PSOt - a Particle Swarm Optimization Toolbox for use with Matlab

Brian Birge
NCSU, MAE Dept.
726 N. West St., #B
Raleigh, NC 27603
birgeb@bellsouth.net

Abstract - A Particle Swarm Optimization Toolbox (PSOt) for use with the Matlab scientific programming environment has been developed. PSO is introduced briefly and then the use of the toolbox is explained with some examples. A link to downloadable code is provided.

I. GENERAL INFORMATION

A. Particle Swarm Optimization Toolbox (PSOt), Summary of Included Files

Main files:

1) PSO – finds min/max of arbitrary MISO functions using PSO
2) trainPSO – Function to train neural nets using PSO
3) tpso1,2,3 – called by trainPSO for training 0 hidden layer, 1 hidden layer, and 2 hidden layer ANNs respectively.

Support files:

4) wrapmat – convert vector(s) into matrix(ces)
5) unwrapmat – convert any 2d matrix(ces) into a row vector(s)
6) normalize – takes a 2D matrix and reformats it to any specified range
7) goplotpso – called by trainpso for graphical display during training

Demo and miscellaneous files:

8) DemoTrainPSO – shows an example of using PSO to train a neural net to the XOR function
9) f6 – Schaffer's f6 function
10) hypara – 4 dimensional hyperbolic paraboloid function
11) hypara2 – 8 dimensional hyperbolic paraboloid function

The toolbox may be downloaded freely as a self extracting zipfile from: http://www4.ncsu.edu/~bkbirge/PSO/PSOt.exe

B. Introduction

With the growth of more mainstream interest in computational intelligence algorithms applied to real-world engineering problems there is a need to build tools that can be used in a wide variety of environments, from classroom to field. Particle Swarm Optimization (PSO) is one recent technique that has been used with great success in the Computational Intelligence arena. With the Particle Swarm Optimization Toolbox (PSOt), users can apply the algorithm to their problem without having to write their own PSO code or really even understand much about PSO. Users can train an Artificial Neural Network (ANN) with PSO and/or apply PSO to their more general problem.

This paper originally came out of an effort to replace standard backpropagation trained ANNs with PSO in order to make better ANNs and was inspired by the freely available Matlab implementation of a Genetic Algorithm function optimizer by Houck, Joines, and Kay referenced at the end of this paper [3].

It is hoped this suite of programs will be useful to classroom lecturers wishing to add a PSO unit as well as fellow researchers in Computational Intelligence.

C. Matlab

Matlab is a scientific computing language developed by Mathworks that is run in interpreter mode on a wide variety of operating systems. It is extremely powerful, simple to use, and can be found in most research and engineering environments. It gets more powerful and specialized with the addition of 'toolboxes', additional functions added to the Matlab environment by various developers for specific tasks or fields. For example, the Neural Net toolbox adds functions that allow one to develop ANNs and the Control System toolbox includes many traditional engineering controls functions. The PSOt aims to add not only stand alone function optimization using PSO but also the ability to interface easily with the Neural Net toolbox, training ANNs with PSO rather than backpropagation. The reader is expected to have a basic understanding of Matlab to use the PSOt. Please refer to the appropriate Matlab documentation to understand the code syntax examples [6].

D. Particle Swarm Optimization (PSO)

Inspired initially by flocking birds, Particle Swarm Optimization (PSO) is another form of Evolutionary Computation and is stochastic in nature much like Genetic Algorithms. Instead of a constantly dying and mutating GA population we have a set number of particles that fly through the hyperspace of the problem. A minimization (or maximization) of the problem topology is found both by a particle remembering its own past best position and the entire

group's (or flock's, or swarm's) best overall position. This algorithm has been shown to have GA like advantages without the big computational hit.

The PSO algorithm is based on the concept that complex behavior follows from a few simple rules [5].

The intricacies of the algorithm are briefly outlined here. For more details and discussion see the references, especially [4], [5], and [8].

II. TECHNICAL INFORMATION

A. PSO Algorithm

The Basic PSO algorithm consists of the velocity and position equations:

$$v_i(k+1)=v_i(k)+g_{1i}(p_i-x_i(k))+g_{2i}(G-x_i(k)) \quad (1)$$

$$x_i(k+1)=x_i(k)+v_i(k+1) \quad (2)$$

i – particle index
k – discrete time index
v – velocity of i^{th} particle
x – position of i^{th} particle
p – best position found by ith particle (personal best)
G – best position found by swarm (global best, best of personal bests)
$\gamma_{1,2}$ – random numbers on the interval [0,1] applied to i^{th} particle

The most used PSO form is to include an inertia term and acceleration constants, hence the Common PSO algorithm:

$$v_i(k+1) = f(k)\,v_i(k) + a_1\left[g_{1i}(p_i-x_i(k))\right] + a_2\left[g_{2i}(G-x_i(k))\right] \quad (3)$$

ϕ - Inertia function
$\alpha_{1,2}$ – Acceleration constants

The inertia function is commonly either taken as a constant 1.4 or as a decreasing linear function in k from 0.9 to 0.4. As training progresses using a linear inertia function, the influence of past velocity becomes smaller.

Reference [8] shows the acceleration constants are most commonly set to both equal 2.

There are also a few more important parameters. These are initial positions of particles (initial weights when training a neural net), the maximum velocity bounds, and number of particles in the swarm.

B. Development/Program Usage

The set of programs that comprise the PSOt are designed with two purposes in mind. The first purpose is to act as a standalone function optimizer. The second purpose is to replace backpropagation with PSO during neural net training.

C. Finding global min/max of MISO function using PSO

The first goal is achieved with the Matlab script 'PSO.m'. It is very simple to use but can be very flexible as well. The function only works with single input single output (SISO) or multi-input single output (MISO) functions that need all the input space searched. It will work with any size dimension of input space, the toolbox miscellaneous functions include 2D, 3D, and 7D functions to experiment with. The PSO function can either minimize or maximize the target function. The basic syntax can be found by typing 'help PSO'.

TABLE I
MATLAB OUTPUT FOR TYPING 'HELP PSO'

```
%[optOUT]=PSO(functname,D), or:
%[optOUT,tr,te]=
%        PSO(functname,D,VarRange,minmax,PSOparams)
%
%Inputs:
% functname-
%   name (string) of matlab function to optimize
% D-
%   # of inputs to the function (problem dimension)
%
%Optional Inputs:
% VarRange-
%  matrix of ranges for input variable,
%  default -100 to 100, form:
%       [ min1 max1
%         min2 max2
%           ...
%         minD maxD ]
%
% minmax-
%  if 0 then funct is minimized
%  if 1 then funct maximized, default=0
```

The 'PSOparams' option is where the PSO specific parameters can be changed.

TABLE II
'PSO.M' PARAMETERS

```
%PSOparams - PSO parameters
%
%P(1)-
%  Epochs between updating display,
%  works with P(13), default = 25.
%
%P(2)-
%  Maximum number of iterations (epochs) to train,
%  default = 2000.
%
%P(3)-
%  population size, default = 20
```

```
%P(4)-
%  maximum particle velocity, default = 4
%
%P(5)-
%  acceleration const #1 (local best influence),
%  default = 2
%
%P(6)-
%  acceleration const #2 (global best influence),
%  default = 2
%
%P(7)-
%  Initial inertia weight, default = 0.9
%
%P(8)-
%  Final inertia weight, default = 0.2
%
%P(9)-
%  Epoch by which inertial weight = final value,
%  default = 1500
%
%P(10)-
%  randomization flag,
%   = 0, random for each epoch
%   = 1, random for each particle at each epoch
%   = 2, random for each dimension of each particle
%        at each epoch
%
%P(11)-
%  minimum global error gradient,
%   if abs(Gbest(i+1)-Gbest(i)) < gradient over
%   certain length of epochs, terminate run,
%   default = 1e-9
%
%P(12)-
%  epochs before error gradient criterion
%  terminates run, default = 50
%
%P(13)-
%  plot flag, shows progress display if =1,
%  default = 1
```

The output of the function is returned as a column vector of the final global best position. The last entry in the vector is the global best value. Optional outputs are the # of epochs it took to reach consensus that optimization had occurred and the value of global best as a function of epoch.

A simple example would be to find the minimum of the sine function from 0 to $\pi/2$:

pso('sin',1,[0,pi/2],0)

The name of the function is surrounded by single quotes, the dimension of the problem is 1 (only one input so only one searchable dimension), the range is from 0 to $\pi/2$, and the minmax flag is set to 0 indicating we want to find the minimum.

Naturally we expect it to return the column vector [0,0]' which it does. It finds it after 1 epoch (iteration) of the algorithm but doesn't stop looking until 50 epochs have gone by with no change in the best global position. That can be changed with P(12).

A more interesting example is the Schaffer f6 function. The function has 2 inputs and one output and looks like a still shot of a pebble dropped into water. It has circular ripples that die out as the radius increases. It has a known global minimum at x,y = 0 and the maximum is the first 'ripple' around the origin. It has lots of local minimums and maximums making it a good test function for the algorithm.

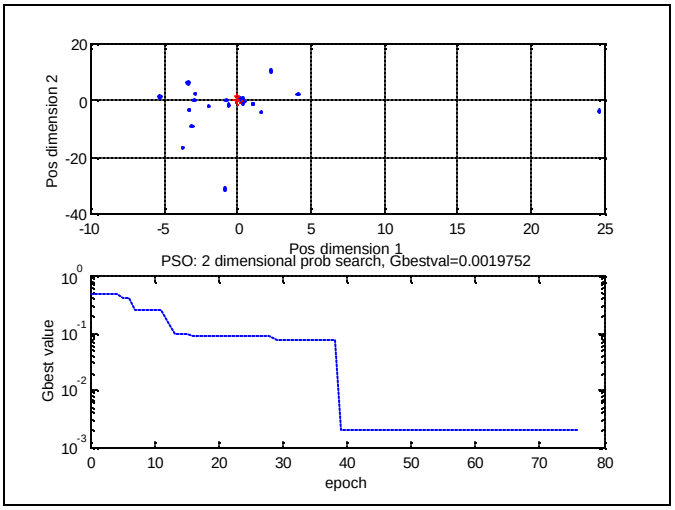
Running the PSO function to find the minimum bound by -100 = x,y = 100 gives the following table.

TABLE III
FINDING THE MINIMUM OF THE 'F6' FUNCTION

```
>> pso('f6',2,[-100,100;-100,100],0)
PSO: 1/2000 iterations, GBest = 0.499643.
PSO: 26/2000 iterations, GBest = 0.0858499.
PSO: 51/2000 iterations, GBest = 0.00197516.
PSO: 76/2000 iterations, GBest = 0.00197516.
PSO: 89/2000 iterations, GBest = 0.00197516.

***** global error gradient too small for too long
***** this means you've got the solution or it got
stuck
ans =
   -4.400531404313024e-002
    4.527939759978006e-003
    1.975158000277422e-003
```

FIGURE I
USING 'PSO.M' TO FIND MINIMUM OF 'F6'



Running the 'PSO.m' function to find the maximum of f6 with the same bounds yields the following table.

TABLE IV
FINDING THE MAXIMUM OF THE 'F6' FUNCTION

```
>> pso('f6',2,[-100,100;-100,100],1)
PSO: 1/2000 iterations, GBest = 0.500179.
PSO: 26/2000 iterations, GBest = 0.973786.
```

```
PSO: 51/2000 iterations, GBest = 0.976315.
PSO: 76/2000 iterations, GBest = 0.976362.
PSO: 101/2000 iterations, GBest = 0.976432.
PSO: 126/2000 iterations, GBest = 0.976432.
PSO: 145/2000 iterations, GBest = 0.976432.

***** global error gradient too small for too long
***** this means you've got the solution or it got
stuck
ans =
   -1.399320469183342e+000
    6.794290702799257e-001
    9.764318246838789e-001
```

For a fun exercise, you can set the maximum velocity parameter to a small fraction, such as 0.04 and set the update display frequency to do so at every iteration. This will really show the behavior of the PSO algorithm, showing the particles 'fly' towards the global best and explore new areas. The reader is encouraged to change all of the parameters around and examine the changes in behavior of the PSO.

D. Training Neural Net using PSO

To train a neural net using the PSO algorithm the function 'trainPSO.m' is used. This function requires that the Neural Net toolbox be installed for use with your copy of Matlab. TrainPSO is modeled after the structure of the Neural Net toolbox and can be used interchangeably with the other training functions such as TrainBP, TrainLM, etc. The only difference is in the training parameters passed to the training algorithm. The format of the function is:

[W1,B1,W2,B2,...,TE,TR] =
TRAINPSO(W1,B1,F1,W2,B2,F2,...,P,T,TP)

W1, W2, W3, B1, B2, B3 are weights and biases for the neural net, initially made randomly or with the built in initialization function that comes with the Neural Net toolbox. F1, F2, F3, are activation function choices such as 'logsig', 'tansig', etc. Refer to the Neural Net toolbox documentation for more details on those and the other parameters [6]. The PSO specific training parameters show up in TP. TP is a row vector of 15 values. They cover the same ground as the straight PSO function with a couple of parameters specific to neural net training.

Typing 'help trainpso' yields the training parameter summary and is shown in the following table.

TABLE V
'TRAINPSO.M' PARAMETERS

```
%Training parameters are:
%TP(1)-
%  Epochs between updating display, default = 100.
%
%TP(2)-
%  Maximum number of iterations (epochs) to train,
%  default = 4000.
%
%TP(3)-
%  Error goal, default=0.02.
%
%TP(4)-
%  Population size, default = 20
%
%TP(5)-
%  Maximum particle velocity, default= 4
%
%TP(6)-
%  Acceleration constant #1, default = 2
%
%TP(7)-
%  Acceleration constant #2, default = 2
%
%TP(8)-
%  Initial inertia weight, default = 0.9
%
%TP(9)-
%  Final inertia weight(iwt),default=0.2
%
%TP(10)-
%  Epoch by which inertial weight = final value,
%  default = 1500
%
%TP(11)-
%  Maximum initial network weight absolute value,
%  default = 100
%
%TP(12)-
%  Randomization flag, default = 2:
%   = 0, random for each epoch
%   = 1, random for each particle at each epoch
%   = 2, random for each dimension of each particle
%        at each epoch
%
%TP(13)-
%  Minimum global error gradient
%     (if SSE(i+1)-SSE(i) < gradient over certain
%      length of epochs, terminate run,
%      default = 1e-9
%
%TP(14)-
%  Error gradient criterion terminates run here,
%  default = 200
%
%TP(15)-
%  Plot flag, if = 1, display is updated
```

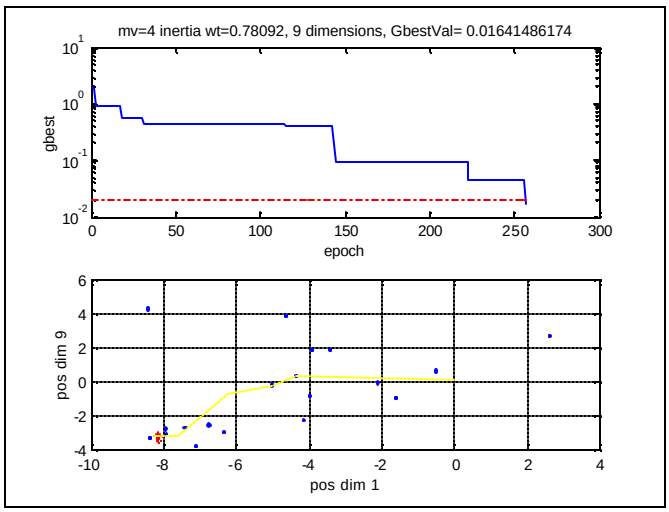
Here is a run of 'DemoTrainPSO' where there is one hidden layer of 2 neurons (9 dimensional optimization problem). The net is trained to approximate the XOR function. It displays the evolution of the inertia weight (iwt) over time as well as the training error and particle positions. The graph also shows the trajectory of the global best position. Only a portion of the training is shown.

TABLE VI
MATLAB OUTPUT FOR 'DEMOTRAINPSO.M'

```
TRAINPSO: 100/1000 epochs,  gbest SSE = 0.428059141
  mv  = 4,  iwt = 0.8537691795
TRAINPSO: 125/1000 epochs,  gbest SSE =
0.3870019042
  mv  = 4,  iwt = 0.8420947298
```

```
TRAINPSO: 150/1000 epochs,  gbest SSE =
0.0943753488
  mv  = 4,  iwt = 0.8304202802
TRAINPSO: 175/1000 epochs,  gbest SSE =
0.0943753488
  mv  = 4,  iwt = 0.8187458306
TRAINPSO: 200/1000 epochs,  gbest SSE =
0.0943753488
  mv  = 4,  iwt = 0.8070713809
TRAINPSO: 225/1000 epochs,  gbest SSE =
0.04639357585
  mv  = 4,  iwt = 0.7953969313
TRAINPSO: 250/1000 epochs,  gbest SSE =
0.04454761798
  mv  = 4,  iwt = 0.7837224817
**************************  Reached Goal
******************
TRAINPSO: 256/1000 epochs,  gbest SSE =
0.01641486174
  mv  = 4,  iwt = 0.7809206137
*************************** end of training
***************
```

FIGURE II
USING 'TRAINPSO.M' TO APPROXIMATE XOR



E. Conclusion

The suite of functions and programs included with the Particle Swarm Optimization Toolbox are useful both in researching PSO behavior and in applying PSO to real world optimization and computational intelligence problems. The PSO algorithms implemented here are computationally intensive and are missing some features that would improve flexibility and work continues on improvement. As it stands, the toolbox allows the researcher/engineer to utilize PSO without having to write custom code from the ground up. The included comments make the suite easily modified to fit more specific types of problems.

F. References

[1] Eberhart, R., Simpson, P., Dobbins, R., Computational Intelligence PC Tools, pp. 212-223, 1996, Academic Press, Inc.

[2] Haykin, S., Neural Networks, a Comprehensive Foundation, Second Edition, 1999, Prentice-Hall.

[3] Houck, C., Joines, J., and Kay M., A Genetic Algorithm for Function Optimization: A Matlab Implementation, *ACM Transactions on Mathematical Software*, Submitted 1996

[4] Kennedy, J., Eberhart, R., Particle Swarm Optimization, *Proc. IEEE Int'l. Conf. on Neural Networks* (Perth, Australia), IEEE Service Center, Piscataway, NJ, **IV:** 1942-1948, 1995

[5] Kennedy, J., Eberhart, R., Shi, Y., Swarm Intelligence, 2001, Academic Press, Inc.

[6] Matlab Online Help, http://www.MathWorks.com

[7] Parsopoulos, K.E., Plagianakos, V.P., Magoulas, G.D., Vrahatis, M.N., Stretching Technique for obtaining global minimizers through Particle Swarm Optmization, *Proc. Particle Swarm Optimization Workshop*, (Indianapolis, IN, USA), pp. 22-29, 2001

[8] Shi, Y., Eberhart, R., Parameter Selection in Particle Swarm Optimization, *Proceedings of the Seventh Annual Conf. on Evolutionary Programming*, pp. 591-601, 1998

[9] van den Bergh, F., Particle Swarm Weight Initialization In Multi-Layer Perceptron Artificial Neural Networks, *In Development and Practice of Artificial Intelligence Techniques* (Durban, South Africa), pp. 41-45, Sept. 1999.