Self-Organizing Swarm (SOSwarm): A Particle Swarm Algorithm for Unsupervised Learning

Michael O'Neill and Anthony Brabazon

***Abstract*— We present a novel self-organizing Particle Swarm algorithm, SOSwarm, that adopts unsupervised learning. Input vectors are projected onto a lower dimensional map space producing a visual representation of the input data in a manner similar to the Self-Organizing Map (SOM) artificial neural network. Particles in the map react to the input data by modifying their velocities using a standard Particle Swarm Optimization update function, and therefore organize themselves spatially within fixed neighborhoods in response to the input training vectors. SOSwarm is successfully applied to four benchmark classification problems from the UCI Machine Learning repository with the novel SOSwarm algorithm outperforming or equaling the best reported results on all four of the problems analyzed.**

I. Introduction

In this proof of concept study we introduce the novel Self-Organizing Swarm (SOSwarm) algorithm, which adopts unsupervised learning with a Particle Swarm Algorithm. In this paper we firstly introduce the Particle Swarm Optimization algorithm (Section II) upon which SOSwarm is based. We then introduce the fundamental concepts of the Self-Organizing Map (SOM), which bears similarities to SOSwarm in Section III. Following a detailed exposition of SOSwarm in Section IV we apply the algorithm to four benchmark problems from the UCI Machine Learning repository (Section V) before drawing conclusions and outlining some possible directions for future investigations with SOSwarm in Section VI.

II. Particle Swarm Optimization

The PSO algorithm was introduced by Kennedy and Eberhart [6] and is described in detail in [5]. In the context of PSO a swarm can be defined as '... a population of interacting elements that is able to optimize some global objective through collaborative search of a space.' [5] (p. xxvii). The nature of the interacting elements (particles) depends on the problem domain, in this study they represent the input parameter values of the problem domain. These particles move (fly) in an n-dimensional search space, in an attempt to uncover ever-better solutions to the problem of interest.

Each of the particles has two associated properties, a current position and a velocity. Each particle also has a memory of the best location in the search space that it has found so far (p_{best}), and knows the best location found to date by all the particles in the population (g_{best}) or in an alternative version of the algorithm, a local neighborhood around each particle (l_{best}). In the local version of the algorithm, each particle is considered to be linked to a subset of the population of particles, and this linkage structure is fixed at the beginning of the optimization process and remains unchanged during it (see Fig.1). Whether the local or global communication version is implemented, at each step of the algorithm, particles are displaced from their current position by applying a velocity (or gradient) vector to them.

The velocity size / direction is influenced by the velocity in the previous iteration of the algorithm (simulates 'momentum'), and the location of a particle relative to its p_{best} and g_{best} (or l_{best}). Therefore, at each step, the size and direction of each particle's move is a function of its own history (experience), and the social influence of its peer group.

A number of variants of the particle swarm algorithm (PSA) exist. The following paragraphs provide a description of a canonical continuous version of the algorithm.

i. Initialize each particle in the population by randomly selecting values for its location and velocity vectors.
ii. Calculate the fitness value of each particle. If the current fitness value for a particle is greater than the best fitness value found for the particle so far, then revise p_{best}.
iii. Determine the location of the particle with the highest fitness and revise g_{best} if necessary.
iv. For each particle, calculate its velocity according to equation 1.
v. Update the location of each particle according to equation 3.
vi. Repeat steps ii - v until stopping criteria are met.

The update algorithm for particle i's velocity vector v_i is:

$$v_i(t+1) = w * v_i(t) + (c_1 * R_1 * (p_{best} - x_i)) + (c_2 * R_2 * (g_{best} - x_i)) \quad (1)$$

where,

$$w = wmax - ((wmax - wmin)/itermax) * iter \quad (2)$$

In equation 1, p_{best} is the location of the best solution found to-date by particle i, g_{best} is the location of the global-best solution found by all particles to date, c_1 and c_2 are the weights associated with the p_{best} and the g_{best} terms in the velocity update equation, x_i is particle i's current location, and R_1 and R_2 are randomly drawn from U(0,1). The parameter w represents a momentum coefficient which

Michael O'Neill and Anthony Brabazon are with the Natural Computing Research & Applications Group, University College Dublin, Belfield, Dublin 4, Ireland (email: m.oneill@ucd.ie and anthony.brabazon@ucd.ie).

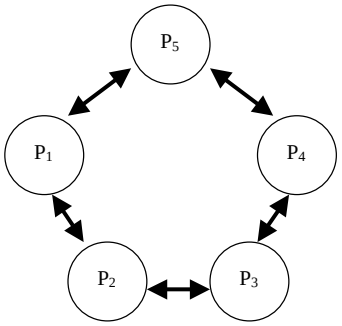

Fig. 1. Ring topology where l_{best} is defined using a 3 node neighborhood (itself and two other nodes)

is reduced according to equation 2 as the algorithm iterates. In equation 2, $itermax$ and $iter$ are the total number of iterations the algorithm will run for, and the current iteration value respectively, and $wmax$ and $wmin$ set the upper and lower boundaries on the value of the momentum coefficient. The velocity update on any dimension is constrained to a maximum value of $vmax$. Once the velocity update for particle i is determined, its position is updated (equation 3), and p_{best} is updated if necessary (equations 4 & 5).

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{3}$$

$$y_i(t+1) = y_i(t) \text{ if, } f(x_i(t)) \leq f(y_i(t)) \tag{4}$$

$$y_i(t+1) = x_i(t) \text{ if, } f(x_i(t)) > f(y_i(t)) \tag{5}$$

After the location of all particles have been updated, a check is made to determine whether g_{best} needs to be updated (equation 6).

$$\hat{y} \in (y_0, \ldots, y_n) | f(\hat{y}) = \max(f(y_0), \ldots, f(y_n)) \tag{6}$$

In each iteration of the algorithm, a particle is stochastically accelerated towards its previous best position and towards a global (or neighborhood) best position, thereby forcing particles to continually search in the most-promising regions found so far in the solution space. The weight coefficients c_1 and c_2 control the relative impact of the p_{best} and g_{best} locations on the velocity of a particle. Low values for c_1 and c_2 allow each particle to explore far away from already uncovered good points (there is less emphasis on past learning), high values of the parameters encourage more intensive search of regions close to these points. The random coefficients r_1 and r_2 ensure that the algorithm is stochastic. A practical effect of r_1 and r_2, is that neither the individual nor the social learning terms are always dominant.

The neighborhood structure plays a critical role in determining the nature of the communication between particles during the search process. If the neighborhood is set at 1 (each particle only communicates with itself), then each particle searches independently of all other particles. If the neighborhood$= N$ (the number of particles in the swarm), all particles can communicate with each other, and we have the g_{best} version of the PSO algorithm.

Particle Swarm algorithms have been successfully applied to a diverse range of problems including Financial Modeling [2], the automatic generation of programs ([10], [11]) using a grammatical representation borrowed from Grammatical Evolution [12], [13], [14], [15], [16], and the construction of Artificial Neural Networks [17].

III. Self-Organizing Map

Self-organizing maps (SOM) [7], [8], [9] are a form of artificial neural network (NN) which can cluster data using unsupervised learning. Unsupervised learning is used when the outputs (clusters) are not known a priori.

The SOM acts to project (compress) input data vectors onto a low-dimensional space, typically a two-dimensional grid structure, thereby producing a visual representation of the input data. The unsupervised learning process is based on measures of similarity amongst the input data vectors. During the training process, the network undergoes self-organization as like input data patterns are grouped or clustered together on the grid structure. SOMs have been utilized for a variety of clustering and classification problems including speech recognition and medical diagnosis [3]. The SOM bears similarities with the traditional statistical technique of Principal Component Analysis (PCA). However, unlike PCA the projection of the input data is not necessarily restricted to be linear.

The SOM consists of two layers, the input layer (a holding point for the input data), and the *mapping* layer (see Fig. 2). The input layer has as many nodes as there are input variables. The two layers are fully connected to each other and each of the nodes in the hidden layer has an associated weight vector, with one weight for each connection with the input layer.

The aim of the SOM is to group like input data-vectors together on the mapping layer, therefore the method is topology preserving as items which are close in the input space are also close in the mapping space. During training the data vectors are presented to the SOM through the input layer one at a time. The nodes in the mapping layer *compete* for the input data vector. The winner is the mapping node whose vector of incoming connection weights most closely resembles the components of the input data vector. The winner has the values of its weight vector adjusted to move them towards the values of the input data vector, and the mapping layer nodes in the neighborhood of the winning node also have their weight vectors altered to become more like the input data vector (a form of co-operation between the neighboring nodes). As more input data vectors are passed through the network, the weight vectors of the mapping layer nodes self-organize. By the end of the training process, different parts of the mapping layer respond strongly to specific regions of input space. Once training of the network is complete, the clusters obtained can be examined in order

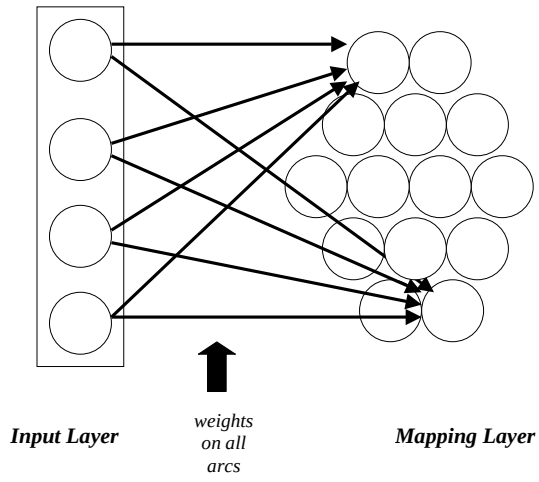

Fig. 2. A SOM with a 2-d mapping layer. On grounds of visual clarity, only the connections between the input layer and two of the mapping layer nodes are shown.

to gain better insight into the underlying dataset. Questions which can be addresses include: what input items have been grouped together, and what are the typical values for each input in a specific cluster?

There are several ways that a PSO-SOM hybrid could be constructed. In [17] and [18] a PSO algorithm was used to refine the weight vectors for a SOM obtained after an initial application of a standard SOM training methodology. In this approach each particle consisted of a complete set of weights for the SOM, and the object was to improve the initial clustering result by applying PSO to the population of weight vectors. The approach in our study differs fundamentally from the above and is outlined in the next section.

IV. Self-Organizing Swarm

The Self-Organizing Swarm (SOSwarm) bears some similarity to an SOM with the adoption of a visual (2D) mapping layer. However, the components of the mapping layer represent particles which move according to an adapted version of the Particle Swarm algorithm.

Instead of adjusting vector values in the map space with respect to the training input vectors alone, as is the case in SOM training, the particles (vectors) in the mapping layer adjust their location using a PSO update function. As such, a social form of learning is adopted that takes into consideration both personal and global or local neighborhood information to adjust a velocity vector associated with each particle. The velocity vector encodes a form of momentum into the search process, which is adjusted automatically over the course of training. During the training of the map in each iteration of the particle swarm algorithm, the swarm is perturbed using the current input vector as l_{best}. Just as for the SOM, the training process is unsupervised. Although we apply the developed maps for classification purposes in this study, class labels are only assigned to mapping nodes once the unsupervised training process is complete. An outline of the SOSwarm algorithm is presented below for classification problem instances.

```
initialize particles in mapping layer randomly

for( max number of iterations )
    for( each input training vector in turn )

          set lbest to the value of the input vector

          set pbest of each particle to be its
          current position

          find particle with closest match to lbest

          denote this particle as the firing particle

          update firing particle's velocity
             and position vectors

          update velocity and location vectors
             of neighbors of firing particle

    endfor
endfor

assign class to each particle using training data

calculate classification accuracy using test data
```

To determine the firing particle (particle that is the closest match to an input vector) a simple error distance calculation is adopted.

$$Firing\ particle = \underset{i}{argmin}\ \|V - P_i\| \tag{7}$$

where V corresponds to the input vector, P_i is the i^{th} particle's position vector, and i is the number of particles in the swarm.

A number of alternative error functions could be adopted such as Euclidean distance outlined in equation 8 (where d corresponds to the dimension of the vector or particle).

$$Firing\ particle = \underset{i}{argmin}\ \left\| \sqrt{\sum_{1}^{d} (V_d - P_{id})^2} \right\| \tag{8}$$

TABLE I
DATASET TRAINING AND TEST SET PARTITION SIZES.

Dataset	Training	Test	Total	# variables	# classes
Wisconsin	559	140	699	9	2
Pima	614	154	768	8	2
New thyroid	172	43	215	4	3
Glass	171	43	214	13	6

A visual representation of SOSwarm is presented in Fig. 3 with the adoption of a 2D mapping layer. The particles are arranged a priori into a fixed neighborhood topology (a simple grid in this example). The firing particle, that is, the particle whose position vector is closest to the current input vector (designated as l_{best}) updates its position vector according to the velocity update equations 1-5. In addition,

particles lying within the fixed neighborhood of the firing particle also adjust their position vectors using the same equations. The size of the neighborhood of particles that are updated in each iteration could be reduced over the course of the algorithm. For the experiments conducted in this study, the neighborhoods are fixed on grounds of simplicity.

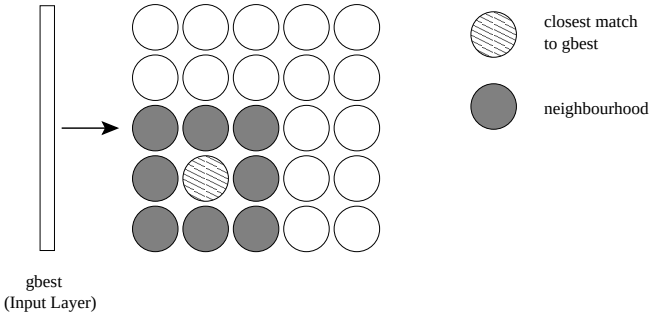

Fig. 3. A Self-Organizing Swarm (SOSwarm) with a 2D mapping layer.

Once the map has been trained, each node in the mapping layer is assigned a class label using a simple majority voting scheme. In calculating the in sample and out sample classification accuracy, the distance between each input data vector and each mapping node is calculated, with the input data vector being assigned the class label of the mapping node it is closest to.

V. Experimental Setup & Results

In order to assess the utility of the SOSwarm algorithm, four classification problems from the UCI Machine Learning Repository [4] are examined. The datasets consist of varying numbers of inputs and known output classes (see Table I). Initially, the SOSwarm clustering algorithm is applied, and then the nodes on the resulting map are labeled. The labeled nodes are then used to classify both the in sample (training) data and the out sample (test) data. We then report the classification accuracies on each dataset.

The following parameters were used for the SoSwarm algorithm, $c_1 = 1.0$, $c_2 = 2.0$, $wmax = 0.9$, $wmin = 0.4$, $cmin = 0$, $cmax = 1$ ($cmax = 10$ for Wisconsin dataset), and $vmax = cmax$. The population of particles was set at 100 (a $10 * 10$ grid structure). The algorithm was run for a total of 10,000 iterations. The parameter values were set after a number of initial trial and error experiments. As the mapping process utilizes a distance metric, the input variables in each dataset were normalized independently in each dimension into the range $[0 \rightarrow 1]$(the Wisconsin dataset was already normalized to the range $[0 \rightarrow 10]$). The datasets were partitioned in training and test sets of the following sizes as illustrated in Table I.

The distance metric in eq. 7 is used to determine the particle that is the closest match, and a fixed grid neighborhood topology is adopted, with the range of the neighborhood as illustrated in Fig. 3. That is, for particles not on the edges of the grid a particle will have at most 8 neighbors, which will be subjected to updates if that particle fires.

Each dataset was recut 10 times between train and test data, and thirty independent runs of the SOSwarm algorithm were conducted on each recut. The classification results obtained for the unseen test data are presented in Table II. The results reported consist of the mean average accuracy, and the mean best accuracy obtained across the thirty runs on each separate data recut. For comparison purposes, the results for the training dataset are also presented (Table III). The results are encouraging with SOSwarm producing a performance surpassing or equaling the best classification accuracies reported in all four problems analyzed.

Comparing the in sample and out sample results, it is notable that the mean best out sample results are similar in quality to the mean best in sample results, indicating that the SoSwarm methodology has generated classifiers, the best of which are generalising well out of sample. In the case of three of the datasets examined, the mean average accuracy has also held up well out of sample.

TABLE II

A comparison of the results obtained on the unseen test data for the Self-Organizing Swarm algorithm across the benchmark problems analyzed averaged across the 10 recuts of the dataset in each case.

Problem	**mean best accuracy (std.dev.)**	**best accuracy**	**UCI reported best accuracy**
Wisconsin breast cancer	0.95 (0.02)	1	0.96
Pima indians diabetes	0.7 (0.03)	0.81	0.76
New thyroid	0.87 (0.04)	1	0.97
Glass	0.54 (0.06)	0.77	0.71

TABLE III

A comparison of the results obtained on the training data for the Self-Organizing Swarm algorithm across the benchmark problems analyzed averaged across the 10 recuts of the dataset in each case.

Problem	**mean avg accuracy (mean std.dev.)**	**mean best accuracy**
Wisconsin breast cancer	0.977 (0.004)	0.985
Pima indians diabetes	0.765 (0.012)	0.789
New thyroid	0.957 (0.013)	0.981
Glass	0.730 (0.037)	0.799

VI. Conclusions & Future Work

This paper presented a novel Self-Organizing Swarm (SOSwarm) algorithm, and illustrated the utility of the algorithm by applying it to four benchmark problems from the UCI Machine Learning repository. Classification accuracies

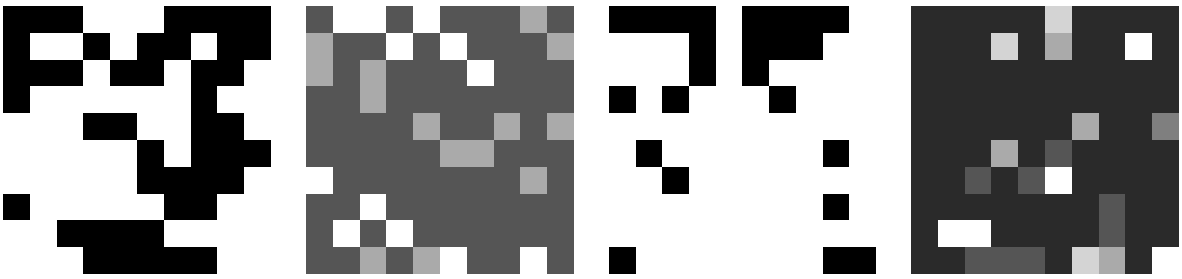

Fig. 4. Example map configurations taken from a random run after training for each of the four problem domains. From left to right they correspond to the Wisonsin breast cancer, New thyroid, Pima Indians diabetes, and Glass problem instances (classes are distinguished with different colour pixels).

reveal that SOSwarm produces competitive results, outperforming or equaling the best reported results on all four problems analyzed.

Given this initial promising study into SOSwarm there are several interesting avenues of future research. A variety of distance metrics could be used in calculating the distance between input vectors and each member of the swarm. In this study, we utilized a simple distance metric, but several other distance metrics could be applied. Another interesting avenue is to investigate the affect of differing neighborhood topologies between the particles in the swarm. It would also be interesting to examine in what circumstances a reducing size of neighborhood over the course of the algorithm would be beneficial. Other possible extensions of the study include the investigation of different swarm sizes, and different velocity update formulations. In addition, although we have applied SOSwarm for classification purposes in this study, it could clearly also be applied for clustering purposes, opening up such potential applications as gene clustering, and customer database segmentation. It would also be interesting to explore the utility of the SOSwarm algorithm for such applications.

REFERENCES

[1] Bonabeau, E., Dorigo, M. and Theraulaz, G. (1999). *Swarm Intelligence: From natural to artificial systems*, Oxford: Oxford University Press.

[2] Brabazon, A. and O'Neill, M. (2006). *Biologically Inspired Algorithms for Financial Modelling*, Berlin: Springer.

[3] Gurney, K. (1997). *An introduction to neural networks*, London: University College London Press.

[4] Hettich, S. & Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases `http://www.ics.uci.edu/~mlearn/MLRepository.html`, Irvine, CA: University of California, Department of Information and Computer Science.

[5] Kennedy, J., Eberhart, R. and Shi, Y. (2001). *Swarm Intelligence*, San Mateo, California: Morgan Kauffman.

[6] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization, *Proceedings of the IEEE International Conference on Neural Networks*, December 1995, pp. 1942-1948, IEEE Press.

[7] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, 43:59-69.

[8] Kohonen, T. (1990). The Self-Organizing Map, *Proceedings of the IEEE*, 78(9):1464-1480.

[9] Kohonen, T. (1998). The SOM Methodology, *in Visual Explorations in Finance with self-organizing maps*, edited by Deboeck, G. and Kohonen, T., p. 159-167, Berlin: Springer-Verlag.

[10] O'Neill, M., Brabazon, A. (2004). Grammatical Swarm, in LNCS 3102 *Proc. of the Genetic and Evolutionary Computation Conference GECCO 2004*, Seattle, WA, USA, pp. 163-174, Springer.

[11] O'Neill, M., Brabazon, A., Adley, C. (2004). The automatic generation of programs for Classification using Grammatical Swarm, in *Proc. of the Congress on Evolutionary Computation CEC 2004*, Portland, OR, USA, pp. 104-110, IEEE Press.

[12] O'Neill, M., Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, Kluwer Academic Publishers.

[13] O'Neill, M. (2001). *Automatic Programming in an Arbitrary Language: Evolving Programs in Grammatical Evolution*, PhD thesis, University of Limerick, 2001.

[14] O'Neill, M., Ryan, C. (2001). Grammatical Evolution, *IEEE Trans. Evolutionary Computation*, 5(4):349-358.

[15] O'Neill, M., Ryan, C., Keijzer M., Cattolico M. (2003). Crossover in Grammatical Evolution. *Genetic Programming and Evolvable Machines*, 4(1):67-93.

[16] Ryan, C., Collins, J.J., O'Neill, M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language, in *Proc. of the First European Workshop on GP*, pp. 83-95, Berlin: Springer-Verlag.

[17] Xiao, X., Dow, E.R., Eberhart, R., Miled, Z.B., Oppelt, R.J. (2004). A hybrid self-organizing maps and particle swarm optimization approach. *Concurrency and Computation: Practice and Experience*, 16(9):895-915.

[18] Xiao, X., Dow, E.R., Eberhart, R., Miled, Z.B., Oppelt, R.J. (2003). Gene-Clustering Using Self-Organizing Maps and Particle Swarm Optimization, in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 22-26 April 2003, Nice, France, IEEE Press.