Evolving Resilient Back-Propagation Algorithm for Energy Efficiency Problem

Yang Fei , Gao Pengdong and Lu Yongquan

High Performance Computing Center, Communication University of China, Beijing, China

Abstract. Energy efficiency is one of our most economical sources of new energy. When it comes to efficient building design, the computation of the heating load (HL) and cooling load (CL) is required to determine the specifications of the heating and cooling equipment. The objective of this paper is to model heating load and cooling load buildings using neural networks in order to predict HL load and CL load. Rprop with genetic algorithm was proposed to increase the global convergence capability of Rprop by modifying a corresponding weight. Comparison results show that Rprop with GA can successfully improve the global convergence capability of Rprop and achieve lower MSE than other perceptron training algorithms, such as Back-Propagation or original Rprop. In addition, the trained network has better generalization ability and stabilization performance.

1 Introduction

Artificial intelligence techniques have emerged as powerful tools that could be used to replace time-consuming procedures. In the last decade, research into ANNs has shown explosive growth, the interest showed to them is mainly due to their ability resides in appealing properties they exhibit: adaptability, learning capability, and ability to generalize [1]. ANNs were developed initially to model biological functions, they learn from experience and rapidly solve hard computational problems. With the spread of computers, later research was directed at exploring the possibilities of using and improving them for performing specific tasks. Nowadays, ANNs are receiving a lot of attention from the international research community with a large number of studies concerning training, structure design, and real world applications, ranging from classification to robot control or vision [2]-[4]. The neural network training task is a capital process in supervised learning, in which a pattern set made up of pairs of inputs plus expected outputs, is known beforehand, and used to compute the set of weights that makes the ANN to learn it.

Back-propagation (BP) [5] is the most popular ANN training algorithm. It is a gradient descent method that aims at minimizing the total root mean squared error (RMSE) between actual and desired outputs of a network by employing gradient information. BP imposes limitations on a type of the ANN activation function. It requires the activation function to be differentiable. Although BP has been successfully applied to various problems, it has a number of drawbacks due to its gradient descent nature. Among shortcomings of BP are slow learning process, tendency to get trapped in local minimum easily, and difficulties in training large ANNs, i.e., ANNs with large number of layers and neurons [6]. Resilient back-propagation (Rprop) is considered the best algorithm, measured in terms of convergence speed, accuracy and robustness with respect to training parameters [7]. Unlike BP algorithm, Rprop do not need to calculate derivatives of the error function, it use only the sign of the derivative to determine the direction of the weight update and thus, can work with non-differentiable or even non-continuous functions. Comparing to the traditional BP algorithm, the Rprop algorithm offers faster convergence and is usually more capable of escaping from local minima [8].

However, the efficiency and efficacy of both BP and Rprop algorithm depend significantly on initial weights which are initialized to small random values in the same interval. The Genetic algorithm (GA), one of the evolutionary algorithms, is a heuristic stochastic search algorithm [9]. Due to the good global searching ability and can learn the near-optimum solution without the gradient information of error functions, More attention has been attracted on combination of evolutionary algorithms and artificial neural networks. In recent years, GA has been applied to complex problems and a lot of valuable conclusions have been obtained [10]-[12].

In this paper, we combine GA and Rprop, with more “global” information like the magnitude of the network batch error, in order to improve the learning speed. That is, to optimize the connection weights of the neural networks with GA, so as to overcome the disadvantage of getting stuck in local minimum easily. First we describe theory basis of Neural Network. Then GA and Rprop algorithm is introduced, respectively. Furthermore, the hybrid algorithm GARP is proposed to perform analysis for energy efficiency problem. Finally, our results are discussed and conclusions are drawn.

2 Artificial neural networks

Artificial Neural Networks (ANNs) was first described by Warren McCulloch and Walter Pitts [13], is a mathematical model that consists of a set of interconnected processing elements, called neurons or nodes. ANNs were inspired by biological nervous systems, which propagate information through many nerve cells connected to each other with nerve fibres. Analogously to natural nervous systems, neurons in ANNs are connected to each other into a processing network, where each neuron receives, modifies and transfers signals. Fig. 1 shows a typical model of an artificial neuron. The links between neurons are called connections. Each connection has a corresponding value, called a connection weight, which is used to modify an incoming signal.

The outgoing signal of each neuron is calculated by means of the activation function (also referred to as the transfer function) expressed by the following equation

$$y_i = f_i(\sum_{j=1}^{n} x_j w_{ij} - \theta_i) \qquad (1)$$

where y_i is the output of a neuron i, x_j is the jth input signal to a neuron i, wij is the connection weight between neurons i and j, θ_i is an internal threshold value of a neuron i, i.e., value, which weighted sum of incoming signals needs to achieve to activate a neuron. Mathematically, the threshold value shows the position of the highest increment of the monotony increasing activation function.

The function is usually chosen such that it adds nonlinearity in the work of a network, but does not change the result significantly. Most popular are S-shape (sigmoid) functions, such as logistic, hyperbolic tangent, Heaviside, etc. However, some linear functions, e.g. piecewise linear function, are also applicable to ANNs with non-gradient learning methods.

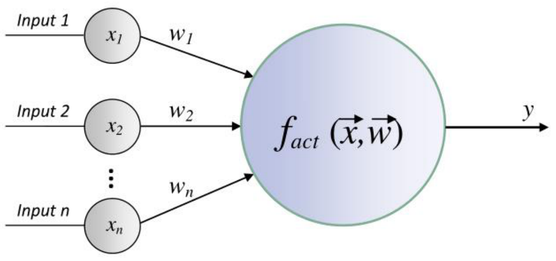

Figure 1. A model of an artificial neuron.

As a first step, the architecture of the network has to be decided. In this work we will concentrate one simple model: the feedforward networks. The feedforward model comprises networks, in which the connections are strictly feedforward, i.e., no neuron receives input from a neuron to which the former sends its output, even indirectly. To be precise, we will consider the so-called multilayer perceptron (MLP), in which units are structured into ordered layers, and connections are allowed only between adjacent layers in an input-to-output sense. Neurons in an ANN are located in layers, which are classified into input layer, hidden layer and output layer. In each neuron, we assign an activation function which will be triggered by weighted input signal. Neurons in one layer are connected, fully or partially, to the Neurons in the next layer. Feedback connections to previous layers are also possible. Fig. 2 shows a typical three layers MLP in which neurons are structured into ordered layers, and weighted connections are allowed only between adjacent layers units or neurons. The network contains two inputs, three hidden and one output neurons.

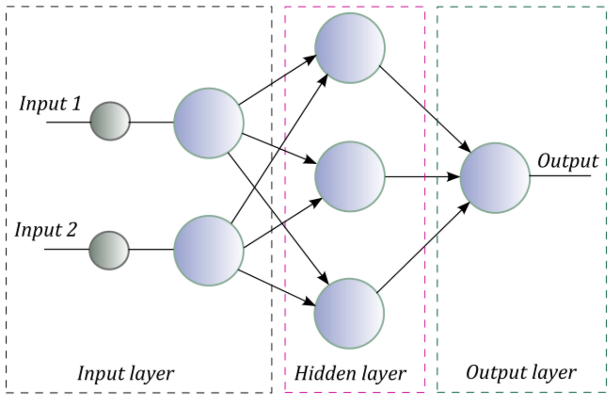

Figure 2. ANN with two input neurons, one hidden layer with three neurons in it, and one output neuron.

For any MLP, after defining the number of layers and the number of units per layer, the last step in the design is to adjust the weights of the network, so that it produces the desired output when the corresponding input is presented. This process is known as training the ANNs or learning the network weights. The network training is iterative processes which adjust weights and threshold, to reduce the error of network at a minimum or stop at a preset training step. As initially stated, we will focus on the learning situation known as supervised training, in which a set of input/desired-output patterns is available. Thus, the ANN has to be trained to produce the desired output according to these examples. The input and output of the network are both real vectors in our case. Then give the input of samples and obtain the output results. The essence of a learning algorithm is the learning rule, which determines how connection weights are changed.

3 The hybrid algorithm

3.1 Genetic algorithm

Genetic algorithms (GAs), developed by John Holland, represent the most popular type of evolutionary algorithms, a search technique used in computing to find exact or approximate solutions to optimization and search problems. GA was inspired by the adaptation and learning capabilities of nature species. Compared to other methods, GA is capable of exploring discontinued spaces of solutions with a minimum background knowledge and domain-based information. GA is directed random search techniques used to look for parameters that provide a good solution to a problem. Essentially they are nothing more than educated guessing. The "education" comes from knowing the suitability of previous candidate solutions and the "guessing" comes from combining the fitter attempts in order to evolve an improved solution.

GAs apply all genetic operators follow the classical order: first, they will select individuals for reproduction, then produce new individuals by applying crossover and finally, modify offspring by means of the mutation operator. After these steps, offspring individuals are included in the new population.

3.2 The resilient propagation algorithm

The Resilient Propagation algorithm (Rprop) is an improved adaptation of the batch backprop algorithm, and for numerous problems it converges very quickly. It uses only sign of the back-propagated gradient to change the bias/weights of the network, instead of the magnitude of the gradient itself. Therefore, the adaptation of the weight-step is not "blurred" by gradient behavior. Instead, each weight has an individual evolving update-value and the size of the weight change is determined by a separate update value. The purpose of the Rprop training algorithm is to eliminate these harmful effects of the magnitudes of the partial derivatives. Only the sign of the derivative can determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. The size of the weight change is determined by a separate update value. Rprop updates weight as follows.

$$\Delta_{ij}(t)=\begin{cases}\eta^{+}\bullet\Delta_{ij}(t-1), & if\ \frac{\partial E}{\partial w_{ij}}(t-1).\frac{\partial E}{\partial w_{ij}}(t)>0\\ \eta^{-}\bullet\Delta_{ij}(t-1), & if\ \frac{\partial E}{\partial w_{ij}}(t-1).\frac{\partial E}{\partial w_{ij}}(t)<0\\ \Delta_{ij}(t-1), & otherwise\end{cases} \tag{2}$$

Where $0<\eta^{-}<1<\eta^{+}$, and η^{+} is empirically set to 1.2 and to 0.5, respectively.

Rprop updates weight can be simply described as follows: Whenever the partial derivative of the error function E with respect to the corresponding weight wij changes its sign, it indicates that the value of last update was too big and the algorithm has jumped over a local minimum. The update-value is decreased by a factor. If the derivate retains its sign, the update-value is slightly increased by the factor in order to accelerate convergence in shallow regions.

Once the update-value for each weight is adapted, the weight-update itself follows a very simple rule: if the derivative is positive (increasing error), the weight is decreased by its update-value, if the derivate is negative, the update-value is added to the weight.

$$\Delta w_{ij}(t)=\begin{cases}-\Delta_{ij}(t), & if\ \frac{\partial E}{\partial w_{ij}}(t)>0\\ +\Delta_{ij}(t), & if\ \frac{\partial E}{\partial w_{ij}}(t)<0\\ 0, & otherwise\end{cases} \tag{3}$$

There is one exception to the rule above. If the partial derivative changes sign,

$$\Delta w_{ij}(t)=-\Delta w_{ij}(t-1),\ \text{if}\ \frac{\partial E}{\partial w_{ij}}(t-1)\cdot\frac{\partial E}{\partial w_{ij}}(t)<0 \tag{4}$$

In order to avoid a double punishment of the update value, there should be no adaption of the update value in the succeeding step. In general, this can be done by setting $\frac{\partial E}{\partial w_{ij}}(t-1)=0$ in the $\Delta_{ij}(t)$ update-rule above.

3.3 GARP mechanism

In this work, GA is used to find a good initial set of connection weights for ANN so that GA-aided Rprop (GARP) may find the global optimum even faster than Rprop only case. The evolution process starts with the generation of an initial population of individuals, where each individual represents a possible set of connection weights for a given ANN topology. After initialization, each individual is evaluated according to its worth regarding to the solving problem. Traditional GAs use binary stings to represent individuals and apply classical crossover and mutation operators. However, modern GAs tends to utilize real-valued representations, since it reduces the length of the genetic string, which is especially important for solving complex problems with many parameters. In this study, each individual (a set of weights for a given ANN) is encoded by concatenation of all connection weights of a network, whereby the connection weights to the same hidden/output node are put together.

The role of the fitness function in EANNs plays the error function, i.e., MSE or RMSE; the fitness function is employed to evaluate the quality of the solution at each epoch. In this work, fitness function is calculated by the average of the error. In the prediction problem, the error is based on the mean squared error (MSE) or root mean squared error (RMSE) in the following formula.

$$MSE=\frac{1}{N}\sum_{i=1}^{N}(\mathrm{y}_{\mathrm{i}}-\hat{\mathrm{y}}_{\mathrm{i}})^{2} \tag{5}$$

$$RMSE=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\mathrm{y}_{\mathrm{i}}-\hat{\mathrm{y}}_{\mathrm{i}})^{2}} \tag{6}$$

$$Fitness=\frac{1}{MSE} \tag{7}$$

Where N is the total number of instances, y_{i} $\hat{\mathrm{y}}_{\mathrm{i}}$ are the actual value and the model output respectively.

Following that, typical steps of the classical EA are performed (see Fig. 3): first, individuals are selected for reproduction based on their fitness; then new individuals are formed by means of genetic operators; and finally, new population is created. The evolution process continues until some termination criteria are reached. A typical cycle of the evolution of connection weights is as follows:

- Generate a population of size k, where each individual represents a set of n connection weights;
- Evaluate fitness of each individual in the population according to the error between actual and desired outputs over the training data;
- Select parents for reproduction based on their fitness;
- Create offspring by applying genetic operators to the parents;
- Form the next generation of k offspring individuals.

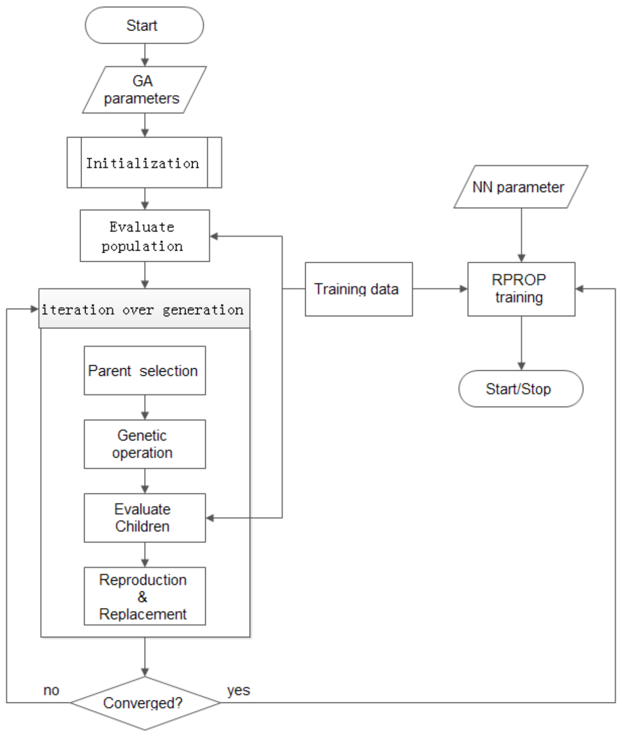

Figure 3. GARP mechanism.

4 Results and discussion

The energy efficiency data set taken from UCI database was used for comparison [14]. The dataset comprises 768 samples and 9 features, aiming to predict two real valued responses. The dataset contains eight attributes and two responses list in Table 1. The aim is to use the eight features to predict each of the two responses: heating load (Y_1) and cooling load (Y_2) [15]. The building parameters used are: relative compactness (X_1), surface area (X_2), wall area (X_3), roof area (X_4), overall height (X_5), orientation (X_6), glazing area (X_7), and glazing area distribution (X_8). We will try to predict heating load and cooling load.

Table 1. The parameters of enegry efficiency data set.

Mathematical	Representation Input or output variable
X_1	Relative compactness
X_2	Surface area
X_3	Wall area
X_4	Roof area
X_5	Overall height
X_6	Orientation
X_7	Glazing area
X_8	Glazing area distribution
Y_1	Heating load
Y_2	Cooling load

To evaluate an ANN, training used 80% of the total patterns and the rest were used for forecasting out of samples using the trained network. We used an 8-7-2 MLP architecture of input-hidden-output units yielding a population vector of 79 members (weights). The weight value range is set to [-10, 10]. The initial GA population is set to 50 and the weights and bias is generated randomly for each individual. The GA was executed for 10 generations on a population of 50 individuals. The activating function of the neurons is the sigmoid function.

In order to evaluate performance of GARP, two other algorithm BP and Rprop are involved. We trace the MSE of the training and testing pattern set after performing 50 independent runs. Table 2 shows the training error and testing error that obtained with GARP algorithm and compares them to the BP and Rprop algorithm. Comparing results are plotted shown in Fig. 4.

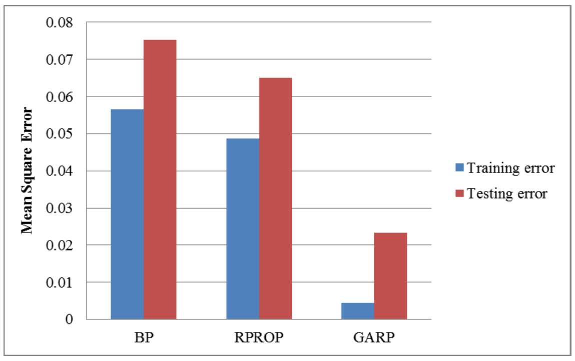

Figure 4. Comparing network performance using different algorithms.

As it has been shown, we can observe that GARP is the faster algorithm, what confirms intuition on the velocity of local search compared to BP and Rprop. The MSE of GARP algorithm is much smaller than that of BP and Rprop. The convergence analysis shows that it carries the characteristic and the advantages of them and without any side effect. In the simulation result, the global convergence capability is further improved compare with BP and the original Rprop, and the training error and testing error is 0.004349 and 0.023233 respectively. It also fixed the low convergence rate problem in Rprop with GA and some of them are even higher than Rprop with BP. Overall, it outperformed BP, Rprop in terms of convergence rate and global convergence capability.

Fig. 5 and Fig. 6 plot the heating load and cooling load from ANN and testing data, respectively. It is shown that good agreement is achieved between GARP prediction and target. The trained GARP model can be able to perform prediction task well. Specifically, starting the evolution process with more generations, Rprop with GA strategy produced more compact networks with smaller testing errors.

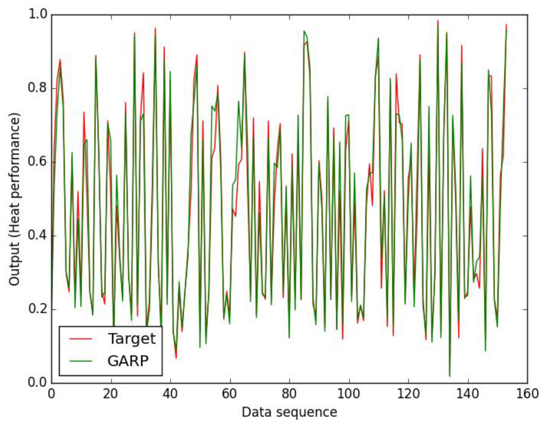

Figure 5. Output heating load from GARP and testing dataset.

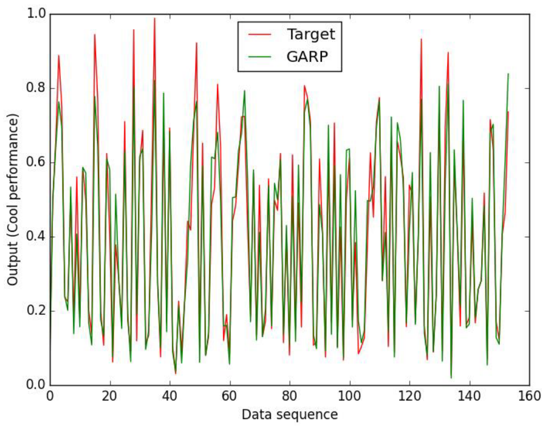

Figure 6. Output cooling load from GARP and testing dataset.

5 Conclusions

It is widely accepted that the Rprop algorithm is one of the best performing first order learning algorithms for multilayer neural networks. In this paper we introduced an efficient improvement of the Rprop algorithm that is based on genetic algorithm. Furthermore, in energy efficiency problem, the differences training error and testing error between Rprop with GA, BP and the original Rprop are statistics. Our results show that GARP algorithm gets lower MSE than the problem-specific algorithms (BP and Rprop). This will improve the convergence rate of the network and reduce the training failure, and the neural network's generalization ability is better than the algorithms that only use BP or Rprop. As a result, evolving neural networks can lead to enhanced performance and make GARP look as a promising algorithm for neural network training. As a future work we plan to add new algorithms to the analysis, and to apply them to more instances. In addition, Rprop, a batch training algorithm, can be parallelized easily to improve the computing efficiency. Therefore, future work will also focus on accelerating the training speed through parallel technique.

References

1. J.H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence., U Michigan Press (1975).
2. B. Temeyer, W.G. Jr, … Symposium, Iowa (2003).
3. H. Hasanzadehshooiili, A. Lakirouhani, A. Šapalas, Archives of Civil and Mechanical Engineering **12**, 477 (2012).
4. J. Fernández, A. Pernía, F.J. Martínez-de-Pisón, R. Lostado, Engineering Structures **32**, 3018 (2010).
5. Y. Hirose, K. Yamashita, S. Hijiya, Neural Networks **4**, 61 (1991).
6. B. Widrow, M.A. Lehr, Proceedings of the IEEE **78**, 1415 (1990).
7. M. Riedmiller, H. Braun, RPROP - A Fast Adaptive Learning Algorithm, Proc. Of ISCIS VII), Universitat (1992).
8. M. Riedmiller, H. Braun, in Neural Networks, 1993., IEEE International Conference On, IEEE **586** (1993).
9. L. Davis, others, Handbook of Genetic Algorithms, Van Nostrand Reinhold New York (1991).
10. X. Yao, Proceedings of the IEEE **87**, 1423 (1999).
11. G.-H. Kim, J.-E. Yoon, S.-H. An, H.-H. Cho, K.-I. Kang, Building and Environment **39**, 1333 (2004).
12. F. Ahmad, N.A.M. Isa, Z. Hussain, R. Boudville, M.K. Osman, in CICSyN **78** (2010).
13. J.-S.R. Jang, C.-T. Sun, Neuro-fuzzy and Soft Computing: a Computational Approach to Learning and Machine Intelligence, Prentice-Hall, Inc. (1996).
14. C. Blake, C.J. Merz, Department of Information and Computer Science **55** (1998).
15. A. Tsanas, A. Xifara, Energy and Buildings **49**, 560 (2012).