Neural Network Learning using Particle Swarm Optimizers

MATT SETTLES
School of Engineering
University of Portland
Portland, OR 97203
U.S.A

BART RYLANDER
School of Engineering
University of Portland
Portland, OR 97203
U.S.A.

Abstract: - This paper presents a method to employ particle swarm optimization in a split architecture injected with a plain 'attractor' configuration. This is achieved by splitting the input vector into two even sub-vectors, each of which is optimized in its own swarm. Then, a plain 'attractor' is injected into each swarm. The application of this technique to neural network training is investigated.

Key-Words: - Particle Swarm, Neural Networks, Split Swarm

1 Introduction

One of the first implementations of Particle Swarm Optimization (PSO) was that of training neural networks [1]. One key advantage of PSO over other optimization algorithms in training neural networks is its comparative simplicity.

This paper describes the results of experimental attempts to improve the performance of the basic PSO by splitting the input vector into two sub-vectors. To do this each sub-vector is allocated its own swarm. A plain swarm containing the entire input vector is then used as an 'attractor' for the two sub-vectors.

PSO is a form of evolutionary computing [2]. As described by Eberhart and Kennedy, the PSO algorithm is an adaptive algorithm based on a social-psychological metaphor; a population of individuals adapts by returning stochastically toward previously successful regions in the search space, and is influence by the successes of their topological neighbors[1].

Various attempts have been made to improve the performance of the baseline PSO with varying success. Eberhart and Shi focus on optimizing the update equations for the particles [1]. Angeline used a selection mechanism in an attempt to improve the general quality of the particles in a swarm [5]. Kennedy uses cluster analysis to modify the update equation so that particles attempt to conform to the center of their clusters rather than attempting to conform to a global best.

A swarm consists of many particles, where each particle keeps track if its position, velocity, best position thus far, best fitness thus far, current fitness, and neighboring particles. The most important attribute being its current position, given by an n-dimensional vector, which corresponds to a potential solution of the function to be minimized.

The velocity vector keeps track of the speed and direction the particle is currently traveling. The particle keeps track of its current fitness (analogous to population members in GA), which is obtained by evaluating an error function at the particle's current position. The best fitness value thus far is retained as well as the particles position at that fitness value. Finally the particle also keeps track of its nearest neighbor's fitness values, for use in updating its velocity.

For a neural network implementation, the fitness value corresponds to a forward propagation through the network, and the position vector corresponds to the weight vector of the network. The particle's best neighbor (yielding the lowest error) and the global best particle are used to guide the particle to new solutions. At the end of the algorithm, the global best particle's position serves as the answer.

During each epoch every particle is accelerated towards its best neighboring position as well as in the direction of the global best positions. This is achieved by calculating a new velocity term for each particle based on its current velocity, the distance from its best neighbor, as well as the distance from the global best position. An inertia weight, reduced linearly by epoch, is multiplied by the current velocity and the other two components are weighted randomly to produce the new velocity value for this

particle, which in turn affects the next position of the particle during the next epoch.

2.1 Particle Swarm Parameters

A number of factors will affect the performance of the PSO. Firstly, the number of particles in the swarm affects the run-time significantly, thus a balance between variety (more particles) and speed (less particles) must be sought. Another important factor in the convergence speed of the algorithm is the maximum velocity parameter. This parameter limits the maximum jump that a particle can make in one step, thus a too large value for this parameter will result in oscillations, while a too small value could cause the particle to become trapped in a local minima.

For the implementation used in this paper, the maximum velocity parameter is set to the default value of 2 (this value was suggested in earlier works in PSO)[1].

2.2 Swarm Behavior

The behavior of the swarm is dictated by the summation of the behaviors of the particles. Each particle "flies" in the direction of a better solution, weighted by some random factor, maybe overshooting, or maybe finding a better or global better position. The interaction between the particles in the swarm helps to prevent straying off, while keeping close to the optimal solution.

This type of behavior seems ideal when exploring large search spaces, especially with a relatively large maximum velocity parameter. Some particles will explore far beyond the current minimum, while the population still remembers the global best. This seems to solve one of the problems of gradient-based algorithms.

2.3 Multi-Swarms

To employ a multi-swarm the solution vector is split amongst the different populations according to some rule; the simplest of the schemes does not allow any overlap between the spaces covered by different populations. To find a solution to the original problem, representatives from all the populations are combined to form the potential solution vector, which, in turn, is passed on to the error function. This adds a new dimension to the survival game: cooperation between different populations.

3 Implementation

The multi-swarm approach involves a trade-off. Increasing the number of swarms leads to a directly proportional increase in the number of error function evaluations. One function evaluation is required for each particle, in each swarm during each epoch. The total number of error function evaluations can be summarized as:

$$F = S \times P \times E \quad (1)$$

where F is the number of function evaluations, S is the number of swarms used, P is the number of particles per swarm and E the number of epochs allowed for training.

3.1 Neural Network Architectures

Neural networks are trained by minimizing an error function in

$$W = (D + 1) \times M + (M + 1) \times C \quad (2)$$

PSOs generate potential solutions is this W-dimensional space using a forward propagation through the neural network to obtain the value of he error function for each particle in the swarm. This error value is then used directly as the particle's, fitness. So, constructing a particle with a lower fitness is synonymous with learning in the network. A forward propagation through the network is a computationally expensive task, so the aim is to find the best possible solution using a limited number of forward propagations through the network.

To test the effectiveness of a split swarm with that of the injected plain swarm, a number of tests were performed on a two – layer feed forward neural network. The following architectures were tested.

Plain architecture - The plain architecture is the standard PSO used for most implementations. It contains just a single swarm and takes the weights for the different layers to produce a W – dimensional vector. This architecture is used as the baseline for the benchmarks.

Esplit architecture - The Esplit, or 'Even' split architecture takes the weights for the different layers and evenly divides them in two. During each epoch the error function is evaluated twice, once for each swarm. The number of epochs is halved to keep the number of error function evaluations constant.

Mixed Plain/Even architecture - The Mixed architecture uses both a plain swarm and an Esplit swarm. The Esplit swarm is evaluated as normal, the particles with the best positions are injected into a plain swarm and the error function is evaluated again. In this architecture the error function is evaluated three times per epoch. For this test the number of epochs is divided by three to keep the number of error function evaluations constant.

4 Case Study – the Iris Data Set

The Iris data set is often considered the baseline for learning algorithms [4]. This data set contains 150 patterns falling into three classes. This is considered to be a simple classification example, as the three classes are linearly separable. A 4-input, 3-layer, and 3-output network architecture was used.

For the purpose of this test 100 random Iris patterns were used for the training session and the remaining 50 used for testing. Experiments were performed using 100 runs per architecture.

Table 1 displays the findings using the three different architectures. The mixed-split/plain architecture performed as well as the even split architecture, but converged much quicker and appears to develop a much more coherent swarm.

	Epoch	Best Error (Train)	Ave. Error (Train)	Error % (Test)
Plain	5000	.010	.032	4.9
Esplit	2500	.13	.43	3.2
Mixed	1667	.014	.014	3.3

Table 1, Comparison of three architectures.

The number of epochs used for each architecture type, normalized the total number of error functions calculated. In all cases a maximum of 5000 error functions were allowed to be calculated. During the training session the error was calculated by using the average sum squared of the differences. Finally the error percentage is computed by the number of wrong answers, during the testing phase, divided by the total.

5 Conclusions

Splitting the vector to be optimized across two swarms for neural network training appears to improve performance on the Iris data set. Injecting in another error calculation via a plain swarm appears to offer the same improved performance, but in less epochs and with a greater consistency among the swarm. A greater consistency among the swarm should provide a more reliable output, with less variance over multiple runs.

The results indicate the mixed split/plain architecture holds promise. A more in-depth analysis of other neural nets is needed before a conclusion can be reached. It would also be interesting to see how well the mixed architecture performs on function minimizations.

References:

[1] J. Kennedy and R. Eberhart. *Swarm Intelligence*.Morgan Kaufmann Publishers, 2001.

[2] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 1995.

[3] *Neural Network FAQ Chapter 1*, pages 8-37,1997.*ftp://ftp.sas.com/pub/neural/FAQ.html*

[4] C. Blake, E. Keogh, and CJ. Merz. *UCI repository of machine learning databases,* 1998. *www.ic.uci.edu/~mlearn/MLRepository.html*

[5] F. van den Bergh and AP Engelbrecht. Cooperative Learning in Neural Networks using Particle Swarm Optimizers, *SACJ / SART,* No 26, 2000