Overlapping Swarm Intelligence for Training Artificial Neural Networks

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

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

***Abstract*—A novel overlapping swarm intelligence algorithm is introduced to train the weights of an artificial neural network. Training a neural network is a difficult task that requires an effective search methodology to compute the weights along the edges of a network. The backpropagation algorithm, a gradient based method, is frequently used to train multilayer feed-forward networks. Gradient based methods might not always lead to a globally optimal solution of the network. On the other hand, training algorithms based on evolutionary computation have been used to train multilayer feed-forward networks in an attempt to overcome the limitations of gradient based algorithms with mixed results. This paper introduces an overlapping swarm intelligence technique to train multilayer feedforward networks. The results show that OSI method performs either on par with or better than the other methods tested.**

***Index Terms*—Backpropagation, machine learning, neural networks, particle swarm optimization, swarm intelligence**

I. Introduction

The backpropagation algorithm has been demonstrated to be an effective strategy for training feedforward neural networks. However as a gradient based method, it is known to suffer from local minima [1]. Additionally, the convergence rate of backpropagation is typically slow even if the learning goal can be achieved [2]. In recent years artificial neural networks have been trained using evolutionary computation methods. Evolutionary approaches such as genetic algorithms, particle swarm optimization, and hybrid approaches, which use both evolutionary and gradient based methods, have become popular to overcome the limitations of backpropagation algorithm [2], [3].

This paper introduces a novel overlapping swarm intelligence algorithm to train multilayer feedforward networks. In our approach, the structure of a neural network is decomposed into individual paths, and we exploit situations where the paths overlap. We seek to train this decomposed network focusing on learning subproblems using individual swarms for each path with inter-swarm communication.

Jing-Ru et al. [2] combined particle swarm optimization (PSO) and the backpropagation algorithm to train feedforward neural networks. In their work, the combined PSO and backpropagation methods result in better performance when compared with the Adaptive Particle Swarm Optimization algorithm (APSOA) and the backpropagation algorithm in both convergence speed and generalization performance [2]. The APSO algorithm differs from traditional PSO in that the inertial weight ω is reduced based upon the amount of time the algorithm has been searching. A feedforward neural network is trained with PSO until either the maximum number of generations is reached or the global best has not changed for ten generations. At that point, the neural network is trained with backpropagation. If backpropagation does not find a better solution than the global best from PSO then it is deemed to be an optimal solution.

Xindi et al. [3] proposed a hybrid PSO-EA algorithm to train recurrent neural networks to predict time series data. In their work, an initial population is created and evaluated. Winners are selected based on the fitness value and these winners are enhanced by PSO. Using these enhanced winners, offspring are generated using evolutionary operators, and these offspring replace less fit members of the population.

Potter [4] introduced cooperative coevolution learning in genetic algorithms in which several populations of function approximators such as neural networks are considered simultaneously. Initially, a separate population of individuals is generated randomly, and the initial fitness of each population member is computed by combining it with a random individual from each of the other populations. After initialization, each individual population is coevolved in a round-robin fashion using a traditional genetic algorithm. The fitness of a population member is obtained by combining it with the current best subcomponent of the remaining populations which are temporarily frozen.

Van den Bergh and Engelbrecht [5], [6] also introduced cooperative learning in PSO to train feedforward neural networks, but their results indicate that performance is sensitive to the degree of interdependence between the variables. In their work [6], they explain that certain deceptive functions could stagnate the evolution of particles when cooperative learning is used in PSO. The authors show that an algorithm that interleaves regular PSO and cooperative PSO performs well on rotated multimodal problems that have a high degree of interdependence [6] .

Haberman and Sheppard [7] proposed overlapping particle swarms for energy-efficient routing in sensor networks. The focus of this work is to find optimal routing strategies to maximize the availability of the overal network. In their work, each particle is both the centroid of swarm and a member of each of its neighboring swarms, and local best is defined as the best state the particle has seen in all of its swarms (i.e., the best of the best). Their approach increases the lifetime of the sensor network by almost a factor of two. Furthermore, their approach was demonstrated to perform significantly better than state-of-the-art energy-aware routing methods. Their work forms the motivation for our work reported here.

II. BACKGROUND

A. Artificial Neural Networks:

Artificial neural networks are connectionist models that attempt to solve computational tasks based on a network of simple computational units (i.e., neurons). Neurons are basic computational units that when combined with appropriate weights between neurons have been shown to solve a variety of problems ranging from classification and pattern recognition to function approximation. Neural network architectures can be feedforward or recurrent.

Single layer feedforward neural networks have an input layer that is connected to an output layer. Only the output layer has computational units (neurons). Multilayer feedforward networks have one or more layers of hidden units between the input and output layer. The hidden units enable the network to extract higher-order properties from the input. A neural network is said to be fully connected if every node is connected to all the nodes in the adjacent forward layer, and if some connections between some neurons are missing it is called a partially connected neural network. Each neuron of the network includes a non-linear activation function that is continuously differentiable. One of the most commonly used activation functions is the logistic function which is given as

$$y = 1/(1 + \exp(-v)), \tag{1}$$

where

$$v = \sum_{i=1}^{m} w_i x_i + b, \tag{2}$$

m is the number of inputs, w_i is a weight, and b is a bias

A recurrent network has at least one feedback loop in the network. One common architecture is the simple recurrent network (Elman Network), which has a hidden layer whose output feeds back to the input of the hidden layer with unit time delays [8].

B. Particle Swarm Optimization:

Particle Swarm Optimization (PSO) proposed by Eberhart and Kennedy [9], is a technique inspired by the social behavior of flocking birds. It is a population based approach where the system is initialized with random solutions (called particles), and search applies an update process where the velocity vectors applied to the particles are determined based on the fitness of states visited by the particles. Eventually, all the particles in a swarm will move closer to an optimum of the fitness function.

Algorithm 1 Particle Swarm Optimization

```
Create and initialize particles
repeat
  for all $x_i \epsilon P$ do
    Calculate fitness of particle $f(x_i)$
    if $f(x_i) < f(p_i)$ then
      $p_i = x_i$
    end if
    if $f(x_i) < f(p_g)$ then
      $p_g = x_i$
    end if
    $v_i = \omega v_i + \mathbf{U}(0, \phi_1) \otimes (p_i - x_i) + \mathbf{U}(0, \phi_2) \otimes (p_g - x_i)$
    $x_i = x_i + v_i$
  end for
until termination criterion is met
```

The PSO algorithm first initializes a swarm of particles randomly over a search space. These particles "fly" with a certain velocity and find a position in the search space after each iteration. On every iteration of the algorithm, the current position of a particle is evaluated against the fitness function. The best position is stored in a vector called p_i (personal best). Also, the position of the particle with the best global fitness is stored in a vector called p_g (global best). At each iteration, the particle's velocity is updated based on the influence of the local best position (p_i) and on the influence of the global best particle (p_g). Then each particle's position is updated by this newly calculated velocity.

The PSO update procedure is described in Algorithm 1. Here P is a swarm of particles, $\mathbf{U}(0, \phi_i)$ is a vector of random numbers uniformly distributed in $[0, \phi_i]$ which is generated for each iteration and for each particle, $\otimes$ is component-wise multiplication, v_i is the velocity of a particle, x_i is the current position of a particle in a search space, and p_i, p_g are the particle's personal best position and the global best neighbor of a particle respectively. The termination criterion in Algorithm 1 is problem dependent. For our problem, it will be based on convergence in mean squared error over a separate validation data set.

There are a few parameters that need to be chosen for Algorithm 1. The population size is chosen depending on the problem. The parameters ϕ_1 and ϕ_2 determine the force of the direction in which the particle is pulled between personal best and global best of the particles. These parameters need to be tuned properly for the PSO to converge. Also, the velocity of the particle is set to a minimum and maximum limit for the particle to control stability. To control the scope of the search, and also to control and perhaps eliminate the limit on velocity, an inertia weight ω is used.

III. OVERLAPPING SWARM INTELLIGENCE

We propose an approach to training feedforward neural networks inpired by the PSO approach but focusing on learning

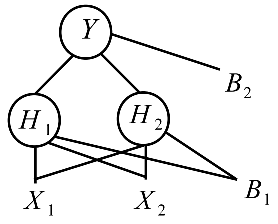


Fig. 1. XOR Neural Network

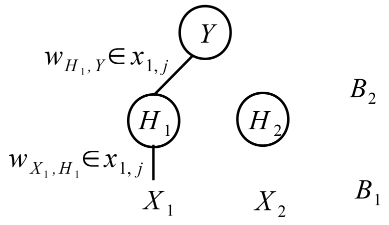


Fig. 2. Swarm S_1

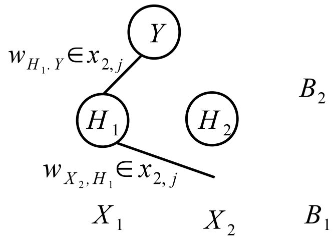
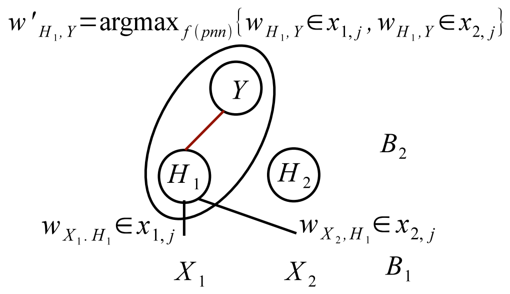


Fig. 3. Swarm S_2



$w'_{H_1,Y} = \mathrm{argmax}_{f(pnn)}\{w_{H_1,Y} \in x_{1,j}, w_{H_1,Y} \in x_{2,j}\}$
Y
B_2
H_1
H_2
$w_{X_1,H_1} \in x_{1,j}$
$w_{X_2,H_1} \in x_{2,j}$
X_1
X_2
B_1

Fig. 4. Overlap of swarm S_1 and S_2

subproblems as in [7]. With our approach, the structure of a neural network is decomposed into paths, where each path originates at an input node and terminates at an output node. Each path corresponds to the set of weights between the nodes that are connected in this path. A segment of weights in a path (i.e., connection between some nodes in the path) overlap with other paths that pass through the same segment. It is this overlap that is exploited in our approach. Each path in the network has a swarm, and inter-swarm communication occurs via the shared segments in the network. Specifically, a common vector of weights gvn (this plays a role similar to swarm global best), is maintained across all swarms that represents a global view of the whole neural network. It is constructed in a given generation by using the best particles from each of the swarms. The resulting inter-swarm communication then facilitates global learning of the weights in the neural network.

The procedure for Overlapping Swarm Intelligence (OSI) is described in Algorithm 2. As mentioned above, each path in the neural network includes a swarm that will have many particles. Each particle in a swarm is a vector of weights between the nodes in that path. Let S be a swarm of swarms, s_i be the ith swarm, $x_{i,j}$ be the current particle state for particle j in swarm i, and $p_{i,j}$ be the personal best state for particle j in swarm i.

To enable the inter-swarm communication, we define gvn be the complete vector of weights that represents the whole neural network constructed from the best particles in each of the swarms. More formally, given a neural network represented as graph $G = (V, E)$ where a directed edge is represented as a pair of vertices (V_m, V_n), then $gvn = E'$ where each $(V'_m, V'_n) \in E' = \arg\max\{p_{i,j}\}$. In other words, the global network gvn is constructed from the edges corresponding to the personal best particles in each of the swarms. Next, each particle must be evaluated within the context of some neural network. To do this, we define $pnn_{i,j}$ to be the "personal neural network" constructed using the weights of particle $x_{i,j}$ in addition to the remaining weights from gvn. More formally,

$$pnn_{i,j} = x_{i,j} \cup \{gvn \setminus sharedgvn_{i,j}\}, \tag{3}$$

where $sharedgvn_{i,j}$ consists of those edges in gvn corresponding to the edges in the particle $x_{i,j}$.

As an example, Figure 1 shows a neural network architecture that can be used to solve the exclusive-OR (XOR) problem. For this network, the path for swarm S_1 is shown in Figure 2 and the path for swarm S_2 is shown in Figure 3. Figure 4, shows the overlap of paths between swarm S_1 and S_2. For this network, seven paths can be generated, and swarms would be created for each path. Specifically, we would have the following:

- S_1: $X_1 - H_1 - Y$
- S_2: $X_1 - H_2 - Y$
- S_3: $X_2 - H_1 - Y$
- S_4: $X_2 - H_2 - Y$
- S_5: $B_1 - H_1 - Y$

Algorithm 2 Overlapping Swarm Intelligence

```
Create and initialize $gvn$
Create and initialize particles in each swarms
repeat
  for all $s_i \in S$ do
    for all $x_{i,j} \in s_i$ do
      Construct $pnn_{i,j}$
      Evaluate particle fitness $f(pnn_{i,j})$
      Assign particle $x_{i,j}$ fitness, $f(x_{i,j}) = f(pnn_{i,j})$
      if $f(x_{i,j}) < f(p_{i,j})$ then
        $p_{i,j} = x_{i,j}$
      end if
      Evaluate global fitness$f(gvn)$
      if $f(x_{i,j}) < f(gvn)$ then
        Update $sharedgvn_{i,j}$
      end if
      Update velocity using equation (6)
      $x_{i,j} = x_{i,j} + v_{i,j}$
    end for
  end for
until termination criterion is met
```

- S_6: $B_1 - H_2 - Y$
- S_7: $B_2 - Y$

Thus the path $X_1 - H_1 - Y$ represents the connection between input X_1, hidden node H_1, and output node Y.

If we were to apply Algorithm 2 to this network, then gvn would be

$$gvn = \{(X_1H_1), (H_1Y), (X_1H_2), (H_2Y), (X_2H_1), (X_2H_2), (B_1H_1), (B_1H_2), (B_2Y)\}, \quad (4)$$

$pnn_{1,2}$ for the second particle in swarm S_1 would be

$$pnn_{1,2} = \{(X_1H_1)_{1,2}, (H_1Y)_{1,2}, (X_1H_2), (H_2Y), (X_2H_1), (X_2H_2), (B_1H_1), (B_1H_2), (B_2Y)\}, \quad (5)$$

and $sharedgvn_{1,2} = (X_1H_1), (H_1Y)$. Finally, $v_{i,j}$, the velocity of particle j in swarm i, would be updated as follows:

$$v_{i,j} = \omega v_{i,j} + \mathbf{U}(0, \phi_1) \otimes (p_{i,j} - x_{i,j}) + \mathbf{U}(0, \phi_2) \otimes (sharedgvn_{i,j} - x_{i,j}). \quad (6)$$

The actual inter-swarm communication process occurs via the construction and use of gvn as described in the next section.

IV. Credit Assignment

For our overlapping swarm-based method of training neural networks to be effective, we need an approach for evaluating the fitness of the individual particles in each swarm. Furthermore, since we are claiming that inter-swarm communication will help in training the neural networks, our fitness evaluation method must also support sharing fitness information between swarms. The approach we have taken is inspired by the work in [4] and [5] where a global network is used as the basis for credit assignment. We implemented this approach through the gvn mechanism described above.

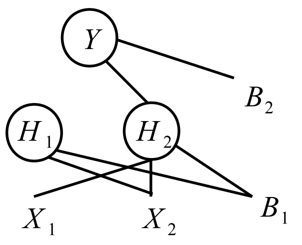


Fig. 5. Part of gvn used for evaluation of swarm S_1

Recall that $x_{i,j}$ denotes the jth particle (consisting of a vector of weights) in the ith swarm. When a particle is evaluated, it is done so as a part of a whole neural net. To instantiate the network to be evaluated, the topology is pre-specified, and the weights are taken as the union of the weights in $x_{i,j}$ and the weights from gvn on edges other than those in $x_{i,j}$. For example, when a particle in swarm S_1 is evaluated, the weights from the particle as in Figure 2 are combined with the weights from the global network as in Figure 5. The resulting network is then used for the fitness calculation. Mathematically, this evaluation network corresponds to $pnn_{i,j}$ as defined in (3) with the specific example being given in (5).

The fitness of the network constructed for each particle is compared to the fitness of the global network that uses all of the weights in gvn. Each particle in a swarm keeps track of the best set of weights found so far with respect to that part of gvn that does not include the evaluated particle's weights. Hence, when a particle is evaluated, the part of gvn that does not include the evaluated particle's weights is treated as constant. Credit is assigned to the individual particle in each swarm based on its performance using the rest of gvn. This greatly simplifies the credit assignment problem.

More specifically, at each generation of the OSI algorithm, the global neural network, gvn, is extracted from the individual particles where each edge weight in the network corresponds to the edge weight in the best particle of each swarm. When an edge is shared between swarms, a competition is held between the edges from the respective personal bests in the global network. That edge yielding better performance on the training data is the one selected for inclusion in gvn.

Each particle is then evaluated on the training set when inserted into the local version of the neural network, $pnn_{i,j}$. Thus the individual particles are evaluated based on their own representation of the current neural network, but the results of their localized search are communicated to overlapping swarms via the edge competition used to define gvn.

V. Experimental Setup

To test the effectiveness of our OSI algorithm, several experiments were performed on 2-layered and 4-layered feed-forward neural networks. A 2-layered network architecture was chosen based on the experimental setup from [5], and a 4-layered network architecture was chosen to run experiments

		Window Size							
Method	Swarm Size	3	5	7	10	12	15	17	20
	2	0.1858	0.1714	0.1714	0.1590	0.1587	0.1583	0.1583	0.1583
	3	0.1272	0.1201	0.1201	0.1134	0.1130	0.1128	0.1128	0.1128
	5	0.1880	0.1342	0.1342	0.1323	0.1313	0.1314	0.1314	0.1314
	7	0.1296	0.1149	0.1149	0.1143	0.1143	0.1143	0.1143	0.1143
OSI	10	0.1246	0.1182	0.1182	0.1182	0.1182	0.1182	0.1182	0.1182
	13	0.1170	0.1128	0.1128	0.1128	0.1128	0.1128	0.1128	0.1128
	15	0.1181	0.1169	0.1169	0.1169	0.1169	0.1169	0.1169	0.1169
	17	0.1103	**0.1083**	0.1083	0.1083	0.1083	0.1083	0.1083	0.1083
	20	0.1110	0.1110	0.1110	0.1110	0.1110	0.1110	0.1110	0.1110
	2	0.8159	0.8159	0.8159	0.7608	0.7608	0.7608	0.7608	0.7608
	3	0.6737	0.6737	0.6737	0.6737	0.6664	0.6664	0.6664	0.6664
	5	0.7350	0.7350	0.7350	0.4618	0.4348	0.3935	0.3906	0.1314
	7	0.6757	0.6757	0.6757	0.4314	0.4242	0.2916	0.2854	0.2854
PSO	10	0.7088	0.4185	0.4185	0.3710	0.3661	0.3637	0.3637	0.3637
	13	0.4105	0.3419	0.3419	0.3348	0.3345	0.3363	0.3332	0.3332
	15	0.3928	0.3731	0.3731	0.3381	0.3381	0.3079	0.3079	0.3079
	17	0.3923	0.3717	0.3717	0.2707	0.2549	0.1913	0.1872	0.1872
	20	0.3269	0.2730	0.2730	0.1846	0.1846	0.1694	**0.1641**	0.1641
BP		0.0918	0.0920	0.0921	0.0923	0.0924	0.0925	**0.0645**	0.0646

TABLE I
IRIS MEAN SQUARED ERROR FOR DIFFERENT SWARM AND WINDOW SIZES

		Window Size							
Method	Swarm Size	3	5	7	10	12	15	17	20
	2	0.2507	0.2464	0.2464	0.2445	0.2392	0.2447	0.2458	0.2458
	3	0.2415	0.2449	0.2449	0.2519	0.2552	0.2510	0.2492	0.2492
	5	0.2700	0.2783	0.2783	0.2813	0.2834	0.2876	0.2782	0.2782
	7	0.2270	**0.2198**	0.2198	0.2240	0.2298	0.2273	0.2318	0.2318
OSI	10	0.2403	0.2423	0.2423	0.2424	0.2407	0.2503	0.2580	0.2580
	13	0.2381	0.2483	0.2483	0.2448	0.2422	0.2390	0.2386	0.2386
	15	0.2390	0.2432	0.2432	0.2534	0.2575	0.2531	0.2648	0.2648
	17	0.2399	0.2382	0.2382	0.2325	0.2364	0.2356	0.2342	0.2342
	20	0.2339	0.2462	0.2462	0.2535	0.2627	0.2649	0.2713	0.2713
	2	0.5241	0.5103	0.5103	0.4838	0.4928	0.4805	0.4848	0.4848
	3	0.5241	0.4256	0.4256	0.4041	0.3951	0.3844	0.3836	0.3836
	5	0.5241	0.5118	0.5118	0.4671	0.4655	0.4558	0.4205	0.4205
	7	0.4161	0.4020	0.4020	0.3664	0.3540	0.3519	0.3382	0.3382
PSO	10	0.4596	0.3871	0.3871	0.3138	0.3105	0.2968	0.2932	0.2932
	13	0.4638	0.4059	0.4059	0.3180	0.3239	0.3110	0.3081	0.3081
	15	0.4276	0.3202	0.3202	0.2589	0.2549	0.2327	**0.2310**	0.2310
	17	0.4568	0.4276	0.4276	0.3520	0.3585	0.2863	0.2592	0.2592
	20	0.4297	0.3745	0.3745	0.3166	0.2554	0.3084	0.3090	0.3090
BP		0.2143	0.2144	0.2145	0.2144	0.2143	0.2153	**0.2068**	0.2075

TABLE II
IONOSPHERE MEAN SQUARED ERROR FOR DIFFERENT SWARM AND WINDOW SIZES

on a "deep" network. Neural networks that have greater than two layers are called deep neural networks. These experiments focused on comparing the OSI methodology with a full PSO training approach, and the standard backpropagation (BP) algorithm using different data sets. For the full PSO approach the weights from different layers were taken and serialized into a single weight vector and optimized using a single swarm [5].

In all the experiments, the initial weights were set randomly in 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 acceleration coefficients ϕ_1 and ϕ_2 were set to 1.49445. These values were chosen based on the results of [10]. To evaluate the performance of algorithms, 33.3% of the data was set aside to test for overfitting. The remaining data set was then divided into training and testing data sets using a 5×2 cross validation procedure.

All the experiments were on classification problems, and four different data sets were used. For the first three, the IRIS, IONOSPHERE, and GLASS data sets were obtained from the UCI machine learning repository [11], and each of these classification problems were solved using 2-layer networks. The IRIS data set contains three classes with a total of 150 instances. One class is linearly separable from the other two; the latter are not linearly separable from each other [11]. For this data set, a 4-input, 3-hidden, and 3-output network architecture was used in the experiments. The IONOSPHERE data set has two classes but has higher input dimension. In total it has 351 instances with 34 inputs [11]. In this case, a 34-input, 5-hidden, and 2-output network architecture was used. The GLASS data set has six classes with a total of 214

		Window Size							
Method	Swarm Size	3	5	7	10	12	15	17	20
	2	0.6054	0.6060	0.6060	0.6060	0.6060	0.6060	0.6060	0.6060
	3	0.6181	0.6181	0.6181	0.6182	0.6182	0.6182	0.6182	0.6182
	5	0.6032	0.6024	0.6024	0.6026	0.6026	0.6026	0.6026	0.6026
	7	0.5977	0.5987	0.5987	0.5988	0.5988	0.6005	0.6005	0.6005
OSI	10	0.5960	0.5956	0.5956	0.5970	0.5970	0.5970	0.5970	0.5970
	13	0.5973	0.5960	0.5960	0.5973	0.5980	0.5978	0.5978	0.5978
	15	0.5901	0.5904	0.5904	0.5885	0.5885	0.5885	0.5885	0.5885
	17	0.5780	**0.5774**	0.5774	0.5784	0.5784	0.5803	0.5803	0.5803
	20	0.5938	0.5952	0.5952	0.5959	0.5959	0.5959	0.5951	0.5951
	2	1.9497	1.1257	1.1257	1.0575	1.0570	1.0568	1.0563	1.0563
	3	1.9497	0.8122	0.8122	0.7805	0.7619	0.7513	0.7488	0.7488
	5	0.8526	0.8187	0.8187	0.7296	0.7273	0.7064	0.7076	0.7076
	7	0.8024	0.7214	0.7214	0.6849	0.6657	0.6471	**0.6438**	0.6438
PSO	10	0.8321	0.7718	0.7718	0.6926	0.6560	0.6562	0.6476	0.6476
	13	0.8174	0.7686	0.7686	0.7006	0.6977	0.6776	0.6768	0.6768
	15	0.7926	0.7421	0.7421	0.7134	0.7126	0.7123	0.7037	0.7037
	17	0.7893	0.7292	0.7292	0.6793	0.6631	0.6498	0.6457	0.6457
	20	0.7951	0.7640	0.7640	0.7033	0.7003	0.6812	0.6834	0.6834
BP		0.5826	0.5826	0.5826	0.5826	0.5826	**0.5825**	0.5825	0.5825

TABLE III
GLASS MEAN SQUARED ERROR FOR DIFFERENT SWARM AND WINDOW SIZES

		Window Size							
Method	Swarm Size	3	5	7	10	12	15	17	20
	2	0.2327	0.2162	0.2162	0.2090	0.2014	0.2013	0.2015	0.2015
	3	0.2133	0.2068	0.2068	0.1916	0.1915	0.1917	0.1917	0.1917
	5	0.2316	0.2084	0.2084	0.2043	0.2038	0.2039	0.2039	0.2039
	7	0.2081	0.2076	0.2076	0.1991	0.1991	0.1990	0.1979	0.1979
OSI	10	0.2187	0.1944	0.1944	0.1916	0.1913	0.1910	0.1911	0.1911
	13	0.1866	0.1846	0.1846	**0.1791**	0.1791	0.1791	0.1791	0.1791
	15	0.2043	0.1870	0.1870	0.1862	0.1862	0.1864	0.1864	0.1864
	17	0.2100	0.1947	0.1947	0.1894	0.1894	0.1894	0.1894	0.1894
	20	0.1978	0.1913	0.1913	0.1887	0.1889	0.1867	0.1867	0.1867
	2	0.2492	0.2489	0.2489	0.2489	0.2489	0.2481	0.2481	0.2481
	3	0.2492	0.2489	0.2489	0.2468	0.2467	0.2467	0.2465	0.2465
	5	0.2499	0.2495	0.2495	0.2492	0.2490	0.2287	0.2458	0.2458
	7	0.2490	0.2490	0.2490	0.2489	0.2486	0.2467	0.2467	0.2467
PSO	10	0.2495	0.2491	0.2491	0.2489	0.2488	0.2487	0.2478	0.2478
	13	0.2495	0.2498	0.2498	0.2495	0.2497	0.2497	0.2486	0.2486
	15	0.2490	0.2490	0.2490	0.2448	0.2393	**0.2385**	0.2387	0.2387
	17	0.2490	0.2490	0.2490	0.2485	0.2483	0.2475	0.2477	0.2477
	18	0.2495	0.2498	0.2498	0.2496	0.2475	0.2466	0.2466	0.2466
BP		**0.3134**	0.3134	0.3134	0.3134	0.3134	0.3134	0.3134	0.3135

TABLE IV
4BIT MEAN SQUARED ERROR FOR DIFFERENT SWARM AND WINDOW SIZES

instances with 9 inputs. Also, it has a highly skewed class distribution [5], that makes it difficult to learn. Here a 9-input, 6-hidden and 6-output network architecture was used. For each of these three data sets, the network architectures were chosen based on the experiments discussed in [5]. For GLASS we note that [5] used an 8-input, 6-hidden and 6- output network architecture, but the actual data set has 9-inputs. Hence it was decided to use 9 inputs for this data set.

The fourth experiment used data generated for a 4-bit parity problem (which we label 4BIT). For this experiment, the data set was produced by randomly generating 1000 data points. Each data point had four inputs (corresponding to the four bits), each of which were generated randomly over the interval [0,1]. From these data points an input was interpreted as if had value “1” if it’s generated value was greater than “0.5” and zero otherwise. For this problem when the number of “1”s in all the inputs was even, then the output value was set to “1”, and when the number of “1”s in all the inputs was odd, then the the output value was set to “0”.

For this experiment, a 4-layered “deep” network with 4-inputs, 4 hidden units in layer 1, 3 hidden units in layer 2, 2 hidden units in layer 3, and 1-output was used. Deep networks trained using gradient-based methods have generally been found to be significantly more difficult to train than neural networks with one or two hidden layers [12]. Gradient descent methods that are frequently used to train neural networks can easily get caught in local minima or on plateaus of the non-convex training criterion. Furthermore, error propagation methods are found to have the significance of the error correction diffused when propagating through several layers.

	OSI	PSO	BP
IRIS	0.1083 ± 0.01	0.1641 ± 0.02	$\mathbf{0.0645 \pm 0.02}$
IONOSPHERE	$\mathbf{0.2198 \pm 0.03}$	$\mathbf{0.2310 \pm 0.04}$	$\mathbf{0.2068 \pm 0.02}$
GLASS	$\mathbf{0.5774 \pm 0.01}$	0.6438 ± 0.01	$\mathbf{0.5825 \pm 0.04}$
4BIT	$\mathbf{0.1791 \pm 0.01}$	0.2385 ± 0.01	0.3134 ± 0.001

TABLE V
MEAN SQUARED ERROR WITH 95% CONFIDENCE INTERVAL

	OSI	PSO	BP
IRIS	$\mathbf{0.0235 \pm 0.01}$	0.0549 ± 0.01	$\mathbf{0.0333 \pm 0.0227}$
IONOSPHERE	$\mathbf{0.1364 \pm 0.02}$	$\mathbf{0.1491 \pm 0.03}$	$\mathbf{0.1255 \pm 0.02}$
GLASS	$\mathbf{0.4557 \pm 0.02}$	0.5337 ± 0.03	$\mathbf{0.4048 \pm 0.04}$
4BIT	$\mathbf{0.2376 \pm 0.02}$	0.4262 ± 0.05	0.5335 ± 0.001

TABLE VI
CLASSIFICATION ERROR WITH 95% CONFIDENCE INTERVAL

Because our method does not propagate error corrections, we expect our OSI approach will be more effective when training deep neural networks.

Experiments were conducted on all data sets with different swarm sizes. The number of particles used in the experiments are $\{2, 3, 5, 7, 10, 13, 15, 17, 20\}$. In our experiments, we attempted to prevent over-fitting using the following approach. Error on the test data was calculated for each swarm size using linear regression with different window sizes (sliding on the number of epochs) and mean squared error on the validation data at that epoch. For each swarm, the following window sizes were used: $\{3, 5, 7, 10, 12, 15, 17, 20\}$. For each window size, if linear regression on the validation data resulted in a non-negative slope then training was stopped, and the mean squared error on the test data at the starting point of window (epoch number) was calculated. Tables I, II, III, and IV show mean squared error on the test data set at the starting point of each window for different swarm sizes. In these tables, column "Window Size" represents the starting point of the window where mean squared error on test data was calculated. "Swarm Size" represents the number of particles used in each swarm. The minimum mean squared error for each data set across all particle and window sizes was calculated for each of the various configurations.

VI. EXPERIMENTAL RESULTS

The results of our experiments are summarized in Table V and in Table VI. In Table V, the columns correspond to the methods tested, and the rows correspond to the minumum mean squared error on the test data. Ninety five percent confidence intervals are also shown. This mean squared error is derived from Tables I, II, III, and IV for each data set. A paired t-test is performed on all methods and data sets using the mean squared error from Table V. Also in Table V, values that in bold indicate the algorithm that performs best on the corresponding data set. If algorithms tie statistically for the best, then each of their values are bolded.

In Table VI, the columns represent methods and rows represent classification error on the test data with 95% confidence intervals. The classification error for each data set is calculated from the particle and window sizes, that were selected for Table V. The bolded values in Table VI also indicate the best performers on each of the data sets.

With the IRIS data set, based on the paired t-test performed on mean squared error, we observed that the OSI method performed better than PSO. Also BP performs better than OSI on this data set. Table I shows mean squared error for each particle size and window size for OSI, PSO, and for each window size for BP algorithm, for the IRIS data set. Based on the paired t-test performed on classification error, we observed that the OSI method performed better than PSO, and BP did not perform significantly different from OSI as seen in Table VI.

On IONOSPHERE, based on the paired t-test performed on mean squared error, we observed that the OSI method performed statistically equivalent to PSO and BP. Table II shows mean squared error for each particle size and window size for OSI, PSO, and for each window size for BP algorithm, for the IONOSPHERE dataset. Based on paired t- test performed on mean squared error, we observed that the OSI method performed statistically equivalent to PSO and BP as seen in Table VI.

Examining a paired t-test performed on mean squared error for GLASS, we found that OSI method performs significantly better than PSO but it is not significantly different from the BP algorithm. Table III shows mean squared error for each particle size and window size for OSI, PSO, and for each window size for BP algorithm, for the GLASS dataset. Based on the paired t-test performed on classification error, we observed that the OSI method performed better than PSO and BP does not perform significantly different from OSI as seen in Table VI.

The 4BIT experiment used a four-layered feedforward neural network with the 4-bit even parity problem. Based on the paired t-test applied to mean squared error, we observed that the OSI method performed better than BP and PSO. Table IV shows the mean squared error for each particle size and window size for OSI, PSO, and for each window size for BP algorithm, when run on the 4BIT dataset. From Tables IV, V, and VI, we see that our technique performed significantly better than the BP algorithm, and we can confirm our hypothesis that the OSI technique provides good generalization on this deep network.

VII. Discussion

The paired t-test on mean squared error shows that the OSI method performed either better than or equal to the other methods on all data sets studied, except BP beats OSI on IRIS. In addition, the paired t-test on classification error shows that the OSI method performed either better or equal to the other methods on all data sets studied including BP on IRIS. We speculate that the OSI method performs as well as it does because of the inter-swarm communication on the overlapping paths in the decomposed network and splitting the neural network into multiple swarms. Inter-swarm communication helps to share information between swarms while each swarm works to optimize an individual path in the network that has connections that overlap with other swarms. Thus, this combination helps the global emergent behavior among all the swarms to yield the strong performance. We note that it was also shown in [5] that various split approaches using the same strategy for credit assignment performed better than regular PSO.

One important property that we noticed from the experiments for the OSI method is that regardless of swarm sizes, the credit assignment strategy using the global view of the neural network across all swarms dominates the results. Because of global credit assignment, there is no significant difference due to swarm size. This can be observed from the Tables I, II, III, and IV that when the particle size increases, the positive change in the mean squared error on the test data was not significant. We believe that even better performance can be achieved by applying a local credit assignment strategy in combination with the inter-swarm communication procedure. This will be tested in future experiments using several local credit assignment strategies combined with a variety of communication mechanisms between the swarms. Specifically, the global network will be eliminated in favor of communicating local views of performance. This would also have the advantage of supporting neural net training in a highly distributed but low communication overhead environment.

VIII. Conclusions and Future Work

Splitting the neural network into multiple paths and training the network with localized swarms has been demonstrated to improve accuracy on certain data sets and on certain network architectures. Indeed, in none of our experiments did this approach degrade classification accuracy. Moreover the OSI methodology performs better than other methods on the tested deep network in terms of accuracy and generalization. For future work, we plan to compare the OSI methodology against the LSPLIT and NSPLIT methods described in [5]. Also, as described above, several alternative local credit assignment strategies will be explored. For example, one such strategy would associate an individual evaluation network with each swarm. In this case, each swarm will continue to optimize only its path weight vectors, and its performance on the local evaluation network will then be shared through inter-swarm communication via the overlapping segments. In addition, different overlapping approaches among swarms will be studied using the OSI method such as node overlap. We also note the results from [5] showing that functions with a high degree of interdependency among its variables will tend to degrade performance with localized methods. We will evaluate the extent to which inter-swarm communication with the OSI method mitigates this problem. Next, generalized function approximation networks as well as recurrent networks will be studied using the OSI method. Finally, since our results are positive for the 4BIT even parity problem with 3-hidden layers, the OSI method will be studied as a strategy for training deep neural network architectures as well.

Acknowledgments

We would like to thank several members of the Numerical Intelligent Systems Laboratory at Montana State University for their comments on early versions of this paper. Specifically, we thank Scott Wahl, Shane Strasser, and Patrick Donnelly. We also thank Brian Haberman from the sister Numerical Intelligent Systems Laboratory at Johns Hopkins University for his comments.

References

[1] M. Gori and A.Tesi, “On the problem of local minima in backpropagation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 1, pp. 76–86, 1992.

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

[3] X. Cai, N. Zhang, G. K. Venayagamoorthy, and D. C. Wunsch, “Time series prediction with recurrent neural networks trained by a hybrid pso-ea algorithm,” *Neurocomputing*, vol. 70, pp. 2342–2353, 2007.

[4] M. A. Potter and K. A. D. Jong, “A cooperative coevolutionary approach to function optimization,” *Parallel Problem Solving From Nature*, vol. 866, pp. 249–257, 1994.

[5] F. van den Bergh and A. Engelbrecht, “Cooperative learning in neural networks using particle swarm optimizers,” *South African Computer Journal*, vol. 26, pp. 84–90, 2000.

[6] F. van den Bergh and A. P. Engelbrecht, “A cooperative approach to particle swarm optimization,” *IEEE Trans. on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[7] B. K. Haberman and J. W. Sheppard, “Overlapping particle swarms for energy-efficient routing in sensor networks,” submitted to Wireless Networking, Spring 2010.

[8] J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

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

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

[11] C. Blake and C.Merz, “UCI repository of machine learning databases,” 1998.

[12] 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.