Automatic Creation of Functional Sub-Networks Using Genetic Algorithms

Paper id 1-AC05-5

Abstract - *Research has shown a strong correlation between the topology and functional capability of neural networks, though difficulties encountered in traditional neural network design and training increase in relation to the size and complexity of the network. Constructing a neural network topology through the integration of modularized, functional sub-networks has been shown to provide a reduction in overall topological complexity and computational requirements. Furthermore, evolutionary computing based optimization techniques may be used to overcome traditional design difficulties. The research presented in this paper outlines a modular neural network approach for the approximation of a Mealy machine example of a sequential logic circuit. The use of a genetic algorithm for the automatic generation of an optimal set of trained functional sub-networks is described. Results concur with the use of evolutionary computing techniques as a method for overcoming traditional neural network design issues and the use of modularization for the reduction of task complexity.*

Keywords: Genetic algorithms, modular neural networks, Mealy machines, evolutionary computation.

1 Introduction

The topology of an artificial neural network (NN), comprising the network architecture, connectivity between neurons, and neuron model parameters, determines the NN's information processing ability. A corresponding operational performance is often dependent on the learning algorithm used to encode the features of a problem across the network's connection weights [13], [22], [27]. As the number of features encoded by a NN increases, a subsequent increase in both NN complexity and computational requirement occurs. However constructing a NN architecture through the integration of modularized, independently trained, functional sub-networks (subnets) has been shown to provide a reduction in overall topological complexity while enhancing the information processing capability and scalability [2], [5], [16]. In the design and construction of modular and non-modular NNs, multi-layer feed-forward network topologies which make use of gradient descent based learning algorithms prove to be one of the most popular implementation strategies for practical NNs [2], [20]. Typically, topological features are predetermined during the design stage, using the time consuming method of trial and error which may fail to produce an optimal solution for a given problem [22]. The efficiency of gradient descent based learning algorithms is dependent on the sensitivity of the network and algorithmic parameters used, such as learning rate, momentum and acceleration terms, with time taken to train a NN increasing exponentially with the number of connection weights [16], [20]. Optimization techniques from evolutionary computing (EC) may be used to address many of the difficulties encountered in the design and implementation of feed-forward NNs [18].

The research presented in this paper attempts to use an EC technique, e.g. genetic algorithms (GAs), in collaboration with NNs for the automatic generation of functional subnets used in a modular NN based approach to complex function approximation. Using a Mealy machine example of the sequential logic circuit for a 3-bit binary counter, an initial objective of task decomposition for subnet selection may be inferred from the implicit decomposition of the sequential logic circuit into logic sub-structures. The objectives of subnet architecture design and subnet training will be achieved through the utilization of a GA. Through the sequential operation of the evolved subnets, the objective of complex function approximation will be realized.

The organization of the paper consists of a brief introduction to the topics of modular and evolutionary approaches to the design and construction of NNs, followed by an overview of the GA implementation, incorporating a custom crossover operator. Details of the Mealy machine example will then be outlined. This is followed by a description of the subnet organization and implementation for the chosen example. Finally, the results obtained from the GA will be presented.

2 Modular Neural Networks

A modular neural network (MNN) represents a class of NN models whose creation was

motivated by a practical need to overcome the limitations inherent in traditional, non-modular NNs, thus providing a variety of architectures and techniques for modularization of topology and learning, plus multi-module integration [1-3], [5], [6], [8], [9], [11], [12], [19], [21]. Through the modularization of an NN topology, a goal task is decomposed into a set of sub-tasks, each learned independently by a subnet, with an aim to reducing high coupling and catastrophic interference incurred during learning. Similarly, the modularization of learning reduces the dimensionality of the set of training patterns for individual subnets, thus improving convergence speed and reducing the computational cost of learning [2]. Methods used for the decomposition of both task and training data include the use of techniques such as Adaptive Resonance Theory and Self Organizing Map networks [1], [2]. A variety of MNN models exist, two of which are illustrated in Figure 1 [2].

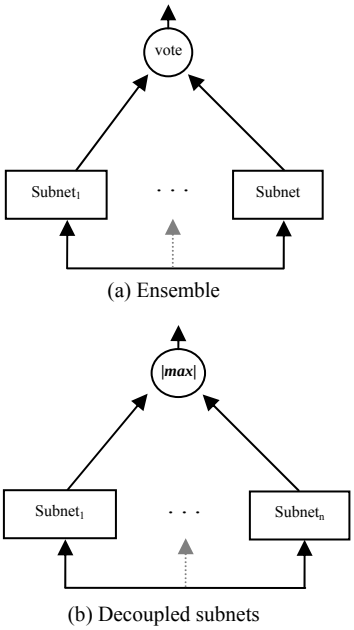

Figure 1. Modular NN topologies

MNN's composed of mixtures of expert networks use competition between the expert networks in order to learn and classify training data [2], [11], [12]. By contrast, cooperative multi-module integration, as used by ensemble based and cooperative MNNs, typically make use of voting schemes to determine the final output from groups of subnets [1-3]. Other methods of multi-module integration include combining the average and weighted average outputs, and the use of the absolute maximum output of all subnets for decoupled MNNs [1], [2], [6], [9], [19]. The modular approach used by the research presented herein is based on the decoupled subnet model. Each subnet will approximate a specific sub-task; the network training error calculated using the error obtained over all subnets per training pattern, and the overall network output obtained through the sequential operation of each subnet.

3 Evolutionary Neural Networks

The collaborative combination of EC and NN techniques may be used to provide solutions for the problems inherent in the design and implementation of an optimal NN architecture and corresponding set of connection weights [18]. By considering the problem of NN topology design as a search problem in the space of all possible topologies, a GA may be used to search the space for an optimal solution. Similarly, the disadvantages associated with gradient descent based learning, such as the requirement for a differentiable error surface, and the search becoming trapped in local minima, may be overcome using a GA's ability to cope with large, multi-modal, complex search spaces without requiring gradient information for the search [16], [20], [27]. Research has shown that using a GA for the construction of NN architecture or determining NN connection weights provides an efficient and effective method of implementation for both large and small scale problem domains, often outperforming traditional design methods and learning algorithms [4], [15], [20], [26], [28]. Architecture and connection weights may be evolved simultaneously through the use of a compound encoding within a single chromosome representation manipulated by the GA. Incorporating such duality in the evolutionary process may help overcome the problems of noisy fitness evaluation [28] and competing conventions [18], which can arise from the evolution of NN architecture alone. Care must be taken in the choice of encoding scheme used as the optimal NN will be obtained from the performance surface encoded by the GA's representation scheme [28].

4 Genetic Algorithms

As one of the most recognized forms of evolutionary algorithms, a GA is the computational simulation of evolutionary processes. Potential solutions to a problem are encoded as genotypes, represented through the use of chromosome strings and manipulated using a set of genetic operators for chromosome selection, recombination (crossover) and mutation [10]. Binary based chromosome representations provide a convenient and simple representation scheme allowing the application of a standard set genetic operators however suffer from limited precision. By contrast,

real-valued chromosome representations may offer increased precision and reduced string length however require the implementation of custom genetic operators [27], [28]. The GA manipulates a population of chromosomes over successive generations, decoding each genotype into phenotype form for evaluation of the potential solution's fitness. Characteristics and behaviors associated with the fittest strings are copied into the next generation, thus enabling the search to proceed toward an optimal solution, and imbuing the GA with the ability to "learn" the characteristics of the optimal solution [17]. A general description of the GA may be found in [7], [10], [17], [24], [25].

The GA used for the automatic creation of a set of functional subnets, as presented in this paper, will simultaneously search for the a set of optimal subnet architectures, corresponding connection weights and neuron model parameters by encoding these as genotypes within a single, real-valued chromosome string. The organization of the chromosome string used to encode a maximum of n subnets is illustrated in Figure 2, where h_n represent the number of hidden neurons in the nth subnet, ${}^{n}w_{ji}$ and ${}^{n}w_{kj}$ represent the connection weights between the input and hidden layer neurons, and the hidden and output layer neurons for the nth subnet respectively, and ${}^{n}b_j$ and ${}^{n}b_k$ represent the hidden and output neuron biases in the nth subnet.

$$\left\{h_1, {}^{1}w_{ji}, {}^{1}w_{kj}, {}^{1}b_j, {}^{1}b_k, \cdots, h_n, {}^{n}w_{ji}, {}^{n}w_{kj}, {}^{n}b_j, {}^{n}b_k\right\}$$

Figure 2. Organisation of chromosome for n subnets

If i, j, k represent the number of inputs, hidden neurons, and outputs, each subnet will require the following number of genes within each chromosome string:

$$\left(1 + (i * j) + (j * k) + j + k\right) \qquad (1)$$

In the initial population of chromosomes, random integers are used to specify the number of hidden neurons, while random real numbers are used for the specification of connection weights and bias values. For each generation all potential subnets from each chromosome are independently evaluated using a set of training patterns. As input patterns are applied to a set of subnets, the sum of the squared error between a target output pattern and the actual output for each subnet is calculated. The summation of the errors over the set of training patterns is determined and subsequently used as a chromosome's objective value. The calculation for the objective value for a single chromosome (Obj_c) is shown in equation (2). The corresponding calculation for the relative fitness (F_c) of a single chromosome is given in equation (3).

$$Obj_c = \sum_{j=1}^{P}\sum_{i=1}^{n}\left(Y_i^{\,j} - \hat{Y}_i^{\,j}\right)^2 \qquad (2)$$

where

Y_i = desired target output pattern,

$\hat{Y}_i$ = estimated output pattern,

n = 1, 2,…, no subnets,

P = 1, 2,…, no input patterns,

$$F_c = \frac{f_i}{\sum_{i=1}^{N} f_i} \qquad (3)$$

where $f_i = \frac{1}{Obj}$, N = 1, 2,…, population size.

After the evaluation of all chromosomes in the population, the relative fitness values are used by the GA's selection operator. The unbiased selection operator, Roulette Wheel Selection [25], is used to select chromosomes from the current population for inclusion in a mating pool. Pairs of parent chromosomes are randomly selected from the mating pool for recombination, according to a crossover rate, P_c, which determines the probability that a chromosome will be used for crossover. The custom recombination operator, termed 'isomorphic crossover', has been implemented to allow crossover to occur between two parent chromosomes consisting of isomorphic subnets represented by real-valued chromosome sub-strings. During isomorphic crossover the topologies of all subnets within the parent chromosomes are examined to determine if crossover is possible. If one or more topologies from both parent chromosomes match, single point crossover is performed on the respective chromosome sub-strings in order to create two new child chromosomes, as depicted in Figure 3.

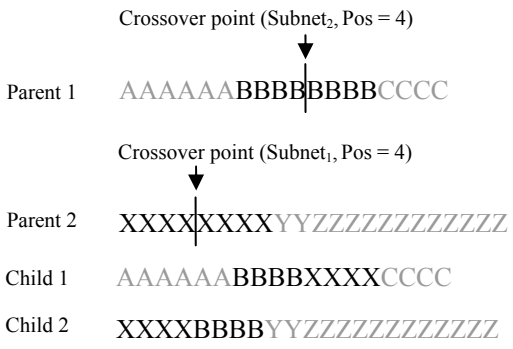

Figure 3. Isomorphic crossover of subnets

If a single topology from one parent matches with multiple topologies from the second parent, a subnet is randomly chosen from the second parent for crossover. If all subnet topologies from both parent chromosomes match, then single point crossover is performed over the entire length of the parent chromosome strings. When no topologies match, non-isomorphic crossover is applied, performing random mutation on gene values within both parent chromosomes. Chromosome strings that have been selected from the mating pool for recombination are always added to the next generation of the population, either as direct copies, if no recombination occurs, or as child chromosomes. Once a new population of chromosomes has been generated the GA's mutation operator is applied. Mutation is performed on randomly selected gene values within the entire population according to a mutation rate, P_m, which determines the probability that mutation will occur on an individual gene. Genes selected for mutation have a value in the interval [0,1] randomly added or subtracted. P_m is set at a higher rate for non-isomorphic crossover than for the standard mutation operator to ensure at least one of the genes in a chromosome selected for recombination will be modified. Using real-valued chromosomes necessitates the use of a single gene to encode the number of hidden neurons in each subnet. As isomorphic crossover only crosses genes within individual subnets, the topology of a subnet may only be modified through mutation, either as a result of non-isomorphic crossover or the mutation operator. The cycle of evaluation, selection, recombination and mutation repeats until a maximum number of generations of chromosomes have been evolved. Figure 4 illustrates cycle of processing performed by the collaborative GA-NN system.

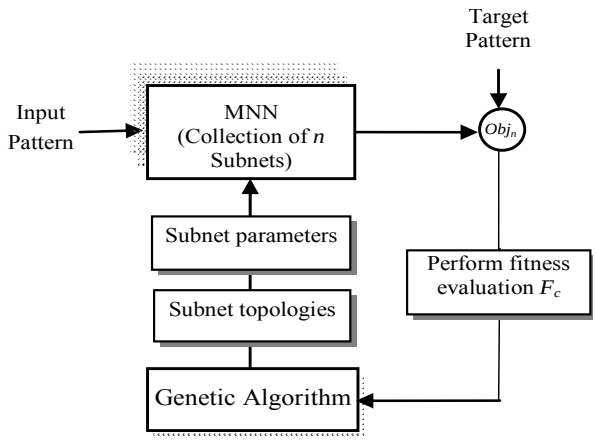

Figure 4. Block diagram of GA-NN system

5 Finite State Machines & Mealy Machine Example

Finite State Machines (FSM) embody a class of abstract machine that represent the behaviour of sequential logic circuits. Characterised by a set of inputs, outputs and internal states, a "memory" of previous inputs is retained through the use of a FSM's internal states. One type of FSM known as a Mealy machine, due to the outputs being dependent on the input and current state, can be defined by the ordered sextuple $M = (Q,I,O,\delta,\beta,q_0)$, where Q is a finite set of internal states, I is a finite set of inputs, O is a finite set of outputs, δ is the transition function $\delta : Q \times I \rightarrow Q$, β is the output function $\beta : Q \times I \rightarrow O$, and $q_0 \in Q$ is the initial state [14], [23].

Sequential circuits consist of a number of combinational logic circuits connected in a feedback loop with state memory circuitry. Binary values stored by state memory circuits, typically using a flip-flop for the storage of two distinct states, define the state of the sequential circuit at a given time *t*. Updating the state of the circuit at time *t+1* requires a function of the inputs and the state of the circuit at time *t*. Synchronisation of the flip-flops is achieved through the use of a clock pulse, upon receipt of which the state memory is updated [14], [23]. A binary counter circuit iterates through a predefined sequence of binary states in response to a series of input pulses, known as "count pulses". When 3-bits are used, the binary count from 000→111 is repeated, each bit of the counter corresponding to the state of a flip-flop at a given time *t*. Figure 5 demonstrates the sequential logic circuit required for a 3-bit binary counter in which a single T flip-flop is used to maintain the state for each bit. The outputs from each flip-flop, denoted C_0, C_1, C_2, form the count sequence [14].

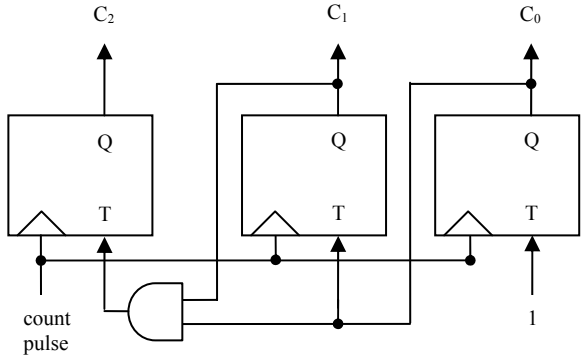

Figure 5. 3-bit binary counter circuit diagram

Labeling the flip-flops, TC_0, TC_1, and TC_2 respectively, the current state of the flip-flops at time *t*, the state of the flip-flops at time *t+1* and the inputs required to affect the state transition are listed in Table 1. The count sequence follows the

sequence of states from Q(t) to Q(t+1) for each flip-flop in response to a value of 1 being received by the first flip-flop TC_0 [14].

Table 1. State transitions for 3-bit binary counter

Q(t)			Q(t+1)			Input sequence		
C_2	C_1	C_0	C_2	C_1	C_0	TC_2	TC_1	TC_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

6 Subnet Implementation

The logic circuit of Figure 5 may be approximated through the use of a MNN, as illustrated in Figure 6. The functionality associated with individual automaton (flip flops TC_0, TC_1, and TC_2) are represented by the subnets, NN TC_0, NN TC_1, and NN TC_2. For each subnet, input and output layer neurons are determined by the inputs and outputs required for each flip flop, with the use of an additional input to each subnet corresponding to the current state, Q(t), of the respective flip flop.

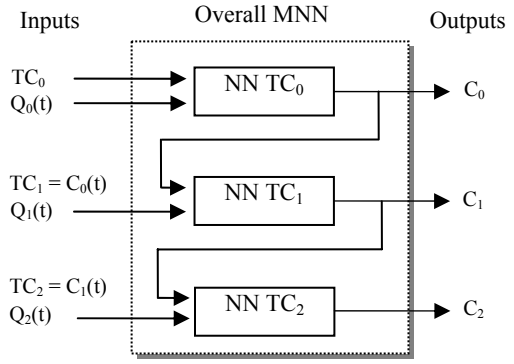

Figure 6. Overall MNN Structure

During subnet training, errors obtained from the outputs of the individual subnets are combined to produce an overall error value, thus retaining the coupling between individual subnets. Once an optimal set of subnets has been evolved, the sequential operation of the three subnets enables the construction of an overall MNN. As illustrated in Figure 6, the input to the first subnet consists of an input sequence, coupled with the current state of the subnet. For the second and third subnets, the input sequences used are determined from the outputs of the previous subnets. The three bit counter output is formed by the concatenating the output from the individual subnets.

7 Experimentation & Results

The MNN developed in Section 6, Figure 6, for the 3-bit binary counter was implemented through the evolution of the chromosome structure presented in Section 4, Figure 2. Each chromosome manipulated by the GA encoded the genotype representations for the three subnets shown in Figure 6. The algorithm for the collaborative GA-NN system, as depicted in Section 4, Figure 4, was repeatedly run for 200 generations, each containing an initial population size of 20 chromosomes, with the eventual production of a set of subnets capable of approximating the desired output patterns from the set of target input patterns. Running the subnets sequentially resulted in the concatenated output displaying the expected count sequence. The choice of training patterns provided an initial challenge with regard to specifying a set of patterns which would allow for the desired output from each of the subnets, while enabling the GA to successfully converge. One caveat of the chosen set of training patterns is that each subnet requires the current state of the flip flop to be approximated, Q(t), to be input along with the corresponding input pattern. When using this combination of inputs to each subnet, only a small number of training patterns were required to affect the correct operation of the evolved subnets. Figure 7 illustrates the average and minimum objective values obtained by the GA over 200 generations.

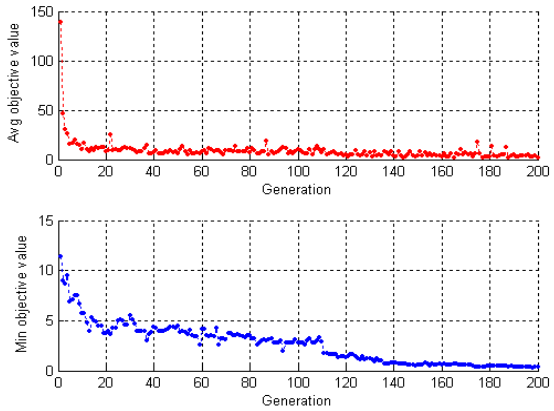

Figure 7. Objective value over 200 generations

As illustrated in Figure 7, convergence occurs rapidly within the first 20 generations, most likely due to the Roulette Wheel Selection algorithm used, followed by a period of oscillation with a slower convergence rate. Generations 140 to 200 provide a reduction in oscillations, resulting in stability of the minimum and average objective values towards the final generation. The minimum objective value, and subsequent chromosome used for further testing was obtained from generation

198. Figure 8 illustrates the corresponding average and maximum raw fitness values obtained.

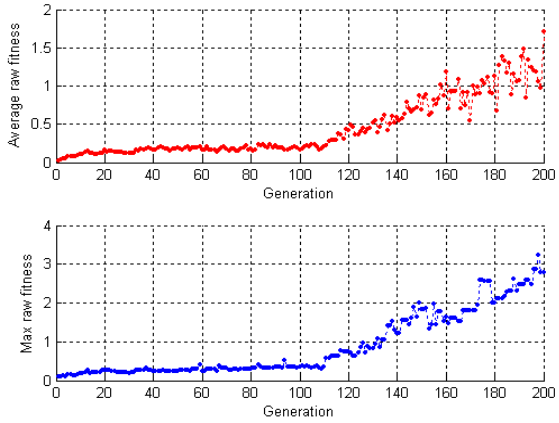

Figure 8. Raw fitness value over 200 generations

In correspondence with the reduction in oscillation and stability of the objective value, both average and maximum raw fitness values rapidly increase in size though oscillate towards the final generations. From the graph of the maximum raw fitness value, it can be seen that the chosen chromosome at generation 198 corresponds to the maximum raw fitness value obtained. The three subnets obtained from the chromosome had 6, 4, and 5 hidden neurons respectively.

8 Conclusion

In this research, the 3-bit binary counter, as a representative Mealy machine, has been decomposed into a set of simple automaton, each approximated by a small feed-forward neural network. The collaborative combination of GAs and NNs has been used to overcome the issues associated with traditional NN design, enabling the automatic creation of an optimal set of subnets, which successfully approximate the desired output of the counter when operated in a sequential manner. Using a modular approach to decomposition of the overall functionality has allowed for a small set of training patterns to be used, thus reducing computation requirements and convergence speeds for the individual subnets. The success of the approach for the chosen example implies that any finite state machine may be approximated in a similar fashion. Further research into the combination of GA and MNN approaches to complex function approximation from fields other than digital logic design would further strengthen the current research.

References

[1] G. Auda, & M. Kamel, "CMNN: Cooperative Modular Neural Networks", Neurocomputing, Vol 20, pp. 189-207, 1998.

[2] G. Auda, & M. Kamel, "Modular Neural Networks: A Survey", International Journal of Neural Systems, Vol 9, No. 2, pp. 129-151, 1999.

[3] G. Auda, M. Kamel, & H. Raafat, "Voting Schemes for Cooperative Neural Network Classifiers", IEEE International Conference on Neural Networks, Vol 3, pp. 1240-1243, 1995.

[4] D. Dasgupta, & D. R. McGregor, "Designing Application-Specific Neural Networks using the Structured Genetic Algorithm", International Workshop on Combinations of Genetic Algorithms and Neural Networks, Baltimore, pp. 87-96, June 1992.

[5] H. M. El-Bakry, "Modular Neural Networks for Solving High Complexity Problems", Proceedings of the International Joint Conference on Neural Networks, Vol 3, pp. 2202-2207, 2003.

[6] N. Garcia-Pedrajas, C. Hervas-Martinez, & D. Ortiz-Boyer, "Cooperative Coevolution of Artificial Neural Network Ensembles for Pattern Classification", IEEE Transactions on Evolutionary Computation, Vol 9, No. 3, pp. 271-302, 2005.

[7] D. E. Goldberg, Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, Reading, 1989.

[8] S. U. Guan, & P. Li, "Feature Selection for Modular Neural Network Classifiers", Journal of Intelligent Systems, Vol 12, No. 3, pp. 173-199, 2002.

[9] S. Hashem, B. Schmeiser, & Y. Yih, "Optimal Linear Combinations of Neural Networks: An Overview", IEEE International Conference on Neural Networks, Vol 3, pp. 1507-1512, 1994.

[10] J. H. Holland, Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence, MIT Press, Cambridge, 1992.

[11] R. A. Jacobs, M. I. Jordan, & A. G. Barto, "Task decomposition through competition in a modular connectionist architecture: the what and where vision tasks", Cognitive Science, Vol 15, No. 2, pp. 219-250, 1991.

[12] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, & G. E. Hinton, "Adaptive mixtures of local experts", Neural Computation, Vol 3, No. 1, pp. 79-87, 1991.

[13] N. Jiang, Z. Zhao, & L. Ren, "Design of structural modular neural networks with genetic algorithm", Advances in Engineering Software, Vol 34, pp. 17-24, 2003.

[14] M. M. Mano, Digital logic and computer design, Prentice-Hall, Englewood Cliffs, 1979.

[15] D. J. Montana, & L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms", Proceedings of 11th International Joint Conference on Artificial Intelligence, Vol 1, pp. 762-767, 1989.

[16] H. Muhlenbein, "Limitations of multi-layer perceptron networks – steps towards genetic neural networks", Parallel Computing, Vol 14, pp. 249-260, 1990.

[17] H. Muhlenbein, & J. Kindermann, "The Dynamics of Evolution and Learning", Connectionism In Perspective, pp. 173-197, 1989.

[18] J. D. Schaffer, D. Whitley, & L. J. Eshelman, "Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art", International Workshop on Combinations of Genetic Algorithms and Neural Networks, Baltimore, pp. 1-37, June 1992.

[19] A. J. Sharkey, "On Combining Artificial Neural Nets", Connection Science, Vol 8, No. 3 & 4, pp. 299-313, 1996.

[20] M. N. H. Siddique, & M. O. Tokhi, "Training Neural Networks: Backpropagation vs Genetic Algorithms", Proceedings of the International Joint Conference on Neural Networks, Vol 4, pp. 2673-2678, 2001.

[21] P. Tino, & J. Sajda, "Learning and Extracting Initial Mealy Automata with a Modular Neural Network Model", Neural Computation, Vol 7, No. 4, pp. 822-844, 1995.

[22] E. Vonk, L. C. Jain, & R. P. Johnson, Automatic Generation of Neural Network Architecture Using Evolutionary Computation, World Scientific Publishing, Singapore, 1997.

[23] J. F. Wakerly, Digital Design: Principles & Practices, Prentice-Hall International, New York, 2000.

[24] P. D. Wasserman, Advanced methods in neural computing, Van Nostrand Reinhold, New York, 1993.

[25] D. Whitley, "An overview of evolutionary algorithms: practical issues and common pitfalls", Information and Software Technology, Vol 43, No. 14, pp. 817-831, 2001.

[26] D. Whitley, T. Starkweather, & C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity", Parallel Computing, Vol 14, pp. 347-361, 1990.

[27] X. Yao, "Evolutionary Artificial Neural Networks", International Journal of Neural Systems, Vol 4, No. 3, pp. 203-222, 1993.

[28] X. Yao, "Evolving Artificial Neural Networks", Proceeding of the IEEE, Vol 87, No. 9, pp. 1423-1447, 1999.