swarm Intelligence (SIS)*, 2011, pp. 1–8.

[7] den Bergh Van and A. P. Engelbrecht, "Cooperative learning in neural networks using particle swarm optimizers: research article," *South African Computer Journal*, vol. 20, no. 26, p. 84, 2000.

[8] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks," in *Proceedings of the IEEE Swarm Intelligence Symposium*, 2003, pp. 110–117.

[9] D. Tsou and C. MacNish, "Adaptive particle swarm optimisation for high-dimensional highly convex search spaces," in *The Congress on Evolutionary Computation (CEC)*, vol. 2, 2003, pp. 783–789 Vol.2.

[10] B. Haberman and J. Sheppard, "Overlapping particle swarms for energy-efficient routing in sensor networks," *Wireless Networks (10220038)*, vol. 18, no. 4, p. 351, 2012.

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

[12] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS)*, 1995, pp. 39–43.

[13] A. Carlisle and G. Dozier, "An off-the-shelf pso," in *Proceedings of the Workshop on Particle Swarm Optimization*, 2001.

[14] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml