Kyung-Min Hong

Energy performance of residential buildings (Part 2) Particle swarm optimization

Outline

1. Background information & Objective
2. Particle swarm optimization
3. Results of optimization
4. Issues of optimization
5. Conclusions

Background information & Objective

- When it comes to efficient building design, the computation of the heating load (HL) is required to determine the specifications of the heating equipment
- The objective of the previous project was to model heating load of buildings using neural networks
- The objective of part two of the project is optimizing building design parameters to achieve the lowest HL based on the neural network model

Ref: A. Tsanas, A. Xifara. (2012). Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools', Energy and Buildings, Vol. 49, pp. 560-567

Dataset

Mathematical representation	Input or output variable	Number of values
X1	Relative compactness	12
X2	Surface area	12
X3	Wall area	7
X4	Roof area	4
X5	Overall height	2
X6	Orientation	4
X7	Glazing area	4
X8	Glazing area distribution	6
Y1	Heating load	586

Ref: A. Tsanas, A. Xifara. (2012). Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools', Energy and Buildings, Vol. 49, pp. 560-567

Particle swarm optimization

The goal of the particle swarm optimization is to find a solution which $f(a) \leq f(b)$, for all search space b.

Steps

1. Initialize Population

 (Random initial positions),

 (Random initial velocities)

2. Evaluate each particle's position according to the objective function.
3. If a particle's current position is better than its previous best position, update it.
4. Determine the best particle (according to the particle's previous best positions).
5. Update particles' velocities according to equation 2
6. Move particles to their new positions according to $x_i(k+1) = x_i(k) + v_i(k+1)$ (1)
7. Go to step 2 until stopping criteria are satisfied.

Ref: Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization.*Swarm intelligence*, *1*(1), 33-57.

Particle swarm optimization

Equations

$$v_i(k+1) = \underbrace{\emptyset(k)v_i(k)}_{\text{Inertia}} + \underbrace{\alpha_1[\gamma_{1i}(p_i - x_i(k))]}_{\text{Personal influence}} + \underbrace{\alpha_2[\gamma_{2i}\big(G - x_i(k)\big)]}_{\text{social influence}} \qquad (2)$$

i – particle index
k – discrete time index
v– velocity of ith particle
x – position of ith 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 ith particle
$\emptyset$- inertia function
α- Acceleration constants

Optimization Procedure

1. Train the feed forward neural network using back propagation technique
2. Create a standalone function of the neural network trained by back propagation
3. Implement the standalone function to the particle swarm optimization to find a set of parameters that result in lowest HL

Program demo

```
clc
clear all

load('input.mat')
load('Target.mat')

inputs = input';
targets = Target';

% Create a Fitting Network
hiddenLayerSize = 30;
net = fitnet(hiddenLayerSize);

net.inputs{1}.processFcns = {'removeconstantrows','mapminmax'};
net.outputs{2}.processFcns = {'removeconstantrows','mapminmax'};


% For a list of all data division functions type: help nndivide
net.divideFcn = 'dividerand';  % Divide data randomly
net.divideMode = 'sample';  % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

net.trainFcn = 'trainlm';  % Levenberg-Marquardt used

net.performFcn = 'mse';  % Mean squared error

net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
  'plotregression', 'plotfit'};


% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs);

trainTargets = targets .* tr.trainMask{1};
valTargets = targets  .* tr.valMask{1};
testTargets = targets  .* tr.testMask{1};
```

Feed forward neural Network performance validation

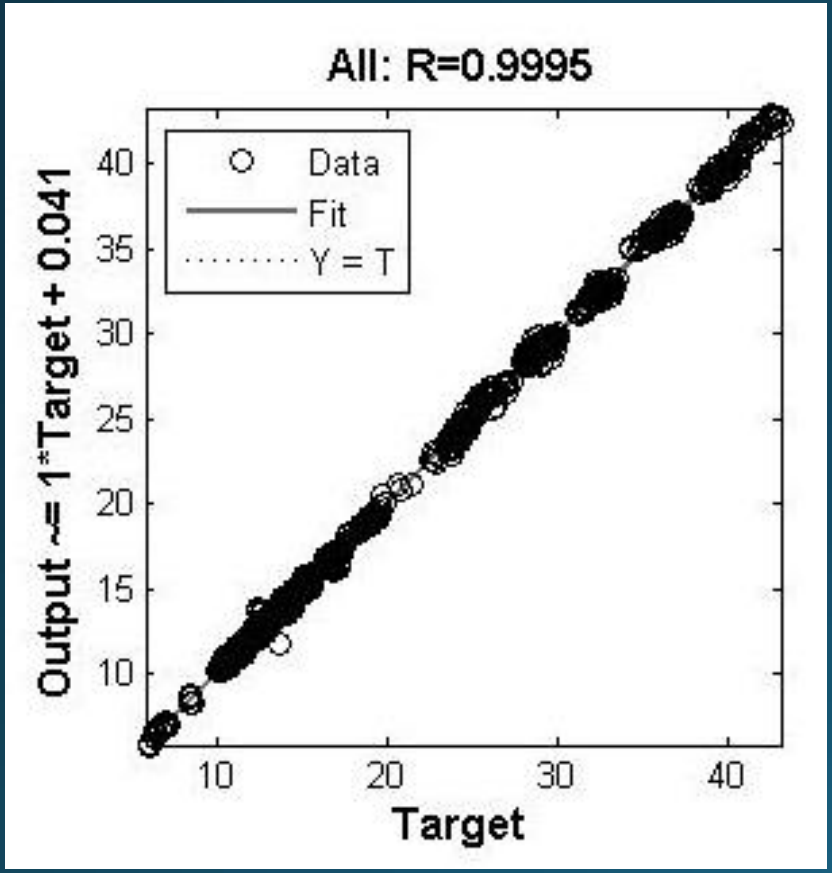

Figure 1. Regression of all datasets

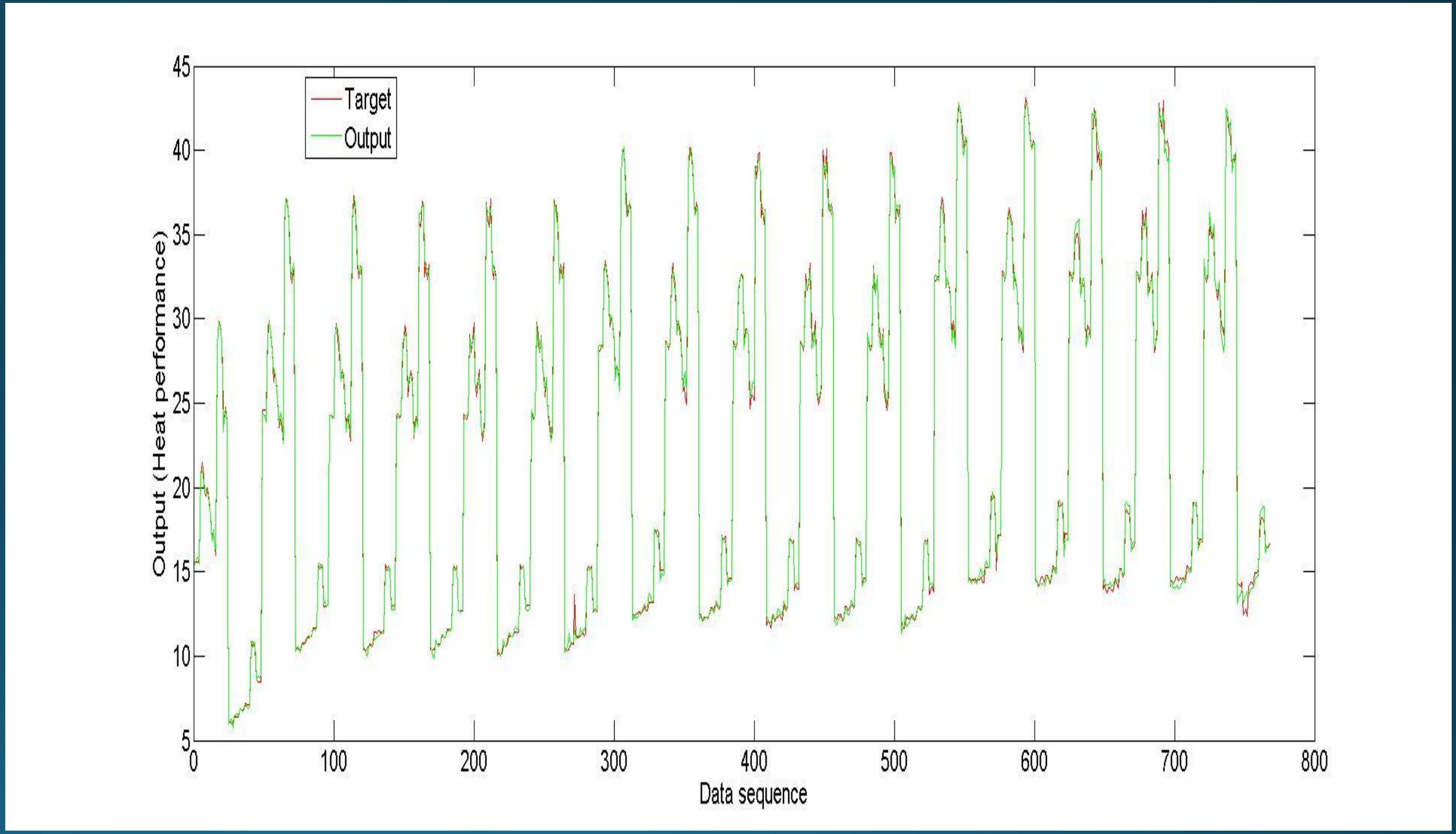

Figure 2. Output from the network and data

Optimization results

The lower and upper boundaries for the search space were limited to the lower and upper boundaries of the data set.

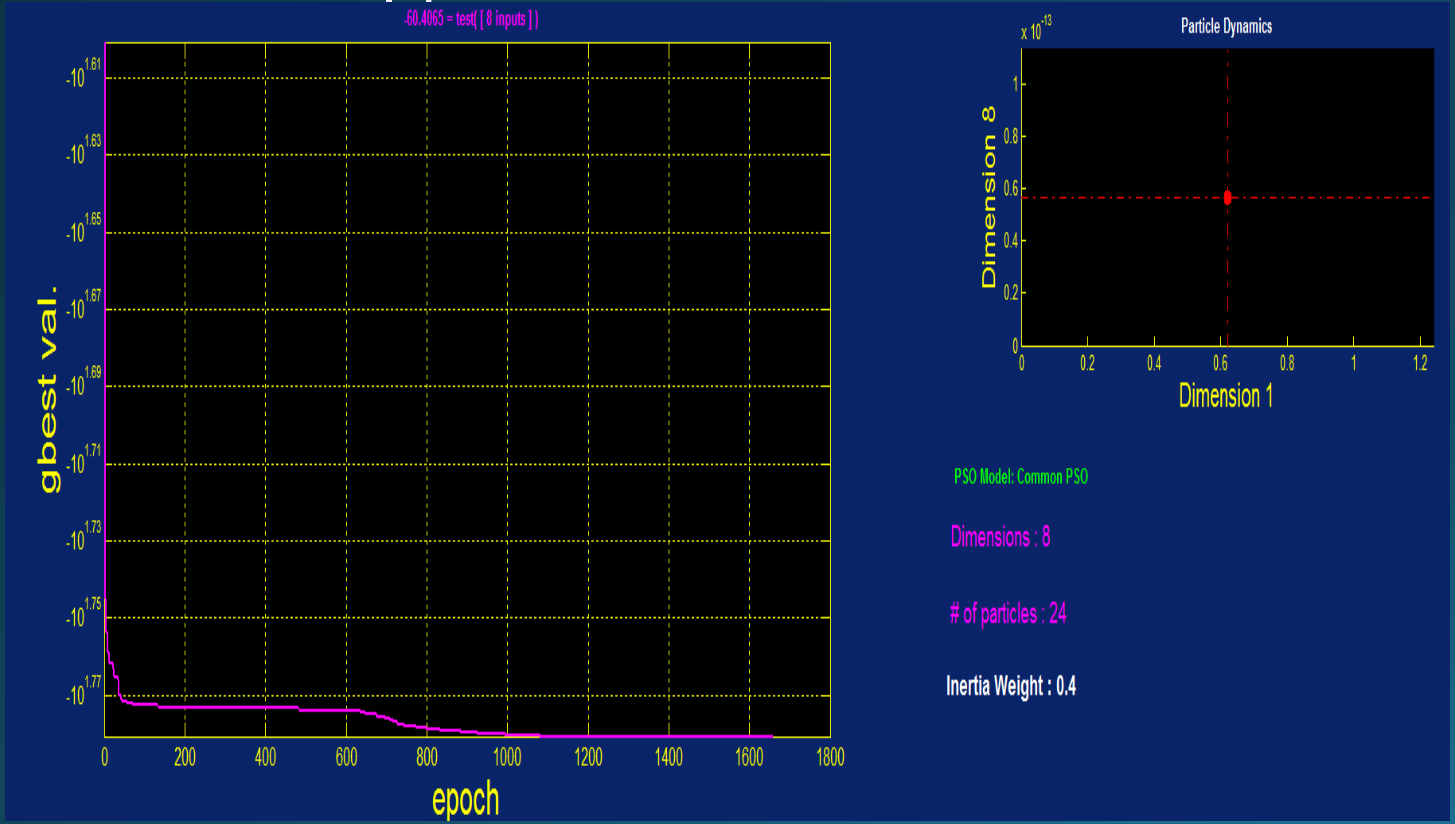

Figure 3. Results of the optimization

Optimized inputs and out put resuls	Values
Relative compactness	0.7804
Surface area	808.5
Wall area	7 245.0
Roof area	112.9486
Overall height	3.5
Orientation	2
Glazing area	0
Glazing area distribution	3.3309
Heating load	-41.1226

Problems with optimization results

1. Every execution of the code resulted in different results

-Randomness of the feed forward neural network contributed to different values of minimum HL

EX) -60.4065,-160.1574, -15.6636, -41.1226

2. The heating load values would result in non-physical negative values

-The cause of the error was due to the lack of data.

To cover a full factorial database you need 5X6X7X4X5X4X7x3=352800, which is significantly larger than 586 datasets used for training.

The reason for the good fit for the neural network validation is because the test data is in the similar group as the training data

Conclusions

- Feedforward neural network is not suitable for optimization when it is trained with small data sets
- PSO does not use the gradient of the problem being optimized, which means PSO does not require that the optimization problem be differentiable
- PSO has small computational cost due to simplicity in algorithm
- Further work is recommended in optimizing models with smaller number of variables to check the feasibility of the optimization of neural networks

Questions?