Evolutionary Programming Versus Artificial Immune System in Evolving Neural Network for Grid-connected Photovoltaic System Output Prediction

[1]SHAHRIL IRWAN SULAIMAN, [2]TITIK KHAWA ABDUL RAHMAN, [3]ISMAIL MUSIRIN AND [4]SULAIMAN SHAARI

[1,2,3]Faculty of Electrical Engineering
[4]Faculty of Applied Sciences
Universiti Teknologi MARA Malaysia
40450 Shah Alam, Selangor
MALAYSIA
[1]shahril_irwan2004@yahoo.com, [2]khawa@salam.uitm.edu.my, [3]i_musirin@yahoo.co.uk,
[4]solarman_s@yahoo.com

Abstract: - This paper presents the evolutionary neural networks for the prediction of energy output from a grid-connected photovoltaic (GCPV) system. Two evolutionary neural network (ENN) models have been proposed using evolutionary programming and artificial immune system (AIS) respectively. The artificial neural network (ANN) employed for these models utilized solar radiation and ambient temperature as its input whereas the kilowatt-hour energy of the GCPV system is the only targeted output. The evolution of ANN involves the search of the optimal number of nodes, the learning rate, the momentum rate, the transfer function and the learning algorithm of a single-hidden layer multi-layer feedforward ANN. The results showed that evolutionary programming-ANN (EPANN) outperformed artificial immune system-ANN (AISANN) in terms of correlation coefficient, *R* as well as computation time. In addition, EPANN had also produced better convergence of the evolving parameters compared to the AISANN.

Key-Words: - artificial neural network (ANN), multi-layer feedforward neural network (MLFNN), photovoltaic (PV), grid-connected photovoltaic system (GCPV), correlation coefficient (*R*), evolutionary programming (EP), artificial immune system (AIS) and prediction.

1 Introduction

Photovoltaic (PV) involves solar electricity converted from solar energy. As any site on earth theoretically receives significant amount of solar energy, solar power generation offers significant advantages compared to other sources of renewable energy which are mostly site dependent. Although solar power generation is traditionally related to stand-alone PV systems, the current application of solar power generation is often associated to the grid-connected PV (GCPV) systems, particularly in urban areas where the conventional utility grid is readily available. In a GCPV system, the PV modules are connected in a specific configuration to form a PV array. The power generated by PV array is then channeled to an inverter which converts the DC power to AC power.

Nevertheless, the operation of GCPV systems strongly depends on various climatic factors such as solar irradiation and ambient temperature [1]. Higher irradiation would result in higher energy output produced by a GCPV system. In contrast, higher ambient temperature would heat up the solar cells inside the modules, thus lowering the effective voltage of the solar cells [2]. As a result, the overall energy output from the system is similarly reduced. In addition, the variation of these climatic factors with respect to time and location has resulted in inconsistent performance of GCPV systems.

Due to the inconsistent output performance, it would be very beneficial if the energy output from the system could be predicted when the expected irradiation and temperature are known. Thus, the system owner could pre-estimate the electricity bill savings or extra income generated by exporting the solar electricity to the utility. One of the available prediction techniques is the artificial neural network (ANN). This technique has been used in predicting PV system output. Firstly, a two-hidden layer multi-layer feedforward was developed to predict the energy (in kWh) output of grid-PV systems using different combination of solar radiation, module temperature and clearness index [3]. Likewise, the output performance of a PV module had specifically modeled using the same architecture but with a two-layer configuration and different inputs [4]. The ANN utilizes solar irradiance, ambient temperature and module temperature as its inputs while voltage and current are identified at its outputs. These studies had proven that

ANN is highly capable of predicting the output of PV systems.

Nevertheless, the selection of ANN design parameters is often difficult and tedious as it is commonly performed using trial and error process. As a result, several methods using evolutionary neural network (ENN) had been proposed to facilitate the design of ANN. Firstly, evolutionary programming (EP) was used to evolve the weights of an ANN [5]. Nevertheless, the proposed algorithm did not focus on the evolution of ANN architecture as the number of nodes was presumed before the training process. As a result, the ANN design may not be optimized. Secondly, an EP based ANN in [6] was developed to evolve both the architecture and weights of an ANN. but the proposed technique had only been demonstrated with a simple XOR problem. Nonetheless, a more realistic approach was proposed in [7]. Although the output of the GCPV system was predicted successfully by evolving the number of nodes in hidden layer, the learning rate and the momentum rate of the ANN, the selection of transfer function and learning algorithm was performed intuitively. Thus, the evolution of the ANN was not comprehensive. Therefore, this paper presents two better approaches towards ANN design using evolutionary programming (EP) and artificial immune system (AIS) as the optimizer in the selection of ANN design parameters.

2 Proposed Artificial Neural Network Model

ANN is a generalization process for mathematical models based on biological nervous system [8]. The fundamental processing element of an ANN is called a neuron. In basic computational model, the neuron collect input signals from other neurons or sources and merge them. It will then perform necessary computation before mapping them to an output. ANN has been preferred over other conventional statistical techniques to solve many prediction problems since it is able to handle complex interconnection problems more effectively compared to its competitors via the large network of processing nodes [9]. Moreover, ANN does not require prior knowledge on the characteristics of data in contrast with the conventional statistical methods which process information based on the statistical values derived from the data distribution presented to a problem.

Although there are different types of ANN architecture used for prediction, the Multi-layer Feedforward Neural Network (MLFFNN) has been widely used in solving many engineering problems due to its good generalization capability and simplicity [10]. Apart from that, MLFNN is also capable of performing non-linear modeling with reasonable accuracy [11]. Hence, MLFFNN was employed in this study. The MLFNN comprises two nodes at the input layer representing two inputs and one node at the output layer representing a single output. In addition, single hidden layer with sigmoid function was chosen as it often produces satisfactory prediction accuracy as long as the selected number of nodes is adequate [12-13]. Besides that, it is also a good estimator for any nonlinear function [14].

An example of a general MLFFNN architecture with single hidden layer is shown in Fig. 1. The MLFNN consists of one input layer, one hidden layer and an output layer. The input layer, hidden layer and output layer consist of two, four and one nodes respectively. The inputs are represented by i_1 and i_2 while the output is represented by t_1.

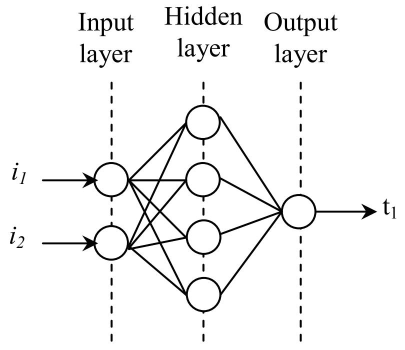

Fig. 1: An example of multi-layer feedforward neural network with single hidden layer.

In MLFNN, the data run rigidly from the input layer to the output layer via several processing units. Despite having no memory, the MLFNN output is heavily dependent on the current input and weight values in the network of nodes. The processing units, also known as the activation functions, can be linear and non-linear, or a combination of both. However, the activation function for each node in the same layer has to be similar. In this study, the activation function in the hidden layer was selected from two non-linear functions, i.e the logistic sigmoid function, *logsig* [15] and the tangent sigmoid function, *tansig* [16]. These non-linear transfer functions would provide output values from 0 to 1. On the other hand, the linear transfer function, *purelin* was chosen for the output layer such that the final output from the network could have any value which is not limited.

Nevertheless, MLFNN often encounters a crucial problem in its implementation. The primary challenge in MLFNN design is to select the appropriate training parameters and topology of the network. This process involves the selection of the optimal number of hidden nodes in its hidden layer, the learning rate and the momentum rate. In many cases, the selection of the number of nodes is conducted using the common trial-and-error method as there is no strict approach for the selection [17]. If the selected number of nodes is too small, the MLFNN is unable to be trained effectively. On the contrary, the training duration would be longer if the selected number of nodes is too large. In fact, too many nodes may result in over-generalization of the prediction [18]. On the other hand, higher learning rate results in slower convergence of the MLFNN whereas smaller learning rate results in more iterative updates required for convergence. Apart from that, if the momentum rate is not optimal, the MLFNN would encounter slower convergence as it is not able to learn the previous weight patterns that would lead to convergence.

In this study, an MLFNN with single hidden layer has been selected to predict the output performance of a grid-connected PV (GCPV) system. The inputs to the MLFNN are the solar irradiation, *SI* and ambient temperature, *AT* data patterns whereas the sole output of the MLFNN is the kilowatt-hour energy generated by the GCPV system. The GCPV system is located at the rooftop of Green Energy Office building, Malaysian Energy Centre (PTM), Bandar Baru Bangi, Malaysia. The system comprises 27kWp mono-crystalline PV array and a 24kW inverter (IG6). Each irradiance and temperature sensor was connected to the inverter while all data were recorded at fifteen minute interval.

The MLFNN design consists of two basic steps, i.e. the training process and the testing process. The training process started with the presentation of training data patterns to the MLFNN. The training data were later normalized to ensure that all input and output data patterns were transcribed into a common range of values [19]. Thus, the search of the correlation among these input and output patterns was simplified. The min-max normalization [20] had been used in this study. It was performed using

$$x_{norm} = \left(Max_{norm} - Min_{norm}\right) \times \left[\frac{x_{actual} - D_{\min}}{D_{\max} - D_{\min}}\right] + Min_{norm} \qquad (1)$$

where x_{norm} is the normalized data value and x_{actual} is the actual data value to be normalized. D_{min} is the minimum actual data value while D_{max} is the maximum actual data value. In addition, Max_{norm} is the predetermined maximum normalized data value which is equal to 1 whereas Min_{norm} is the predetermined minimum normalized data value which is equal to -1.

After normalization, the training data were propagated throughout the network using a learning algorithm. The Levenberg-Marquardt algorithm, *trainlm* has been commonly used as the learning algorithm because it has frequently outperformed other gradient-descent based learning algorithms [21-22]. The *trainlm* operates effectively such that the prediction error can be decreased at the end of each iterative update during the training process. Nevertheless, besides *trainlm*, the scaled conjugate gradient algorithm, *trainscg* had also been proven to produce satisfactory prediction performance of grid-connected PV system output [23]. In addition, the BFGS quasi-Newton backpropagation algorithm, *trainbfg* was found to outperform *trainlm* in training MLFNN [16]. Apart from that, *trainbfg* is also expected to have similar performance with *trainlm*. However, for larger size of network, more extensive computation is required for *trainbfg* [24]. Therefore, as these learning algorithms had proven useful [25-26], an optimal learning algorithm must be selected to suit the prediction task implemented in this study.

The training performance was quantified using the correlation coefficient, *R* which represents the relationship between the target output and the simulated output. The values of *R* can vary from 0 to 1. The best prediction performance is achieved when *R* is equal to unity.

Once the training was completed, testing process was performed to validate the training process. Different data patterns were initially presented to the trained network. Then, the network is simulated using the new data patterns. Finally, the *R* of the data patterns was computed. In this study, the training process was improved by presenting an automatic selection of the number of hidden nodes, the learning rate and the momentum rate using EP and AIS.

3 Proposed Evolutionary Programming-ANN Algorithm

EP in [27] is a branch of evolutionary computing (EC)-based search technique which classified under the artificial intelligence (AI). EP basically comprises several important processes such as initialization, fitness evaluation, mutation and selection [7]. In this study, EP was used to search for the optimal number of hidden nodes in its hidden layer, x_1 the learning rate, x_2, the momentum rate, x_3, the transfer function, x_4 and the training algorithm, x_5 during the MLFNN training such that the *R* of the prediction was maximized. These training parameters represent the decision variables required to be evolved in EP. The proposed evolutionary neural network (ENN) algorithm, known as the evolutionary programming- ANN (EPANN), was written in Matlab software package. The algorithm is implemented using the following steps

Step 1: (Initialization)- generate *M* population of sets of random numbers, x_1, x_2, x_3, x_4 and x_5. x_1 was set to have integer values between 1 to 20. On the other hand, both x_2 and x_3 were set to have continuous values between 0 and 1. In addition, x_4 was transcribed to a value of either 1 (*logsig*) or 2 (*tansig*). x_5 was set to be either 1 (*trainlm*), 2 (*trainscg*) or 3 (*trainbfg*) as these learning algorithms had been proven successful in many ANN training cases. Each set of random numbers forms the initial candidates of the optimal solutions known as parent. The fitness value for each parent is then evaluated by training the MLFNN to determine the *R* value of the prediction.

Step 2: (Mutation)- Mutate each parent based on statistical values obtained from the parent population. Each parent is mutated using the following equation to produce a mutated parent known as offspring.

$$x'_{i,k} = x_{i,k} + N_{i,k}\left(0, \sigma_{i,k}\right) \tag{2}$$

where $x_{i,k}$ is the value of *i*-th parent for the *k*-th decision variable and $N_{i,k}(0,\sigma_{i,k})$ is the Gaussian random number with mean zero and standard deviation $\sigma_{i,k}$. The value of $\sigma_{i,k}$ can be computed using

$$\sigma_{i,k} = \beta \cdot \left(\frac{f_{i,k}}{f_{k,\max}}\right)\left(x_{k,\max} - x_{k,\min}\right) \tag{3}$$

where β is the scaling factor for mutation and $f_{i,k}$ is the fitness value of the *i*-th parent in the current population for the *k*-th decision variables. At this stage, *M* offspring was produced. These offspring present another set of candidates for the optimal solutions.

Step 3: (Fitness evaluation)- Determine the fitness value for each offspring using the training process described in step 1.

Step 4: (Selection)- Select the best *M* candidates among the 2*M* population of parents and offspring according to their fitness values. *R* is used to describe the fitness value of the candidates. Candidates with lower *R* value are preferred compared to the candidates with larger *R*. Once selection has been performed, the stopping criteria are tested to terminate the evolution. The stopping criteria are described as follows:

$$R_{\max} - R_{\min} \leq 0.001 \tag{4}$$

$$x_{1,\max} - x_{1,\min} = 0 \tag{5}$$

If the stopping criteria are not satisfied, step 2 is repeated. Otherwise, evolution is stopped and the best *N* candidates are presented as the optimal solutions.

4 Proposed Artificial Immune System-ANN Algorithm

The second ENN algorithm, known as artificial immune system-ANN (AISANN) proposed in this study was developed using AIS. AIS is often described as an adaptive-based optimization technique inspired by the observation of immune function, theories and paradigms [28]. Major processes of AIS algorithm based on clonal selection principle include initialization, cloning, hyper-mutation and selection. The AIS algorithms can be mainly classified into two categories, namely the population-based algorithms and the network-based algorithms. In this study, a sub-class of the population-based algorithm modified from the standard clonal selection algorithm had been introduced to evolve the MLFNN. In clonal selection process, the potential candidates are presented as antibodies while the expected solutions are considered as antigens. As the affinity values of the antibodies can be different, the cloning of antibodies is usually performed based on the affinity values before affinity maturation is performed using a hyper-mutation process. Nevertheless, in this study, the proposed algorithm was developed using uniform clonal process with the exclusion of hypermutation [29]. The mutation of each clone was performed using constant mutation rate as pervious study had shown that this mutation method had been adequate to provide satisfactory results [30]. Apart from that, the affinity calculation was associated to the *R* for simplicity. The proposed AISANN is almost similar to the proposed EPANN explained previously with the modifications of the steps outlined in the EPANN algorithm. The AISANN algorithm was also written in Matlab.

- Cloning process: After the initialization of the *M* parents has been completed, clone each parent population such that n_c clones are produced from each x_1, x_2, x_3, x_4 and x_5. Thus, the new number of parents in the population becomes $M \times n_c$.
- Mutation: during the mutation process, instead of *M* offspring, $M \times n_c$ offspring will be produced.
- Selection: select the best *M* candidates from the $M \times n_c$ offspring based on their fitness values. Parent population should be excluded during the selection to avoid redundancies.

5 Results and Discussion

The values of the selected MLFNN settings are shown in Table 1. These values were fixed throughout the training process. Similar number of data patterns used for the prediction was allocated for both training and testing. In addition, the mean square error goal was set to be 0.001, which is sufficiently small to yield a satisfactory prediction for majority of data patterns. Besides that, the maximum number of iterative updates was found to 600. This value was adequate for the training as most of the trained patterns managed to converge much earlier before reaching the maximum number of iterative updates.

Upon the completion of training process, testing process had been performed to stop the training process. The overall computation time for both training and testing processes was recorded in each proposed ENN. The overall training and testing performance is discussed in section 5.1. On the other hand, the detailed fitness performance of each algorithm is presented in section 5.2. As the MLFNN operation is random in nature, inconsistent fitness values were expected despite having similar MLFNN design parameters. As a result, besides the maximum and minimum fitness values for each evolution, the average fitness value and the standard deviation were also computed to provide a better representation of the fitness behavior for each generation. The standard deviation profile for each algorithm is presented in section 5.3. In addition, the convergence of the number of nodes in hidden layer, the learning rate, the momentum rate, the transfer function and the learning (training) algorithm are presented in section 5.4, 5.5, 5.6 and 5.7 respectively.

Table 1: Fixed ANN training parameter settings

Parameters	Value
Number of training patterns	386
Number of testing patterns	386
Number of nodes in the input layer	2
Number of nodes in the output layer	2
Mean square error goal	10^{-3}
Maximum number of iterative updates	600

5.1 Overall performance

The results of the optimal MLFNN design parameters as well as the performance of EPANN and AISANN are compared in Table 2. Twenty set of random numbers were generated to form the initial population in both algorithms. In addition, the number of clones was set to 5 in the AISANN algorithm.

In Table 2, the optimal number of nodes in the hidden layer was found to be 4 using both algorithms. The learning rates employed by EPANN were found to be higher than those in AISANN. The minimum and maximum learning rates in EPANN were discovered to be 0.7720 and 0.7796 respectively. On the other hand, the minimum and maximum learning rates in AISANN were discovered to be 0.2724 and 0.5574 respectively. Nevertheless, the momentum rates in EPANN showed better convergence than the momentum rates in AISANN. The maximum momentum rate of AISANN was approximately twice higher than the maximum momentum rate of EPANN but the minimum momentum rates for both algorithms are almost equal. Apart from that, unlike the transfer function of AISANN which did not converge to a single function at the end of the evolution process, the transfer function of EPANN managed to converge to a similar function. The optimal transfer function of the hidden layer in EPANN was found to be *logsig* while the AISANN had failed to produce a single transfer function. However, both EPANN and AISANN showed that *trainbfg* is the optimal learning algorithm. Besides that, the maximum and minimum fitness values found in the final population of both algorithms were approximately the same. Nonetheless, the average R value in AISANN was 0.008% higher than the average R value in EPANN while the standard deviation of fitness in AISANN was slightly lower than the standard deviation in EPANN.

Table 2: Prediction performance of EPANN and AISANN

Parameter/Results	EPANN	AISANN
Number of initial parent, M	20	20
Number of clones, n_c	-	5
Number of nodes in hidden layer, x_1	4	4
Minimum value of x_2	0.7720	0.2724
Maximum value of x_2	0.7796	0.5574
Minimum value of x_3	0.1033	0.1012
Maximum value of x_3	0.1154	0.2857
Minimum value of x_4	1 (*logsig*)	1 (*logsig*)
Maximum value of x_4	1 (*logsig*)	2 (*tansig*)
Minimum value of x_5	3 (*trainbfg*)	3 (*trainbfg*)
Maximum value of x_5	3 (*trainbfg*)	3 (*trainbfg*)
Minimum R in training, dimensionless	0.99854	0.99862
Maximum R in training, dimensionless	0.99871	0.99872
Average R in training, dimensionless	0.99858	0.99866
Standard deviation	4.9×10^{-5}	3.12×10^{-5}
R in testing	0.99839	0.99832
Required number of iteration before stoppage	18	6
Average computation time, in seconds	4345.83	8721.62

In contrast, the testing process showed that the R value of EPANN was 0.007% higher than the R value of AISANN. Apart from that, although the required number of iterations in EPANN was thrice higher than the required number of iterations in AISANN, EPANN had shown faster computation time compared to AISANN. EPANN was found to be almost two times faster than

AISANN. Therefore, after considering the *R* performance of EPANN which were relatively similar to *R* of AISANN as well as the superior computation time, EPANN was selected as the better option for implementing the ENN.

5.2 Fitness performance

The fitness performances of EPANN and AISANN are illustrated in Fig. 2 and Fig. 3 respectively. In Fig. 2, the fitness values in EPANN had actually converged since the first evolution according to equation 4. However, the number of nodes had not come to an absolute convergence. Thus, the evolution process continued and stopped at the eighteen evolution. As the final minimum fitness values obtained were very close to unity, the prediction task was deemed successful. Besides that, the average fitness also showed a fluctuating trend towards the maximum value. However, the final average fitness value was slightly lower than the maximum average fitness achieved during the evolution process. While the ANN operation is random in nature, this phenomenon is acceptable since the number nodes had not yet converged to its final value. Similarly, the maximum fitness value had fluctuated before reaching a value which was very close to its maximum value.

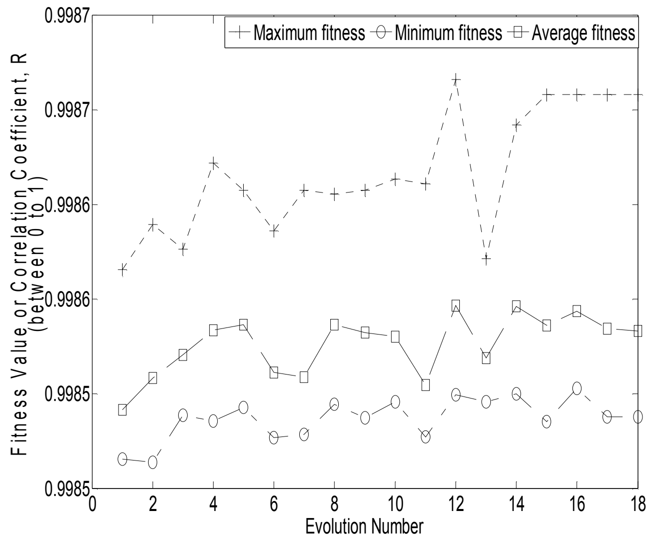

Fig. 2: Fitness convergence of EPANN

On the other hand, in Fig. 3, the fitness values had initially converged at the first evolution before the algorithm reached the overall convergence on the sixth evolution. Apart from that, unlike the maximum fitness value, the average fitness and minimum fitness in AISANN had shown a consistent increasing trend before convergence. When compared to EPANN during the first six evolutions, the AISANN had shown better consistency of fitness values throughout the evolution.

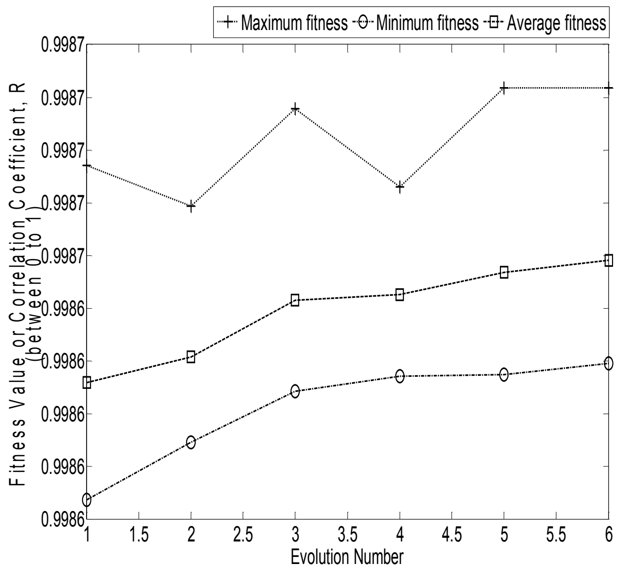

Fig. 3: Fitness convergence of AISANN

5.3 Standard deviation performance

In terms of standard deviation, both ENN models showed fluctuating standard deviation values throughout the evolution process. The standard deviation performance of EPANN and AISANN are illustrated in Fig. 4 and Fig. 5 respectively. The standard deviation was found to be directly proportional to the difference between the minimum and maximum fitness value in the evolution process. At the end of the evolution process, the final standard evolution values produced by both algorithms were found to be approximately equal and sufficiently low. Thus, the degree of variation among the fitness values in the final population for each algorithm was low, which subsequently indicated good prediction results.

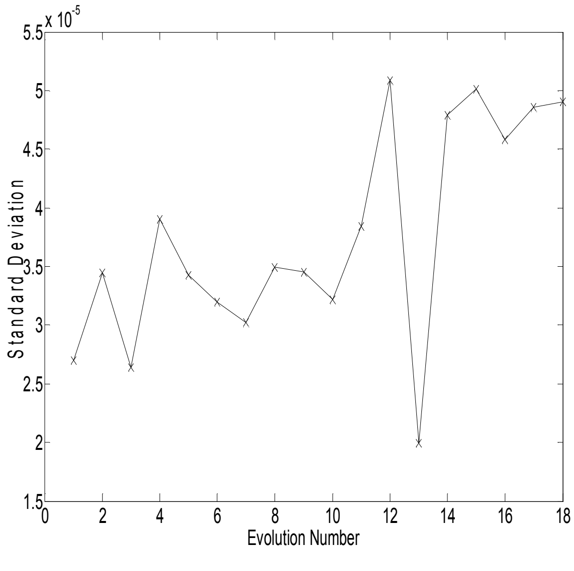

Fig. 4: Standard deviation performance of EPANN

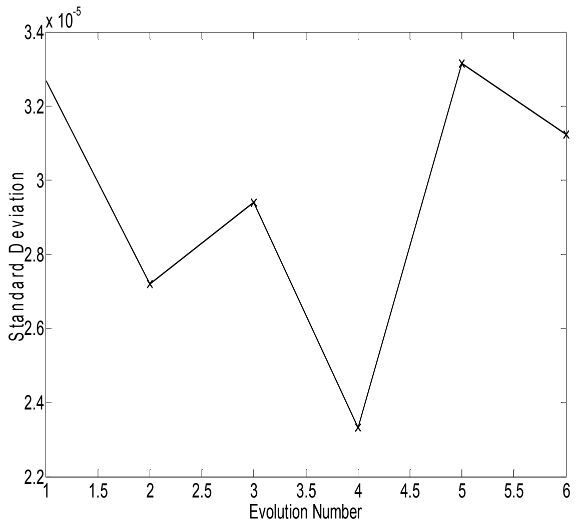

Fig. 5: Standard deviation performance of AISANN

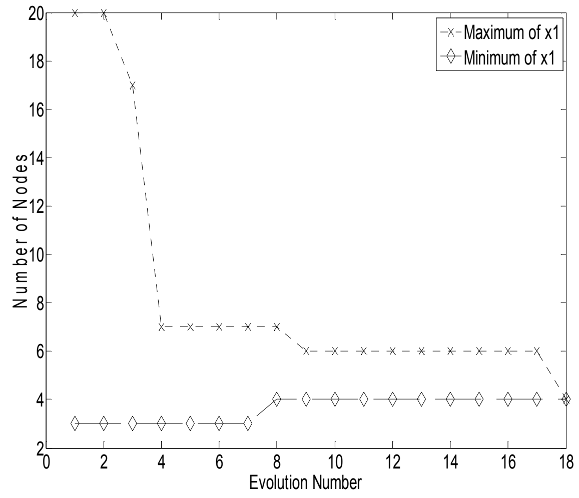

Fig. 6: Convergence of number of nodes in EPANN

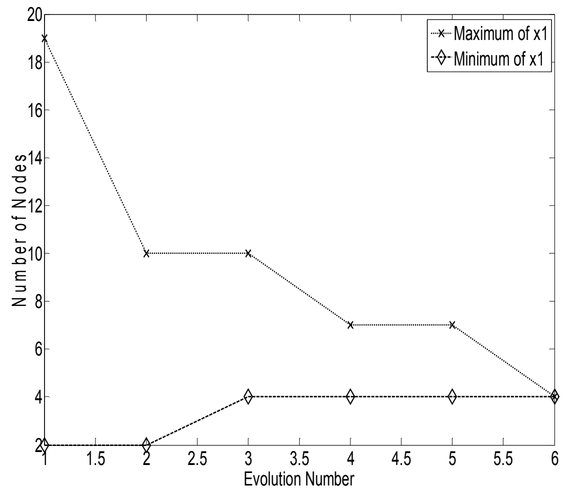

Fig. 7: Convergence of number of nodes in AISANN

5.4 Number of nodes

Besides that, the convergence of number nodes in the hidden layer in EPANN and AISANN are shown in Fig. 6 and Fig. 7 respectively. In Fig. 6, the number of nodes in EPANN managed to converge only at the eighteen evolution. In contrast, the number of nodes in AISANN had converged on the sixth evolution. In addition, the maximum and minimum number of nodes in AISANN had reached the optimal value at an earlier evolution compared to those in EPANN. The minimum number of nodes in AISANN had reached the optimal value at the third evolution while the minimum number of nodes in EPANN only reached the optimal value on the eighth evolution. On the other hand, the maximum number of nodes in AISANN had reached the optimal value on the sixth evolution whereas the EPANN produced the optimal value only at the eighteen evolution. However, when their performance were averaged out based on the overall computation time, the EPANN had actually produced faster optimal results compared to AISANN.

5.5 Learning rates and momentum rates

In Fig. 8, the EPANN had produced a better convergence of the learning rate and the momentum rate at the end of the evolution process compared to the learning rate and momentum rate of AISANN shown in Fig. 9. In EPANN, the final difference between the maximum and minimum learning rates was 0.0076 whereas the final difference in AISANN was 0.2850. Thus, the EPANN had produced 97.3% lower final difference in the learning rate compared to AISANN. On the other hand, the final difference between the maximum and minimum momentum rates in EPANN was 0.0121 while the final difference in AISANN was 0.1845. Correspondingly, the EPANN had yielded a final difference which is 93.4% lower than the final difference produced by AISANN. Due to the lower final difference of learning rate and momentum rates, the EPANN had actually outperformed AISANN in selecting the optimal values for both training parameters.

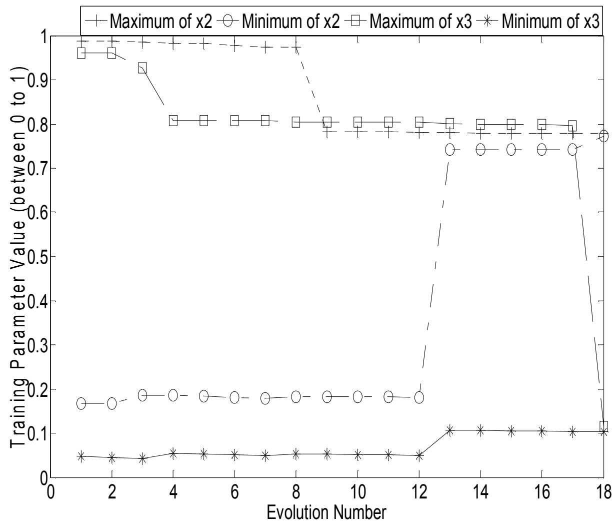

Fig. 8: Convergence of learning rate and momentum rate in EPANN

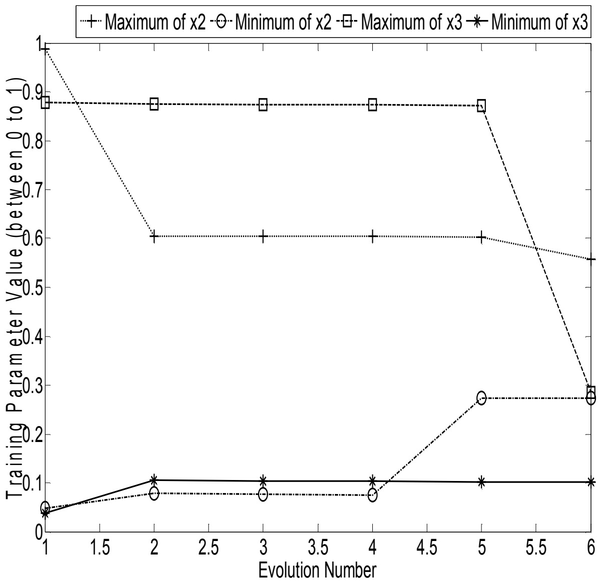

Fig. 9: Convergence of learning rate and momentum rate in AISANN

5.6 Transfer functions

During the selection of the optimal transfer function for the hidden layer, the EPANN had managed to produce an absolute convergence of the transfer function code. The optimal transfer function was *logsig*. This result was obtained after the eighteen evolution as illustrated in Fig. 10. In contrast, the AISANN had failed to yield an absolute convergence of the transfer function as shown in Fig. 11. Both *logsig* and *tansig* were found to be comparably suitable for the AISANN training process. Due to the absolute convergence demonstrated by EPANN, EPANN had been found to clearly outperform the AISANN in selecting the optimal transfer function. In addition, the optimal transfer function for the ANN was *logsig*.

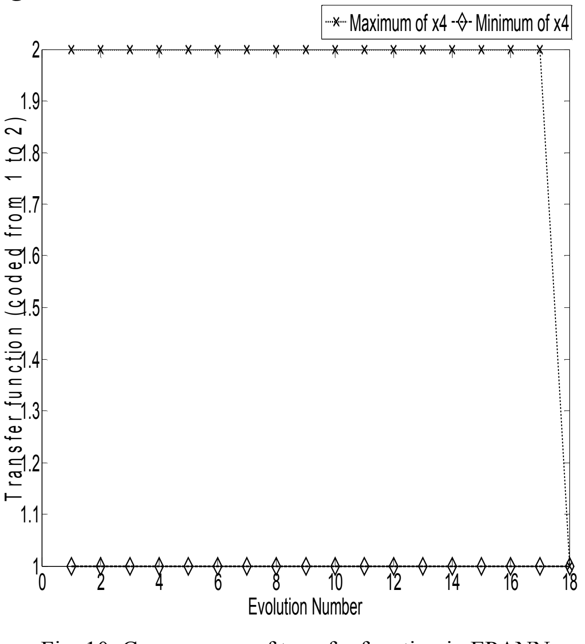

Fig. 10: Convergence of transfer function in EPANN

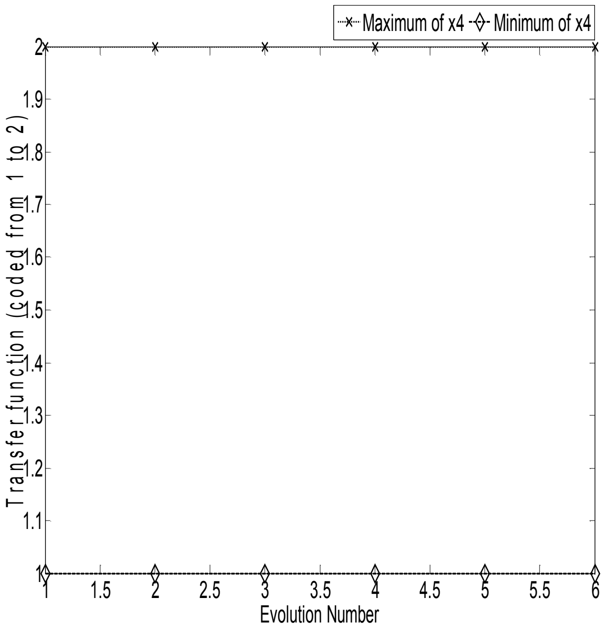

Fig. 11: Convergence of transfer function in AISANN

5.7 Learning algorithms

In the search of the optimal learning algorithm for the prediction model, both EPANN and AISANN had yielded *trainbfg* as the optimal training algorithm as shown in Fig. 12 and Fig. 13 respectively. In addition, the learning algorithm had converged on the second evolution in both EPANN and AISANN. Thus, both algorithms were found to require less effort in determining the optimal learning algorithm compared to other MLFNN design parameters. Therefore, *trainbfg* had clearly performed better than its competitors.

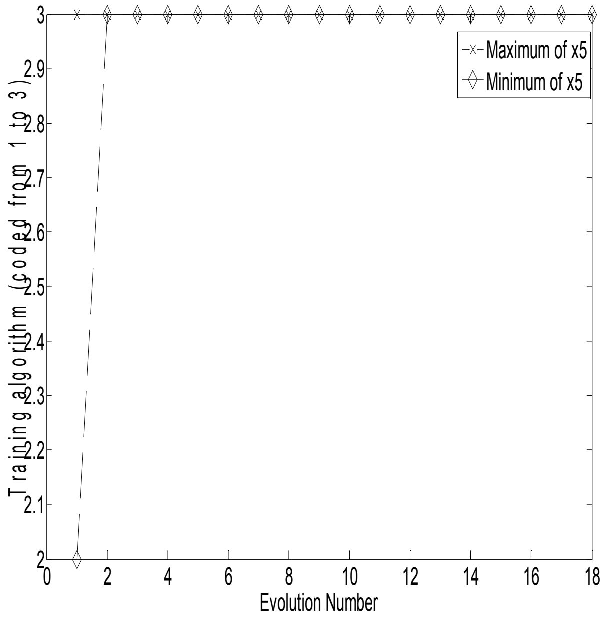

Fig. 12: Convergence of learning algorithm in EPANN

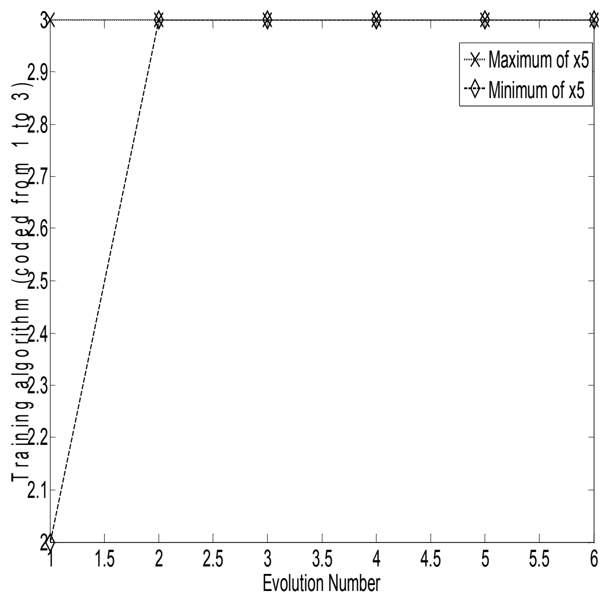

Fig. 13: Convergence of learning algorithm in AISANN

5 Conclusion

In this study, two intelligent algorithms have been developed for evolving the MLFNN for predicting the kWh power output from a grid-connected PV system. The EPANN algorithm was found to be superior compared to AISANN algorithm in terms of computation time, although the average *R* value produced by EPANN was slightly lower than the average *R* value in AISANN. Nevertheless, the EPANN had yielded higher value of *R* value during the testing process. Moreover, despite having almost similar performance in the search of the optimal fitness value, number of nodes in the hidden layer and the learning algorithm, the EPANN had outperformed AISANN in the search of the learning rate, momentum rate and transfer function. Therefore, EP was selected as the better optimizer for the proposed evolutionary MLFNN.

References:

[1] S. I. Sulaiman, T. K. A. Rahman, and I. Musirin, Optimizing one-hidden layer neural network design using evolutionary programming, in *Proceedings of the 5th International Colloquium on Signal Processing & Its Applications*, 6-8 March 2009, pp. 288-293.

[2] S. Shaari and A. M. Omar, *Grid-connected Photovoltaic System Design and Installation*. Bangi: Pusat Tenaga Malaysia, 2008.

[3] I. Ashraf and A. Chandra, Artificial Neural Network Based Models for Forecasting Electricity Generation of Grid Connected Solar PV Power Plant, *Int. Journal of Global Energy Issues*, Vol. 21, Vo. 1/2, 2004, pp. 119-130.

[4] M. Balzani and A. Reatti, Neural Network Based Model of a PV Array for the Optimum Performance of PV System, in *Proc. 2005 PhD Research in Microelectronics and Electronics Conf.*, Vol. 2, pp. 123-126.

[5] D. B. Fogel, L. J. Fogel, and V. W. Porto, Evolutionary Programming for Training Neural Networks, in *International Joint Conference on Neural Networks*, 1990, pp. 601-605.

[6] W. Gao, Study on new evolutionary neural network, in *Proceedings of the Second International Conference on Machine Learning and Cybernetics*, 2003, pp. 1287-1292.

[7] S. I. Sulaiman, T. K. A. Rahman, I. Musirin, and S. Shaari, Optimizing three-layer neural network model for grid-connected photovoltaic system output prediction, in *Proceedings of the Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA 2009)*, 25-26 July 2009, pp. 338-343.

[8] S. Kumar, *Neural Networks A Classroom Approach*, Singapore: McGraw-Hill, 2005, p. 61.

[9] C. Fontes, P. R. B. Guimaraes, A. C. Gondim, S. B. Neves, and L. Carvalho, Development of an artificial neural network for predicting oil melting point through pattern recognition, in *4th Mercosur Congress on Process Systems Engineering*, 2005.

[10] D. W. Gao, P. Wang, and H. Liang, Optimization of hidden nodes and training times in ANN-QSAR model, *Environmental Informatics Archives,* Vol. 5, 2007, pp. 464-468.

[11] E. Inohira and H. Yokoi, An optimal design method for artificial neural networks by using the design of experiments, *Journal of Advanced Computational Intelligence and Intelligent Informatics,* Vol. 11, No. 6, 2007, pp. 593-594.

[12] F. Wang, V. K. Devabhaktuni, C. Xi, and Q.-J. Zhang, Neural network structures and training algorithms for RF and microwave applications, *International Journal of RF and Microwave CAE,* Vol. 9, 1999, pp. 216-240.

[13] K. K. Aggarwal, Y. Singh, P. Chandra, and M. Puri, Bayesian regularization in a neural network model to estimate lines of code using function points, *Journal of Computer Sciences,* Vol. 1, No. 4, 2005, pp. 505-509.

[14] T. Senjyu, H. Takara, K. Uezato, and T. Funabashi, One-hour-ahead Load Forecasting Using Neural Network, *IEEE Transactions on Power Systems,* Vol. 17, No. 1, 2002, pp. 113-118.

[15] H. S. Hippert, C. E. Pedreira, and R. C. Souza, Neural Neworks for Short-term Load Forecasting: A Review and Evaluation, *IEEE Transactions on Power Systems,* Vol. 16, No. 1, 2001, pp. 44-55.

[16] S. R. I. Gabran, S. Zhang, M. M. A. Salama, R. R. Mansour, and C. George, Real-time automated neural-network sleep classifier using single channel EEG recording for detection of narcolepsy episodes, in *30th Annual International IEEE EMBS Conference*, 2008, pp. 1136-1139.

[17] O. Kisi, "Multi-layer perceptrons with Levenberg-Marquardt training algorithm for suspended sediment concentration prediction and estimation," *Hydrological Sciences,* Vol. 49, No. 6, 2004, pp. 1025-1040.

[18] N. Subramanian, A. Yajnik, and R. S. R. Murthy, Artificial neural network as an alternative to multiple regression analysis in optimizing formulation parameters of cytarabine liposomes, *AAPS PharmSciTech,* Vol. 5, No. 1, 2004, pp. 1-9.

[19] G. O. Tirian and C. B. Pinca, Applications of neural networks in continuous casting, *WSEAS Transactions on Systems,* Vol. 8, No. 6, 2009, pp. 693-702.

[20] K.L Priddy and P.E. Keller, *Artificial Neural Networks: An Introduction*, New Delhi: Prentice Hall of India, 2007, p. 121.

[21] M. T. Hagan and M. B. Menhaj, Training feedforward networks with the Marquardt algorithm, *IEEE Transactions on Neural Networks,* Vol. 5, No. 6, 1994, pp. 989-993.

[22] M. Zandieh, A. Azadeh, B. Hadadi, and M. Saberi, Application of artificial neural networks for airline number of passenger estimation in time series state, *Journal of Applied Science,* Vol. 9, No. 6, 2009, pp. 1001-1013.

[23] S. I. Sulaiman, T. K. A. Rahman, I. Musirin, and S. Shaari, Asessment of Different Training Algorithms in ANN Model for Grid-Photovoltaic System Output Prediction, in *Progress of Solar Energy Research & Development*, 21-22 October 2008, pp. 101-107.

[24] N. V. N. I. Kiran, M. P. devi, and G. V. Lakshmi, Effective control chart pattern recognition using artificial neural networks, *IJCSNS International Journal of Computer Science and Network Security,* Vol. 10, No. 3, 2010, pp. 194-199.

[25] J. R. Salinas, F. Garcia-Lagos, G. Joya, and F. Sandoval, Sine-fitting multiharmonic algorithms implemented by artificial neural networks, *Neurocomputing,* Vol. 72, No. 16-18, 2009, pp. 3640-3648.

[26] A. Abraham, Meta learning evolutionary artificial neural networks, *Neurocomputing,* Vol. 56, 2004, pp. 1-38.

[27] H.-P. Schwefel, On the evolution of evolutionary computation, *Computational Intelligence: Imitating Life*, J. Zurada, R. M. II, and C. Robinson, Eds., 1994, pp. 147-159.

[28] L. N. d. Castro and J. I. Timmis, Artificial immune systems as a novel soft computing paradigm, *Soft Computing - A Fusion of Foundations, Methodologies and Applications,* Vol. 7, No. 8, 2003, pp. 526-544.

[29] I. Musirin, N. D. M. Radzi, M. M. Othman, M. K. Idris, and T. K. A. Rahman, Voltage profile improvement using unified power flow controller via artificial immune system, *WSEAS Transactions on Power Systems,* Vol. 3, No. 4, 2008, pp. 194-204.

[30] S. Ishak, A. F. Abidin, and T. K. A. Rahman, Static Var compensator planning using artificial immune system for loss minimisation and voltage improvement, in *Proceedings of the National Power and Energy Conference*, Kuala Lumpur, 2004, pp. 41-45.