การหาคำตอบดีที่สุดของเครือข่ายประสาทแบบย้อนกลับ: การเปรียบเทียบระหว่างอัลกอริทึมพันธุกรรมและการค้นหาทาบู

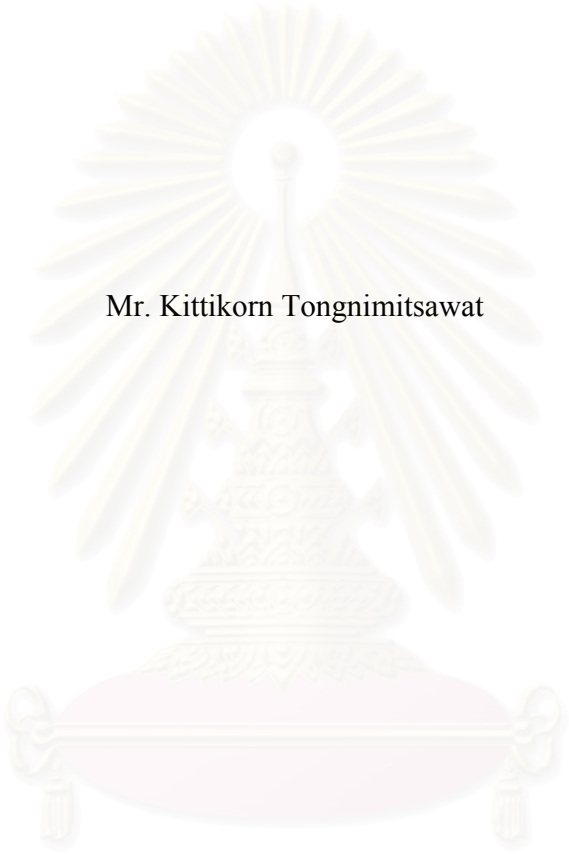
นาย กิตติกร ทองนิมิตสวัสดิ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2547

ISBN 974-53-1509-5

GLOBAL OPTIMIZATION OF RECURRENT NEURAL NETWORKS: A COMPARISON OF THE GENETIC ALGORITHM AND TABU SEARCH

Mr. Kittikorn Tongnimitsawat



A Thesis Submitted in Partial Fulfillment of the Requirements
For the Degree of Master of Science in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2004

ISBN 974-53-1509-5

Thesis Title	Global Optimization of Recurrent Neural Networks: A Comparison of the Genetic Algorithm and Tabu Search
By	Mr. Kittikorn Tongnimitsawat
Field of study	Computer Science
Thesis Advisor	Associate Professor Boonserm Kijsirikul, Ph. D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Master's Degree.

…… Dean of the Faculty of Engineering
(Professor Direk Lavansiri, Ph. D.)

THESIS COMMITTEE

…… Chairman
(Yunyong Teng-amnuay, Ph. D.)

…… Thesis Advisor
(Associate Professor Boonserm Kijsirikul, Ph. D.)

…… Member
(Associate Professor Somchai Prasitjutrakul, Ph. D.)

…… Member
(Thongchai Rojkangsadan)

กิตติกร ทองนิมิตสวัสดิ์ : การหาคำตอบดีที่สุดของเครือข่ายประสาทแบบย้อนกลับ: การเปรียบเทียบระหว่างอัลกอริทึมพันธุกรรมและการค้นหาทาบู. (GLOBAL OPTIMIZATION OF RECURRENT NEURAL NETWORKS: A COMPARISON OF THE GENETIC ALGORITHM AND TABU SEARCH) อ. ที่ปรึกษา : รศ. ดร. บุญเสริม กิจศิริกุล, 65 หน้า. ISBN 974-53-1509-5.

วิทยานิพนธ์นี้เสนอการนำอัลกอริทึมพันธุกรรมและการค้นหาทาบูมาประยุกต์ใช้กับการหาคำตอบที่ดีที่สุดของเครือข่ายประสาทแบบย้อนกลับแล้วเปรียบเทียบกับการเรียนรู้โดยการใช้แบ็คพรอพาเกชัน

ผลที่ได้พบว่าการเรียนรู้เครือข่ายประสาทแบบย้อนกลับด้วยอัลกอริทึมพันธุกรรมและการค้นหาทาบูจะส่งผลให้สามารถเรียนรู้ข้อมูลได้ดีกว่าการเรียนรู้โดยใช้แบ็คพรอพาเกชัน เนื่องจากอัลกอริทึมพันธุกรรมและการค้นหาทาบูสามารถหลุดออกจากบริเวณที่เป็นโลคอลได้ ส่งผลให้ค่าความผิดพลาดที่ได้จากอัลกอริทึมพันธุกรรมและการค้นหาทาบูมีค่าน้อยกว่าการเรียนรู้โดยการใช้แบ็คพรอพาเกชัน

อย่างไรก็ตาม การนำอัลกอริทึมพันธุกรรมและการค้นหาทาบูมาประยุกต์ใช้กับการหาคำตอบที่ดีที่สุดของเครือข่ายประสาทแบบย้อนกลับจะใช้เวลาในการเรียนรู้นานกว่าการเรียนรู้โดยการใช้แบ็คพรอพาเกชันซึ่งจะใช้ได้ดีในข้อมูลบางชุด

ภาควิชา.....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อนิสิต.....
สาขาวิชา.....วิทยาศาสตร์คอมพิวเตอร์..... ลายมือชื่ออาจารย์ที่ปรึกษา.....
ปีการศึกษา.....2547..... ลายมือชื่ออาจารย์ที่ปรึกษาร่วม.....
4471403421 : MAJOR COMPUTER SCIENCE

KEYWORD: Artificial Neural Network / Backpropagation / Genetic Algorithm / Optimization / Recurrent Neural Network /Tabu Search

KITTIKORN TONGNIMITSAWAT : GLOBAL OPTIMIZATION OF RECURRENT NEURAL NETWORKS: A COMPARISON OF THE GENETIC ALGORITHM AND TABU SEARCH. THESIS ADVISOR : ASSC. PROF. BOONSERM KIJSIRIKUL, Ph. D., 65 pp. ISBN 974-53-1509-5.

This thesis presents the methodology of applying Genetic Algorithm and Tabu Search in finding the global optima in Recurrent Neural Network. Then the result is compared with Backpropagation, the legacy method.

The result depicts that Genetic Algorithm and Tabu Search can help Recurrent Neural Network performs better than Backpropagation. This is because the Genetic Algorithm has a cross-over operator to jump off of local optima whilst Tabu Search employs Tabu list to prevent re-cycling search as well as using long term memory to make the searching broader.

However, Genetic Algorithm and Tabu Search take more time to find out the solution. In a short time running, Backpropagation can find a solution in some dataset better than others.

Department Computer Engineering	Student's Signature.....
Field of study Computer Science	Advisor's signature.....
Academic year 2004	Co-advisor's signature.....

ACKNOWLEDGEMENTS

This thesis is dedicated to my family and to all whom I was studied with. Especially, I would like to pay gratitude to my thesis advisor, Assc. Prof. Boonserm Kijsirikul, Ph. D., for his insight, helpful, and understanding both technically and personally throughout the period of this thesis study. In addition, I would to thank to the thesis committee for their reviewing in this thesis.

Thanks to my family, colleagues, friends, and everyone who gave such a nice cheer up and helpful suggestion. Also thanks to Mr. Suparuck Chaiyapan to be a helping hand of computer programming.

TABLE OF CONTENTS

Page

TABLE OF FIGURES

Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 PROBLEM IDENTIFICATION

Artificial Neural Network (ANN) is said to be one of the machine learning techniques modeled from biological human brain network. The ANN comprises of two main network topologies, Feed Forward Neural Network (FNN) and Recurrent Neural Network (RNN). FNN is one-way flow started from input neurons through output units. In contrast, RNN is two-way flow from input to output and vice versa. The RNN is mostly theoretically used in forecasting the time series information. Incidentally, increase of interest in applying artificial neural networks to business application is evident in both the academic and trade literature. This popularity is driven by the ability of the ANN to accurately approximate unknown functions and their derivatives [**1**].

Most applications of ANN use some variation of gradient technique called Backpropagation (BNN) for optimizing the networks. The past success of Backpropagation for optimizing neural networks is undeniable yet it has also been shown to be inconsistent in its application and unpredictable due to the local nature of its search.

Since gradient techniques, such as Backpropagation, converge optima locally [**2**], they often become trapped at suboptimal solution depending upon the possibility of the initial random starting point. Since obtaining an optimal solution is the goal of ANN training, a global search technique seems more suitable for this difficulty nonlinear optimization problem. Even though there were many attempts to reach the global optima of Backpropagation, the problem still exists.

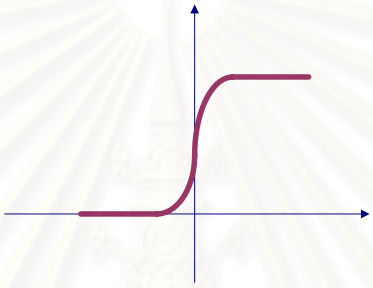
Escaping local optima can be done by many techniques such as Genetic Algorithm (GA), Tabu Search (TS), Scatter Search (SS), and Simulated Annealing (SA). Many researchers successfully currently demonstrated how to train the regular information into FNN to meet the optimal solution by applying *meta-heuristic*, GA [**3**] [**4**] and TS [**2**], whereas training the information in RNN with meta-heuristic is still questionable. Hence, this research would like to fulfill this missing part.

1.2 OBJECTIVE OF THE RESEARCH

The objective of this research is to train the RNN and then compare the results of two techniques, Genetic Algorithm and Tabu Search.

1.3 SCOPE OF THE RESEARCH

1. This research is to apply GA and TS to train RNN to find the optimal solution, global optima.
2. The RNN employs sigmoid function named logistic function as shown in Figure 1.
3. The training is conducted by adjusting weight vector only, adjusting network topology is excluded.
4. The training data set is generated by six given mathematic formulas.
5. The training data set are both with and without noise.
6. The results will be compared to each other and with the Back propagation technique.



$$f(net) = \frac{1}{1 + e^{(-\alpha net)}}$$

Figure 1: Logistic Function

1.4 DETAIL SCHEDULES

The detail schedules of this research shown below:

1. Search and study previous works in FNN and RNN.
2. Search and study previous works in BNN, GA, and TS.
3. Design the data structure of weights and parameters suitable with GA and TS.
4. Generate training data sets.
5. Implement the code of BNN, GA, and TS
6. Repeat step 1 to 5 as required.
7. Give the conclusion.

1.5 EXPECTED OUTCOME

The worth of this work is to study the methods of training GA and TS on RNN and to demonstrate how GA and TS can skip the local optima into global position.

CHAPTER 2

THEORIES AND RELATED RESEARCHS

2.1 ARTIFICIAL NEURAL NETWORK

2.1.1 Definition of Artificial Neural Network

An Artificial Neural Network is a structure composed of a number of interconnected units (artificial neurons) [**1**]. Each unit has an input/output (I/O) characteristic and implements a local computation or function. The output of any unit is determined by its I/O characteristic, its interconnection to other units, and (possibly) external inputs. Although "hand crafting" of the network is possible, the network usually develops an overall functionality through one or more forms of training.

2.1.2 Fundamental of Neural Network Concepts

A neural network is a dynamic system; its state changes over time in response to external input or an output state [**1**]. As the definition, the overall computational model consists of a reconfigurable interconnection of simple elements. Figure 2 depicts two small-scale sample networks, where units are denoted by circles and interconnections are shown as arcs. Figure 2 (a) depicts a *nonrecurrent* interconnection strategy, containing no closed interconnection paths. Note the depiction of units grouped in layers. By contrast, Figure 2 (b) illustrates a *recurrent* network interconnection strategy, where the arbitrary interconnection flexibility allows closed-loop (feedback) paths to exist. This allows the network to exhibit far more complex temporal dynamic compared with the strategy of Figure 2 (a). Also note that network topologies may be either static or dynamic. Finally, notice that some units in Figure 2 interface directly with the outside world, whereas others are *hidden* or internal.

The network topology design is subjective. The *Perceptron* is a two-layer neural network, i.e. input layer and output layer, while *Multilayer Perceptron* is a three-or-more-layer neural network. The additional layer from simple Perceptron is *hidden layer*. Both Perceptron and the multilayer Perceptron are trained with *error-correction learning*, which means that the desired response for the system must be known. The traditional means to determine response, error, is *Backpropagation*. However, individual units implement a local function, and the overall network of interconnected units displays a corresponding functionality. Analysis of this functionality, except through training and test examples, is often difficult. Moreover, the application usually determines the required functionality; it is the role of the ANN designer to determine network parameters that satisfy these specifications.

To be useful, neural networks must be capable of storing information, for example, they must be trainable. Neural systems are trained in the hope that they will subsequently display correct associative behavior when presented with new pattern to recognize or classify. That is, the objective in the training process is for the network

to develop an internal structure enabling it to correctly identify new, similar patterns. Modifying patterns of inter-element connectivity as a function of training data is a key learning approach. In other words, the system knowledge, experience, or training is stored in the form of network interconnections.

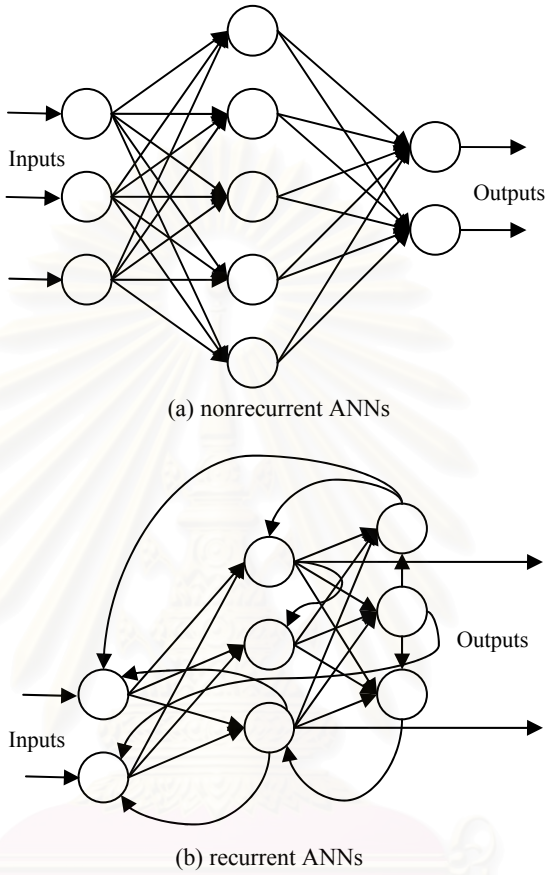


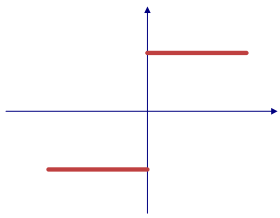
Figure 2: Basic Topologies for Neural Network

2.1.3 Activation Functions

The activation function is located in the neurons (circle units) and is responsible for combining inputs and forming outputs. Its input-output equation usually is

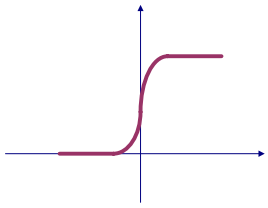
$$y = f(net) = f\left(\sum_{i=1}^{D} w_i x_i + b\right)$$

where D is the number of inputs, x_i are the inputs to the neurons, w_i are the weights, and b is a bias term. The activation function f is a threshold function. It has many forms, depending upon designer requirement, for instance, bipolar (Figure 3), logistic (Figure 4), and hyperbolic tangent, tanh (Figure 5). Logistic and hyperbolic tangent yield a sigmoid shape for the nonlinearity.



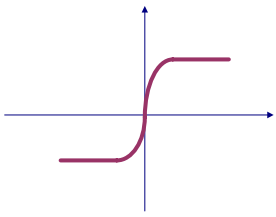
$$f(net) = \begin{cases} 1 & for \quad net \geq 0 \\ -1 & for \quad net < 0 \end{cases}$$

Figure 3: Bipolar Activation Function



$$f(net) = \frac{1}{1 + e^{(-\alpha net)}}$$

Figure 4: Logistic Function



$$f(net) = \tanh(\alpha net)$$

Figure 5: Hyperbolic Tangent Function

Bipolar is quite simple to use for classifying training data set into parts. If *net* is greater than or equal to zero, the function gives 1, else gives -1. However, bipolar function generates an unconnected line which led to be unable to find derivative function. Dislike Bipolar function, Logistic function has a continuous line. Logistic function will fire values between [0,1] while Hyperbolic Tangent function produces values between [-1,1].

2.1.4 Feed Forward Neural Network

The *Feed Forward Neural Network* (FNN) is composed of a hierarchy of processing units, organized in a series of two or more mutually exclusive sets of neurons or layers [**1**]. The first, or input, layer serves as a holding site for the inputs applied to the network. The last, or output, layer is the point at which the overall mapping of the network input is available. Between these two layers lies zero or more layers of *hidden units*; it is in these internal layers that additional remapping or computing takes place.

Links, or weights, connect each unit in one layer only to those in the next higher layer. There is an implied directionality in these connections, in that the output of a unit, scaled by the value of a connecting weight, is fed forward to provide a portion feedforward network. The network as shown consists of a layer of d input units (L_i), a layer of c output units (L_o), and a variable number of internal or "hidden" layers (L_{hi}) of units. Observe the feedforward structure, where the inputs are directly connected only to units in L_o, and the outputs of layer L_k are connected only to units in layer L_{k+1}.

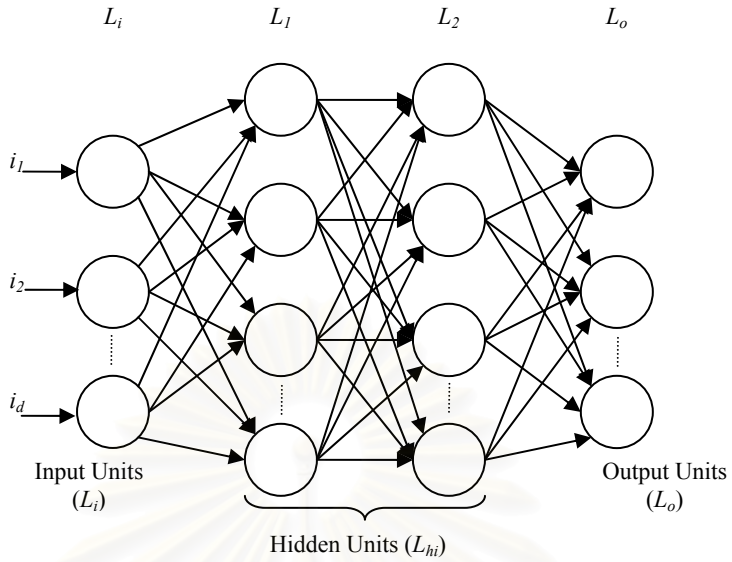


Figure 6: Layered feed forward network structure

2.1.5 Recurrent Neural Network

The FNN provided a trainable mapping from input to output. The mapping concept is modified somewhat for recurrent networks, since there is not necessarily a group of units that serve as inputs and outputs. Instead, the outputs of all units compose the network state. A *Recurrent Neural Network* (RNN) still maps, but it maps *states* into *states*. The network input is the initial state, and the mapping is through one or more states to the network outputs. The RNN is widely used in training the time-series information such as to predict the position of vehicles for next ten minutes. Hence, the forecasting should be based on the current position, velocity, acceleration, and any other factors, if any. Figure 7 shows the simple Jordan Neural Network adding one more group called *state units*. The state unit is to capture the last step information from outputs to be the addition input flow for hidden layers [**1**].

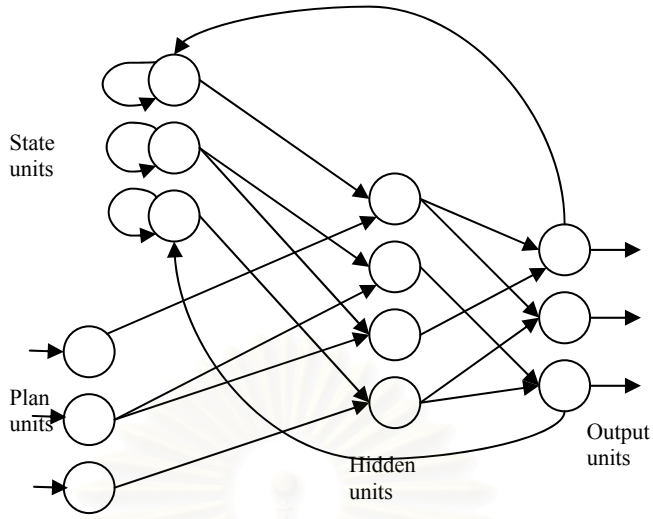


Figure 7: Example of recurrent Jordan net used for temporal sequence generation

2.1.6 Backpropagation

Definition of Backpropagation

Error-Backpropagation is a technique used to train multilayer feed-forward neural networks. The additional layer is *hidden*, or internal, layer. The numbers of neurons in the input and output layers are dictated by the sizes of the inputs and outputs for the problem we are trying to learn. But we can create the hidden layer as much as we desire [**5**].

Fundamental of Backpropagation

The values of neurons in the input layer are designated with the column vector *I*. The values of neurons in the hidden layer are designated with the column vector *H*. The values of neurons in the output layer are designated with the column vector *O*. The output values, we desire the output layer to produce and are trying to train it to produce, will be designated with the column vector *C*, stand for "*correct"*. Because we have two layers of synapse, we now have two matrices which define their weights: W and V. The connection to an output neuron O_i from a hidden neuron H_j has a weight W_{ij}. The connection to a hidden neuron H_j from an input neuron I_k has a weight V_{jk} [**5**].

This kind of neural network learns functions with any real value, not just Boolean type functions. This means that the neurons in the network can output any real number. To handle this, we need a new function describing the output value of a neuron.

$$O_i = \sigma(\sum_j W_{ij} H_j)$$

$$H_i = \sigma(\sum_k V_{jk} I_k)$$

$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

As discussed, the lower case sigma is the sigmoid function named logistic function. This sigmoid function is an S-Curve, which is 0 as negative infinity. As we approach 0, it begins to rise to 0.5. Then it continues to rise until it reaches 1 at infinity as shown in Figure 4. We can think of this function as a kind of smoothed form of the step functions. Other functions can be used besides the sigmoid, but it is particularly popular because it has a first derivative which is very simple, so it makes the computation easy.

The value of an input neuron is the input we have provided as before. The hidden neurons sum their weighted inputs from the input neurons and run it through the sigmoid function, and the result is their value. The output neurons then sum *their* weighted inputs from the hidden units, and run that through the sigmoid function, and the result is their value.

Unlike in the perceptron, the initial weights of a multilayer neural network start with small random values centered around 0 -- perhaps random values between -.01 and .01, for example. This is because if the weights are all 0, the network cannot learn -- the weights must be non-zero to learn.

To train the neural network, we present the inputs and determine the values of the hidden layer and of the output layer. We compare the results of the output layer to the correct results we are trying to train the network to produce. Then we modify the weights in W and V so that they are closer to producing that output. The rule we use for modifying the weights is known as the ***delta rule***, because it changes each weight according to how much say it had in the final outcome (the delta, or partial derivative of the output with respect to the weight). Before we derive the delta rule for a two-layer feed-forward neural network, here is the rule itself:

$$\Delta W_{ij} = \alpha(C_i - O_i) O_i (1 - O_i) H_j$$

$$\Delta V_{jk} = \alpha \sum_i (C_i - O_i) O_i (1 - O_i) \bullet W_{ij} H_j (1 - H_j) I_k$$

This rule is applied to all the weights at the same time. Use the old W weights in the V equation. The alpha value (α) is the ***learning rate***.

The overall procedure for training a Back propagation neural network is similar to the perceptron procedure and works as follows. Given a set of input vectors and corresponding expected output vectors for a function,

1. Pick an input/expected output vector pair at random.
2. Present the input vector pair to the input neurons.
3. Let the values flow through the neurons up to the output neurons.
4. Read the network's output vector.
5. Using the delta learning rule, modify each weight according to the difference between output vector and the expected output vector.
6. Go to #1.

The procedure finishes when the network is reliably producing something very close to the expected output for every input we provide it. Remember, this network operates on *continuous* values, so we cannot nail the expected output exactly. What we shall aim for is that the network will *converge* to the correct output at the limit.

Sometimes a Backpropagation network will converge for some inputs but fail to converge for others. This is because the network got caught in a *suboptimum solution* and cannot find its way out of it and on to the global optimum. Backpropagation has this property because it is a *greedy algorithm*, and so it is not guaranteed to work; we may have to run it many times to get all the inputs to converge.

One way to help the neural network converge is to lower the learning rate; it will take longer to learn, but will have a better chance of finding the global optimum. Another way to help it is to increase the number of neurons in the hidden layer. However, this second approach has a downside: if we increase the neurons in the hidden layer by too much, then the network will learn *exactly* the inputs we provide it, but will not be able to come up with a *general solution*. Often we cannot provide *all* the inputs to a network because there are just too many. In this case, we want the network to learn the function from just a subset, and successfully generalize what it learned to all possible inputs as a whole. Large numbers of hidden layer neurons make this less likely to succeed.

2.2 GENETIC ALGORITHM

A genetic algorithm is a robust search and optimization algorithm developed by John Holland that is loosely based on population genetics [**5**]. Genetic algorithms mimic the evolutionary principles and chromosomal processing in natural genetics to seek solutions from a vast search space at reasonable computation costs.

A genetic algorithm is an iterative procedure, and consists of a constant-size population of individuals. Each individual is represented by finite array of symbols, know as a *chromosome*. Each individual chromosome encodes a possible solution in the given problem space. Every chromosome is assigned a fitness value derived form a performance measure defined by the criteria to be optimized in the problem at hand. The algorithm starts out with an initial population of individuals that is generated at random. At each evolutionary step, known as a generation, the population of solutions is modified to a new population by applying three operators similar to natural genetic operators: reproduction, crossover, and mutation.

2.2.1 Reproduction

Reproduction generates a mating pool by selecting good fitness chromosomes from the population. Although there are numerous reproduction operators reported in the literature, the essential idea in all of them is the same: chromosome with fitnesses above-average are picked from the current population and chromosomes with fitnesses below-average are removed from the population. This procedure may involve maintaining multiple copies of good chromosomes in order to keep the population size fixed. The reproduction operator acts as a filtering mechanism for the selection of good chromosomes in a population.

2.2.2 Crossover

After reproduction, the crossover operator is applied to chromosomes of the mating pool. Once again, there are a number of crossover operators, but almost all of them pick two chromosomes from the mating pool at random and then exchange some portion of the chromosomes. In a single-point crossover operation, a crossover site is chosen at random and all bits to the right of the crossover site are exchanged between the two chromosomes.

It should be intuitively clear that from such an exchange operation, good sub-chromosomes from either parent chromosome can be combined to form a better child chromosome provided the right crossover site is chosen. Since we do not know the best crossover site in advance, a random site is chosen. This is simple to implement, but random crossover sites can generate children chromosomes that have a poorer fitness than the parents themselves. This is, however, not a problem, since the GA will automatically eliminate such poor quality chromosomes during the next reproduction cycle.

In a two-point crossover operator, two sites along the chromosome are chosen at random and the sub-chromosomes included between these sites are exchanged between the parents. This procedure is directly extendable to a multi-point crossover operator. The limiting case is called uniform crossover which exchanges every bit between parents with a certain probability. The crossover operator helps search the parameter space while preserving information from parents maximally, the latter being necessary since parent chromosomes are instances of good chromosomes selected using the reproduction operator.

2.2.3 Mutation

In addition to the crossover, a mutation operator is also used to enhance the search in a GA. The mutation operator flips a bit in a chromosome with a very small mutation probability. Mutation is necessary to maintain diversity in a population which would otherwise converge very quickly to very similar chromosomes.

We can also apply a mutation operator in adding and deducting some value from the gene. This is for a mutation operator on a real-number type gene.

2.2.4 Algorithm

Below is an algorithm for genetic algorithm in a general purpose.

1. Set k = 0; form initial population, P(0)
2. Evaluate P(k)
3. If stopping criterion satisfied, then stop
4. Select M(k) from P(k)
5. Evolve M(k) to form P(k+1)
6. Set k = k + 1, go to step 2

During the execution of the Genetic Algorithm, we keep track of the *best-so-far* chromosome; that is, the chromosome with the highest fitness of all chromosomes evaluated. After each iteration is completed, the best-so-far chromosome serves as the candidate for the solution to the original problem. Indeed, we may even copy the best-so-far chromosome into each new population, a practice referred to as *elitism*. The elitist strategy may result in domination of the population be super chromosomes. However, practical experience suggests that elitism often improves the performance of the algorithm.

The stopping criterion can be implemented in a number of ways. For example, a simple stopping criterion is to stop after a pre-specified number of iterations. Another possible criterion is to stop when the fitness for the best-so-far chromosome does not change significantly from iteration to iteration.

2.3 TABU SEARCH

Fred Glover initially introduced and later developed Tabu Search (TS) into a general framework [**7**]. Tabu Search can be thought of as an iterative descent method. An initial solution is probably randomly generated and a neighborhood around that solution is examined. If a new solution is found in the neighborhood that is preferred to the original, then the new solution replaces the old and the process repeats. If no new solution is found to improve the old function evaluation, then unlike a gradient descent procedure which would stop at that point, a local minimum, the TS algorithm may continue by accepting a new value that is worse than the old value. Therefore, a collection of solution in a given neighborhood is generated and the final solution would be based on the best solution found so far. To keep from cycling, an additional step is included that prohibits solution from recurring for user defined number of solutions, this *tabu list* is generated by adding the last solution to the beginning of the list and discarding the oldest solution from the list. During this procedure, the best solution found so far is retained. From the subset of acceptable neighborhoods the best solution is chosen. In a problem in which there is no known solution, a given value for the maximum number of iterations can be used to terminate the process. When this number of iterations has taken place with no new improvement over the best solution, the algorithm will terminate.

More particularly, TS is based on the promise that problem solving, in order to qualify as intelligent, must incorporate *adaptive memory* and *responsive exploration*.

The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memoryless designs that heavily rely on semi random process implementing a form of sampling.

The emphasis on responsive exploration in TS, whether in a deterministic (complete) or probabilistic (partial) implementation, derives from the supposition that a bad strategic choice can yield more information than a good random choice. In a system that uses memory, a bad choice based on strategy can provide clues about how the strategy may profitably be changed.

2.3.1 Use of Memory

The memory structures in TS refer to four principal dimensions, Recency, Frequency, Quality, and Influence. *Recency-based* memory and *Frequency-based* memory complement each other, and have important characteristics we amplify in later sections. The quality dimension refers to the ability to differentiate the merit of solutions visited during the search. In this context, memory can be used to identify elements that are common to good solutions or to paths that lead to such solutions. Operationally, quality becomes a foundation for incentive-based learning, where inducements are provided to reinforce actions that lead to good solution and penalties are provided to discourage actions that lead to poor solution. The flexibility of these memory structures allow the search to be guided in a multi-objective environment, where the goodness of a particular search direction may be determined by more that one function. The tabu search concept of quality is broader than the one implicitly used by standard optimization methods.

The fourth dimension, influence, considers the impact of the choices made during the search, not only on quality but also on structure. The influence of choices on particular solution elements incorporates an additional level of learning.

The memory used in TS is both *explicit* and *attributive*. Explicit memory records complete solution, typically consisting of elite solution visited during the search. An extension of this memory records highly attractive but unexplored neighbors of elite solutions. The memorized elite solution are used to expand the local search.

Alternatively, TS uses attributive memory for guiding purposes. This type of memory records information about solution attributes that change in moving from one solution to another. For example, in a graph or network setting, attributes can consist of nodes or arcs that are added, dropped or repositioned by the moving mechanism. In production scheduling, the index of jobs may be used as attributes to inhibit or encourage the method to follow certain search directions.

2.3.2 Intensification and Diversification

Two highly important components of TS are intensification and diversification strategies. Intensification strategy is based on modifying choice rules to encourage

move combinations and solution features historically found good. They may also initiate a return to attractive regions to search them more thoroughly. Since the elite solution must be recorded in order to examine their immediate neighbourhoods, explicit memory is closely related to the implementation of intensification strategy.

2.3.3 Aspiration Criteria

Sometimes, the situation, consisting of tabu active in tabu list, reaches good solution. The aspiration criteria is to switch off the tabu active status on which the situation with tabu active is better than what we have seen. This rule, aspiration criteria, will override the tabu action in hope that the way we break the rule can reach the optima.

CHAPTER 3

RESEARCH MEDTHODOLOGY

To reach what we expect from this research, we need to design the model to do the tests described in details below.

1. Prepare datasets to be trained and to be tested.
2. Design the network topology and data structure.
3. Study the way to apply Genetic Algorithm and Tabu Search in Neural Networks.
4. Do the experiment and give the result analysis.

3.1 DATASET PREPARATION

The dataset for training and testing is generated along with six experiment problems [**12**] [**9**] [**2**] [**10**]. The data will be used in every methodologies, GA, Tabu Search, and Back Propagation. The training set will be generated randomly once within the proper small range, e.g. [-100, 100]. The range may be various on each problem. Problems numbered one to four are taken from Schuster [**10**], while the fifth problem is the reputable Mackey-Glass chaos time series equation [**9**] [**2**] and the sixth problem has a big depth of local optima.

1. $X_t = 4X_{t-1}(1 - X_{t-1})$
2. $X_t = 4X_{t-1}^3 - 3X_{t-1}$
3. $X_t = X_{t-1}^2 - 1.8$
4. $X_t = X_{t-1}^2 - 1.6$
5. $Y_t = Y_{t-1} + 10.5\left[\frac{.2Y_{t-5}}{1 + (Y_{t-5})^{10}} - .1Y_{t-1}\right]$
6. $z = 3(1-x^2)e^{-x^2-(y+1)^2} - 10(\frac{x}{5} - x^3 - y^5)e^{-x^2-y^2} - \frac{e^{-(x+1)^2-y^2}}{3}$

Since this research aimed to apply the models and techniques to the real world where the data are hardly clean enough, we added some *noise* into the data we trained to test the model approaching to the real situation. The noise we made is to adding some small figure, for example 0.001, to the data randomly.

For the testing dataset, there are two means to test. First is *in-training data*, the dataset to be tested is the same set as trained. The second set to test the model is to use the data as *interpolation data.* It means the data to be tested will be in subset of training data. For example, we prepare 200 elements for each training dataset. After the training is completed, the testing dataset may be the first 150 records within the training dataset.

Since the training dataset can be prepared in two groups, data with noise and data without noise, and the testing data set will be two groups, in-training and interpolation, this means there are totally four combinations for every single problem to train and test the network.

Besides, the training and testing datasets will be normalized in order to prepare the data to suit with the network. Normalization can be useful when using techniques that perform mathematical operations such as multiplication directly on the value, such as neural networks, and K-means clustering since the normalization does not change the order of the values. Data normalization, furthermore, will rescale the input and output data into zero to one by using the calculation below.

$$x_{normalized} = \frac{x - Min(data)}{Max(data) - Min(data)}$$

For a number of data to be trained and tested, we use 300 data in training and 200 data in testing. The testing data set is from a series of training data. The result will show the error from both training and testing.

3.2 NETWORK TOPOLOGY DESIGN

This research employs Elman Recurrent Network as a model to test the algorithm which Jeff Elman invented in 1990 [**3**]. Elman network is a three-layer network, with the addition of a set of "context units" in the input layer. There are connections from the middle (hidden) layer to these context units fixed with a weight of one. The context units will memorize the previous outputs from a hidden layer. At each time step, the input is propagated in a standard feed-forward fashion, and then a learning rule, usually back-propagation, is applied. The fixed back connections result in the context units always maintaining a copy of the previous values of the hidden units (since they propagate over the connections before the learning rule is applied). Thus the network can maintain a sort of state, allowing it to perform such tasks as sequence-prediction that are beyond the power of a standard multi-layer perceptron. The Elman architecture in this research is shown in Figure 8.

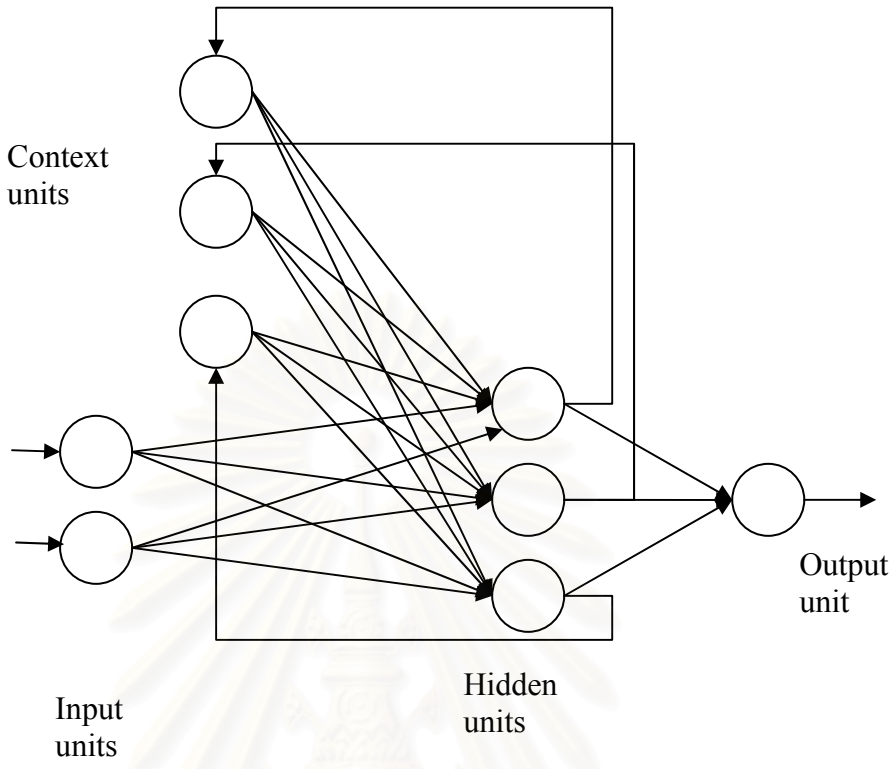


Figure 8 : Elman Recurrent Neural Network

Since the sixth function for generating the training and testing datasets employs three variables, *x* and *y* as independent variables, *z* as dependent variable, the network designed in this thesis will be two units in the input layer and one unit in the output layer. For the functions with one independent variable, like the function number one to five, we adapt the training data set by adding zero as an input for another variable.

For the hidden layer, the number of hidden units to optimize the solution is still questionable and has much discussed nowadays. Ben Krose and Patrick van der Smagt talked about the number of hidden units in the network. The number of hidden units should be large enough to learn the pattern of training data. Yet if it is too large, the network may tend to learn the data instead of learning the pattern, and it will cause over fitting.

The parameters set in the Elman network for this research are shown in the table below.

Table 1 : Neural Network parameters employs in this research

Parameter	Setting
Number of Input Nodes	2
Number of Hidden Nodes	4
Number of Output Nodes	1
Number of Hidden Layer	1
Transfer Function	Sigmoid Function

3.3 DATA STRUCTURE DESIGN

Since this research will apply other learning techniques on the network, besides Back Propagation, we need to design the data structure align with the network topology for Genetic Algorithm and Tabu Search. This will help doing the operation in GA and TS easier. The sequence of array of network is shown below.

Array	$W_{i,h}$	$W_{h,o}$	W_{ob}	W_{hb}	$W_{c,h}$

Where
$W_{i,h}$ = Weight from input units to hidden units
$W_{h,o}$ = Weight from hidden units to output units
W_{hb} = Bias for hidden units
W_{hc} = Bias for the output unit
$W_{c,h}$ = Weight from context units to hidden units

3.4 APPLYING GENETIC ALGORITHM TO TRAINING THE NETWORK

We employ Genetic Algorithm to Elman Network by creating a set of array of weights. Each gene represents weight in the network as well as the chromosome represents a set of weights, as shown in the previous section.

Recall to the procedures of genetic algorithm. We apply genetic algorithm to the network in the following means.

1. Initialize weights in the network and place every single weight into chromosome.
2. Do the Mutation
    a. Random one chromosome going to be mutated.

   b. Random gene position.
   c. Operate the random gene (b) with small change. The change may be plus or minus.
   d. Repeat 2.a to meet the mutation rate required.
3. Do the Crossover
   a. Random two chromosomes going to be mated.
   b. Random gene position.
   c. Cross two chromosomes (a) with the random point (b)
   d. Repeat 2.a to meet the crossover rate required.
4. Rank the chromosome by using Rank Method
   a. Calculate the error of the network corresponding to each choices of chromosome.
   b. Rank the chromosome by the error.
5. Select the chromosomes to be survived in next generation.
6. Repeat step 1 until the termination criteria is met.

The parameter set for Genetic Algorithm used this research is shown in the table below.

Table 2 : Genetic Algorithm parameters employed in this research

Parameter	Setting
Number of Chromosome in each generation	50, 100
Mutation Rate	20%, 40%, 60%
Crossover Rate	20%, 40%, 60%
Crossover Point	Single Point

When the procedure is finished, the minimum error and time spending are reported.

3.5 APPLYING TABU SEARCH TO TRAINING THE NETWORK

This research uses both Long Term and Short Term Memory strategies. The procedures of the implementation are as follows.

3.5.1 Long Term Memory Strategy

1. Initialize Long Term Memory.
2. Initialize weights for the network to be different from the existing in Long Term Memory.

3. Search the local neighbour for the optimum error with Short Term Memory Strategy.
4. Place the solution found in the Long Term Memory.
5. Repeat step 2 until the termination criteria is met.

3.5.2 Short Term Memory Strategy

1. Try to change every single element of the weight array by adding and deducting with the small value.
2. Determine the error of all new networks from step 1
3. Select the best network in step 2 only if the operation we selected is not in tabu list. However, the Tabu is acceptable if the best network found in step 2 is the best so far since we start training.
4. Add the operation what we have done in the Tabu List.
5. Delete the expired list from the Tabu List.
6. Repeat step 1 until the termination criteria is met.

As Long Term Memory Strategy aims to find the global optima by jumping around the solution space, the initial weights for the next search should be as far as possible from the existing network weights. If there is no data in Long Term Memory, the initial weights will be generated by randoming a set of numbers. If there are network weights existed in the Long Term Memory, the new initial weights can be calculated by averaging the weights in the same position of all network and then multiplying by -1.

Since the main property for Tabu Search is to prevent the operation to loop as a cycle. In this research, there are two kinds of Tabu. *Add Tabu List* is used when the operation is about deducting a weight with some little value. This list uses for preventing the weights that will not be added back within some next tenures. This tenure is called *Add Tabu Tenure*. On the other hand, *Minus Tabu List* is used when the operation is about to added. This will prevent the weight to get back to the same value by deducting in next loops. The tenure for this operation called *Minus Tabu Tenure*.

The parameter set for Genetic Algorithm used this research is shown in the table below.

Table 3 : Tabu Search parameters employs in this research

Parameter	Setting
Number of Loops in neighbour search (short term search)	300
Number of Short Term Memory run	3
Add Tabu Tenure	30, 60, 90, 120, 150
Minus Tabu Tenure	30, 60, 90, 120, 150

When the training is finished, the best network found so far will be reported.

3.6 TERMINATION CRITERIA

Normally, the termination criteria can be three different ways, the number of epochs or loops running, the number of time spending, and percent of error decreased from previous loop. The percent of error decreased from previous loop can apply well in Back Propagation since the error would tend to decrease in every step of training, but not for Genetic Algorithm and Tabu Search.

In Genetic Algorithm and Tabu Search, the error may be not decreased for next ten loops or more as the algorithms are random based in selecting the mutation and crossover. The random process in the current loop for both Genetic Algorithm and Tabu Search will not guarantee that the new network is going to have less error than that of previous loop. Then we cannot employ this method to stop training in this research.

3.7 ERROR MEASUREMENT

When the network training is finished, the error is measured. The data for all training and testing then will be passed through the network and the error of comparing the target and the output will be accumulated. *Sum Square Error* is used as a tool to measure the error.

3.8 SYSTEM ENVIRONMENT

The experiment is conducted with a personal computer. The computer hardware and software tested in this paper are shown as follows.

Table 4 : System environment employed in the paper

Environment	Value
Operation System	Window XP SP2
Development Suit	Microsoft Visual Basic 6
Processor	Pentium Centrino 1.3 GHz
Memory	768 MB
Hard Disk	60 GB

CHAPTER 4

RESULT ANALYSIS

4.1 BEST RESULT

There are three methodologies to train six datasets. Each dataset contains some parameters as described in chapter three. The best parameters to generate minimum average testing error for all three algorithms are shown below.

Table 5 : Best result for training with Genetic Algorithm – data without noise

Function	Average Testing Error	Average Training Error	No of Chromosome	Mutation Rate	Cross Over Rate	Generation	Time (Sec)
1	0.248	0.445	100	40%	60%	1000	810
2	0.199	0.284	100	40%	60%	1000	776
3	2.080	3.311	100	60%	20%	1000	361
4	0.093	0.246	100	60%	60%	1000	905
5	6.101	7.885	100	20%	20%	1000	293
6	0.734	0.918	100	40%	20%	1000	404

Table 6 : Best result for training with Tabu Search – data without noise

Function	Average Testing Error	Average Training Error	Add Tenure	Minus Tenure	Time (Sec)
1	0.506	0.813	60	150	260
2	0.558	0.929	90	150	257
3	5.629	7.985	120	60	177
4	0.355	0.618	90	150	258
5	6.087	8.569	120	150	237
6	0.328	0.525	60	150	259

Table 7 : Best result for training with Back Propagation – data without noise

Function	Average Testing Error	Average Training Error	Epoch	Learning Rate	Time (Sec)
1	0.395	0.632	300	0.75	11
2	1.711	2.616	500	1.25	19
3	25.877	37.205	300	0.75	13
4	0.813	1.253	300	1.25	10
5	8.618	13.536	500	1.25	19
6	0.228	0.317	500	1.25	20

RNN trained with Backpropagation almost reaches the minimum SSE within a short period, less than 20 seconds. Functions number one, two, four, and six are easy trained by Backpropagation. Genetic Algorithm and Tabu Search take longer time to converge to the best performance.

Functions number three and five show the local optima convergence in case of Backpropagation. When the training procedure is finished, the very high SSE remains. In contrast, RNN trained with Genetic Algorithm and Tabu Search can escape the local optima by using their properties. Genetic Algorithm employs crossover operation to jump out off local optima while Tabu Search uses both short term memory and long term memory strategies. These two algorithms give us a better performance but may take a very long time.

4.2 EXPERIMENT RESULTS

Refer to the tables shown in the appendices; they depict the experimental result for every combination. We can conclude from the experiments as follows.

Genetic Algorithm

For Genetic Algorithm, the increase of number of chromosomes, mutation rate, crossover rate, and number of generations, can reduce the SSE for both training and testing datasets, but the algorithm may take longer time to finish. Moreover, Genetic Algorithm can adjust the network weights to gain less error for the problem with local optima. The ability of skipping local optima is undeniable.

Tabu Search

For Tabu Search, the increase of Tabu Tenure cannot ensure that we will get a better solution, but increase of number of short term search running will. However, Tabu Search has a good chance to overwhelm a problem with local optima such as problems number three and five. This is because Tabu Search employs tabu list to prevent the cycling search and then the solution space may be explored wider enough to overcome the local optima. Moreover, Tabu Search applied in this thesis uses long term memory to initialize the network weight as far as previous ones. This strategy will let the search begins next round searching in different area and it may gain a better solution. However, it takes time to complete.

Backpropagation

The legacy optimization Backpropagation can reduce the error very fast since it is a mathematical based algorithm. The procedure to adjust the network weight can be easily calculated and applied to the network. This algorithm suits for the data with a small number of local optima.

For the network trained with data with noise, we can do the prediction as well. The error is quite similar with the network trained with data without noise. However, in the real world, after training is completed, just a few series will be predicted. Then it is not required to predict or forecast as many series of data as we did in the experiments..

Recalled to Genetic Algorithm and Tabu Search, one of their implementation steps is to adjust the network weights by adding or deducting the existing weights with a small number. We did the experiment by adding one variable to the learning process. This variable is to multiply the small number before changing the network weights. The result showed that the function can learning faster but cannot gain a better network and the error of the network is quite similar to the network with no multiplier variable.

In conclusion, the problem with local optima, Genetic Algorithm and Tabu Search can reduce the network error but they may take a long time whilst Backpropagation will take a shorter time to proceed but may stop at local optima.

Next is the sample of graph representation of function five. Figure 9 depicts the network error in the training process by using Backpropagation. This shows that this function converges to local optima. Figure 10 shows the target and the output from the network trained by Backpropagation.

Figure 11 to Figure 14 show the training capability of Genetic Algorithm and Tabu Search for the function which has local optima. The error from these two methods decrease overtime.

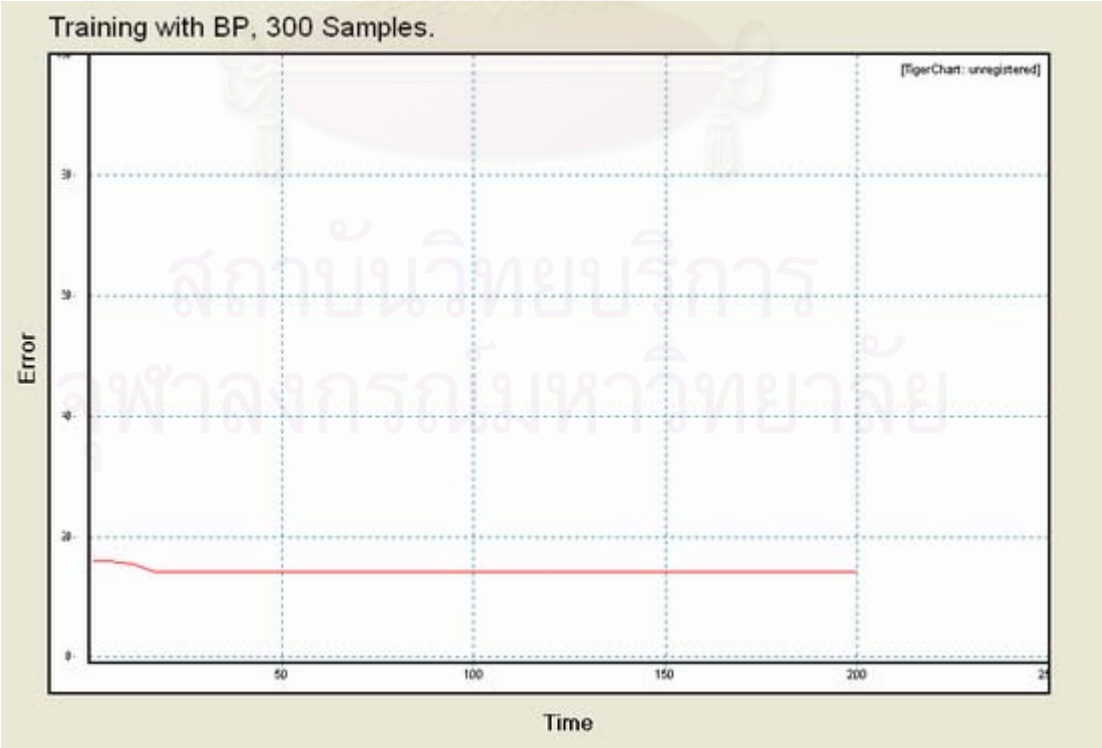


Figure 9 : Graphical representation of error in function 5 trained with Backpropagation

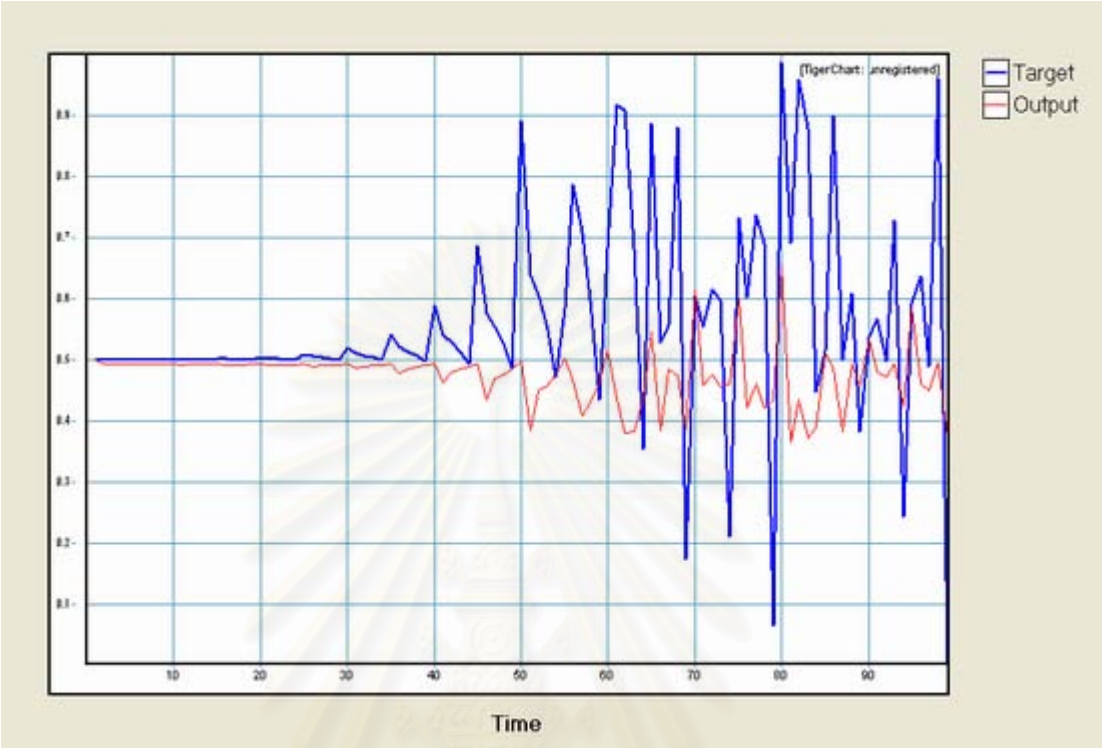


Figure 10 : Graphical representation of function 5 trained with Backpropagation

Figure 11 : Graphical representation of error in function 5 trained with Genetic Algorithm

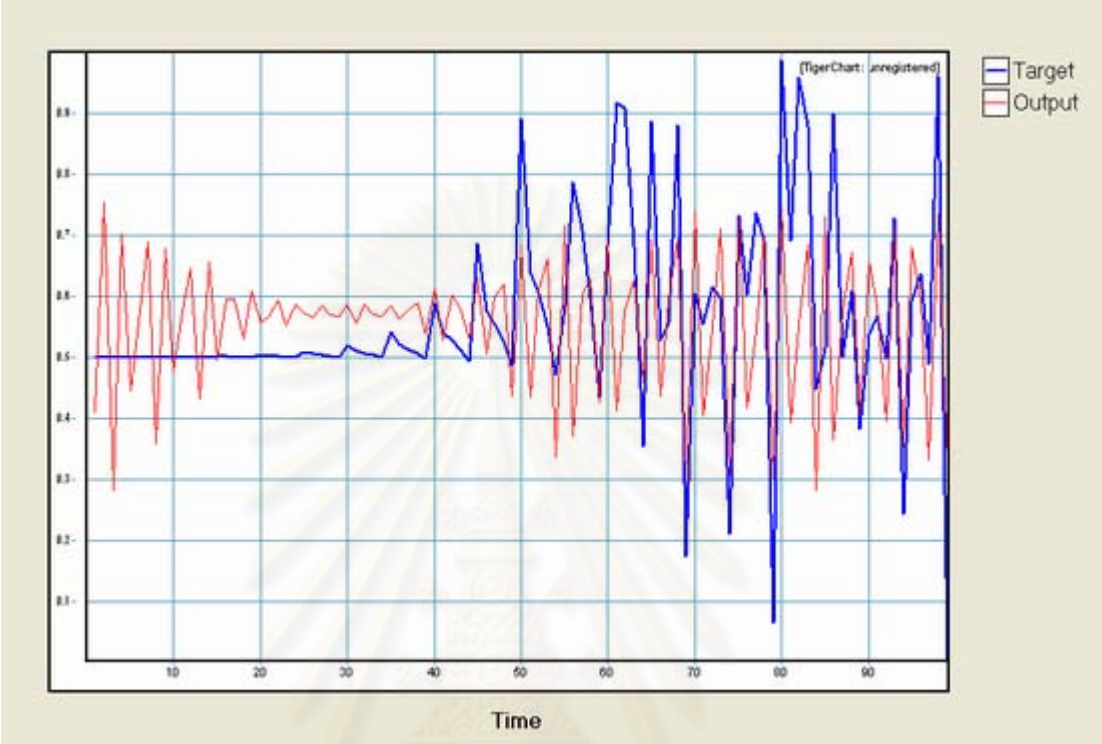


Figure 12 : Graphical representation of function 5 trained with Genetic Algorithm

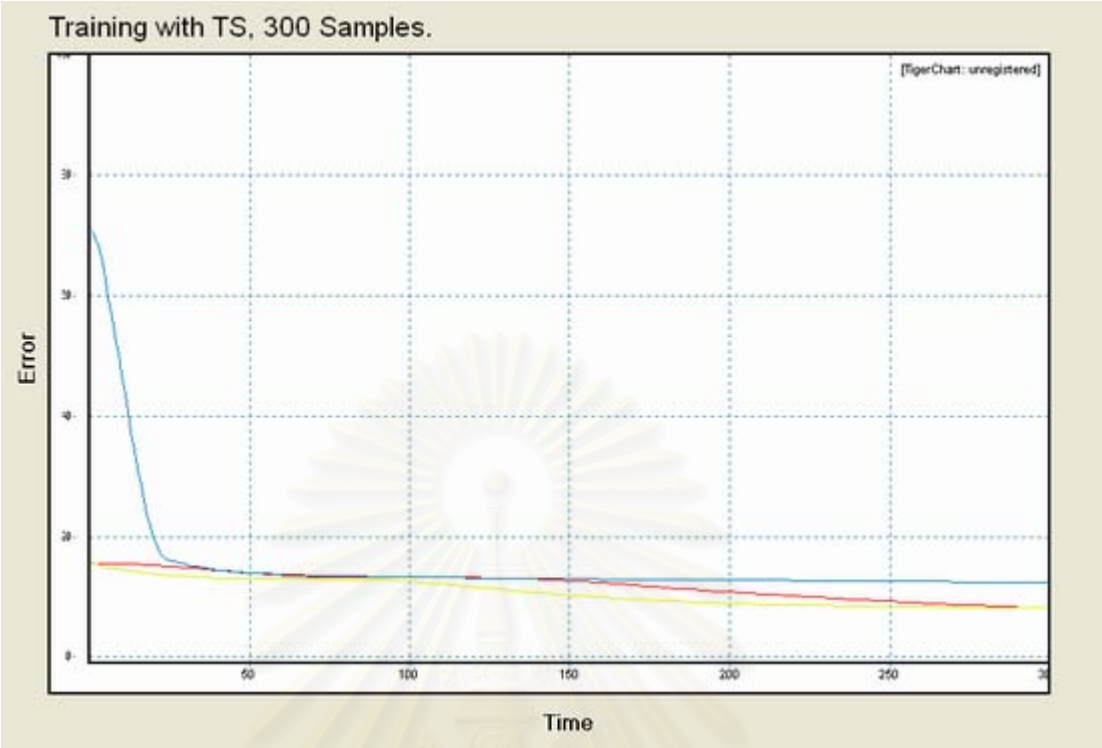


Figure 13 : Graphical representation of error in function 5 trained with Tabu Search

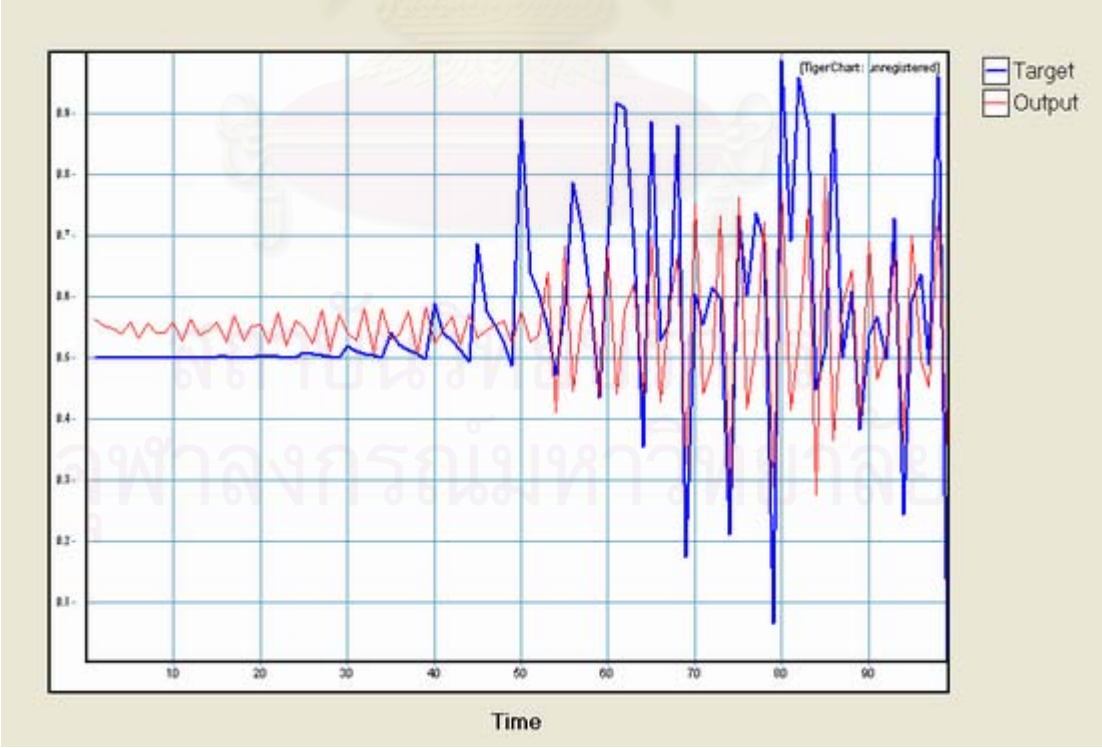


Figure 14 : Graphical representation of function 5 trained with Tabu Search

CHAPTER 5

CONCLUSION & FURTHER RESEARCH DIRECTIONS

5.1 CONCLUSION

This research depicts the methodology of training Genetic Algorithm and Tabu Search on Recurrent Neural Network. The result reveals that Backpropagation can learn functions quickly yet it fall in local optima. On the other hand, we can train RNN by using Genetic Algorithm and Tabu Search to escape the local optima but they may take longer time.

5.2 FURTHER RESEARCH DIRECTIONS

1. The number of hidden units and network topology directly affect the learning performance. Other RNN architectures, for example Jordan network, Hopfield network, suit for different purposes. The result from different architecture yields different result.

2. Backpropagation used in this paper is a simple version. Enhanced or extend Backpropagation, such as adding momentum, may yield a better result for function three.

3. Hybrid learning between Backpropagation and other metaheuristic searches may yield a better performance. In the beginning of learning, Backpropagation may be used to fast learning. When it reaches local optima, we can apply metaheuristic algorithms, such as Genetic Algorithm, and Tabu Search, to skip the local optima.

REFERENCES

1. Robert J. Schalkoff. (1997). **Artificial Neural Networks**. Singaport: McGraw-Hill International.
2. Randall S. Sexton, Bahram Alidace, Robert E. Dorsey, John D. Johnson. (1998). **Global Optimization for Artificial Neural Networks: A Tabu Search Application.** USA: College of Business, Ball State University.
3. Jeffrey L. Elman. (1990). Finding Structure in Time. **Cognitive Science 14**. pp. 179-211.
4. Randall S. Sexton, Robert E. Dorsey, John D. Johnson. (1998). **Toward Global Optimization of Neural Networks: A Comparison of the Genetic Algorithm and Backpropagation.** USA: College of Business, Ball State University.
5. Sean Luke. (2002). **Neural Networks lecture 2: Backpropagation**. URL: http://cs.gmu.edu/~sean/cs480/neuronlecture2/.
6. Satish Kumar. (2005). **Neural Networks A Classroom Approach**. Singapore: McGraw-Hill.
7. Fred Glover, Manuel Laguna. (2002). **Tabu Search In Handbook of Applied Optimization.** Oxford University Press. pp. 194-208.
8. Chen, J.R., & Mars, P. (1990). Step size Variation Methods for Accelerating the Back Propagation Algorithm. **Proceedings of The International Joint Conference on Neural Networks**. pp. 601-604. USA: Washington DC.
9. Randall S. Sexton. (1998). Identifying irrelevant input variables in chaotic time series problems: Using the Genetic algorithm for training neural networks. **Journal of Computational Intelligence in Finance**. pp. 34-41.
10. Schuster, H. (1995). **Deterministic chaos: An introduction**. New York: Wienhiem.
11. Berry, M.J.A., and Linoff, G. (1997). **Data Mining Techniques**. New York: John Wiley & Sons. p. 323.
12. Edwin K.P. Chong, Stanislaw H. Zak. (2001). **An Introduction to Optimization**. New York: A Wiley-Interscience Publication.
13. Fred Glover, Manuel Laguna. (1997). **Tabu Search**. Boston: Kluwer Academic Publishers.
14. De Falco, A. Iazzetta, P. Natale, E. Tarantino. (1998). Evolutionary Neural Network of Nonlinear Dynamics Modeling. **Research Institute in Parallel Informational Systems**. National Research Council of Italy.
15. De Falco, A. Iazzetta, P. Natale, E. Tarantino. (1998). Optimizing Neural Networks for Time Series Prediction. **Research Institute in Parallel Informational Systems**. National Research Council of Italy.
16. N. Wanas, G. Auda, M. Kamel and F. Karray (1998). On the optimal number of hidden nodes in a neural network. **IEEE 11th Canadian Conference on Electrical and Computer Engineering (CCECE'98)**. pp. 918-921. Canada: Waterloo.
17. P.A. Castillo, J.J. Merelo, J. Gonzalez, A. Prieto, V. Rivas, G. Romero. (1999). G-Prop-III: Global Optimization for Multilayer Perceptrons using an Evolutionary Algorithm. **Proceedings of the Genetic and Evolutionary Computation Conference**. USA: Florida.

APPENDIX

Table 8 : Function 1 trained by RNN with Genetic Algorithm – data without noise

Function	No of Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
1	50	20%	20%	500	12.957	8.702	72
1	50	20%	40%	500	1.988	1.156	115
1	50	20%	60%	500	1.182	0.744	182
1	50	40%	20%	500	2.040	1.225	97
1	50	40%	40%	500	0.845	0.507	143
1	50	40%	60%	500	1.473	0.891	187
1	50	60%	20%	500	2.273	1.377	127
1	50	60%	40%	500	1.198	0.694	166
1	50	60%	60%	500	1.297	0.821	214
1	100	20%	20%	500	6.345	4.192	147
1	100	20%	40%	500	4.759	2.526	240
1	100	20%	60%	500	3.997	2.507	341
1	100	40%	20%	500	1.141	0.668	199
1	100	40%	40%	500	3.313	1.948	299
1	100	40%	60%	500	1.498	0.907	397
1	100	60%	20%	500	2.020	1.277	261
1	100	60%	40%	500	2.765	1.691	400
1	100	60%	60%	500	2.006	1.051	460
1	50	20%	20%	1000	1.579	0.980	144
1	50	20%	40%	1000	1.150	0.696	230
1	50	20%	60%	1000	0.818	0.476	337
1	50	40%	20%	1000	0.799	0.487	190
1	50	40%	40%	1000	0.695	0.424	311
1	50	40%	60%	1000	0.757	0.469	376
1	50	60%	20%	1000	1.242	0.769	264
1	50	60%	40%	1000	0.838	0.502	333
1	50	60%	60%	1000	1.552	0.962	461
1	100	20%	20%	1000	2.419	1.516	291
1	100	20%	40%	1000	1.892	1.001	484
1	100	20%	60%	1000	0.675	0.402	672
1	100	40%	20%	1000	1.580	0.899	397
1	100	40%	40%	1000	1.432	0.910	593
1	100	40%	60%	1000	0.445	0.248	810
1	100	60%	20%	1000	0.540	0.310	507
1	100	60%	40%	1000	0.652	0.397	863
1	100	60%	60%	1000	0.682	0.425	900

Table 9 : Function 2 trained by RNN with Genetic Algorithm – data without noise

Function	No of Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
2	50	20%	20%	500	1.782	1.143	83
2	50	20%	40%	500	1.921	1.200	137
2	50	20%	60%	500	1.366	0.862	172
2	50	40%	20%	500	2.856	1.800	102
2	50	40%	40%	500	1.075	0.725	144
2	50	40%	60%	500	0.933	0.441	190
2	50	60%	20%	500	1.303	0.737	122
2	50	60%	40%	500	0.839	0.529	169
2	50	60%	60%	500	1.171	0.730	217
2	50	20%	20%	1000	1.026	0.696	170
2	50	20%	40%	1000	2.368	1.460	277
2	50	20%	60%	1000	1.818	1.053	336
2	50	40%	20%	1000	0.421	0.284	205
2	50	40%	40%	1000	0.622	0.425	283
2	50	40%	60%	1000	0.570	0.362	376

Function	No of Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
2	50	60%	20%	1000	0.590	0.398	243
2	50	60%	40%	1000	0.457	0.298	337
2	50	60%	60%	1000	0.564	0.384	433
2	100	20%	20%	500	2.576	1.621	152
2	100	20%	40%	500	2.471	1.607	247
2	100	20%	60%	500	0.765	0.433	540
2	100	40%	20%	500	0.739	0.474	199
2	100	40%	40%	500	1.410	0.839	298
2	100	40%	60%	500	0.574	0.365	395
2	100	60%	20%	500	1.175	0.680	253
2	100	60%	40%	500	0.769	0.475	354
2	100	60%	60%	500	1.187	0.734	454
2	100	20%	20%	1000	1.220	0.798	298
2	100	20%	40%	1000	0.796	0.520	484
2	100	20%	60%	1000	1.036	0.582	783
2	100	40%	20%	1000	0.901	0.565	400
2	100	40%	40%	1000	0.772	0.415	586
2	100	40%	60%	1000	0.284	0.199	776
2	100	60%	20%	1000	0.619	0.357	500
2	100	60%	40%	1000	0.496	0.217	740
2	100	60%	60%	1000	0.398	0.244	895

Table 10 : Function 3 trained by RNN with Genetic Algorithm – data without noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
3	50	20%	20%	500	36.386	28.210	48
3	50	20%	40%	500	36.071	28.690	79
3	50	20%	60%	500	34.068	26.460	110
3	50	40%	20%	500	36.360	27.154	65
3	50	40%	40%	500	34.348	27.947	97
3	50	40%	60%	500	36.229	28.724	128
3	50	60%	20%	500	36.164	28.255	82
3	50	60%	40%	500	34.732	28.522	113
3	50	60%	60%	500	35.804	28.249	146
3	50	20%	20%	1000	37.070	29.731	96
3	50	20%	40%	1000	11.097	12.308	157
3	50	20%	60%	1000	32.990	24.763	236
3	50	40%	20%	1000	34.689	28.301	130
3	50	40%	40%	1000	34.333	27.908	192
3	50	40%	60%	1000	34.904	28.378	254
3	50	60%	20%	1000	36.378	28.421	163
3	50	60%	40%	1000	35.217	28.113	226
3	50	60%	60%	1000	36.791	28.904	291
3	100	20%	20%	500	35.145	27.540	101
3	100	20%	40%	500	24.339	18.854	169
3	100	20%	60%	500	36.068	28.571	231
3	100	40%	20%	500	30.607	24.894	140
3	100	40%	40%	500	27.151	21.663	210
3	100	40%	60%	500	36.947	27.910	270
3	100	60%	20%	500	35.899	30.117	177
3	100	60%	40%	500	35.089	27.595	244
3	100	60%	60%	500	21.795	17.779	322
3	100	20%	20%	1000	36.611	27.840	200
3	100	20%	40%	1000	34.850	29.724	326
3	100	20%	60%	1000	37.112	27.781	449
3	100	40%	20%	1000	35.360	29.659	272
3	100	40%	40%	1000	36.061	29.403	403
3	100	40%	60%	1000	12.533	9.091	573
3	100	60%	20%	1000	3.311	2.080	361
3	100	60%	40%	1000	33.720	29.681	489
3	100	60%	60%	1000	35.439	29.422	627

Table 11 : Function 4 trained by RNN with Genetic Algorithm – data without noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
4	50	20%	20%	500	1.967	1.317	78
4	50	20%	40%	500	1.095	0.689	139
4	50	20%	60%	500	0.817	0.529	331
4	50	40%	20%	500	1.913	1.275	107
4	50	40%	40%	500	1.411	0.875	154
4	50	40%	60%	500	1.921	1.283	190
4	50	60%	20%	500	0.968	0.612	124
4	50	60%	40%	500	1.033	0.630	169
4	50	60%	60%	500	0.410	0.268	215
4	50	20%	20%	1000	1.075	0.690	187
4	50	20%	40%	1000	0.943	0.612	255
4	50	20%	60%	1000	0.380	0.178	736
4	50	40%	20%	1000	1.681	1.107	219
4	50	40%	40%	1000	1.071	0.675	287
4	50	40%	60%	1000	0.648	0.405	375
4	50	60%	20%	1000	1.038	0.712	240
4	50	60%	40%	1000	0.655	0.417	337
4	50	60%	60%	1000	0.672	0.419	428
4	100	20%	20%	500	1.645	1.098	146
4	100	20%	40%	500	1.746	1.158	242
4	100	20%	60%	500	1.696	1.130	336
4	100	40%	20%	500	1.258	0.811	201
4	100	40%	40%	500	0.581	0.378	297
4	100	40%	60%	500	1.463	0.950	398
4	100	60%	20%	500	2.206	1.455	253
4	100	60%	40%	500	0.532	0.377	357
4	100	60%	60%	500	1.113	0.704	456
4	100	20%	20%	1000	0.589	0.391	296
4	100	20%	40%	1000	1.581	1.034	482
4	100	20%	60%	1000	1.427	0.908	664
4	100	40%	20%	1000	0.495	0.319	398
4	100	40%	40%	1000	0.735	0.477	586
4	100	40%	60%	1000	0.311	0.209	797
4	100	60%	20%	1000	1.216	0.924	505
4	100	60%	40%	1000	0.534	0.349	708
4	100	60%	60%	1000	0.246	0.093	905

Table 12 : Function 5 trained by RNN with Genetic Algorithm – data without noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
5	50	20%	20%	500	8.826	6.956	71
5	50	20%	40%	500	9.787	7.230	116
5	50	20%	60%	500	9.701	7.286	163
5	50	40%	20%	500	8.466	6.488	95
5	50	40%	40%	500	8.442	6.577	141
5	50	40%	60%	500	9.085	6.611	188
5	50	60%	20%	500	8.234	6.483	121
5	50	60%	40%	500	7.807	6.117	168
5	50	60%	60%	500	7.629	6.700	215
5	50	20%	20%	1000	8.260	6.555	141
5	50	20%	40%	1000	7.917	6.874	232
5	50	20%	60%	1000	7.935	6.629	324
5	50	40%	20%	1000	12.259	10.146	189
5	50	40%	40%	1000	7.878	6.643	282
5	50	40%	60%	1000	7.516	7.004	376
5	50	60%	20%	1000	8.187	6.513	238
5	50	60%	40%	1000	7.708	6.722	336
5	50	60%	60%	1000	7.895	6.747	427
5	100	20%	20%	500	8.522	6.580	148

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
5	100	20%	40%	500	8.097	6.357	241
5	100	20%	60%	500	7.522	6.135	336
5	100	40%	20%	500	8.204	6.249	200
5	100	40%	40%	500	8.294	6.505	299
5	100	40%	60%	500	7.704	6.766	397
5	100	60%	20%	500	7.405	6.933	255
5	100	60%	40%	500	7.516	6.623	352
5	100	60%	60%	500	8.245	6.410	454
5	100	20%	20%	1000	7.885	6.101	293
5	100	20%	40%	1000	8.118	6.276	479
5	100	20%	60%	1000	7.338	6.853	666
5	100	40%	20%	1000	7.715	6.767	398
5	100	40%	40%	1000	7.160	6.221	593
5	100	40%	60%	1000	6.818	6.561	786
5	100	60%	20%	1000	7.441	6.405	499
5	100	60%	40%	1000	7.094	6.929	706
5	100	60%	60%	1000	6.467	6.381	906

Table 13 : Function 6 trained by RNN with Genetic Algorithm – data without noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
6	50	20%	20%	500	2.096	1.256	73
6	50	20%	40%	500	2.074	1.282	115
6	50	20%	60%	500	1.945	1.220	162
6	50	40%	20%	500	2.083	1.155	95
6	50	40%	40%	500	1.974	1.197	140
6	50	40%	60%	500	1.749	1.127	187
6	50	60%	20%	500	1.342	1.016	119
6	50	60%	40%	500	1.316	0.883	167
6	50	60%	60%	500	2.087	1.355	215
6	50	20%	20%	1000	2.163	1.227	140
6	50	20%	40%	1000	1.487	0.972	230
6	50	20%	60%	1000	1.668	1.164	324
6	50	40%	20%	1000	1.969	1.153	188
6	50	40%	40%	1000	1.709	1.129	280
6	50	40%	60%	1000	1.529	1.127	373
6	50	60%	20%	1000	1.748	1.145	237
6	50	60%	40%	1000	1.463	0.905	332
6	50	60%	60%	1000	1.569	1.021	424
6	100	20%	20%	500	1.856	1.180	146
6	100	20%	40%	500	1.826	1.131	237
6	100	20%	60%	500	2.088	1.299	335
6	100	40%	20%	500	1.374	0.981	201
6	100	40%	40%	500	1.798	1.254	296
6	100	40%	60%	500	2.212	1.348	390
6	100	60%	20%	500	1.887	1.100	255
6	100	60%	40%	500	2.030	1.120	354
6	100	60%	60%	500	1.317	0.996	466
6	100	20%	20%	1000	1.913	1.308	294
6	100	20%	40%	1000	1.790	1.038	478
6	100	20%	60%	1000	1.637	1.219	663
6	100	40%	20%	1000	0.918	0.734	404
6	100	40%	40%	1000	1.095	0.858	591
6	100	40%	60%	1000	1.780	1.148	781
6	100	60%	20%	1000	2.063	1.212	510
6	100	60%	40%	1000	1.494	1.085	712
6	100	60%	60%	1000	1.595	1.135	955

Table 14 : Function 1 trained by RNN with Tabu Search – data without noise

Function	Add Tenure	Minus Tenure	No Short Term Memory	Avg Training Error	Avg Testing Error	Time (Sec)
1	30	30	3	0.963	0.614	268
1	30	60	3	1.069	0.695	263
1	30	90	3	1.069	0.686	267
1	30	120	3	0.954	0.611	260
1	30	150	3	1.037	0.671	259
1	60	30	3	0.918	0.568	263
1	60	60	3	1.063	0.686	262
1	60	90	3	1.710	1.078	264
1	60	120	3	1.918	1.238	256
1	60	150	3	0.813	0.506	260
1	90	30	3	1.029	0.658	259
1	90	60	3	1.001	0.647	259
1	90	90	3	2.193	1.414	246
1	90	120	3	2.634	1.704	262
1	90	150	3	1.760	1.131	249
1	120	30	3	1.056	0.677	253
1	120	60	3	1.155	0.710	259
1	120	90	3	2.011	1.295	249
1	120	120	3	1.201	0.763	259
1	120	150	3	1.010	0.639	259
1	150	30	3	1.039	0.679	259
1	150	60	3	1.060	0.686	259
1	150	90	3	1.903	1.240	260
1	150	120	3	0.936	0.603	249
1	150	150	3	0.967	0.611	258
6	150	90	3	0.974	0.627	259
6	150	120	3	0.625	0.384	253
6	150	150	3	1.756	1.155	259

Table 15 : Function 2 trained by RNN with Tabu Search – data without noise

Function	Add Tenure	Minus Tenure	No Short Term Memory	Avg Training Error	Avg Testing Error	Time (Sec)
2	30	30	3	1.202	0.653	257
2	30	60	3	2.185	1.383	257
2	30	90	3	2.122	1.375	257
2	30	120	3	1.390	0.712	257
2	30	150	3	1.415	0.702	257
2	60	30	3	1.254	0.800	256
2	60	60	3	1.892	1.222	257
2	60	90	3	2.106	1.338	257
2	60	120	3	1.465	0.731	257
2	60	150	3	1.297	0.820	257
2	90	30	3	1.897	1.240	257
2	90	60	3	1.289	0.820	261
2	90	90	3	0.995	0.638	258
2	90	120	3	1.323	0.687	257
2	90	150	3	0.929	0.558	257
2	120	30	3	0.959	0.602	258
2	120	60	3	1.369	0.881	257
2	120	90	3	2.046	1.302	257
2	120	120	3	0.989	0.646	257
2	120	150	3	1.257	0.805	258
2	150	30	3	1.308	0.820	487
2	150	60	3	1.902	1.247	259
2	150	90	3	2.018	1.390	265
2	150	120	3	1.274	0.904	268
2	150	150	3	1.370	0.690	267

Table 16 : Function 3 trained by RNN with Tabu Search – data without noise

Function	Add Tenure	Minus Tenure	No Short Term Memory	Avg Training Error	Avg Testing Error	Time (Sec)
3	30	30	3	12.769	9.602	173
3	30	60	3	9.387	6.776	170
3	30	90	3	11.839	8.403	173
3	30	120	3	22.944	18.065	174
3	30	150	3	12.374	9.478	177
3	60	30	3	10.247	7.566	174
3	60	60	3	17.559	12.705	173
3	60	90	3	14.141	10.924	173
3	60	120	3	17.145	12.541	174
3	60	150	3	9.829	8.045	174
3	90	30	3	9.502	6.830	174
3	90	60	3	13.377	10.175	173
3	90	90	3	9.802	7.593	173
3	90	120	3	12.203	9.358	174
3	90	150	3	13.224	10.013	174
3	120	30	3	9.955	7.775	173
3	120	60	3	7.985	5.629	177
3	120	90	3	11.954	8.996	174
3	120	120	3	13.663	9.938	173
3	120	150	3	13.195	9.944	174
3	150	30	3	9.393	6.841	173
3	150	60	3	15.188	10.761	174
3	150	90	3	22.942	18.147	174
3	150	120	3	9.640	6.724	174
3	150	150	3	13.156	10.584	174

Table 17 : Function 4 trained by RNN with Tabu Search – data without noise

Function	Add Tenure	Minus Tenure	No Short Term Memory	Avg Training Error	Avg Testing Error	Time (Sec)
4	30	30	3	0.780	0.513	262
4	30	60	3	1.013	0.669	258
4	30	90	3	0.649	0.425	258
4	30	120	3	0.620	0.405	258
4	30	150	3	0.630	0.409	259
4	60	30	3	0.926	0.603	258
4	60	60	3	0.603	0.383	258
4	60	90	3	0.601	0.355	259
4	60	120	3	0.595	0.393	234
4	60	150	3	0.610	0.359	259
4	90	30	3	0.669	0.433	258
4	90	60	3	0.675	0.450	259
4	90	90	3	0.693	0.449	258
4	90	120	3	0.656	0.428	258
4	90	150	3	0.618	0.355	258
4	120	30	3	0.644	0.436	258
4	120	60	3	0.685	0.429	258
4	120	90	3	0.956	0.625	259
4	120	120	3	0.646	0.414	259
4	120	150	3	0.919	0.606	259
4	150	30	3	0.617	0.403	261
4	150	60	3	0.627	0.400	259
4	150	90	3	0.729	0.476	259
4	150	120	3	0.622	0.360	249
4	150	150	3	0.747	0.489	259

Table 18 : Function 5 trained by RNN with Tabu Search – data without noise

Function	Add Tenure	Minus Tenure	No Short Term Memory	Avg Training Error	Avg Testing Error	Time (Sec)
5	30	30	3	7.775	6.289	259
5	30	60	3	7.650	6.154	267
5	30	90	3	7.949	6.088	241
5	30	120	3	8.924	7.158	204
5	30	150	3	9.299	7.705	258
5	60	30	3	8.478	6.358	258
5	60	60	3	8.029	6.525	258
5	60	90	3	9.045	6.920	258
5	60	120	3	8.741	6.501	258
5	60	150	3	9.159	7.368	258
5	90	30	3	9.223	7.389	258
5	90	60	3	8.447	6.457	258
5	90	90	3	9.072	7.505	258
5	90	120	3	8.469	6.464	258
5	90	150	3	8.665	6.832	259
5	120	30	3	9.150	7.142	258
5	120	60	3	8.532	6.820	259
5	120	90	3	9.100	7.591	258
5	120	120	3	8.009	6.258	223
5	120	150	3	8.569	6.087	237
5	150	30	3	8.014	6.912	259
5	150	60	3	9.114	7.650	258
5	150	90	3	8.321	6.298	258
5	150	120	3	7.903	6.249	233
5	150	150	3	8.679	6.804	259

Table 19 : Function 6 trained by RNN with Tabu Search – data without noise

Function	Add Tenure	Minus Tenure	No Short Term Memory	Avg Training Error	Avg Testing Error	Time (Sec)
6	30	30	3	1.722	1.198	201
6	30	60	3	2.091	1.387	197
6	30	90	3	0.768	0.476	258
6	30	120	3	0.733	0.460	264
6	30	150	3	2.335	1.329	225
6	60	30	3	0.794	0.536	258
6	60	60	3	0.623	0.357	258
6	60	90	3	0.657	0.395	252
6	60	120	3	1.829	1.236	199
6	60	150	3	0.525	0.328	259
6	90	30	3	0.648	0.412	259
6	90	60	3	0.517	0.375	260
6	90	90	3	0.950	0.686	234
6	90	120	3	0.596	0.406	242
6	90	150	3	1.907	1.138	202
6	120	30	3	0.637	0.443	259
6	120	60	3	1.927	1.094	194
6	120	90	3	1.711	1.178	218
6	120	120	3	1.945	1.090	209
6	120	150	3	1.106	0.806	259
6	150	30	3	1.921	1.086	200
6	150	60	3	0.995	0.607	259

Table 20 : Function 1 to 6 trained by RNN with Back Propagation – data without noise

Function	Epoch	Learning Rate	Training Error	Testing Error	Time (Sec)
1	300	0.75	0.632	0.395	11.1
1	300	1.25	0.695	0.437	11
1	500	0.75	0.646	0.407	19.9
1	500	1.25	0.701	0.440	19.8
2	300	0.75	3.345	2.083	11.3
2	300	1.25	3.075	2.047	11.5
2	500	0.75	2.825	1.913	19.9
2	500	1.25	2.616	1.711	19.2
3	300	0.75	37.205	25.877	13
3	300	1.25	38.380	26.444	9
3	500	0.75	37.195	25.876	23.1
3	500	1.25	38.380	27.639	16
4	300	0.75	1.355	0.882	10.4
4	300	1.25	1.253	0.813	10.4
4	500	0.75	1.292	0.853	18.3
4	500	1.25	1.379	0.892	18.5
5	300	0.75	14.205	9.482	11
5	300	1.25	13.712	9.245	11.1
5	500	0.75	13.981	9.073	19.1
5	500	1.25	13.536	8.618	18.8
6	300	0.75	0.565	0.434	13.9
6	300	1.25	0.433	0.321	13.8
6	500	0.75	0.425	0.314	20.4
6	500	1.25	0.317	0.228	20.1

Table 21 : Function 1 trained by RNN with Genetic Algorithm – data with noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
1	50	10	10	500	18.657	12.466	86
1	50	10	10	1000	0.928	0.545	177
1	50	10	20	500	25.241	17.295	177
1	50	10	20	1000	1.376	0.806	280
1	50	10	30	500	1.158	0.706	195
1	50	10	30	1000	1.312	0.758	389
1	50	20	10	500	6.648	4.245	116
1	50	20	10	1000	1.006	0.593	231
1	50	20	20	500	2.101	1.334	173
1	50	20	20	1000	1.962	1.213	343
1	50	20	30	500	2.015	1.292	227
1	50	20	30	1000	1.198	0.731	455
1	50	30	10	500	1.178	0.682	147
1	50	30	10	1000	2.227	1.259	292
1	50	30	20	500	3.316	1.979	204
1	50	30	20	1000	0.767	0.460	407
1	50	30	30	500	0.732	0.439	260
1	50	30	30	1000	0.763	0.436	520
1	100	20	20	500	7.029	4.573	184
1	100	20	20	1000	1.241	0.754	359
1	100	20	40	500	2.127	1.342	299
1	100	20	40	1000	1.067	0.671	585
1	100	20	60	500	1.902	1.241	411
1	100	20	60	1000	1.545	0.887	811
1	100	40	20	500	1.425	0.849	249
1	100	40	20	1000	0.896	0.516	481
1	100	40	40	500	3.389	2.131	370
1	100	40	40	1000	2.384	1.044	723
1	100	40	60	500	1.971	1.324	492
1	100	40	60	1000	1.037	0.624	960

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
1	100	60	20	500	1.052	0.581	321
1	100	60	20	1000	0.967	0.598	626
1	100	60	40	500	0.918	0.560	446
1	100	60	40	1000	0.787	0.421	861
1	100	60	60	500	0.754	0.455	565
1	100	60	60	1000	0.687	0.414	1118

Table 22 : Function 2 trained by RNN with Genetic Algorithm – data with noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
2	50	10	10	500	2.456	1.594	87
2	50	10	10	1000	4.160	2.619	172
2	50	10	20	500	1.546	0.960	142
2	50	10	20	1000	0.582	0.392	282
2	50	10	30	500	3.258	2.083	200
2	50	10	30	1000	0.706	0.395	397
2	50	20	10	500	2.079	1.247	118
2	50	20	10	1000	0.916	0.579	236
2	50	20	20	500	0.728	0.454	173
2	50	20	20	1000	0.166	0.130	344
2	50	20	30	500	1.049	0.707	230
2	50	20	30	1000	0.622	0.403	455
2	50	30	10	500	1.211	0.774	147
2	50	30	10	1000	0.568	0.330	292
2	50	30	20	500	0.951	0.644	204
2	50	30	20	1000	0.435	0.284	404
2	50	30	30	500	0.709	0.431	263
2	50	30	30	1000	0.221	0.154	521
2	100	20	20	500	2.376	1.521	184
2	100	20	20	1000	0.773	0.469	362
2	100	20	40	500	1.610	1.051	300
2	100	20	40	1000	1.628	0.986	582
2	100	20	60	500	1.323	0.814	417
2	100	20	60	1000	0.581	0.363	820
2	100	40	20	500	0.673	0.404	252
2	100	40	20	1000	0.573	0.352	494
2	100	40	40	500	0.980	0.469	439
2	100	40	40	1000	0.310	0.222	772
2	100	40	60	500	1.049	0.466	495
2	100	40	60	1000	0.607	0.252	978
2	100	60	20	500	1.083	0.521	323
2	100	60	20	1000	0.404	0.270	630
2	100	60	40	500	0.858	0.410	444
2	100	60	40	1000	0.520	0.297	874
2	100	60	60	500	0.810	0.316	572
2	100	60	60	1000	0.697	0.380	1116

Table 23 : Function 3 trained by RNN with Genetic Algorithm – data with noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
3	50	10	10	500	32.284	24.808	87
3	50	10	10	1000	30.772	25.008	172
3	50	10	20	500	33.904	28.196	141
3	50	10	20	1000	23.466	22.426	280
3	50	10	30	500	31.337	24.251	199
3	50	10	30	1000	30.710	22.928	393
3	50	20	10	500	28.602	23.417	117
3	50	20	10	1000	26.134	22.321	229
3	50	20	20	500	17.175	14.665	173
3	50	20	20	1000	16.685	17.356	341
3	50	20	30	500	20.262	16.625	228
3	50	20	30	1000	19.678	14.379	457
3	50	30	10	500	30.180	25.153	147
3	50	30	10	1000	19.166	15.243	291
3	50	30	20	500	23.780	21.737	205
3	50	30	20	1000	23.516	18.245	406
3	50	30	30	500	21.173	23.364	261
3	50	30	30	1000	15.570	15.533	519
3	100	20	20	500	31.940	26.010	185
3	100	20	20	1000	30.258	25.080	362
3	100	20	40	500	27.526	23.148	296
3	100	20	40	1000	20.281	16.460	587
3	100	20	60	500	31.818	26.449	416
3	100	20	60	1000	22.912	19.215	821
3	100	40	20	500	30.153	23.285	251
3	100	40	20	1000	24.676	20.782	491
3	100	40	40	500	31.604	25.527	372
3	100	40	40	1000	12.507	14.456	734
3	100	40	60	500	17.883	15.414	490
3	100	40	60	1000	14.598	11.151	964
3	100	60	20	500	19.780	19.853	320
3	100	60	20	1000	13.469	12.299	624
3	100	60	40	500	18.835	18.682	444
3	100	60	40	1000	12.844	11.155	870
3	100	60	60	500	26.011	21.397	572
3	100	60	60	1000	15.810	14.622	1138

Table 24 : Function 4 trained by RNN with Genetic Algorithm – data with noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
4	50	10	10	500	1.973	1.307	86
4	50	10	10	1000	1.725	1.118	169
4	50	10	20	500	1.920	1.259	140
4	50	10	20	1000	1.113	0.700	279
4	50	10	30	500	1.981	1.337	195
4	50	10	30	1000	0.912	0.551	390
4	50	20	10	500	1.969	1.306	116
4	50	20	10	1000	2.026	1.331	228
4	50	20	20	500	1.172	0.772	172
4	50	20	20	1000	0.222	0.145	343
4	50	20	30	500	0.770	0.479	228
4	50	20	30	1000	1.155	0.741	455
4	50	30	10	500	0.788	0.426	147
4	50	30	10	1000	0.836	0.534	290
4	50	30	20	500	0.472	0.343	205
4	50	30	20	1000	0.318	0.205	405
4	50	30	30	500	1.662	1.102	262
4	50	30	30	1000	1.918	1.281	518

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
4	100	20	20	500	1.639	1.059	183
4	100	20	20	1000	1.097	0.716	361
4	100	20	40	500	0.731	0.448	298
4	100	20	40	1000	0.402	0.211	592
4	100	20	60	500	1.659	1.074	408
4	100	20	60	1000	0.898	0.589	808
4	100	40	20	500	0.514	0.337	249
4	100	40	20	1000	0.776	0.516	495
4	100	40	40	500	0.942	0.529	372
4	100	40	40	1000	0.487	0.355	729
4	100	40	60	500	1.125	0.736	481
4	100	40	60	1000	0.438	0.272	962
4	100	60	20	500	1.627	1.043	319
4	100	60	20	1000	0.362	0.255	628
4	100	60	40	500	0.754	0.428	444
4	100	60	40	1000	0.545	0.374	870
4	100	60	60	500	1.138	0.719	565
4	100	60	60	1000	0.123	0.077	1133

Table 25 : Function 5 trained by RNN with Genetic Algorithm – data with noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
5	50	10	10	500	9.426	6.491	86
5	50	10	10	1000	8.035	6.722	86
5	50	10	20	500	9.552	7.097	141
5	50	10	20	1000	8.286	6.607	279
5	50	10	30	500	8.522	6.609	195
5	50	10	30	1000	12.052	10.532	389
5	50	20	10	500	9.144	6.884	116
5	50	20	10	1000	7.411	6.835	228
5	50	20	20	500	7.975	6.383	171
5	50	20	20	1000	7.599	6.561	343
5	50	20	30	500	8.219	6.403	228
5	50	20	30	1000	7.670	6.478	454
5	50	30	10	500	8.680	6.304	146
5	50	30	10	1000	8.244	6.174	287
5	50	30	20	500	8.193	6.386	205
5	50	30	20	1000	7.798	6.236	404
5	50	30	30	500	7.895	6.770	260
5	50	30	30	1000	11.005	9.311	522
5	100	20	20	500	9.486	7.531	185
5	100	20	20	1000	8.173	6.750	358
5	100	20	40	500	8.565	6.506	296
5	100	20	40	1000	7.815	6.812	584
5	100	20	60	500	8.952	6.665	414
5	100	20	60	1000	7.863	6.339	808
5	100	40	20	500	8.338	6.676	246
5	100	40	20	1000	7.702	6.355	485
5	100	40	40	500	12.991	10.102	370
5	100	40	40	1000	7.302	5.716	727
5	100	40	60	500	8.877	7.037	483
5	100	40	60	1000	7.575	6.214	963
5	100	60	20	500	12.559	10.087	313
5	100	60	20	1000	8.564	6.375	620
5	100	60	40	500	7.727	7.393	436
5	100	60	40	1000	8.197	6.453	862
5	100	60	60	500	7.424	6.511	565
5	100	60	60	1000	8.031	6.639	1100

Table 26 : Function 6 trained by RNN with Genetic Algorithm – data with noise

Function	Chromosome	Mutation Rate	Cross Over Rate	Generation	Avg Training Error	Avg Testing Error	Time (Sec)
6	50	10	10	500	2.034	1.157	88
6	50	10	10	1000	1.608	1.175	174
6	50	10	20	500	2.242	1.310	142
6	50	10	20	1000	1.455	1.063	284
6	50	10	30	500	2.266	1.173	197
6	50	10	30	1000	2.053	1.183	395
6	50	20	10	500	2.127	1.242	119
6	50	20	10	1000	1.234	0.854	245
6	50	20	20	500	2.441	1.461	180
6	50	20	20	1000	1.432	1.080	345
6	50	20	30	500	1.680	1.216	232
6	50	20	30	1000	1.199	0.873	462
6	50	30	10	500	1.787	1.118	149
6	50	30	10	1000	1.462	1.200	296
6	50	30	20	500	1.769	1.168	205
6	50	30	20	1000	1.737	1.183	411
6	50	30	30	500	1.387	1.053	265
6	50	30	30	1000	1.068	0.854	527
6	100	20	20	500	1.905	1.130	185
6	100	20	20	1000	1.777	1.163	355
6	100	20	40	500	1.557	1.093	302
6	100	20	40	1000	1.434	1.059	604
6	100	20	60	500	2.049	1.207	415
6	100	20	60	1000	2.040	1.156	815
6	100	40	20	500	1.979	1.126	251
6	100	40	20	1000	1.268	0.936	493
6	100	40	40	500	1.894	1.165	376
6	100	40	40	1000	1.787	1.080	745
6	100	40	60	500	1.590	1.183	502
6	100	40	60	1000	1.397	0.885	992
6	100	60	20	500	1.869	1.178	317
6	100	60	20	1000	0.820	0.648	641
6	100	60	40	500	1.406	0.940	446
6	100	60	40	1000	1.411	1.086	887
6	100	60	60	500	1.430	1.061	577
6	100	60	60	1000	1.650	1.248	1142

Table 27 : Function 1 trained by RNN with Tabu Search – data with noise

Function	Add Tenure	Minus Tenure	Avg Training Error	Avg Testing Error	Time (Sec)
1	30	30	0.835	0.578	280
1	30	60	1.209	0.733	278
1	30	90	1.022	0.659	277
1	30	120	0.989	0.630	280
1	30	150	1.013	0.624	361
1	60	30	0.978	0.644	282
1	60	60	1.116	0.689	294
1	60	90	1.217	0.777	292
1	60	120	1.071	0.688	278
1	60	150	1.181	0.760	279
1	90	30	1.035	0.673	279
1	90	60	1.008	0.637	278
1	90	90	1.626	1.039	270
1	90	120	1.141	0.734	278
1	90	150	0.839	0.539	278
1	120	30	0.876	0.554	274
1	120	60	0.966	0.595	278
1	120	90	1.949	1.252	279
1	120	120	0.800	0.509	279
1	120	150	0.930	0.606	278
1	150	30	1.085	0.690	279

Function	Add Tenure	Minus Tenure	Avg Training Error	Avg Testing Error	Time (Sec)
1	150	60	1.201	0.774	279
1	150	90	1.139	0.724	280
1	150	120	1.062	0.675	279
1	150	150	1.099	0.697	264

Table 28 : Function 2 trained by RNN with Tabu Search – data with noise

Function	Add Tenure	Minus Tenure	Avg Training Error	Avg Testing Error	Time (Sec)
2	30	30	0.971	0.557	280
2	30	60	1.164	0.737	278
2	30	90	0.936	0.591	279
2	30	120	2.271	1.543	279
2	30	150	1.175	0.702	286
2	60	30	1.413	0.682	281
2	60	60	1.360	0.689	282
2	60	90	1.209	0.729	280
2	60	120	1.005	0.637	279
2	60	150	0.952	0.587	280
2	90	30	0.948	0.555	272
2	90	60	1.285	0.619	280
2	90	90	1.315	0.658	280
2	90	120	1.100	0.698	280
2	90	150	1.359	0.712	280
2	120	30	0.786	0.492	279
2	120	60	1.467	0.749	279
2	120	90	1.023	0.644	280
2	120	120	2.041	1.407	280
2	120	150	1.359	0.694	280
2	150	30	1.278	0.846	278
2	150	60	1.467	0.977	280
2	150	90	1.429	0.692	280
2	150	120	1.186	0.715	281
2	150	150	1.258	0.796	280

Table 29 : Function 3 trained by RNN with Tabu Search – data with noise

Function	Add Tenure	Minus Tenure	Avg Training Error	Avg Testing Error	Time (Sec)
3	30	30	17.288	13.619	279
3	30	60	20.381	14.644	279
3	30	90	22.081	17.891	278
3	30	120	20.293	14.918	280
3	30	150	18.100	12.871	279
3	60	30	18.562	14.651	279
3	60	60	15.143	11.981	280
3	60	90	20.106	15.824	278
3	60	120	22.122	18.060	279
3	60	150	18.229	14.141	279
3	90	30	19.799	16.234	279
3	90	60	19.677	15.339	279
3	90	90	24.893	18.745	279
3	90	120	21.331	15.390	280
3	90	150	19.727	15.015	281
3	120	30	20.475	14.906	279
3	120	60	20.170	14.606	279
3	120	90	20.071	15.333	278
3	120	120	23.645	18.351	279
3	120	150	20.982	15.356	280
3	150	30	18.179	14.073	280
3	150	60	21.600	15.579	280
3	150	90	18.415	13.259	280
3	150	120	16.549	12.954	280
3	150	150	20.101	15.839	281

Table 30 : Function 4 trained by RNN with Tabu Search – data with noise

Function	Add Tenure	Minus Tenure	Avg Training Error	Avg Testing Error	Time (Sec)
4	30	30	0.759	0.469	279
4	30	60	0.577	0.378	279
4	30	90	0.884	0.590	279
4	30	120	0.674	0.443	280
4	30	150	0.742	0.477	280
4	60	30	0.698	0.445	279
4	60	60	0.771	0.502	255
4	60	90	0.588	0.384	279
4	60	120	0.636	0.371	281
4	60	150	0.635	0.421	280
4	90	30	0.690	0.434	279
4	90	60	0.713	0.425	280
4	90	90	0.895	0.582	281
4	90	120	0.586	0.360	259
4	90	150	0.641	0.420	264
4	120	30	0.995	0.664	287
4	120	60	0.643	0.417	284
4	120	90	0.738	0.488	280
4	120	120	0.634	0.399	280
4	120	150	0.901	0.577	279
4	150	30	0.816	0.538	279
4	150	60	0.664	0.428	280
4	150	90	0.641	0.374	281
4	150	120	0.790	0.513	281
4	150	150	0.621	0.405	280

Table 31 : Function 5 trained by RNN with Tabu Search – data with noise

Function	Add Tenure	Minus Tenure	Avg Training Error	Avg Testing Error	Time (Sec)
5	30	30	9.273	7.676	278
5	30	60	9.208	7.587	278
5	30	90	9.335	7.684	279
5	30	120	9.152	7.328	279
5	30	150	9.232	7.425	246
5	60	30	9.326	7.644	279
5	60	60	9.023	7.577	279
5	60	90	9.061	7.491	280
5	60	120	9.086	7.485	279
5	60	150	8.172	6.367	279
5	90	30	8.539	6.383	279
5	90	60	8.050	6.994	280
5	90	90	8.004	6.370	211
5	90	120	8.510	6.559	280
5	90	150	9.319	7.806	280
5	120	30	9.131	7.224	278
5	120	60	8.680	6.536	279
5	120	90	9.044	7.614	280
5	120	120	8.569	6.774	281
5	120	150	8.039	6.209	215
5	150	30	8.413	6.227	279
5	150	60	8.575	6.434	280
5	150	90	9.022	7.477	280
5	150	120	8.440	6.427	209
5	150	150	8.997	7.409	280

Table 32 : Function 6 trained by RNN with Tabu Search – data with noise

Function	Add Tenure	Minus Tenure	Avg Training Error	Avg Testing Error	Time (Sec)
6	30	30	0.724	0.420	280
6	30	60	1.643	1.000	253
6	30	90	0.590	0.370	280
6	30	120	1.098	0.640	280
6	30	150	1.976	1.140	218
6	60	30	1.110	0.755	280
6	60	60	0.900	0.675	280
6	60	90	1.175	0.749	280
6	60	120	0.778	0.485	280
6	60	150	0.713	0.440	280
6	90	30	2.102	1.173	209
6	90	60	0.758	0.540	280
6	90	90	0.639	0.416	280
6	90	120	0.522	0.372	279
6	90	150	1.130	0.695	281
6	120	30	0.716	0.501	279
6	120	60	1.974	1.145	237
6	120	90	0.570	0.378	280
6	120	120	0.536	0.336	280
6	120	150	1.296	0.802	240
6	150	30	1.121	0.653	280
6	150	60	0.629	0.428	280
6	150	90	0.543	0.377	275
6	150	120	1.157	0.794	281
6	150	150	0.708	0.555	281

Table 33 : Function 1-6 trained by RNN with Back Propagation – data with noise

Function	Epoch	Learning Rate	Avg Training Error	Avg Testing Error	Time (Sec)
1	300	0.75	0.627	0.404	11
1	300	1.25	0.700	0.442	11
1	500	0.75	0.644	0.404	20
1	500	1.25	0.709	0.432	20
2	300	0.75	3.346	2.086	11
2	300	1.25	3.081	2.037	11
2	500	0.75	2.818	1.917	18
2	500	1.25	2.612	1.715	19
3	300	0.75	37.203	25.867	14
3	300	1.25	38.381	26.435	9
3	500	0.75	37.195	25.877	23
3	500	1.25	38.382	27.632	16
4	300	0.75	1.359	0.873	10.4
4	300	1.25	1.258	0.821	10.4
4	500	0.75	1.284	0.858	18.3
4	500	1.25	1.386	0.893	18.5
5	300	0.75	14.210	9.491	11
5	300	1.25	13.709	9.249	11.1
5	500	0.75	13.975	9.082	19
5	500	1.25	13.533	8.608	19
6	300	0.75	0.556	0.439	14
6	300	1.25	0.433	0.324	15
6	500	0.75	0.426	0.319	20.4
6	500	1.25	0.319	0.237	20.1

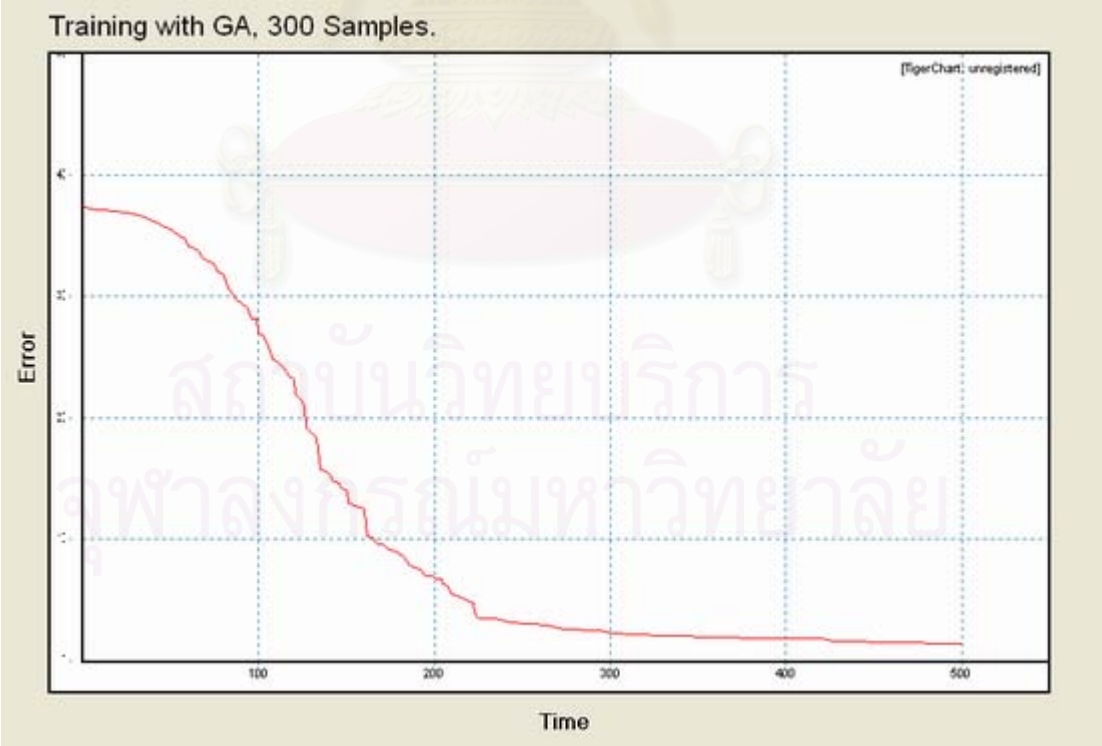


Figure 15 : Graphical representation of error in function 1 trained with Genetic Algorithm

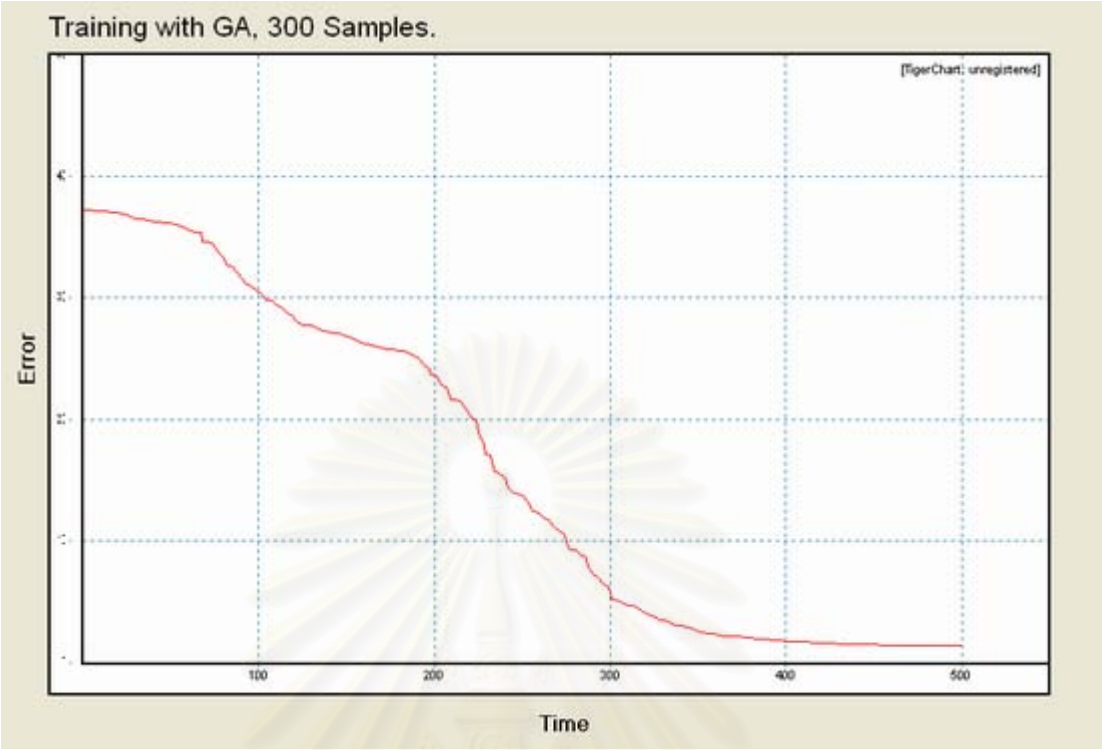


Figure 16 : Graphical representation of error in function 2 trained with Genetic Algorithm

Figure 17 : Graphical representation of error in function 3 trained with Genetic Algorithm

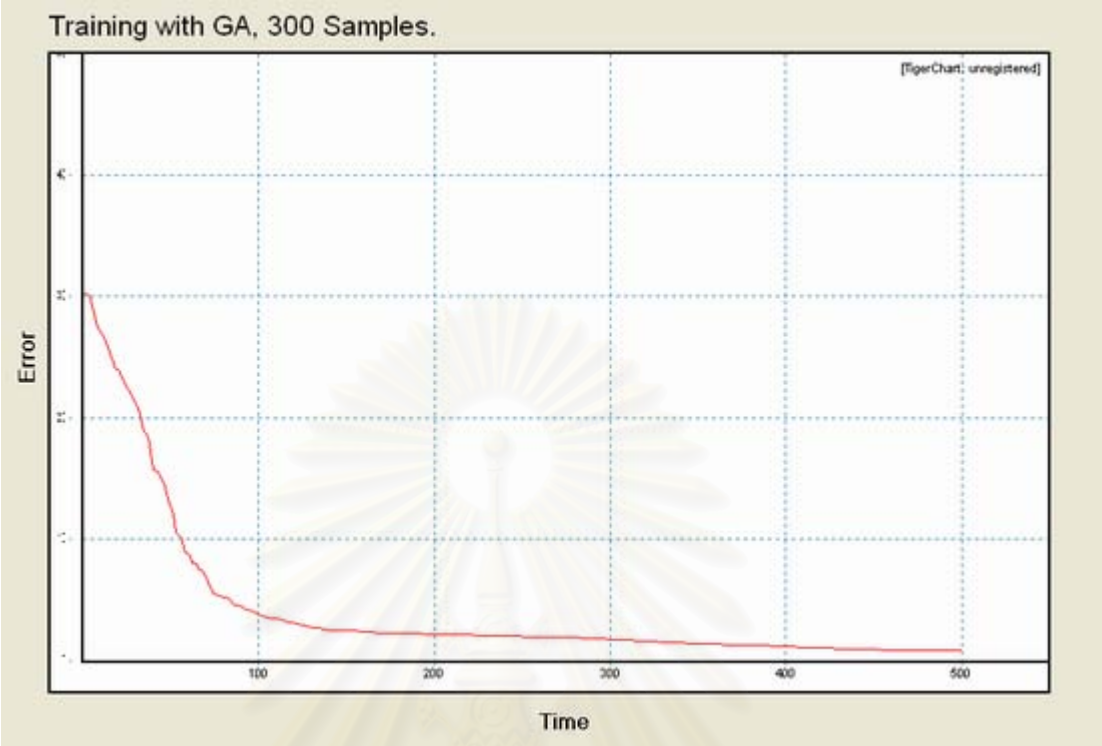


Figure 18 : Graphical representation of error in function 4 trained with Genetic Algorithm

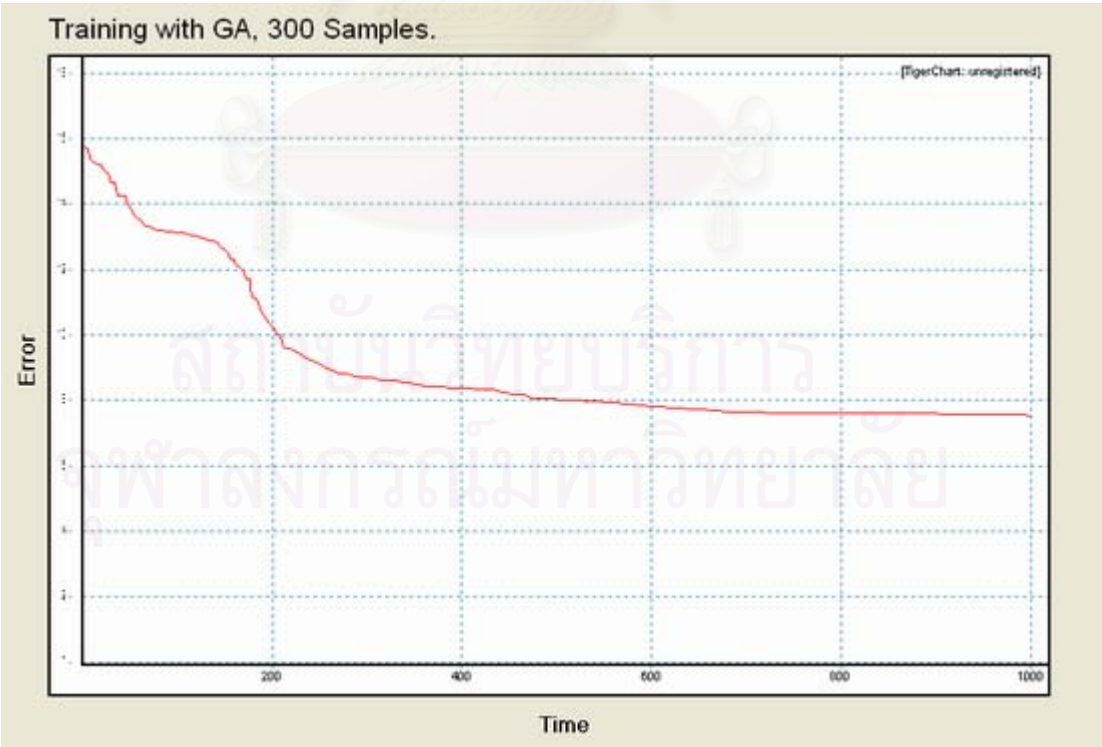


Figure 19 : Graphical representation of error in function 5 trained with Genetic Algorithm

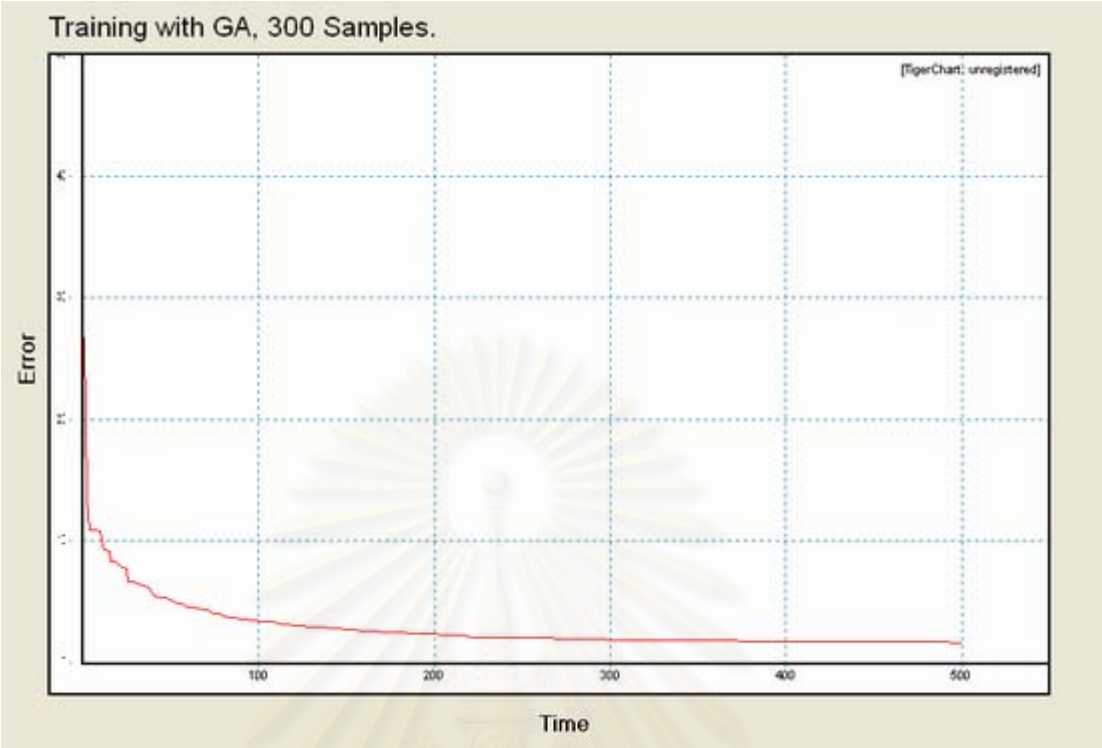


Figure 20 : Graphical representation of error in function 6 trained with Genetic Algorithm

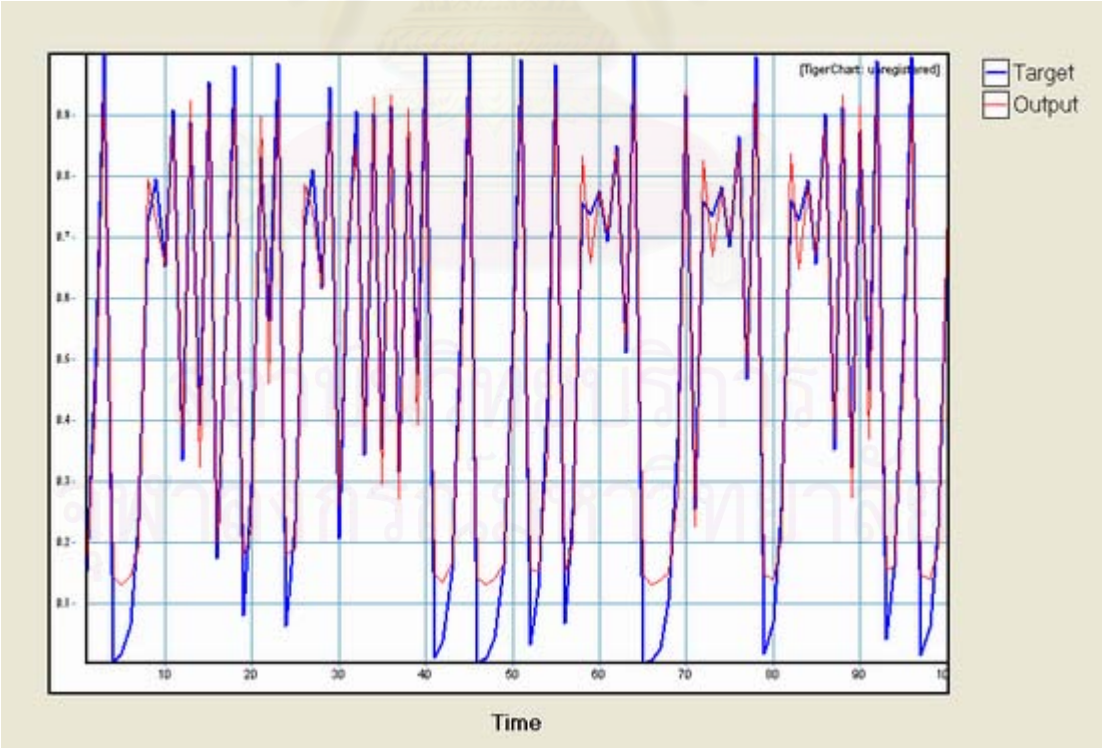


Figure 21 : Graphical representation of function 1 trained with Genetic Algorithm

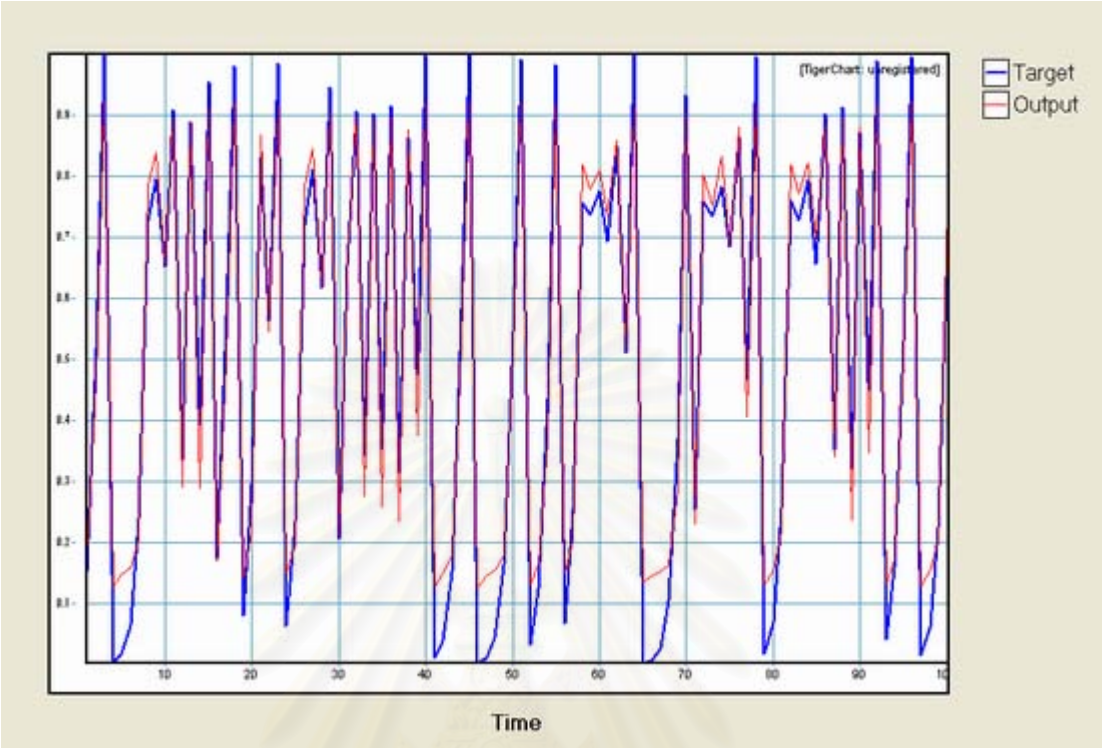


Figure 22 : Graphical representation of function 2 trained with Genetic Algorithm

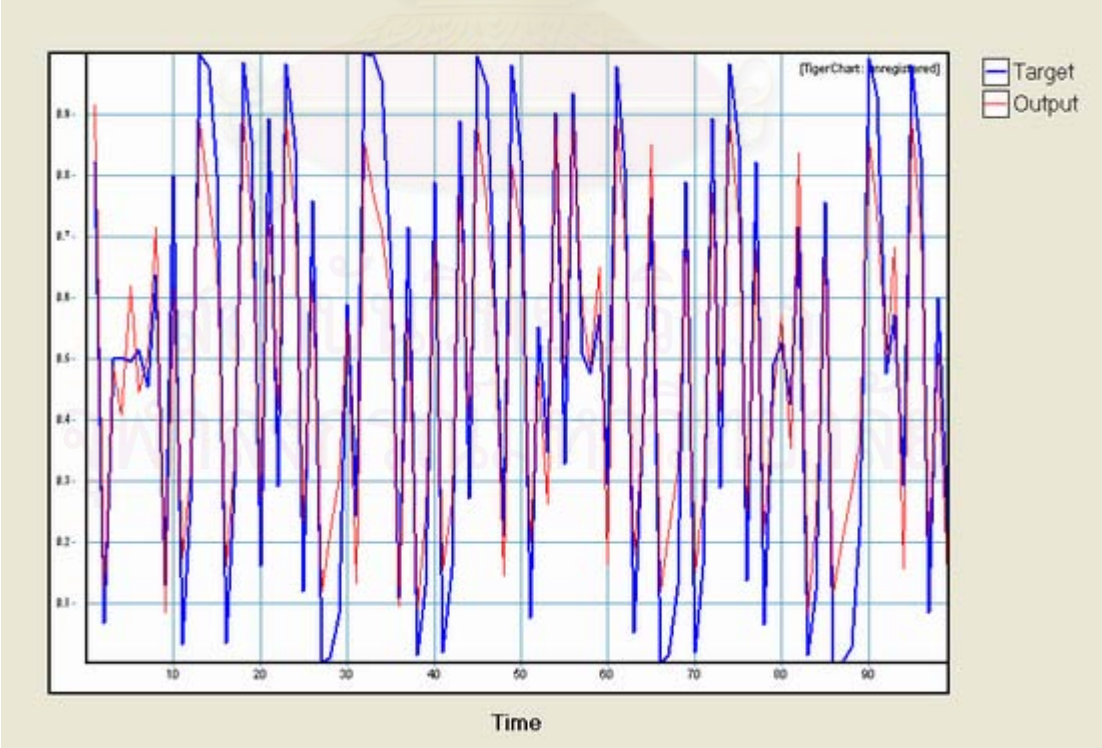


Figure 23 : Graphical representation of function 3 trained with Genetic Algorithm

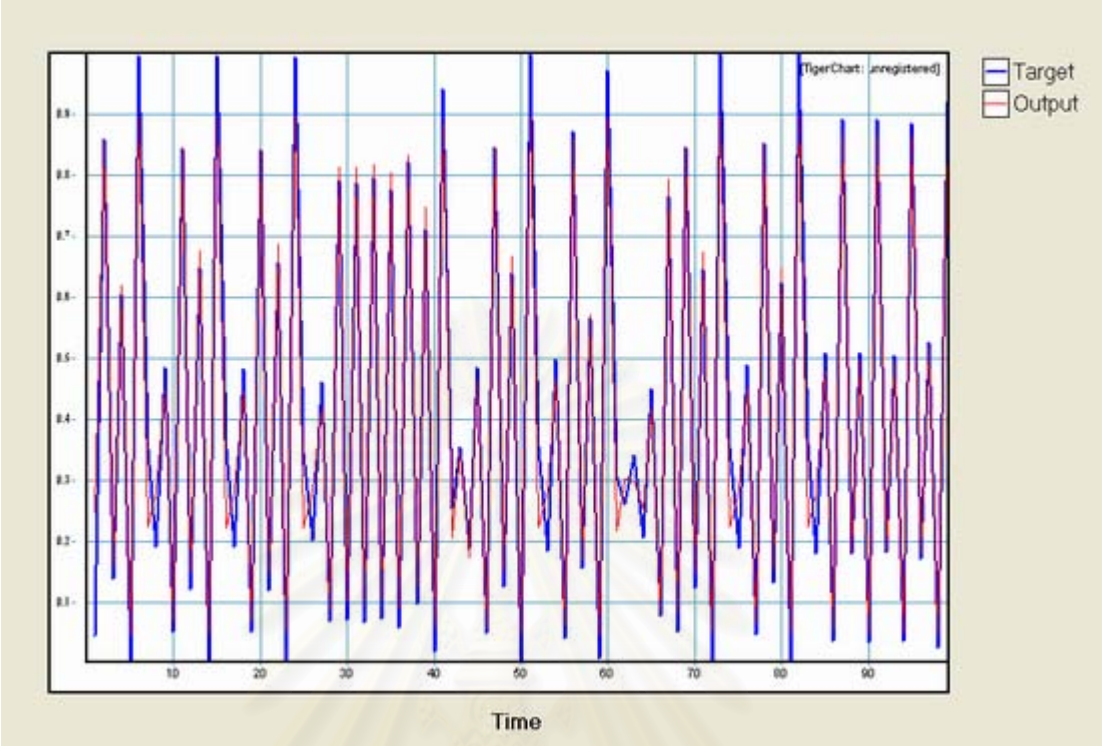


Figure 24 : Graphical representation of function 4 trained with Genetic Algorithm

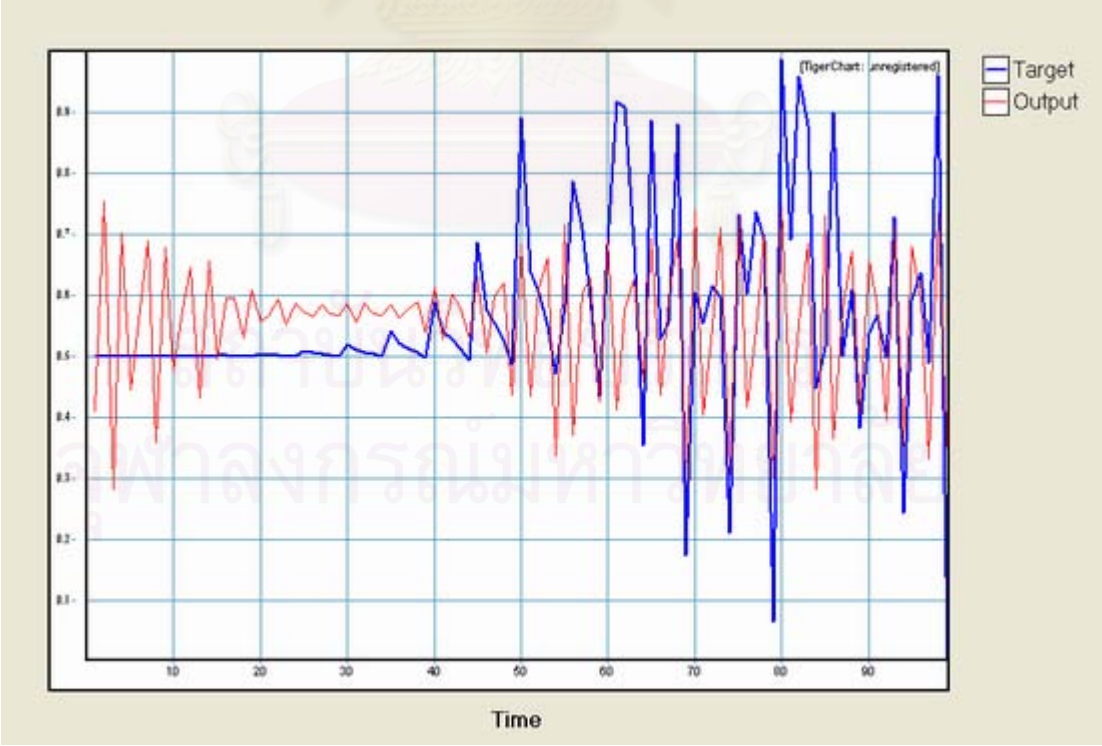


Figure 25 : Graphical representation of function 5 trained with Genetic Algorithm

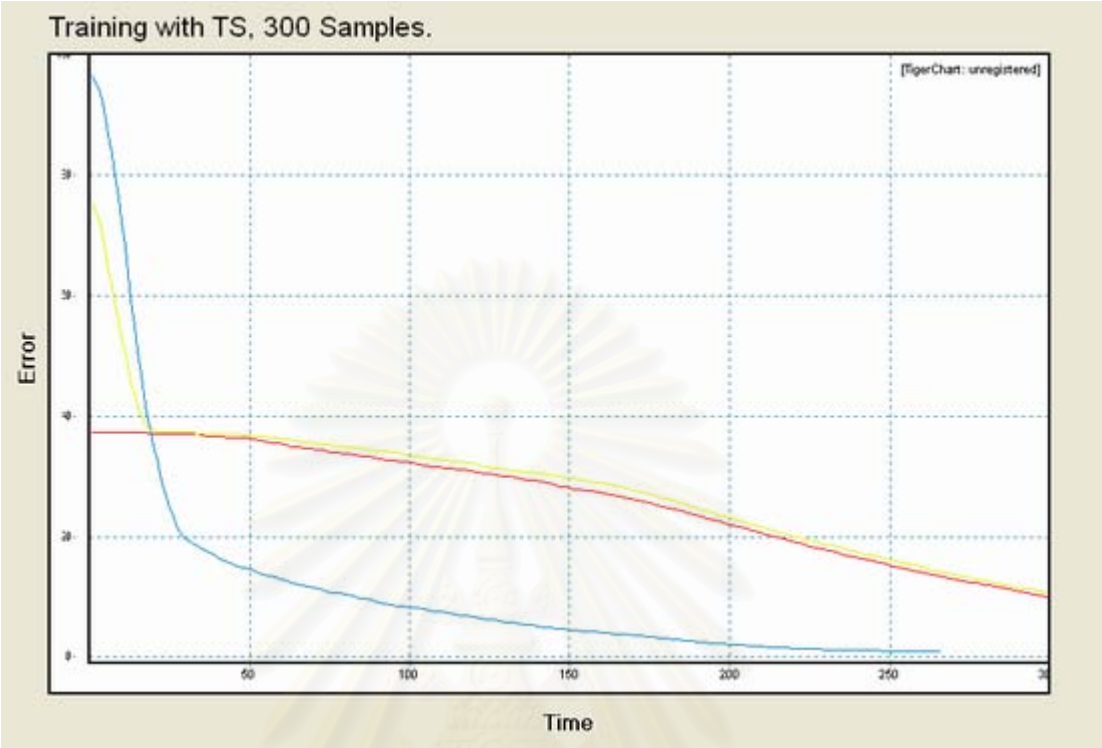


Figure 26 : Graphical representation of error in function 1 trained with Tabu Search

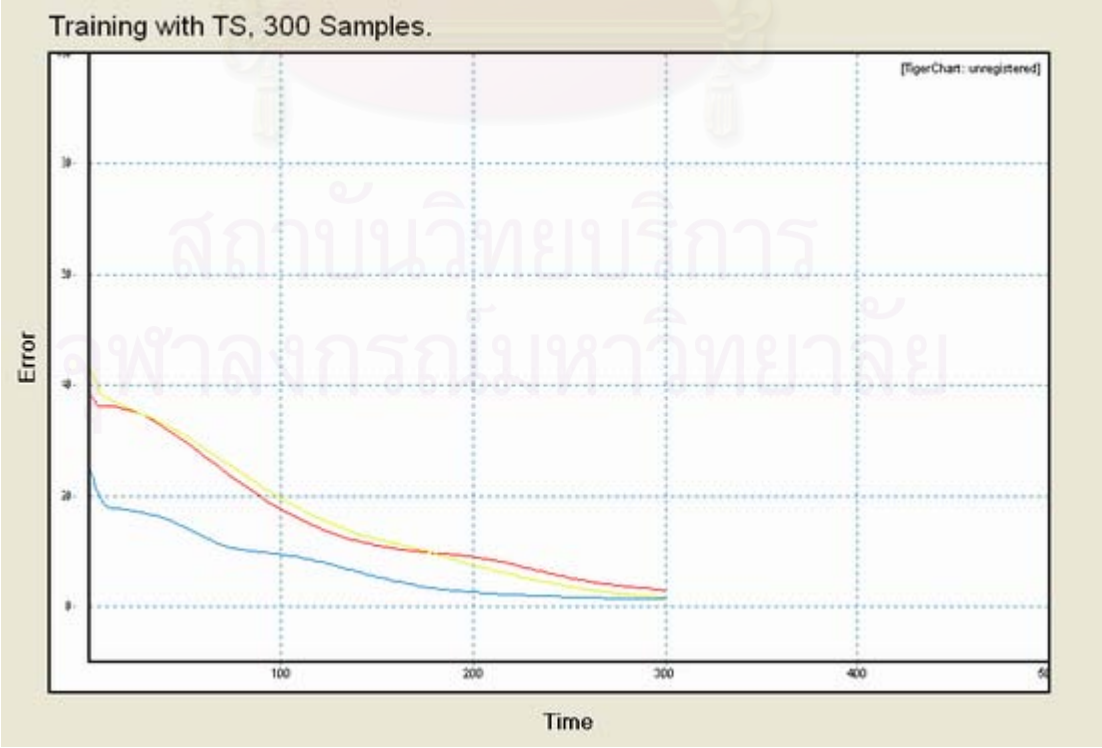


Figure 27 : Graphical representation of error in function 2 trained with Tabu Search

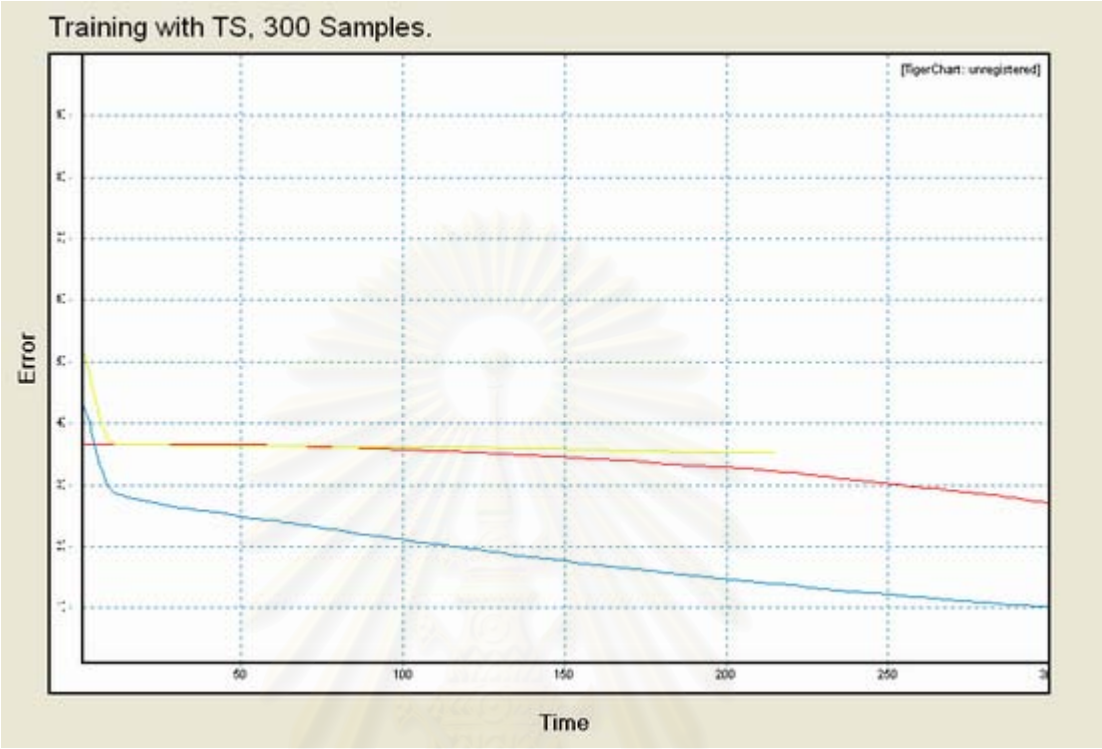


Figure 28 : Graphical representation of error in function 3 trained with Tabu Search

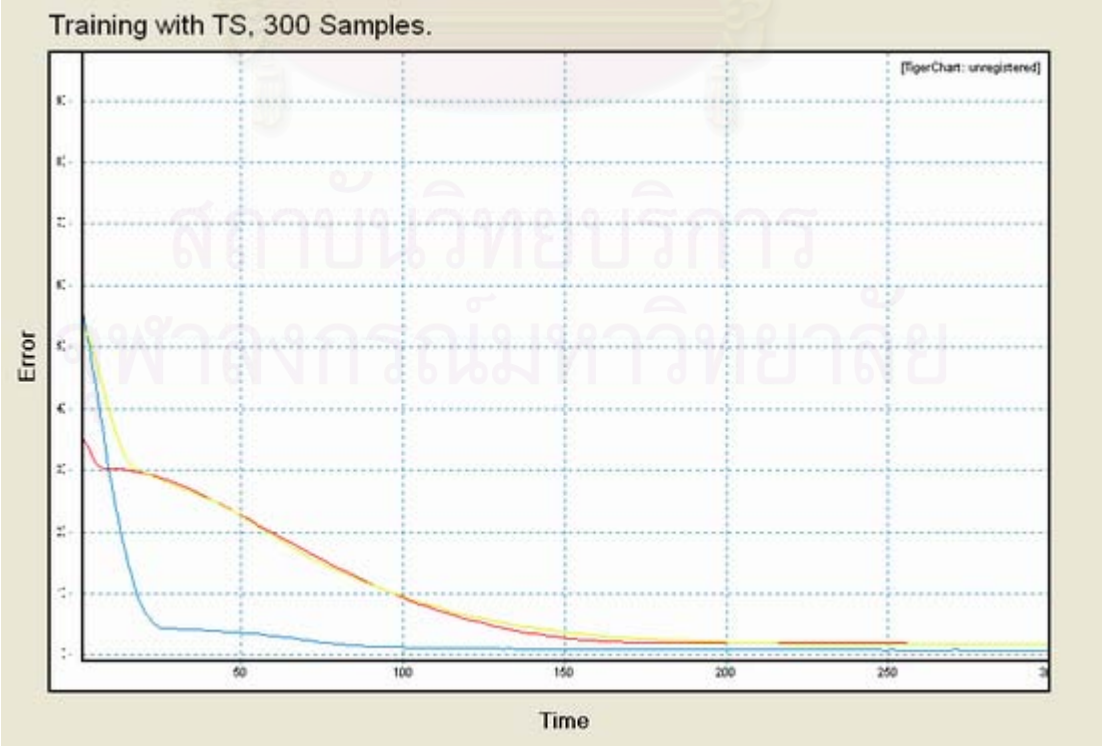


Figure 29 : Graphical representation of error in function 4 trained with Tabu Search

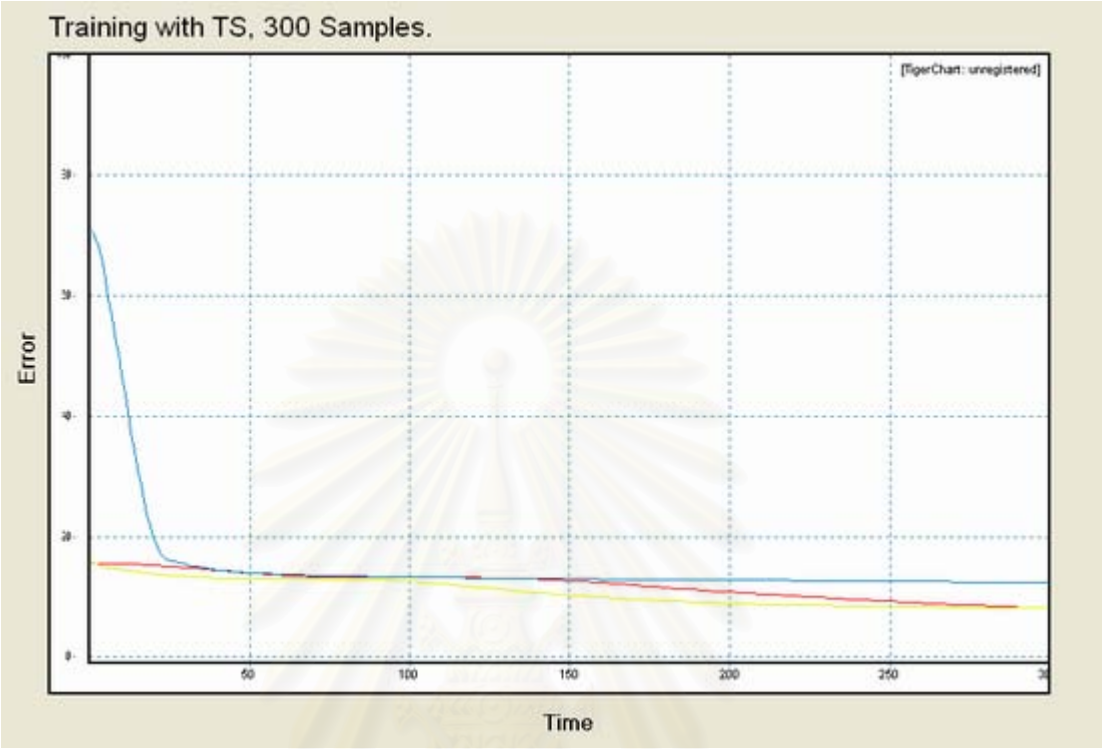


Figure 30 : Graphical representation of error in function 5 trained with Tabu Search

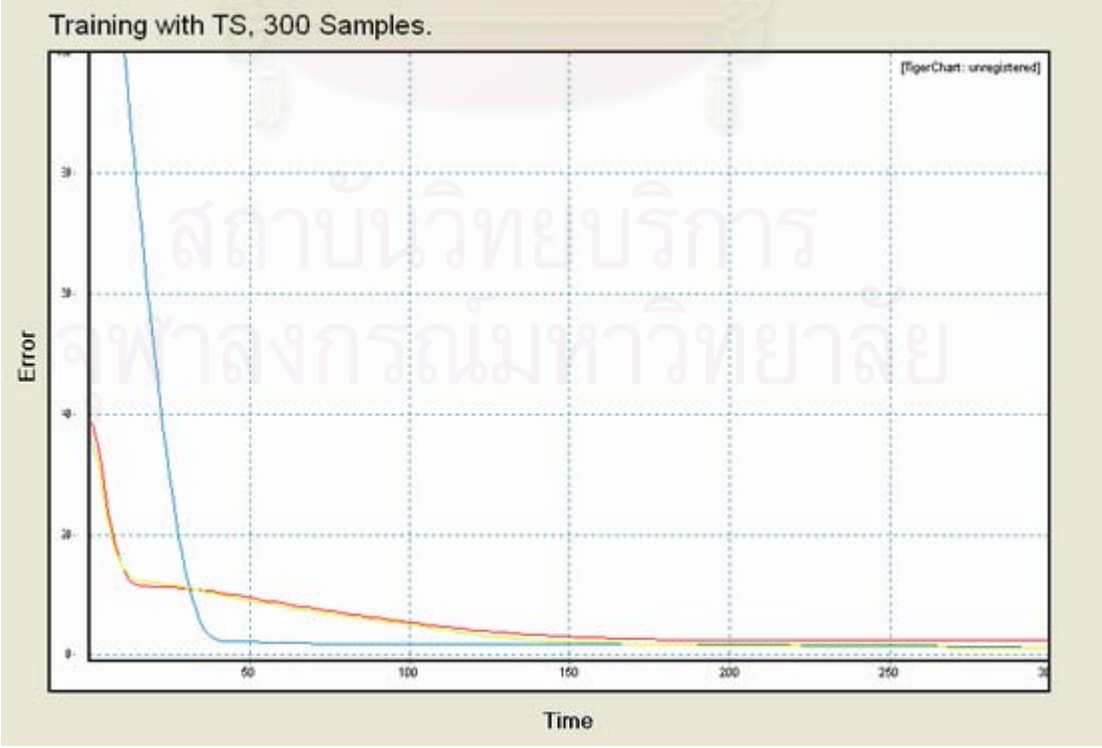


Figure 31 : Graphical representation of error in function 6 trained with Tabu Search

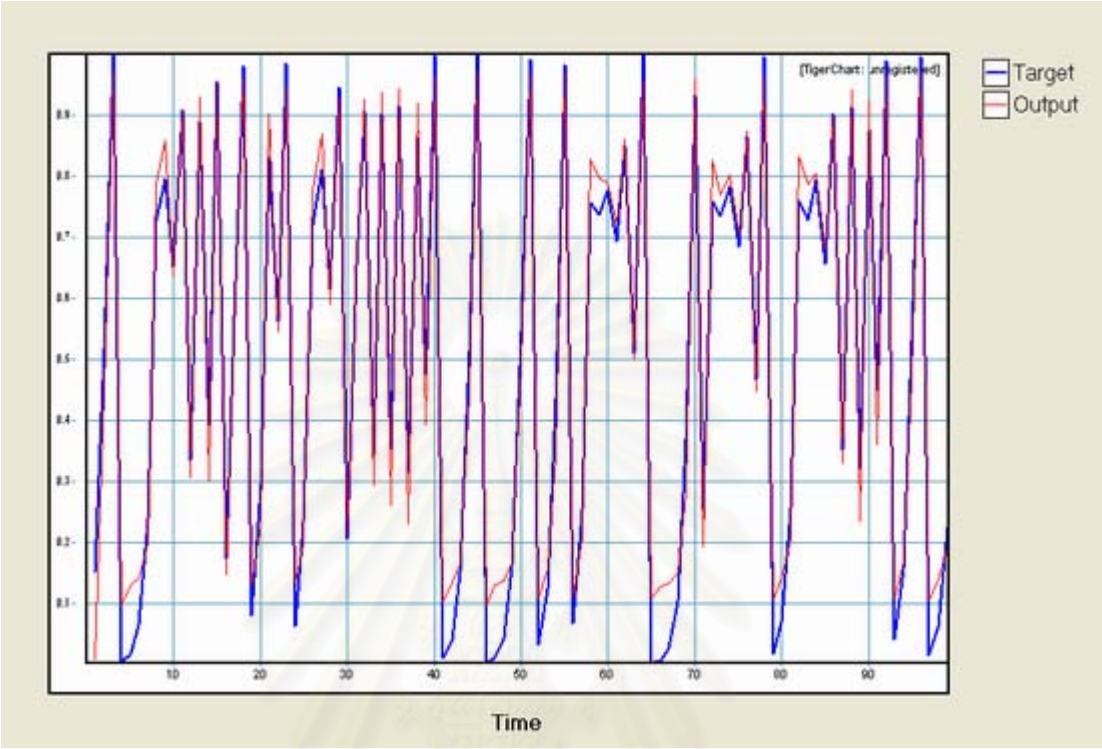


Figure 32 : Graphical representation of function 1 trained with Tabu Search

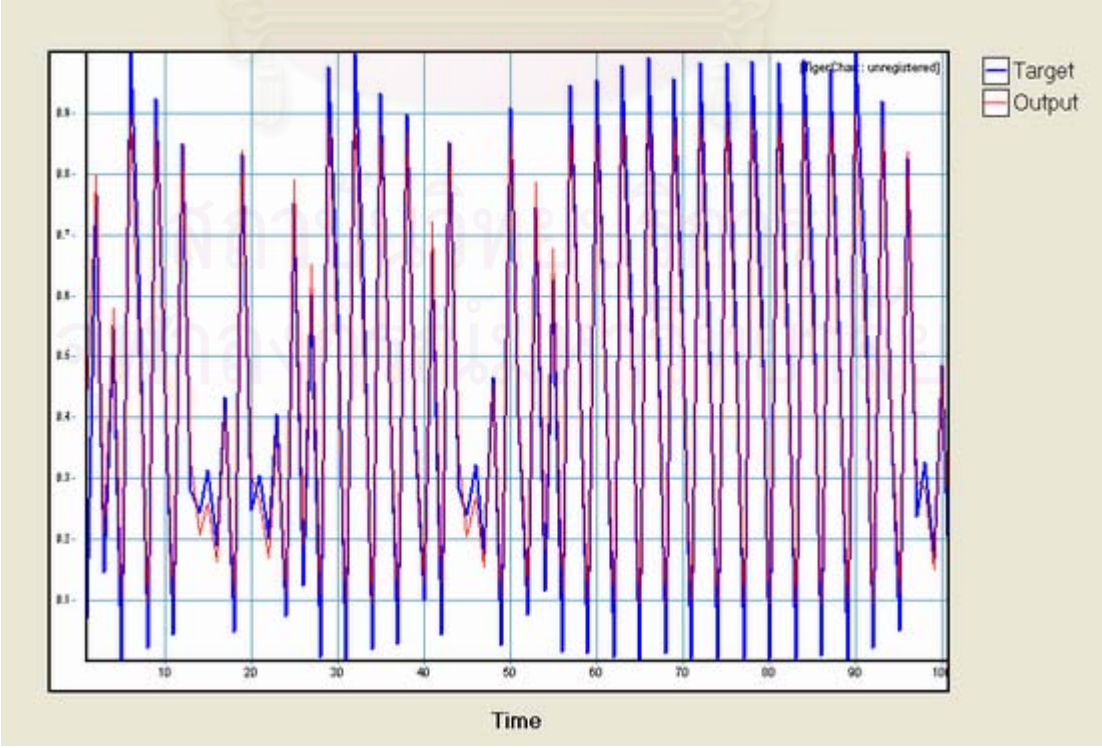


Figure 33 : Graphical representation of function 2 trained with Tabu Search

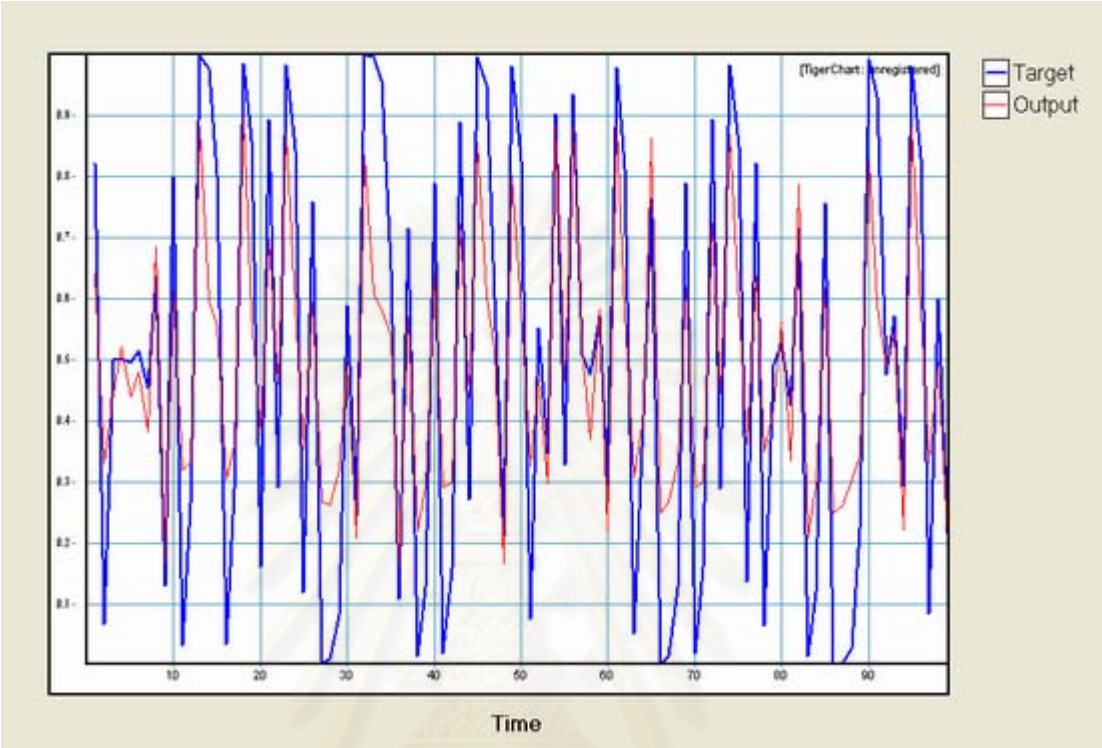


Figure 34 : Graphical representation of function 3 trained with Tabu Search

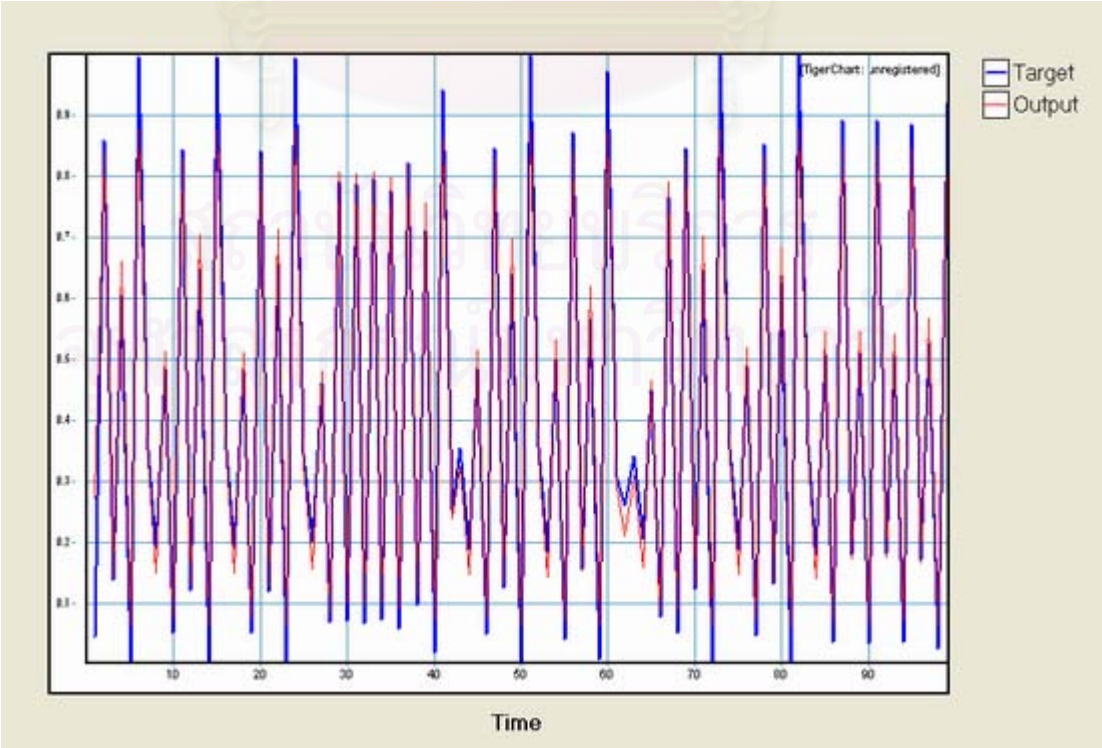


Figure 35 : Graphical representation of function 4 trained with Tabu Search

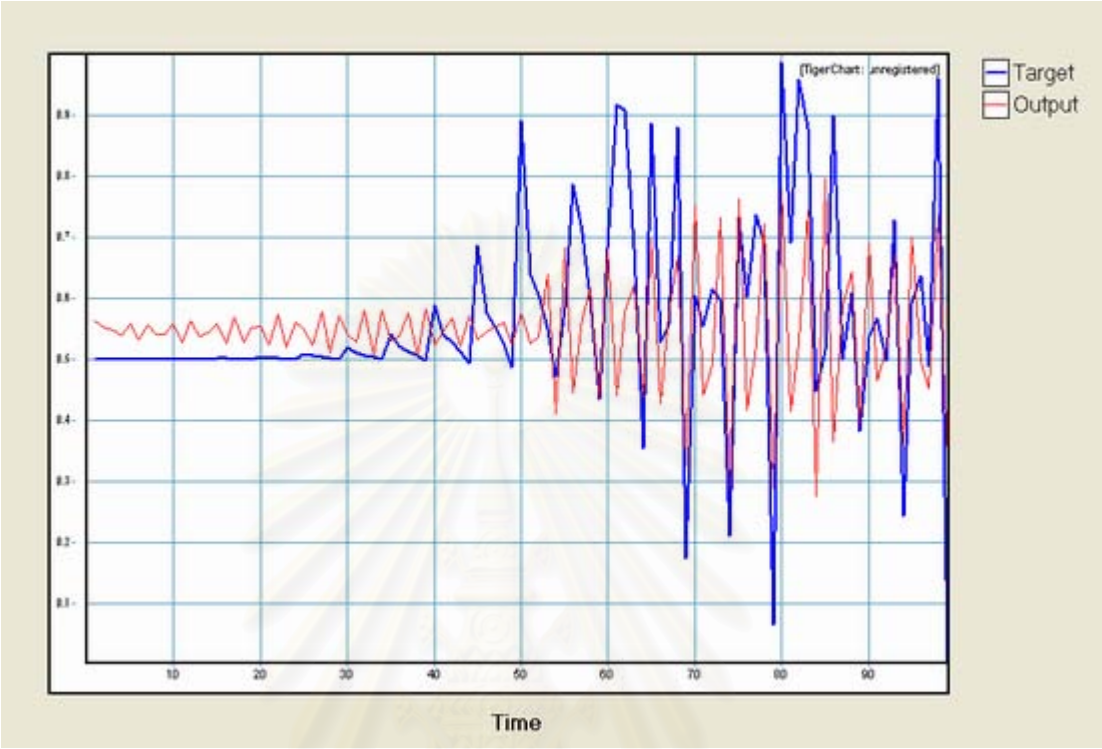


Figure 36 : Graphical representation of function 5 trained with Tabu Search

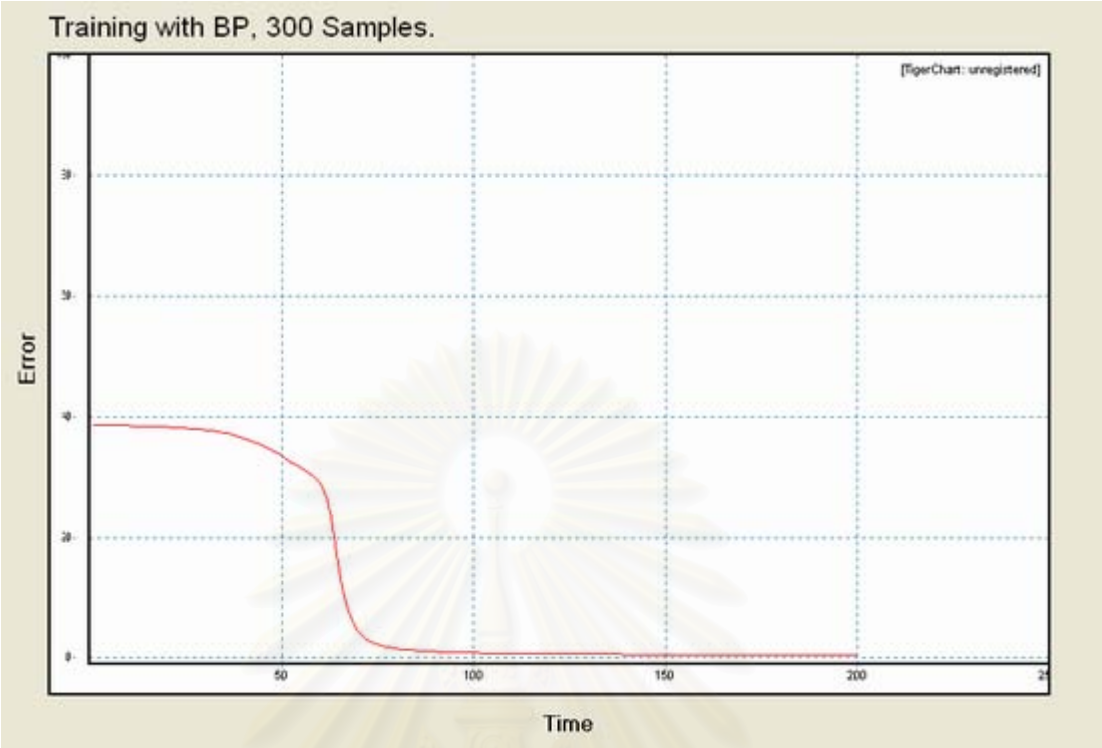


Figure 37 : Graphical representation of error in function 1 trained with Backpropagation

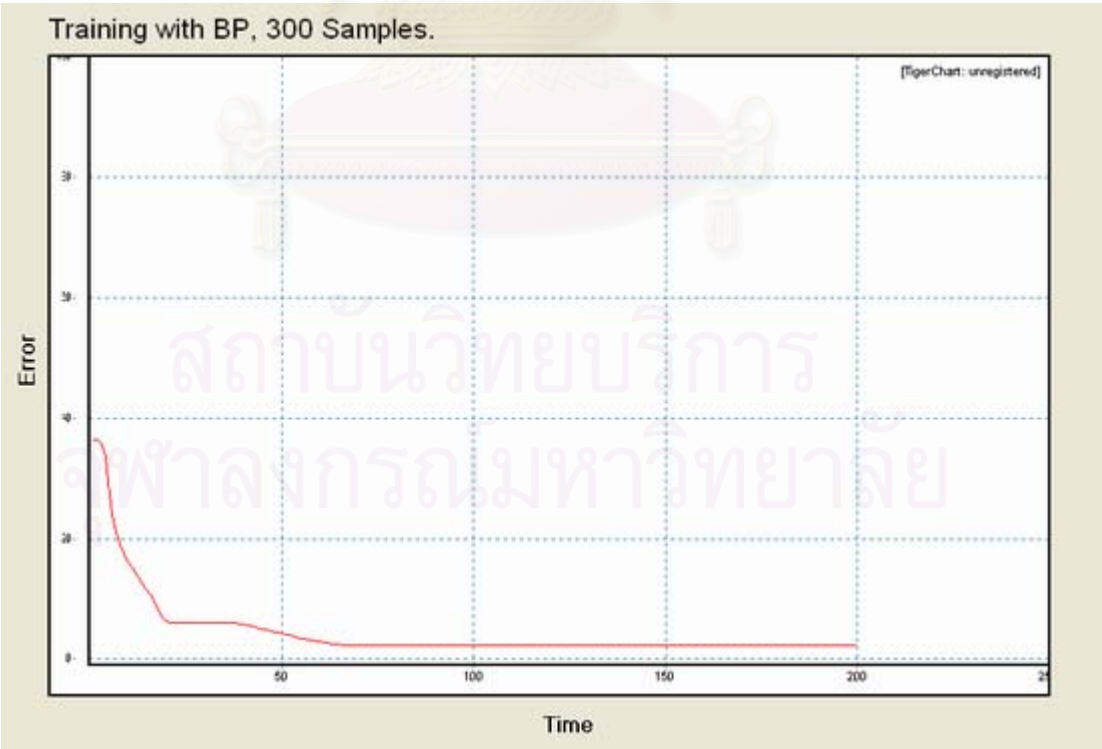


Figure 38 : Graphical representation of error in function 2 trained with Backpropagation

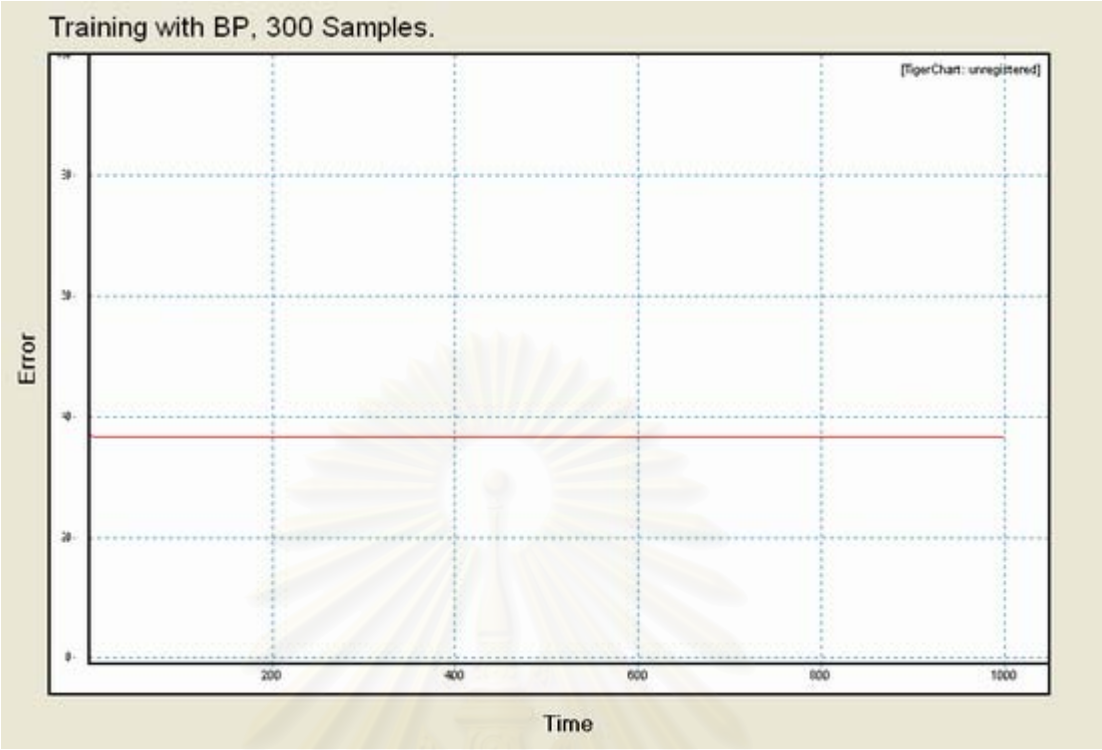


Figure 39 : Graphical representation of error in function 3 trained with Backpropagation

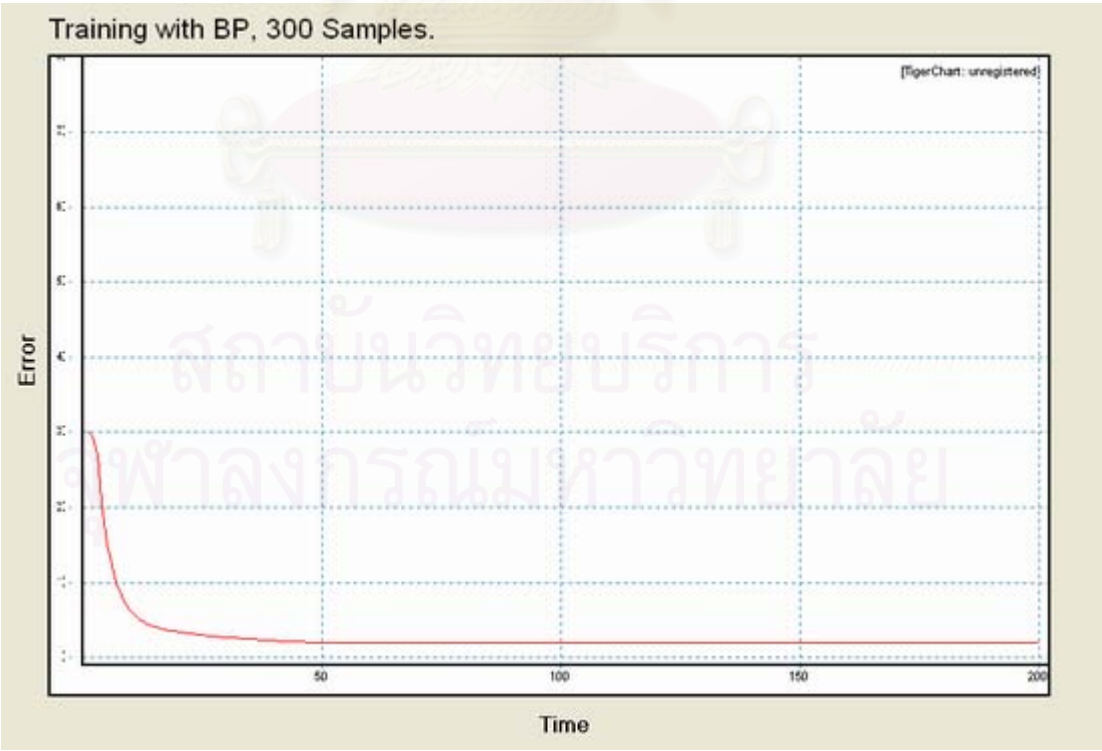


Figure 40 : Graphical representation of error in function 4 trained with Backpropagation

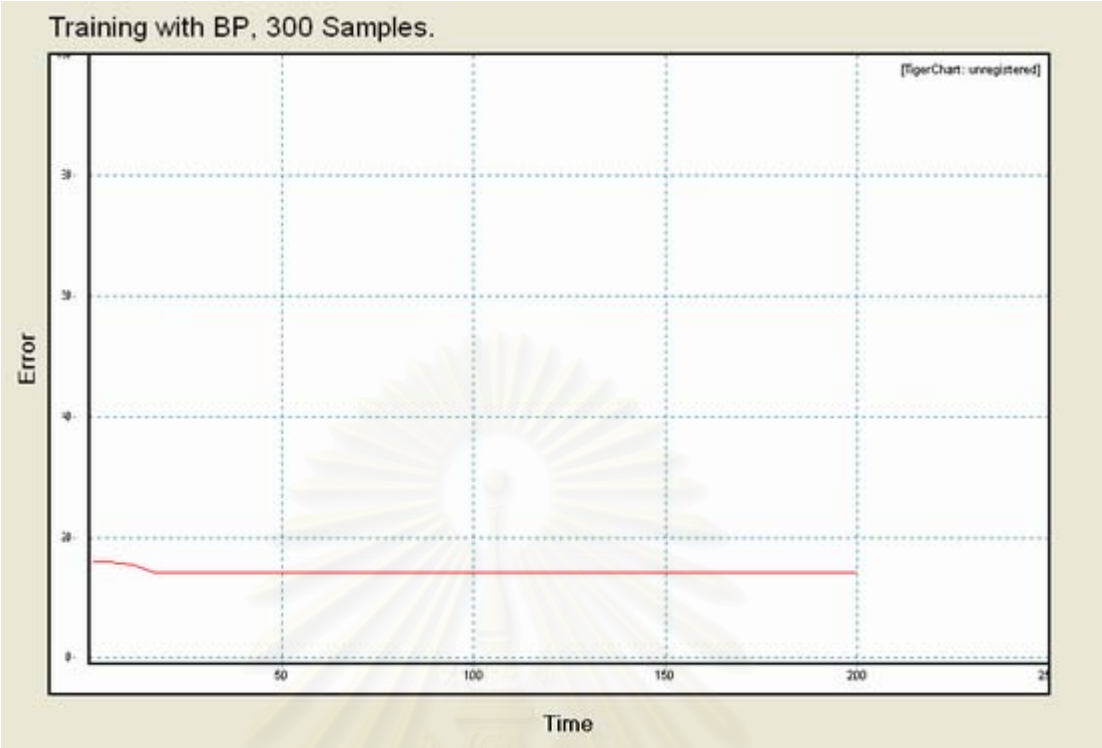


Figure 41 : Graphical representation of error in function 5 trained with Backpropagation

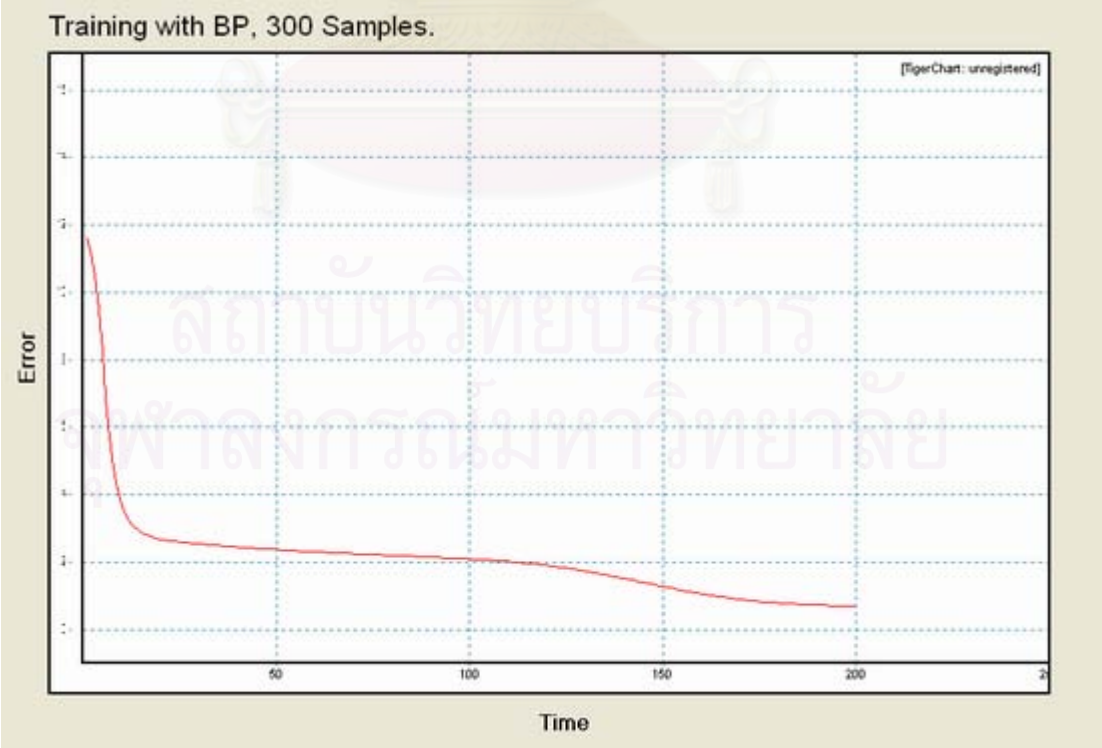


Figure 42 : Graphical representation of error in function 6 trained with Backpropagation

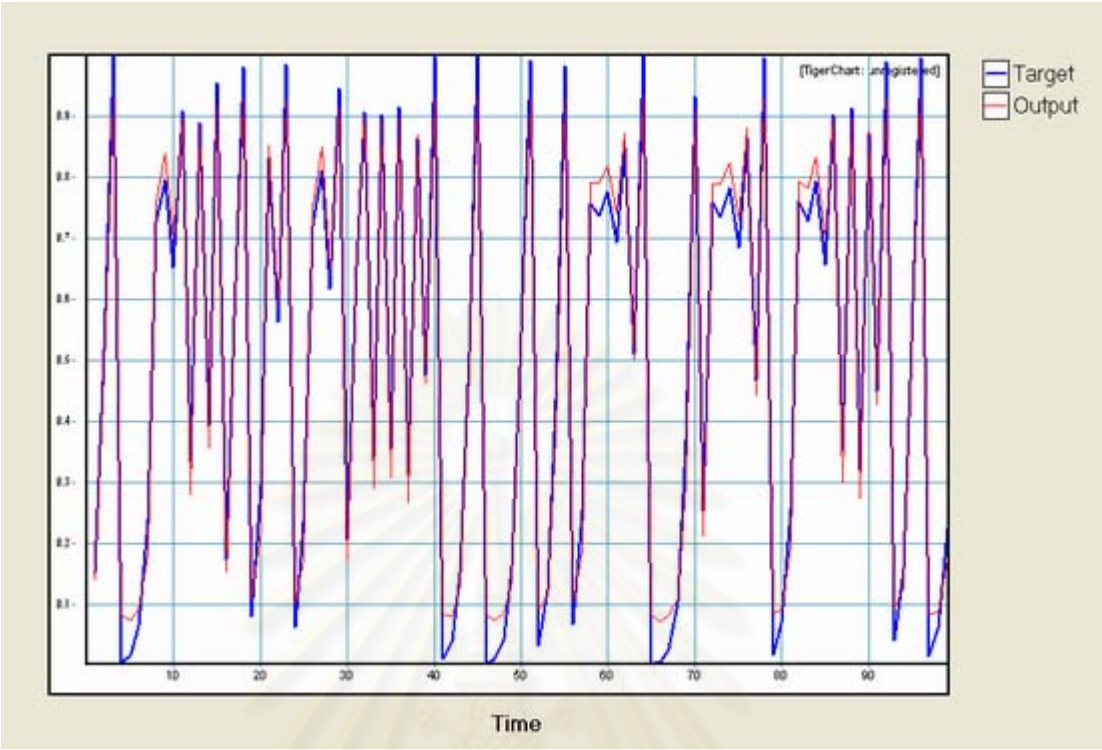


Figure 43 : Graphical representation of function 1 trained with Backpropagation

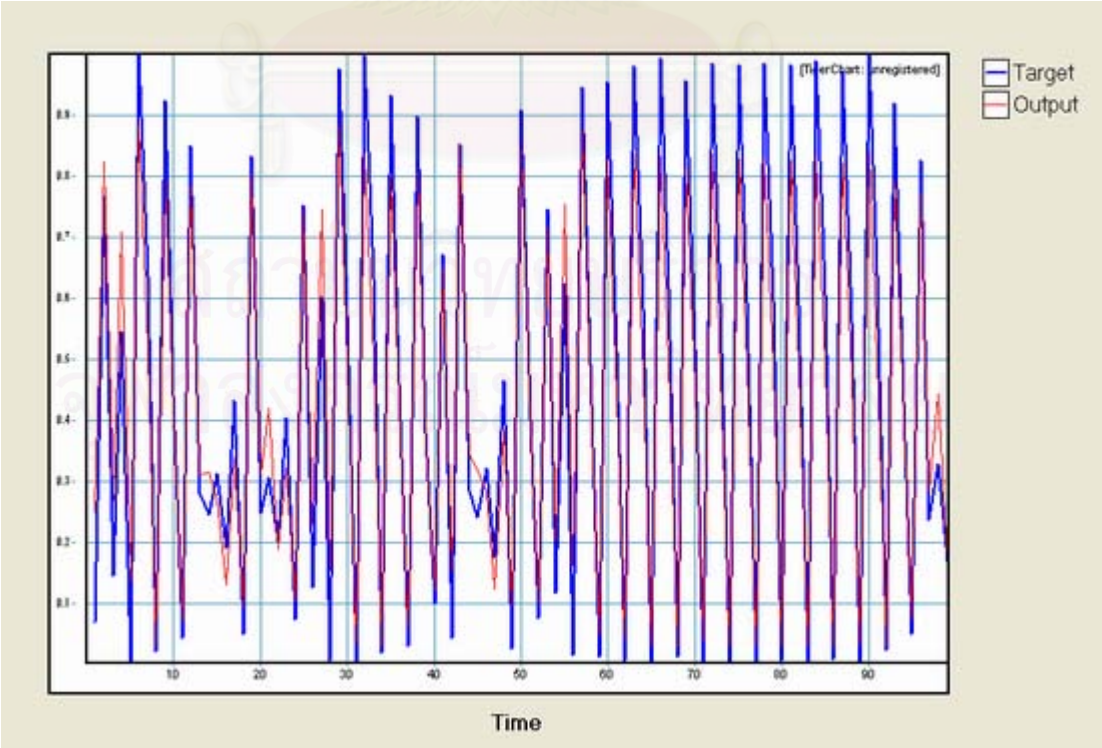


Figure 44 : Graphical representation of function 2 trained with Backpropagation

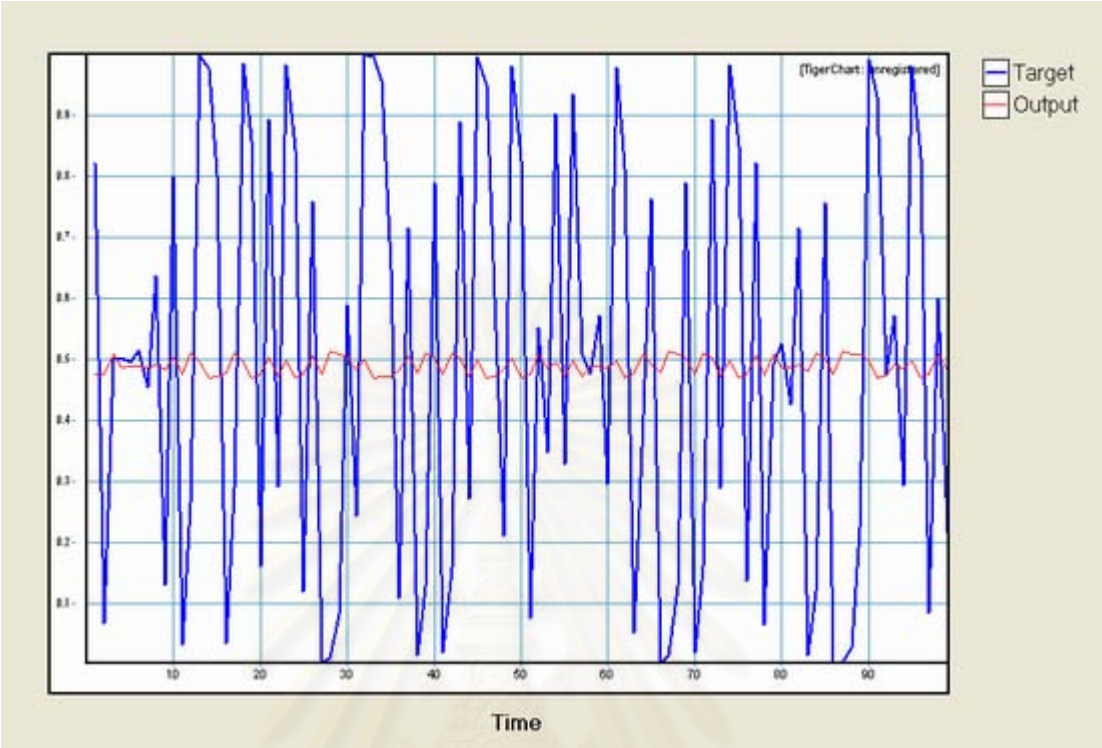


Figure 45 : Graphical representation of function 3 trained with Backpropagation

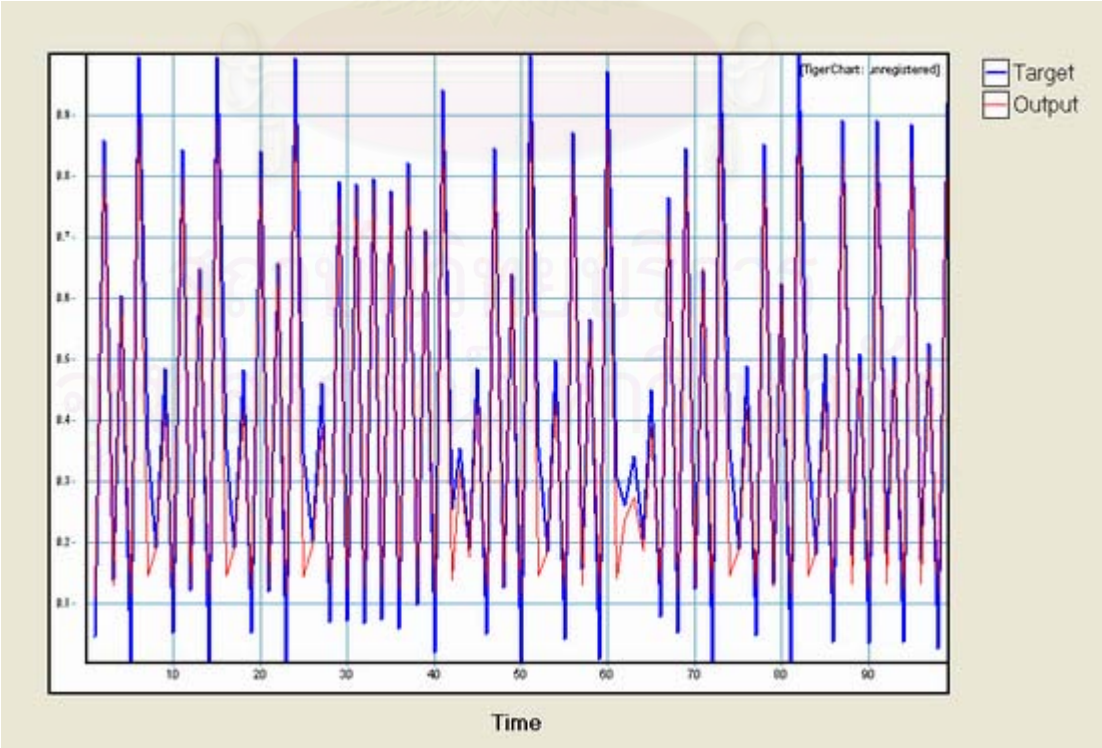


Figure 46 : Graphical representation of function 4 trained with Backpropagation

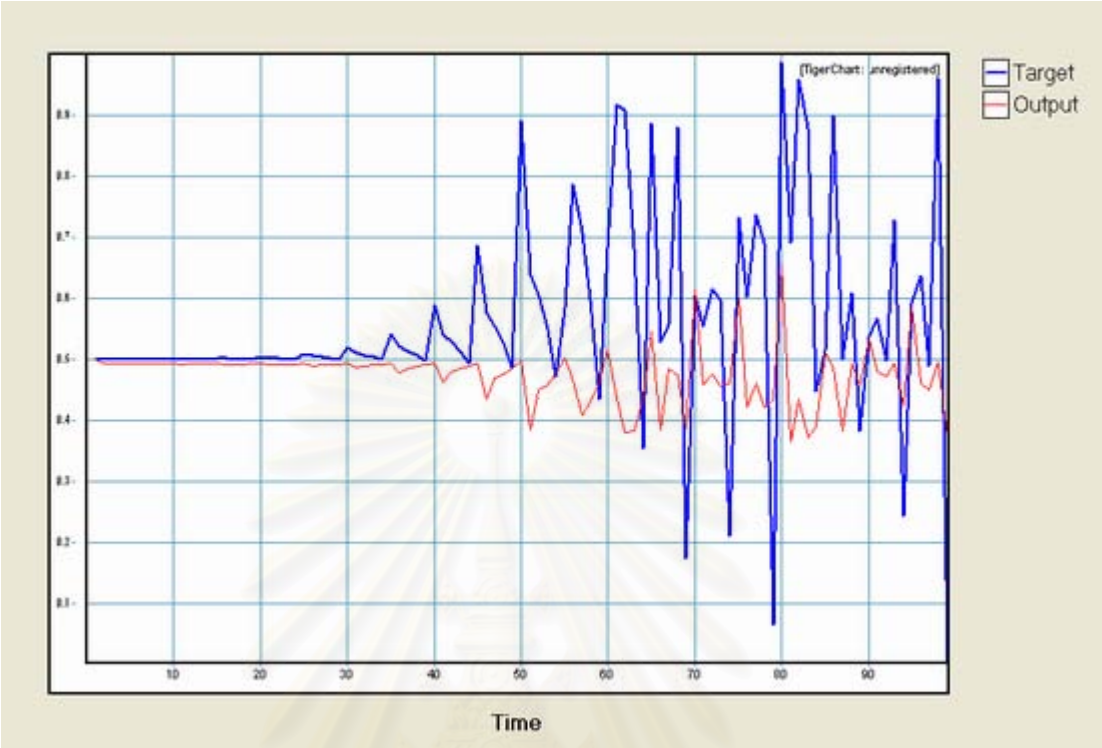


Figure 47 : Graphical representation of function 5 trained with Backpropagation

BIOGRAPHY

Mr. Kittikorn Tongnimitsawat was born in Phayao of Thailand on April 29, 1974. He had high school education at The Prince Royal's College, Chiangmai. He also graduated in Accounting as a Bachelor Degree as well as Master of Business Administration as Master Degree at Faculty of Business Administration of Chiangmai University.