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 (pbest), and knows the best location found so far by all the particles in the population (gbest) or in an alternative version of the algorithm, a local neighborhood around each particle (lbest). 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 pbest and gbest (or lbest). 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 pbest.
iii. Determine the location of the particle with the highest fitness and revise gbest 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 vi is:

\[ v_i(t + 1) = w \times v_i(t) + (c_1 \times R_1 \times (p_{best} - x_i)) + (c_2 \times R_2 \times (g_{best} - x_i)) \] (1)

where,

\[ w = w_{max} - ((w_{max} - w_{min})/iter_{max}) \times iter \] (2)

In equation 1, pbest is the location of the best solution found to-date by particle i, gbest is the location of the global-best solution found by all particles to date, c1 and c2 are the weights associated with the pbest and the gbest terms in the velocity update equation, x_i is particle i’s current location, and R1 and R2 are randomly drawn from U(0,1). The parameter w represents a momentum coefficient which
is reduced according to equation 2 as the algorithm iterates. In equation 2, \( \text{itermax} \) and \( \text{iter} \) are the total number of iterations the algorithm will run for, and the current iteration value respectively, and \( \text{wmax} \) and \( \text{wmin} \) 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 \( \text{vmax} \). Once the velocity update for particle \( i \) is determined, its position is updated (equation 3), and \( g_{\text{best}} \) is updated if necessary (equations 4 & 5).

\[
x_i(t+1) = x_i(t) + v_i(t+1)
\]

\[
y_i(t+1) = y_i(t) \text{ if, } f(x_i(t)) \leq f(y_i(t))
\]

\[
y_i(t+1) = x_i(t) \text{ if, } f(x_i(t)) > f(y_i(t))
\]

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

\[
\hat{y} \in (y_0, \ldots, y_n) | f(\hat{y}) = \max(f(y_0), \ldots, f(y_n))
\]

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 \( p_{\text{best}} \) and \( g_{\text{best}} \) 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 \( g_{\text{best}} \) 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. 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...
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 \( l_{\text{best}} \). 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.

1. Initialize particles in mapping layer randomly
2. For (max number of iterations)
   1. For (each input training vector in turn)
   2. Set \( l_{\text{best}} \) to the value of the input vector
   3. Set \( p_{\text{best}} \) of each particle to be its current position
   4. Find particle with closest match to \( l_{\text{best}} \)
   5. Denote this particle as the firing particle
   6. Update firing particle’s velocity and position vectors
   7. Update velocity and location vectors of neighbors of firing particle
3. Endfor
4. Endfor
5. Assign class to each particle using training data
6. 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.

\[
\text{Firing particle} = \arg\min_i \| V - P_i \| \tag{7}
\]

where \( V \) corresponds to the input vector, \( P_i \) is the \( i^{\text{th}} \) 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).

\[
\text{Firing particle} = \arg\min_i \left\| \sum_{d=1}^{d} (V_d - P_{id})^2 \right\| \tag{8}
\]

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training</th>
<th>Test</th>
<th>Total</th>
<th># variables</th>
<th># classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wisconsin</td>
<td>559</td>
<td>140</td>
<td>699</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>Pima</td>
<td>614</td>
<td>154</td>
<td>768</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>New thyroid</td>
<td>172</td>
<td>43</td>
<td>215</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Glass</td>
<td>171</td>
<td>43</td>
<td>214</td>
<td>13</td>
<td>6</td>
</tr>
</tbody>
</table>

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 \( l_{\text{best}} \)) 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.

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>mean best accuracy (std.dev.)</th>
<th>best accuracy</th>
<th>UCI reported best accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wisconsin breast cancer</td>
<td>0.95 (0.02)</td>
<td>1</td>
<td>0.96</td>
</tr>
<tr>
<td>Pima indians diabetes</td>
<td>0.7 (0.03)</td>
<td>0.81</td>
<td>0.76</td>
</tr>
<tr>
<td>New thyroid</td>
<td>0.87 (0.04)</td>
<td>1</td>
<td>0.97</td>
</tr>
<tr>
<td>Glass</td>
<td>0.54 (0.06)</td>
<td>0.77</td>
<td>0.71</td>
</tr>
</tbody>
</table>

### Table III

<table>
<thead>
<tr>
<th>Problem</th>
<th>mean avg accuracy (mean std.dev.)</th>
<th>mean best accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wisconsin breast cancer</td>
<td>0.977 (0.004)</td>
<td>0.985</td>
</tr>
<tr>
<td>Pima indians diabetes</td>
<td>0.765 (0.012)</td>
<td>0.789</td>
</tr>
<tr>
<td>New thyroid</td>
<td>0.957 (0.013)</td>
<td>0.981</td>
</tr>
<tr>
<td>Glass</td>
<td>0.730 (0.037)</td>
<td>0.799</td>
</tr>
</tbody>
</table>

### 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
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


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 Wisconsin breast cancer, New thyroid, Pima Indians diabetes, and Glass problem instances (classes are distinguished with different colour pixels).