A Gene-Regulated Nested Neural Network

Romi Rahmat[1], Muhammad Pasha[2], Mohammad Syukur[3], and Rahmat Budiarto[4]
[1]Faculty of Computer Science and Information Technology, University of Sumatera Utara, Indonesia
[2]School of Computer Sciences, University Sains Malaysia, Malaysia
[3]Faculty of Mathematics and Natural Sciences, University of Sumatera Utara, Indonesia
[4]College of Computer Science and Information Technology, Albaha University, Saudi Arabia

Abstract: *Neural networks have always been a popular approach for intelligent machine development and knowledge discovery. Although, reports have featured successful neural network implementations, problems still exists with this approach, particularly its excessive training time. In this paper, we propose a Gene-Regulated Nested Neural Network (GRNNN) model as an improvement to existing neural network models to solve the excessive training time problem. We use a Gene Regulatory Training Engine (GRTE) to control and distribute the genes that regulate the proposed nested neural network. The proposed GRNNN is evaluated and validated through experiments to classify accurately the 8bit XOR parity problem. Experimental results show that the proposed model does not require excessive training time and meets the required objectives.*

Keywords: *Neural networks, gene regulatory network, artificial intelligence, bio-inspired computing.*

Received May 13, 2013; accepted July 21, 2013; published online December 3, 2014

1. Introduction

Biologically inspired computing has been an active area of research for the past decades. This approach is implemented with the first wave of human neuron-connection modeling, which is also called a neural network. The wave continues with the learning theory introduced by [7], which is known as hebbian learning and the introduction of the Genetic Algorithm (GA) [5]. This second wave is currently being developed with increasingly complex models, such as self-organizing maps [10], recurrent neural networks [12] and spiking neural networks [4].

The problems in neural network models are mostly related to the intensity and quantity of input data. The massive input data that flow to the input layer in a neural network prolong training times and cause training malfunctions. A nested neural network model is proposed to address this problem by using data input partitions, which are distributed among several neural networks in the nested network architecture. Several studies have experimented with and tested nested neural networks in a number of applications such as image compression [11], hierarchical cluster model, which has been described as a parallel neural network in the neocortex model of the brain and others [2, 13, 18, 19, 20]. In this work, we aim to improve the architecture and efficiency of the nested neural network model by incorporating genetic properties inspired by the Gene Regulatory Network (GRN) of humans to develop a more dynamic structure.

The modeling of human GRN from a biological perspective to a computational perspective is an ongoing effort started by researchers from the bioinformatics group of INRIA Rhône-Alpes [3]. Several models have been produced to represent GRN with different development methods, including simple and imprecise GRN modeling by using kauffman boolean network with boolean vectors for genes state [1], extensive representation by using bayesian and regression networks with transitional probabilities [9], artificial hopfield-type neural networks as the core principle of GRN computational model development [14], as well as others [8, 16, 17].

We have proposed the Gene Regulatory Training Engine (GRTE) in our previous work to experiment with the proben1 benchmark dataset as uncorrelated data [6]. We modified the GRTE further to solve a crystallography data analysis problem [15]. In the current paper, we propose a Gene-Regulated Nested Neural Network (GRNNN) model that uses an extended GRTE with the ability to evolve according to the structure of the partitioned correlated input data.

The remainder of this paper is organized as follows: Section 2 presents our proposed GRNNN. Section 3 describes the extended GRTE. Section 4 shows the implementation and experimental set-up for the evaluation of the proposed model. Section 5 displays the results and discussion. Section 6 concludes and discusses the direction of future works.

2. GRNNN

Correlated data can be described as a set of partitioned data that is related to each other. In this section, we introduce GRNNN, which expedites the training and analysis of correlated input data. In the proposed GRNNN architecture, neural networks form a sub-

network where the output of one network is the input of the next network. The structure of this sub-network depends on its gene properties. When we group neural networks into a nested neural network with several layers Φ, where $\Phi=\{\Phi_1, \Phi_2, ..., \Phi_k\}$, the number of GRNNN layers depend on the number of partitioned input and can be defined as follows:

$$k = \frac{n}{2} - 2^{(k-1)} \quad (1)$$

Where k is the number of layers Φ that occur in the Outer Neural Network (ONN) and n is the number of input partitions.

Figure 1 shows the general architecture of the GRNNN model, which consists of several sub-network layers that are referred to as the Inner Neural Network (INN). These sub-network layers connect with each other to form the main network, that is, the ONN. By using the number of ONN layers k, the number of neuro genes N_{NG} for the GRNNN can be calculated as follows:

$$N_{NG} = \sum_{i=0}^{k} 2^i \quad (2)$$

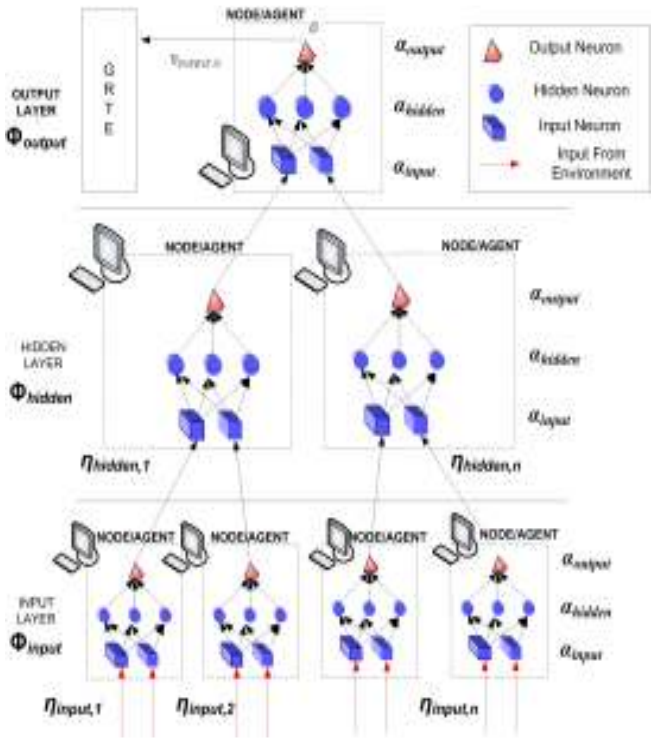


Figure 1. GRNNN model architecture.

Before presenting the GRNNN algorithm, let us briefly define some notations used in the algorithm:

Φ_k: Layers of ONN where k is the number of ONN layers.
η_h: INN, where h is the number of neural networks.
α_m: Layers of the inner neural network, where m is the number of layers in INN.
μ_p: Neurons where p is the number of neurons in the INN layers.
w_{ij}: Weight connection from neuron i to j.
σ_{ij}: Activation state of every neuron.
O: Final output.

- *Step* 1: The number of outer layers $\Phi=\{\Phi_1, \Phi_2, ..., \Phi_k\}$ is initialized by using Equation 1, inner layers $\alpha_=\{\alpha_1, \alpha_2, \ldots, \alpha_m\}$, ONN neurons $\eta_h=\{\eta_1, \eta_2, ..., \eta_h\}$, neurons within INNs μ_i and weight connections ω_{ij}.
- *Step* 2: By using the initialized ONN and INN structure $\Phi_1= \{\eta_{1,1}, \eta_{1,2}, \ldots, \eta_{1,p}\}$, $\Phi_2= \{\eta_{2,1}, \eta_{2,2}, \ldots, \eta_{2,p}\}$ and $\Phi_k= \{\eta_{k,1}, \eta_{k,2}, ..., \eta_{k,p}\}$, neurons $\{\mu_{layer,1}, \mu_{layer,2}, \ldots, \mu_{layer,p}\}$ are constructed for every neural network $\eta_{(layer, variable)}$ where every layer is $\{\alpha_1, \alpha_2, \ldots, \alpha_{m+p}\}$ and weight connection is $\{\omega_{11}, \omega_{12}, \omega_{21}, \ldots, \omega_{ij}\}$.
- *Step* 3: The number of node/agent is created based on Equation 2.
- *Step* 4: The GRNNN architecture is set based on the given gene. Based on Equation 1, we create the number of INN in every ONN by using the following Equation:

$$\Phi_j = 2^{(k-j)}, \; where \; k \in N, \; j = \{1, 2, 3, ..., k\} \quad (3)$$

- *Step* 5: The number of input neurons in the ONN input layer μ_α is expressed as follows:

$$\mu_a = \frac{\#input(n)}{\#inputlayernodes(\Phi_1)} \quad (4)$$

- *Step* 6: The number of input neurons in the hidden layer Φ_2 is set with respect to the output neurons from the previous layer Φ_1 and the number of input neurons in the output layer Φ_k.is set with respect to the output neurons from the hidden layer Φ_1.

3. GRTE

The proposed GRNNN uses an extension of the GRTE [18] to create and manage gene-like properties. The extended GRTE uses three different sets of gene, which determines the structure and parameters of the GRNNN.

- The First Gene: Is the core GRNNN gene that defines the neural network training algorithm that is used in the GRNNN and the number of nodes involved in the parallelization process.

Number of Node { 1, 2 , … , *n*}	Neural Network Training Algorithm {*MLP, BPNN*, etc.,}

- The Second Gene: Is the gene variable of the INN. This gene contains the variable of the neural network parameters for every node in the layer. These parameters include the number of neurons, input nodes, hidden nodes and output nodes, as well as the epoch learning and momentum rates.

Number of Input Neurons	Number of Hidden Neurons	Number of Output Neurons	Number of Hidden Layer	Epoch	Learning Rate	Momentum Rate

- The Last Gene: Is the gene variable of the ONN. This gene contains the parameters for the construction of the ONN, including the number of nodes, inputs, hidden neurons and output neurons.

Before we describe the fitness function, we first have to introduce the objective function. The objective function is basically used to verify and measure the objectivity of a neural network. We find that the use of the root-mean-square error is suitable for our architecture (i.e., multi-layer feed-forward networks). The root-mean-square error can be formulated as follows:

$$E^p = \sqrt{\frac{1}{N_o}\sum_{o=1}^{N_o}(d_o^p - y_o^p)^2} \quad (5)$$

Where E^p represents the root of the difference between the desired output value d_o^p for unit o with a particular pattern p and actual network output y_o^p for every training sample, in which N_o is the number of output units.

Every gene has its own objective function; that is, we can obtain the average objective function E_{AVG} for all genes by the following Equation:

$$E_{AVG} = \frac{\sum_{i=0}^{N_{NG}} E_p}{N_{NG}} \quad (6)$$

Equation 6 can be calculated after GRTE training. After the calculation, a different data sample will be trained in each partition data. For example, the data sample in the neural network for the input layer can be bigger than the data in the hidden or output layer. Once the GRTE sends the gene to the environment, the GRNNN produces the final output O_{CD}. The final output O_{CD} is used to measure the error with the expected output E_o. Thus, we can obtain the fitness of the output F_o by the following:

$$F_o = \left|E_o - O_{CD}\right| \quad (7)$$

We obtain the fitness function in one generation of GRTE for GRNNN from Equations 5, 6 and 7. The fitness function is expressed as follows:

$$F_{CD} = E_{AVG} \cdot F_o = \frac{\sum_{i=0}^{N_{NG}} E_p}{N_{NG}} \cdot \left|E_o - O_{CD}\right| = \frac{\sum_{i=0}^{N_{NG}} \sqrt{\frac{1}{N_o}\sum_{o=1}^{N_o}(d_o^p - y_o^p)^2}}{N_{NG}} \cdot \left|E_o - O_{CD}\right| \quad (8)$$

All GRTE variables have different functions and purposes in GRNNN construction. Thus, we can perform the optimization for these variables. The optimization is conducted by applying a GA in GRTE. Before we proceed to the GA inside the GRTE, we have to describe the mutation regulatory.

After we analyze the gene representation of the GRTE, the genetic operator that can occur in this representation is only a mutation operator; thus, no crossover and parent selection occurs (i.e., every gene is a single individual that has to be mutated with mutation regulatory). Mutation regulatory consists of algorithms and equations to control the mutation of a gene. We aim to mutate several entities. The algorithm and equations are provided below.

The mutation regulatory for input neurons is related to the data variance to be used. Suppose that, we want to apply the GRNNN input data in n-bit parity problem, that is we have to have n inputs. Assume that we need a multi-layer GRNNN; n has to be $2 \leq n \leq \infty$. Based on Equation 3, we can denote the number of inputs NI as follows:

$$NI = \Phi_1 \quad (9)$$

Mutation regulatory for hidden neurons is quite different from the mutation regulatory for input neurons. The sequence of Algorithms 1 shows that this algorithm adapts to search for the best gene by mutating the hidden layer and the neurons inside the layer. The mutation regulatory for the proposed GRNNN can be described in the following algorithm:

Algorithm 1: Hidden neurons mutation regulatory.

1. initialize(*NH*=2) *and* (*Epoch*=10).
2. mutate_hidden_neuron
  NH=*NH*+(2.(*Random*(4)))
3. mutate_epoch
  (*Epoch*=*Epoch*+(10.(*Random*(4)))
4. begin If (E_p-E_{p-1}|<0.1)
5. *stop_hidden_neuron_mutation;*
6. *mutate_epoch*
  (*Epoch*=*Epoch*+(100.(*Random*(30)))
7. End If

Another factor in optimizing the objective/fitness value is mutation's epoch. We can combine both hidden neurons *NH* and epoch as one equation for GRNNN because of the nature of the partitioned data. The equations are as follows:

$$Epoch = NH \cdot 10 \text{ and } NH = \frac{Epoch}{10} \quad (10)$$

Given that, we have different data partitions, the appropriate number of output neurons *NO* for GRNNN depends on the data and should be *NO*>1. For GRNNN, the number of output neurons is *NO*=1. For rate and momentum, the number can also be mutated. However, for simplicity, we set a fixed number with rate≥0.5 and momentum≥0.7 for both GRNNN modes. Algorithm 2 is an adaptive algorithm with fitness function F_{CD} or F_{UD} as termination criterion. GRTE is also involved in the training, transmission and retrieval of GRNNN results.

GRTE employs a simple GA with a mutation operator. The Algorithm 2 shows how GRTE adapts the GA algorithm from Lines 8 to 14. Our GRTE differs from the common GA because it does not use crossover and parent selection (every gene act as a single parent) and random population initialization does not occur in the algorithm. We present the complete diagram of the GRTE model in Figure 2.

Algorithm 2: GRTE algorithm.

1. intializeGene( ).
2. while not(F_{CD}≤ 0.030) || (F_{UD}≤0.030) *do*
3. trainingGene( )
4. send_neural_network( )
5. execute_neural_network( )
6. retrieve_result( )

7. evaluate_fitness_function for GRNNN( )
8. while(0.01≤ *Ep*≤ 0.98) *do*
9. execute_mutation_regulatory( )
10. evaluate_fitness_function for GRNNN( )
11. end while
12. create_new_solution( )
13. send_and_compare_to_population( )
14 choose_best_solution( )
15. end while

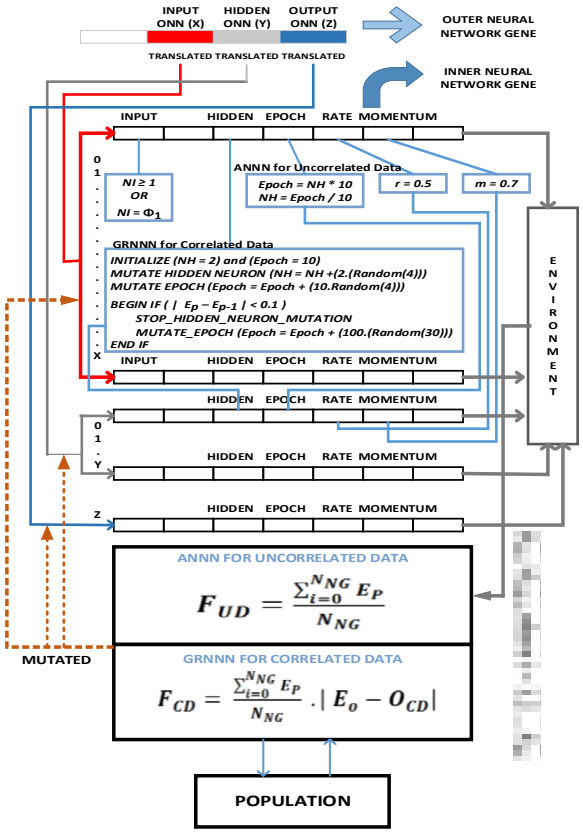


Figure 2. GRTE extension model to cover the proposed GRNNN.

The fitness function and mutation regulatory is an important GRTE element. These two methods support the proposed GRNNN and allow the extended GRTE model to become more adaptive. Figure 2 describes the positions of these two methods in GRTE. The ONN should be translated first to construct the INN. The genes produced by the INN have to be trained before the genes are sent to the environment. Results from the environment will be analyzed by using the fitness function and placed in the population. Given that the fitness function does not affect the mutation in the ONN level, the mutation process will be implemented in the INN. The rule of mutation will be conducted by mutation regulatory to control the objectivity of the gene. The GRTE will then create new genes for training that will be sent to the environment. This iteration will stop after the fitness function reaches the termination point.

4. Implementation and Validation

From the application perspective, we can divide our methodology into client/agent application and server application. Three different modules exist in the agent application as shown in Figure 3. First, a data collector is purposed to collect data from benchmark datasets. This data should be partitioned. Second, an input analyzer is used to analyze and classify the data. The input analyzer can also be used as a pre-processing data module. Finally, the neural network engine of the agent is used to run the neuro-gene engine that has been injected to the node by using the dataset as input resources.

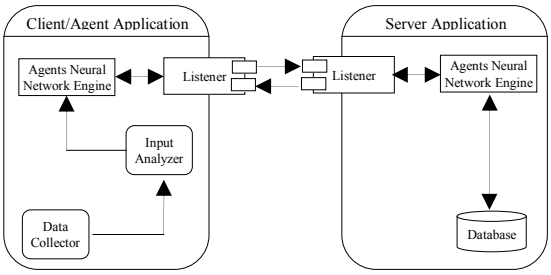


Figure 3. Client/agent-server applications.

The server has one module and one database. The module is the GRN engine, which implements the GRTE to create the neuro gene and acts as the home base of neural network training and GA. The GRN engine is connected to the database to record gene activity and population.

The process flow in the environment is as follows. First, the agents are activated. Thereafter, an agent will request neuro genes to the server. The server then provides the neuro gene. Every agent should start the neural network activity inside the node and the architecture will be built automatically inside the environment based on the given neuro genes. Every agent will record the information of their activities for later use in the GRTE as the parameters for performance measurements.

The network acts as the pre-elementary requirement of our model in setting up several computers that will act as agents. The number of required computers depends on the number of neurons in the ONN.

4.1. XOR Parity Problem Data Set-up

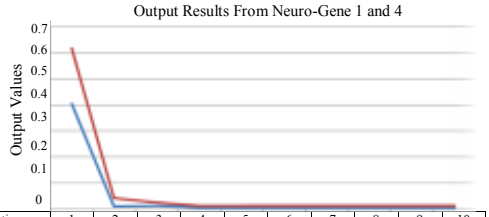
We design the first experiment to test the proposed GRNNN model by using an 8bit XOR parity problem. We can define the number of input in one particular INN as 8/4= 2 by using Equation 1. Therefore, we have to divide eight inputs into two separate inputs. The diagram in Figure 1 shows how the inputs are fed to the GRNNN. We train seven datasets for this parity problem experiment. However, we can minimize the agents into four agents only (agent in input layer) because these particular agents can also act as agents for the hidden and output layers of the ONN. Another important thing in this data experiment is the creation of a truth table for the 8bit parity problem, which consists of 256 data samples.

5. Analysis and Results

The XOR parity problem can be categorized as a linearly separable problem. The 8bit XOR parity dataset contains 256 samples, whereas the eight inputs

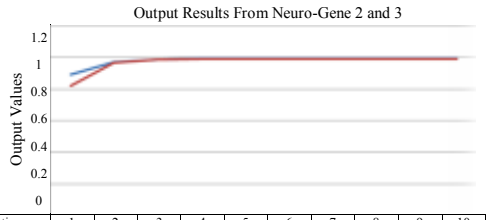
of the datasets are divided by four neuro genes. Thus, one neuro gene in the input layer receives two inputs only. In this section, we report the results obtained when experimenting our proposed method with the 8bit XOR parity problem.

In this part, we present the result obtained from GRNNN in the environment. We use four agents to perform GRNNN calculations, which produce seven neuro genes. We also use the default value, that is, (0, 0, 1, 0, 0, 1, 1 and 1), as the 8bit input. The expected result of every 2bit input is based on the 4bit truth table. In Figures 4, 5 and 6, we present the output result from the neuro genes in the input and hidden layers. The output results are achieved from GRNNN after 10 generations.



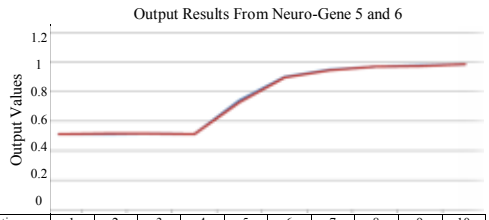
Generations	1	2	3	4	5	6	7	8	9	10
Input (0, 0) A	0.4041	0.0119	0.0137	0.0067	0.0067	0.0067	0.0067	0.0067	0.0067	0.0067
Input (1, 1) D	0.6179	0.0424	0.0253	0.0131	0.0131	0.0131	0.0131	0.0131	0.0131	0.0131

Figure 4. GRNNN results for the first and third neuro-gene in input layer of outer neural network.



Generations	1	2	3	4	5	6	7	8	9	10
Input (1, 0) B	0.8931	0.9694	0.9858	0.9925	0.9925	0.9925	0.9925	0.9925	0.9925	0.9925
Input (0, 1) C	0.8201	0.9657	0.9854	0.9911	0.9911	0.9911	0.9911	0.9911	0.9911	0.9911

Figure 5. GRNNN results for the second and fourth neuro-gene in input layer of outer neural network.

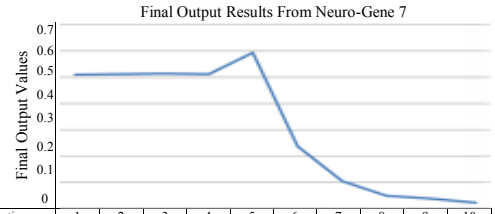


Generations	1	2	3	4	5	6	7	8	9	10
Input (A, B)	0.5112	0.5133	0.5142	0.5121	0.7416	0.9009	0.9489	0.9697	0.9786	0.9855
Input (C, D)	0.5106	0.5141	0.5159	0.5135	0.7271	0.8953	0.9458	0.9689	0.9748	0.9847

Figure 6. GRNNN results for the first and second neuro-gene in hidden layer of outer neural network.

Figure 4 shows the result from neuro genes 1 and 4 with (0, 0) and (1, 1) as inputs, respectively. Figure 5 shows the result from neuro genes 2 and 3 with (1, 0) and (0, 1) as inputs, respectively. After 10 generations, the GRNNN exhibits the best result in the fourth generation.

Once the input layer of the ONN finishes the recognition, the outputs of the four neuro genes have to pass the next hidden layers directly. Figure 6 shows the result of the input from the output of the previous result in Figures 4 and 5. The first neuro gene in the hidden layer receives inputs from neuro genes 1 and 2, whereas the second neuro gene in the hidden layer receives input from neuro genes 3 and 4 Figure 7.

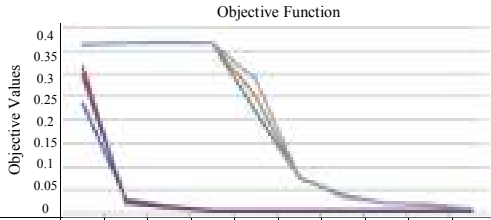


Generations	1	2	3	4	5	6	7	8	9	10
Final Output	0.5109	0.5137	0.5151	0.5129	0.5939	0.2361	0.1045	0.0486	0.0364	0.0212

Figure 7. Final output results from the last node of output layer.

The next step is to pass the outputs from the two neuro genes in the ONN hidden layer to act as inputs for the last neuro gene in the output layer. Figure 7 shows the final result O_{CD} of the whole GRNNN in every generation. Our GRNNN provides the best result in the 10th generation, where it reaches O_{CD}= 0.021. This final output is important in the fitness function of the GRTE.

Figure 8 describes the objective value obtained by using the objective function E^P in Equation 4. E^P should be lower in every generation to prove that our neural network evolves properly. The four neuro genes in the input layer obtain the optimum and minimum errors. For example, the first generation E^P of all neural networks in the input layer reaches ±0.3. When the neural network mutates, E^P becomes ±0.03 in the second generation. All neural networks in the input layer reaches convergence from the fourth generation.



Generations	1	2	3	4	5	6	7	8	9	10
Input Layer 1	0.2365	0.0294	0.0152	0.006	0.006	0.006	0.006	0.006	0.006	0.006
Input Layer 2	0.3156	0.0241	0.0106	0.0054	0.0054	0.0054	0.0054	0.0054	0.0054	0.0054
Input Layer 3	0.2964	0.0267	0.0135	0.0069	0.0069	0.0069	0.0069	0.0069	0.0069	0.0069
Input Layer 4	0.2929	0.0248	0.0148	0.0071	0.0071	0.0071	0.0071	0.0071	0.0071	0.0071
Hidden Layer 1	03633	0.3658	0.3678	0.3657	0.2189	0.0776	0.0394	0.022	0.0169	0.0105
Hidden Layer 2	0.3635	0.3656	0.3675	0.3657	0.2517	0.0749	0.0381	0.0211	0.0162	0.0101

Figure 8. GRTE objective values E^P for GRNNN.

Thus, for the hidden and output layers, we can see that the movement of the objective function changes in the fourth generation. The movement of the objective function is static from the first to third generation. For example, the E^P of all neural networks in the input layer reaches ±0.35 in the fourth generation. E^P becomes ±0.07 in the sixth generation when the neural network mutates. All neural networks in the hidden and output layers reach convergence starting from the sixth generation. Finally, the entire neural networks converge in the 10th generation.

The objective function also shows the accuracy of our neural network. The accuracy is measured by the correctness percentage of the objective values of the final neural networks in the 10th generation. We verify the accuracy in Table 1.

Table 1. GRNNN accuracy.

	Generations							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**AVG**
Accuracy (%)	99.4	99.5	99.4	99.3	99	99	99	99.2

In Figures 9 and 10, we present the results of the hidden neuron and epoch in every generation to show the adaptive architecture created by our proposed method. Based on our mutation regulatory, the hidden neurons and epoch are the entities that need to be mutated. Figure 9 shows the result of the hidden neuron and epoch mutations. The number of hidden neurons and epochs adapt for each generation. For example, in the third generation, hidden neuron= 6 and epoch= 40. In the fourth generation, hidden neuron= 10 and epoch= 100. The mutation of hidden neurons stops in the fourth generation because it is inappropriate, whereas the epoch mutation continuously evolves.

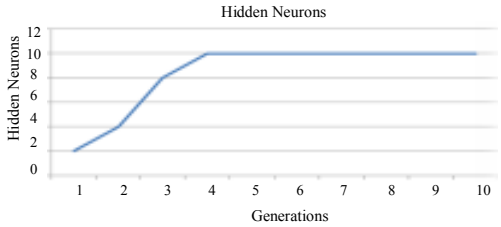


Figure 9. Hidden neurons in GRNNN.

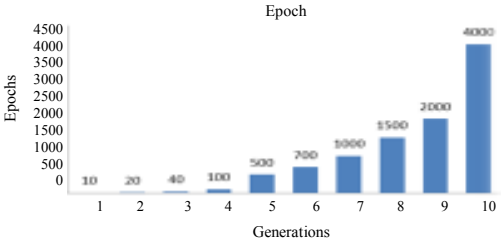


Figure 10. Epochs in GRNNN.

In Figure 11, it clearly shows how mutation regulatory provides a rule in the GRTE flows. The blue line shows that the mutation only occurs in the first four generations and the mutation at the remaining generation never occurs again. This observation can be attributed to the termination criteria ($0.01 \leq Ep \leq 0.98$) for the mutation regulatory, which is created to minimize the GRTE workload. The red line shows the mutation change of the hidden and output layers, which continuously evolve until it meets the termination criteria.

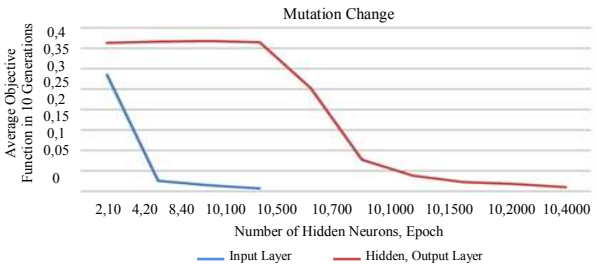


Figure 11. Mutation change result from mutation regulatory method in GRTE.

We perform sequential training (without agents) and parallel training (with agents) to measure the training speed of our proposed GRNNN. Figure 12 shows the result of the training time in sequential and parallel modes (by using agent) in 10 generations. The agent based parallel implementation of the GRNNN significantly improves the training speed.

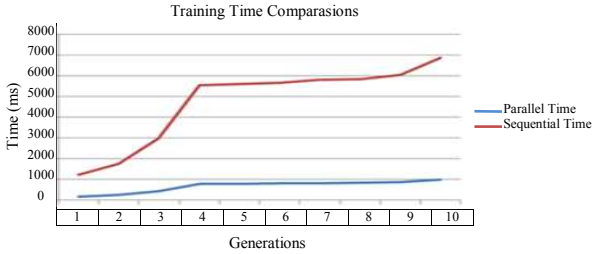
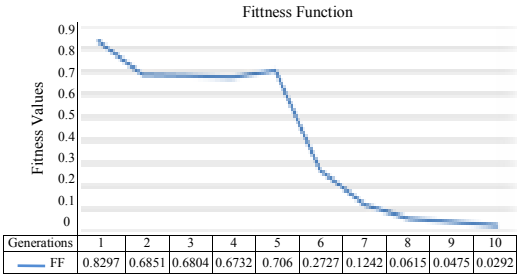


Figure 12. Training time comparisons for xor parity problem.

The last result described in Figure 13 is the fitness function, which is retrieved from GRTE based on Equation 7. The fitness function is the main termination criterion of GRTE for its evolving algorithm. Figure 13 clearly shows that the evolution is terminated when the fitness function F_{CD} reaches≤ 0.030.



Generations	1	2	3	4	5	6	7	8	9	10
FF	0.8297	0.6851	0.6804	0.6732	0.706	0.2727	0.1242	0.0615	0.0475	0.0292

Figure 13. Fitness function results for GRNNN.

6. Conclusions

We have proposed the GRNNN by using an extended GRTE as the adaptive controller and an engine to distribute neuro genes in the execution environment. We apply the 8bit parity problem dataset to validate our proposed approach. The results show a high accuracy of 99% for the classification results as well as showcasing GRNNN ability to evolve its structure for optimum results. Our results also show that the proposed GRNNN, which has the ability to obtain each network trained in parallel, does not require excessive training time compared with the sequential training of normal neural networks. Future works include applying GRNNN to solve computationally complex scientific problems with large datasets while enhancing its algorithm further.

Acknowledgement

This work was supported in part by Universiti Sains Malaysia Research University (RU) grant no. 1001/PKOMP/817047. The authors also would like to

acknowledge the cross collaboration between Universiti Sains Malaysia and University of Sumatera Utara, that have resulted in this paper.

References

[1] Akutsu T., Miyano S., and Kuhara S., "Identification of Genetic Networks from a Small Number of Gene Expression Patterns under the Boolean Network Model," available at: http://psb.stanford.edu/psb-online/proceedings/psb99/Akutsu.pdf, last visited 1999.

[2] Awad M., "Input Variable Selection using Parallel Processing of RBF Neural Networks," *the International Arab Journal of Information Technology*, vol. 7, no. 1, pp. 6-13, 2010.

[3] De H., "Modeling and Simulation of Genetic Regulatory Systems: A Literature Review," *Journal of Computational Biology*, vol. 9, no. 1, pp. 67-102, 2002.

[4] Ghosh S. and Adeli H., "Spiking Neural Networks," *the International Journal of Neural Systems*, vol. 19, no. 4, pp. 295-308, 2009.

[5] Golberg D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[6] Hasibuan A., Rahmat F., Pasha F., and Budiarto R., "Adaptive Nested Neural Network (ANNN) Based on Human Gene Regulatory Network (GRN) for Gene Knowledge Discovery Engine," *the International Journal of Computer Science and Network Security*, vol. 9, no. 6, pp. 43-54, 2009.

[7] Hebb D., *The Organization of Behavior*, John Wiley and Sons, 1949.

[8] Kasabov N., Benuskova L., and Wysoski G., "Computational Neurogenetic Modelling: Gene Network within Neural Networks," *in Proceedings of International Joint Conference on Neural Networks*, Budapest, Hungary, pp. 1203-1208, 2005.

[9] Kato M., Tsunoda T., and Takagi T., "Inferring Genetic Networks from DNA Microarray Data by Multiple Regression Analysis," available at: https://www.jsbi.org/pdfs/journal1/GIW00/GIW00F12.pdf, last visited 2000.

[10] Kohonen T., *Self-Organizing Maps*, Springer, 2001.

[11] Kumar K. and Mahalingam N., "Nested Neural Networks for Image Compression," *in Proceedings of the 10th International Conference on Global Connectivity in Energy, Computer, Communication and Control*, New Delhi, India, pp. 369-372, 1998.

[12] LukošEvičIus M. and Jaeger H., "Survey: Reservoir Computing Approaches to Recurrent Neural Network Training," *Computer Science Review*, vol. 3, no. 3, pp. 127-149, 2009.

[13] Manzhos S., Wang X., Dawes R., and Carrington T., "A Nested Molecule-Independent Neural Network Approach for High-Quality Potential Fits," *Journal of Physical Chemistry*, vol. 110, no. 16, pp. 5295-5304, 2006.

[14] Marnellos G. and Mjolsness D., Modeling *Neural Development*, MIT Press, 2003.

[15] Pasha F., Rahmat F., Budiarto R., and Syukur M., "A Distributed Autonomous Neuro-Gen Learning Engine and its Application to the Lattice Analysis of Cubic Structure Identification Problem," *the International Journal of Innovative Computing, Information and Control*, vol. 6, no. 3, pp. 1005-1022, 2010.

[16] Sehgal M., Gondal I., and Dooley L., "AFEGRN: Adaptive Fuzzy Evolutionary Gene Regulatory Network Reconstruction Framework," *in Proceedings of International Conference on Fuzzy Systems*, Vancouver, Canada, pp. 1737-1741, 2006.

[17] Vohradsky J., "Neural Network Model of Gene Expression," *Journal of the Federation of American Societies for Experimental Biology*, vol. 15, no. 3, pp. 846-854, 2001.

[18] Wu T. and Kareem A., "Modeling Hysteretic Nonlinear Behavior of Bridge Aerodynamics via Cellular Automata Nested Neural Network," *Journal of Wind Engineering and Industrial Aerodynamics,* vol. 99, no. 4, pp. 378-388, 2011.

[19] Xiong X., Wang J., Niu T., and Song Y., "A Hybrid Model for Fault Diagnosis of Complex Systems," *Electronics Optics and Control*, vol. 16, no. 2, pp. 56-59, 2009.

[20] Yap H., Guan L., and Liu W., "A Recursive Soft-Decision Approach to Blind Image Deconvolution," *IEEE Transactions on Signal Processing*, vol. 51, no. 2, pp. 515-526, 2003.

Romi Rahmat received his BCs and MSc degrees in computer science from University Sains Malaysia in 2007 and 2008 respectively. Currently, he is a lecturer in Faculty of Computer Science and Information Technology, University of Sumatera Utara, Medan, Indonesia. His research interests include image and signal processing, intelligent system, mobile computing and neuro-genetic system.

Muhammad Pasha is a research fellow at the School of Computer Sciences, Universiti Sains Malaysia. He holds PhD degree in computer science from University Sains Malaysia in 2009 in addition to membership of the IEEE. His main interests are in the area of brain inspired computing, machine intelligence, intelligent network monitoring, medical image analysis, neuro imaging, and healthcare IT.

Mohammad Syukur is a Professor at the Faculty of Mathematics and Natural Sciences, University of Sumatera Utara, Medan, Indonesia. Currently, he is the head of Crystallography and Radiation Physics Research Laboratory. His research interests include material physics, X-ray diffraction methods, artificial intelligence application and crystallography. He has written two book chapters and numerous refereed research papers.

Rahmat Budiarto received his BSc degree from Bandung Institute of Technology in 1986, MEng and phDEng in computer science from Nagoya Institute of Technology in 1995 and 1998 respectively. Currently, he is a professor at the College of Computer Science and Information Technology, Albaha University, Saudi Arabia. He is also a research fellow at InterNetWorks Research Lab, UUM-CAS, Malaysia. His research interests include next generation network (IPv6 and smart networks), intelligent network monitoring system and security, intelligent systems and brain modeling.