ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **19**, Numbers 1-2, 2016, 98–115

Evolutionary Networks with Generalized Neurons Applied in Battery SoC Estimation

Ioan DUMITRACHE[1], Dragoş C. POPESCU[1,2]

[1] Automatic Control and Systems Engineering Department,
University Politehnica of Bucharest
[2] Horia Hulubei National Institute for R&D in Physics
and Nuclear Engineering (IFIN-HH)

E-mail: {ioan.dumitrache,dragos.popescu}@acse.pub.ro

Abstract. In this paper is presented a new hybrid methodology using a Generalized Neuron with a Genetic Algorithm. After a short presentation of the capabilities of GN is shown a possibility to optimize the parameters by using a GA. As a case study is proven the efficiency of this new GNGA architecture in optimization of the estimation for battery State of Charge. Are presented some simulation results in comparison with the conventional Artificial Neural Networks.

1. Introduction

Intelligence provides the power to derive the answer and not simply arrive to the answer in context of problem solving. We are interested to create an intelligent machine which will work in a similar way as human beings. One of the important area in soft computing is intuitive consciousness / wisdom which must be included in the computers.

Soft computing methodologies (Artificial Neural Networks, Fuzzy Logic, Evolutionary Computing, Probabilistic Reasoning, Machine Learning, Wisdom based Expert Systems) mimic consciousness and cognition [4]. In soft computing the problem to be solved is represented in a such way that the "state" of the system can somehow be calculated and compared with some desired state. By using evolutionary computing and neural computing adaptation of the system's parameters and structure allows slowly convergence towards the solution.

Soft Computing (SC) is a partnership in which each methodology has contribution in problem solving. Therefore SC may be viewed as a foundation component for the emerging field of conceptual intelligence.

One particular feature of SC is acquisition of intelligence and knowledge from inaccurate and uncertain data. Main components of SC, inspired from natural phenomena are Artificial Neural Networks, Fuzzy logic and Genetic Algorithms having a real synergism [10, 12, 9]. SC provides the ability of parallel computing as in human brain and also reasoning and learning in an uncertain and imprecise environmental conditions.

Considering different features of these main methodologies in soft computing it is possible to develop some hybrid methodologies with better capabilities to solve complex problems. In last 10 years, several adaptive hybrid soft computing frameworks have been developed for decision support, process control, robotics, classification and pattern recognition. Some hybrid methodologies use a combination of different knowledge representation schemes, decision making models and learning strategies to solve computational tasks.

Have been developed different hybrid architectures by combination, transformation, fusion and association for some applications [1]. Most applied hybrid methodologies are Neuro-Fuzzy (FALCON, ANFIS, FINEST, FUN) Geno-Fuzzy (EFUNN, SONFIN) and Geno-Neuro-Fuzzy. Developing intelligent systems by hybridization is an open-ended rather than a conservative concept. In Fig.1 are shown different mutual interactions between intelligent methodologies to create hybrid soft computing architectures.

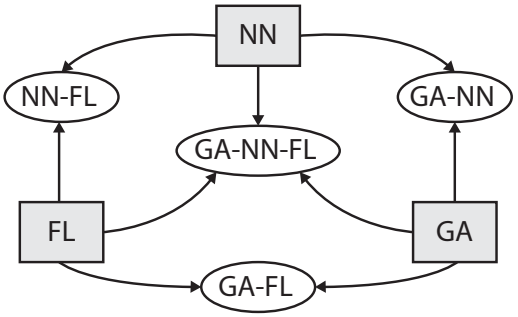

Fig. 1. Mutual interactions between intelligent methodologies.

Each methodology plays an important role in the development of hybrid architectures by complementarity and integration using combination, fusion, transformation and association schemes [10].

Neural Networks stores knowledge in a distributive manner within its weights which have been determined by learning. The topology of the network, the number of neurons in different layers, the weights and the training procedure are the most important characteristic and must be designed in this distributed knowledge representation.

Evolutionary computation works with a set of candidate that form populations in evolution towards better candidate solution via the selection operation and genetic operators (crossover and mutation).

2. GA-ANN hybrid systems

The problem to find a set of weights for a given ANN and a set of training examples is a NP-complete problem.

It is known that for a training set T and a given n-dimensional argument x, an associated target value t will be approximated by the neural network output:

$$T = \{(x_i, t_i), i = 1, ..., n\}$$

with:

$$E_t = \sum_{i=1}^{n} ||y_i - t_i||^2 \text{ and } y_i \text{ as the network output}$$

To optimize the ANN it is necessary to create an evolutionary ANN as a general framework for investigating various aspects of simulated evolution and learning.

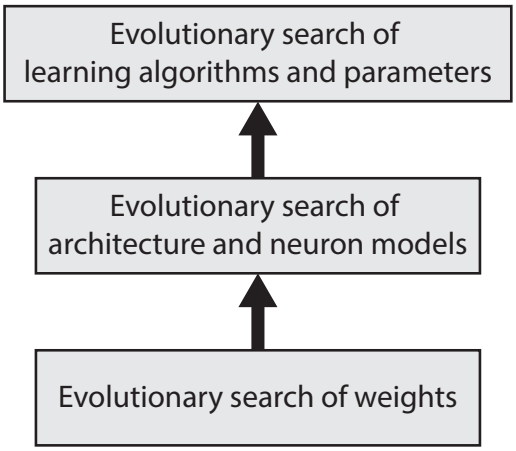

Fig. 2. Steps for optimizing Neural Networks using evolutionary strategies.

The connection weights may be represented as binary strings with a certain length and the whole network is encoded by concatenation of all the connection weights in the chromosome. For the optimal neural network, the learning algorithms must be adapted dynamically in function of the architecture and the given problem. The adaptation could be achieved by constructive and distributive algorithms [1].

For optimizing ANN and FS the evolutionary algorithms are the best. But the success in applications depend on the genotype representation of the different layers. The population-based collective learning process, self adaptation and robustness are the most important features of genetic algorithms in comparison with other global optimization techniques.

Evolutionary algorithms and ANN are the theoretical results based on biological principles. For optimal design of neural networks by using an evolutionary algorithm could try to adapt the size of the network to problem, starting with a small network and adding layers and units until is found a solution or starting with a layer network and remove the unnecessary parts.

To design a neural network we have to follow the stages [3]:

1. a preprocessing of the data is made
2. the network topology and codification is chosen (network inputs ans outputs, coding, number of hidden layers and number of neurons per layer and the type of connectivity)
3. the network training method and learning parameters are chosen

For evolution of ANN must consider: adaptation of weights, adaptation of the network topology, adaptation of the learning rules.

For coding and representation we may use the binary real representation with direct or indirect coding [3]. Weight evolution is influenced by initial values of them and of the way how the GA changes them.

An optimal network architecture could be obtained as a result of a search problem in the architecture space, where each point represents a possible network architecture.

3. The Generalized Neuron model

The Generalized Neuron (GN) belongs to the class of bio-inspired models and methods. It was developed by D. K. Chaturvedi [4] with the aim of having a low complexity modeling tool, yet powerful. It is based on classic artificial neurons which are improved using fuzzy compensatory operators [14, 15].

The GN was successfully used in various applications. In Power Grids it was primarily used for modeling highly nonlinear and complex systems like power generators but also for control purposes (frequency and phase stabilizers) [5]. Good results similar to classical control theory adaptive solutions were obtained by developing control schemes implying a preliminary offline learning and also an online update. The GN was also used for short term load forecasting [7]. Here the GN was modified for achieving better results, thus, nine new variations were obtained.

The GN was used for developing an automatic aircraft landing system in [6] and also to predict the interest rate [16].

The core of the GN are two artificial neurons (Fig.3). The first neuron has summation aggregation with sigmoid activation. The other one has product aggregation with gaussian activation. Each neuron is suitable for mapping distinct types of problems because each activation function has its own specific nonlinearity and each aggregation function has its own way of processing the inputs. The output of the GN is obtained by applying a fuzzy compensatory operator over the two neuron outputs. Through this the benefits given by the two neurons are put together giving to the GA another degree of flexibility. The diversity of this architecture makes it suitable in modeling nonlinear and complex processes.

The GN has $x_1, x_2, ..., x_i, ...$ inputs, $w_{\Sigma 1}, w_{\Sigma 2}, ...$ are their weights for the summation neuron, $w_{\Pi 1}, w_{\Pi 2}, ...$ are their weights for the product neuron and w is the weight for the compensatory operator.

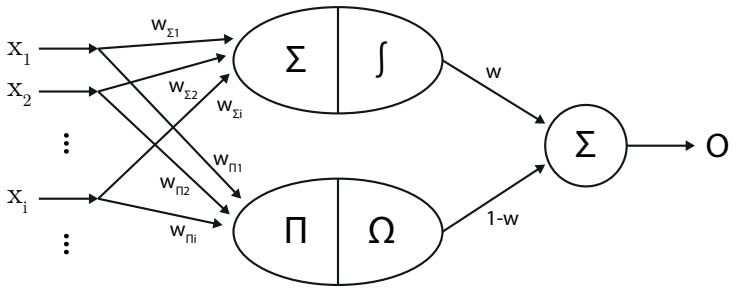

Fig. 3. Generalized neuron architecture [4].

Sigmoid and gaussian activation functions have the following expressions:

$$\Sigma(z) = \frac{1}{1 + e^{-z}} \tag{1}$$

$$\Omega(z) = a\, exp\left(-\frac{(z-b)^2}{2c^2}\right) + d \tag{2}$$

where a, b, c, d have their known meaning.

The number of parameters is very small compared to classic Artificial Neural Networks where good results are achieved for large hidden layers which increase the complexity of the model. The complexity of the GN is:

$$N = 2n + 1 \tag{3}$$

where n is the number of inputs.

3.1. Forward calculation

For using the GN in real applications and also for developing learn algorithms for it, it is very important to compute the output based on a set of inputs. The output for the summation neuron is:

$$O_{\Sigma} = \frac{1}{1 + e^{-\lambda_s \, s\_net}} \tag{4}$$

where $s\_net = \sum w_{\Sigma i} x_i + x_{0\Sigma}$ and $x_{0\Sigma}$ is the bias from the summation neuron.

The output for product neuron is:

$$O_{\Pi} = e^{-\lambda_p \, pi\_net^2} \tag{5}$$

where $pi\_net = \prod w_{\Pi i} x_i + x_{0\Pi}$ and $x_{0\Pi}$ is the bias from the product neuron.

The output from the GN is:

$$O = O_{\Pi}(1 - w) + O_{\Sigma} \, w \tag{6}$$

3.2. Learning Algorithm

In every parametric model a learning algorithm finds the optimal set of parameters such that, the model will act as close as possible to a desired behavior.

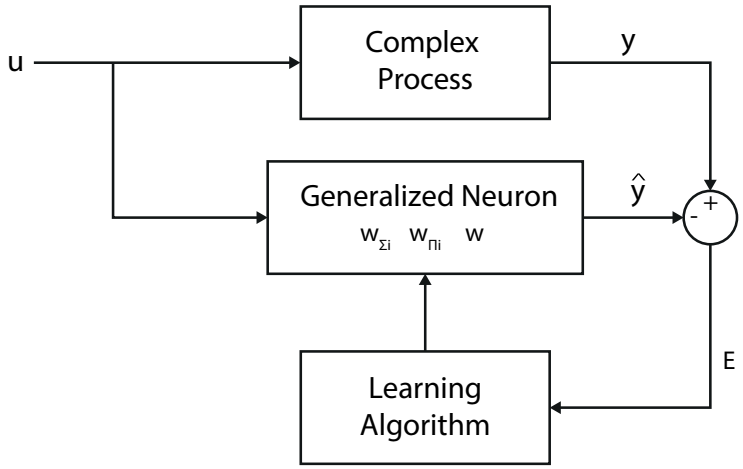

Fig. 4. Learning Algorithm for GN.

The GN learns by minimizing the error between its output ($\hat{y}$) and the desired output (y). Usually, the desired output is obtained by acquiring data from a real process that needs to be modeled. The Learning Algorithm tunes the weights and the biases in order to minimize the prediction error (Fig.4).

The prediction error is defined as follows:

$$E_P = \frac{1}{2} \sum E_i^2 \tag{7}$$

where $Ei = y_i - \hat{y}_i$

The indexing shows that the minimization is done over a big set of in-out data called learning set. This error is knows in literature as Residual sum of squares (RSS).

Computing the weights is achieved by solving an optimization problem where the prediction error is actually a cost function:

$$W = \arg\min_{W} E_P(W) \tag{8}$$

where W is a set containing all the weights and biases.

In the original paper [4] the Learning Algorithm is developed by solving the optimization problem using The Gradient Descent method. This approach is compared with an ANN using four standard benchmark problems.

On the EX-OR Problem the GN learns three times faster being three times more precise than an ANN. On the Mackey-Glass Time Series Problem, although the GN learns 50% slower, it achieves 10 times the precision of ANN. On the Character Recognition Problem, the GN learns 5 times faster and is almost 10 times more precise. On the "SIN(X1)*SIN(X2)" Problem the GN learns almost 100 times faster being 10 times more precise.

4. GNGA - Generalized Neuron with Genetic Algorithm learning

The Generalized Neuron was developed for mapping complex and nonlinear problems but having a low complexity model which achieves optimal computational effort. It was proposed as an efficient alternative to ANN and Neuro-Fuzzy Architectures. This goals aim toward real-time applications where low computational effort is desired.

For mapping complex and nonlinear systems / phenomena large training sets are necessary in order to catch all fine specificities and details. In fact, the bigger the training set, the better is the fidelity of the model. Because the GN has very few parameters ($N = 2n + 1$ where n is the number of inputs) it is very hard to train the GN for large training sets. The algorithm simply not converge. The problem can be intuitively seen as an overdetermined system of equations. This causes the cost function to be non-convex and to have many local minimum points.

We came up with the idea of using a Genetic Algorithm (GA) for learning the GN. We named this new hybrid architecture GNGA - Generalized Neuron with Genetic Algorithm learning. In our approach the hybridization is done by combination.

The GNGA can be used in two types of applications. If the complex process that is mapped is time invariant, the learning can be done only one time in the beginning (or rarely). In this case the GA can run for a long period of time (hours, days) to get better results.

On the other hand, if the process (structure, parameters, behavior) changes over time, an online update strategy needs to be envisaged. New and valuable information about the process variations is brought only after a considerable time frame. Thus, an update mechanism that runs very often would bring severe instability in the system. It is more efficient to use for each update iteration information acquired over a longer time frame. Thus, the GA has enough time to search for an improved model. Because the process varies, it is possible that while the GA updates the model, it will start to loose its precision. Because of this, when choosing the length of the time frame for updating the model, the user needs to find the perfect balance between how much the GA manages to improve and how much the loss of precision affects its application.

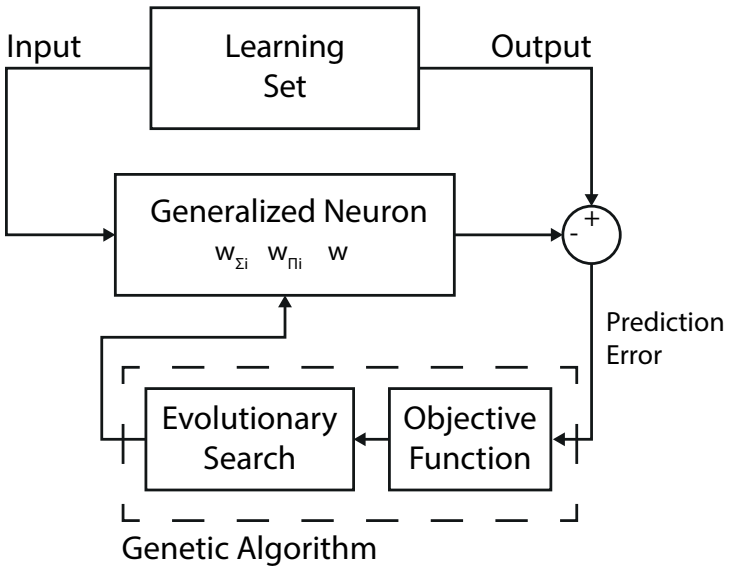

Fig. 5. GNGA Architecture.

The architecture of the GNGA uses a Learning Set like in any parametric model learning (Fig.5). The Learning Set embeds a considerable amount of characteristics from the process that has to be modeled. The GA contains two principal components: an objective function which expresses the prediction error as a cost and an Evolutionary Search unit that searches for an individual that has a lower cost (higher fitness).

5. SoC estimation using the GNGA

In order to test the performance of the GNGA we approached a classical problem in literature - SoC (State of Charge) estimation for a battery. In electrical applications (especially Electric Vehicles) it is very important to estimate

and predict essential battery parameters necessary for their safety operation (SoC, SoH, cycle life) [13].

Mathematical modeling of batteries is a very complex task because their dynamics are described by nonlinear differential equations with concentrated parameters. In [8] is developed a complex mathematical model for Lithium batteries, with the goal of estimating several important parameters. In [18] the behavior of the batteries is approximated with an electric circuit. Its parameters are estimated and adjusted in real-time, which provides very good SoC estimation results.

More recent works in predicting the SoC of a battery were done using intelligent methods. In [17] the SoC is estimated using an ANFIS model achieving an estimation error smaller than 3%. In [19], the ANFIS model is improved by using a reduced form genetic algorithm for learning . Thus, it is achieved a faster learning rate and lower estimation error than conventional learning methods.

5.1. Learning experiment

The definition of the SoC is:

$$SoC = \frac{C_{rem}[Ah]}{C[Ah]}, \tag{9}$$

where C is the total (nominal) battery capacity and C_{rem} is the remaining battery capacity (its capacity in a certain moment).

We can estimate the SoC by estimating the remaining capacity because its nominal capacity is a constant. The remaining capacity of a battery is defined as follows:

$$C[Ah] = \int_0^T i(t)\mathrm{d}t. \tag{10}$$

The remaining capacity of a battery is the electrical energy that can be provided during a total discharge.

It is determined by an extensive correlation analysis [2] that the most significant inputs for prediction models are the terminal voltage and the discharging current.

Fig. 6. SoC estimator architecture.

Consequently, the SoC estimator (Fig.6) will have as inputs the terminal voltage of a battery and the discharging current and as output the estimated remaining capacity.

Real learning data for the experiment were acquired from a LiPo Turnigy Nano-Tech battery, having the following characteristics:

- Nominal Capacity: 300 mAh
- Nominal Voltage: 3.7V - one LiPo cell
- Maximum discharge rate: 35C
- Maximum charging rate: 15C

Data acquisition experiment was done using an intelligent battery charger, Thunder T6 Balance Charger / Discharger (Fig.7).

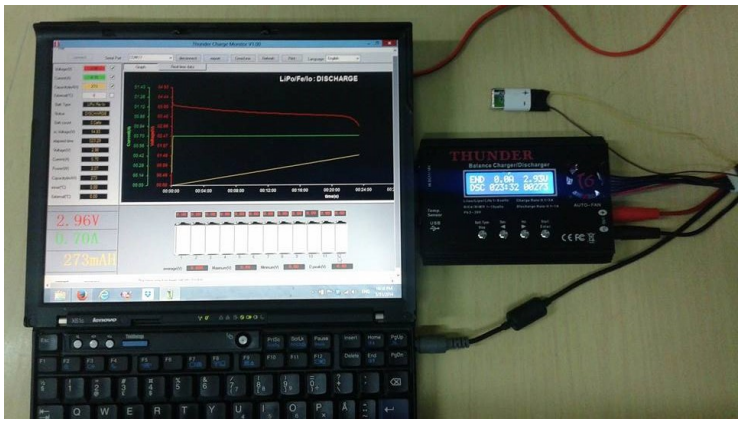

Fig. 7. Data acquisition experiment.

The experiment consisted of 6 total battery discharges made at 6 different discharge currents: 0.5A (1.66C), 0.6A (2C), 0.7A (2.33C), 0.8A (2.66C), 0.9A (3C) and 1A (3.33C). This data is used for learning. After finishing this set of experiments, another discharge was made (at 0.7A) to acquire data for model validation.

Discharge time is highly influenced by the magnitude of the discharge current (Fig. 8). The bigger the discharge current, the faster the battery discharges.

Remaining capacity decreases linearly because the discharge current is constant during an experiment (Fig.9).

The correlation between the terminal voltage and the remaining capacity is highly nonlinear (Fig.10). This set of functions is exactly the estimation the GNGA have to learn.

6. Experimental Results

For the experiment we have used Matlab environment in which we processed the data and developed the GNGA, the learning algorithm and the testing. For

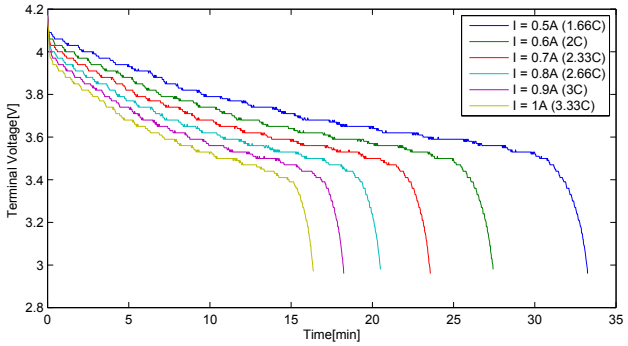

Fig. 8. Terminal voltage variation during different current discharges.

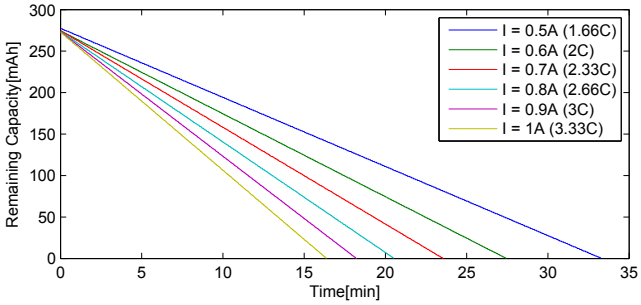

Fig. 9. Remaining capacity variation during different current discharges.

the Genetic Algorithm we have used GAOT (Genetic Algorithm Optimization Toolbox) [11] which is also implemented in Matlab.

As we saw in the previous sections, the parameters of the GN are the weights (summation, product and final output). In our application we decided to add two more parameters in order to give more flexibility to the model. We also tune and learn using the GA the parameters λ_s and λ_p from the activation of the two neurons (4), (5).

Consequently, in our application where we have two inputs in the GNGA (terminal voltage and discharge current) and one output (remaining capacity) the model will have the following parameters:

- $w_{\Sigma 1}$ and $w_{\Sigma 2}$ - input weights for summation neuron
- $x_{0\Sigma}$ - bias for summation neuron
- λ_s - sigmoid activation parameter
- $w_{\Pi 1}$ and $w_{\Pi 2}$ - input weights for product neuron

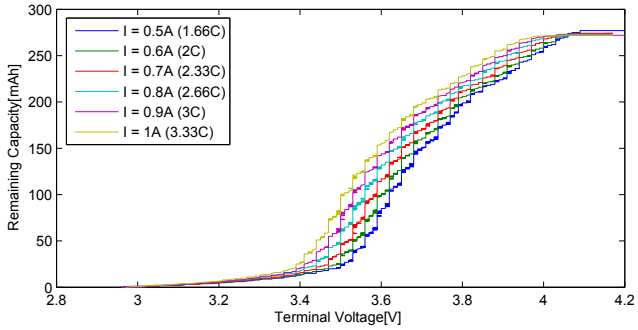

Fig. 10. Terminal voltage and remaining capacity correlation functions.

- $x_{0\Pi}$ - bias for product neuron
- λ_p - gaussian activation parameter
- w - final output weight (fuzzy compensatory operator)

The complexity of the model is proportional to the number of parameters which is $N = 9$.

As stated before, the experimental data acquired during the discharges of the LiPo battery was divided in two sets: a learning set containing 5633 in-out samples made by putting together the data from all the six discharges (Fig.11); a validation set containing 950 in-out samples (Fig.12). The validation data was necessary for testing the capacity of the model to generalize the behavior it learned.

In the original GN paper [4] it was advised that for maximum learning performance the training set should be scaled in $[0, 0.9]$ interval. Consequently, in our application we also scaled the data accordingly to the plots (Fig.11, Fig.12)

For the GA training we have tested more learn meta-parameters in order to find the optimal ones. We achieved good results using:

- search bounds: $[-5, 5]$
- population size: 30
- iterations: 2000
- select function: Normalized Geometric Select with 0.3 select probability
- crossover function: Arithmetic Crossover with 4 crossover points
- mutation function: Uniform Mutation with 8 individuals

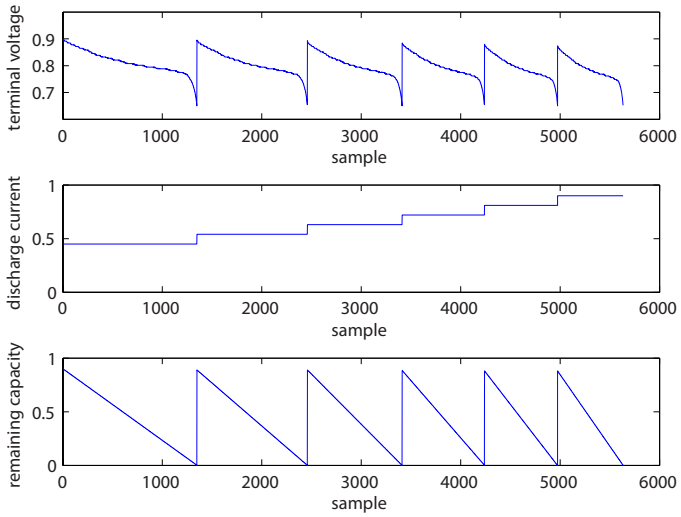

Fig. 11. Train data used in GNGA for the SoC estimator.

Table 1. GNGA parameters training result

$w_{\Sigma 1}$	$w_{\Sigma 2}$	$x_{0\Sigma}$	λ_s	$w_{\Pi 1}$	$w_{\Pi 2}$	$x_{0\Pi}$	λ_p	w
4.6929	0.1058	-3.8644	4.9983	-2.2429	-3.7757	-1.1953	0.04980	1.1602

A complete explanation for these meta-parameters and how GAOT works could be found in [11]. The algorithm ran for about 1.5 h finding the following parameters:

Choosing optimal meta-parameters is an application oriented problem. The approach should be trial and error, the parameters are chosen after several tests but is possible that with time, the user will build knowledge and expertise. Thus, the user could be able to chose the right meta-parameters based on the application specificities with the help of his knowledge.

The result of the learning could be found in Fig.13 where the GNGA successfully maps the data achieving an average absolute error of about 3.9%.

The test using the validation data shows that the GNGA has very good generalization capability. Although the GNGA was learned using different data, it achieves even better average results on new data (Fig.14). On the validation data the GNGA achieves 3.7% absolute average error.

The results for the GNGA were compared to an artificial neural network. The ANN classification was done using Matlab neural network toolbox and the same learning set as previously. The network was configured to have 2 input neurons (for each model input), 20 hidden neurons for good nonlinear classification and one output neuron.

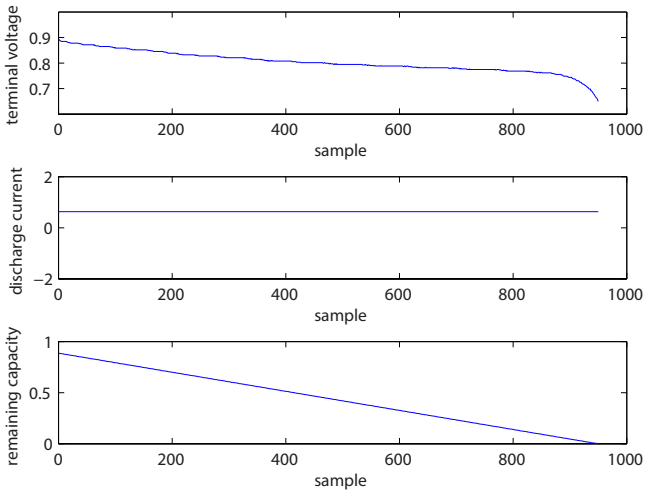

Fig. 12. Validation data used for testing the capacity of the GNGA model to generalize.

The performance achieved by the ANN over the training set are presented in Fig.15. The first plot represents how the ANN estimates the remaining capacity over the real one. The second plot is a comparison between GNGA and ANN for the absolute estimation error. The estimation precision of the ANN is around three times better than the GNGA, achieving an absolute average error of 0.73%. This indeed is a weakness of the GNGA, but we have to take in account that the ANN in nine times more complex than it (having $N = 81$ parameters). The complexity of a model increases its precision but entails implementation constraints which in real-time and low energy applications are critical.

The same comparison was made over the validation set (Fig.16). Here also the absolute average error of the ANN is better but the generalization capability is weaker because, comparing with the GNGA, the absolute average error increases a little.

7. Conclusions

The paper presents new contribution in developing a hybrid intelligent methodology by using Genetic Algorithm and Generalized Neuron.

Some experimental results prove the efficiency of this new GNGA methodology applied in the estimation of State of Charge for an electric battery. The simplicity of the architecture and the small number of parameters which must be tuned recommend this new methodology to solve some real-time complex problems.

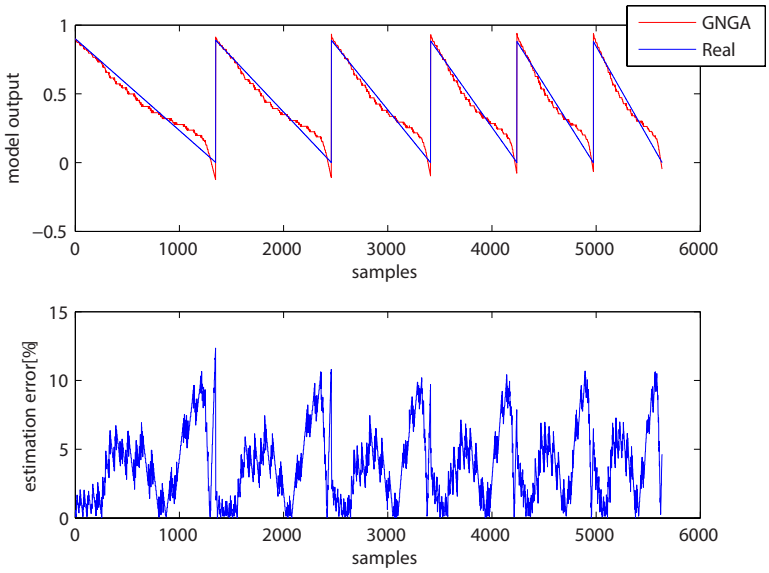

Fig. 13. GNGA learning result. Comparison and error between the estimated and the real remaining capacity after the learning.

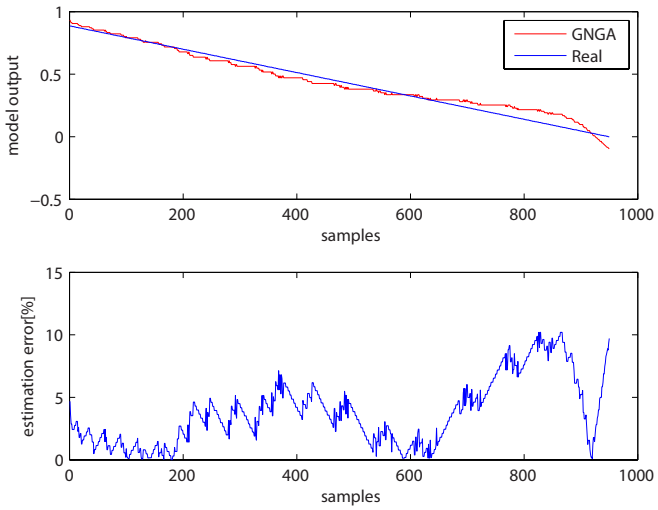

Fig. 14. GNGA learning result. Comparison and error between the estimated and the real remaining capacity using the validation data.

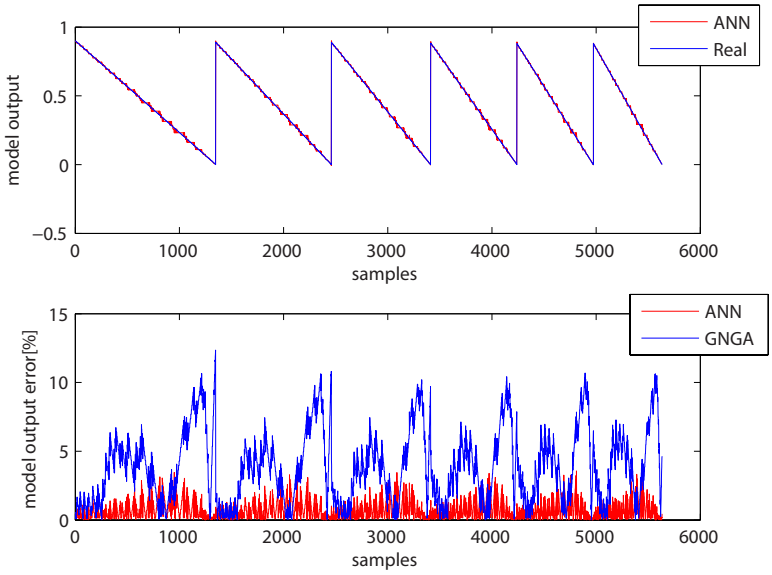

Fig. 15. Learning results comparison between GNGA and ANN.

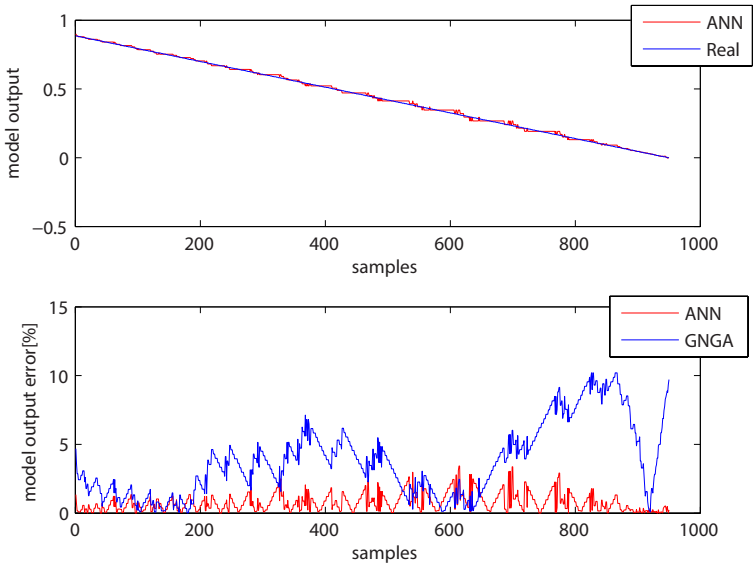

Fig. 16. Performance comparison between GNGA and ANN for the validation set.

References

[1] Ajith Abraham. *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, chapter Neuro Fuzzy Systems: State-of-the-Art Modeling Techniques, pages 269–276. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[2] C.H. Cai, D. Du, and Z.Y. Liu. Battery state-of-charge (soc) estimation using adaptive neuro-fuzzy inference system (anfis). In *Fuzzy Systems, 2003. FUZZ '03. The 12th IEEE International Conference on*, volume 2, pages 1068–1073 vol.2, May 2003.

[3] P. A. Castillo, M. G. Arenas, J. J. Castillo-Valdivieso, J. J. Merelo, A. Prieto, and G. Romero. *Advances in Soft Computing: Engineering Design and Manufacturing*, chapter Artificial Neural Networks Design using Evolutionary Algorithms, pages 43–52. Springer London, London, 2003.

[4] D. K. Chaturvedi. *Soft Computing*, volume 103 of *Studies in Computational Intelligence*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[5] D.K. Chaturvedi and O.P. Malik. Generalized neuron-based PSS and adaptive PSS. *Control Engineering Practice*, 13(12):1507 – 1514, 2005. Special Section on Power Plants and Power Systems ControlSpecial Section on Power Plants and Power Systems Control.

[6] K. D. Chaturvedi, R. Chauhan, and K. P. Kalra. Application of generalised neural network for aircraft landing control system. *Soft Computing*, 6(6):441–448.

[7] K. D. Chaturvedi, M. Mohan, K. R. Singh, and K. P. Kalra. Improved generalized neuron model for short-term load forecasting. *Soft Computing*, 8(1):10–18, 2003.

[8] N.A. Chaturvedi, R. Klein, J. Christensen, J. Ahmed, and A. Kojic. Algorithms for advanced battery-management systems. *Control Systems, IEEE*, 30(3):49–68, June 2010.

[9] M. Dragoicea, I. Dumitrache, and D.S.Cuculescu. Multi-behavioral model based autonomous navigation of the mobile robots. *International Journal Automation Austria*, 11(1):1–20, 2003.

[10] I. Dumitrache. Intelligent autonomous systems. *Revue Roumaine des Sciences Techniques, Bucharest*, (3):439–454, 2000.

[11] Christopher R. Houck, Jeffery A. Joines, and Michael G. Kay. A genetic algorithm for function optimization: A matlab implementation. *NCSU-IE TR 95*, (9), 1995.

[12] I.Dumitrache and M. Dragoicea. Intelligent techniques for cognitive mobile robots. *Journal of Control Engineering and Applied Informatics*, 5(2):3–8, 2004.

[13] E. Kim, Jinkyu Lee, and K.G. Shin. Real-time prediction of battery power requirements for electric vehicles. In *Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on*, pages 11–20, April 2013.

[14] Masaharu Mizumoto. Pictorial representations of fuzzy connectives, Part I: Cases of t-norms, t-conorms and averaging operators. *Fuzzy Sets and Systems*, 31(2):217–242, 1989.

[15] Masaharu Mizumoto. Pictorial representations of fuzzy connectives, Part II: Cases of compensatory operators and self-dual operators. *Fuzzy Sets and Systems*, 32(1):45–79, August 1989.

[16] Kumar Sanjeev and Chaturvedi D.K. Prediction of interest rate using generalized neural method. *International Journal of Computer Information Systems and Industrial Management Applications*, 3(1):738–745, 2011.

[17] Wu T., Wang M., Xiao Q., and Wang X. The soc estimation of power li-ion battery based on anfis model. *Smart Grid and Renewable Energy*, 3(1):51–55, 2012. doi: 10.4236/sgre.2012.31007.

[18] Kwo Young, Caisheng Wang, Le Yi Wang, and Kai Strunz. Electric vehicle battery technologies. In Rodrigo Garcia-Valle and Joo A. Peas Lopes, editors, *Electric Vehicle Integration into Modern Power Networks*, New York, 2013. Springer-Verlag.

[19] Lee Yuang-Shung, Kuo Tsung-Yuan, and Wang Wei-Yen. Fuzzy neural network genetic approach to design the soc estimator for battery powered electric scooter. In *Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, volume 4, pages 2759–2765 Vol.4, 2004.