8-2007

Personalized Health Monitoring Using Evolvable Block-based Neural Networks

Wei Jiang
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Wei Jiang entitled "Personalized Health Monitoring Using Evolvable Block-based Neural Networks." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Seong G. Kong, Major Professor

We have read this dissertation and recommend its acceptance:

Donald W. Bouldin, Itamar Elhanany, J. Wesley Hines, Gregory D. Peterson

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Wei Jiang entitled "Personalized Health Monitoring Using Evolvable Block-based Neural Networks." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Dr. Seong G. Kong

Major Professor

We have read this thesis
and recommend its acceptance:

Dr. Donald W. Bouldin

Dr. Itamar Elhanany

Dr. J. Wesley Hines

Dr. Gregory D. Peterson

Accepted for the Council:

Carolyn Hodges

Vice Provost and Dean of Graduate School

(Original signatures are on file with official student records)

PERSONALIZED HEALTH MONITORING USING EVOLVABLE BLOCK-BASED NEURAL NETWORKS

A Dissertation

Presented for the

Doctor of Philosophy Degree

The University of Tennessee, Knoxville

Wei Jiang

Aug. 2007

Dedication

This dissertation is dedicated to my parents and my wife.

Acknowledgements

I would like to thank all the individuals who have helped me in one way or another in finishing this dissertation.

I am deeply indebted to my academic advisor Dr. Seong G. Kong for his guidance and consistent support throughout my graduate study at University of Tennessee, and for giving me the opportunity to work on the block-based neural network project. I also wish to thank Dr. Gregory D. Peterson for the invaluable discussions and for serving on my committee. I am very grateful for the advice and service of my other committee members, Dr. Donald W. Bouldin, Dr. J. Wesley Hines and Dr. Itamar Elhanany.

I would like to acknowledge Saumil Merchant for clarifying me on various FPGA issues and for general discussions on BbNN research, and Sangwoo Moon for the valuable discussions on research and programming and for reviewing my papers. I also want to thank other colleagues: Sang Ki Park, Shaoyu Liu, Zheng Du, Yang Bai, and Oh Kyu Kwon.

Last but not least, my thanks go to my wife Lidan, and my family members for their love, support and encouragement through my life.

Abstract

This dissertation presents personalized health monitoring using evolvable block-based neural networks. Personalized health monitoring plays an increasingly important role in modern society as the population enjoys longer life. Personalization in health monitoring considers physiological variations brought by temporal, personal or environmental differences, and demands solutions capable to reconfigure and adapt to specific requirements. Block-based neural networks (BbNNs) consist of 2-D arrays of modular basic blocks that can be easily implemented using reconfigurable digital hardware such as field programmable gate arrays (FPGAs) that allow on-line partial reorganization. The modular structure of BbNNs enables easy expansion in size by adding more blocks. A computationally efficient evolutionary algorithm is developed that simultaneously optimizes structure and weights of BbNNs. This evolutionary algorithm increases optimization speed by integrating a local search operator. An adaptive rate update scheme removing manual tuning of operator rates enhances the fitness trend compared to pre-determined fixed rates. A fitness scaling with generalized disruptive pressure reduces the possibility of premature convergence. The BbNN platform promises an evolvable solution that changes structures and parameters for personalized health monitoring. A BbNN evolved with the proposed evolutionary algorithm using the Hermite transform coefficients and a time interval between two neighboring R peaks of ECG signal, provides a patient-specific ECG heartbeat classification system. Experimental results using the MIT-BIH Arrhythmia database demonstrate a potential for significant performance enhancements over other major techniques.

Table of Contents

List of Figures

List of Tables

Chapter 1 INTRODUCTION

1.1 Motivation and Research Goals

With the world population enjoying longer life, personalized health monitoring for old people capable of early detection of abnormal conditions becomes increasingly important. Also, people working in dangerous environments (e.g. military personnel, firefighters, and over-sized vehicle drivers) benefit from continuous monitoring of health conditions for prediction of various dangerous states such as losing consciousness and heart infarct. Personalization is essential in health monitoring applications in the means that patient-specific situation such as history, gender and age usually need to be considered in making medical decisions. In addition, physiological variations exist due to temporal or environmental differences. Personalized health monitoring considers the variations among patients or patient groups and demands solutions that can reconfigure and adapt to specific needs. Various measurements can be utilized in providing health monitoring including ECG, EKG, respiration rate, blood pressure and so on.

Electrocardiogram (ECG) has become an important routine clinic practice to monitoring heart activities. According to American Heart Association, cardiovascular disease (CVD) caused deaths account for 38% of the total deaths in United States in 2003 [1]. Since 1900, every year CVD caused more deaths than other forms of sources including cancer and accidents except 1918. Continuous monitoring of heart conditions provides quick alarms for emergency rescue and thereby helps reduce the risk of sudden cardiac death. Heart monitoring is especially important for older people or patients who have survived cardiac arrest, ventricular tachycardia, or cardiac syncope.

A unique property of ECG signals lies in its big variation among different situations. ECG signals show great difference for different individuals. Even for the

same individual, heartbeat patterns significantly change with time of the day and under different situations. While normal sinus rhythm originates from the sinus node of heart, arrhythmias have various origins and indicate a wide variety of heart problems. Under different situations, same symptoms of arrhythmia produce different morphologies due to their origins such as premature ventricular contraction (PVC) [2][3].

A possible solution to tackle the big variations in ECG signals is to use a huge set of dataset that include as much as possible representative heartbeat samples, to train a classifier and then use the trained classifier to classify the unseen data. However, a classifier trained for a large set of training data will inevitably need a very large size in order to consider numerous exceptions brought by the large size of the training data. It is also difficult to train and generalize a classifier with a large size using a large set of training data.

Block-based neural networks (BbNNs) [4] consist of a two-dimensional (2-D) array of modular basic blocks. BbNNs have structures that can be easily implemented using reconfigurable digital hardware such as field programmable gate arrays (FPGAs) that allow on-line partial reorganization of internal structures due to modular characteristics of BbNN architecture and simultaneous optimization of structure and weights. The modular structure of BbNNs enables easy expansion in size by adding more blocks.

Evolvable classifiers based on block-based neural networks change the structure and configurations as well as internal parameters to cope with the heartbeat variations due to personal or temporal differences, and have demonstrated a potential for performance improvement over conventional techniques for ECG signal classification [5][6][7][8].

The main objective of this dissertation is to demonstrate the unique capabilities of the BbNN platform in personalized health monitoring where the dynamic nature of the problem needs an evolvable solution to tackle the changes in operating environments. Example of target applications includes personalized ECG heartbeat classification.

Another goal in this dissertation is to design an evolutionary algorithm to optimize simultaneously the structure and weights of block-based neural networks. The previous evolutionary algorithm provides an effective optimization technique in finding optimal structure and weights for block-based neural networks, but the convergence speed is often too slow as well as it is limited to binary representation of internal weights.

1.2 Contributions

In this dissertation, we work on development of optimization algorithms for BbNN configuration and demonstration of the capabilities of the BbNN approach in various dynamic environments. Contributions in finishing this dissertation are summarized in the following.

Computationally efficient optimization of block-based neural networks. We describe a computationally efficient evolutionary algorithm that simultaneously optimizes structure and weights of BbNNs (Chapter 3, pp. 33-64). Fitness scaling and local search techniques are developed to circumvent the deficiencies of inefficient and slow optimization frequently encountered in previous evolution scheme. Feedforward implementation of BbNNs is considered to facilitate hardware implementation and enables the use of local search (Section 3.1, pp. 33-39). Evolutionary operators are designed to work directly on the phenotype of BbNN individuals that eliminates the encoding/decoding procedure between BbNN phenotype and genotype as in conventional evolutionary algorithms (Section 3.2.3, pp. 43-51). To speed up the optimization, a local search operator based on gradient descent is integrated with the evolutionary algorithm (Section 3.2.3.3, pp. 48-51). A fitness scaling with generalized disruptive pressure that favors individuals at two extreme ends makes an effective approach for searching in mountainous function landscape of BbNNs (Section 3.2.2, pp. 41-43). An adaptation scheme that rewards or penalizes an operator based on its past performance automatically updates the

parameters during evolution without manual adjustment of operator rates (Section 3.2.4, pp. 51-53).

Personalized ECG heartbeat classification. A personalized ECG heartbeat classification scheme is implemented based on the BbNN platform. (Chapter 4, pp. 65-99). The structure and weights of a selected BbNN are evolved for a patient using training data consisting of both common and patient-specific heartbeat patterns. The Hermite transform coefficients and a time interval between the two neighboring R peaks of ECG signal are used as the input to the network. The evolved BbNN provides a personalized monitoring system that classifies each heartbeat into one of five classes recommended by Association for the Advancement of Medical Instrumentation (AAMI). Simulation results using the MIT-BIH Arrhythmia Database demonstrate a high accuracy of 98.1% and 96.6% on average for the detection of ventricular ectopic beats (VEBs) and supraventricular ectopic beats (SVEBs), respectively. These results are significant improvements over previously published results for ECG heartbeat classification. The fault tolerance ability of BbNNs on ECG signal classification is studied under two types of fault modes: Global Gaussian noise and local impulse noise. Experiment results demonstrate the fault tolerance of BbNNs by showing that the level of performance degradation is proportional to the severity of noise.

Accelerated local search using blockwise least squares learning. Observing the slow optimization speed of gradient descent search operator (GDS), I use a blockwise least squares learning method (BLS) as an alternative to the GDS for applications where highly accurate results are desired such as nonlinear function approximation (Chapter 5, pp. 100-118). Two examples are studied including Mackey-Glass time series prediction and a practical heater exchanger nonlinear system identification problem. Computer simulations demonstrate that BLS converges faster with orders of magnitude than the gradient-based search.

1.3 Dissertation Outline

This dissertation is organized as in the following:

Chapter 2 introduces background knowledge for this dissertation. This chapter briefly reviews the development of artificial neural networks and evolvable hardware. The introduction of block-based neural network model focuses on discussions about the structure of BbNN, the previous optimization scheme, issues in hardware implementation using reconfigurable computing platform and comparison with the Cellular Neural Networks (CNN) model.

Chapter 3 describes an evolutionary optimization method for block-based neural networks that simultaneously optimizes the structure and weights of the network. This algorithm uses a generalized fitness scaling that can adjust disruptive pressure depending on applications. The section of evolutionary operators discusses in detail crossover, mutation and the gradient descent search operator. An adaptive rate update scheme proposed to replace manual tuning follows. In the end of this chapter, implementation platform is discussed that is followed by an illustrative example showing the effect of various parameters in the algorithm.

Chapter 4 proposes personalized ECG signal classification based on the BbNN model. An evolvable hardware platform is described. Details on the challenges of ECG signal classification, the experimental ECG data, feature extraction and other issues are discussed. The later part of this chapter compares the performance of the proposed method with other techniques and studies the issue of fault tolerance of BbNNs in ECG signal classification.

Chapter 5 introduces an accelerated local search method that uses the least squares principle. This local search method is compared to gradient descent search in terms of convergence speed. The performance of the EA algorithm with the enhanced local search operator is demonstrated by two dynamic system approximation problems.

Chapter 6 concludes this dissertation with future research directions suggested.

Chapter 2 BACKGROUND

2.1 Introduction

This dissertation finds its foundation in the general theory of artificial neural networks (ANNs). An introduction of ANNs is first presented with focus on multilayer perceptrons, cellular neural networks and the neural network learning methods. Then, brief reviews of evolvable hardware and block-based neural networks are given.

2.2 Artificial Neural Networks

Artificial neural networks were inspired by Man's desire to produce systems that are capable of performing complex tasks excelled by the human brain. The field of artificial neural networks covers a vast number of theories and applications and reflects a number of interdisciplinary research efforts. A detailed review of ANN theory is beyond the scope of this document. This section provides only a brief review of ANN theory that is closely related to the main work of this dissertation.

2.2.1 The Biological Neural Network

It is helpful to gain some knowledge about biological neural network as ANNs draw much of its inspiration from the biological nervous system. Human brains are made up of thousands of thousand of neurons of many varieties connected with each other via a vast number of interconnections. The simplified model of a typical biological neuron is shown in Figure 1, which reveals only important computational features and ignores the details that differentiate neurons of different types. The dendrites are input

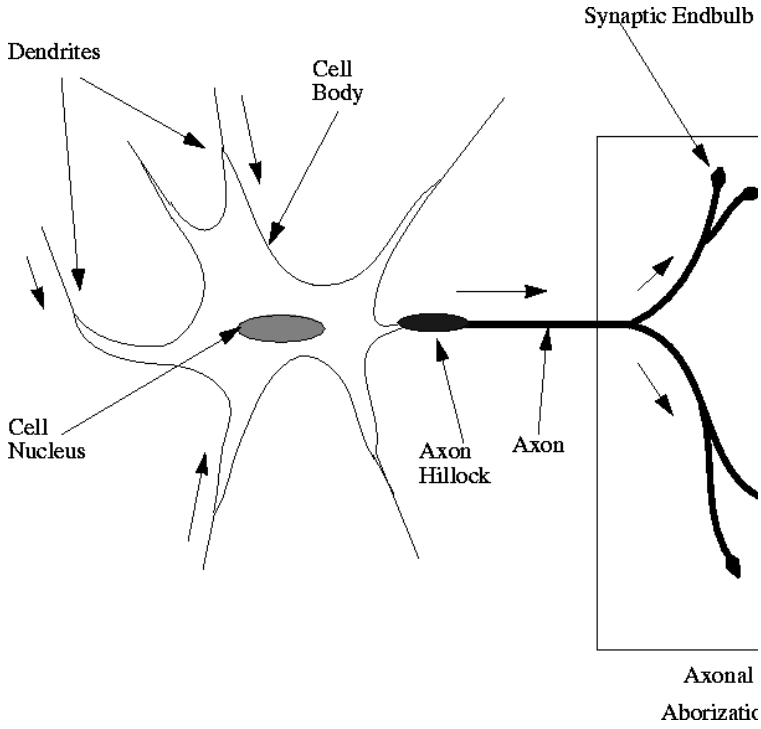


Figure 1: Model of a biological neuron.

Source: http://www.mines.edu/Academic/courses/math_cs/macs570/node11.html.

channels for the neuron. The cell body is the primary processing unit and the axon hillock sums input signals that are transmitted from neighboring neural cells. The axon has many branches connected to other cells forming output channels.

Each neuron receives signals, processes them and sends the outputs to other neurons. Specifically, all the inputs to the cell are summed up. The sum is then processed by a threshold function producing an output signal that propagates down the axon to other connected neurons through branches of the axon. Those branches are connected to the dendrites of other neurons through junctions called synapses. Output signals from one neuron modified at synapses become the input signals to the connected neuron. During this modification process, the activation potential from the pre-synaptic neuron is either lowered or raised, which can be interpreted as a weighting operation of the input signals.

Each neuron can be considered as a basic signal processing element. Billions of neurons are connected to form complex neural networks, each of which can learn to perform a certain task. It is true that the functional capability of a single neuron is limited; however, the neural networks formed with a large number of basic neurons can learn to perform very complex tasks.

2.2.2 ANNs: History and Applications

The desire to emulating the working mechanisms of human brains motivates the development of ANNs. The emulation has been limited to some behavior characteristics of brains mainly due to our limited knowledge about human brain. A lot of joint effort from interdisciplinary researchers has been devoted to ANN research with many exciting work resulted.

The first advance in modern neural network came in 1943 when Warren McCulloch and Walter Pitts wrote a paper [10] on how neurons might work in which they modeled the arithmetic and logical functionality of a simple neural network. In the following, Donald Hebb proposed a learning mechanism in biological neurons [11]. In 1959, Bernard Widrow and Marcian Hoff at Stanford University introduced

the neural network model [12] called ADALINE that is trained using least mean square (LMS) algorithm. ADALINE was the first neural network model applied in real world applications. The development of Perceptron proposed by Rosenblatt in 1958 demonstrates the promise of neural networks in computation [13]. A single-layer perceptron was able to classify continuous valued inputs into one of two classes. Unfortunately, Minsky and Papert later showed in their famous 1969 book [14] that Perceptron is limited only to linearly separable problems. They proved that perceptron neural network cannot solve problems that are not linearly separable. The publication of their book has generated a great impact on neural network research and brought dark to the promises of ANNs. That illusion was not changed until the well-known error back-propagation algorithm was proposed by Rumelhart *et al*. [15][16]. Other major neural network models include self-organizing maps by Kohonen [17] and Hopfield models by Hopfield [18].

Artificial neural networks have drawn a lot of interest from many fields. ANNs have been successfully used in a wide variety of application domains such as system identification, time series prediction, classification, expert systems, etc. Some examples of the applications include speech recognition, face recognition, adaptive signal processing, financial prediction, bioinformatics, control system design, optimal scheduling of task assignments, and electronic circuit layout design.

2.2.3 Multilayer Perceptrons

Let us first look at Rosenblatt's perceptron model shown in Figure 2. Perceptron model consists of a linear combiner and a hard limiter. The linear combiner sums the linear combination of the inputs applied to the synapses of the neuron. A bias is usually applied to the linear combiner too. The sum from the combiner is then subject to a hard limiter to produce an output. The output is +1 or -1 depending on the input to the hard limiter is positive or negative. Mathematically, a neuron computes the output according to the following equation:

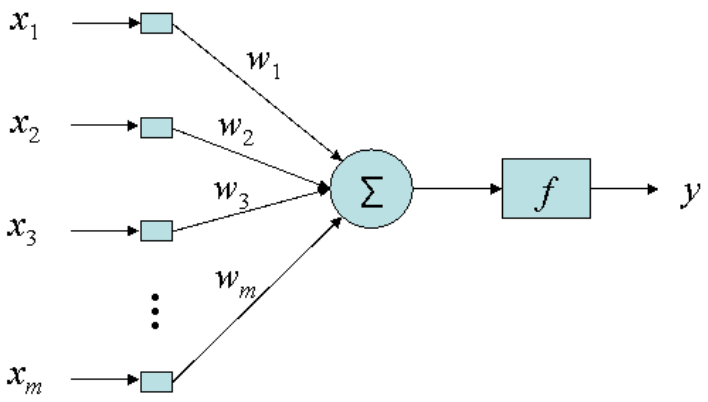


Figure 2: Rosenblatt's Perceptron model.

$$y = f(v) = \begin{cases} +1 & ,if\ v \geq 0 \\ -1 & ,if\ v < 0 \end{cases} \tag{1}$$

where v is the net activation applied to the neuron and is computed according to

$$v = \sum_{i=1}^{m} w_i x_i + b \tag{2}$$

where x_i is an externally applied stimuli and w_i denotes the synaptic weights of the perceptron. The bias to the neuron is denoted by b.

The perceptron described is able to classify the set of inputs x_1, x_2, …, x_m into one of two classes, C_1 and C_2. The decision boundary given by the perceptron is simply a hyperplane defined by:

$$\sum_{i=1}^{m} w_i x_i + b = 0 \tag{3}$$

The decision rule is to assign a point in the m-dimensional input space to C_1 if the

output from the perceptron is positive; otherwise, the point is assigned to C_2.

This two-layered architecture was found to be able to implement simple logic functions. However, due to a lack of usable training algorithm, perceptron model is limited to have only two layers, which severely limits the capability of perceptrons. As pointed out by Minsky and Papert in their book entitled, *Perceptrons: An Introduction to Computational Geometry* [14], the two-layered perceptron cannot solve the problem even as simple as the XOR classification.

The book by Minsky and Papert reveals the severe limit of two-layered perceptron model and suggests adding hidden layers in order to extend the capability of Perceptrons. However, the addition of hidden layers requires new learning algorithm that is capable of training perceptrons with more than two layers. Such an algorithm did not appear until the advent of back-propagation (BP) algorithm proposed by Rumelhart *et al*. in 1986. BP algorithm provides a computationally efficient approach to training multilayer perceptrons. BP algorithm lifts the limitations of two-layered perceptrons and enables the perceptrons to solve complex practical problems. Since its publication, BP algorithm has gained popularity in neural network field due to its simplicity and effectiveness in solving practical problems. In the following, the idea of BP algorithm is briefly visited.

Figure 3 shows the architecture of a multilayer perceptron. The network is composed of an input layer, one or more hidden layers and an output layer. Each neuron in a layer is connected to any neuron in the next layer. Signals progress in the network in a forward direction, from input layer to hidden layer and to output layer.

BP algorithm consists of two passes: forward pass of input signal and backward pass of error signal. In the forward pass, input signals x_i (i = 1, 2, …, m) propagate from the inputs layer through the hidden layer to output layer producing network outputs y_k (k = 1, 2, …, n).

$$\begin{aligned} y_k &= f\left(v_k\right) \\ &= f\left(\sum_{j=1}^{J} w_{jk} z_j + b_k\right) \end{aligned} \tag{4}$$

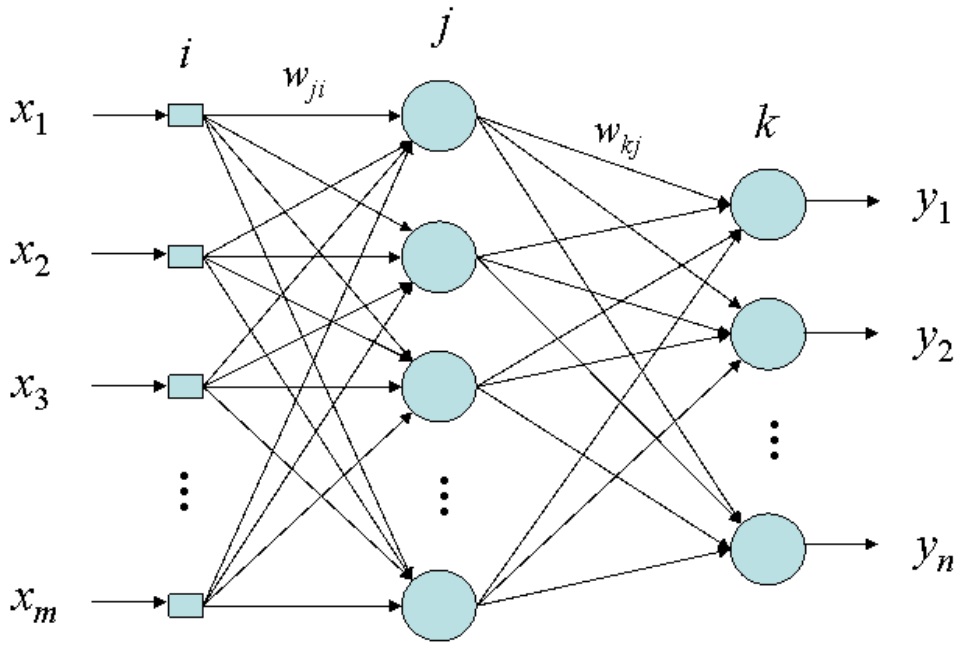


Figure 3: Feedforward multilayer perceptrons.

where $f(\cdot)$ is the activation function and z_j is output from the jth node in the hidden layer:

$$z_j = f\left(\sum_{i=1}^{I} w_{ji} x_i + b_j\right) \tag{5}$$

In a backward pass, error signals generated at output nodes are back propagated from output layer to previous layers. In order to derive the BP learning procedure, let us define in the following equation an error criterion that the algorithm wants to minimize.

$$\varepsilon(w) = \frac{1}{2}\sum_{k=1}^{K}\left(t_k - y_k\right)^2 \tag{6}$$

where t_k and y_k are the target and actual output at the kth node of output layer. Beginning with an initial guess, successive weight vectors are generated such that the error is reduced at each iteration. The simple gradient steepest descent updates the weights according to the following equation [19]:

$$\Delta w = -\eta \frac{\partial \varepsilon}{\partial w} \tag{7}$$

where η is the learning rate.

For the weights w_{kj} that connects hidden node to output node, we can compute the gradient using chain rule:

$$\frac{\partial \varepsilon}{\partial w_{kj}} = \frac{\partial \varepsilon}{\partial v_k}\frac{\partial v_k}{\partial w_{kj}} \tag{8}$$

As we have

$$\frac{\partial \varepsilon}{\partial v_k} = \frac{\partial \varepsilon}{\partial z_k}\frac{\partial z_k}{\partial v_k} = -\left(t_k - z_k\right) f'\left(v_k\right) \tag{9}$$

and

$$\frac{\partial v_k}{\partial w_{kj}} = y_j \tag{10}$$

So by using Eq. (9) and Eq. (10), Eq. (8) can be rewritten as in the following

$$\frac{\partial \varepsilon}{\partial w_{kj}} = -\left(t_k - z_k\right) f'\left(v_k\right) y_j \tag{11}$$

The update equation for w_{kj} is therefore given in the following equation:

$$\Delta w_{kj} = -\eta \frac{\partial \varepsilon}{\partial w_{kj}} = \eta \left(t_k - z_k\right) f'\left(v_k\right) y_j \tag{12}$$

For the weights w_{ji} that connects input node to hidden node, we can again use the chain rule to compute the gradient:

$$\frac{\partial \varepsilon}{\partial w_{ji}} = \frac{\partial \varepsilon}{\partial v_j} \frac{\partial v_j}{\partial w_{ji}} \tag{13}$$

Since the following equation holds

$$\frac{\partial \varepsilon}{\partial v_j} = \left(-\sum_k \left(t_k - y_k\right) f'\left(v_k\right) w_{kj} \right) f'\left(v_j\right) \tag{14}$$

Let us use the following notations

$$\delta_k = \left(t_k - y_k\right) f'\left(v_k\right) \tag{15}$$

and

$$\frac{\partial v_j}{\partial w_{ji}} = x_i \tag{16}$$

Then Eq. (13) can be formulated as in the following equation using Eqs. (14)-(16)

$$\frac{\partial \varepsilon}{\partial w_{ji}} = \left(-\sum_k \delta_k w_{kj} \right) f'\left(v_j\right) x_i \tag{17}$$

The update equation for w_{ji} is therefore given in the following equation:

$$\Delta w_{ji} = -\eta \frac{\partial \varepsilon}{\partial w_{ji}} = \eta \left(\sum_k \delta_k w_{kj} \right) f'\left(v_j\right) x_i \tag{18}$$

From above derivation, it is clear that the activation function needs to be differentiable in order to apply the back-propagation learning algorithm. Examples of popular choices are log sigmoidal and tangent sigmoidal function that are shown in Figure 4.

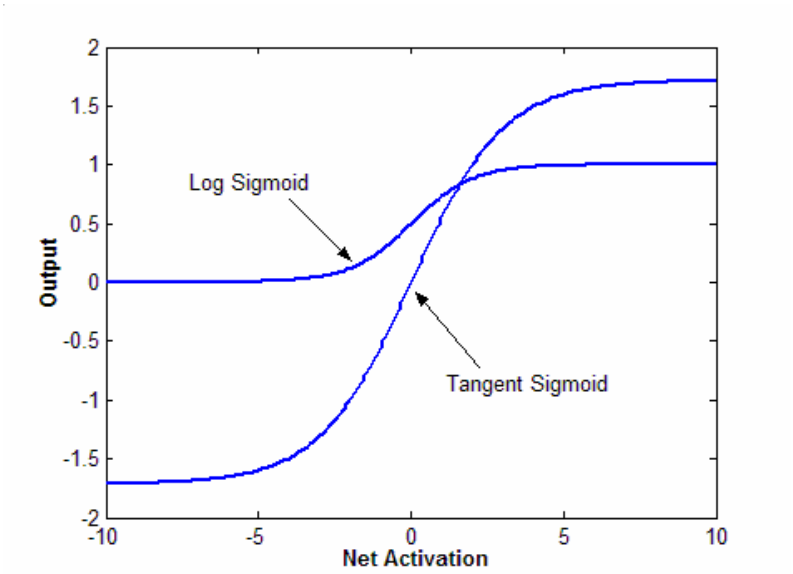


Figure 4: Examples of popularly used activation functions.

2.2.4 Cellular Neural Networks (CNNs)

CNNs are a class of artificial neural networks that feature a regular array of component cells and local interconnections among the cells. The CNN model was invented by Chua and Yang in their seminal papers published in 1988 [20][21]. Unlike digital computers, CNNs process signals in continuous-time space due to the use of analog elements in constructing the cells. Unlike digital computers that execute instructions sequentially, the cells in a CNN process signal in parallel. A CNN is suited for VLSI implementation because of its local interconnections.

A cellular neural network is composed of *N*-dimensional array of basic circuit elements called cells. A cell is connected only to its neighbor cells. Adjacent cells interact with each other directly through connection weights. Cells not directly connected may affect each other through propagation effect. A two-dimensional CNN with a size of 3×5 is shown in Figure 5, in which any cell is connected to its 1-neighborhood cells. The links connecting two cells indicate interactions between them. In general, a cell can be connected to its *r*-neighborhood cells. The *r*-neighborhood of a cell located at the *i*th row and the *j*th column in a CNN, denoted by $C(i,j)$ is defined as:

$$N_r(i,j) = \left\{ C(k,l) \middle| \max\left\{ |k-i|, |l-j| \right\} \le r \right\}, 1 \le k \le M, 1 \le l \le N \tag{19}$$

where *M* and *N* denote the rows and columns of the network, respectively.

The basic elements in a cell include linear capacitors, linear resistors, linear and nonlinear controlled sources and independent sources. The controlled sources can be implemented using operational amplifiers. All cells in a CNN share the same circuit structure and element values. A cell $C(i,j)$ has direct connections to its neighbors through two kinds of weights: the feedback weights arranged in the Feedback Template and the control weights arranged in the Control Template. A block diagram of the cell is shown in Figure 6. The state equation of the cell $C(i,j)$ is given in the following:

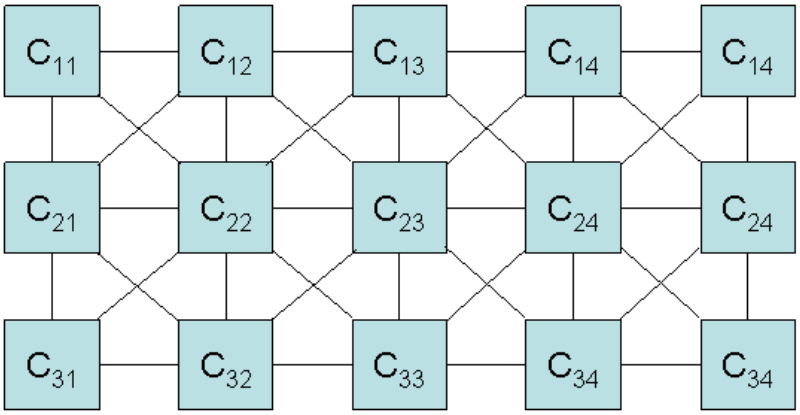


Figure 5: A two-dimensional cellular neural network with a size of 3×5.

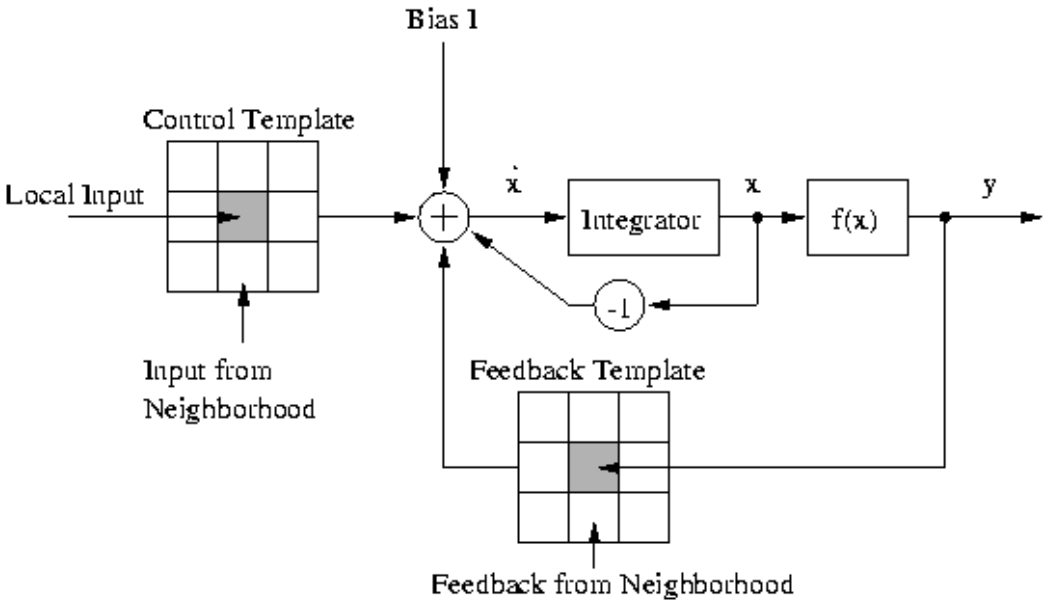


Figure 6: The block diagram of a cell of CNNs. Source: [22].

$$C\frac{dx_{ij}}{dt} = -\frac{1}{R}x_{ij} + \sum_{k}\left(a_k y_k + b_k u_k\right) + I \tag{20}$$

where C and R denote the capacitor and resistor in the circuit, respectively. Coefficients a_k and b_k are the weights in the Feedback and Control Template. Index k denotes a specified neighborhood of the cell $C(i,j)$. Variables x_{ij}, u_k and y_k correspond to the state, input and output of the cell. The output of cell is given as in the following piece-wise linear equation:

$$y_{ij} = \frac{1}{2}\left(\left|x_{ij}+1\right| - \left|x_{ij}-1\right|\right) \tag{21}$$

Cellular neural networks find applications in high speed parallel signal processing such as image processing and pattern recognition [23]. By choosing appropriate coefficients in the Control and Feedback Templates, CNNs are able to perform such image processing tasks as noise removal, edge detection and character recognition. The CNN model is found to be orders-of-magnitude faster than a PC-based solution in a task involving in processing an image of size 128×128 [23].

2.2.5 Neural Network Learning Methods

Artificial neural networks offer a distribution-free approach to universal function approximation and pattern classification. ANNs have become an important and commonly used computation model in a wide range of application areas. The power of neural networks lies in their general applicable capability.

A neural network can be used only after its internal weights are properly trained for the target problem. A training procedure involves in applying a set of training patterns to the network, computing the errors at output node and adjusting the weights to minimize the errors. The back-propagation algorithm described in preceding section has become a commonly used training protocol since it is proposed in 1986 [15][16]. Despite of the successful application of the standard BP algorithm in

solving many problems, it suffers from several drawbacks. The major drawback is the extreme slow rate at which the algorithm converges to a satisfactory solution. The training time also increases greatly as the complexity of the problem goes up.

The effort for improving the standard BP algorithm has never stopped with constant appearance of new training methods. A number of improved algorithms were proposed in literature [24][25][27][28][29][30]. Scalero and Tepedelenlioglu [24] proposed an algorithm that tries to minimize the mean squared error between the actual and desired summation outputs from a neuron. A set of linear equations is constructed for a neuron from which the associated weights are solved using Kalman filter technique. The desired summation output to a node in hidden layers is however estimated in a way similar to the error back-propagation procedure as in the BP algorithm.

König and Bärmann [25] proposed the Least Squares Back Propagation algorithm (LSB) for training feedforward neural networks based on linear least squares and layer-by-layer optimization. For each layer, the weight optimization is formulated as a linear least squares problem that minimizes the mean squared error between the actual and desired linear neuron outputs. Solving the linear least squares problem at output layer (layer L) produces an optimal set of weights for the layer. The desired output vector for the layer preceding the output layer (i.e. layer L-1) is determined by solving another linear least squares problem based on the optimal weights of output layer just obtained. The acquired output vector is then transformed by a matrix to bring its bounds into range of the activation function. Then the optimal weights for layer L-1 is obtained by the solution of linear squares. This procedure is repeated for other layers. Although LSB is superior to the standard BP algorithm in terms of convergence speed, it suffers from the "stalling" problem and instability due to fluctuating initial weight solution of output layer. It also needs to transform the estimated target output into the range of the activation function utilized.

Ergezinger and Thomsen [27] proposed to optimize the FNN layer by layer. The network to be optimized is assumed to have one hidden layer that uses sigmoidal activation function. The output layer uses a linear activation function. The algorithm

first optimizes the weights of the output layer and then the weights of the hidden layer. This procedure is performed iteratively. The weights of output layer are obtained by solving a set of linear equations to minimize the mean squared error between desired and actual outputs. For the weights of hidden layer, the nonlinear part (due to the sigmoidal activation function) is first approximated using first order Taylor series. A cost function is then constructed as a combination of the mean squared error and the quality of the linear approximation. Minimizing this cost function gives the optimal solution of the weights of the hidden layer. Comparing to [25], this method optimizes the weights of the output layer in a similar way, but it is different in optimizing the weights of the hidden layer.

Wang and Chen [28] proposed a layer-by-layer optimization method. First, the weights and net inputs to the output layer are solved using matrix inversion simultaneously. Second, the weights of the hidden layer are optimized with matrix inversion. This procedure is repeated until stop criterion is met. For efficient matrix inversion, recursive least-square parameter estimation and recursive least parameter estimation with dynamic forgetting factor (from a reference) are utilized. (cf. Scalero's paper where Kalman filter is used for recursive least square filtering.) The problem of this approach is that the desired output of the hidden layer obtained via matrix inversion may go beyond its allowed range (defined by the sigmoid activation function).

Yam and Chow [29] combined the linear squares method and BP method. The weights of the output layer are obtained by solving a least squares problem. The weights of hidden layers are updated using BP algorithm with momentum. Both learning rates and momentum constant are adaptively adjusted to improve convergence speed and stability.

Abid *et al*. [30] proposed a new form of error criterion, which is the summation of the standard BP criterion and a weighted error term based on the desired and target linear output of the output layer. The gradient descent rule of BP algorithm is adopted for the new error function. The authors gave a proof that shows the gradient descent

using the new error functions converges faster than that using the standard error function.

In the other hand, the design of suitable ANN architecture has relied heavily on human experts who have sufficient knowledge on the neural network model used and the problem domain. Commonly, a trial-and-error procedure is performed in finding a suitable structure for a particular problem. As the complexity of the problem domain increases, manual design becomes more difficult and unmanageable. Autonomous determination of network structure and connection weights is an important issue in automated design of ANNs.

Evolutionary algorithms (EAs) [31][32], inspired by the mechanism of natural selection and evolution, seeks a global optimum from a vast search space. EAs utilize a selection scheme that implements the survival-of-the-fittest principle, and various evolutionary operators that emulate the process of natural evolution. Evolutionary learning provides an optimal solution for non-convex optimization problems, where popular gradient-based learning algorithms fail [33][34]. Evolutionary search procedures have been successful in solving diverse optimization problems [35], and designing neural networks [36].

Evolvable artificial neural networks [36] use the evolutionary algorithms as an essential form of adaptation or learning to find network architecture and the corresponding parameters for a given problem without human intervention. There have been different approaches to EA-based neural network optimization: structure optimization only [37], weight optimization with fixed network structure [38][39][40], and simultaneous optimization of both structure and weights [41][42][43][44]. Hybrid algorithms have been proposed to combine the global search ability of the EA and fine-tuning of local search methods [36]: EA-based structure optimization with gradient-based weight learning [45], weight optimization for a fixed structure network using both EA and back-propagation [46][47][48].

2.3 Evolvable Hardware

2.3.1 Reconfigurable Computing (RC)

The concept of reconfigurable computing dates back to around 1960 when Gerald Estrin in a paper [53] proposed a hybrid computing system that is composed of a standard microprocessor and reconfigurable hardware resource. While the main microprocessor controls the behavior of the reconfigurable hardware, the latter can reconfigure its internal connections to perform specific tasks at a speed of dedicated hardware. Typical RC platforms are circuit boards that house reconfigurable digital hardware such as FPGAs and other related hardware resources.

FPGAs consist of an array of Configurable Logic Blocks (CLBs) and configurable interconnections between them. The functional CLBs are often implemented as look-up tables (LUTs) and can be configured to implement various Boolean functions. Each LUT can implement a specific Boolean function by loading appropriate bit patterns into it. These CLBs are connected using configurable interconnections. A complex digital logic circuit can be formed by routing the input signals to CLBs of various functions and output signal to output pins.

A number of companies have built a wide variety of RC boards. These boards differ in the number of on-board FPGAs, capacity of the FPGAs, and other on-board hardware resources. An example of the system, Amirix AP100 board from Amirix Systems, is shown in Figure 7. This particular FPGA board features a Xilinx Virtex-II Pro FPGA – XC2VP30, two on-chip PowerPC 405 processors, on-chip block RAMs and multiplier blocks. The RC boards are typically connected to a host microprocessor. A host program is usually utilized to control the reconfiguration and initialization of the FPGA and handle communication between the host processor and the board.

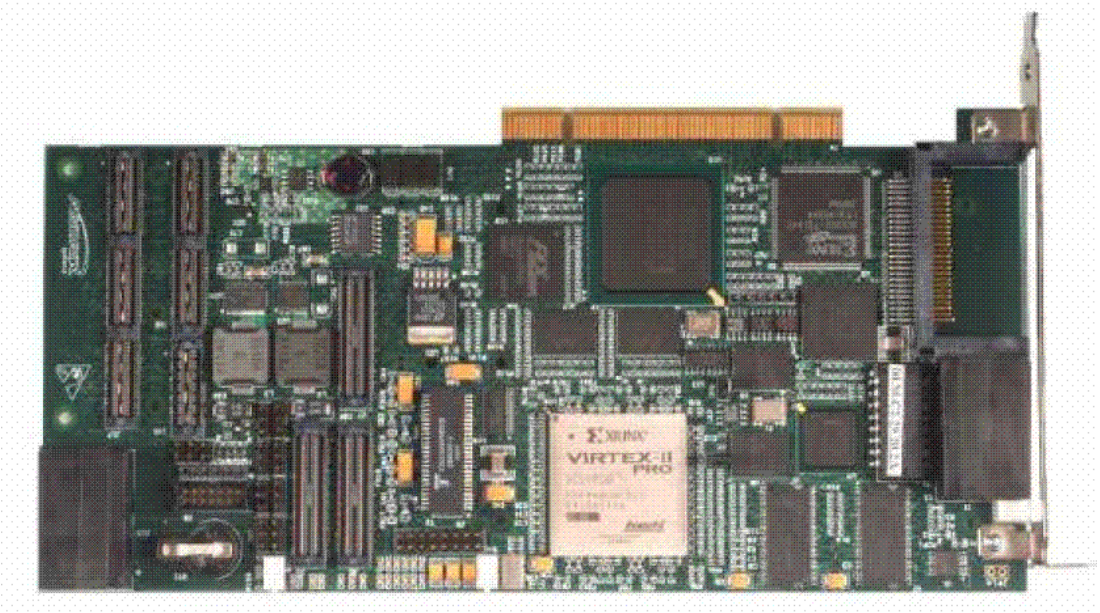


Figure 7: Amirix AP100 Board (Copyright of Amirix Systems).

2.3.2 Evolvable Hardware Using FPGA

Evolvable hardware refers to using evolutionary algorithms to designing electronic circuits automatically. The most widely used hardware is FPGA, although other types of hardware are also used like Field Programmable Analog Arrays. The hardware can be evolved in one of two ways: *extrinsic* or *intrinsic* according to DeGaris [54]. In extrinsic evolution, offline evolution is performed on a software model of the hardware system. Intrinsic evolution of hardware evolves the hardware online in which the hardware is directly changed by the evolutionary algorithm and the I/O measured from the hardware affects the search process of the evolutionary algorithm.

DeGaris divided the evolvable hardware into two categories based on whether the hardware is in the loop of online evolution. In the author's opinion, there exists another distinction among various evolvable hardware approaches based on at what level the hardware is actually evolved. The hardware can be evolved at different levels, for example, at gate level [55][56], at function level [57][58] or at neural network level [59][60]. In gate level evolution, configuration bits of FGPA cell logic function and interconnections are evolved. The evolution is based on primitive gates such as AND and OR. Example of gate level evolution is the groundbreaking work reported in 1996 by Adrian Thompson at the University of Sussex [55]. In his research, Thompson used the evolutionary algorithm to evolve a tone discriminator using an FPGA from Xilinx. The task involves in using 100 FGPA logic cells to evolve a circuit that could discriminate between square waves of 1kHz and 10kHz, without the use of clock signal. This task is not easy due to the lack of clock signal and the fact that the input periods are much longer than the propagation delay of the logic cell. However, the evolutionary algorithm was able to find a solution that discriminates the two tones successfully after 3,500 generations of evolution. The importance of Thompson's work lies in that it is the first demonstration of successful online hardware evolution and it opens the door for future research in this exciting area.

Despite the success, there are several limitations coming with gate level evolution. In Thompson's work, it is found that the evolved circuit is sensitive to physical location of the circuit in the device and the temperature. It also lacks the flexibility of porting to other FOGA devices other than the one used during evolution. Another major drawback of gate level evolution is the scalability problem that refers to the greatly increased difficulty in evolving an FPGA with larger gates (that result in genotypes with larger sizes).

Later on, function-level evolution was proposed [57][58] to tackle scalability. In function-level evolution, the evolution is based on higher-level functions such as adder, sin, multipliers instead of the primitive gates. Although the function-level approach was able to evolve circuits for relatively complex task, it requires human selection of functions for specific applications. A recent effort tackling scalability utilizes a decomposition strategy in the evolution of large combinational circuits [56].

A latest trend in evolvable hardware aims at evolving circuits at neural network level, or designing ANNs using FGPA [59][60]. In 2003, a research group at Ecole Polytechnique Federale de Lausanne (EPFL) implemented an evolvable hardware system using spiking neural networks in an effort to build an intelligent processor for robot navigation [59]. In their design, an Altera FPGA with 200,000 gates was used to build a spiking neural network with run-time reconfigurability of the network connectivity. Evolving network connectivity has been done via software simulation and the suitable chromosome is then downloaded to the FPGA. This approach is extrinsic in nature because the hardware is not in the loop of the evolutionary procedure.

In 2004, Dennis Earl at the University of Tennessee attacked the neural network level evolution using unconstrained artificial neural network [60]. In this design, the connectivity among neurons is not constrained that is compared to the work in [59] where only a subset of all the possible connections is allowed. Two strategies are implemented. In the first strategy, the evolutionary algorithm runs in software, but every candidate network is realized in a hardware description language, compiled, synthesized, and downloaded into the FPGA. The performance of this network in the

FPGA is measured and fed back to the evolutionary algorithm. The target problem is frequency recognition that requires the circuit responding linearly to square waves of increased frequency from 10Hz to 70 Hz. After 300 generations of evolution, an ANN network is identified that could approximate the desired response closely. Because every network has to go through a procedure of compilation, synthesize and downloading before it can be realized in FPGA, the evolutionary process is very slow. In fact, a single generation took 4.8 hours and 300 generations would take 2 months. This design demonstrates an intrinsic evolvable hardware system; however, the extreme slow speed makes it impractical in real-world applications. The other strategy implements a flexible structure for ANNs in FPGA that avoid the need of hardware reconfiguration for each network structure. However, this strategy puts a limitation on the maximum size of the network that can be implemented in FPGA due to the extra resources used in implementing the flexible structure. The reconfiguration speed is also affected by the data transfer rate between FPGA and the host. Overall, the latter strategy reduces the time per generation to 1.2 hours compared to 4.8 hours of the first strategy.

2.4 Block-based Neural Networks

Block-based neural networks (BbNNs) model [4] provides a unified approach to the two fundamental problems of artificial neural network design: simultaneous optimization of structures and weights and implementation using reconfigurable digital hardware. An integrated representation of network structure and connection weights of BbNNs offers simultaneous optimization by use of the evolutionary algorithm. Block-based neural networks have a suitable structure for implementation using re-configurable hardware. The network can be easily expanded in size by adding more blocks. BbNNs can be implemented by use of reconfigurable digital hardware such as FPGAs that can modify (reconfigure) its internal structure dynamically in response to the operating environments [49]. Such characteristics enable BbNNs to fine-tune the structure and weights “on the fly” to cope with

changing environments. BbNNs have been applied to various practical problems such as mobile robot navigation [4], pattern recognition [50], time series prediction [51][52] and ECG signal classification [5][6][7][8].

2.4.1 Network Structure

A BbNN can be represented by a 2-D array of blocks. Each block is a basic processing element that corresponds to a feedforward neural network with four variable input/output nodes. A block is connected to its four neighboring blocks with signal flow represented by an arrow between the blocks. Leftmost and rightmost blocks are laterally interconnected. Signal flow uniquely specifies the internal configurations of a block as well as the overall network structure. Figure 8 illustrates the network structure of an $m \times n$ BbNN with m rows and n columns of blocks labeled as B_{ij}. The first row of blocks B_{11}, B_{12}, ..., B_{1n} is an input layer and the blocks B_{m1}, B_{m2}, ..., B_{mn} form an output layer. BbNNs with n columns can have up to n inputs and n outputs. Redundant input nodes take a constant input value and the output nodes not used are ignored. Due to its modular characteristics, a BbNN can be easily expanded to build a larger-scale network. The BbNN can have a multiple number of middle layers ($m \geq 1$). The size of a BbNN is only limited by the capacity of a reconfigurable hardware.

2.4.2 Optimization of BbNNs

Optimization of BbNN includes both structure and weight learning. Structure learning refers to determination of internal configuration of blocks and weight optimization determines weights of internal configurations for given training data. The structure and weights of block-based neural networks are optimized using a genetic algorithm (GA).

Network structure and connection weights of an individual BbNN are encoded

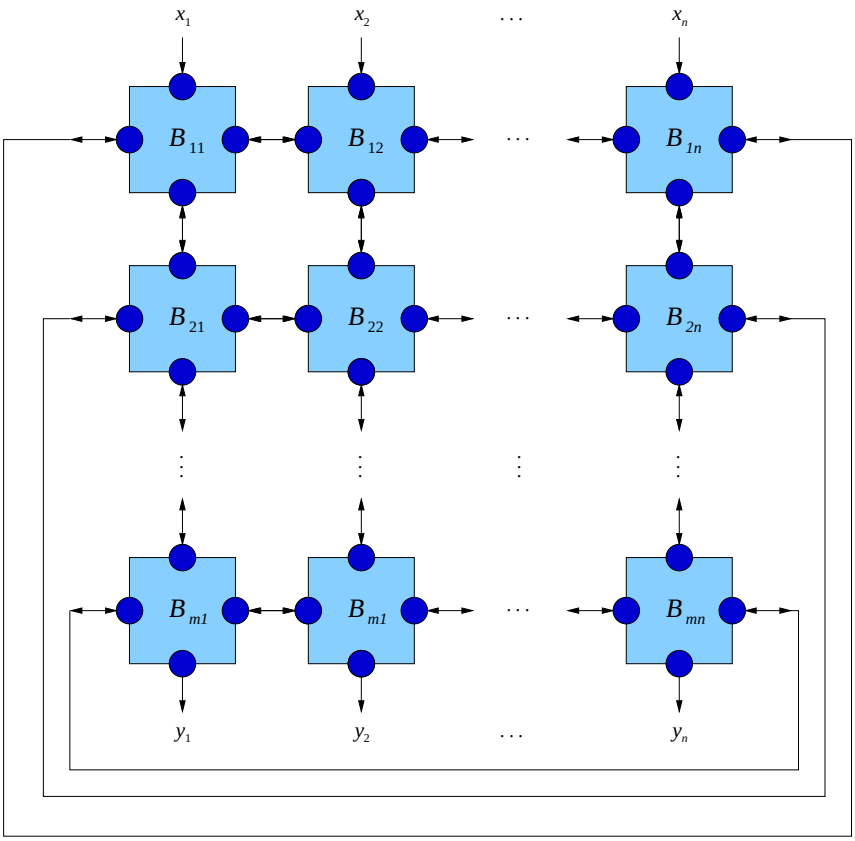
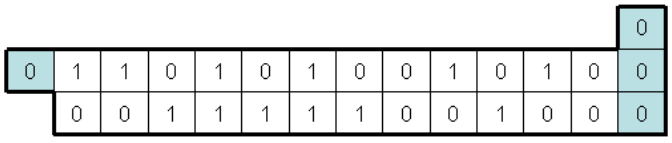
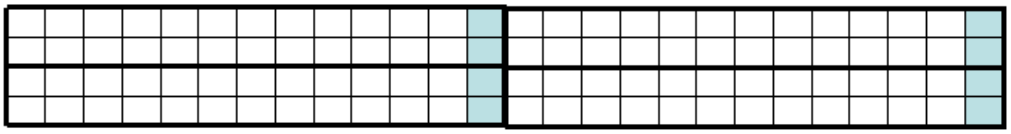


Figure 8: Structure of block-based neural networks.

to form a chromosome for optimization using the GA. The overall structure of a BbNN can be effectively encoded with binary directions of signal flow. Signal flow provides an integrated representation of BbNN structure and internal configurations. The signal flow determines the structure and the internal configuration of a BbNN using a sequence of binary numbers. Any connection between the blocks is represented with either 0 or 1. Bit 0 denotes down (↓) and left (←), and bit 1 indicates up (↑) and right (→) signal flows. Figure 9(a) shows the encoding scheme for a basic block of BbNN in which white boxes denote connection weights and colored boxes are structure bits. The weights are represented with 4-bit binary numbers. The signal flow bits associated with the blocks in the input and output stages are all zeros and therefore are not included in structure encoding. Figure 9(b) shows the chromosome representation of a 2×2 BbNN with each of its four blocks encoded with the scheme illustrated in Figure 9(a). Neighboring blocks share signal flows and the common structure bits are therefore the same.



(a)



(b)

Figure 9: Chromosome representation of BbNN, (a) block encoding, (b) network encoding.

An initial population of BbNN chromosomes is generated that represents a set of individual BbNNs as candidate solutions for the given problem. Each generation of the genetic algorithm involves three main components: fitness evaluation, selection, and genetic operation. The current population of BbNNs are evaluated and ranked in terms of fitness value. The population goes through the selection and the genetic operation until the maximum fitness reaches the desired value. The genetic algorithm for BbNN optimization proceeds as in the following:

1) An initial population of BbNN chromosomes is randomly generated.

2) Each chromosome in the population is mapped into the corresponding individual BbNN network. The quality of the BbNN networks in the current population is measured in terms of a pre-defined fitness function. The fitness function is defined such that a BbNN network with higher fitness value corresponds to a better solution to the target problem.

3) A new population of chromosomes is generated based on current population. Fitter individuals with higher fitness values are selected using a selection method and their corresponding chromosomes undergo a set of genetic operations to producing offspring. The genetic operators used include crossover, mutation, copy and inversion. Each operator is applied with a pre-selected probability. The new population consisting of newly produced offspring replaces the current population. The best individual in current population is included in the new population to implement the elitism strategy.

4) Steps 2) and 3) are repeated until a satisfactory solution is found.

2.4.3 BbNNs on FPGA

The flexible and modular architecture of BbNNs facilitates the implementation of BbNNs on RC platforms. Block-based neural networks can be implemented using reconfigurable digital hardware such as FPGAs that can modify and fine-tune its internal structure "on the fly" during the evolutionary optimization procedure. A library-based approach [49] creates a library of VHDL modules of the BbNN basic blocks and pieces together these basic modules to form a custom BbNN network, which is then synthesized, placed and routed, and downloaded to an FPGA. This initial approach gains some flexibility in that some parameters like block internal weights are software configurable, but it suffers from the major problem that any change in the network structure would require a new hardware design and FPGA reconfiguration.

A recent effort [56] implements a "smart block" that can be software reconfigured to work as any one of the basic blocks. Therefore, the structure of the network can be reconfigured via software removing the need of hardware redesign and FPGA reconfiguration. The design was implemented on Amirix AP130 board shown in Figure 7. This approach implements a complete System-on-Chip (SoC) design with the evolutionary algorithm running on PowerPC and the reconfigurable BbNN network implemented in FPGA. Research work has been carried out to implement a complete evolutionary algorithm on FPGAs that results in performance improvement over software implementation [62][63].

2.4.4 A Comparison between BbNN and CNN

There are similarities shared by the CNN and BbNN model. To some extent, the two models resemble each other by adopting a regular array of basic units and local interconnections among those units.

However, there are clear distinctions between the two models. In a BbNN, blocks are arranged in layers with the first and last row being the input and output layer.

Information flows from the input layer, through middle layer(s) and at last reach the output layer. Each block in a BbNN is only connected to its four immediate neighboring blocks. On the contrary, in a CNN, cells are locally interconnected with each cell being a separate dynamic system. Each cell in a CNN is connected to the cells within its *r*-neighborhood. In the extreme case, each is connected to all the other cells in the network. Also, the basic unit in the two models functions differently. The function of a block in a BbNN can be described using a set of linear summations and nonlinear activation functions, while the dynamics of a cell is governed by a set of partial differential equations. The output from the output neuron in a BbNN has a range determined by the activation function. In the CNN case, the output from a cell was proven to converge to a value of either +1 or -1. Moreover, while both the structure and internal weights in a BbNN are optimized using an evolutionary algorithm, the CNN template coefficients are selected initially using cut-and-try techniques and later with a variety of methods including Genetic Algorithm, fuzzy design technique, and even neural network techniques. Last, while the BbNN model targets at applications where the dynamic nature of the problem needs an evolvable solution, CNNs are found to be advantageous in applications as high speed visual computing.

Chapter 3 EVOLUTIONARY OPTIMIZATION OF BBNN

Block-based neural network model has modular structures of two-dimensional basic blocks suited for implementation using reconfigurable digital electronic hardware such as FPGAs that allow on-line partial reorganization of internal structures. The structure and internal weights of BbNNs are simultaneously optimized with an evolutionary algorithm. The evolutionary algorithm provides an effective optimization technique in finding optimal structure and weights for block-based neural networks, but the convergence speed of evolutionary algorithm-based learning is often too slow. This chapter introduces an evolutionary algorithm that utilizes local search operator to increase the convergence speed of optimization of BbNNs.

3.1 Block-based Neural Network Model

Figure 10 shows the structure of feedforward implementation of an $m \times n$ BbNN with m rows and n columns of blocks labeled as B_{ij}. A block is connected to its four neighboring blocks with signal flow represented by an incoming or outgoing arrow between the blocks. The vertical signal flows are all considered downward. A feedforward implementation facilitates hardware implementation of block-based neural networks and enables the use of gradient-based local search. Artificial neural networks implemented using digital hardware such as FPGAs have been confined to feedforward architectures [64][65]. Implementation of feedback BbNN architecture in digital hardware can cause unstable network output. Moreover, a long propagation delay and the use of extra hardware resources to store the network states are unavoidable in feedback implementation. The feedforward implementation also enables the usage of gradient-based local search that combined with global search can

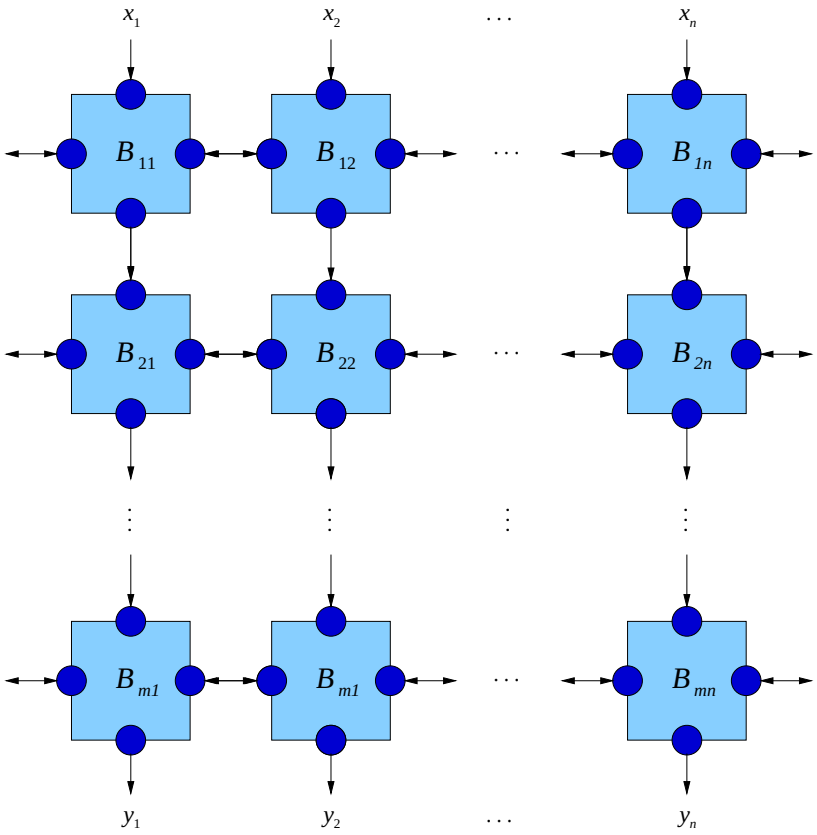


Figure 10: Feedforward implementation of block-based neural networks.

potentially increase the optimization speed significantly [36]. In the following, a theorem regarding the number of possible structure combinations in a BbNN is presented, following by a corollary for the case of feedforward implementation of BbNNs.

THEOREM 1: The number of all the possible structures of the block-based neural network of the size $m \times n$ is $2^{(2m-1)n}$.

Proof: A BbNN of the size $m \times n$ has m rows and n columns of basic blocks. The number of horizontal connections (signal flows) between the blocks in a stage equals the number of columns (n). Since there are m rows of blocks, the total number of horizontal signal flows is

$$N_h = mn \tag{22}$$

The number of vertical connections of a column is (m+1). However, all the blocks in the input and output stages have fixed signal flows (0), which are not responsible for a difference combination of the BbNN structure. Therefore, the total number of vertical connections that affects the structure becomes

$$N_v = (m-1)n \tag{23}$$

Then the total number of all the signal flow bit settings of the $m \times n$ BbNN is

$$N = N_h + N_v = (2m-1)n \tag{24}$$

The number of all the possible BbNN structures equals the number of all the combinations of signal flows. So the number of all the possible structures equals $2^N = 2^{(2m-1)n}$. □

Corollary: The number of all the possible structures of the feedforward implementation of the block-based neural network of the size $m \times n$ is 2^{mn}.

Proof: From Theorem 1, the total number of horizontal signal flows is

$$N_h = mn \tag{25}$$

In the feedforward implementation, the vertical signal flows of all the blocks are fixed downward. Therefore, the total number of all the signal flow bit settings of the $m \times n$ BbNN is equal to the total number of horizontal signal flows. The number of all the possible BbNN structures equals the number of all the combinations of signal flows. So the number of all the possible structures is equal to $2^{N_h} = 2^{mn}$. □

Thus, the number of possible structure combinations for a given BbNN is determined by the number of rows and columns. For a BbNN with a size of 2×7, the number of all the possible structures will be 2,097,152. The feedforward implementation of the same size network will have 16,384 possible structures.

Internal configuration of a BbNN is characterized by the input-output connections of the nodes. A node can be an input or an output according to the internal configuration determined by the signal flow. An incoming arrow to a block specifies the node as an input, and output nodes are associated with outgoing arrows. Generalization capability emerges through various internal configurations of a block. A block can be represented by one of the three different types of internal configurations. Figure 11(a) shows a block with one input and three outputs (1/3). A block in Figure 11(b) has three inputs and an output (3/1). Figure 11(c) corresponds to the type of two inputs and two outputs (2/2).

The four nodes inside a block are connected with each other with the weights. The signal u_i denotes the input and v_j indicates the output of the block, in which the subscripts indicate the node positions. The top, bottom, left and right node has indices

1, 2, 3, and 4, respectively. A weight w_{ij} therefore denotes a connection from node i to node j. A block can have up to six connection weights including the bias. For the case of two inputs and two outputs (2/2), there are four weights and two biases. The 1/3 case has three weights and three biases, and the 3/1 three weights and one bias.

The overall signal flows determine the input-output computation path, along which an input signal $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ propagates through the blocks from top to bottom and generates a network output $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$. A block B_{ij} has four horizontal and vertical neighbors. Let us denote this set of neighbors of B_{ij} by $N(B_{ij})$ given by:

$$N\left(B_{ij}\right) = \left\{B_{i+1,j}, B_{i-1,j}, B_{i,j+1}, B_{i,j-1}\right\} \tag{26}$$

The block B_{ij} is connected with its four neighbors by either incoming or outgoing arrow depending on the signal flow. We further use $I(B_{ij})$ to denote the subset of $N(B_{ij})$ that are connected to block B_{ij} with outgoing arrows. The computation stage of block B_{ij} is computed according to the following equation:

$$s = \max_k \left(s^k\right) + 1, \quad k \in I\left(B_{ij}\right) \tag{27}$$

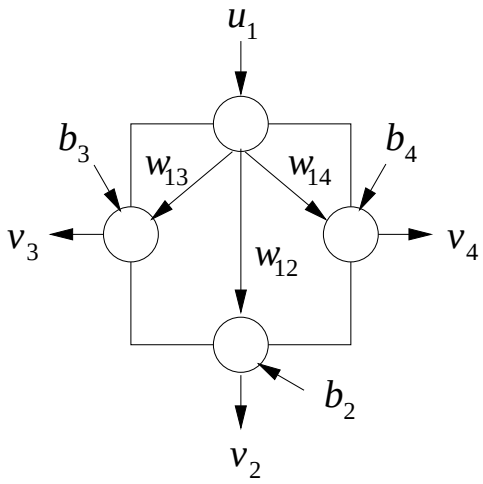
$I(B_{ij})$ may include 0, 1, 2, or 3 neighbors of block B_{ij} depending on its block configuration. When a block B_{ij} is in the input layer and $I(B_{ij})$ is a null set, its computation stage equals one.

For a block B_{ij} in the network, its output node produces an output v_q for the activation with an activation function $h(\cdot)$:

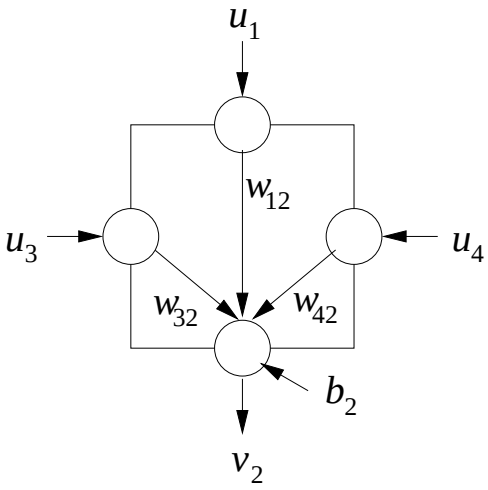
$$v_q = h\left(g_q\right), q \in D \tag{28}$$

The net activation to the node is computed according to the following equation:

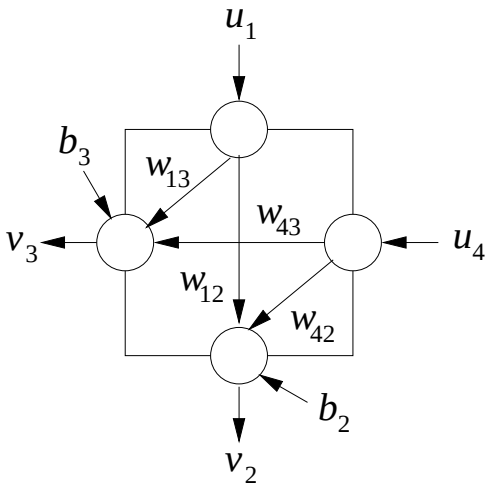
$$g_q = \sum_{p \in C} w_{pq} u_p + b_q, q \in D \tag{29}$$



(a)



(b)



(c)

Figure 11: Three possible internal configuration types of a block. (a) One input and three outputs (1/3), (b) Three inputs and one output (3/1), (c) Two inputs and two outputs (2/2).

where u is the input to the block. C and D are the respective index sets of input nodes and output nodes in the block. For the type 1/3 basic block shown in Figure 11(a), $C = \{1\}$ and $D = \{2, 3, 4\}$. For blocks of the type 3/1, $C = \{1, 3, 4\}$ and $D = \{2\}$. The type 2/2 blocks have $C = \{1, 4\}$ and $D = \{2, 3\}$. The term b_q is the bias term to the qth node.

The computation stage associated with a block represents its priority, according to which the outputs of the block are computed. The blocks in lower stages are calculated earlier than those in higher stages. The blocks in the first calculation stage have the highest priority for output calculation. The input signal $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ is passed through the network from the blocks in lower stages to those in higher stages generating the output $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$.

3.2 Evolutionary Optimization of BbNN

3.2.1 Overview

Evolutionary optimization of BbNN involves three main procedures: selection, variation operation and reinsertion. A parent individual (or a pair of parent individuals for crossover) is selected using tournament selection, varied with a selected operator, and reinserted into the population replacing a chosen inferior individual. Before parent selection, the fitness is rescaled with a generalized disruptive pressure that favors both good and bad individuals. An operator rate update scheme adaptively adjusts rate parameters considering an operator's effectiveness in improving fitness and the current fitness trend.

A pseudo-code description of the evolutionary algorithm is shown in Figure 12. After the random generation of initial population, the algorithm enters into the evolution loop. The current population is first evaluated to update individual fitness values. The fitness rescaling by disruptive pressure ensures the selection of some individuals with low fitness. The algorithm then starts the variation operation stage.

```
Generate randomly an initial population of BbNNs;
k = 0;
do{
   Evaluate fitness values;
   if( desired maximum fitness is achieved )
      break;
   else{
      k = k + 1;
      Fitness rescaling by the disruptive pressure;
      Parent selection;
      Variation operation;
      Reinsertion;
   }
   if ((k % T) equals zero )
      Update operator rates;
 }while( maximum number of iteration not reached )
Save the best individual produced;
```

Figure 12: The evolutionary algorithm for BbNN optimization.

An operator is selected based on a uniform probability distribution. Applying this operator to the parent(s) chosen with tournament selection produces a new offspring. This offspring is reinserted into the population replacing an individual chosen with a tournament selection. This evolution process runs for a fixed number of T generations. After each T generations of evolution, the operators' rates are updated based on their past performance and current fitness trend. The evolution algorithm is terminated either by finding a satisfactory solution or after a certain number of generations.

It is clear that two neighboring populations differ by a single individual in this incremental evolutionary algorithm [66][67]. In the generational EA model, a new population is produced and it replaces the old population. An incremental EA is preferred over the generational model in order to reduce the computational and memory requirements at each generation.

3.2.2 Fitness Scaling and Selection

The search space in many problems can be rather multi-peaked or mountainous. The search space of block-based neural networks resembles a mountainous characteristic due to two reasons. Firstly, each of the possible structures will lead to a local optimum if the weights are properly optimized. Secondly, for a given structure, the weight space can also contain many peaks. In a "Needle-in-a-Haystack" problem, the global optimum is surrounded by poor solutions and isolated from other good regions. The proportional selection that favors good individuals was criticized for its inefficiency in finding the global optimum in such problems [68][69]. A selection scheme with disruptive pressure devotes more trials to both superior and inferior individuals and helps improve search performance as one of the solutions for such problems [68]. A popular disruptive pressure method modifies the fitness by taking an absolute difference of the fitness with the average fitness [69]:

$$f_d = \left| f - f_{avg} \right| \tag{30}$$

where f denotes the actual fitness, f_{avg} is the average fitness, and f_d is the fitness function rescaled with disruptive pressure.

In this study, a modified fitness scaling function with generalized disruptive pressure is used [8].

$$f_d = \left| \; f - f_{\min} - \omega\left(f_{avg} - f_{\min}\right) \; \right| \tag{31}$$

where f_{min} denotes the minimum fitness value. The scaling function adjusts the degree of being selected of an individual whose fitness is near the minimum by controlling the parameter ω that adjusts the degree of disruptive pressure in the range of $0 \le \omega \le 1$. For ω = 0, the scaling function becomes a linear function with no disruptive pressure. When ω = 1, the fitness function becomes the usual disruptive pressure that centers at f_{avg} as in Eq. (30). In this paper, $\omega = 0.6$ was used. Figure 13 shows the relationship between the fitness f and the new fitness f_d rescaled with the generalized disruptive pressure. f and f_d have a linear relationship with a discontinuity at $f_{min} + \omega$ (f_{avg} - f_{min}). The fitness scaling function scales the fitness f_d to have the range between 0 and f_{max} - f_{min} + ω (f_{avg} - f_{min}). As evolution procedure goes on, the average fitness tends to near at the maximum fitness f_{max}. The fitness scaling method with the modified disruptive pressure assures that the bending point locates between the two fitness values f_{min} and f_{avg}.

Two selection processes are present in the evolutionary algorithm: parent selection for variation operation and survivor selection for reinsertion [67]. Parent selection picks one or a pair of individuals from old population for variation operation. The roulette-wheel selection finds individuals in proportional to the fitness value. Despite its popularity, roulette-wheel selection may have problems as premature convergence in early phase of evolution or genetic drift in later phase of evolution. Tournament selection picks out the best one from c randomly chosen individuals [70]. Tournament selection has the same effects as both fitness proportional sampling and selection probability adjustment [71][72]. It has the useful property of not requiring global knowledge of the population. Tournament size, c closely adjusts the selection pressure. A larger tournament size imposes a higher

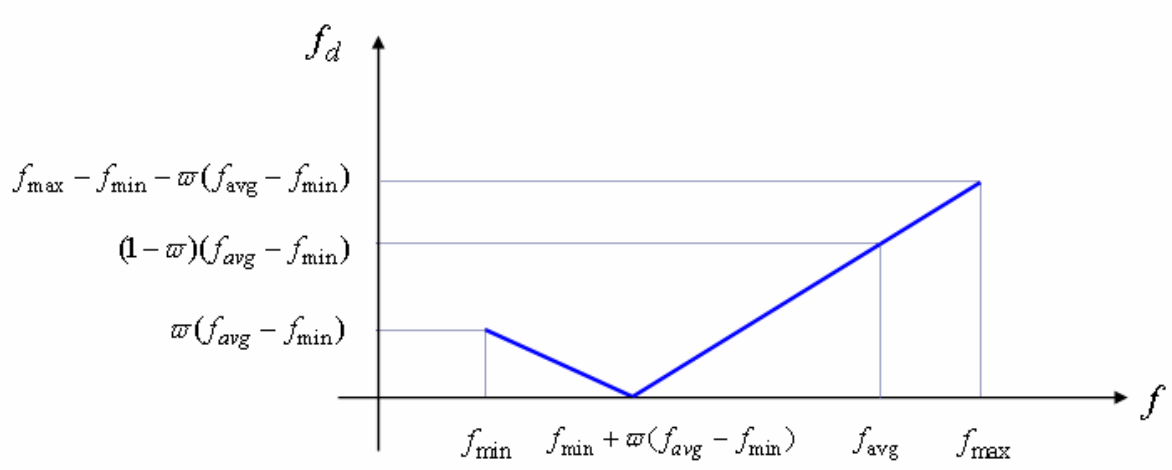


Figure 13: Fitness scaling with generalized disruptive pressure.

selection pressure. Binary tournament used in this paper for parent selection finds the better one of the two individuals selected randomly, i.e., $c = 2$.

Survivor selection determines which member of the current population to be replaced with the newly produced offspring. The commonly used scheme of replacing the worst implements an elitism strategy that keeps the best trait found so far, however it is likely to cause premature convergence because an outstanding individual can quickly take over the entire population under such a scheme [67]. In this paper, a tournament selection that picks the worst individual among c (= 5 in this paper) randomly selected individuals is used. The new offspring is reinserted into the population and it replaces the chosen individual.

3.2.3 Evolutionary Operators

The proposed optimization scheme of BbNN includes two types of genetic operators (crossover and mutation), and a local search operator called gradient descent search

(GDS). Crossover exchanges substructures between two individuals and mutation randomly changes a unit in an individual. The GDS operator searches for better solution in the direction of gradient descent for an individual. All operators directly work on the phenotype of selected BbNN individuals that eliminates the encoding/decoding procedure between the genotype and phenotype of BbNN individuals. The operator rate determines the intensity an operator is applied. The proposed update scheme adaptively adjusts an operator's rate based on both its effectiveness in improving the fitness and current fitness trend.

3.2.3.1 Crossover

Crossover and mutation serve as basic genetic operators used to evolve the structure and weights of the BbNN. For a crossover operation of a pair of BbNNs, a group of signal flows is randomly selected. The selected signal flows are exchanged according to the crossover probability. After the exchange, the internal structure of a block is reconfigured according to the new signal flows. As a result, some weights in a BbNN will have corresponding weights in the other BbNN and some will be alone. Corresponding weights will be updated by a weighted combination of the two weights w_{c1} and w_{c2}.

Crossover operation can be done in two steps: signal flow and connection weights. Figure 14 shows an example of crossover operation. For two individual BbNNs, signal flow bits of the same size and same location are exchanged. Figure 14 (a)(b) demonstrates two individual BbNNs before the crossover operation. Three basic blocks B_{22}, B_{23}, and B_{24} are randomly selected for crossover. Two individual BbNNs in Figure 14 (c)(d) are after crossover operation. Internal configurations of the block after crossover are rearranged.

Crossover operation based on the signal flow takes the following manipulations.

i. Select signal flow bits to be crossed over.
ii. Identify the blocks and the connection weights of the blocks that are connected to the signal flow. Crossover operation can be done for the two

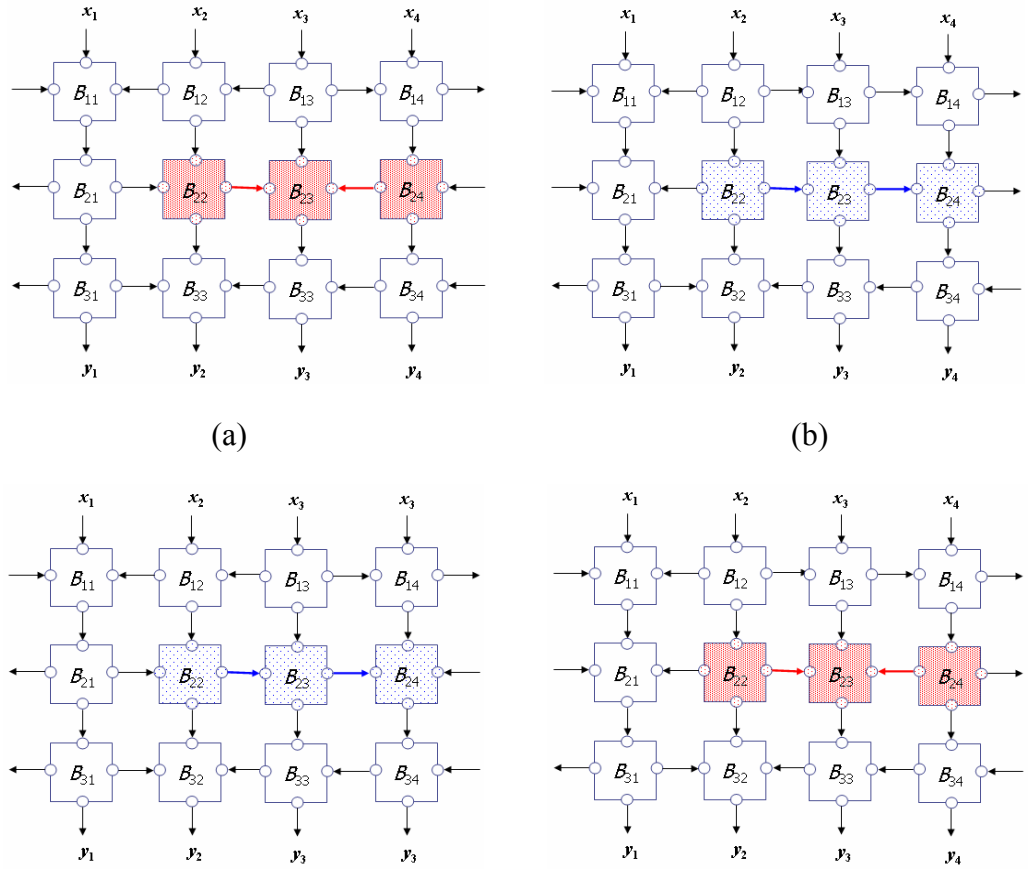


Figure 14: Crossover operation example of two individual 3×4 BbNNs. (a)(b) Two individual BbNNs before crossover, (c)(d) Two individual BbNNs after crossover.

cases:

a. Signal flow remains unchanged after crossover

Take two corresponding weights w_{c1} and w_{c2} from the two blocks. The crossover operator for real-valued weights is defined as:

$$w_{c1}^{*} = \lambda w_{c1} + (1-\lambda) w_{c2}$$
$$w_{c2}^{*} = (1-\lambda) w_{c1} + \lambda w_{c2} \tag{32}$$

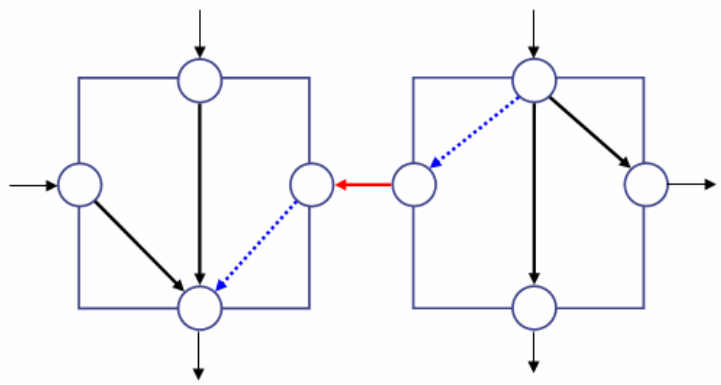
where λ denotes a uniform random number in [0, 1].

b. Signal flow changes after crossover

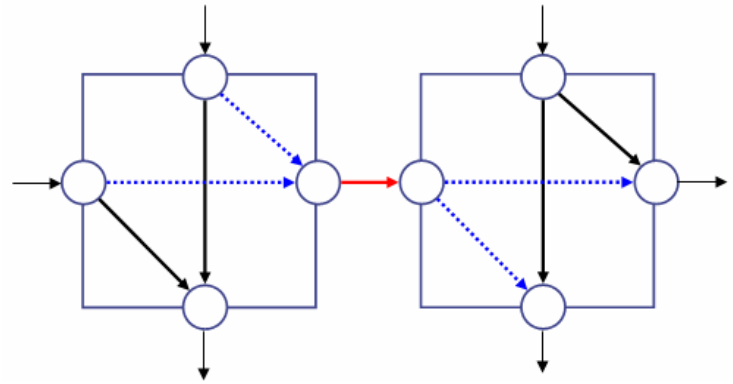
Changed signal flow modifies internal configuration of the block. New connection weights generated accordingly are initialized with Gaussian random numbers having zero mean and unit variance, while not connected weights are removed.

Figure 15 shows an example of crossover for the case ii-b in the above. In Figure 15(a), the signal flow bit between the blocks indicates leftward connection. The two connection weights that are connected to this signal flow are represented as dotted arrows. Assume that the signal flow bit is flipped after the crossover of the signal flow bits. Then internal configurations of the two blocks will be changed as in Figure 15(b), where the weights connected to the changed signal flow are represented in dotted arrows. Changing the signal flow will affect several connection weights of the two neighboring blocks. Newly generated weights are randomly initialized, while pre-existing weights remain the same. Inactive weights are not used in the crossover operation.

The proposed crossover operator has advantage that we can optimize network structure and connection weights at the same time. In early stage of learning, BbNN individuals have a variety of network structures. When the evolution process goes on sufficiently, only relatively small number of possible structures survives. As a result, crossover for signal flow will not change internal configurations as well as structure. So as evolution goes on, the optimization task will be mostly weight optimization.



(a)



(b)

Figure 15: Internal configuration due to changing signal flow.
(a) Before crossover, (b) After crossover.

3.2.3.2 Mutation

Mutation operator randomly adds a perturbation in an individual according to the mutation rate. Block-based neural network has different mutation rates for the structure bit string and the weights. Structure mutation means an operation flipped a signal flow bit according to the structure mutation rate. When signal flow is reversed after mutation, all the irrelevant weights are removed and created with a random value on a proper direction. A weight selected for mutation will be updated with:

$$w_{mt}^{*} = w_{mt} + r \tag{33}$$

where r denotes a zero mean, unit variance Gaussian noise.

3.2.3.3 Gradient Descent Search (GDS)

The gradient descent search operator updates the weights in a BbNN. A GDS operator searches for optimal weights based on gradient descent methods. There are two steps within a GDS epoch: forward pass of the inputs to compute network outputs and backward pass of error signals to update internal weights. After completing the calculation for the blocks in stage s, inputs of the blocks in stage s+1 are updated with the outputs from the blocks in stage s, followed by calculating the outputs of the blocks in stage s+1. This forward pass is not complete until the output calculation for the blocks in the last computation stage is finished.

In backward passes, the error signals propagate from the blocks in higher stages to those in lower stages. The error criterion function is defined as:

$$\varepsilon = \frac{1}{2}\sum_{l=1}^{M}\left\|\boldsymbol{d}^{l} - \boldsymbol{y}^{l}\right\|^{2} \tag{34}$$

where M is the total number of training patterns. The two vectors $\boldsymbol{d}^{l}$ and $\boldsymbol{y}^{l}$ are the target and actual outputs for the l-th training pattern respectively. Beginning with an initial guess, successive weight vectors are generated such that the error is reduced at each iteration, i.e.:

$$\varepsilon\left(\mathbf{w}\left(k+1\right)\right)<\varepsilon\left(\mathbf{w}\left(k\right)\right) \tag{35}$$

The successive cost reduction can be implemented with a class of gradient methods. The simple gradient steepest descent updates the weights according to the following equation [19][73]:

$$\mathbf{w}\left(k+1\right)=\mathbf{w}\left(k\right)-\eta\left(k\right)\nabla\varepsilon\left(\mathbf{w}\left(k\right)\right) \tag{36}$$

or in its component form:

$$\Delta w_{pq}=-\eta\frac{\partial\varepsilon}{\partial w_{pq}},\quad p\in C,q\in D \tag{37}$$

where η is the learning rate. The increment Δw_{pq} of the internal weight of a block can be deduced according to the generalized delta rule [15][16][73]. We first rewrite the error function in Eq. (34) using the formula that $y_j=v_{mj,2}$ to:

$$\varepsilon=\frac{1}{2}\sum_{l=1}^{M}\sum_{j=1}^{n}\left(d_j^l-v_{mj,2}^l\right)^2 \tag{38}$$

Then, for the weights associated with output nodes connecting to network outside, we can use chain rule to compute the derivative:

$$\begin{aligned}\frac{\partial\varepsilon}{\partial w_{mj,p2}}&=\frac{\partial\varepsilon}{\partial v_{mj,2}}\frac{\partial v_{mj,2}}{\partial g_{mj,2}}\frac{\partial g_{mj,2}}{\partial w_{mj,p2}}\\&=-\sum_{l=1}^{M}\left(d_j-y_j\right)h'\left(g_{mj,2}\right)u_{mj,p}\end{aligned} \tag{39}$$

where h' is the first derivative of the activation function. Let us define the sensitivity of the error criterion to the changes of net input of the output node to be:

$$\begin{aligned}\rho_{mj}&=\frac{\partial\varepsilon}{\partial g_{mj,2}}\\&=-\left(d_j-y_j\right)h'\left(g_{mj,2}\right)\end{aligned} \tag{40}$$

We can then rewrite Eq. (39) in the following equation:

$$\frac{\partial \varepsilon}{\partial w_{mj,p2}} = \sum_{l=1}^{M} \rho_{mj} u_{mj,p} \tag{41}$$

For the weights associated with output nodes in stage *s* that are not connected to network outside, we again use the chain rule to get the derivative:

$$\frac{\partial \varepsilon}{\partial w_{ij,pq}} = \frac{\partial \varepsilon}{\partial v_{ij,q}} \frac{\partial v_{ij,q}}{\partial g_{ij,q}} \frac{\partial g_{ij,q}}{\partial w_{ij,pq}} \tag{42}$$

where

$$\frac{\partial g_{ij,q}}{\partial w_{ij,pq}} = u_{ij,p} \tag{43}$$

In analogy to Eq. (40), let us define the sensitivity for an output node in stage *s* in Eqs. (44), (45) and (46) depending on the node position.

$$\rho_{ij,2} \equiv h'\left(g_{ij,2}\right) \sum_{q \in D_{i,j+1}} \rho_{i,j+1,q} w_{i,j+1,2q} \tag{44}$$

$$\rho_{ij,3} \equiv h'\left(g_{ij,3}\right) \sum_{q \in D_{i-1,j}} \rho_{i-1,j,q} w_{i-1,j,3q} \tag{45}$$

$$\rho_{ij,4} \equiv h'\left(g_{ij,4}\right) \sum_{q \in D_{i+1,j}} \rho_{i+1,j,q} w_{i+1,j,4q} \tag{46}$$

Finally, we get

$$\frac{\partial \varepsilon}{\partial w_{ij,pq}} = \sum_{l=1}^{M} \rho_{ij,q} u_{ij,p} \tag{47}$$

Thus, the weight update for a block can be summarized in the following equation:

$$\Delta w_{pq} = \eta \sum_{l=1}^{M} \rho_q u_p, \quad p \in C, q \in D \tag{48}$$

where η is the learning rate, ρ_q is the sensitivity of the output node, and u_p is the input to the input node. C and D are the index sets of input and output nodes.

The sensitivities of the output nodes of the blocks in calculation stage s are first calculated, and then the internal weights of these blocks are updated by adding the increment Δw_{pq} given in Eq. (48). After calculating the sensitivities and updating the weights of all blocks in stage s, the sensitivities of the nodes of the blocks in stage s-1 are computed with the weights update followed. This procedure continues until the calculation of the blocks in the first computation stage is finished.

GDS operation stops when either the maximum number of epochs is reached or the fitness stops increasing. The maximum number of epochs is tuned based on some simulations, and 8 epochs are found to work well for the test data in this paper. Too big epoch will increase the computation time of every iteration.

3.2.4 Operator Rate Update

An operator rate determines the probability according to which the operator is applied. The proposed update scheme automatically adjusts an operator's rate based on both its effectiveness in improving the fitness and current fitness trend. Operator rates are updated every evolution period and kept unchanged during each evolution period. In the (k+1)-th evolution period, the first step is to assign a probability to each of the operators based on its performance during the k-th period. Then, the operator rates are increased if the maximum fitness has not been improved during the past evolution period. The performance of an operator during the k-th evolution period is measured by the effectiveness defined as:

$$E_k = \frac{N_e(k)}{N_t(k)} \tag{49}$$

where N_e is the number of generations within each evolution period that an operator produces offspring with higher fitness value than that of its parent(s), and N_t is the total number of generations that an operator is selected in the same evolution period. Thus, the value of E_k has a range from 0 (least effective) to 1 (most effective). The effectiveness of an operator determines its probability in the next evolution period, $p(k+1)$, as defined in the following equation:

$$p(k+1) = \begin{cases} \min\{p(k) + \alpha \ln k, p_{\max}\}, & \text{if } E_k \geq 0.5 \\ \max\{p(k) - \alpha \ln k, p_{\min}\}, & \text{if } 0 \leq E_k < 0.5 \end{cases} \tag{50}$$

where $\alpha \ln k$ is rate adjustment during an evolution period, and p_{max} and p_{min} denote the maximum and minimum rate allowed for an operator. The scaling factor α controls the amount of rate adjustment and has been set to 0.02 experimentally. The lower and upper limit for p_{max} and p_{min} are 0 (never apply an operator) and 1 (always apply an operator), respectively. Usually, p_{max} is set to a big value (1.0 in the experiments) to ensure an operator that has been effective can be applied with high frequency; while p_{min} is set to a small nonzero value (0.1 in the experiments) such that an less effective operator still get an chance to be applied. Overall, the update scheme uses high operator rates in early evolution stages, and then gradually decreases the rates of the less effective operators but keeps the higher operator rates for those effective operators.

The next step in rate adjustment considers the fitness trend during the past evolution periods. If the improvement for maximum fitness has been stalled before a solution is found, the algorithm tends to be trapped into a local maximum. It is thus desired to perform more searches in order to help the search escape from the local solution, and the operator rates are accordingly increased to consider such situation as described in the following equation:

$$p'(k+1) = \min\{p(k+1) + \alpha' \ln k, p_{\max}\}, \quad \text{if } \Delta\text{Fitness}_{\max}(k) \approx 0 \tag{51}$$

where $\alpha' > \alpha$ and $\Delta\text{Fitness}_{\max}(k) \approx 0$ means the maximum fitness has not been improved during the k-th evolution period. An operator rate is determined according

to the performance and the current fitness trend. An operator will maintain a high rate if it is effective in generating fitter individuals, otherwise its rate will be gradually decreased. If the maximum fitness has not been improved before a desired solution is found, the operator rates will be increased to do denser searches.

3.2.5 Implementation Platform

The evolutionary algorithm described in preceding sections was implemented under PC environment using Microsoft Visual C++ 6.0 programming language. Figure 16 shows a screenshot of the graphical user interface (GUI) designed for evolutionary optimization of BbNN.

This GUI allows user to change various parameters for BbNN and the evolutionary algorithm through a pop-up window. The BbNN network size and activation functions of neurons can be configured by the user. The parameters that govern the running of the evolutionary algorithm, like population size, maximum fitness, stop generation, and minimum operator rates, etc., can also be changed by the user. Users can save a successful individual BbNN into a file as well as recall it later.

3.3 A Test Example

3.3.1 XOR Problem

The proposed learning algorithm is tested on the simple XOR problem, in which two identical inputs generate an output of one and two different inputs produce negative one. A 2x3 BbNN is chosen for the simulation. The first and second input blocks receive the XOR input and the third output block serves as the network output. A sigmoid activation function of the form:

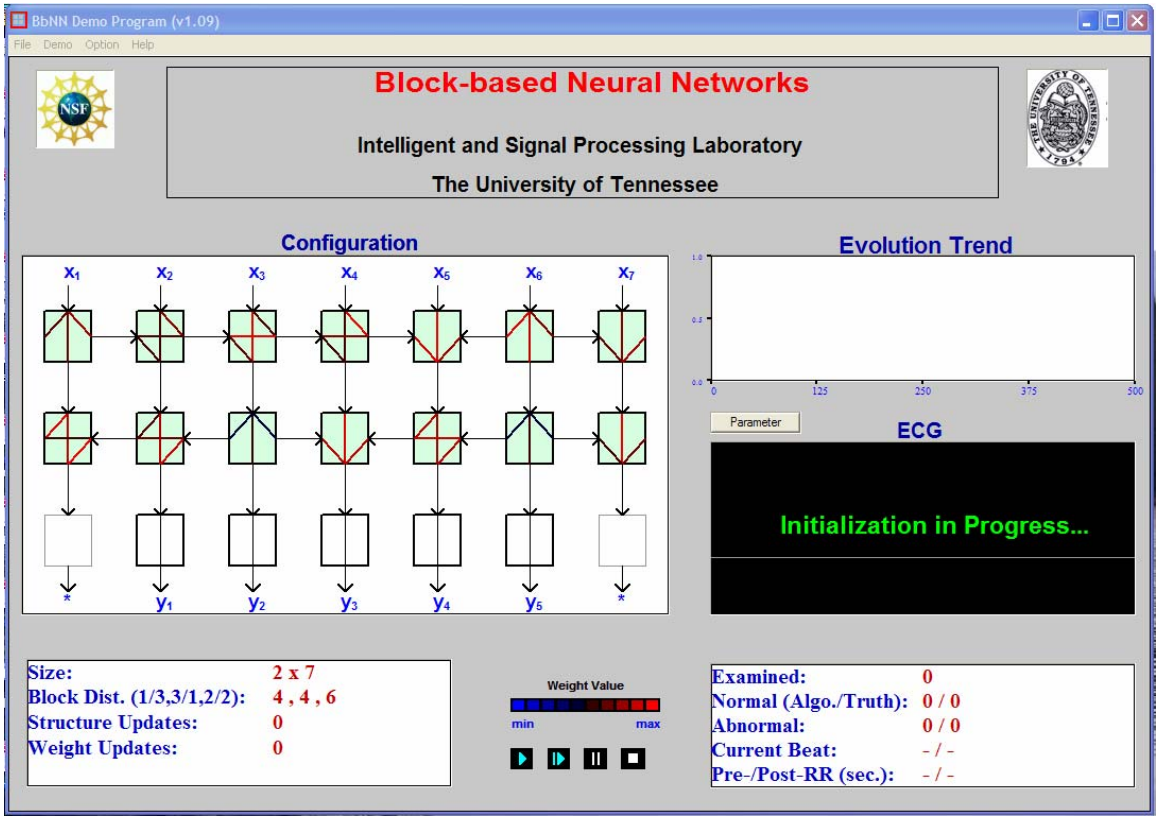


Figure 16: Graphical user interface for block-based neural networks.

$$h(g) = a\left(\frac{2}{1+e^{-bg}} - 1\right) \tag{52}$$

is used, in which a = 1.716 and b = 2/3 are chosen. This set of values makes $h'(0) \approx 1$, the linear range $-1 < g_j < 1$, and the second derivative achieve its extrema at approximately ±2 [74]. The fitness used to evaluate the quality of candidate BbNNs is defined in the following equation:

$$\text{Fitness} = \frac{1}{1 + \frac{1}{M}\sum_{l=1}^{M} \left\| \boldsymbol{d}^l - \boldsymbol{y}^l \right\|^2} \tag{53}$$

where M denotes the numbers of training patterns. d and y are desired and actual output responses. The stop condition is that either the target fitness (0.95) or a maximum epoch (5000) is met. The other parameters of the algorithm are listed in Table 1.

Table 1: Parameters of the evolutionary algorithm for XOR problem

Parameter	Value
Population	80
Maximum Generations	5, 000
Maximum Fitness Value	0.95
GDS Epoch	8
GDS Learning Rate	0.2
Disruptive Pressure	0.9
Tournament Size	2
Rate Update Interval	12
Initial Operator Rate	1.0
Minimum Operator Rate	0.1

3.3.2 Experimental Results

Evolutionary algorithms are applied to evolve the selected BbNN. Figure 17 shows a typical fitness trend. The dotted and solid line corresponds to the average and maximum fitness. The evolution stops when the desired fitness is met after approximately 2,100 generations. Figure 18 demonstrates structure evolution process in terms of the number of occurrences of different structures. A solid line indicates the occurrences of a near-optimal structure. The others represent three non-optimal structures. The number of BbNNs with a near-optimal structure increases during the evolution and becomes dominant and relative stable in the population after about 500 generations.

Figure 19 demonstrates an operator rate update trend. The GDS rate is almost constant with some fluctuations through entire evolution process while the crossover operator favors a high rate with bigger fluctuations than the GDS. The rates for two mutation operators have similar trend that decreases slowly overall and increases sometimes when fitter individuals are generated by the mutations or the fitness has not been improved.

Figure 20 plots the network structure of the evolved BbNN among 100 random trials for XOR classification. The numbers on the arrow are occurrence counts of the same signal flows among 100 individual BbNNs. The output y indicates the category of a test pattern. The redundant output nodes are marked by *. Inputs x_1 and x_2 are the inputs to the BbNN.

In order to study the effect of various parameters, we dissect the evolutionary algorithm to remove some components. The evolutionary algorithms with and without GDS are first compared in terms of their convergence behavior. Both algorithms are run for a fixed number of generations for 100 times. Figure 21(a) shows the averaged maximum fitness along with standard deviation and the number of successful runs after evolution. A trial is successful if the desired fitness value is met before the maximum generation is reached. The GDS operator produces higher averaged maximum fitness and more successful runs compared to EA only case.

Figure 21(a) shows the average generations and running time that the two algorithms take to reach the desired fitness level. The EA with GDS operator takes much less generations and time than without GDS case.

The effect of fitness scaling and adaptive rate adjustment scheme is analyzed using similar approach. Figure 22(a) shows the comparisons of maximum fitness achieved number of successful runs. Figure 22(b) shows the total generations and actual running time to reach the desired fitness level. The EA algorithm using fitness scaling generates more successful runs than the EA without fitness scaling. The use of fitness scaling does not affect much on the generations and running time.

Last, the effect of the use of adaptive operator rates is analyzed. The EA using adaptive rates and the EA with fixed rates (0.8 for crossover and GDS, 0.2 for structure mutation and weight mutation) are run for 100 times with their performance compared. Figure 23(a) shows the comparisons of maximum fitness achieved number of successful runs. Figure 23 (b) shows the total generations and actual running time to reach the desired fitness level. The EA with adaptive rates guarantees higher fitness values and more successful runs on average than the EA using fixed operator rates. In terms of generations and running time, the two algorithms are comparable with the EA with adaptive rates takes a bit more generations and time that is probably due to more searches are used when the search tends to fall into a local maximum.

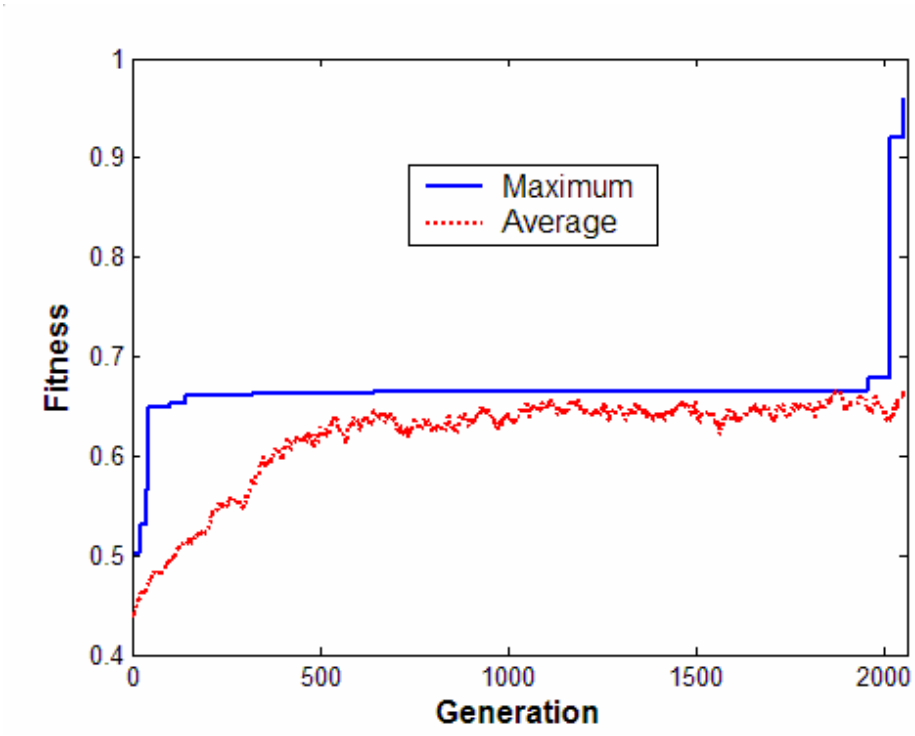


Figure 17: The evolution trend of BbNN for XOR classification.

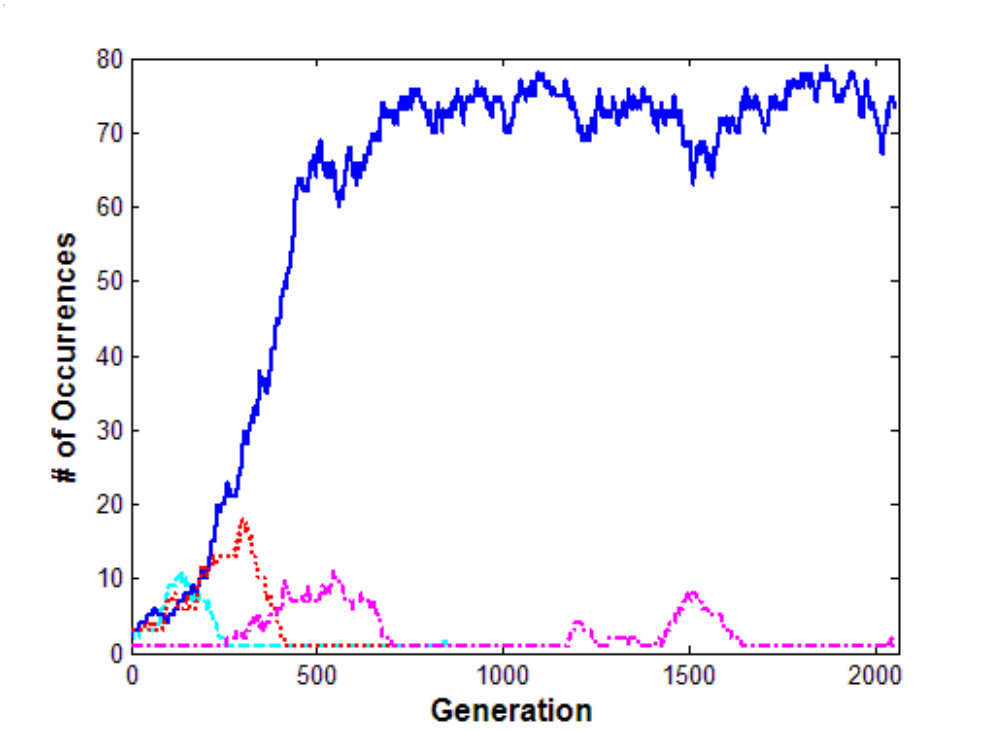


Figure 18: Number of occurrences of particular structures during evolution.

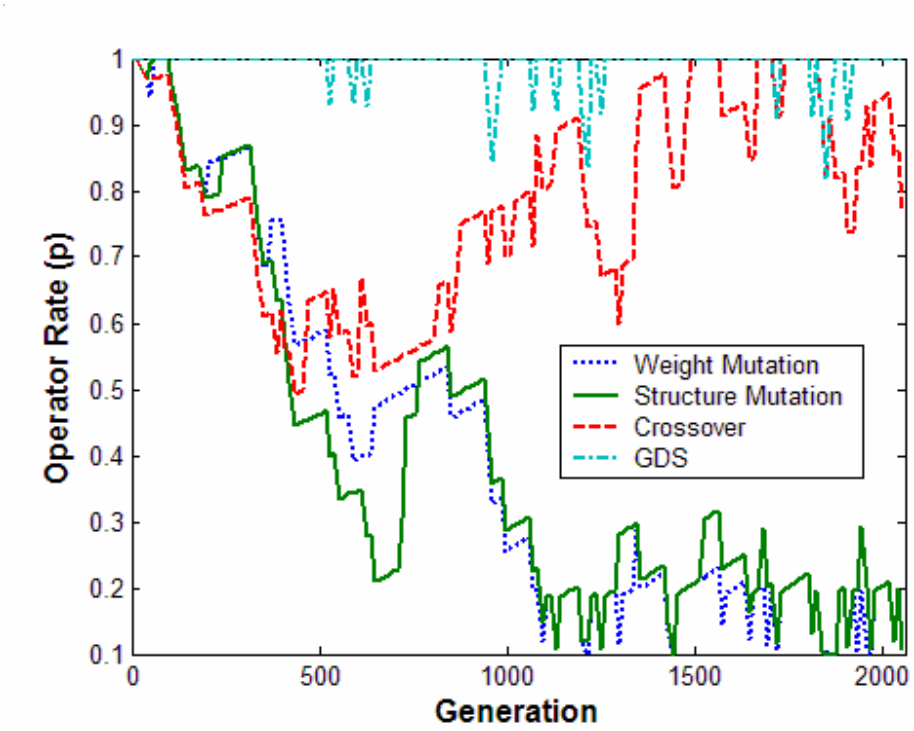


Figure 19: An adaptive operator rate adjustment scheme.

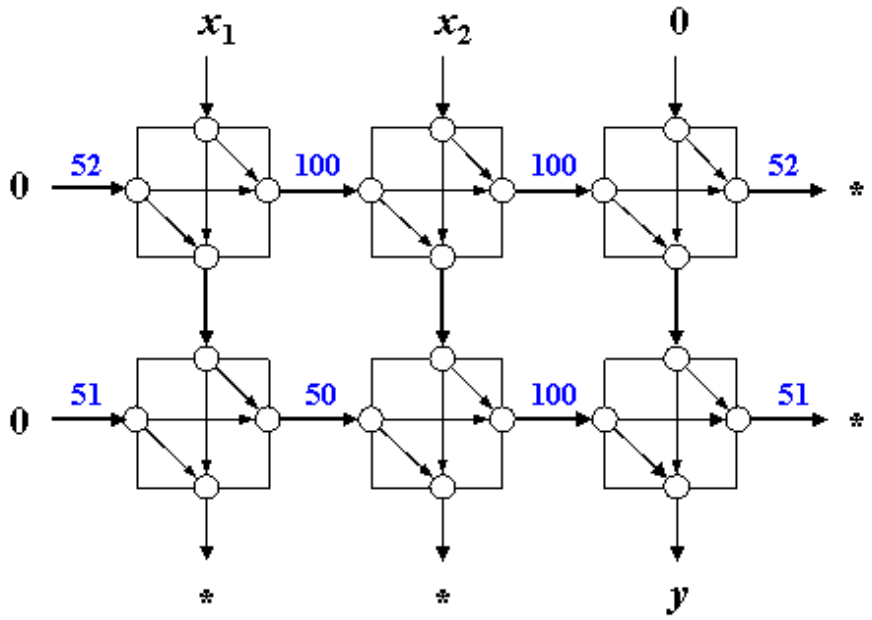
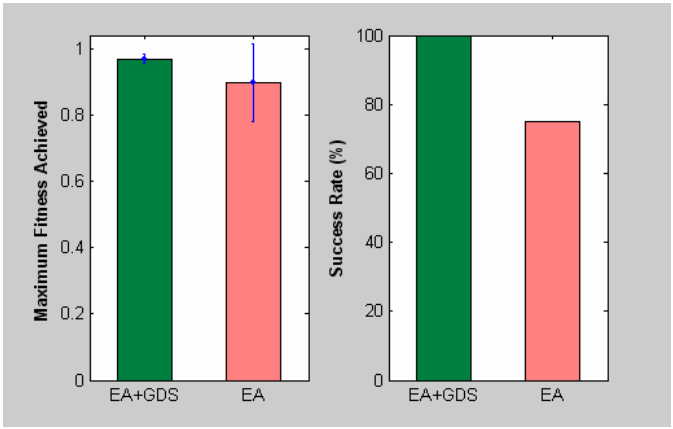
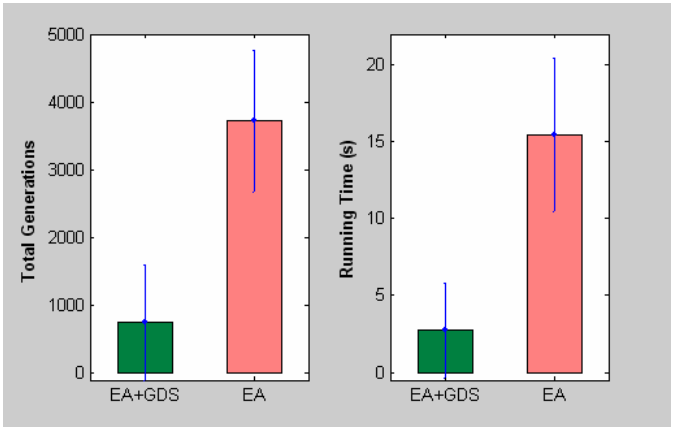


Figure 20: The evolved BbNN for XOR classification.

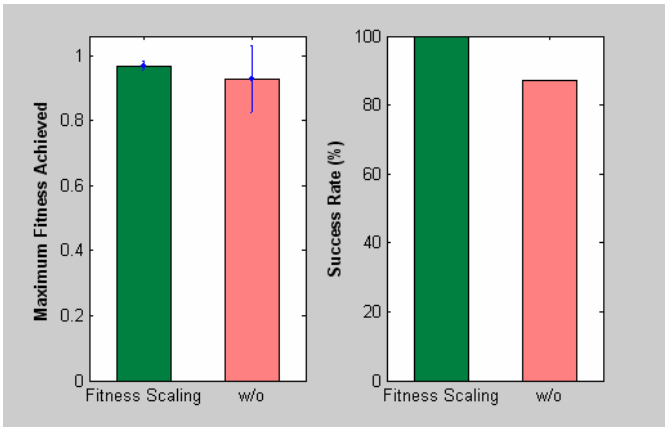


(a)

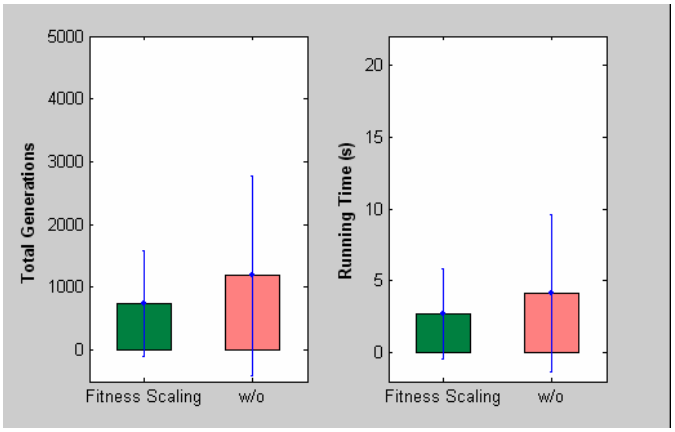


(b)

Figure 21: Comparison of the evolutionary algorithm with and without GDS in terms of (a) final fitness achieved and (b) convergence speed.

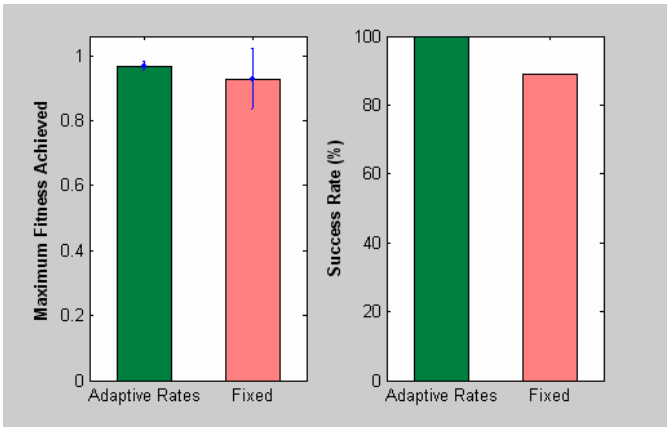


(a)

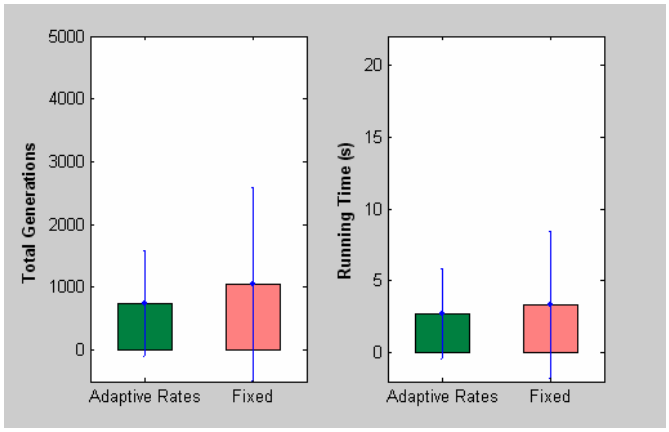


(b)

Figure 22: Comparison of the evolutionary algorithm with and without fitness scaling in terms of (a) final fitness achieved and (b) convergence speed.



(a)



(b)

Figure 23: Comparison of the evolutionary algorithm with adaptive and fixed rate scheme in terms of (a) final fitness achieved and (b) convergence speed.

Chapter 4 PERSONALIZED ECG HEARTBEAT CLASSIFICATION

Electrocardiogram (ECG) has become an important routine clinic practice to monitoring heart activities. Analysis of heartbeat patterns may reveal the symptoms indicating that the heart needs immediate attention. This chapter describes personalized ECG heartbeat classification using block-based neural networks, which is motivated by the observation that a classifier with fixed structure and internal weights and trained with a limited number of data may not be able to tackle the big variations in ECG signals. In the following, an introduction on ECG signal classification is first presented.

4.1 Introduction

4.1.1 Electrocardiogram

ECG is a diagnostic tool that records the electrical activity of heart. The commonly used ECG is the standard twelve lead ECG that examines the electrical activity of the heart from twelve different points of view including V1, V2, …, V6, I, II, III, aVR, aVL and aVF [75]. While no single point of view could provide a complete picture of the heart, the twelve points of view provide complementary information about the heart.

There are total three types of waves occurred in a single heartbeat. The first one is called P wave that corresponds to the contractions of both atrial of a heart. The second is a series of three waves, known as QRS complex that reflects the ventricular contractions. The QRS complex has been an important feature of heartbeat signals in the detection of arrhythmia waveforms. The last T wave is recorded when ventricles

are repolarizing. The three basic waves occur sequentially in the order of P, QRS and T wave. Figure 24 illustrates a single heartbeat. Figure 25 shows the first five beats of ECG record #201 from MIT-BIH Arrhythmia Database [1].

4.1.2 Challenges in ECG Signal Classification

Correctly classifying heartbeats is the first important step toward identifying an arrhythmia. AAMI recommended practice groups the normal and various abnormal types into five heartbeat classes that include class *N* (beats originating in the sinus node), class *S* (supraventricular ectopic beats), class *V* (ventricular ectopic beats), class *F* (fusion beats), and class *Q* (unclassifiable beats) [9].

It has been a challenge to classify ECG beats in achieving high performance possibly due to the big variations in ECG heartbeat patterns. A large inter-individual variability in the ECG waveforms is observed within different individuals and patient

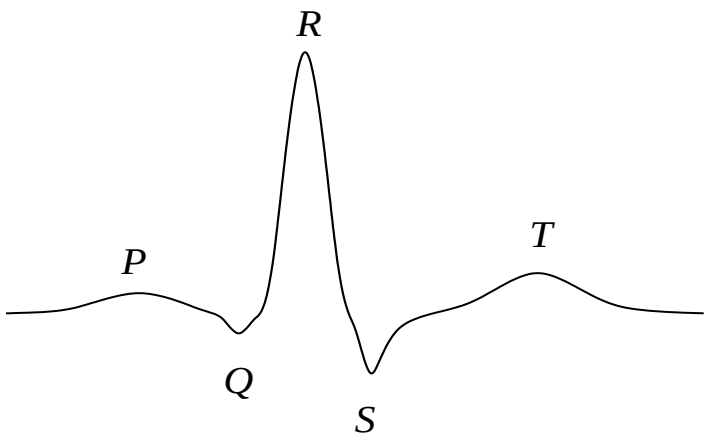


Figure 24: The three waves in a single heartbeat.

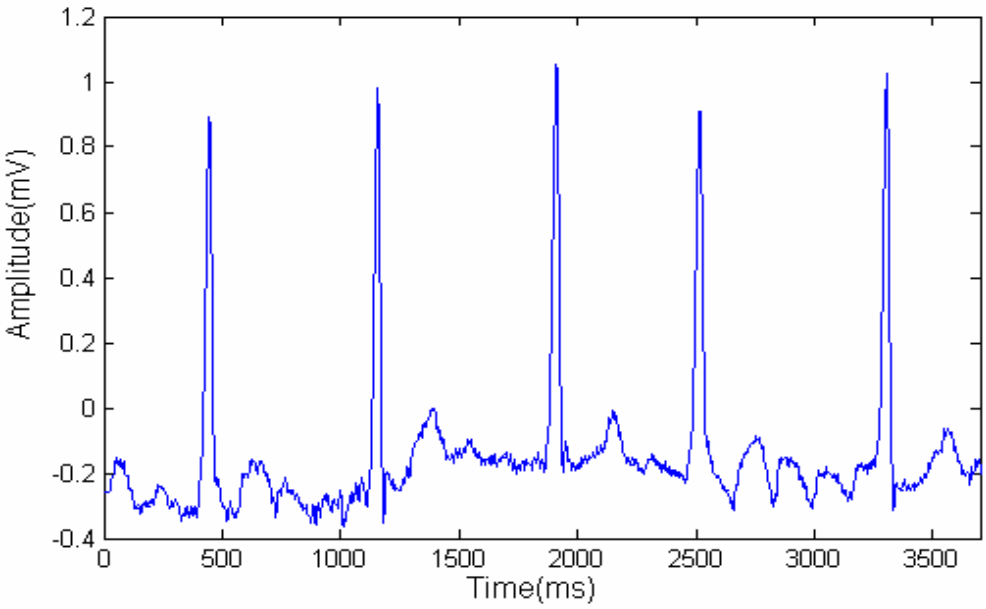


Figure 25: Heartbeat examples from MIT-BIH Arrhythmia Database.

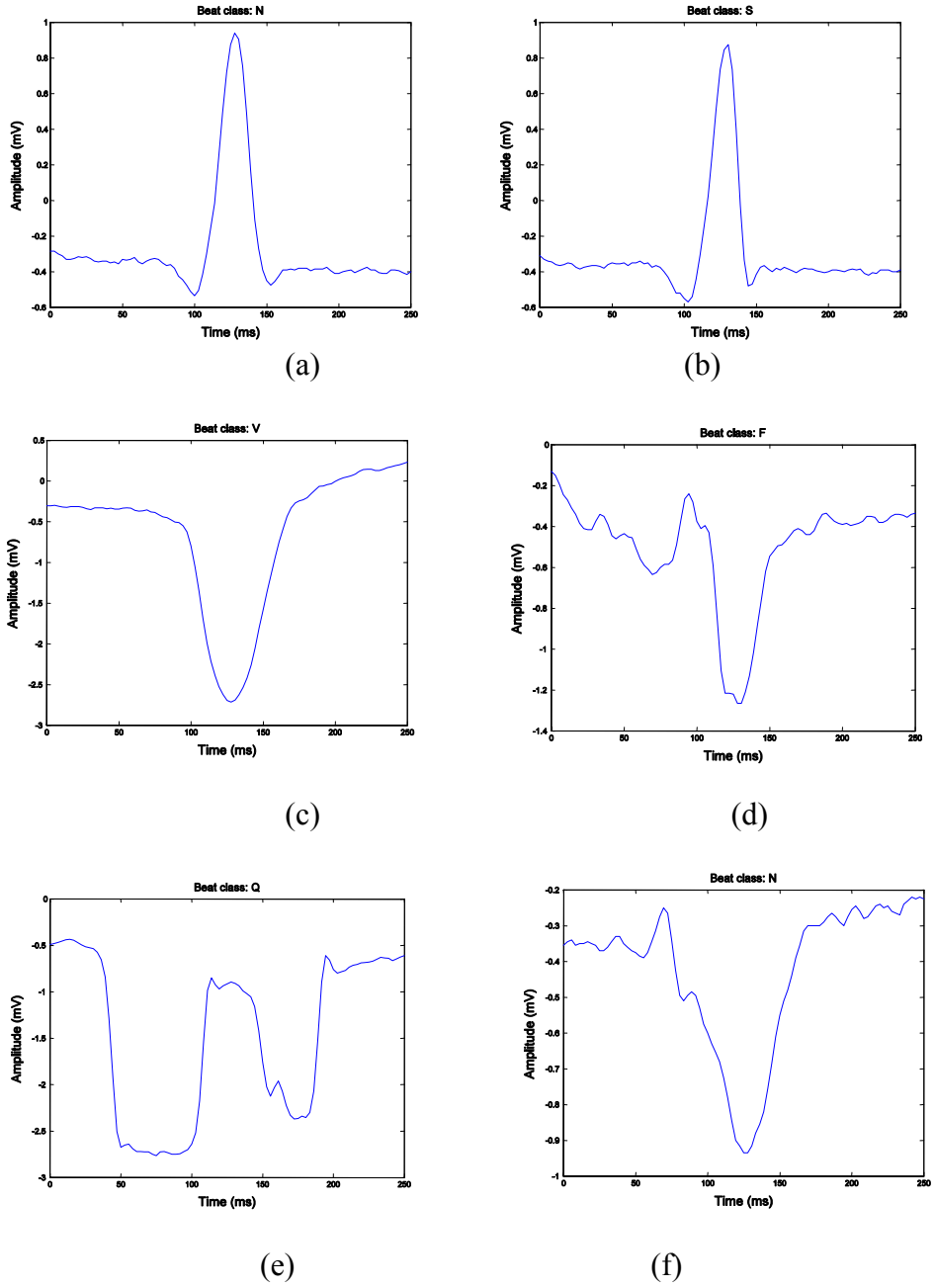


Figure 26: Examples of AAMI beat classes from MIT-BIH Arrhythmia database, (a) Class *N* (beat #1 of record 100), (b) Class *S* (beat #8 of record 100), (c) Class *V* (beat #1907 of record 100), (d) Class *F* (beat #471 of record 108), (e) Class *Q* (beat #361 of record 101), (f) Class *N* (beat #1 of record 108).

groups due to physiological and geometrical differences between the hearts [76]. Consequently, the sensitivity and specificity of ECG classification algorithms are often low. Figure 26 shows example beats of each of the five classes from MIT-BIH Arrhythmia database [1]. Note that the beats belonging to the same normal class in Figure 26(a) and (f) demonstrate significant morphological difference; while the different classes of *N* and *S* beats shown in Figure 26(a) and (b) possess quite similar shapes.

4.1.3 Previous Approaches for ECG Classification

In the past decades, a number of methods have been proposed to classify ECG heartbeats into different categories [3][77]-[90]. Among them, different types of features are first extracted from detected heartbeats including morphological features, heartbeat intervals, frequency domain features and wavelet transform coefficients, etc. After the extraction of features, a certain classification technique is applied to classify the heartbeats into normal or one of the abnormal types. Such methods include linear discriminant analysis (LDA), support vector machine (SVM), artificial neural networks, mixture-of-experts methods [86], and statistical Markov models [87][88]. Unsupervised clustering of ECG complexes using self-organizing maps (SOM) is also proposed.

Hu *et al*. [86] proposed an artificial neural network method based on MLP trained with BP algorithm. They used the original data samples as input to the network and the dataset contains 6,474 QRS complex templates from MIT-BIH Arrhythmia database. A two-layer MLP network of the size 51-25-2 reported an average accuracy of 90% for the classification of normal and abnormal heartbeats for the selected dataset.

The method in [86] studied the problem of distinguishing VEB from non-VEB beats. The algorithm exploited a Mixture-of-Experts (MOE) method and employed a test set of 20 recordings that excluded records without premature ventricular contractions (PVCs). A global expert was developed using both unsupervised self-

organizing map (SOM) and supervised learning vector quantization (LVQ) based on a common set of ECG training data. A local expert was developed similarly but based solely on patient-specific training data. The decisions from both classifiers are then linearly combined using coefficients from a gating network that is trained with another set of patient-specific ECG data. The MOE method achieved an accuracy of 94.0% for distinguishing ventricular ectopic beats (VEB) from non-VEB heartbeats. Despite of the performance improvement, this work was limited at the detection of VEB beats. Besides, the fact that three separate neural networks need to be trained for a single patient makes this method somehow inefficient.

Lagerholm *et al*. [79] proposed a method for unsupervised clustering of ECG heartbeats into 25 clusters. Their method uses Hermite function representation of QRS complexes and self-organizing maps (SOM). Their clustering results correspond to a classification rate of 98.5% if the dominant beat of a cluster can be correctly identified.

Chazal *et al*. [77] proposed a method that consists of linear discriminants (LDs) and various sets of morphology and heartbeat interval features. The 44 non-paced recordings from MIT-BIH database were divided into two sets with approximate proportion of beat types and total beat numbers (about 50,000 heartbeats). The first set was used to evaluate the performance of different classifier configurations in order to select a final classifier. The second set served as the independent test data used to evaluate the final performance of the selected classifier. For each heartbeat, various features based on ECG morphology, heartbeat intervals and RR-intervals were extracted and combined into eight feature sets. The performance of each feature set in classification was then evaluated to determine the best configuration that is then used to classify the beats in the second datasets. The performance evaluation for the second dataset reported an accuracy of 97.4% for VEB detection and 94.6% for SVEB detection.

Osowski *et al*. [83] presented a method using support vector machine (SVM) for heart beat recognition. Two different types of features, Hermite characterization and High Order Statistic, have been used in the classification system. The training

and test data include 6690 and 6095 heartbeat patterns selected from MIT-BIH database. The overall accuracy of heart beat recognition is 95.91% for normal rhythm and 12 different types of arrhythmias.

Despite the widely available methods, their performances leave room for further improvement. The sensitivity reported is usually insufficient. For example, for VEB detection, the method in [86] reported a sensitivity rate of 77.7% and a classification rate of 97.4%, and the method in [77] achieved a sensitivity rate of 82.6% and a classification rate of 94.0%. For SVEB detection, 75.9% sensitivity rate and 94.6% classification rate are reported in [77]. Apparently, there is a need for better classification performance, especially higher sensitivity rate.

4.2 Personalized ECG Signal Classification

4.2.1 Evolvable Hardware Platform

Advance of embedded systems and other related resources on many of the present generation FPGA boards enables the on-board evolution of block-based neural networks. Figure 27 shows a schematic diagram of the proposed method, in which dotted and solid arrows correspond to respective training and testing phase.

Hermite function transform extracts the features from the incoming ECG heartbeats and input the features to the other two blocks. An evolutionary algorithm finds the structure and weights of a selected BbNN based on training patterns. The "trained" network is obtained, is downloaded into the reconfigurable FPGA chip. The configured BbNN classifies the current ECG beat into one of five classes. If the performance of the evolved network is degraded due to changes in the environment or the subject, the evolution switch will activate the BbNN evolution block, and the search for a fitter BbNN classifier is initialized. Hence, the BbNN classifier continues to reconfigure itself in order to provide consistent performance.

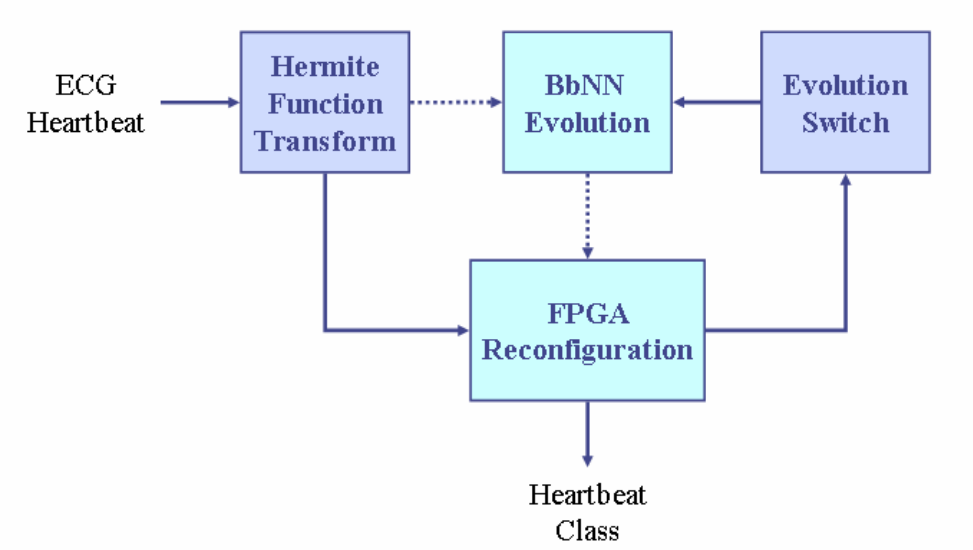


Figure 27: Heartbeat monitoring using block-based neural networks.

4.2.2 The ECG Data

The MIT-BIH Arrhythmia Database [1] provides the ECG signals used in the experiments. The database contains 48 records obtained from 47 different individuals (Two records came from the same patient). Each record contains 2-channel ECG signals measured for 30 minutes. Twenty-three records (numbered from 100 to 124, inclusive with some numbers missing) serve as representative samples of routine clinical recordings. The remaining 25 (numbered from 200 to 234, inclusive with some numbers missing) records include unusual heartbeat waveforms such as complex ventricular, junctional, and supra-ventricular arrhythmias.

Continuous ECG signals were filtered using a bandpass filter with a passband from 0.1 to 100 Hz. Filtered signals were then digitized at 360 Hz. The beat locations

are automatically labeled at first and verified later by independent experts to reach consensus. The whole database contains more than 109,000 annotations of normal and 14 different types of abnormal heartbeats.

Specific software or interface library are needed to view or read the signals contained in this database. *Wave* is a useful computer program that can be used to view and analyze ECG signals [91], but it does not provide a way to extract the signals from an ECG recording. The Waveform Database interface library (WFDB library) [91] is a set of functions that are callable by C functions to access digitized and annotated signals. WFDB_tools is a collection of Matlab functions that enable Matlab users to have full access to the WFDB library within Matlab environment. Figure 28 shows the procedure of reading ECG signal samples from an ECG recording.

The normal and various abnormal types have been combined into five heartbeat classes according to AAMI recommended practice [9] that include class *N* (beats originating in the sinus node), class *S* (supraventricular ectopic beats), class *V* (ventricular ectopic beats), class *F* (fusion beats) and class *Q* (unknown beats). The

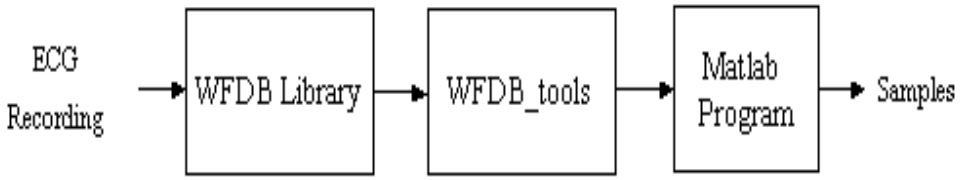


Figure 28: The procedure in reading ECG signals.

mapping from the MIT-BIH heartbeat types to the AAMI heartbeat types is summarized in Table 2.

The records having paced beats were excluded for experiments. The remaining records are divided into two sets. The first set contains 20 records numbered from 100 to 124 and is intended to provide common training data. The second set is composed of the rest records numbered from 200 to 234 and each of the records in this set will be used in the test. The training data for a patient consist of two parts, the first part coming from the common set and being the same for all testing patients. The other part are the heartbeats from the first five minutes of the patient's ECG recording, which conforms to the AAMI recommended practice that allows at most 5-minute of recordings from a subject to be used for training purpose [86][9]. The remaining beats of the record serve as test patterns.

4.2.3 Feature Extraction

Basis function representations have been shown to be an efficient feature extraction method for ECG signals [92][93]. The most useful basis functions include Karhunen-Loeve (KL) and Hermite functions. While KL expansion provides optimal signal representation in the mean square error sense, Hermite basis function expansion has a unique width parameter that is an efficient parameter to represent ECG beats with different QRS duration. Hermite basis functions have been widely used in representing QRS complexes [80][81] and ECG data compression [94]. The coefficients of Hermite expansions characterize the shape of QRS complexes and serve as input features.

Hermite basis functions are given by the following equation:

$$\phi_l(t,\sigma) = \frac{1}{\sqrt{\sigma 2^l l! \sqrt{\pi}}} e^{-t^2/2l^2} H_l\left(\frac{t}{\sigma}\right) \tag{54}$$

Table 2: Mapping from MIT-BIH heartbeat types to AAMI heartbeat classes

AAMI heartbeat class	**MIT-BIH heartbeat types**
N (Sinus node beat)	Normal beat Left branch block beat Right branch block beat Atrial escape beats Junctional escape beat
S (Supraventricular ectopic beat)	Atrial premature beat Aberrated atrial premature beat Junctional premature beat Supraventricular premature beat
V (Ventricular ectopic beat)	Premature ventricular contraction Ventricular escape beat
F (Fusion beat)	Fusion of ventricular and normal beat
Q (Unknown beat)	Paced beat Fusion of paced and normal beat Unclassified class

where σ is the width parameter and approximately equal to the half-power duration. $H_l(t/\sigma)$, called the Hermite polynomials, are defined in Eq. (55). Figure 29 shows the first five Hermite basis functions.

$$H_n(x)=\begin{cases}1, & n=0\\ 2x, & n=1\\ 2xH_{n-1}(x)-2(n-1)H_{n-2}(x), & n\geq 2\end{cases} \tag{55}$$

Hermite functions are orthonormal for any fixed value of width σ:

$$\sum_{t=-\infty}^{\infty}\phi_q(t,\sigma)\phi_l(t,\sigma)=\delta_{ql} \tag{56}$$

This useful property enables the calculation of expansion coefficients of an arbitrary signal. Specifically, the QRS complex is extracted as a 250-ms window centered at the R peak, which is sufficient to cover both normal and wider-than-normal QRS signals [75][81]. If we denote a QRS complex as $x(t)$, then it can be approximated by a combination of Hermite basis functions:

$$\hat{x}(t)=\sum_{l=0}^{L-1}c_l(\sigma)\phi_l(t,\sigma) \tag{57}$$

where $\hat{x}(t)\to x(t)$ as $L\to\infty$. Multiplying $\phi_q(t,\sigma)$ to both sides of Eq. (57) and summing them up over time, we can get the set of coefficients by applying the orthonormal property in Eq. (56):

$$c_l(\sigma)=\sum_{t=-\infty}^{\infty}\phi_l(t,\sigma)x(t) \tag{58}$$

In Eq. (58) the expansion coefficients c_l depend on the width σ. In order to determine the optimal σ, we stepwise increase σ up to its upper bound to minimize the summed square error between the actual and approximated complex. The upper bound of the width parameter σ for a given L is determined using the algorithm described in [79].

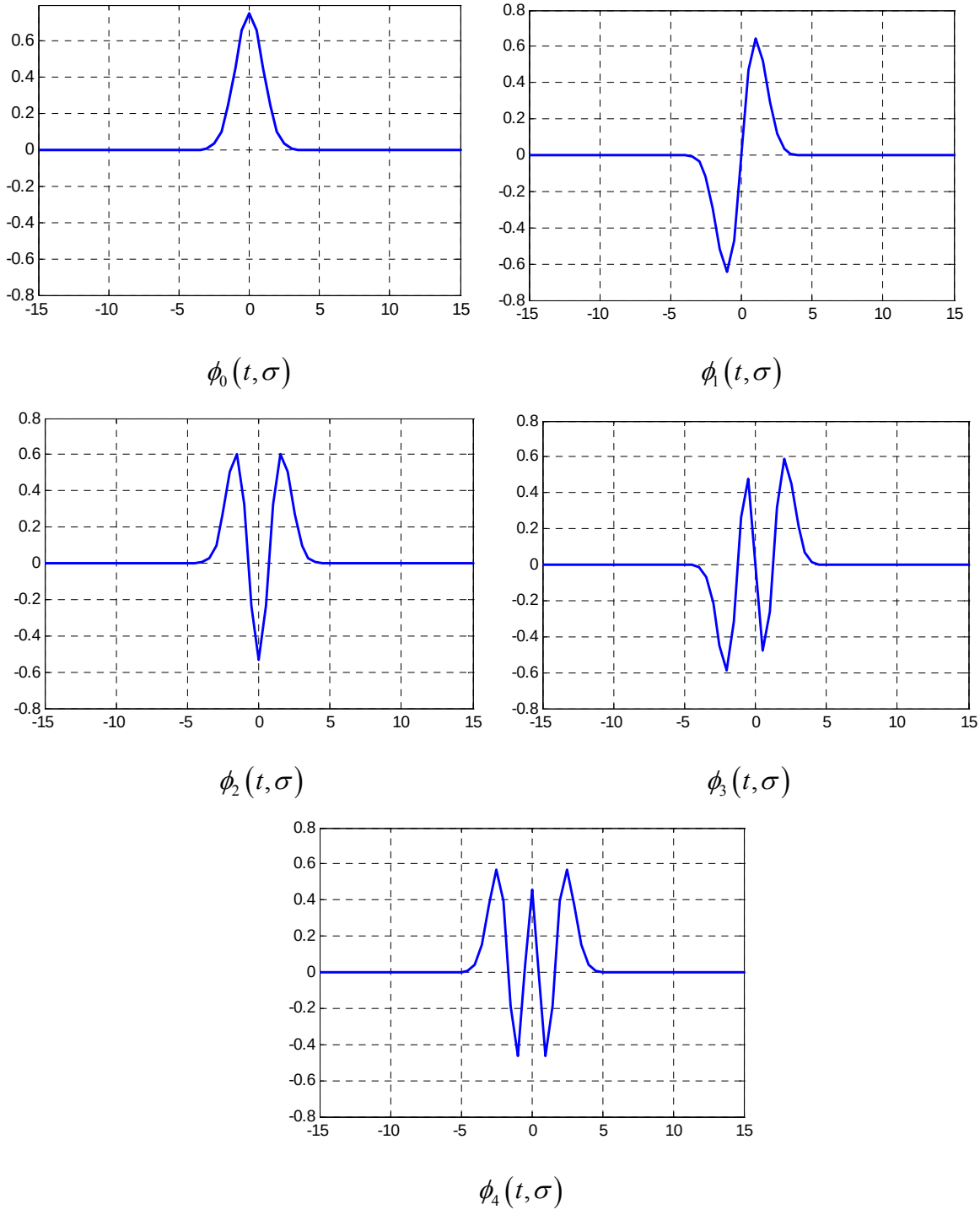


Figure 29: The first five Hermite basis functions with $\sigma = 1$.

Different number of expansion coefficients can be used to approximate a QRS complex. The approximation error depends on the number of coefficients. There is a tradeoff between approximation error and computation time. More coefficients lead to smaller errors, but the computation burden significantly increases when more basis functions are used. We decided to use five Hermite functions that allow for good representation for the QRS complexes and fast computation of the coefficients as well. When five Hermite functions are used, the representation error for different types of beats is acceptable according to a study in [79]. Besides the basis function coefficients c_i and width parameter σ, the time interval between two neighboring R peaks t_R is included to discriminate normal and premature heart beats.

4.2.4 Fitness Function

Fitness function evaluates the quality of the problem solutions. The fitness of a BbNN individual is defined in the following equation:

$$\text{Fitness} = \frac{\beta}{1+\frac{1}{n_{ob}M_1}\sum_{l=1}^{M_1}\left\|\boldsymbol{d}_c^l - \boldsymbol{y}_c^l\right\|^2} + \frac{\gamma}{1+\frac{1}{n_{ob}M_2}\sum_{l=1}^{M_2}\left\|\boldsymbol{d}_s^l - \boldsymbol{y}_s^l\right\|^2} \tag{59}$$

where β and γ are two weights summing to one. M_1 and M_2 are the numbers of samples in the common and patient-specific training data, respectively. n_{ob} is the number of output blocks.

Both common and patient-specific training patterns are considered in the fitness function. While patient-specific data may serve as the training data for evolving BbNN specialized for a patient, the inclusion of common training data is useful when the small segment of patient-specific samples contains few arrhythmia patterns. To construct the common dataset, representative beats from each class are randomly sampled. Since the number of beat instances from each class differs drastically with the normal beats having ten times more than the other beat types, it is important to construct the common dataset in a 'fair' way in order to prevent a few classes dominating the common training data [79][86]. To this end, no *N*-type beats are

selected from the common dataset (there always exist sufficient *N*-type beats in patient-specific data); Different percentages of the other four classes are chosen as: 5% of *V*-type (64 beats), 30% of *S*-type (58 beats), all *F*-type (13 beats) and all *Q*-type (7 beats). Therefore, there are total 142 beats in the common set. The number of beats in the patient-specific training data varies due to the difference in the heart rates of different patients. The user-defined weighting constants β and γ control the relative importance of each term in the final fitness. In the simulation reported in the following, the two constants equals 0.2 and 0.8, respectively. This assignment of control values implies that the correct classification of patient-specific patterns is of more importance.

4.3 Experimental Results

4.3.1 Training Parameters

Selection of a BbNN network structure needs to be considered from two aspects. First, the number of columns has to be equal to or greater than the number of input features. Second, the number of rows should be selected so that the network has sufficient complexity to model a given problem. A small network size is preferred, provided that it achieves the desired performance. Too big a network runs the risk of overfitting that might cause poor generalization performance, and require a more complex optimization process because of high degrees of freedom in search space. In the experiment, a 2×7 network was selected as a minimum-size BbNN that accepts seven inputs. The desired output for the target category and non-target categories are respectively set to 1.0 and –1.0. There are total five classes (*N*, *S*, *V*, *F* and *Q*), so the desired output for a training pattern is a vector of five elements. The parameters used for the EA in the following simulation are listed in Table 3. The same set of parameters has been used for all test records without fine-tuning for specific patients.

4.3.2 Evolution Trends

We apply the evolutionary algorithm to evolve the selected BbNN for a patient. A typical fitness trend is shown in Figure 30. The dotted and solid line corresponds to the average and maximum fitness, respectively. The evolution stops when the stop fitness is met after approximate 1200 generations.

Figure 31 demonstrates the structure evolution process, in which the percentage of occurrences of several structures is shown. The solid line indicates the occurrences of a near-optimal structure. The other three lines represent three non-optimal structures. The number of BbNNs with a near-optimal structure increases during the evolution and become dominant in the population after about 600 generations.

The operator rate trend is demonstrated in Figure 32. The GDS rate is almost constant with some fluctuations during the whole evolution. The crossover maintains an overall high rate with bigger fluctuations than GDS. The rates for two mutation operators have similar trend that decreases slowly overall and increases sometimes when fitter individuals are generated by mutations or the fitness has not been improved (cf. to Figure 30).

Figure 33 shows the effect of the adaptive rate update scheme in terms of the convergence speed of maximum fitness. In the fixed rate case, the GDS and the crossover use a high rate (0.8) and the mutations use a low rate (0.2). The fitness trends are averaged over 10 independent runs. Each error bar shows a standard deviation of the maximum fitness at every 200 generations. The error pattern for the fixed rate case is similar to that for the adaptive rate update scheme. The EA with adaptive rates achieves noticeably higher fitness value on average after the conventional evolution procedure. The fitness scaling also enhances fitness levels. The EA+GDS algorithm without fitness scaling produces the mean and maximum fitness values of 0.900 and 0.909 in 10 trials, which can be compared to 0.920 and 0.942 when the fitness scaling is applied.

In order to study the effect of the proposed GDS operator, we dissect the evolutionary algorithm to remove the GDS operator. The evolutionary algorithms

with and without GDS are then compared in terms of their convergence speed. Both algorithms are run for a fixed number of generations 10 times. Figure 33 shows the averaged maximum fitness trend during evolution. While the EA without GDS slowly improves the fitness, the GDS enhanced EA quickly increases the fitness initially and at a slower speed at the last stage.

A BbNN classifier is evolved specifically for each patient. Both structure and internal weights of a BbNN are optimized with the evolutionary algorithm. Figure 34 shows the network structure of the BbNNs evolved from 24 patients. The numbers on the arrow are occurrence counts of the same signal flows among 24 individual BbNNs. The maximum output y_i indicates the classified ECG type. The redundant output nodes are marked by *. Inputs x_1, …, x_5 to the BbNN are the five Hermite transform coefficients c_1, ..., c_5. The input x_6 is the Hermite width σ and x_7 is the time interval t_R between two neighboring R-peaks.

Table 3: Parameters of the evolutionary algorithm for ECG signal classification

Parameter	Value
Population	80
Maximum Generations	3, 000
Maximum Fitness Value	0.92
GDS Epoch	8
GDS Learning Rate	0.001
Disruptive Pressure	0.6
Tournament Size	2
Rate Update Interval	12
Initial GDS Rate	1.0
Minimum Operator Rate	0.1

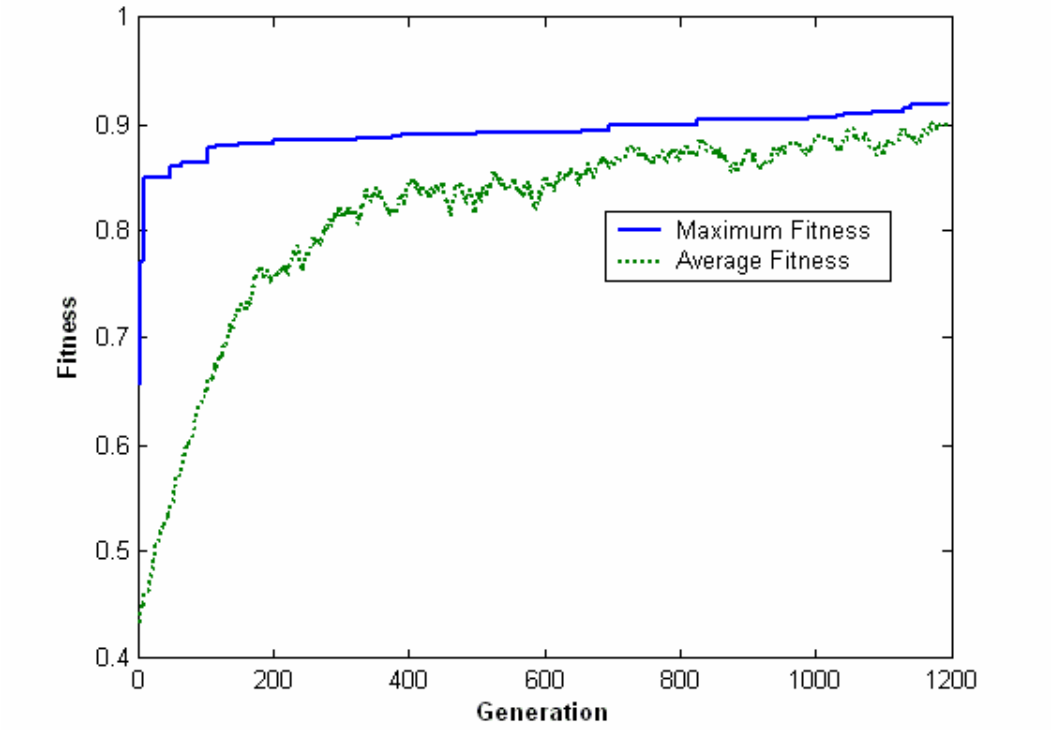


Figure 30: Fitness trend of BbNN evolution.

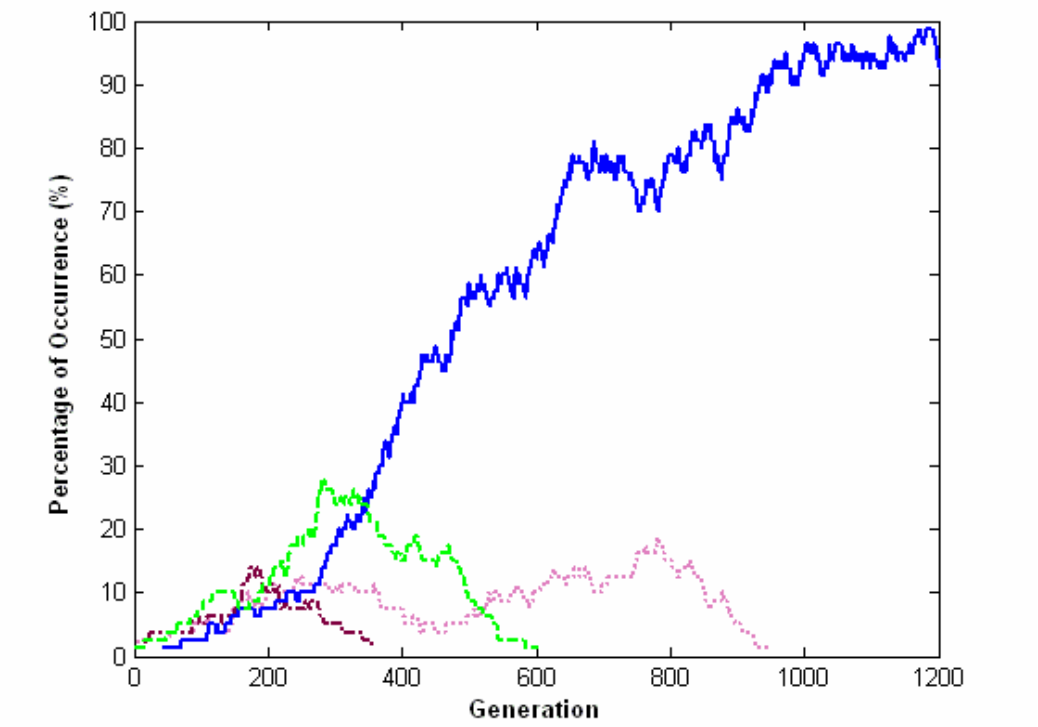


Figure 31: The percentage of occurrences of particular structures during the evolution.

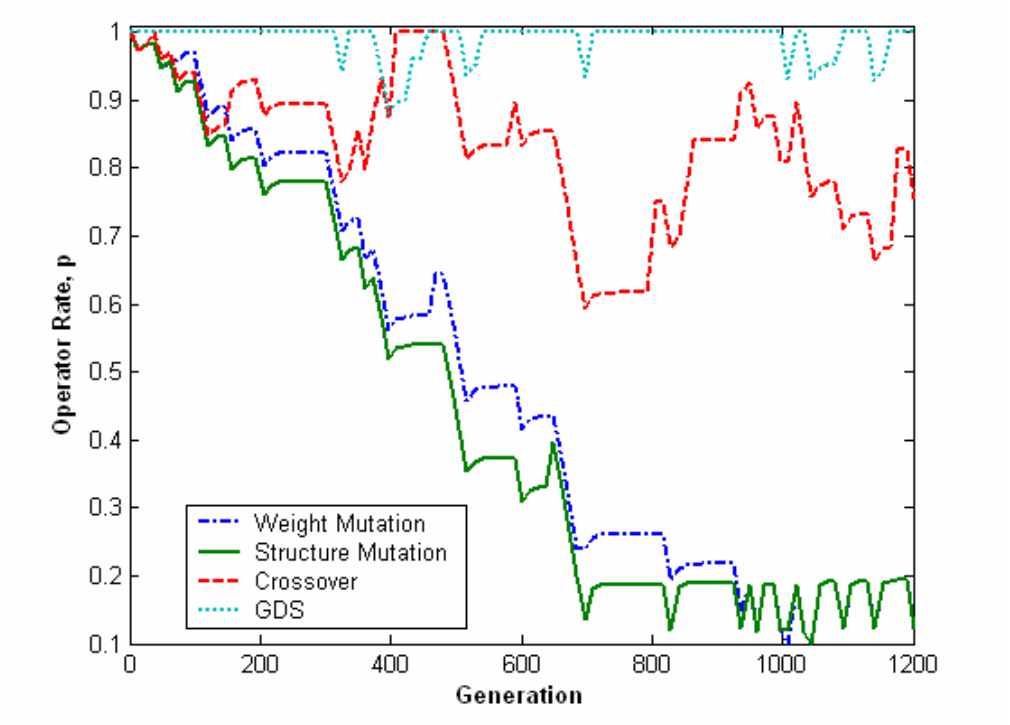


Figure 32: Evolution trend of operator rate.

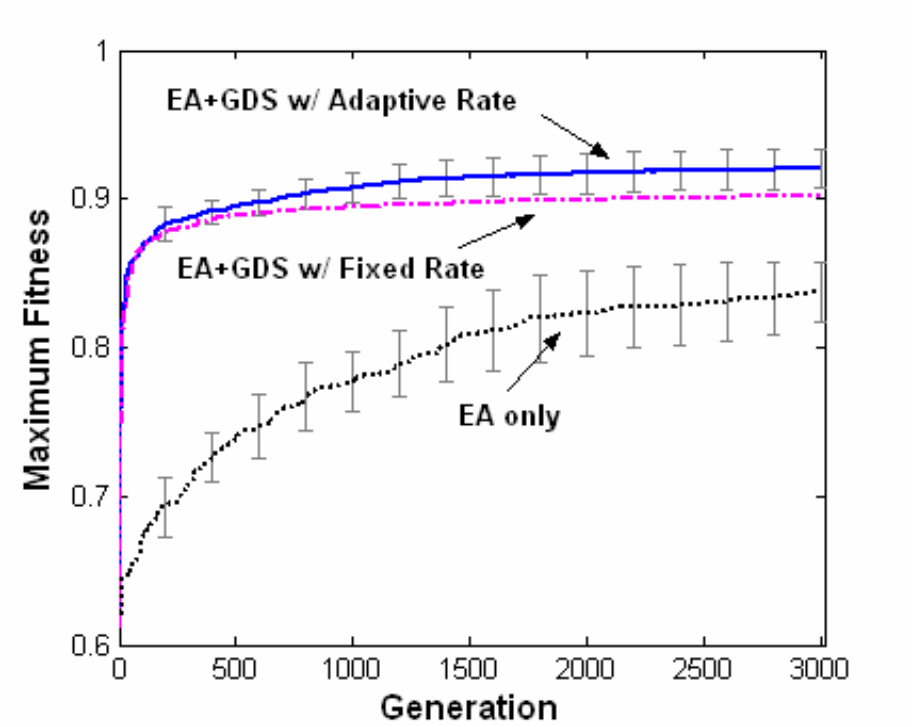


Figure 33: Comparison of fitness trend between EA with adaptive and fixed rates.

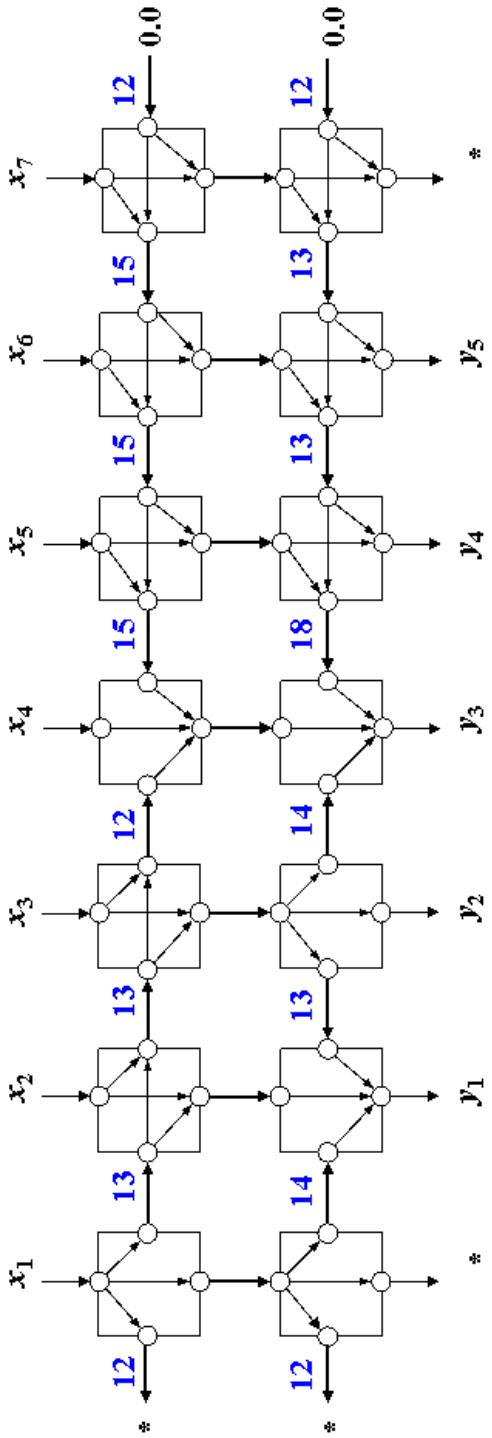


Figure 34: The BbNN structure evolved from 24 patients.

4.4 Classification Results

Heartbeat classification is performed for test records. Classification statistics of ECG heartbeat patterns for test records are reported in Table 4. Two sets of performance are reported: the detection of VEBs and detection of SVEBs in accordance with the AAMI recommendations [9][77]. Table 5 defines the terms of true positive (TP), true negative (TN), false negative (FN) and false positive (FP) for the detection of VEBs and SVEBs.

Four performance measures, classification accuracy (*Acc*), sensitivity (*Sen*), specificity (*Spe*), and positive predictivity (*PP*), are further defined in the following. Classification accuracy is defined as the ratio of the number of correctly classified patterns (TP and TN) to the total number of patterns classified. Sensitivity is the correctly detected events (VEBs/SVEBs) among the total number of events and equals to TP divided by the sum of TP and FN. Specificity refers to the rate of correctly classified non-events (non-VEBs/non-SVEBs) and is therefore the ratio of TN to the sum of TN and FP. Positive predictivity refers to the rate of correctly classified events in all detected events and is therefore the ratio of TP to the sum of TP and FP.

The classification of ventricular fusion (*F*) or unknown beats (*Q*) as VEBs does not contribute to the calculation of classification performance according to AAMI recommended practice [9][77]. Similarly, performance calculation for detecting SVEBs does not consider the classification of unknown beats as SVEBs. Each experiment was repeated ten times and the averaged results were recorded. Each experiment was performed ten times. The coefficient of variation (CV) measures dispersion of a probability distribution and is defined as the ratio of standard deviation to mean, which allows comparison of the variation of populations that have significantly different mean values. The CVs for true positive beats of *N*, *S*, *V*, and *F* types are 0.9%, 1.3%, 4.0%, and 48.3%, respectively. The variations for *N*, *S* and *V* types are small but type *F*, which has a very small number of instances.

Table 4: Beat-by-beat classification results.

Truth	Classification Result				
	N	S	V	F	Q
N	41303	311	198	24	0
S	1051	1181	101	2	0
V	431	198	4165	14	1
F	152	48	193	219	0
Q	5	0	2	1	0

Table 5: Definition of TP, FP, TN and FN for detection of VEBs and SVEBs.

	VEB					SVEB				
Truth	Classification Result									
	N	S	V	F	Q	N	S	V	F	Q
N	TN	TN	FP	TN	TN	TN	FP	TN	TN	TN
S	TN	TN	FP	TN	TN	FN	TP	FN	FN	FN
V	FN	FN	TP	FN	FN	TN	FP	TN	TN	FN
F	TN	TN	-	TN	TN	TN	FP	TN	TN	TN
Q	TN	TN	-	TN	TN	TN	-	TN	TN	TN

For VEB detection, the sensitivity was 86.6%, the specificity was 99.3%, the positive predictivity was 93.3%, and the overall accuracy was 98.1%. For SVEB detection, the sensitivity was 50.6%, the specificity was 98.8%, the positive predictivity was 67.9%, and the overall accuracy was 96.6%. From the results, the performance of SVEB detection is not as good as VEB detection, and the possible reasons include the more diverse types in *S* class and lack of *S* class training patterns in patients [1][77].

4.5 Performance Comparison

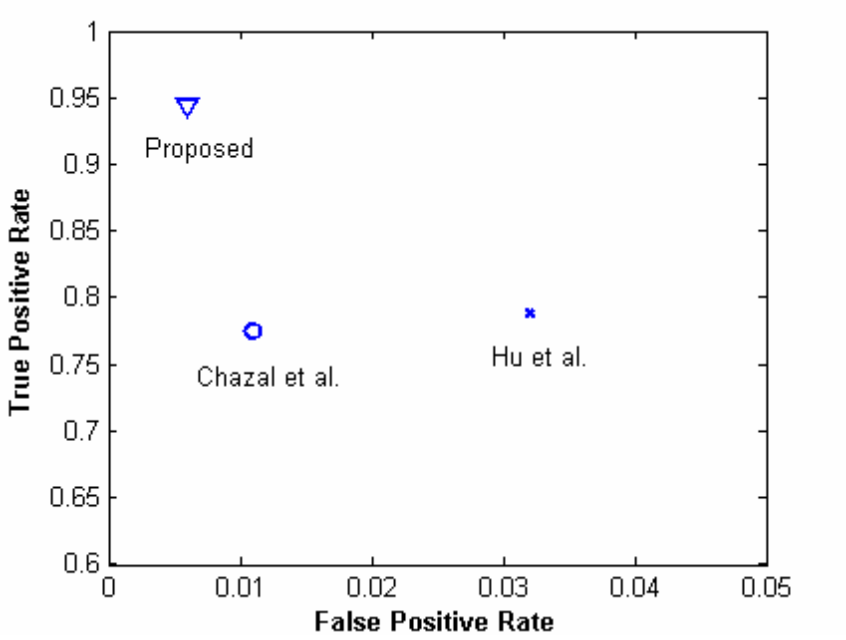
The proposed technique is compared to earlier work using the AAMI standards. An automatic heartbeat classification method [77] is based on linear discriminants and various sets of morphology and heartbeat interval features. The database was divided into two sets with each containing 22 recordings. The best classifier configuration determined using the first set was used to classify the heartbeats in the second set for performance evaluation. A neural network method based on mixture-of-experts concept [86] distinguishes VEB from non-VEB beats. The algorithm employs a test set of 20 recordings that excluded records without premature ventricular contractions. A global expert was developed using both unsupervised self-organizing map and supervised learning vector quantization based on a common set of ECG training data. A local expert was developed similarly but based solely on patient-specific training data. The decisions from both classifiers are then linearly combined using coefficients from a gating network that is trained with another set of patient-specific ECG data.

A comparison of the classification results among the three methods is given in Figure 35 and Table 6. The compared results of VEB detection were based on the 11 recordings that were common to all three studies. The compared results of SVEB detection were based on the 14 recordings that were common to both this study and [77]. Figure 35 presents the false positive rate (FPR, equivalent to one minus the specificity) versus true positive rate (TPR, equivalent to the sensitivity) from each method as a point in the receiver operating characteristics (ROC) curve [95]. The

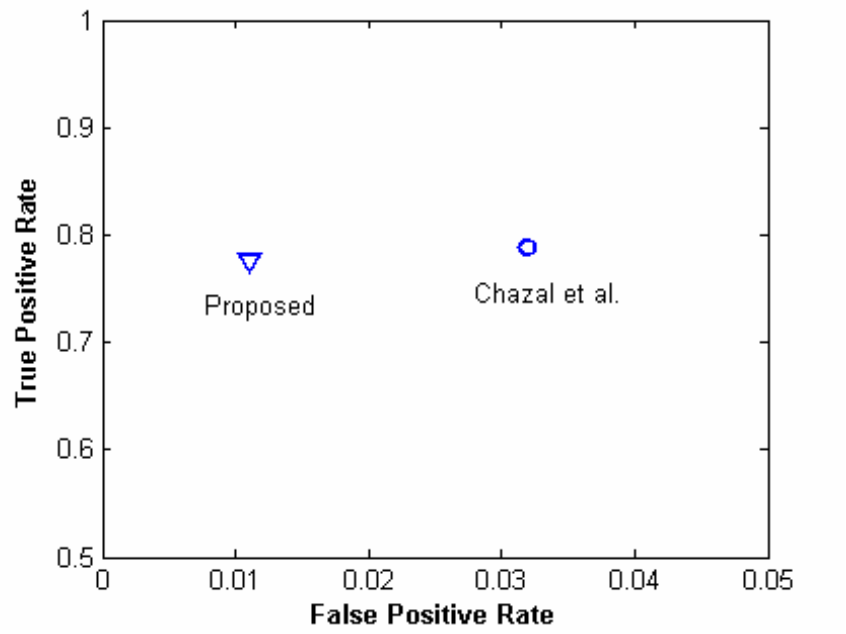
upper-left corner of the ROC curve (TPR = 1.0, FPR = 0.0) is the optimal solution. The point representing a pair of TPR and FPR that is closer to the upper-left corner corresponds to a better solution. From the plots, the proposed method generated more accurate results than the other two methods.

In Table 6, the numerical values of sensitivity, specificity, positive predictivity and overall accuracy for the three methods are presented. These results show that the proposed method outperformed the other two methods in terms of sensitivity, specificity and positive predictivity and produced notably higher overall classification accuracy for VEB detection. Comparing to the method in [77], the proposed method produced comparable sensitivity and significant better specificity, positive predictivity and overall classification accuracy for SVEB detection.

There are some other works in literature involving various classification techniques. It is interesting to compare our results with the others, although the comparisons are not exact because the other methods either use a subset of the MIT-BIH database or aims at identifying specific beat types. Hu *et al*. [84] proposed an artificial neural network method based on multi-layer perceptrons (MLP) trained with back-propagation algorithm. A two-layer MLP network of the size 51-25-2 reported an average accuracy of 90% for the classification of normal and abnormal heartbeats and 84.5% in classifying the beats into 13 beat types according to the MIT-BIH Database annotations. The use of multilayer perceptrons and Fourier transform features resulted in 2% of mean error for 3 rhythm types based on 700 test QRS complexes [3]. Osowski *et al*. [83] presented a heartbeat classification method using support vector machine (SVM) and two different types of features, Hermite characterization and high-order cumulants. The overall accuracy of heart beat recognition is 95.91% for normal and 12 abnormal types. One thing needs to be pointed out that these comparisons are not exact because the methods compared either use a subset of the MIT-BIH database or aim at identifying specific beat types.



(a)



(b)

Figure 35: Comparison of true positive rate and false positive rate for the three algorithms in terms of VEB detection (a), and SVEB detection (b).

Table 6: Performance comparison regarding VEB and SVEB detection.

Method	VEB				SVEB			
	Acc	*Sen*	*Spe*	*PP*	*Acc*	*Sen*	*Spe*	*PP*
Hu *et al*. [86]	94.8	78.9	96.8	75.8	-	-	-	-
Chazal et al. [77]	96.4	77.5	98.9	90.6	92.4	76.4	93.2	38.7
Proposed	98.8	94.3	99.4	95.8	97.5	74.9	98.8	78.8

4.6 Fault Tolerance of BbNN for ECG Classification

Fault tolerance refers to the ability of continuous operation of a system when fault occurs within the system. A system with good fault tolerance degrades its performance proportional to the degree of severity of the fault occurred, which is compared to a system without such capability that would breakdown regardless of the degree of fault.

In order to learn the fault tolerance ability of BbNNs for ECG signal classification, experiments are conducted to study the effect of noise. We want to specifically address two questions. The first question is how the classification performance is affected by noise. The second is can BbNN recover its functionality from such event.

Two types of fault modes are studied: global and local noise. In the global mode, the whole network is corrupted by Gaussian noise. Additive Gaussian noises with various levels of variations determined by signal-to-noise ratio (*SNR*) values are used in the simulation. The *SNR* is defined in the following equation:

$$SNR = 10\log_{10}\frac{\frac{1}{n}\sum_{i=1}^{n} w_i^2}{\sigma^2} \tag{60}$$

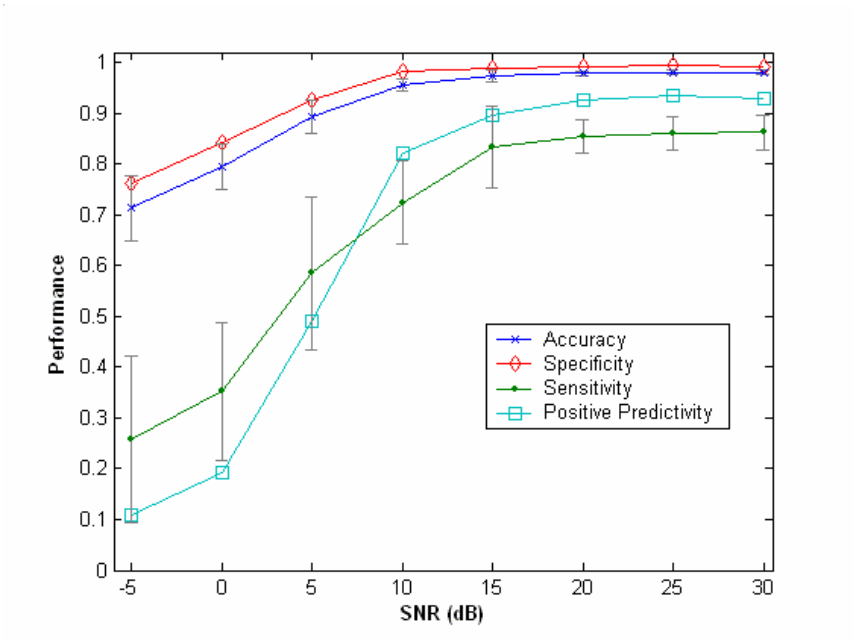
where w_i denotes the weight in a BbNN and n is the total number of weights in the BbNN. σ^2 is the variance of the Gaussian noise. In the simulation, the weights of the evolved BbNNs for 24 test records from Section 4.4 are corrupted using Gaussian noise generated with a *SNR* value. Eight levels of noise, with *SNR* values of -5, 0, 5, 10, 15, 20, 25 and 30 dB, are considered to represent noise with varying severity. The classification performance (in terms of *Acc*, *Sen*, *Spe* and *PP*) of the BbNNs with corrupted weights is recorded and shown in Figure 36. The error bars along the curves for *Acc* and *Sen* indicate the standard deviation among 10 trials. The error patterns for the other two measures are similar and skipped in the figure for visual clarity. Figure 37 presents FPR versus TPR from each noise level as a point in the ROC curve. Comparing the results shown in Figure 36 with those in no noise case reported in Section 4.4, it is observed that the noise tends to degrade the classification performance in both VEB and SVEB detection. However the degradation becomes less severe as the noise weakens. When *SNR* is higher than 15 dB, the performance degradation is negligible. Among the four measures, *Sen* and *PP* are more sensitive than *Acc* and *Spe* to the levels of the noise.

In the second fault mode, local impulse noise is simulated. Instead of the whole network, only part of the network is assumed to be corrupted by impulse noise. Specifically, a specified percentage of the total weights in the evolved BbNNs for 24 test records from Section 4.4 are randomly selected and the selected weights are then set to 0 in order to simulate the local impulse noise. Ten levels of noise severity, with percentages varying from 5% to 50% with an increment of 5%, are considered. The classification performance of the BbNNs with corrupted weights is recorded and shown in Figure 38. The error bars indicate the standard deviation among 10 trials. Figure 39 presents FPR versus TPR from each noise level as a point in the ROC curve. From the figures, it is clear that the impulse noise degrades the classification

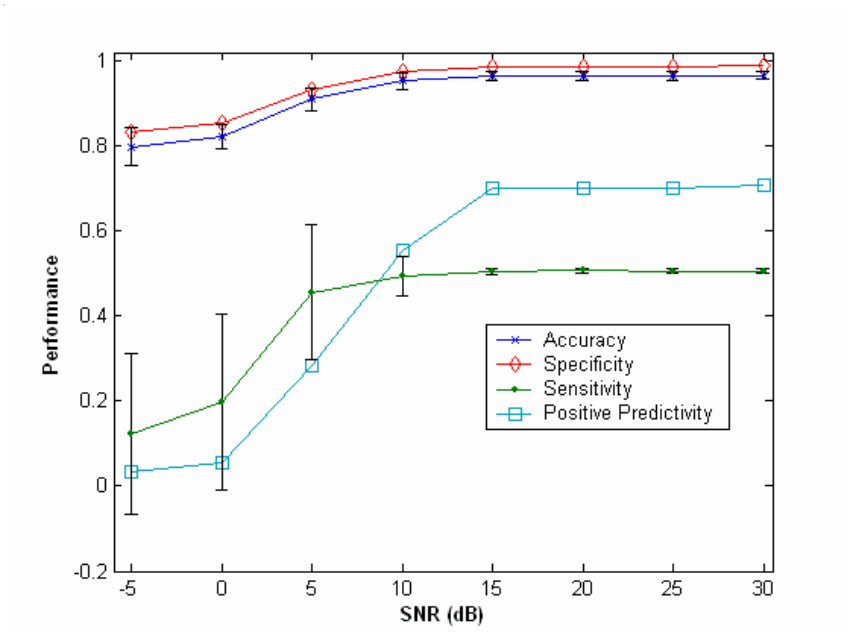
performance with degradation proportional to the severity of the noise. The higher percentage of weights is corrupted, the severer the performance degradation is.

It is interesting to compare the effects of the two types of noise modes. In both modes, the noise tends to degrade the classification performance with the degree of degradation in proportion to the noise levels (noise strength in global mode and noise width in local mode). When *SNR* is getting higher (e.g. bigger than 15 dB) in Gaussian noise case, the performance degradation becomes negligible. However, this performance degradation pattern is not observed in the impulse noise case.

In the next, experiments are conducted to study whether BbNN can recover its functionality from noise. Specifically, the effect of noise on fitness values is studied. In the simulation, initial evolution is performed on a population of BbNN individuals using the proposed evolutionary algorithm. This evolution is stopped after 3,000 generations when convergence is observed. Then low Gaussian noise with a *SNR* of 5dB is added to corrupt the weights of the BbNNs in the population. Following the noise addition, a recovery evolution is applied to the noise corrupted population to recover the BbNN functionality in terms of fitness values. Figure 40(a) shows the evolution trend of maximum and average fitness values during the initial evolution (the first 3,000 generations) and recovery evolution (beginning from the 3,001st generation) after noise corruption. The error bars indicate the standard deviation among 10 trials for every 200 generations. From the figure, both maximum and average fitness values dropped significantly after the weights are corrupted by the noise. However, the maximum fitness value is able to gradually recover from the noise and after 1,000 generations it reaches to a level that is comparable to the one achieved at the end of initial evolution. A more severe noise level with *SNR* of 0 dB is also studied. The evolution trend of maximum and average fitness values during the initial evolution and recovery evolution after noise corruption is shown Figure 40(b). It demonstrates a similar overall trend to the low noise case. However, the fitness values dropped heavier and the recovery evolution takes more generations to recover the fitness compared to the low noise case.

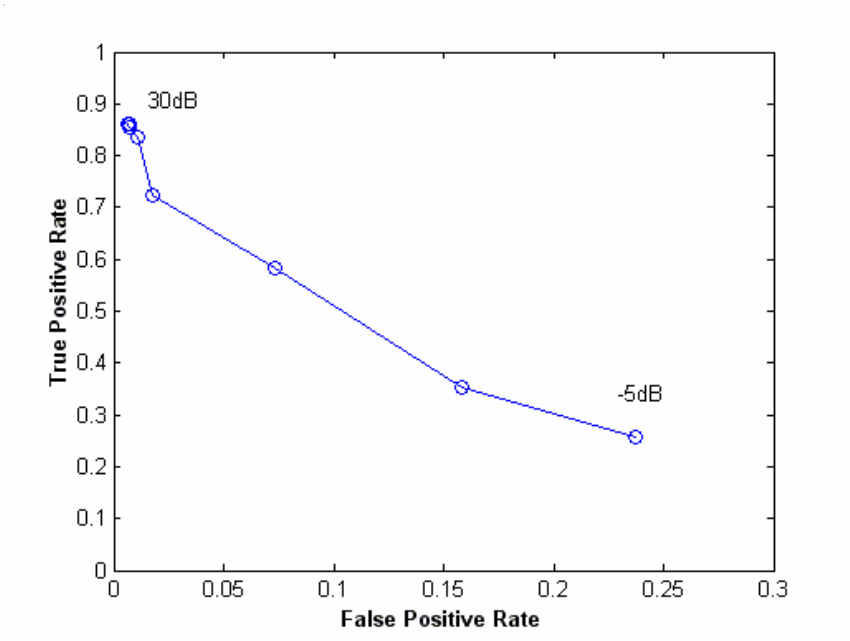


(a)

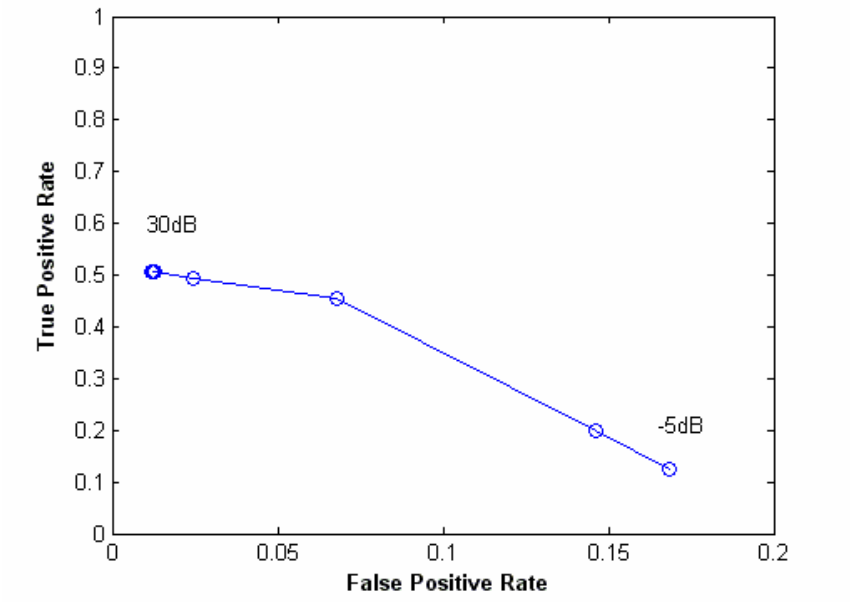


(b)

Figure 36: The effect of Gaussian noise on BbNN classification performance, (a) VEB detection, (b) SVEB detection.

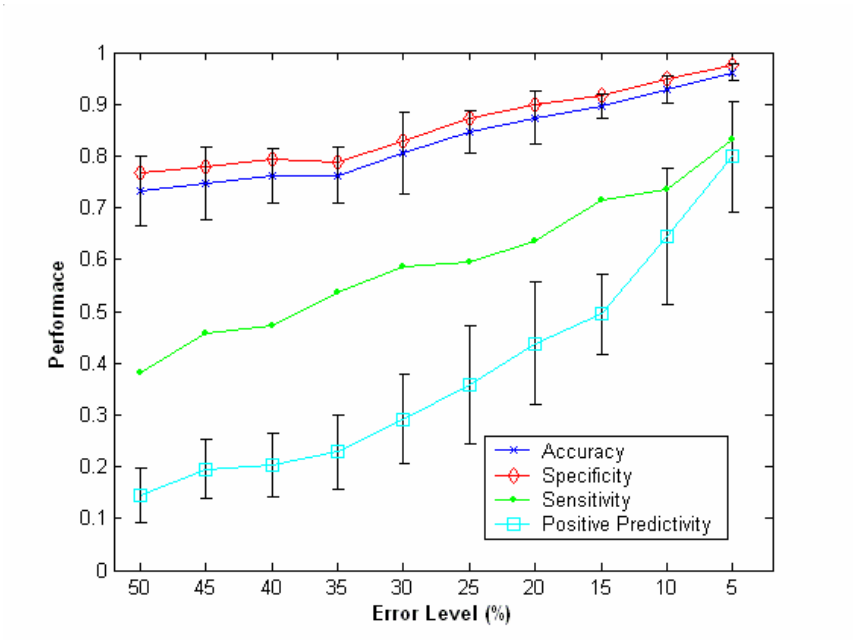


(a)

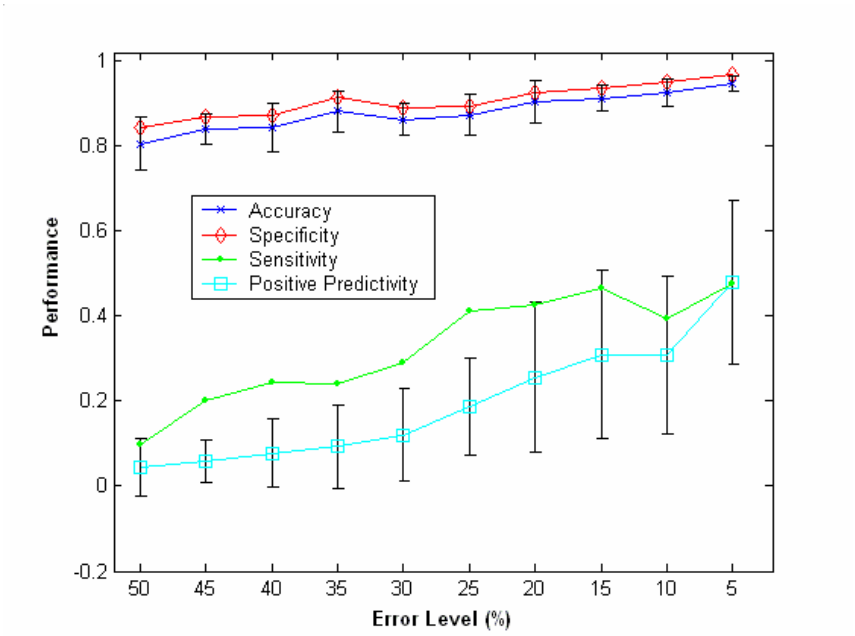


(b)

Figure 37: Comparison of true positive rate and false positive rate for different levels of Gaussian noise for VEB detection (a), and SVEB detection (b).

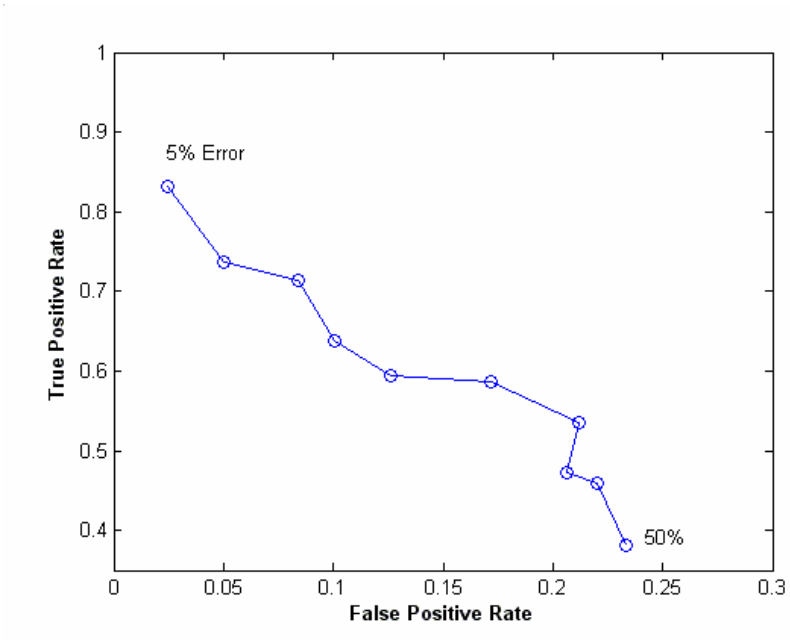


(a)

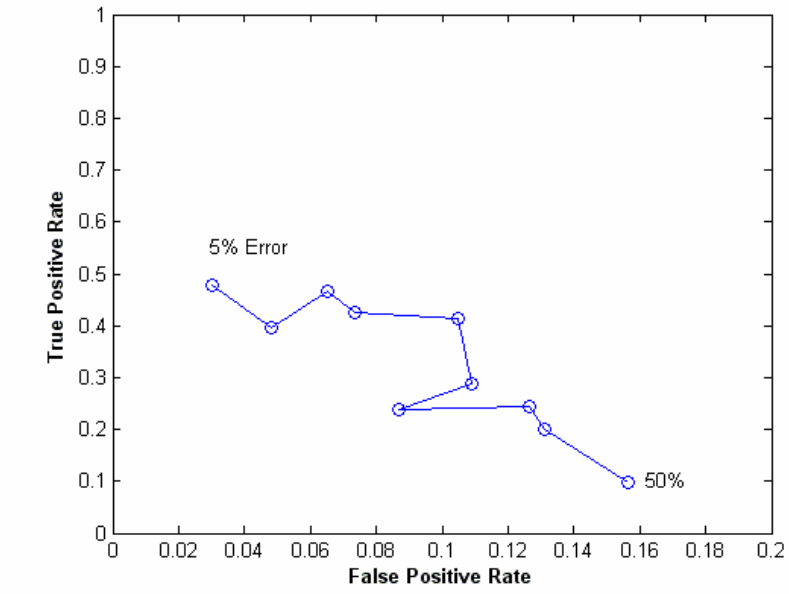


(b)

Figure 38: The effect of impulse noise on BbNN classification performance, (a) VEB detection, (b) SVEB detection.

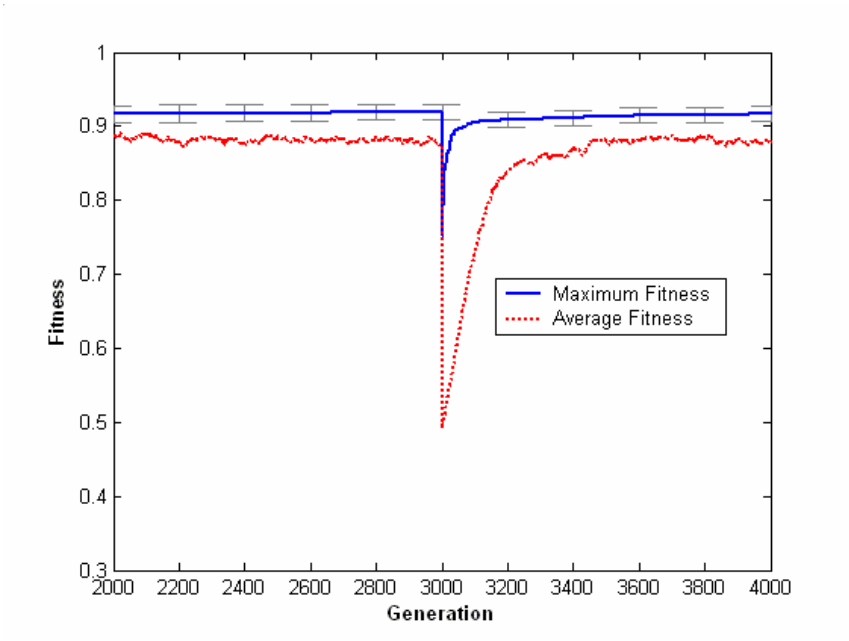


(a)

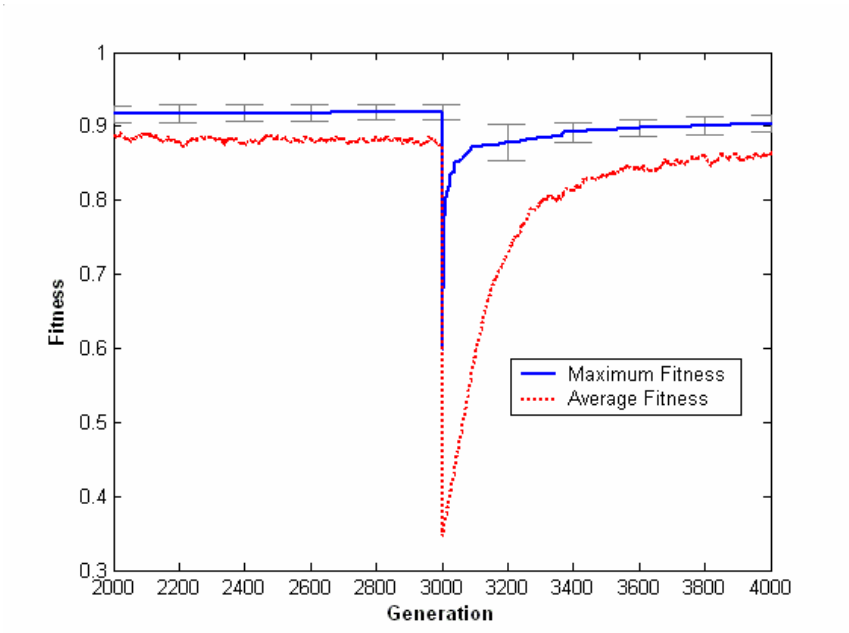


(b)

Figure 39: Comparison of true positive rate and false positive rate for different levels of impulse noise for VEB detection (a), and SVEB detection (b).



(a)



(b)

Figure 40: Evolution trend of BbNN with different levels of noise, (a) Low noise (*SNR* = 5 dB), (b) Severe noise (*SNR* = 0 dB).

Chapter 5 ACCELERATED LOCAL SEARCH USING BLOCK-WISE LEAST SQUARES LEARNING

BbNNs provide a model-free approximation approach for nonlinear dynamic systems. This chapter provides examples of dynamic system approximation using block-based neural networks. A gradient descent search was introduced in chapter 3 and used in the evolutionary algorithm as a local search operator. It is shown that the inclusion of the GDS operator in the evolutionary algorithm results in faster convergence speed and the algorithm performs well for ECG heartbeat classification. However, for applications that require highly accurate results like chaotic time series prediction, the use of GDS becomes questionable due to the slow speed associated with gradient-based search procedure. The cause of the slow speed is because that many epochs are usually needed for GDS to converge to a satisfactory solution. Moreover, a set of parameters like the learning rate need to be tuned to get optimized performance for a specific application. Observing these limitations of GDS operator, this chapter proposes a least squares learning as an alternative to the gradient descent search for dynamic system approximation [52].

5.1 Introduction

Dynamic system approximation is a research area that finds applications in fields varying from weather forecasting, chaotic time series prediction, to system identification and remote sensing. The general goal in dynamic system approximation is to construct a model that can predict the future behavior of a process based on observed past instances. In the example of time series prediction, predicted output is obtained using past and current observations. Typical inputs contain past samples of the series up to a certain length. System identification has application in many disciplines where a mathematical model is needed for modeling a physical system.

Predicated system output is generated using past system observations and current system inputs. Various modeling techniques can be identified ranging from those building a model dynamical system to black-box modeling technique using ANNs. A common drawback of systems with fixed structures is underfitting or overfitting, which is caused due to the lack of knowledge on the functional form and the order of the dynamics.

5.2 Blockwise Least Squares Learning (BLS)

The gradient-based learning methods for feedforward multilayered neural networks have major drawbacks such as slow convergence speed. Many iterations are often needed to reach an acceptable accuracy. In the other hand, linear least squares-based (LSB) approaches [25][26][27][28] that use linear least squares techniques and layer-by-layer optimization are found to have faster convergence compared to gradient-based methods. Unlike the iterative process that needs a learning rate, LSB approaches don't need user-supplied parameters. In the following, a blockwise least squares learning adopted from the LSB algorithm [25] is discussed.

The basic idea in the LSB algorithm is to construct a linear system for each layer in a MLP network and solve this system using linear least squares. A layer-by-layer optimization procedure is followed to optimize the weights in a network [25]. Considering the fact that the weights in a layer of BbNN are only sparely connected, it would not be possible to apply the least squares method for BbNN in the way as in the LSB algorithm. In the BLS algorithm, a blockwise optimization procedure is performed that the internal weights of each block are optimized by solving a set of linear equations and the blocks in the network are optimized from higher stages to lower stages. Each block in the network corresponds to a simple feedforward neural networks and its optimization is treated separately. For blocks with known desired outputs, the internal weights are optimized by minimizing the least squares criterion. For other blocks with unknown target outputs, weight optimization is completed using estimated desired outputs. Optimization of the weights of blocks in higher

stages is performed earlier than the blocks in lower stages. A training epoch consists of weight learning for all blocks from the last stage to the first stage. The training process is finished after a stop criterion is met.

5.2.1 Training a Single Block

Each block in a BbNN makes a simple feedforward neural network. A detailed view of a basic block of type 3/1 is shown in Figure 41. For a set of inputs, the equation to computing the linear outputs of the block can be written in matrix format as:

$$G = UW \tag{61}$$

where U is the input to the block with each of its columns being outputs from an input neuron and the first column being the output from a constant bias node and each row representing a data sample vector. W is the internal weights of the block. Each column of output matrix G is the linear summation applied to the nonlinearity of a node (Refer to Figure 41) to generate the block output. The dimensions of each of the

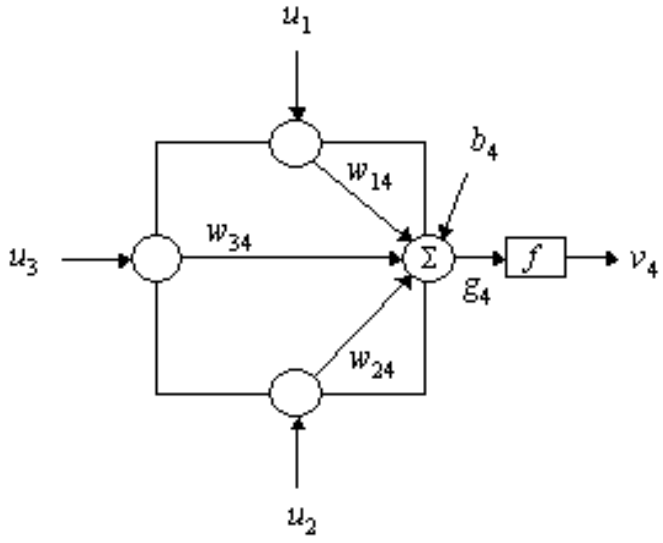


Figure 41: A detailed view of a basic block.

three basic block types are determined by the number of learning samples and number of input/output neurons of the block. Let the number of samples be N, the numbers of input and output neurons of a block be (m-1) and n, then it is clear that $\mathrm{U} \in \Re^{N \times m}$, $\mathrm{W} \in \Re^{m \times n}$ and $G \in \Re^{N \times n}$. Hence, for the 1/3 type block shown in Figure 11(a), there are $\mathrm{U} \in \Re^{N \times 2}$, $\mathrm{W} \in \Re^{2 \times 3}$ and $G \in \Re^{N \times 3}$. For the 2/2 type block, there are $\mathrm{U} \in \Re^{N \times 3}$, $\mathrm{W} \in \Re^{3 \times 2}$ and $G \in \Re^{N \times 2}$. For the 3/1 type block, there are $\mathrm{U} \in \Re^{N \times 4}$, $\mathrm{W} \in \Re^{4 \times 1}$ and $G \in \Re^{N \times 1}$.

Determining the optimal weights W can be formulated as a linear least squares problem:

$$\min \quad \left\| \mathrm{UW} - \mathrm{D} \right\|_2 \tag{62}$$

where D is the target output. For the three block types, solving the minimization problem in Eq. (62) is to find the linear square solution for an over-determined system since $N >> m$ in most cases. The least square solution for an over-determined system can be determined using QR decomposition together with Householder transformation [96]. To be specific, let us first write the original problem in the equivalent component form (associated with each output neuron) as in the following:

$$\min \quad \left\| \mathrm{U}w - d \right\|_2 \tag{63}$$

where w and d denote any column of W and D, respectively. If $\mathrm{U} \in \Re^{N \times m}$, it can be decomposed into the product of an orthogonal matrix $\mathrm{Q} \in \Re^{N \times N}$ and an upper triangular matrix $\mathrm{R} \in \Re^{N \times m}$, i.e.:

$$\mathrm{U} = \mathrm{QR} \tag{64}$$

It is reasonable to assume that U has full column rank since $N >> m$. Therefore the economy-sized QR decomposition is given as follows

$$\mathrm{U} = \mathrm{Q}_n \mathrm{R}_n \tag{65}$$

where Q_n consists of the first n columns of Q and R_n consists of the first n rows of R. The least square solution w_{ls} to Eq. (63) can be computed through back-substitution from the following equation:

$$R_n w_{ls} = Q_n^T d \tag{66}$$

After the optimal weights are determined, a set of desired input S is sought for in order to reduce further the least squares error. With the weights W and target outputs D known, the desired input S is the least square solution to the following problem:

$$\min \quad \|SW - D\|_2 \tag{67}$$

where W is the optimal weights determined from Eq. (62). Since the output from a bias node is constant, the entries in the first column of the desired input S are fixed. Therefore, the original minimization problem needs to be modified to take into account this constraint. The modified minimization problem is given in the following:

$$\min \quad \|S_c W_c - D\|_2 \tag{68}$$

where W_c is same to W excluding its first row and S_c equals S except its first column of constant bias output is discarded. Now let us translate this problem into standard least squares formulation and write down its component form that corresponds to each training sample:

$$\min \quad \left\|\left(s_c W_c - d_c\right)^T\right\|_2 \tag{69}$$

where s_c denotes any row of S_c and d_c denotes any row of D, respectively. The problem given in Eq. (69) can be a square, an over-determined or an under-determined system depending on the type of the block. When the block is 2/2 type, the system is square. It is an over-determined system when the block is 1/3. For type 3/1 block, the system is under-determined. The method used to solve Eq. (63) can be utilized to find the least square solution for square or over-determined system.

For a general under-determined system, it has either no solution or infinite solutions. In our base, there are infinite solutions as the left-null space of the system is empty and the null space is not. Therefore, the minimal 2-norm solution is found through the following procedure [96]. For clarity, let us rewrite the Eq. (69) as in the following

$$\min \quad \left\| \mathrm{W}_c^{\mathrm{T}} s_c^{\mathrm{T}} - d_c^{\mathrm{T}} \right\|_2 \tag{70}$$

Let compute QR decomposition of $\mathrm{W}_c \in \Re^{n\times(m-1)}$ and get

$$\mathrm{W}_c = \mathrm{Q}\begin{bmatrix} \mathrm{R} \\ 0 \end{bmatrix} \tag{71}$$

where $\mathrm{R} \in \Re^{n\times n}$. Then Eq. (70) becomes

$$\begin{bmatrix} \mathrm{R}^{\mathrm{T}} & 0 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = d_c^{\mathrm{T}} \tag{72}$$

where

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \mathrm{Q}^{\mathrm{T}} s_c^{\mathrm{T}} \tag{73}$$

with $z_1 \in \Re^n$ $z_2 \in \Re^{m-n-1}$. Let $z_2 = 0$, the minimum 2-norm solution follows

$$s_c^{\mathrm{T}} = \mathrm{Q}\begin{bmatrix} z_1 \\ 0 \end{bmatrix} \tag{74}$$

where z_1 is solved from Eq. (72).

The thus acquired desired input becomes the target output for the blocks connected to current block. However, due to the use of nonlinear sigmoidal function, the output from output node is bounded. The acquired input S has to be transformed

to bring its range into that of the activation function. To that purpose, a transformation matrix is used [25]. After the optimal weights W and desired input S are determined, the learning process for the block is completed.

5.2.2 Training a Block-based Neural Network

The computation stage *s* associated with a block denotes the priority according to which each block is trained. The blocks in higher stages are trained earlier than those in lower stages. The blocks within the same stages are trained with the same priority.

The BLS algorithm for BbNN can be summarized in the following:

1) Generate randomly initial internal weights for each block in the network.

2) Propagate all patterns through the network from blocks in lower computation stages to blocks in higher stages producing outputs.

3) Update the weights for the block in stage *s* using Eq. (62).

4) Update the input for the block in stage *s* (the desired output for the connected block in stage *s*-1) using Eq. (67).

5) Repeat steps 3) - 4) for each block in stage *s*-1.

6) If end condition is met, stop learning; otherwise, go to step 2).

5.2.3 Computation Complexity

The number of multiplications required to solve a linear least squares problem using QR decomposition and Householder transformation equals $M \times n \times (m+n)$, in which M and n are the dimensions of input matrix and n and m are dimensions of weight matrix. The computation complexity of optimizing the block shown in Figure 41 will be $O(20M)$. As a comparison, the GDS optimization of the same block type will have a complexity of $O(4M)$. Thus, the operation complexity of both algorithms is only linearly correlated to the number of examples. The BLS algorithm takes more operations than the GDS algorithm per epoch. However, the experiment results presented in the following section show that BLS is much faster than GDS since BLS

takes only a few epochs compared to hundreds of epochs of GDS to reach comparable or lower error level.

5.3 Experimental Results

This section presents experimental results for two dynamic system approximation problems: one is the well-known Mackey-Glass time series prediction and the other is a realistic nonlinear system identification problem. GDS and BLS are used as a stand-alone optimization procedure for a fixed structure BbNN, and their performance in terms of convergence speed is compared. Then the EA only algorithm and the EA with local search algorithms, namely, evolutionary operators plus GDS (referred as EA+GDS) and evolutionary operators plus BLS (referred as EA+BLS), are also compared. All the algorithms are implemented using Visual C++ 6.0 and run under a PC platform with Pentium 4 2.80 GHz CPU. The fitness function is defined as:

$$\text{Fitness} = \frac{1}{1 + \sum_{i=1}^{M} \left\| \boldsymbol{d}^i - \boldsymbol{y}^i \right\|^2} \tag{75}$$

where M denotes the number of training samples. $\boldsymbol{d}^i$ and $\boldsymbol{y}^i$ are the desired and actual outputs when the ith pattern is presented. The parameters used for the EA algorithms are listed in Table 7.

5.3.1 Time Series Prediction

The time series prediction is to estimate future behavior of a process based on observations up to current time, which can be modeled using the following equation:

$$\hat{x}(t+I) = f\left(x(t), x(t-D), ..., x(t-qD)\right) \tag{76}$$

Table 7: Parameters of the evolutionary algorithm for dynamic system approximation

Parameter	Value
Population	80
Maximum Generations	1, 000
Maximum Fitness Value	0.95
GDS Epoch	50
GDS Learning Rate	0.001
Disruptive Pressure	0.6
Tournament Size	2
Rate Update Interval	12
Initial GDS Rate	1.0
Minimum Operator Rate	0.1

where t denotes current time index and positive integer q is called the order of the model. The function $f(\cdot)$ represents the functional input-output relationship of a time series prediction process.

The Mackey-Glass (M-G) time series is a chaotic time series simulating blood flow [97] and it is one of the widely investigated benchmark examples in time series prediction. The M-G time series can be represented using the following differential equation:

$$\dot{x}(t) = -ax(t) + \frac{bx(t-\tau)}{1+x^{10}(t-\tau)} \tag{77}$$

The system can exhibit fixed points, limit cycles, or chaotic behaviors for different values of τ. The M-G time series values at integer points were obtained by applying the fourth-order Runge-Kutta method to find solution to Eq. (77). The

parameters used are $a = 0.1$, $b = 0.2$ and $\tau = 17$. For this particular value of τ, the system exhibits a chaotic behavior. Figure 42 shows an M-G time series generated for experiments where the dotted vertical line marks the beginning of test phase. Among the total 500 data points, 300 of them are used to train a selected BbNN with fixed structure and the remaining 200 points serve as test data.

The time series data in lagged space $x(t)$, $x(t\text{-}1)$, $x(t\text{-}2)$ and $x(t\text{-}3)$ are inputs to the BbNN and $x(t+1)$ is the output from the network, i.e., $I = q = 1$ and $D = 3$. Starting from a randomly generated initial set of weights, both GDS and BLS algorithms are applied to optimize the internal weights of a BbNN with two rows and four columns. The learning rate selected for GDS is 0.05 that allows faster convergence based on some initial trials. There is no parameter to set for BLS algorithm.

The performance of the BLS algorithm regarding convergence speed and prediction accuracy is compared to that of GDS. Table 8 lists the numerical comparison between the two methods. The BLS usually takes only one or a few epochs to reach the error level that the GDS algorithm does not achieve after 1,000 epochs. The actual CPU running time of BLS is also significantly less than that of GDS algorithm. The Mean Squared Error (MSE) as the error criterion is also compared between the two methods for both training and test data. The BbNN trained with BLS algorithm achieves the error level that is nearly 10 times less than that achieved with GDS after 1000 epochs. The BbNN trained with BLS is found to generalize well to the test data that is not seen before. The errors for training and test data are comparable.

Next, the performance of the EA only and EA with local search operator algorithms is compared. The time series data in lagged space $x(t)$, $x(t\text{-}1)$, $x(t\text{-}2)$ and $x(t\text{-}3)$ are inputs to the BbNN and $x(t+1)$ is the output from the network, i.e., $I = q = 1$ and $D = 3$. An initial population is randomly generated. The three algorithms are applied to find optimal BbNN structure and weights. Figure 43 compares the maximum fitness values after 1,000 generations for the three algorithms averaged among 10 trials. The use of either GDS or BLS operator in the EA algorithm results

in higher fitness value than EA only case. The Mean Squared Error (MSE) corresponding to the maximum fitness value is computed and listed in Table 9. The EA+GDS and EA+BLS algorithms produce lower training and test error than EA only algorithm. Between the two EA algorithms that use a local search operator, EA+BLS algorithm performs noticeably better than EA+GDS.

The convergence speed of the EA algorithms is compared in terms of achieving the same level of fitness value (The level compared is the average fitness value that the EA+GDS achieves after evolution). Table 10 shows the number of generations and CPU running time. From the table, the EA only method failed to achieve this level of fitness. Between the two EA methods using local search, EA+BLS algorithm takes much less time than EA+GDS.

Figure 44 plots an evolved BbNN using the EA+BLS algorithm for Mackey-Glass time series prediction. An output node y gives a future prediction based on inputs of past observations. All other redundant output nodes are marked by *. Inputs $x_1,\ldots, x_4$ to the BbNN are the time series data in lagged space $x(t)$, $x(t\text{-}1)$, $x(t\text{-}2)$ and $x(t\text{-}3)$. Figure 45 shows the typical 1-step prediction results from the evolved BbNN for the test data. The estimates from EA+BLS resemble the most to the truth data among the three algorithms compared.

5.3.2 Nonlinear System Identification

Conventional techniques for nonlinear system identification utilizing mathematical models require the structure of the model must be known in advance. Block-based neural networks provide a general model-free approach for identifying nonlinear systems. The system in interest is a practical liquid-saturated steam heat exchanger [98], where water is heated by pressurized saturated steam through a copper tube. The input variables are the liquid flow rate, the steam temperature, and the inlet liquid temperature. The system output is the outlet liquid temperature. In this experiment, the steam temperature and the inlet liquid temperature are kept constant to their nominal values. The system model can be described as in Eq. (78):

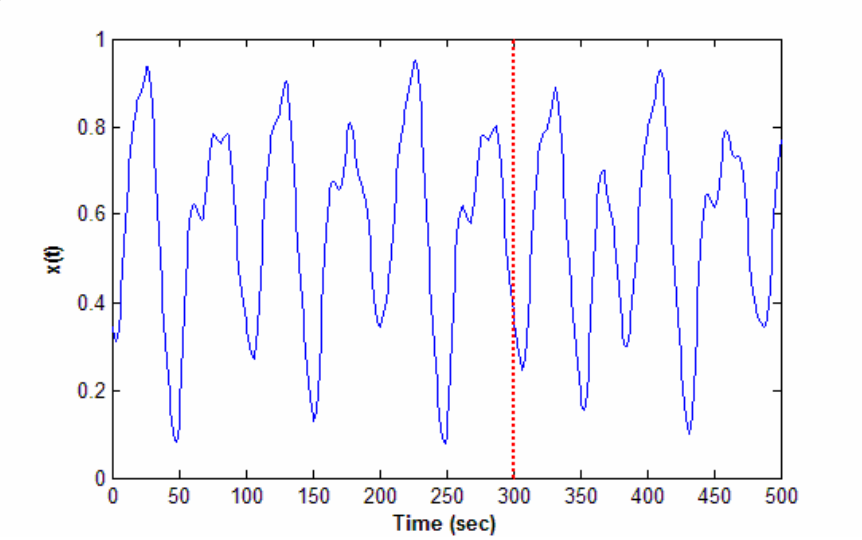


Figure 42: A Mackey-Glass time series.

Table 8: Performance comparison for M-G time series prediction

Method	Epoch	Time (s)	MSE (Train/Test)
GDS	1,000	15	$4.22/4.71\times10^{-3}$
BLS	1	0	$4.69/4.39\times10^{-4}$

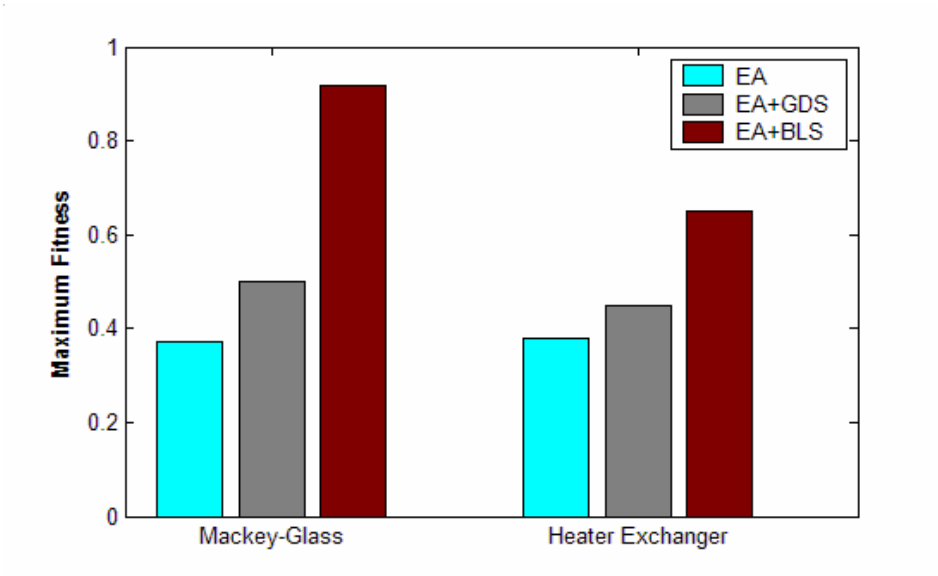


Figure 43: Comparison of achieved maximum fitness among the EA algorithms.

Table 9: Comparison of mean squared error for the EA algorithms

Method	Mackey-Glass (Training/Test)	Heater Exchanger (Training/Test)
EA	$6.02/5.99\times10^{-3}$	$5.44/5.65\times10^{-3}$
EA+GDS	$3.39/3.46\times10^{-3}$	$4.12/4.20\times10^{-3}$
EA+BLS	$3.00/2.95\times10^{-4}$	$1.79/1.58\times10^{-3}$

Table 10: Comparison of convergence speed for the EA algorithms

Method	Mackey-Glass (Generation/Time)	Heater Exchanger (Generation/Time)
EA	Failed	Failed
EA+GDS	1000/172s	1000/215s
EA+BLS	11/1s	9/1s

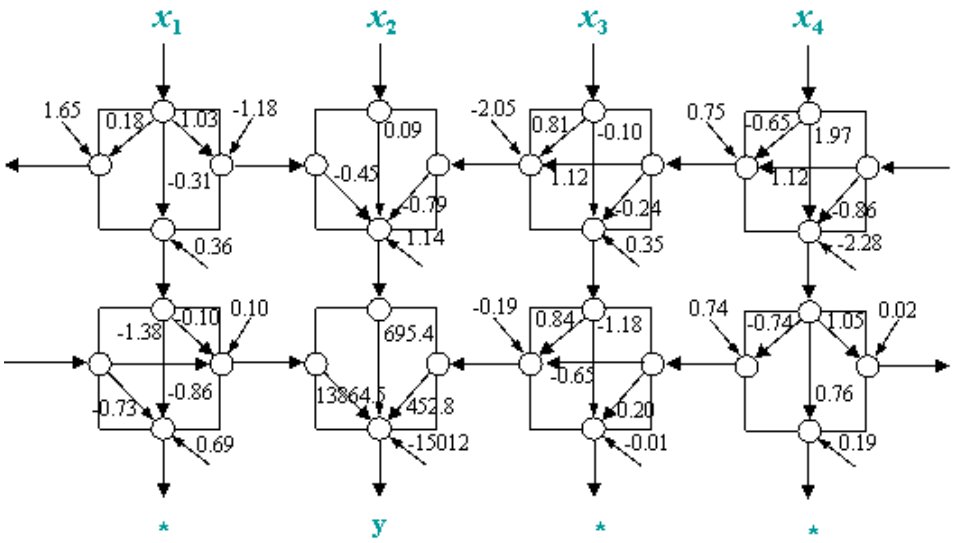


Figure 44: Evolved BbNN for M-G time series prediction after 1000 generations.

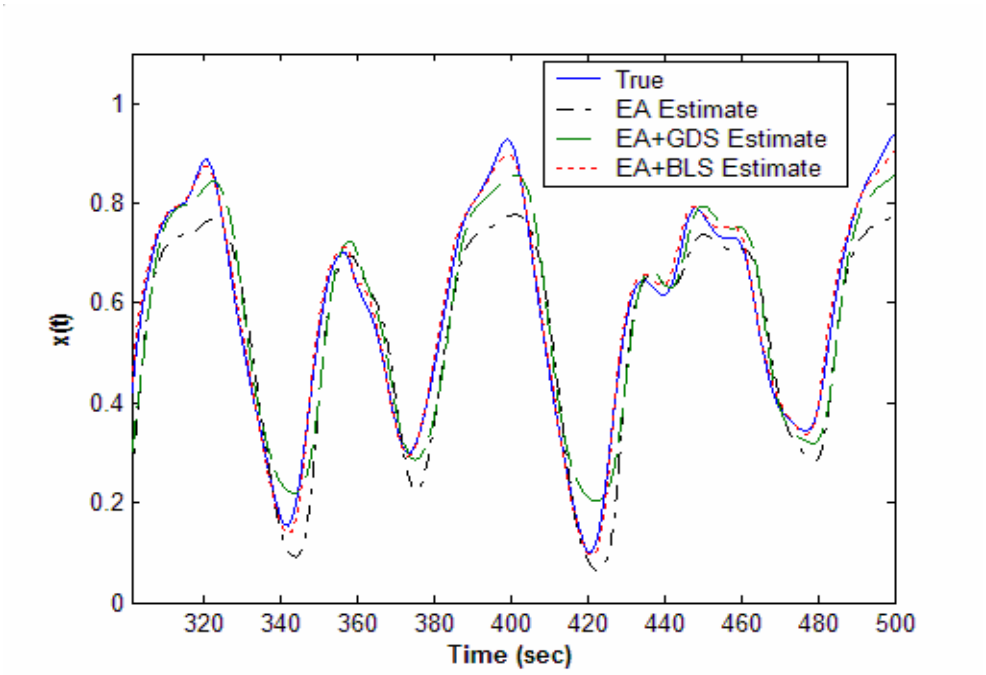


Figure 45: Test results of M-G time series prediction.

$$\hat{y}(t) = f\left(y(t-1), y(t-2), ..., y(t-D_1); u(t), u(t-1), ..., u(t-D_2)\right) \quad (78)$$

in which u and y denote the system input and output, respectively.

The set of data employed in training the neural network has a large impact on the quality of the identified system model, which means the set of training data needs to include as much information as possible about the dynamics of the system. It is therefore important to construct a balanced set of training data that covers the whole system operation range. To this end uniformly distributed input over the process range are generated and serve as system input. Figure 46 shows the corresponding fluid outlet temperature in which dotted vertical line separates the training and test data.

The system output data $y(t\text{-}1)$, $y(t\text{-}2)$ and $y(t\text{-}3)$ and lagged input $x(t\text{-}1)$, and $x(t)$ are inputs to the BbNN and $y(t)$ is the output from the network, i.e., $D_1 = 1$ and $D_2 = 3$. Starting from a randomly generated initial set of weights, GDS and BLS algorithms are applied to optimize the internal weights of a BbNN with two rows and five columns. The first 300 input-output pairs in the data set are training data and the remaining data serve as independent test samples.

The performance of the BLS algorithm regarding convergence speed is compared to that of GDS. The learning rate selected for the GDS is 0.05 that allows good convergence performance. Table 11 lists the comparison of numerical results between the two methods. The BLS algorithm takes only one or a few epochs to reach an error level that the GDS algorithm does not achieve after 1,000 epochs. The actual CPU running time of BLS is also significantly less than that of GDS algorithm. Similar to the case of M-G time series prediction, the BbNN trained with BLS is found to generalize well to the test data that is not seen before. The errors for training and test data are comparable.

Next, the performance of the EA only and EA plus local search algorihtms is compared. The system output data $y(t\text{-}1)$, $y(t\text{-}2)$ and $y(t\text{-}3)$ and lagged input $x(t\text{-}1)$, and $x(t)$ are inputs to the BbNN and $y(t)$ is the output from the network, i.e., $D_1 = 1$ and $D_2 = 3$. The three algorithms are applied to find optimal BbNN structure and weights

starting from randomly generated populations repeated for 10 times. Figure 43 compares the maximum fitness values after 1,000 generations for the three algorithms averaged among the 10 trials. The use of either GDS or BLS operator in the EA algorithm results in higher fitness values than EA only case. The EA algorithms with local search ability produce lower training and test error than EA only algorithm according to Table 8. Between the two algorithms using both evolutionary and local search operators, EA+BLS algorithm performs noticeably better than EA+GDS. When comparing the convergence speed among the three algorithms, EA only method failed to achieve the same level of fitness value that is achieved by the two algorithms with local search according to Table 10. When comparing the two methods that use both types of operators, EA+BLS algorithm takes much less time than EA+GDS.

Figure 47 plots an evolved BbNN using the EA+BLS algorithm for heater exchanger system identification. An output node y gives an estimated fluid outlet temperature based on inputs and outputs in lagged space. All other redundant output nodes are marked by *. Inputs x_1 and x_2 to the BbNN are the time series data in lagged space $x(t)$, $x(t\text{-}1)$, and x_3, x_4 and x_5 correspond to $y(t\text{-}1)$, $y(t\text{-}2)$ and $y(t\text{-}3)$. Figure 48 shows the typical output estimates from the evolved BbNN for the test data. The estimate from EA+BLS produces the least amount of error among the three methods compared.

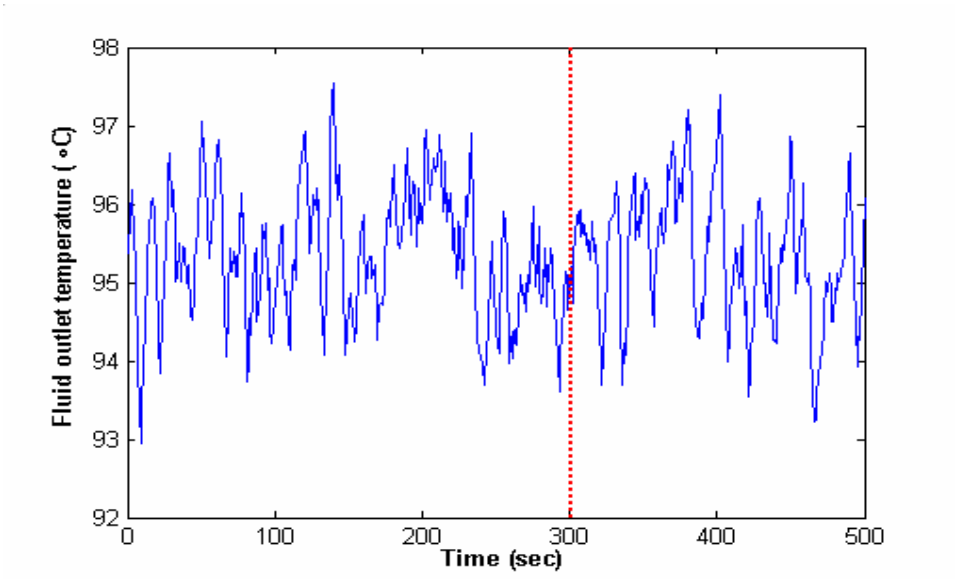


Figure 46: Fluid outlet temperature of a practical heat exchanger.

Table 11: Performance comparison for nonlinear heat exchanger identification

Method	Epoch	Time (s)	MSE(Train/Test)
GDS	1,000	20	$4.95/5.07\times10^{-3}$
BLS	1	0	$2.44/2.12\times10^{-3}$

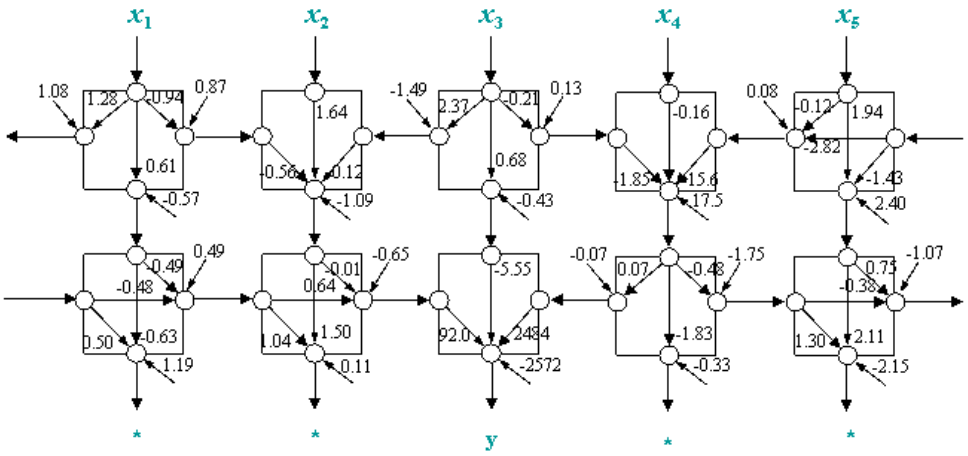


Figure 47: Evolved BbNN for heater exchanger system identification after 1000 generations.

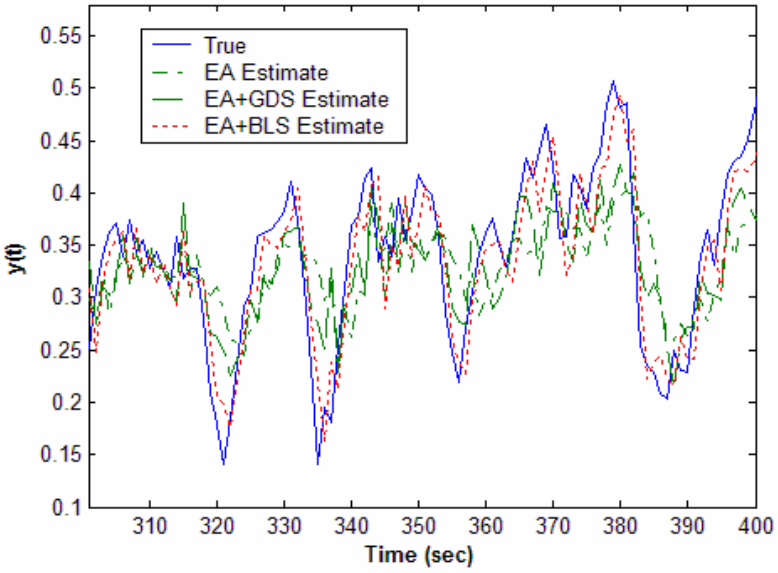


Figure 48: Outlet temperature of the simulated process and BbNN prediction.

Chapter 6 CONCLUSIONS

6.1 Conclusions

This dissertation presents personalized health monitoring using evolvable block-based neural networks. As a specific example, personalized ECG heartbeat classification is demonstrated using the BbNN approach. In the following, conclusions of this dissertation are drawn.

A computationally efficient evolutionary algorithm that simultaneously optimizes the structure and weights of block-based neural networks is developed. In addition to the evolutionary operators of crossover and mutations, this algorithm utilizes local search operators that are based on gradient descent principle and linear least squares method. The use of local search operator greatly increases the optimization speed of the evolutionary algorithm. In order to remove manual tuning of operator rates, an adaptive rate update scheme that rewards or penalizes an operator based on its past performance is proposed. A fitness scaling with generalized disruptive pressure that favors individuals at two extreme ends reduces the possibility of premature convergence. The use of both adaptive rate update and fitness scaling ensures higher fitness values.

The BbNN platform provides a viable approach for personalized ECG heartbeat classification. Evolvable classifiers based on block-based neural networks can change the structure and configurations as well as internal parameters to cope with the heartbeat variations due to personal or temporal differences. A BbNN evolved with the proposed evolutionary algorithm using the Hermite transform coefficients and a time interval between two neighboring R peaks of ECG signal, provides a patient-specific heartbeat classification system. Experimental results using the MIT-BIH Arrhythmia database demonstrate a high accuracy of 98.1% and 96.6% on average for the detection of ventricular ectopic beats (VEBs) and supraventricular ectopic beats

(SVEBs), respectively, a significant performance improvement over other major techniques. Also, experimental study on fault tolerance of BbNNs demonstrates that the level of performance degradation is proportional to the severity of noise for ECG signal classification.

The BbNN approach method provides a general model-free technique for dynamic system approximation. A blockwise least squares learning method (BLS) is proposed as an alternative to the gradient descent search for applications where highly accurate results are desired. Experimental results based on Mackey-Glass time series prediction and nonlinear system identification reveal that BLS converges faster with orders of magnitude compared to the gradient-based procedure. The use of local search operator in the evolutionary algorithm produces higher fitness values that lead to smaller prediction errors.

6.2 Future Directions

6.2.1 Issues on Fault Tolerance

Fault tolerant systems are desirable in many applications. For example, in deep space exploration where physical space is often very limited, a system capable of fault recovery is of great value compared to the typical sparse approach. We studied fault tolerance of BbNNs for ECG classification. Preliminary experiments demonstrate some fault tolerance ability of BbNNs. It was shown that the degree of performance degradation due to noisy weight connections is proportional to the level of severity of the corruption noise. Also, the functionality of BbNNs can be gradually restored through the use of recovery evolution. In the author's opinion, research on fault tolerance of BbNNs can be extended to include discussions on fault recovery in more hardware-oriented environments. For that purpose, two specific issues need to be addressed.

The first issue is efficient fault recovery. The performance of reconfigurable hardware can be degraded by faults. Various fault sources exist such as radiation, thermal fatigue, oxide breakdown and electromigration [99]. The resulting faults include stuck-at faults, shorts and opens, and interconnect delay faults [101]. A number of methods have been proposed for fault recovery [99][100] in reconfigurable hardware. However, few efforts have been devoted to fault recovery at neural network level (i.e. designing ANNs using reconfigurable hardware). Further investigations can be conducted on efficient methods for recovering functionalities of BbNNs in the event of such faults.

Another issue is practicality. The combination of evolvable hardware and evolutionary algorithm provides reconfiguration that can be utilized for fault recovery. While restoring functionality is essential to fault recovery, time constraint should also be considered to make fault recovery practical [102]. Hardware reconfiguration can be a very time consuming process. As reviewed in previous section 2.3.2, one generation of the intrinsic evolution in [60] took 4.8 hours and a successful evolution would take months if hundreds of generations are needed. For most online applications, this amount of time is not practical. The time constraint on reconfiguration depends on the specific application and it varies from application to application due to different recovery deadlines [102]. Therefore it is important to consider the time constraint for a specific application in fault recovery when designing BbNNs using reconfigurable hardware.

6.2.2 Lazy Learning Methods for ECG Signal Classification

Lazy learning methods are a class of statistical regression models that store training instances in memory and answer a new query by resorting to the relevant instances stored. In its simplest form, lazy learning predicts the output of a query by finding a set of nearest neighbors and voting on the outputs of those neighbors. Another form of typical lazy learning methods, known as locally weighted learning, uses locally weighing strategy to combine outputs of relevant training samples determined

through a distance measure. Compared to supervised learning methods such as neural networks, lazy learning methods avoid the procedure for training model parameters and offers higher flexibility in fitting local features of target surface. Lazy learning methods have been successfully used in many application domains including robot control, modeling time series, reinforcement learning and others. A latest survey on locally weighted learning is found in [103].

Among the vast literature on ECG signal classification, most approaches adopt a "global" strategy [3][77][80][81][83]-[90] in the means that the parameters associated with a classifier are optimized by minimizing an error metric between the target and actual outputs for a set of labeled training patterns. In the later retrieval operation, the trained "global" classifier is used to classify unseen heartbeat patterns. Unlike such "global" strategies, in a lazy learning method, the output of a query is determined by combining the outputs of neighboring points with known labels. There are a few works that tackle ECG signal classification using lazy learning methods. A simple nearest neighbor approach in [84] using Euclidean distance for determining relevance reported a smaller than 75% classification accuracy for detecting abnormal ECG heartbeat patterns based on a limited set of data set containing 6474 samples. However, such moderate performance can expect to be greatly enhanced by using locally weighted learning and tuning parameters for ECG classification. An interesting future research topic would be to investigate whether and how lazy learning methods could contribute to improving the performance of ECG signal classification.

BIBLIOGRAPHY

Bibliography

[1] American Heart Association, *Heart Disease and Stroke Statistics - 2006 Update*, American Heart Association, Dallas, Texas, 2006.

[2] R. Mark and G. Moody, *MIT-BIH Arrhythmia Database Directory*, (http://ecg.mit.edu/dbinfo.html).

[3] K. Minami, H. Nakajima, and T. Toyoshima, "Real-Time Discrimination of Ventricular Tachyarrhythmia with Fourier-Transform Neural Network," *IEEE Trans. on Biomedical Engineering*, Vol. 46, No. 2, pp. 179-185, Feb. 1999.

[4] S. W. Moon and S. G. Kong, "Block-based Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 12, No. 2, pp. 307-317, 2001.

[5] W. Jiang, S. G. Kong, and G. D. Peterson "ECG Signal Classification with Evolvable Block-based Neural Networks," *Proc. Int'l Joint Conf. on Neural Networks (IJCNN-2005)*, Vol. 1, pp. 326-331, July 2005.

[6] W. Jiang and S. G. Kong, "Evolvable Block-based Neural Networks for Heart Monitoring," *Proc. USA-Korea Conference on Science and Engineering*, Irvine, CA, Aug. 2005.

[7] W. Jiang, S. G. Kong, and G. D. Peterson, "Continuous Heartbeat Monitoring Using Evolvable Block-based Neural Networks," *Proc. Int'l Joint Conf. on Neural Networks (IJCNN-2006)*, Vancouver, Canada, July 2006.

[8] W. Jiang and S. G. Kong, "Block-based Neural Networks for Personalized ECG Heartbeat Classification," in press, *IEEE Trans. on Neural Networks*, Nov. 2007.

[9] Association for the Advancement of Medical Instrumentation, *Recommended Practice for Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms*, 1987.

[10] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biology*, Vol. 5, No. 4, pp. 115-133, 1943.

[11] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, John Wiley & Sons, New York, NY, 1949.

[12] B. Widrow and M. E. Hoff, "Adaptive Switching Circuits", *IRE Western Electric Show and Convention Record 1960*, Part 4, pp. 96-104, Aug. 1960.

[13] F. Rosenblatt, "The Perceptron: A Probalistic Model for Information Storage and Organization in the Brain," *Psychological Review*, Vol. 65, No. 6, pp. 386-408, 1958.

[14] M. L. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, 1969.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Back-propagating Errors," *Nature*, Vol. 323, No. 6088, pp. 533-536, Oct. 1986.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Eds, *Parallel Distributed Processing*, Vol. 1, Ch. 8, pp. 318-362, MIT Press, Cambridge, MA, 1986.

[17] T. Kohonen, "Self-organized Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, Vol. 43, pp. 59-69, 1982.

[18] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proc. of the National Academy of Science*, Vol. 79, pp. 2554-2558, 1982.

[19] D. P. Bertsekas, *Nonlinear Programming*, 2nd edition, Athena Scientific, Belmont, MA, 1999.

[20] L. O. Chua and L. Yang, "Cellular Neural Networks: Theory," *IEEE Trans. on Circuits and Systems*, Vol. 35, No. 10, pp. 1257-1272, Oct. 1988.

[21] L. O. Chua and L. Yang, "Cellular Neural Networks: Applications," *IEEE Trans. on Circuits and Systems*, Vol. 35, No. 10, pp. 1273-1290, Oct. 1988.

[22] http://www.isi.ee.ethz.ch/~haenggi/CNN_web/architecture.html.

[23] L. O. Chua and T. Roska, *Cellular Neural Networks and Visual Computing: Foundations and Applications*, Cambridge University Press, UK, 2002.

[24] R. S. Scalero and N. Tepedelenlioglu, "A Fast Algorithm for Training Feedforward Neural Networks," *IEEE Trans. on Signal Processing*, Vol. 40, No. 1, pp. 202-210, 1992.

[25] F. B. König and F. Bärmann, "A Learning Algorithm for Multilayered Neural Networks Based on Linear Least Squares Problems," *Neural Networks*, Vol. 6, pp. 127-131, 1993.

[26] J. Y. F. Yam and T. W. S. Chow, "Accelerated Training Algorithm for Feedforward Neural Networks Based on Least Squares Method," *Neural Processing Letters*, Vol. 2, No. 4, pp. 20-25, 1995.

[27] S. Ergezinger and E. Thomsen, "An Accelerated Learning Algorithm for Multilayer Perceptrons: Optimization Layer by Layer," *IEEE Trans. on Neural Networks*, Vol. 6, No. 1, pp. 31-42, 1995.

[28] G. Wang and C. Chen, "A Fast Multilayer Neural-Network Training Algorithm Based on the Layer-By-Layer Optimizing Procedures," *IEEE Trans. on Neural Networks*, Vol. 7, No. 3, pp. 768-775, 1996.

[29] J. Y. F. Yam and T. W. S. Chow, "Extended Least Squares Based Algorithm for Training Feedforward Networks," *IEEE Trans. on Neural Networks*, Vol. 8, No. 3, pp. 806-810, 1997.

[30] S. Abid, F. Fnaiech, and M. Najim, "A Fast Feedforward Training Algorithm Using a Modified Form of the Standard Backpropagation Algorithm," *IEEE Trans. on Neural Networks*, Vol. 12, No. 2, pp. 424-430, 2001.

[31] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd ed., Wiley-IEEE Computer Society Press, 2001.

[32] T. Bäck, U. Hammel, and H.-P. Schwefel, "Evolutionary Computation: Comments on the History and Current State," *IEEE Trans. on Evolutionary Computation*, Vol. 1, No. 1, pp. 3-17, Apr. 1997.

[33] N. K. Kasabov and Q. Song, "DENFIS: Dynamic Evolving Neural-Fuzzy Inference System and Its Application for Time Series Prediction," *IEEE Trans. on Fuzzy Systems*, Vol. 10, No. 2, pp. 144-154, Apr. 2002.

[34] X. Yao and Y. Liu, "A New Evolutionary System for Evolving Artificial Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 8, No. 3, pp. 694-713, May 1997.

[35] D. Dasgupta and Z. Michalewicz, *Evolutionary Algorithms in Engineering Applications*, Springer-Verlag, 1997.

[36] X. Yao, "Evolving Artificial Neural Networks," *Proceedings of the IEEE*, Vol. 87, No. 9, pp. 1423-1447, Sept. 1999.

[37] X. Yao and Y. Shi, "A Preliminary Study on Designing Artificial Neural Networks Using Co-evolution," *Proc. IEEE Int'l Conf. Intell. Contr. Instrumentation*, Singapore, pp. 149-154, 1995.

[38] D. Whitley and T. Starkweather, "Optimizing Small Neural Networks Using A Distributed Genetic Algorithm," *Proc. Int'l Joint Conf. Neural Networks*, Hillsdale, NJ: Lawrence Erlbaum, Vol. 1, pp. 206-209, 1990.

[39] D. Whitley, T. Starkweather, and C. Bogart, "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity," *Parallel Computing*, Vol. 14, pp. 137-170, 1995.

[40] M. Scholz, "A Learning Strategy for Neural Networks Based on A Modified Evolutionary Strategy," *Proc. Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, eds. Heidelberg: Springer-Verlag, pp. 314-318, 1991.

[41] M. Vittorio, "Genetic Evolution of the Topology and Weight Distribution of Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 5, No. 1, pp. 39-53, 1994.

[42] T. Kumagai, M. Wada, S. Mikami, and R. Hashimoto, "Structured Learning in Recurrent Neural Network using Genetic Algorithm with Internal Copy Operator," *Proc. IEEE Int'l Magnetics Conf.*, pp. 651-656, 1997.

[43] J. R. McDonnell and D. Waagen, "Evolving Recurrent Perceptrons for Time Series Modeling," *IEEE Trans. on Neural Networks*, Vol. 5, No. 1, pp. 24-38, 1994.

[44] P. J. Angeline, G. M. Sauders, and J. B. Pollack, "An Evolutionary Algorithm that Constructs Recurrent Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 5, No. 1, pp. 54-65, 1994.

[45] D. J. Montana and L. Davis, “Training Feedforward Neural Networks Using Genetic Algorithms,” *Proc. 11th Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 762-767, 1989.

[46] S.-W. Lee, “Off-line Recognition of Totally Unconstrained Handwritten Numerals Using Multilayer Cluster Neural Network,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 6, pp. 648-652, June 1996.

[47] S. L. Hung and H. Adeli, “A Parallel Genetic/Neural Network Learning Algorithm for MIMD Shared Memory Machines,” *IEEE Trans. on Neural Networks*, Vol. 5, No. 6, pp. 900-909, Nov. 1994.

[48] H. Zhang and M. Ishikawa, “A Hybrid Real-Coded Genetic Algorithm with Local Search,” *Proc. 12th Int’l Conf. on Neural Information Processing (ICONIP2005)*, pp. 732-737, Taipei, R.O.C., 2005.

[49] S. Kothandaraman, Implementation of Block-based Neural Networks on Reconfigurable Computing Platforms, MS Thesis, University of Tennessee, Aug. 2004.

[50] S. W. Moon and S. G. Kong, “Pattern Recognition with Block-based Neural Networks,” *Proc. Int’l Joint Conf. on Neural Networks (IJCNN-2002)*, pp. 992-996, May 2002.

[51] S. G. Kong, “Time Series Prediction with Evolvable Block-based Neural Networks,” *Proc. Int’l Joint Conf. on Neural Networks (IJCNN-2004)*, Vol. 2, pp. 1579-1583, July 2004.

[52] W. Jiang and S. G. Kong, "A Least-Squares Learning for Block-based Neural Networks," in press, *Dynamics of Continuous, Discrete and Impulsive Systems*, Special Volume: Advances in Neural Networks - Theory and Applications, 2007.

[53] G. Estrin, "Organization of Computer Systems - The Fixed Plus Variable Structure Computer," *Proc. Western Joint Computer Conf.*, pp. 33-40, New York, 1960.

[54] H. DeGaris, "Evolvable Hardware: Principles and Practice," http://www.cs.usu.edu/degaris/papers/CACM-E-Hard.html, Aug. 1997.

[55] A. Thompson, An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics, *Proc. 1st Int'l Conf. on Evolvable Systems*, 1996.

[56] E. Stomeo, T. Kalganova, and C. Lambert, "Generalized Disjunction Decomposition for Evolvable Hardware," *IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics*, Vol. 36, No. 5, Oct. 2006.

[57] T. Higuchi, M. Murakawa, M. Iwata, I. Kajitani, W. Liu, and M. Salami, "Evolvable Hardware at Function-Level," *Proc. IEEE Int'l Conf. Evol. Comput.*, pp. 187-192, Apr. 1997.

[58] T. Kalganova, "An Extrinsic Function-Level Evolvable Hardware Approach," *Proc. 3rd EuroGP*, R. Poli and W. Banzhaf, Eds, Edinburgh, U.K., pp. 60-75, Apr. 2000.

[59] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano, "Hardware Spiking Neural Network with Run-Time Reconfigurable Connectivity in an Autonomous Robot," *Proc. 2003 NASA/DoD Conference on Evolvable Hardware*, pp 189-198, 2003.

[60] D. Earl, "Development of an FPGA-based Hardware Evaluation System for use with GA-designed Artificial Neural Networks," Ph.D. Dissertation, University of Tennessee, May 2004.

[61] S. Merchant, G. D. Peterson, S. K. Park, and S. G. Kong, "FPGA Implementation of Evolvable Block-based Neural Networks," *Proc. Congress on Evolutionary Computation (CEC-2006)*, Vancouver, Canada, July 2006.

[62] P. Martin, "A Hardware Implementation of a GP System Using FPGAs and Handel-C," *Genetic Programming and Evolvable Machine*, Vol. 2, No. 4, pp. 317-343, 2001.

[63] B. Shackleford, G. Snider, R. Carter, E. Okushi, M. Yasuda, K. Seo, and H. Yasuura, "A High Performance, Pipelined, FPGA-based Genetic Algorithm Machine," *Genetic Programming and Evolvable Machine*, Vol. 2, No. 1, pp. 33-60, 2001.

[64] N. M. Botros and M. Abdul-Aziz, "Hardware Implementation of An Artificial Neural Network Using Field Programmable Gate Arrays (FPGA's)," *IEEE Trans. on Industrial Electronics*, Vol. 41, No. 6, pp. 665-667, 1994.

[65] F. Yang and M. Paindavoine, "Implementation of An RBF Neural Network on Embedded Systems: Real-time Face Tracking and Identity Verification," *IEEE Trans. on Neural Networks*, Vol. 14, No.5, pp. 1162-1175, 2003.

[66] T. C. Fogarty, "An Incremental Genetic Algorithm for Real-Time Optimisation," *Proc. Int'l Conf. on Systems, Man and Cybernetics 1989*, pp. 321-326, 1989.

[67] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, Springer-Verlag, 1998.

[68] M. Li and H.-Y. Tam, "Hybrid Evolutionary Search Method Based on Clusters," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 23, No. 8, pp. 786-799, 2001.

[69] T. Kuo and S.-Y. Hwang, "A Genetic Algorithm with Disruptive Selection," *IEEE Trans. on Systems, Man and Cybernetics - Part B*, Vol. 26, No. 2, pp. 299-307, 1996.

[70] T. Blickle and L. Thiele, "A Mathematical Analysis of Tournament Selection," *Proc. 6th Int'l Conf. on Genetic Algorithms (ICGA-95)*, pp. 9-16, 1995.

[71] B. A, Julstrom, "It's All the Same to Me: Revisiting Rank-Based Probabilities and Tournaments," *Proc. Congress on Evolutionary Computation (CEC-99)*, Vol. 2, pp. 1501-1505, 1999.

[72] J. Sarma and K. De Jong, "An Analysis of Local Selection Algorithms in a Spatially Structured Evolutionary Algorithm," *Proc. 7th Int'l Conf. on Genetic Algorithms (ICGA-97)*, pp. 181-186, 1997.

[73] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd Edition, Upper Saddle River, N.J.: Prentice Hall, 1999.

[74] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd Edition, New York: Wiley-Interscience, 2000.

[75] J. T. Catalano, *Guide to ECG analysis*, Lippincott, 2002.

[76] R. Hoekema, G. J. H. Uijen, and A. van Oosterom, "Geometrical Aspects of the Interindividual Variability of Multilead ECG Recordings," *IEEE Trans. on Biomedical Engineering*, Vol. 48, No. 5, pp. 551-559, 2001.

[77] P. de Chazal, M. O'Dwyer, and R. B. Reilly, "Automatic Classification of Heartbeats Using ECG Morphology and Heartbeat Interval Features," *IEEE Trans. on Biomedical Engineering*, Vol. 51, No. 7, pp. 1196-1206, 2004.

[78] C. Alexakis, H. O. Nyongesa, R. Saatchi, N. D. Harris, C. Davis, C. Emery, R. H. Ireland, and S. R. Heller, "Feature Extraction and Classification of

Electrocardiogram (ECG) Signals Related to Hypoglycaemia," *Proc. Computers in Cardiology*, Vol. 30, pp. 537-540, 2003.

[79] M. Lagerholm, C. Peterson, G. Braccini, L. Edenbrandt, and L. Sörnmo, "Clustering ECG Complexes Using Hermite Functions and Self-organizing Maps," *IEEE Trans. on Biomedical Engineering*, Vol. 47, No. 7, pp. 838-848, 2000.

[80] H. Haraldsson, L. Edenbrandt, and M. Ohlsson, "Detection Acute Myocardial Infarction in the 12-lead ECG Using Hermite Expansions and Neural Networks," *Artificial Intelligence in Medicine*, Vol. 32, pp. 127-136, 2004.

[81] T. H. Linh, S. Osowski, and M. Stodolski, "On-line Heart Beat Recognition Using Hermite Polynomials and Neuro-Fuzzy Network," *IEEE Trans. on Instrumentation and Measurement*, Vol. 52, No. 4, pp. 1224-1231, 2003.

[82] P. de Chazal and R. B. Reilly, "A Comparison of the ECG Classification Performance of Different Feature Sets," *Proc. Computers in Cardiology*, Vol. 27, pp. 327-330, 2000.

[83] S. Osowski, L. T. Hoai, and T. Markiewicz, "Support Vector Machine-based Expert System for Reliable Heartbeat Recognition," *IEEE Trans. on Biomedical Engineering*, Vol. 51, No. 4, pp. 582-589, Apr. 2004.

[84] Y. H. Hu, W. J. Tompkins, J. L. Urrusti, and V. X. Afonso, "Applications of Artificial Neural Networks for ECG Signal Detection and Classification," *Journal of Electrocardiology*, Vol. 26 (Suppl.), pp. 66-73, 1994.

[85] R. Silipo and C. Marchesi, "Artificial Neural Networks for Automatic ECG Analysis," *IEEE Trans. on Signal Processing*, Vol. 46, No. 5, pp. 1417-1425, 1998.

[86] Y. Hu, S. Palreddy, and W. J. Tompkins, "A Patient-Adaptable ECG Beat Classifier Using a Mixture of Experts Approach," *IEEE Trans. on Biomedical Engineering*, Vol. 44, No. 9, pp. 891-900, 1997.

[87] D. A. Coast, R. M. Stern, G. G. Cano, S. A. Briller, "An Approach to Cardiac Arrhythmia Analysis using Hidden Markov Models," *IEEE Trans. on Biomedical Engineering*, Vol. 37, No. 9, pp. 826-836, 1990.

[88] R. V. Andreao, B. Dorizzi, and J. Boudy, "ECG Signal Analysis through Hidden Markov Models," *IEEE Trans. on Biomedical Engineering*, Vol. 53, No. 8, pp. 1541-1549, 2006.

[89] S. B. Ameneiro, M. Fernández-Delgado, J. A. Vila-Sobrino, C. V. Regueiro, and E. Sánchez, "Classifying Multichannel ECG Patterns with an Adaptive Neural Network," *IEEE Engr. in Medicine and Biology*, Vol. 17, No. 1, pp. 45-55, 1998.

[90] M. Fernández-Delgado and S. B. Ameneiro, "MART: A Multichannel ART-Based Neural Network," *IEEE Trans. on Neural Network*, Vol. 9, No. 1, pp. 139-150, 1998.

[91] http://www.physionet.org/physiotools.

[92] N. Ahmed, P. J. Milne, and S. G. Harris, "Electrocardiographic Data Compression via Orthogonal Transforms," *IEEE Trans. on Biomedical Engineering*, Vol. 22, No. 6, pp. 484-487, 1975.

[93] L. Sörnmo, P. O. Börjesson, M. E. Nygårds, and O. Pahlm, "A Method for Evaluation of QRS Shape Features Using A Mathematical Model for the ECG," *IEEE Trans. on Biomedical Engineering*, Vol. 28, No. 10, pp. 713-717, 1981.

[94] R. Jane, S. Olmos, P. Laguna, and P. Caminal, "Adaptive Hermite Models for ECG Data Compression: Performance and Evaluation with Automatic Wave Detection," *Proc. Computers in Cardiology 1993*, pp. 389-392, 1993.

[95] J. A. Swets, "Measuring the Accuracy of Diagnostic Systems," *Science*, Vol. 240, pp. 1285-1293, June 1998.

[96] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd Ed, Johns Hopkins University Press, Baltimore, MD, 1996.

[97] M. C. Mackey and L. Glass, "Oscillation and Chaos in Physiological Control Systems," *Science*, Vol. 197, pp. 287-289, 1977.

[98] S. Bittanti and L. Piroddi, "Nonlinear Identification and Control of a Heat Exchanger: A Neural Network Approach," *Journal of Franklin Institute*, Vol. 334B, No. 1, pp.135-153, 1997.

[99] R. F. Demara and K. Zhang, "Automous FPGA Fault Handling through Competitive Runtime Reconfiguration," *Proc. NASA/DoD Conference of Evolution Hardware*, 2005.

[100] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica, "Fault-Tolerant Evolvable Hardware Using Field-Programmable Transistor Arrays," *IEEE Trans. on Reliability*, Vol. 49, No. 3, pp. 305-316, 2000.

[101] P. R. Menon, W. Xu, and R. Tessier, "Design-Specific Path Delay Testing in Lookup Table-based FPGAs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 5, pp.867-877, 2006.

[102] G. W. Greenwood, "On the Practicality of Using Intrinsic Reconfiguration for Fault Recovery," *IEEE Trans. on Evolutionary Computation*, Vol. 9, No. 4, pp. 398-405, 2005.

[103] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally Weighted Learning," *Artificial Intelligence Review*, Vol. 11, No. 1-5, pp. 11-73, 1997.

VITA

Wei Jiang was born in Nanchong, Sichuan, China in 1977. He received a Bachelor of Science degree in Electrical Engineering in 2000 from Sichuan University, Sichuan, China and a Master of Science degree in Computer Engineering in 2004 from the University of Tennessee, Knoxville, U.S.A. Since he joined in the Department of Electrical and Computer Engineering at the University of Tennessee in 2002, he has worked in the Lab directed under Dr. Seong G. Kong as a research assistant. He has been working on a 3-year project sponsored by National Science Foundation, titled "Evolving Block-based Neural Networks for Dynamic Environments". His major research areas include signal/image processing, artificial neural networks, and artificial intelligence. He will complete his Doctor of Philosophy degree in Computer Engineering in summer 2007.