International Journal of Architectural, Civil and Construction Sciences
ISSN: 2415-1734
Vol:6, No:9, 2012

Self-evolving Neural Networks Based on PSO and JPSO Algorithms

Abdussamad Ismail, Dong-Sheng Jeng

***Abstract*—**A self-evolution algorithm for optimizing neural networks using a combination of PSO and JPSO is proposed. The algorithm optimizes both the network topology and parameters simultaneously with the aim of achieving desired accuracy with less complicated networks. The performance of the proposed approach is compared with conventional back-propagation networks using several synthetic functions, with better results in the case of the former. The proposed algorithm is also implemented on slope stability problem to estimate the critical factor of safety. Based on the results obtained, the proposed self evolving network produced a better estimate of critical safety factor in comparison to conventional BPN network.

***Keywords*—**Neural networks, Topology evolution, Particle swarm optimization.

I. Introduction

DEVELOPING neural networks involves not only optimizing synaptic weights but also choosing a suitable processing function as well as optimizing the network architecture. However, considering the combination of discrete and continuous parameters involved, it is such an extremely challenging task to optimize the network topology and the network parameters at the same time [1].

Classical topology optimization methods such as incremental learning algorithm and pruning technique are likely to lead to a convergence at a sub-optimal network configuration due to the manner in which the network size is increased in the case of incremental learning and the way in which the complexity of the network is reduced the case of the pruning [2]. To overcome the problem associated with aforementioned approaches, a variety of bio-inspired evolutionary concepts of have been employed to simultaneously optimize network topology and parameters. These include the genetic algorithm (GA) based algorithms such as EPNet [3] and NEAT [4]. In EPNet, a population of networks with randomly generated topology and synaptic weights are subjected to a series of mutation cycles. Each mutation cycle involves parametric mutation, where the networks' synaptic weights are updated and structural mutation in which the nodes or connections are added or removed.

The mutations are carried out repeatedly until a satisfactory network is obtained. The NEAT algorithm, on the other hand, seeks to avoid the inefficient cross over operation associated with EPNet by starting with a population of smallest possible networks, then gradually increasing their complexity as learning goes on. The downside of NEAT, however, is the intricate cross-over procedure involved while updating the network topology.

A.Ismail is a research student in the Division of Civil Engineering, University of Dundee, U.K. e-mail: asiiiuk@ gmail.com).

D-S. Jeng is a Professor in the Division of Civil Engineering, University of Dundee, U.K. (e-mail: dsj@dundee.ac.uk).

Particle swarm optimization (PSO), another type of bio-inspired technique, has also been successfully used in evolving neural networks [5,6,7]. Although PSO is computationally simpler than GA-based algorithms, the key disadvantage of the self evolution algorithms developed based PSO is the random generation of network topology, which tends to compromise the computational efficiency of the optimization process.

This paper presents an approach to network topology evolution in which Jumping particle swarm optimization technique (JPSO) is used to optimize the topology and activation function while using a combination of back-propagation and PSO techniques to optimize the network continuous parameters. Another important feature of the proposed technique is that, like NEAT, the complexity of the network is gradually increased, beginning with simple architecture. A number of synthetic functions are used to compare the performance of the proposed optimization technique with conventional BPN with various activation functions. A slope stability problem is also used to further assess the prediction quality and the complexity of the network developed using the proposed algorithm.

II. Jumping Particle Swarm Optimization (JPSO)

JPSO algorithm is a combinatorial optimization algorithm proposed by developed by Martınez-Garcıa and Moreno-Pe´rez [8] that bears some resemblance with the discrete version of particle optimization (DPSO) developed by Kennedy and Eberhart [9] with regards to the gravitation of particles towards better positions, but differs from DPSO in that the change in particle position in the case of the former is not based on the concept of particle velocity. A particle in JPSO updates its position by jumping from its current position to a new position under the influence of particle's experience, global best position as well as its `explorative tendency. The possible trajectories of particle jumping are shown graphically in Figure 1. The particle's position is updated using the following equation:

$$\boldsymbol{x}_{t+1} = \lambda_1 \otimes \boldsymbol{x}_t \oplus \lambda_2 \quad \boldsymbol{b} \quad \otimes_3 \quad \boldsymbol{g} \oplus \qquad \otimes\lambda \qquad (1)$$

where $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$ are the vectors of current and future particle positions in the discrete search space. The parameters λ_1, λ_2 and λ_3 are probabilities of jumping randomly, towards the best particle position and to the best swarm position respectively. $\boldsymbol{b}$ and $\boldsymbol{g}$ are, respectively, the particle best and global best positions. The particle position updating is carried out as follows:

$$x_{i,t+1} = \begin{cases} x_{i,t} * \rho & P_{x_{i,t} \to \rho} = \lambda_1 \\ x_{i,t} * b_i & P_{x_{i,t} \to b_i} = \lambda_2 \\ x_{i,t} * g_i & P_{x_{i,t} \to g_i} = \lambda_3 \end{cases} \qquad (2)$$

$$\lambda_1 + \lambda_2 + \quad_3 = 1\lambda$$

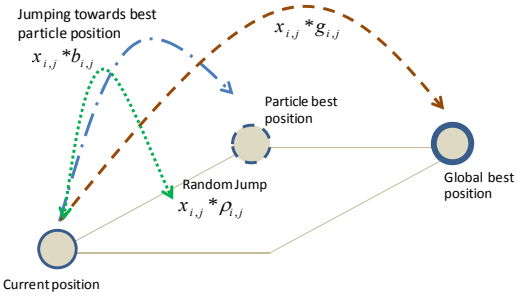

Fig. 1 Graphical representation of jumping particle in topology space

in which ρ is a random binary number. The * operator is implemented by stochastically modifying the features of the current particle with some features of its attractor. The updated position determined using equation (2) could be worse than the current one, therefore a random local search is carried out around the updated position to find a better solution. Due to the mixed nature of optimization problem in this work, the local search is carried out using few steps of back-propagation algorithm. Also, the parameter corresponding to random jump, λ_1 is reduced to zero but compensated by resetting the positions of a portion of particle swarm at certain intervals. Thus the values of λ_2 and λ_3 sum up to 1 in the present work. The proposed JPSO algorithm is represented by the flowchart in Figure 2.

III. Self-Evolving Network

The proposed self evolution process begins by generating a population of neural nets, each having a random synaptic connections and synaptic parameters. The connection parameters are binary, assuming a value of 1 if there is a connection between two nodes and 0 if otherwise (Figure 3). They are updated using a jumping particle swarm optimization (JPSO) procedure described briefly in section 2. The synaptic weights of individual networks in the population are updated using a combination of PSO and BP algorithm. The advantage of putting together the two techniques is to take the advantage of global search capability of the former and the ability of the later to perform local search. The algorithm involves updating the synaptic weights using PSO for a number of iterations, and then further updating the weights of best performing particles in the swarm BP algorithm for few steps. This alternative use of PSO and BP is repeated until a sufficiently accurate result is obtained. In order to guard against the tendency of particles convergence at suboptimal co-ordinate, duplicate particles have their positions reset randomly at the end of each cycle of PSO iterations. The positions of least performing particles in the swarm population are also randomly reset in order to improve the topology search capability, having removed the random jumping aspect of JPSO.

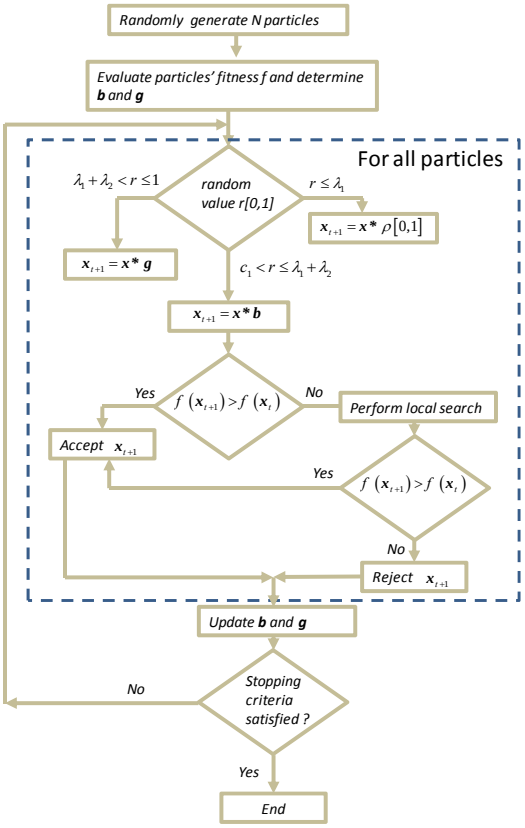

Fig. 2 Flowchart describing JPSO algorithm

When the accuracy doesn't improve with further training and there is a desire to further improve it, the complexity of the network is up scaled by adding more nodes, one node at a time. At this point, the so far acquired information is preserved by retaining the current particles best positions (both topology and synaptic weights) while the topology of the swarm members is modified. The rationale here is to facilitate the development of simpler networks and to achieve the desired accuracy while minimizing the computational burden of having to deal with unnecessary large network size as the case is with some models available in the literature [15, 16, and 17]. The algorithm of self-evolving network is summarised in the following steps:

1. Initialize a particle swarm population of N size, with each particle representing neural networks with a single hidden node and randomly generated set of synaptic weights and connection parameters.
2. Evaluate the fitness of each particle and update the best particle and global positions.
3. Use PSO/JPSO to update particle co-ordinates for certain number of iterations in the following sub-steps:
   a. Use PSO to update the weight vector of each particle
   b. Use JPSO to update the connection parameters of each particle.
   c. Update the particle best position and the best swarm position
4. If convergence is sufficient then go to 9. Else continue

5. Reset randomly the binary and continuous parameters of duplicate particles. Also, reset in the same manner, the binary parameters of certain fraction of the swarm with poor fitness.
6. Select best particles and update their continuous parameters using some steps of BP. If the training is satisfactory go to 9. Else continue.
7. If number of iterations is less than maximum number then go back to step3. Else continue
8. Generate N particles with one additional node over the current number of nodes. Replace all current particles with newly generated particles while retaining the current particle best positions (topology and synaptic weight). Then go back to step 3.
9. Terminate algorithm and return result.

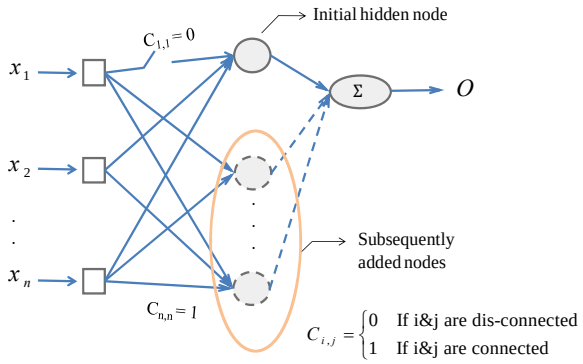

Fig. 3 Topology of self-evolving network

To assist the swarm of partially connected networks in the search for best network, a parallel swarm of fully connected networks but with the same number of nodes is simultaneously optimized., with the former learning from the later whenever the best swarm position in later is more accurate.

A. Activation function

The choice of suitable activation function is crucial to a successful development of neural networks due to its strong influence on the complexity and accuracy of the later. Sticking to popular processing functions like sigmoid function as par the usual practice doesn't guarantee optimality under all circumstances. [10,11]. in light of this argument, a combination of several functions is proposed as activation function. The proposed activation function is expressed as follows:

$$f(\boldsymbol{x}) = \sum_{1}^{n} k_i \alpha_i \varphi_i(\boldsymbol{x}) \tag{3}$$

where n is the number of sub-functions φ_i in the activation function. α_i is an adaptive coefficient; k_i is a binary number. $\boldsymbol{x}$ is the vector of inputs to the node. The function of the binary number is to let the associated sub-function be part of the activation function if by assuming a value of 1 and excluding the associated sub-function by assuming a value of zero.

The coefficient α_i throws some weight behind the sub-functions making up the activation function in accordance with their relative importance to the output of the neuron. The α_i coefficient is adjusted in the same manner as the synaptic weights are, while the binary parameter, k_i, is optimized alongside the connectivity parameter c_i. The sub-functions considered in this work are represented by the following equations:

Linear: $\varphi_1(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$ (4a)

Sinusoid: $\varphi_2(\boldsymbol{x}) = \sin(\boldsymbol{w}^T\boldsymbol{x} + b)$ (4b)

Sigmoid: $\varphi_3(\boldsymbol{x}) = \dfrac{1}{1+e^{\boldsymbol{w}^T\boldsymbol{x}+b}}$ (4c)

Wavelet: $\varphi_4(\boldsymbol{x}) = (\boldsymbol{w}^T\boldsymbol{x} + a)e^{\frac{(\boldsymbol{w}^T\boldsymbol{x}+b)^2}{2}}$ (4d)

$\boldsymbol{w}$ represents the vector of synaptic weights of input signals, whereas a and b are the biases.

B. Error function

Mean square error (MSE), the most widely used error function in network training, is often criticized for its tendency to lead the network to over fitting. One of the methods of enhancing is the weight decay, where regularization term, a function of synaptic weights, is added to MSE error function. The weak point of the weight decay method is that it only reduces the values of synaptic weights to small values without reducing the network size [12]. A different approach was proposed by Jin *et al* [13], where the regularization term is based on the number of connections with the aim of effectively removing the redundant weights from the network. The error term used in this paper is based on the Jin *et al*'s approach due to its ability to deal more efficiently with the issue of network complexity.

IV. Experimental Results

To demonstrate how a neural network evolves using the proposed algorithm, function approximation and system identification problems are considered. The nature of the problems is described and the results of network model simulations are discussed in the following sub-sections.

A. Wavelet function

A wavelet function of three variables, represented by equation 10, is used in this case to generate 225 datasets with input variables x_1, x_2 and x_3 randomly generated and varied from -10 to +10. 150 sets were used to develop networks, while 75 sets of data were used for validation.

$$f_3 = \frac{10\sin(2x_1 + 2x_2 - x_3)}{\pi(2x_1 + 2x_2 - x_3)} \tag{5}$$

From the results summary in Table I, it can be seen that the optimized network based on the proposed algorithm returns

the best result. It can also be observed that despite having the smallest number of network parameters, it turns out to be the most accurate. Relatively less accurate results are obtained in the case of fully connected network with all functions switched on. The performance of the networks based on sinusoid and sigmoid activation is the lowest in the table. It can also be noted that their performance has not improved with increased number of neurons.

TABLE I
SUMMARY OF TRAINING AND TESTING RESULTS (WAVELET FUNCTION)

Type of Network	No of nodes	Number of network parameters	N-RMSE (training)	N-RMSE (testing)
Optimum Network	1	14	0.00109	0.00112
Wave	14	84	0.013	0.03324
Sinusoid	12	60	0.08933	0.17657
	14	70	0.09038	0.22144
Sigmoid	5	25	0.07284	0.08725
	14	70	0.11123	0.14088
All functions	3	52	0.08933	0.06149

B. Narendra-Li system

A non-linear system identification problem described by Narendra and Li [14] is used here to assess the relative accuracy of the proposed method. The non-linear system is represented by the following set of equations:

$$y(t) = \frac{{}_1x_1(t)}{1+0.5\sin x_2(t)} + \frac{x_2(t)}{1+0.5\sin x_1(t)} + e(t)$$
$$x_1(t+1) = \left(\frac{x_1(t)}{1+x_2^2(t)}+1\right)\sin x_2(t)$$
$$x_2(t+1) = x_2(t)\cos x_2(t) + x_1(t)e^{-\frac{x_1^2(t)+x_2^2(t)}{8}} + \frac{u^3(t)}{1+u^2(t)+0.5\cos[x_1(t)+x_2(t)]} \quad (6)$$

where *u(k)* and *y(k)* are, respectively, the input and output signals at time *t*. six inputs consisting of three past network inputs and three past network outputs are used in order to provide the network with enough memory to identify the system. A total of 800 data sets are generated for training, with *u(k)* as defined by the following equation:

$$u(t) = \sin\frac{2\pi t}{10} + \sin\frac{2\pi t}{25},\ t = 1, 2.....n \quad (7)$$

A uniformly distributed noise of ranging from -1 to +1 is added to the output in order to test the robustness of the model.

A testing data set consists of 400 data points generated in the same manner as the training set, but with no noise added. From the training and testing results in Table II, It can be seen that despite the inferior accuracy of proposed model compared to the performance of the wavelet network, its topology is much simpler. The trade off by the self-evolving net between accuracy and simplicity seems a reasonable one.

TABLE II
SUMMARY OF TRAINING AND TESTING RESULTS (NARENDRA-LI SYSTEM)

Type of Network	No of nodes	Number of network parameters	N-RMSE (training)	N-RMSE (testing)
Self - evolving Net	1	24	0.08462	0.03383
Wave	6	54	0.07563	0.02645
Sinusoid	6	48	0.08119	0.03319
Sigmoid	7	56	0.08001	0.03133
All functions	4	107	0.07692	0.03322

C. Slope stability problem

Stability of slopes is one of the key geotechnical engineering design problems that has been extensively studied for decades. The slope stability is commonly assessed by evaluating the factor of safety against failure. The factor of safety of a slope is defined as the ratio between the forces that resist and forces that overcome the resistance to slope failure along a possible slip surface. The widely used method of estimating the safety factor is the limit equilibrium method of slices, in which the sliding tendency of a mass of soil that exists within the envelope bounded by the slope and a potential failure surface investigated. A typical section through is shown in figure 4. Some of the versions of limit equilibrium method include Bishop's simplified method [15], Janbu's method [16] and Morgenstern and Price[17]. One of the challenges associated with this method is how to locate the slip surface corresponding to the worst factor of safety. The conventional practice is based on trial and error, where analysis is repeated for number of trial slip surfaces, out of which the surface giving rise to minimum safety factor is selected. In the present work, a self –evolving neural network is used to directly estimate the critical factor of safety based on inputs such as the geometrical properties of the slope, soil shear strength parameters as well as the effects of seepage. The idea is to make slope stability check simpler, faster but accurate.

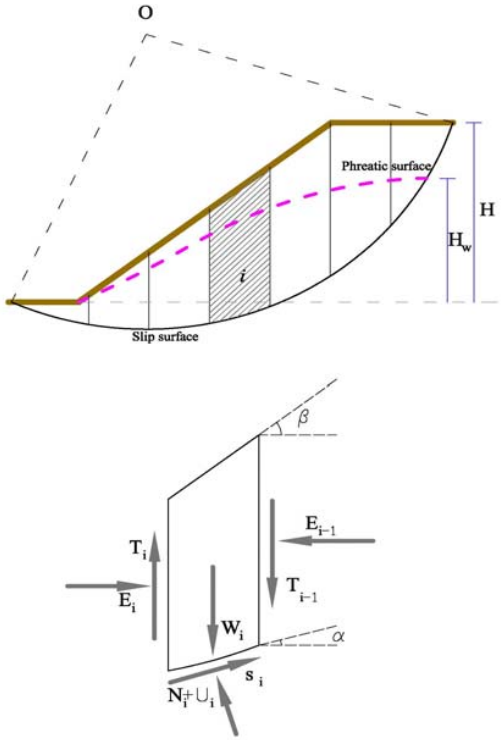

Fig. 4 Section through an earth slope (method of slices)

1. Input parameters

The stability of a slope in a homogeneous soil depends on the slope geometry (height and angle of slope), the soil properties (cohesion, angle of internal friction and unit weight) and. The factor of safety (F.S) as a function of various controlling variables mentioned above can be stated as follows:

$$F.S = f\left(\tan\phi, \frac{c}{\gamma}, H, \cot\beta, \frac{H_w}{H}\right) \quad (8)$$

where ϕ, c and γ are the friction angle, cohesion and unite weight of the slope material respectively. H is the slope height; β the slope angle; and H_w the height of water behind the slope. The parameters on the right hand side of equation 8 serve as inputs to the network, F.S being the output.

2. Database

The database used in the present work consists of 455 sets of data obtained from the results of slope stability analysis carried out on homogeneous slopes using Slope/W slope stability analysis software. A wide range of soil properties as well as slope geometrical parameters is represented in the database. Table III provides the summary of the database characteristics.

TABLE III
STATISTICAL PROPERTIES OF SLOPE DATABASE

Parameter	x_{max}	x_{min}	μ	σ
φ	60	0	28.2857	24.1377
C	200	0	53.011	63.8241
H	40	5	12.3517	8.3143
$\cot\beta$	2.4	1	1.1613	0.3017
H_w	25	0	2.2813	4.5598
F.S	15.595	0.102	3.0923	3.2773

3. Networks Training and validation

The database was split into training and testing sets. A total of 303 data sets were used for training, while the remaining 152 sets were earmarked for testing. To facilitate generalization, the data is divided in such a way that both the training and testing data statistically belong to the same population. For the sake of comparison, the conventional BPN networks were also trained, alongside the proposed self-evolving network.

Results of the optimized network predictions are plotted against the training and testing data in the scattergrams shown in Figures 5(a) and 5(b) respectively. It can be seen from the figures that the network gives a good correlation with both training and testing data (R^2 =0.9935 for training and R^2= 0.9918 for testing). The performance of the optimized network is compared BPN networks with various types of activation functions in Table IV. It is evident from the results that the proposed model outperforms the other networks not only because it returns minimum error in both training and testing, but also due to its relatively few parameters.

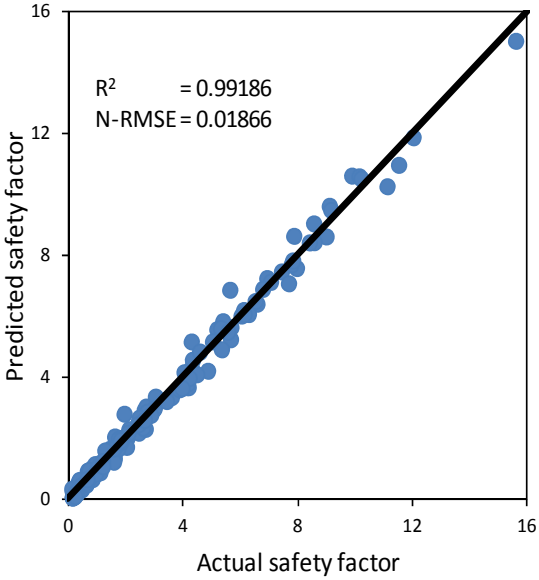

Fig. 5(a) Comparison of Actual safety factor versus self-evolving model predictions (training data)

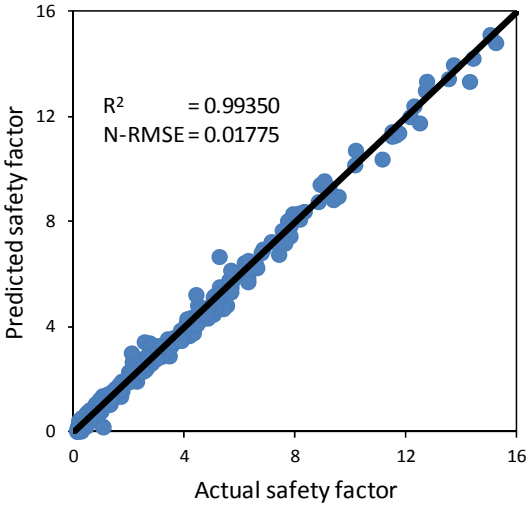

Fig. 5(b) Comparison of of Actual safety factor versus self-evolving model predictions (testing data)

TABLE IV
SUMMARY OF TRAINING AND TESTING RESULTS FOR VARIOUS TYPES OF NETWORK (SLOPE STABILITY PROBLEM)

Type of Network	No of nodes	Number of network parameters	N-RMSE (training)	N-RMSE (testing)
Self-evolving Network	2	26	0.01775	0.01866
Wave	5	40	0.03315	0.10842
Sinusoid	7	49	0.07921	0.97808
Sigmoid	7	49	0.04359	0.11247
All functions	3	72	0.01432	0.03694

Further comparison was also made between self evolving network based on equation (4) and a simpler activation function represented by the following equation:

$$f(\boldsymbol{x}) = k_1\alpha_2\left(\boldsymbol{w}_1^T\boldsymbol{x}\right) + k_1\alpha_2\prod_1^n x_i^{w_i} \quad (9)$$

where n is the number of inputs. Other parameters are defined in section 2. The training and testing results of the two networks are put together in Table V. It can be noticed from the table that although the more complicated network is more accurate, the accuracy of the network based on a combination of linear and product unit processing functions is reasonable. The ability of the latter to achieve such a precision with much simpler topology is particularly impressive.

TABLE V
PERFORMANCE COMPARISON OF TWO SELF-EVOLVING MODELS (SLOPE STABILITY PROBLEM)

Activation functuion	No of nodes	Number of network parameters	N-RMSE (training)	N-RMSE (testing)
Equation (4)	2	26	0.01775	0.01866
Equation (8)	2	9	0.02349	0.0257

V. CONCLUSION

The task of simultaneous optimization of the topology and synaptic weights of neural networks is desirable but highly challenging. This paper proposes an algorithm to optimize both the architecture and synaptic weights of a neural network at the same time. The key features, which make the proposed algorithm distinct from other methods available in the literature, are the ability to grow from a very small network to a complex without a loss of information while maintaining the capability of exploring the search space. A data generated from a wavelet function and Narendra-Li system and slope stability analysis were used to compare the prediction capability of networks based on the proposed self –evolution algorithm and conventional BPN networks. The results showed that self evolution algorithm produces much smaller network sizes without compromising accuracy. With regards to slope stability problem, the predictions of self evolving network based on a combination of linear and product unit processing functions gives are remarkably accurate, considering the few parameters associated with the network.

ACKNOWLEDGMENT

The authors are highly appreciative of financial support by Petroleum Technology Development Fund, Federal Republic of Nigeria.

REFERENCES

[1] Miller, G. F., Todd, P. M. and Hegde, S. U. (1989). “Designing neural networks using genetic algorithms.” Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, pp. 379–384.
[2] Angeline, P. J., Saunders, G. M. and Pollack, J. B. (1994). “ An evolutionary algorithm that constructs recurrent neural networks, IEEE Transactions on Neural Networks, 5, 54/65
[3] Yao, X. and Liu, Y. (1997). “A New Evolutionary System for Evolving Artificial Neural Networks.” IEEE Transactions on Neural Networks, 8-3:694-713
[4] Stanley, K. and Miikkulainen, R. (2002). “Evolving Neural Networks through Augmenting Topologies.” Evolutionary Computation, 10(2): 99-127
[5] Xian-Lun T., Yon-Guo L. and Ling Z. (2007). “A hybrid particle swarm algorithm for the structure and parameter optimization of feedforward neural networks. “ LNCS 4493:213-218.
[6] Yu, J., Wang, S. and Xi, L. (2008). “Evolving artificial neural networks using an improved PSO and DPSO.” Neurocomputing, 71:1054–1060
[7] Kiranyaz, S., Ince, T., Yildirim, A. and Gabbouj, M. (2009).”Evolutionary artificial neural networks by multi-dimensional particle swarm optimization.” Neural Networks (2009) in press.
[8] Mat`ınez Garc`ıa ,F. J. and Moreno P´erez J. A. (2008) “Jumping Frogs Optimization: a New Swarm Method for Discrete Optimization.” , Technical Report DEIOC 3/2008, Department of Statistics, O.R. and Computing, University of La Laguna, Tenerife, Spain
[9] Kennedy, J. and Eberhart, R. C. (1997). “A Discrete Binary Version of the Particle Swarm Algorithm.” Proceedings of IEEE Conference on Systems, Man, and Cybernetics, iscataway, New Jersey,USA. 4104–4109.
[10] Sopena, J.M., Romero, E. and Alquezar, R. (1999); “Neural networks with periodic and monotonic activation functions: a comparative study in classification problems.” Ninth International Conference on Artificial Neural Networks (ICANN ‘99). 1:323 - 328.

[11] Wong, K.-W., Leung, C.S. and Chang, S.-J. (2002). "Use of periodic and monotonic activation functions in multilayer feedforward neural networks trained by extended Kalman filter algorithm." IEEE Proceedings. Image Signal Processing, 149 (4), 217 – 224.
[12] Jin, Y., Okabe, T. and Sendhoff, B. (2004). "Neural network regularization and ensembling using multi-objective evolutionary algorithms." Congress on Evolutionary Computation (CEC'04), IEEE
[13] Reed , R.D. and Marks, R.J. (1999). *Neural Smithing*. The MIT Press
[14] Narendra, K. S. and Li, S.-M.(1996). Neural networks in control systems. In P. Smolensky,M. C. Mozer, and D. E. Rumelhart, editors, Mathematical Perspectives on Neural Networks, chapter 11, pages 347–394. Lawrence Erlbaum Associates
[15] Bishop, A .W. (1955). "The use of the slip circle in the stability analysis of slopes." Geotechnique, 5: 7-17.
[16] Janbu, N.(1973). Slope Stability Computations. Embankment Dam Engineering - Casagrande Volume, R.C. Hirschfeld and S.J. Poulos, eds., John Wiley and Sons, New York, pp 47-86.
[17] Morgensternn.,R ., and Price,V .E. (1967). "A numerical method for solving the equations of stability of general slip surfaces." Computer Journal, 9: 388-393.