DOSI: Training Artificial Neural Networks using Overlapping Swarm Intelligence with Local Credit Assignment

Nathan Fortier
Department of Computer Science
Montana State University
EPS 357, PO Box 173880
Bozeman MT, 59717-3880
Email: nathan.fortier@msu.montana.edu

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

Karthik Ganesan Pillai
Department of Computer Science
Montana State University
EPS 357, PO Box 173880
Bozeman MT, 59717-3880
Email: k.ganesanpillai@cs.montana.edu

***Abstract*—A novel swarm-based algorithm is proposed for the training of artificial neural networks. Training of such networks is a difficult problem that requires an effective search algorithm to find optimal weight values. While gradient-based methods, such as backpropagation, are frequently used to train multi-layer feedforward neural networks, such methods may not yield a globally optimal solution. To overcome the limitations of gradient-based methods, evolutionary algorithms have been used to train these networks with some success. This paper proposes an overlapping swarm intelligence algorithm for training neural networks in which a particle swarm is assigned to each neuron to search for that neuron's weights. Unlike similar architectures, our approach does not require a shared global network for fitness evaluation. Thus the approach discussed in this paper localizes the credit assignment process by first focusing on updating weights within local swarms and then evaluating the fitness of the particles using a localized network. This has the advantage of enabling our algorithm's learning process to be fully distributed.**

I. Introduction

While the backpropagation (BP) algorithm has been shown to be an effective method for training feedforward neural networks, it typically has a slow convergence rate [1] and is known to suffer from local minima [2]. To overcome these limitations alternative approaches such as particle swarm optimization (PSO) algorithms, genetic algorithms, and hybrid approaches which use backpropagation combined with evolutionary approaches have been used. This paper proposes an overlapping swarm intelligence algorithm to train feedforward neural networks using localized particle swarms. In our approach a particle swarm is associated with each neuron in the network. Swarms corresponding to neurons that are directly connected in the network will periodically communicate during the training process.

The communication scheme utilized in our approach is the main contribution of this paper. Specifically, existing approaches require localized swarms to interact with a single, global network to evaluate fitness and, thereby, determine updates to the weights. Our approach, on the other hand, associates local networks for fitness evaluation that propagate through a system of localized networks. Ultimately, the set of localized networks converge to highly-fit networks that have benefited from the PSO process training the weights. In effect, we have extended the approach to using localized swarms to "solve" the local credit assignment problem to enable the entire learning process to be distributed. We have chosen to name this algorithm Distributed Overlapping Swarm Intelligence (DOSI).

II. Background

A. Artificial Neural Networks

Artificial neural networks are models consisting of a network of simple computational units called neurons. When neurons are combined using appropriate weights the resulting networks have been shown to solve difficult problems such as classification, regression, control, and time series prediction.

Single layer feedforward neural networks have several inputs that feed a single output layer. These inputs are connected via weights, and only the output layer has neurons. Multilayer feedforward neural networks, on the other hand, have one or more hidden layers between the input and output layers that also contain neurons. Hidden layers allow neural networks to extract higher-order properties from the input. Each neuron in a network contains a continuously differentiable activation function. One of the most common activation functions is the logistic function,

$$y = \frac{1}{1 + e^{-(\sum_{i=1}^{m} w_i x_i + b)}}$$

where m is the number of inputs, x_i is the i^{th} input, w_i is its weight, and b is a bias.

B. Particle Swarm Optimization

The Particle Swarm Optimization (PSO) algorithm, first proposed by Eberhart and Kennedy [3], is a search technique based on the social behavior of flocking birds and fish schools. PSO is a population-based search technique in which the population is initialized with random solutions called particles. The search process updates the positions of each particle-based on that particle's corresponding velocity vector. A particle's velocity vector is updated based on the fitness of the states visited by that particle. Eventually all particles will move closer to an optimum in the search space. The pseudocode for the traditional PSO algorithm is presented in Algorithm 1.

PSO begins by randomly initializing a swarm of particles over the search space. On each iteration of the algorithm the fitness of a particle, x_i, is calculated using the fitness function, $f(x_i)$. The personal best position for that particle is stored

978-1-4673-2743-5/12/$31.00 ©2012 IEEE

Algorithm 1 Particle Swarm Optimization

repeat
  for each particle position $x_i \in \mathbf{P}$ **do**
    Evaluate position fitness $f(x_i)$
    if $f(x_i) > f(p_i)$ **then**
      $p_i \leftarrow x_i$
    end if
    if $f(x_i) > f(p_g)$ **then**
      $p_g \leftarrow x_i$
    end if
    $v_i \leftarrow \omega v_i + \mathrm{U}(0, \phi_1) \otimes (p_i - x_i) + \mathrm{U}(0, \phi_2) \otimes (p_g - x_i)$
    $x_i \leftarrow x_i + v_i$
  end for
until termination criterion is met

in the vector p_i. The global best position found among all particles is stored in the vector p_g. At the end of each iteration a particle's velocity, v_i, is updated based on p_i and p_g. The use of both personal best and global best positions in the velocity equation ensures diverse responses within the swarm. This is an important aspect of the algorithm that provides a balance between exploration and exploitation.

In Algorithm 1, $\mathbf{P}$ is the particle swarm, $\mathrm{U}(0, \phi_i)$ is a vector of random numbers uniformly distributed in the interval $[0, \phi_i]$, $\otimes$ is component-wise multiplication, v_i is the velocity of a particle and x_i is the position of a particle.

Three parameters need to be defined for this algorithm:

- ϕ_1 determines the maximum force with which a particle is pulled toward p_i;
- ϕ_2 determines the maximum force with which a particle is pulled toward p_g;
- ω is the inertia weight.

The inertia weight ω is used to control the scope of the search and eliminate the need for a maximum velocity. Even so, it is customary to specify maximum velocity as well.

III. Related work

Zhang *et al.* [4] introduced a PSO algorithm to learn the structure and weights of feedforward neural networks. They tested their algorithm on two problems in the medical domain and their results indicate that networks learned by their algorithm have good generalization ability and accuracy.

Zhang *et al.* [1] proposed a hybrid PSO-BP algorithm to search for the weights of feedforward neural networks. In their algorithm a feedforward neural net is trained using a PSO until convergence. After the PSO algorithm has converged, backpropagation is used to train the network. Their results indicate that their algorithm is better than the backpropagation algorithm and the adaptive particle swarm optimization algorithm in terms of convergence speed and accuracy.

Cai *et al.* [5] introduced an algorithm to train recurrent neural networks to predict missing values from time series data using a hybrid of PSO and a customized evolutionary algorithm (EA). The EA used in Cai's approach is similar to an evolution strategy as it has both mutation and selection but no crossover operation. In this algorithm a population of solutions is initialized and evaluated. Individuals are then selected based on their fitness and these individuals are improved using PSO. The improved individuals are then used as parents in the EA to generate offspring.

Bergh and Engelbrecht [6] proposed a several methods to train feedforward neural networks using PSOs in a cooperative configuration. These methods include NSPLIT in which there is a single particle swarm for each neuron in the network and LSPLIT in which there is a swarm assigned to each layer of the network. All approaches described by Bergh and Engelbrecht require the use of a global network and their results indicated that performance of these swarms is dependent on the degree of interdependence between the variables.

Haberman and Sheppard [7] introduced an energy-efficient routing protocol for sensor networks based on overlapping particle swarms that ensures reliable path selection while minimizing the energy consumption for the route selection process. Their algorithm extends the life of the sensor network and was shown to perform significantly better than current energy-aware routing protocols.

Pillai and Sheppard [8] extended the work of [7] by developing an overlapping swarm intelligence algorithm for training the weights of deep artificial neural networks. In their approach the structure of the network is separated into paths where each path begins at an input node and ends at an output node. Each of these paths has a swarm that learns the weights for that path of the network. A common vector of weights is maintained across all swarms that describes a global view of the network. This vector is constructed by combining the weights of the best particles in each of the swarms. While this method was shown to outperform the backpropagation algorithm and the traditional PSO algorithm, it requires one swarm for every possible path through the network. As a result, the number of swarms required grows exponentially as the depth of the network increases, thus making their approach intractable for deep networks.

Our algorithm improves on the work of [8] and [6] by requiring a smaller number of swarms than the OSI method and by no longer requiring the construction of a global network for each swarm. We hypothesize that our algorithm will perform at least as well as the algorithms proposed by [8] and [6] in terms of mean squared error and classification error.

IV. Approach

Here we describe our approach to training feedforward neural networks based on the PSO algorithm but focusing on learning sub-problems similar to the approaches in [6], [7], and [8]. In our approach we associate a swarm with each neuron in the network. A neuron's corresponding swarm will learn only the weights associated with that neuron. Swarms corresponding to connected neurons will periodically communicate with each other. This approach is similar to the NSPLIT architecture proposed by [6] but unlike NSPLIT, our architecture does not require that each particle be evaluated as part of a global network. Instead, each swarm s_i has a personal neural network pnn_i and the weights learned by each swarm are communicated to the neighboring swarms and inserted into those swarm's pnn's through a periodic communication mechanism. Because our algorithm does not require a global network to be shared between the swarms, and localizes the credit assignment process, the learning process can be fully distributed. Algorithm 2 shows the pseudo-code for our DOSI algorithm.

The pseudocode in Algorithm 2 begins by initializing a random neural network. This network is set as the initial

Algorithm 2 The DOSI Algorithm

```
Initialize a neural network N
for each neuron, n ∈ N do
  Let s be the swarm associated with neuron n
  s.pnn ← N
end for

repeat
  for i ← 0 to iter do
    for each swarm s do
      for each particle, p ∈ s do
        Evaluate particle fitness f(p)
        if f(p) > p's personal best fitness then
          Update p's personal best position and fitness
        end if
        if f(p) > the global best fitness then
          Update global best position and fitness for s
        end if
        Update p's velocity and position
      end for
    end for
  end for

  for each neuron, n ∈ N do
    Let s be the swarm associated with neuron n
    Let w be the input weights of n in the s.pnn
    Set w to the object parameters of the best particle in s
  end for

  for each edge, e ∈ N do
    Let n_i and n_j be the neurons that are connected by e
    Let s_i and s_j be the swarms associated with n_i and n_j
    ShareWeights(s_i, s_j)
  end for
until termination criterion is met

Let F be the final network.
for each swarm s do
  Let p_g be the best particle in s
  Insert the weights learned by p_g into F
end for
```

Algorithm 3 The ShareWeights Function

```
Let s_i and s_j be the two swarms that are to share weights
Let pnn_i and pnn_j be the personal neural networks of s_i
and s_j respectively

for each weight w in the neural network do
  if s_i.δ_w > s_j.δ_w then
    Insert pnn_i's value for w into pnn_j
    Increment s_j.δ_w
  end if
  if s_j.δ_w > s_i.δ_w then
    Insert pnn_j's value for w into pnn_i
    Increment s_i.δ_w
  end if
end for
```

neural network for each swarm. While it would be possible to initialize a different neural network for each swarm, this would likely slow the algorithm's convergence time. Next the algorithm enters the learning procedure. This procedure begins by running $iter$ iterations of the evaluate and update step for each swarm where the variable $iter$ is a tunable parameter. Each swarm evaluates the fitness of a particle by setting the weights of that swarm's neuron inside the swarm's pnn to the particle's object parameters and evaluating the network's error. Once $iter$ iterations of the PSO algorithm have been run for each swarm, the swarms corresponding to connected neurons communicate via the ShareWeights function (Algorithm 3).

To share weights, each swarm keeps track of a variable δ_w for each weight w. These variables indicate how current each weight in the swarm's personal network is. A larger value of δ_w indicates that its value was learned more recently while a smaller value for δ_w indicates that the weight was learned less recently. A value of zero for δ_w indicates that no learning or sharing has taken place to modify this weight in the swarm's pnn, while a value of infinity indicates that the weight has been learned by the PSO algorithm. Values for δ_w are initially set to zero for all weights other than the weights learned by that swarm. Values for δ_w are set to infinity for all weights that the swarm is to learn. Algorithm 3 allows swarms to share weights that have been learned through either the PSO algorithm or through the sharing process with the swarms to which they are connected. Also, Algorithm 3 causes weights learned through the PSO algorithm to trickle through the network of swarms as the sharing occurs.

For the network shown in Figure 1 each particle has a two dimensional position vector. Each dimension represents one of the input weights for a swarm's corresponding neuron. For the network shown in Figure 1 the global best particle in s_2 has the position vector $(0.23, 0.24)$ while the global best particle in s_1 has the position vector $(0.11, 0.12)$.

Figure 1 presents an example of the ShareWeights function. Before the function executes each swarm's pnn only contains the weights learned by that swarm. δ_w for s_1, s_2, and s_3 is set to infinity for all weights to be learned by s_1, s_2, and s_3. δ_w for all other weights is set to zero. After $ShareWeights(s_1, s_3)$ is executed pnn_1 contains the weights learned by s_3 and pnn_3 contains the weights learned by s_1. Also, $s_1.\delta_w$ has now been incremented to one for each weight learned from s_3 and $s_3.\delta_w$ now been incremented to one for each weight learned from s_1. Once $ShareWeights(s_2, s_3)$ has been executed pnn_2 contains the weights learned by s_3 and pnn_3 contains the weights learned by s_2. Also, since $s_3.\delta_w > s_2.\delta_w$ for the input weights on node 1, pnn_2 now passes the weights learned from s_1 to s_3. Finally, $s_2.\delta_w$ has now been incremented to one for each weight learned from s_3 and $s_3.\delta_w$ has now been incremented to one for each weight learned from s_2.

V. Experimental Setup

To evaluate our algorithm, several experiments were performed on two-layered and four-layered feedforward neural networks. These experiments focused on comparing our algorithm with the OSI algorithm discussed in [8] and the LSPLIT and NSPLIT algorithms discussed in [6]. We did not compare with backpropagation or full PSO since previous work demonstrated the superiority of OSI on these problems [8]. In our experiments the initial weights were set randomly in

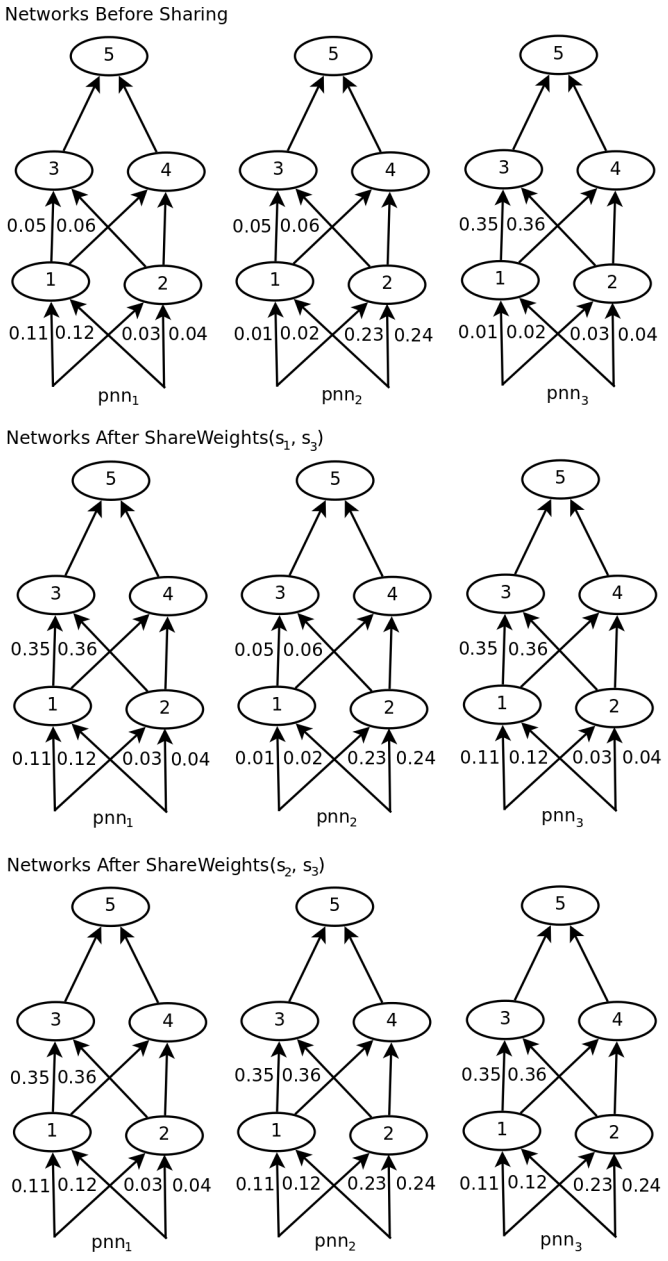

Fig. 1. Example of the ShareWeights function

the range of $[-3, 3]$ and the velocity for each particle was set randomly in the range of $[-2, 2]$. The inertia weight ω was set to 0.729 and both acceleration coefficients, ϕ_1 and ϕ_2, were set to 1.49445. These parameter values were based on the results of Eberhart and Shi [9]. To evaluate the performance of the algorithms the data was divided into training and testing data sets using a 5 x 2 cross-validation procedure. The experiments were repeated using several swarm sizes. The different swarm sizes used were 2, 3, 5, 7, 10, 13, 15, 17, 20.

For each swarm size, we performed pairwise comparisons using a paired student t-test for each pair of algorithms. These paired student t-tests were performed for both classification error and mean squared error to test for statistical significance. For all t-tests we used a 98% confidence interval. For comparison we use the same data sets and network structures as in [8].

For the first experiment the IONOSPHERE data set obtained from the UCI machine learning repository was used [10]. The data set contains only two classes, but has 34 input dimensions. This data set contains 351 instances. This experiment uses a two-layered neural network with 34 inputs, five hidden neurons, and one output neuron.

For the second experiment the IRIS data set was used. This data set contains three classes and has 150 instances. One class is linearly separable from the other two while the others are not linearly separable from one another. For this experiment a four-input, three-hidden, and one-output network was used.

For the third experiment the GLASS data set was used. This data set contains six classes and has 214 instances. Each instance has nine inputs. This data set is known to have a highly skewed class distribution, making it difficult to learn. For this data set a nine-input, six-hidden, and one-output network architecture was used.

For the fourth experiment we generated data for a four-bit parity problem using the same strategy as in [8]. The data set consisted of 1000 randomly generated data points. Each data point consisted of four inputs generated over the interval [0, 1] and one output. An input was interpreted to have a value of "1" if its value is greater than 0.5 and a value of "0" otherwise; although, the generated floating point values were used for training and testing. When the number of "1"s in all inputs was even, the output was set to "1"; otherwise, the output was set to "0".

For this data set a four-layered neural network with four inputs, four neurons per hidden layer, and one output neuron was used. Neural networks that have more than two layers, such as the ones to be used in this experiment, are called deep neural networks. Deep networks trained using backpropagation have been found to be much more difficult to train than networks with less than two hidden layers [11].

TABLE I
IRIS MEAN SQUARED ERROR FOR DIFFERENT SWARM SIZES

	Swarm Size								
Method	2	3	5	7	10	13	15	17	20
DOSI	0.1451	**0.0995**	0.1098	0.0675	0.0748	0.0508	0.0630	0.0561	0.0570
LSPLIT	0.1801	0.1634	0.1203	0.1066	0.0916	0.0846	0.0893	0.0770	0.0615
NSPLIT	0.1401	**0.1021**	0.0578	0.0726	0.0443	0.0435	0.0342	0.0324	0.0266
OSI	0.1137	**0.1065**	0.1214	0.0836	0.0802	0.0890	0.1003	0.0999	0.0928

TABLE II
IONOSPHERE MEAN SQUARED ERROR FOR DIFFERENT SWARM SIZES

	Swarm Size								
Method	2	3	5	7	10	13	15	17	20
DOSI	**0.1906**	0.1830	**0.1772**	**0.1676**	**0.1561**	**0.1527**	0.1468	0.1609	**0.1458**
LSPLIT	0.2407	0.2112	0.2146	0.2015	0.1935	0.1823	0.1853	0.1799	0.1855
NSPLIT	**0.2080**	0.2047	**0.1747**	**0.1667**	**0.1376**	**0.1314**	0.1365	0.1353	**0.1246**
OSI	**0.1566**	0.1606	**0.1579**	**0.1680**	**0.1496**	**0.1594**	0.1568	0.1551	**0.1619**

TABLE III
GLASS MEAN SQUARED ERROR FOR DIFFERENT SWARM SIZES

	Swarm Size								
Method	2	3	5	7	10	13	15	17	20
DOSI	0.1131	0.1189	0.1144	0.1321	0.1264	0.1167	0.1078	0.1056	0.1161
LSPLIT	0.1274	0.1174	0.1193	0.1191	0.1141	0.1231	0.1144	0.1298	0.1257
NSPLIT	0.1419	0.1147	0.1045	0.0972	0.0978	0.1074	0.0785	0.0800	0.0808
OSI	0.1047	0.1164	0.1149	0.1132	0.1069	0.1078	0.1173	0.1138	0.1109

TABLE IV
4BIT MEAN SQUARED ERROR FOR DIFFERENT SWARM SIZES

	Swarm Size								
Method	2	3	5	7	10	13	15	17	20
DOSI	0.2513	0.2503	0.2511	0.2521	0.2532	0.2624	0.2516	0.2512	0.2599
LSPLIT	0.2664	0.2541	0.2513	0.2504	0.2508	0.2501	0.2505	0.2522	0.2512
NSPLIT	0.2504	0.2504	0.2507	0.2504	0.2519	0.2496	0.2510	0.2516	0.2508
OSI	0.2468	0.2447	0.2460	0.2450	0.2471	0.2429	0.2502	0.2467	0.2477

TABLE V
IRIS CLASSIFICATION ERROR FOR DIFFERENT SWARM SIZES

	Swarm Size								
Method	2	3	5	7	10	13	15	17	20
DOSI	0.5444	0.1937	0.1187	0.0823	0.0373	0.0423	0.0542	**0.0263**	0.0474
LSPLIT	0.4384	0.1508	0.0831	0.0619	0.0545	0.0504	0.0421	0.0466	0.0415
NSPLIT	0.5035	0.2193	0.0415	0.0384	0.0405	0.0398	0.0450	**0.0251**	0.0330
OSI	0.3349	0.2093	0.1442	0.0652	0.1200	0.0599	0.0782	0.0476	0.0629

TABLE VI
IONOSPHERE CLASSIFICATION ERROR FOR DIFFERENT SWARM SIZES

	Swarm Size								
Method	2	3	5	7	10	13	15	17	20
DOSI	0.2237	**0.1544**	**0.0993**	**0.0982**	0.0792	0.0745	0.0875	0.0764	0.0915
LSPLIT	0.2407	0.2731	0.2458	0.1589	0.1339	0.1321	0.1040	0.1231	0.1027
NSPLIT	0.2327	**0.1480**	**0.0846**	**0.0859**	0.0950	0.0744	0.0738	0.0802	0.0790
OSI	0.1960	**0.1514**	**0.0969**	**0.0875**	0.0827	0.0816	0.0851	0.0801	0.0729

TABLE VII
GLASS CLASSIFICATION ERROR FOR DIFFERENT SWARM SIZES

	Swarm Size								
Method	2	3	5	7	10	13	15	17	20
DOSI	0.7698	0.7472	0.6540	0.6586	0.5969	0.5773	0.6455	0.6545	0.5662
LSPLIT	0.7974	0.7642	0.7056	0.6425	0.6136	0.6102	0.5920	0.6018	0.6045
NSPLIT	0.7821	0.7133	0.5918	0.5670	0.5650	0.5619	0.5630	0.5998	0.5515
OSI	0.7555	0.7165	0.7108	0.6513	0.5967	0.6666	0.5875	0.5914	0.6323

TABLE VIII
4BIT CLASSIFICATION ERROR FOR DIFFERENT SWARM SIZES

	Swarm Size								
Method	2	3	5	7	10	13	15	17	20
DOSI	0.4727	0.4794	0.4843	0.4900	0.4982	0.4970	0.4916	0.5060	0.5031
LSPLIT	0.4863	0.4772	0.4764	0.4795	0.5131	0.5085	0.5057	0.5066	0.5151
NSPLIT	0.4700	0.4701	0.4780	0.4833	0.5047	0.5119	0.5016	0.5130	0.4992
OSI	0.4464	0.4395	0.4250	**0.4088**	0.4432	0.4628	0.4560	**0.4088**	**0.4309**

VI. EXPERIMENTAL RESULTS

Our experimental results are summarized in Tables I through VIII. In these tables, each row denotes the method used and each column denotes the swarm size used. For all tables, bold values indicate that the corresponding algorithm's performance is statistically significantly better than the other algorithms on the data set given the corresponding swarm size. Algorithms that tie statistically for best are bolded. No values for a given swarm size are bolded if all algorithms tie statistically for best. Tables I through IV present the minimum mean squared error on the test data while tables V through VIII present the classification error.

For the IRIS data set, we observe that, based on the paired t-tests on mean squared error, none of the algorithms significantly outperform the others for most swarm sizes. However, when the swarm size was set to 3, all other algorithms outperformed LSPLIT. The paired t-tests on classification error also indicate that none of the algorithms significantly outperform the other for most swarm sizes. However, for swarm size 17 DOSI and NSPLIT outperformed LSPLIT and OSI in terms of classification error.

With the IONOSPHERE data set, based on the paired t-tests performed on mean squared error, we observe that DOSI, NSPLIT, and OSI all significantly outperform LSPLIT for most swarm sizes. Based on the paired t-tests performed on classification error, we observe that DOSI, NSPLIT, and OSI significantly outperform LSPLIT when the swarm size is set to 3, 5, or 7.

With the GLASS data set, based on the paired t-tests performed on mean squared error and classification error, we observe that none of the algorithms significantly outperform the other.

For the 4BIT data set, based on the paired t-tests performed on mean squared error, we observe that none of the algorithms significantly outperform the others for the various swarm sizes. However, the paired t-tests performed on classification error indicate that OSI outperformed the other three methods when the swarm size was set to 7, 17, and 20. This may be because the OSI algorithm provides better generalization on the 4BIT problem than our algorithm. However, the paired t-tests on classification error and mean squared error both indicate that DOSI performed statistically equivalent to LSPLIT and NSPLIT.

VII. DISCUSSION

The paired t-tests on mean squared error indicate that DOSI performed either equal to or better than the other methods on all data sets studied. The paired t-tests on classification error indicate that DOSI performed either equally to or better than the other methods on all data sets studied except for the 4BIT data set. On the 4BIT data set OSI statistically outperformed DOSI when the swarm size was set to 7, 17, and 20. We hypothesize that OSI's superior performance is due to the competition between overlapping swarms. It would be interesting to test this hypothesis by adding similar inter-swarm competition to DOSI to determine if it improves performance. We plan to explore this mechanism in future work.

Although DOSI had slightly higher classification error than OSI in some of the 4BIT cases, it has much lower computational complexity since it only requires a swarm for each neuron rather than requiring a swarm for every possible path through the network. This lower complexity allows DOSI to be used as a learning technique on much larger neural networks. Also, the lack of a global network allows the learning process to be distributed across multiple machines. Such distribution is desirable for large neural networks but is complicated by the global network when considering any of the other methods examined here.

VIII. CONCLUSIONS AND FUTURE WORK

In the paper by Pillai and Sheppard [8] it was shown that the OSI method is effective for training feedforward neural networks with some indication that it would also perform well on deep networks. Unfortunately, their approach requires a swarm for every path in the network. Using this approach, the number of swarms, S, that would need to be generated and managed is

$$S = P = I \cdot N^H \cdot O.$$

where P is the number of paths through the network, I is the number of inputs, N is the number of neurons per hidden layer, H is the number of hidden layers, and O is the number of outputs. Thus, as layers are added to a network the number of paths through the network grows exponentially.

The NSPLIT method [6], on the other hand, is very fast, given the fact local swarms are specified only for each individual node. However, the experimental results demonstrated that, while NSPLIT is effective on shallow networks, as network complexity increased, the effectiveness of the approach tended to break down.

In this paper, we succeeded in combining the advantages of both OSI and NSPLIT, first by enabling a sharing of information between the local swarms, and second by managing the associated computational complexity. The former enabled the training of various types of feedforward networks (including

the deep network for the 4BIT problem), and the latter made the associated process tractable. Furthermore, by focusing on a local fitness evaluation rather than requiring a shared, global network, our resulting DOSI algorithm enables the entire learning process to be fully distributed. As a result, DOSI was shown to perform competitively with OSI and NSPLIT in terms of mean squared error and classification error while carrying the corresponding benefits of enabling distributed implementation and supporting deep network learning.

For future work we will explore methods for inter-swarm competition to improve our algorithm's performance, especially on deep networks. For example we will explore the case where a swarm associated with a neuron will search for both the input and output weights of that neuron. Swarms that learn the same weights would then compete with one another during inter-swarm communication. This approach will increase the amount of sharing between neighboring swarms without introducing a combinatorial increase in computational burden. We will test the competitive sharing approach to NSPLIT and LSPLIT, as well as other learning methods (e.g., deep belief networks [12] and convolutional networks [13]), specifically on deep structure learning problems.

REFERENCES

[1] J.-R. Zhang, J. Zhang, T.-M. Lok, and M. R. Lyu, "A hybrid particle swarm optimization-backpropagation algorithm for feedforward neural network training," *Applied Mathematics and Computation*, vol. 185, pp. 1026–1037, 2007.

[2] M. Gori and A.Tesi, "On the problem of local minima in backpropagation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 1, pp. 76–86, 1992.

[3] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*, vol. IV, 1995, pp. 1942–1948.

[4] C. Zhang, H. Shao, and Y. Li, "Particle swarm optimization for evolving artificial neural networks," in *Proceedings of the IEEE Conference on System, Man, and Cybernetics*, vol. 4, 2000, pp. 2487–2490.

[5] X. Cai, N. Zhang, G. K. Venayagamoorthy, and D. C. Wunsch, "Time series prediction with recurrent neural networks trained by a hybrid PSOEA algorithm," *Neurocomputing*, vol. 70, pp. 2342–2354, 2007.

[6] F. van den Bergh and A. Engelbrecht, "Cooperative learning in neural networks using particle swarm optimizers," *South African Computer Journal*, vol. 26, pp. 94–90, 2000.

[7] B. K. Haberman and J. W. Sheppard, "Overlapping particle swarms for energy-efficient routing in sensor networks," *Wireless Networks*, vol. 18, no. 4, pp. 351–363, 2012.

[8] K. G. Pillai and J. W. Sheppard, "Overlapping swarm intelligence for training artificial neural networks," in *Proceedings of the IEEE Swarm Intelligence Symposium*, April 2011, pp. 1–8.

[9] R. C. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the IEEE Congress on Evolutionary Computing*, vol. 1, 2000, pp. 84–88.

[10] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[11] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *Journal of Machine Learning Research*, vol. 10, pp. 1–40, 2009.

[12] G. E. Hinton and Y.-W. Teh, "A fast learning algorithm for deep belief networks," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.