

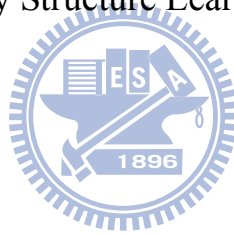
國立交通大學

電控工程研究所

博士論文

具演化式結構學習能力之類神經網路及其預測之應用

Neural Network with Evolutionary Structure Learning and Its Prediction Application



研究生：楊世宏

指導教授：陳永平 博士

中華民國一百年十二月

具演化式結構學習能力之類神經網路及其預測之應用

Neural Network with Evolutionary Structure Learning and Its Prediction Application

研究生：楊世宏

Student : Shih-Hung Yang

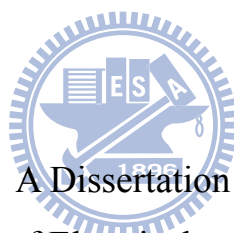
指導教授：陳永平 博士

Advisor : Dr. Yon-Ping Chen

國立交通大學

電控工程研究所

博士論文



Submitted to Institute of Electrical and Control Engineering

College of Electrical Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical and Control Engineering

December 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年十二月

誌謝

感謝傅立成老師、徐國鎧老師、林進燈老師、宋開泰老師、楊谷洋老師、高立人老師撥冗給予口試以及寶貴的建議，讓我從另一個層面來思考如何做研究，對我將來的工作有莫大的幫助。

感謝我的指導老師陳永平博士，指導我如何組織論文，撰寫論文，釐清我想表達的意思，也包容我走的比別人慢，提供舒適的研究空間，願意每個禮拜花時間跟我討論，從討論過程中，可以得到相當多經驗與傳承。我從陳永平老師的討論當中學習很多，他曾經陪著我改過無數的論文，幫助我突破眾多難關，耐心解決很多困境，在我艱困時，拉我一把，陳老師用他寬大的氣度、冷靜的思考、積極的人生觀，帶著我走過許多荊棘之路，也讓我見識以及學習到重要且關鍵的做人處事態度，我從他身上學到的比從論文上還多，謹以此論文代表我對他的致謝。

這幾年來是我求學時光最扎實的日子，清大江老師與楊師母常常邀請我至他們家作客，在他們溫暖的屋子裡，喝著香醇的果汁，吃著美味的餅乾，分享他們的故事，不斷鼓勵我，也給予我很多寶貴的經驗與知識，在這裡感謝他們的陪伴。

感謝學長葉天德博士在我低迷、徬徨、無助的時候拉我一把，支持與鼓勵我向前，認可我的研究成果，提升我的信心，在關鍵的時刻，提供我寶貴的意見，使我可以放心向前走。

感謝同袍丁桓展博士在 VSS 實驗室一同打拼，度過漫長的求學生涯，他義氣相挺的個性，有求必應，只要他知道的，一定不吝回答，他灑脫的個性，減輕我許多壓力與煩惱，讓人生更順暢。

感謝 VSS 實驗室學弟振芳、孫齊、咨瑋協助口試的庶務，也感謝實驗室所有

學弟精神上的支持。

謝謝小啾、孟穎、玉禎老師提供我寶貴的意見，讓我在徬徨時，能夠冷靜下來，先整理好心情，再來處理事情。感謝手語 91 的支持，記得曾與你們共患難。感恩好友河志、峻延、宏鈞經常鼓勵我，透過聊天來紓解我的壓力，他們的義氣相挺，持續地互相關心，讓我安心度過許多難關。

感謝系辦溫嘉雯小姐在行政業務上的幫助，也感謝系辦林滿足小姐、陳英芝小姐、施德喜先生、曾桂香小姐在計畫業務上的幫助。感謝黃俊德提供博士論文的格式範本，讓寫作更順利。

感恩承瑄師姑、玫芬師姑、博文師伯帶著我登上各種舞台，也同時照顧我的心，讓我以他們為榜樣，學習他們的價值觀來看世界，選擇更為正確的人生方向。

最重要的是感謝我的家人對我的支持，讓我完成學業，體諒我這段時間經濟上的限制，長時間在外地打拼，感謝我兩位妹妹，感謝我的爸爸與媽媽。

楊世宏

一百年十二月

具演化式結構學習能力之類神經網路及其預測之應用

研究生：楊世宏

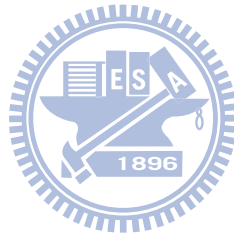
指導教授：陳永平 博士

國立交通大學電控工程研究所博士班

摘 要

本論文提出一以前饋式類神經網路輔助之灰色模型及其相關的線上參數學習與結構學習演算法，此模型採用一階單變數灰色模型來預測訊號，再使用前饋式類神經網路補償灰色模型的預測誤差，此外，本論文提出一線上批次訓練法來即時更新類神經網路的權重值，於是，此模型可以執行預測且持續地適應動態的訊號變化。為了有效地設計此模型的結構，本論文提出一種以神經元為基礎的結構學習，稱為共生結構學習演算法，來建立類神經網路的拓撲結構，此演算法首先建構一神經元族群，再由神經元族群建立類神經網路族群，由於神經元族群裡的神經元包含雙曲線正切與線性活化函數，此演算法能任意且輕易地發展串聯式網路與前饋式網路，此演算法進一步根據共生進化的概念，在神經元族群裡執行神經元交配與突變，其所發展的前饋式類神經網路輔助之灰色模型將執行訊號預測且持續地以線上批次訓練法調適模型於環境中。另一方面，本論文提出一種以網路為基礎的結構學習，稱為演化式建構與修剪演算法，用演化的方式結合建構與修剪的概念，來設計類神經網路的拓撲結構。此演算法從一群具有最簡單結構的類神經網路開始，即一群只有一顆連接單一輸入單元的隱藏層神經元的類神經網路，此演算法採用網路交配與突變來增加隱藏層神經元以及鏈結，用以提升類神

經網路的訊號處理能力，此外，本論文提出一以叢集為基礎之修剪法用隨機的方式來除去不重要的神經元，也提出一以年齡為基礎之生存者選擇法來移除較老且可能具有複雜結構的類神經網路，接著引進新的且具有最簡單結構的類神經網路。數值模擬與實驗結果將展現所提出的方法在預測問題上的有效及可行性。



Neural Network with Evolutionary Structure Learning and Its Prediction Application

Student : Shih-Hung Yang

Advisor : Dr. Yon-Ping Chen

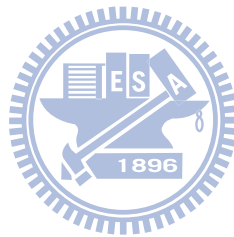
Institute of Electrical and Control Engineering

National Chiao Tung University

Abstract

This dissertation proposes a feedforward-neural-network-aided grey model (FNAGM) and its related on-line parameter learning as well as structure learning algorithms. The FNAGM uses a first-order single variable grey model (GM(1,1)) to predict signal and adopts a feedforward neural network (NN) to compensate the prediction error of GM(1,1). Furthermore, an on-line batch training is proposed to update the weights of NN in real-time. Thus, FNAGM can precisely predict and adapt itself to the dynamical change of the signal. To design the structure of FNAGM efficiently, a neuron-based structure learning, called symbiotic structure learning algorithm (SSLA), is proposed to establish the topology of NN. The SSLA constructs a neuron population and then builds a network population from the neuron population, and it can arbitrarily develop cascade NNs and feedforward NNs in an easy way. Further, SSLA carries out neuron crossover and mutation on the neuron population according to the idea of symbiotic evolution. The evolved FNAGM is applied to predict the signal and continuously adapt itself to the environment by the on-line batch training. On the other hand, a network-based structure learning, called evolutionary constructive and

pruning algorithm (ECPA), is proposed to design the topology of NN by incorporating constructive and pruning methods in an evolutionary way. The ECPA starts from a set of NNs with the simplest possible structures, one hidden neuron connected to an input node. It then adds hidden neurons and connections by using the network crossover and mutation to increase the processing capabilities of NNs. Furthermore, a cluster-based pruning is proposed to prune insignificant neurons in a stochastic way. An age-based survival selection is proposed to delete old NNs with potentially complex structures and then introduce new NNs with the simplest possible structures. Numerical and experimental results of prediction problems show the effectiveness and feasibility of the proposed methods.



CONTENTS

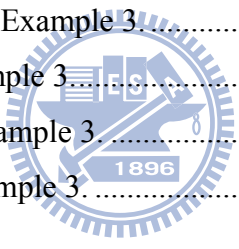
摘要	i
ABSTRACT	iii
誌謝	i
CONTENTS	vii
LIST OF FIGURES	ix
LIST OF TABLES	xi
SYMBOLS	xii
Chapter 1 Introduction.....	1
1.1 Motivation	1
1.2 Literature Survey	3
1.2.1 On-Line Parameter Learning.....	3
1.2.2 Structure Learning.....	5
1.3 Organization of Dissertation.....	10
Chapter 2 On-Line Parameter Learning for Prediction	12
2.1 Feedforward-Neural-Network-Aided Grey Model.....	12
2.1.1 Neural Networks	12
2.1.2 First-Order Single Variable Grey Model.....	14
2.1.3 Structure of FNAGM	16
2.2 On-Line Parameter Learning of FNAGM	17
2.2.1 On-Line Batch Training	18
2.2.2 Convergence Analysis	21
2.3 Numerical Results.....	22
2.3.1 Example 1: Disturbance Prediction.....	22
2.3.2 Example 2: Chaotic Time Series Prediction.....	25
2.4 Experimental Results.....	27
2.4.1 Trajectory Prediction.....	28
2.4.2 Tracking Control	31
2.5 Summary.....	34
Chapter 3 Neuron-Based Structure Learning for Prediction	35

3.1	Structure Learning Based on Symbiotic Evolution	35
3.2	Symbiotic Structure Learning Algorithm	36
3.2.1	Initialization Phase	37
3.2.2	Evaluation Phase	41
3.2.3	Reproduction Phase.....	43
3.3	Numerical Results.....	48
3.3.1	Example 1: Chaotic Time Series Prediction.....	49
3.3.2	Example 2: Object Trajectory Prediction	53
3.4	Summary.....	56
Chapter 4	Network-Based Structural Learning for Prediction	58
4.1	Basic Concept of Evolutionary algorithm	58
4.2	Evolutionary Constructive and Pruning Algorithm	59
4.2.1	Encoding Scheme and Design Mechanism	61
4.2.2	Network Crossover.....	62
4.2.3	Network Mutation	64
4.2.4	Cluster-Based Pruning.....	65
4.2.5	Age-Based Survival Selection.....	68
4.3	Numerical Results.....	69
4.3.1	Example 1: Chaotic Time Series Prediction.....	71
4.3.2	Example 2: Forecasting the Number of Sunspots	76
4.3.3	Example 3: Vehicle Count Prediction	79
4.3.4	Effect of CBP and ABSS.....	83
4.3.5	Discussion	85
4.4	Summary.....	86
Chapter 5	Conclusion and Future Work	88
	Bibliography	90
	Vita.....	102
	Publication List.....	103

LIST OF FIGURES

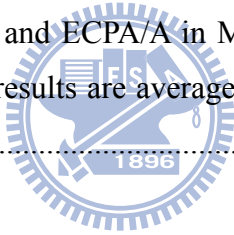
Fig. 2.1	Feedforward-neural-network-aided grey model.....	17
Fig. 2.2	Absolute prediction errors of GM(1,1), Advanced GM(1,1) and FNAGM with $m = 4$ in Example 1.....	25
Fig. 2.3	Absolute prediction errors of GM(1,1), Advanced GM(1,1) and FNAGM with $m = 4$ in Example 2.....	27
Fig. 2.4	(a) Eye-Robot. (b) Experimental environment where a participant held a red object.	28
Fig. 2.5	Trajectory of the object.....	30
Fig. 2.6	Prediction error of $x(k)$ for GM(1,1), advance GM(1,1), and FNAGM.	30
Fig. 2.7	Prediction error of $y(k)$ for GM(1,1), advance GM(1,1), and FNAGM.	31
Fig. 2.8	Tracking error of $x(k)$ using GM(1,1).....	32
Fig. 2.9	Tracking error of $x(k)$ using Advanced GM(1,1).....	33
Fig. 2.10	Tracking error of $x(k)$ using FNAGM.....	33
Fig. 3.1	Architecture of the proposed forecasting system.	37
Fig. 3.1	Coding of neuron and three examples.....	38
Fig. 3.2	Graphical representation of neurons for three examples.....	38
Fig. 3.3	(a) An NN constructed via three neurons. (b) Equivalent cascade NN model.	39
Fig. 3.4	Neuron population and network population.	41
Fig. 3.5	Neuron Crossover.	45
Fig. 3.6	Neuron Mutation.....	45
Fig. 3.8	Neuron reproduction.....	46
Fig. 3.9	Flowchart of SSLA.	48
Fig. 3.10	Mackey-Glass time series.....	49
Fig. 3.11	Evolved NN of FNAGM-SSLA for Example 1.....	50
Fig. 3.12	Absolute prediction errors in Example 1. (a) GM(1,1) [3]; (b) Advanced GM(1,1) [23]; (c) NNon [79]; (d) NNoff [79]; (e) FNAGM; (f) FNAGM-SSLA.....	52
Fig. 3.13	Object trajectory captured by Eye-Robot.	53
Fig. 3.14	Evolved NN of FNAGM-SSLA for Example 2.....	54

Fig. 3.15	Absolute prediction errors in Example 2. (a) GM(1,1) [3]; (b) Advanced GM(1,1) [23]; (c) NNNon [79]; (d) NNoff [79]; (e) FNAGM; (f) FNAGM-SSLA.....	55
Fig. 4.1	Major steps performed in ECPA.....	60
Fig. 4.2	An example of a network crossover.	64
Fig. 4.3	An example of a network mutation.	66
Fig. 4.4	Evolution progress for Example 1.....	73
Fig. 4.5	Evolved NNs for Example 1.....	73
Fig. 4.6	Prediction error for Example 1.....	75
Fig. 4.7	Multiple-step prediction error for Example 1.....	75
Fig. 4.8	Evolution progress for Example 2.....	77
Fig. 4.9	Evolved NNs for Example 2.....	77
Fig. 4.10	Training results for Example 2.....	78
Fig. 4.11	Testing results for Example 2.....	79
Fig. 4.12	Evolution progress for Example 3.....	81
Fig. 4.13	Evolved NNs for Example 3.....	81
Fig. 4.14	Training results for Example 3.....	82
Fig. 4.15	Testing results for Example 3.....	82



LIST OF TABLES

Table 2.1	Comparison of different input numbers of NN of FNAGM in Example 1. .	23
Table 2.2	Comparison of the prediction errors of GM(1,1), Advanced GM(1,1) and FNAGM in Example 1.	24
Table 2.3	Comparison of the prediction errors of GM(1,1), Advanced GM(1,1) and FNAGM in Example 2.	26
Table 2.4	Comparison of prediction error and computation time.	29
Table 3.1	Comparison of prediction results for Example 1.....	53
Table 3.2	Comparison of prediction results for Example 2.....	56
Table 4.1	Prediction results for Example 1.	74
Table 4.2	Prediction results for Example 2.	79
Table 4.3	Prediction results for the hourly vehicle count time series.....	83
Table 4.4	Performance of ECPA and ECPA/A in Mackey-Glass, sunspot, and vehicle count time series. All results are averaged over 10 independent runs, where * refers to RMSE.	84



SYMBOLS

FNAGM

N_s	total step size
n	input length of GM(1,1)
γ	bias of GM(1,1) in (2.15)
m	input number of NN
p	number of hidden neurons of NN
N	maximum size of batch training pattern
$\mathbf{v}[n+m]$	initial weight vector of NN
$\mathbf{u}[k]$	input vector at time k
$g(\cdot)$	activation function
$x[k]$	discrete data at time k
$\mu[n+m+1]$	initially positive scalar for updating the weight vector
β	constant to adjust $\mu[k]$
\mathbf{P}	training pattern
\mathbf{B}	batch training pattern
\mathbf{G}	Jacobian matrix

SSLA

w	output weight
a	activation function type
N_p	population size
P	number of groups
d	number of individuals in each group
φ	number of epochs in backpropagation algorithm

f	fitness
p_c	crossover probability
p_m	mutation probability
r	random number with a uniform distribution between [0, 1]
N_h	number of hidden neurons
N_c	number of connections
T_c	computation time of each prediction step

ECPA

σ	significance of hidden neuron
S	sensitivity of the network output to the output of hidden neuron
Age	age of neural network
R_c	connection ratio
T_c	computation time of evolution



Chapter 1 Introduction

1.1 Motivation

Prediction, computing the trends of time series in the future, is a type of problem generally encountered in many research fields. Many numerical algorithms that can accurately predict time series have been proposed, including the autocorrelation method [1], covariance method [2], grey theory [3], and Kalman filter [4]. However, these methods require suitable system equation and initial values which are designed heuristically [5], [6]. Recently, investigators have focused on data-driven approaches based on neural networks (NNs) due to their learning abilities and powerful prediction capability so that the implicit nonlinear relationships can be extracted from historical data without human experience [7]-[10].

NNs were first developed to imitate biological neural systems and are organized into several interconnected simple processing units called neurons or nodes. NNs learn from examples and historical data, even when the input-output relationships are unknown [11]. Thus, NNs can accurately solve problems without prior knowledge when sufficient observed data are supplied. This property is useful for evaluating numerous forecasting problems because acquiring data is easier than making good theoretical guesses about certain systems.

Two important issues are discussed in NN research: parameter learning and structure learning. The parameter learning basically consists of two classes of learning strategies, called batch training and on-line training. The batch training adjusts the

network by using a large number of prepared training data and performs weight search in a fixed error surface. Although the weight search converges, it is easily trapped in a local minimum. In order to achieve high forecasting accuracy, however, a large amount of data for successful training is needed and lots of training time is also required. It, therefore, may not be suitable for some requirements, such as real-time operations, limited memory size or on-line adaptation necessity [12]–[15]. As opposed to the batch training, on-line training adjusts NN according to a single training pattern, a pair of input and target, at a time. It is particularly adequate for the purpose to simultaneously execute signal prediction and learn to improve the performance [16],[17]. Although the on-line training usually takes much less computation time to adjust the network than the batch training does, the former is not easy to achieve accurate solutions as well as the latter. Therefore, design of an appropriate on-line training algorithm is necessary to perform on-line prediction and continuous adaptation for real-time application.

Another important issue of NN is structure selection, which involves determining an appropriate structure to accurately fit the underlying function described by the training data [18]. A structure that is too large may precisely fit the training data but may provide poor generalization due to overfitting of the training data. Conversely, an architecture that is too small saves computational costs but may not possess sufficient processing ability to accurately approximate the underlying function. Therefore, structure selection should consider both network complexity and goodness of fit. This dissertation proposes two structure learning algorithms: symbiotic structure learning algorithm (SSLA) and evolutionary constructive and pruning algorithm (ECPA). SSLA, a neuron-based structure learning approach, can automatically determine both the number of hidden neurons and topology of NN through a symbiotic evolution. ECPA, a network-based structure learning approach, can evolve NN through evolutionary

constructive and pruning approaches. Results of this dissertation demonstrate the effectiveness of the proposed methods.

1.2 Literature Survey

This section presents the literature survey of on-line parameter learning and structure learning.

1.2.1 On-Line Parameter Learning

On-line parameter learning is one of the most useful and popular methods for many real-world applications. It continuously adapts model parameters to the environment after each training pattern is presented at each time step. For time series prediction tasks, NNs are used to predict the next data point according to the last finite number of data points. The training patterns are presented in their natural order and maintain their natural dependencies. The on-line parameter learning constructs a stochastic error surface in nature due to the use of pattern-by-pattern updating of weights. This training seems to introduce some sort of randomness and help escape from local minimum [19], [20]. Mathematically, there is no guarantee of its stability or the convergence of the weight search. In [21], researcher found that if the learning rate is small enough, the weight search would converge. Basically, on-line gradient descent has been the first effective approach for training NNs through error backpropagation [22]. However, one difficulty using on-line learning for prediction applications is the sensitivity to the selection of training parameters and training patterns. The other difficulty is to determine the Hessian on-line when only a single training example is available at one time step [13],[23]. This is especially undesirable when the prediction accuracy is

requirement of using on-line approaches.

Aside from NNs, grey theory is used to cope with the systems with partial information or dynamic model [24]. Based on the grey theory, the first-order single variable grey model (GM(1,1)) has been developed and applied to various on-line prediction problems, such as power demand forecasting [1], electricity demand forecasting [25] and control of a humanoid robot [26]. GM(1,1) requires only a few number of historical data to adapt its parameters on-line and is used to predict exponential signals in real time. However, it is inadequate to predict other types of signals. In [27], Advanced GM(1,1) adopts the Lagrange polynomial to estimate the prediction error of GM(1,1) and indeed improves the prediction.

Based on the approach of “mixture of experts”, some researchers have integrated GM(1,1) and NN to enhance the prediction according to their complementary merits. Generally, the ways to combine GM(1,1) and NN can be categorized into four classes [28]: simple combination, serial combination, strengthening grey system with neural network, building neural network with the aid of grey system. The simple combination integrated the outputs of both GM(1,1) and NN with the coefficients determined via least mean square method [29], minimum theory of statistical variance [30], [31], and Shapley value method [32]. One kind of serial combination made use of the accumulation generation operation (AGO) to transform the original data into first order AGO (1-AGO) data with improved regularity and then fed the 1-AGO data to the input layer of NN [33], [34]. Zhu [35] proposed a kind of serial combination to fed the prediction output of GM(1,1) to the input layer of NN which was trained by the momentum algorithm. In addition, Chang and Tsai [36] improved the control and environment parameters of GM(1,1) by a support vector regression, denoted by SVRGM and employed a statistical methodology GARCH to fit the time series.

Consequently, a back-propagation neural network was used to tune the weighted-average between SVRGM and GARCH. In strengthening grey system with neural network, Hsu and Chen [37] applied NN to estimate the residual sign of GM(1,1) and the work in [38], [39] forecasted the prediction error of GM(1,1). For the ways building NN with the aid of grey system, Yeh et al. [40] trained GreyART network which combines grey relational analysis and adaptive resonant theory network and then evaluated GreyART network by the testing patterns generated by GM(1,1).

The aforementioned studies show that the fusion scheme of GM(1,1) and NN could outperform the individual ones. However, these models are trained off-line and only suitably applied to the prediction problems for hourly [30], daily [29], monthly [33], and yearly [35] time-series. To carry out continuous adaptation for prediction problem in real-time, it is required to perform the prediction and training during the sampling interval. For most of the practical cases in real-time operation, the sampling time is less than one second, much longer than the training time needed for one training pattern [41]. Thus, it is able to perform more than one pattern learning, such as batch pattern, during one sampling interval. Therefore, this dissertation proposes a feedforward-neural-network-aided grey model (FNAGM) based on GM(1,1) and NN to learn the prediction error of GM(1,1). Furthermore, an on-line batch training is proposed to continually adapt the network to the dynamical change for real-time prediction.

1.2.2 Structure Learning

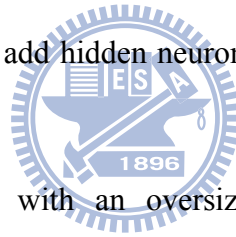
For prediction purposes, it has been shown that a feedforward NN with a single hidden layer is sufficient to achieve any desired accuracy [42], [43]. In most applications, NNs are fully connected, i.e., all inputs are fully connected to all hidden

neurons. Numerous studies have shown that partially connected NNs have better storage capability per connection than fully connected NNs [44]-[46]. Furthermore, partially connected NNs can yield improved generalization capabilities with reduced cost in terms of hardware and processing time [47], [48]. However, how to determine the optimal numbers of hidden neurons and connections remains an open question.

Among several algorithms for designing three-layered NNs, the most frequently used algorithms are the constructive, pruning, and constructive-pruning algorithms [49], [50]. A constructive algorithm starts with a minimal NN architecture, a three-layered NN with one hidden neuron. The algorithm adds hidden neurons to the minimal NN, one-by-one, during the training phase. The advantage of the constructive algorithm is that the initial phase can simply set the number of hidden layers and neurons as one each. However, deciding when to add hidden neurons or connections and when to stop the addition process is difficult.

A pruning algorithm starts with an oversized architecture and then deletes unnecessary hidden neurons or connections, either during training or upon convergence to a local minimum. Each iteration of the pruning algorithm determines which unit, i.e., which hidden neuron or connection, to prune via its relevance or significance. Several pruning criteria have been proposed, for example, sensitivity analysis [51] and magnitude-based pruning [52]. Sensitivity analysis is based on Taylor expansion and reflects the ways in which the derivatives of a performance function can be applied to quantify a system's response to unit perturbations [53], [54]. Magnitude-based pruning assumes that small weights are irrelevant [55]. However, no criterion can be used to determine the initially oversized architecture for a given problem [49].

In the constructive algorithm, the architecture of NN may become oversized if the addition procedure is not appropriately stopped. A number of algorithms have attempted



to combine constructive and pruning algorithms to solve the aforementioned problem [56]-[58]. These constructive-pruning algorithms first estimate the number of hidden neurons and/or connections via a constructive method. A pruning method is then used to delete the inappropriate hidden neurons and/or connections to find a near-optimal architecture for a given problem. However, determining when to stop the pruning procedure is difficult [59].

Several researchers have developed methods for designing NNs using evolutionary algorithms (EAs). EAs emerged as a biologically plausible approach for adapting various NN parameters such as weight values and architectures [60]-[63]. Unlike constructive, pruning, and constructive-pruning algorithms, EAs perform a search employing a population of NNs rather than a single NN. The population-based stochastic search technique uses crossover, mutation, and selection operators in each generation to improve the NN population in the search space. In contrast, the constructive, pruning, and constructive-pruning algorithms apply predefined and greedy search strategies to determine near-optimal NN architectures. These strategies are appropriate for some tasks but may be inappropriate for other tasks because the greedy search strategies may direct the search process toward architecturally local optima, which is a problem inherent to any greedy approach [64]. EAs can avoid the architecturally local optima problem by using non-monotonic search methods. Gutiérrez *et al.* [65] adopted an evolutionary programming algorithm and a simulated annealing method to produce a radial basis function neural network with simplest structure possible for classification problems. Oong and Isa [66] achieve the global and local search to evolve NNs via adapting the mutation probability and the step size of the weight perturbation. Caballero *et al.* [67] use a Pareto-based multiobjective optimization methodology based on a memetic evolutionary algorithm for multiclass

problems.

Recently, several studies have been proposed to employ various EAs to prune NNs. Mantzaris *et al.* [68] pruned Probabilistic Neural Network by genetic algorithm to minimize the number of diagnostic factors, and therefore minimized the number of input nodes and hidden layers. Curry and Morgan [69] proposed a modified feedforward neural network which is pruned and optimised by means of Differential Evolution for seasonal data. Huang and Du [70] use particle swarm optimization to prune the radial basis probabilistic neural networks. Masutti and Castro [71] combined characteristics from self-organizing networks and artificial immune systems to solve the traveling salesman problem and pruned neurons which are not related to a city. Furthermore, numerous works have been done to perform EAs and pruning methods separately or simultaneously. Kaylani *et al.* [72] incorporated prune operator into a genetic algorithm as a mutation operator to design ARTMAP architecture for classification problems. Goh *et al.* [73] developed a hybrid multiobjective evolutionary approach for adaptation of NNs structures and a geometrical approach in identifying hidden neurons to prune for classification problems. Hervás-Martínez *et al.* [74] applied an evolutionary algorithm to design the structure and weights of a product-unit neural network, and finally used a backward stepwise procedure to prune variables sequentially until no further pruning can be made to improve the fit. However, most encoding schemes must predefine the chromosome length, which is problem-dependent. This user-defined length may affect the flexibility of problem representation and EA efficiency [75]-[77].

EAs are generally global search algorithms which explore the search space stochastically by a number of heuristics while gradient descent methods are local search algorithms which solve the problem with priori known derivatives information. A global search algorithm can exhibit good exploration ability while a local search algorithm can

show good exploitation performance. With an appropriate coordination of global search and local search, it has been shown that collaboration between global search and local search performs better than pure population-based global search algorithms or stand-alone local search algorithms [78]. In order to maintain a proper balance between global explorations and local exploitations, it is better to execute exploration and exploitation operations alternatively during evolution [79]. Nevertheless, to establish a subtle coordination of global and local search algorithms and determine how long should local search be run [80] are not easy and under investigation.

According to the aforementioned vantages of constructive, pruning, and evolutionary algorithms, this dissertation proposes two novel approaches to design NNs: symbiotic structure learning algorithm (SSLA) and evolutionary constructive and pruning algorithm (ECPA). SSLA, a neuron-based structure learning, attempts to design the NN structure of FNAGM according to symbiotic evolution. It evolves neuron population by fitness-sharing algorithm and constructs cascade NNs via neurons with different activation functions. Then the evolved FNAGM can accurately predict time series and further improve prediction error through the on-line batch training. On the other hand, ECPA, a network-based structure learning, directs the evolution of the NN topology using constructive and pruning methods in an evolutionary manner. It increases the complexity of NN by constructive method and prunes insignificant neurons on a probability basis to avoid the exponential growth of NN structure. Furthermore, the algorithm deletes old NNs with possibly complex structures and inserts newborn NNs with simple structures. In brief, ECPA integrates constructive, pruning, and evolutionary algorithms in an attempt to efficiently evolve compact NNs.

1.3 Organization of Dissertation

The objective of this dissertation is to develop evolutionary structure learning algorithms of NNs for prediction purpose. Organization and objective of each chapter are as follows.

In Chapter 2, FNAGM is proposed for real-time prediction. FNAGM integrates an GM(1,1) and an NN where GM(1,1) is used to predict the signal and NN is adopted to learn the prediction error of GM(1,1). Furthermore, an on-line batch training, an on-line parameter learning algorithm, is proposed to adjust the weights of NN in FNAGM on-line. Thus, FNAGM could simultaneously achieve prediction and on-line parameter learning.

In Chapter 3, a neuron-based structure learning approach, SSLA, is proposed to evolve the structure of FNAGM, i.e., the number of hidden neurons and the topology of NN. The idea behind SSLA is to evolve neuron population and construct NN from the neuron population where each neuron shares the fitness from the participating NN. SSLA performs neuron crossover and mutation to the neuron population and finally evolves appropriate structure of FNAGM. The evolved FNAGM could be applied to predict the signal and further learn the prediction error by on-line batch training.

In Chapter 4, a network-based structure learning approach, ECPA, is proposed to design compact structure of NN for prediction. In ECPA, a variable-length chromosome representation is adopted to describe NNs with different architectures. Thus, it is not necessary to predefine the length of the chromosome, and this makes the use of memory more efficient. Furthermore, ECPA introduces the concept of constructive method into the crossover and mutation operations in a manner that allows the initial structure of the NN to be simply set as a minimal network containing one hidden neuron with a single

connection to one input. The crossover and mutation operations then enlarge the architecture by adding hidden neurons and connections. ECPA then prunes the resulting NNs via a newly developed scheme consisting of cluster-based pruning (CBP) and age-based survival selection (ABSS).

Chapter 5 concludes this dissertation with discussion and suggestions for future work.



Chapter 2 On-Line Parameter Learning for Prediction

In this chapter, a feedforward-neural-network-aided grey model (FNAGM) and its corresponding on-line parameter learning algorithm, on-line batch training algorithm, is presented for prediction. FNAGM, which integrates a first-order single variable grey model (GM(1,1)) and a neural network (NN), is designed to not only predict the signals but also continually adapts itself to the dynamical change. The system process consists of three phases: initialization phase, GM(1,1) prediction phase and FNAGM prediction phase. First, some parameters required in FNAGM are chosen in the initialization phase. Then, a one-step-ahead predictive value is generated in the GM(1,1) prediction phase. Finally, an NN is used to learn the prediction error of GM(1,1) and compensate it in the FNAGM prediction phase. Significantly, an on-line batch training is proposed to adjust the weights of NN according to Levenberg-Marquardt algorithm in real-time.

2.1 Feedforward-Neural-Network-Aided Grey Model

This section first describes NNs and GM(1,1), and then presents the proposed FNAGM which combines a GM(1,1) and an NN.

2.1.1 Neural Networks

A feedforward NN processes input vector with one direction, forward, to hidden layer and then output layer. For forecasting purposes, the theoretical work shows that a

single hidden layer is sufficient [81]. Thus, a general feedforward NN with m input neurons, p hidden neurons and one output neuron is applied for one-step-ahead prediction. Let's define the input vector and weights first. For time series prediction, the input vector of the network at time k is $\mathbf{u}[k] = [u_1[k] \ u_2[k] \ \dots \ u_m[k]]^T$. Since the on-line batch training which will be described in Section 2.2 is adopted as the learning strategy, the time index k of the weights is important and denotes that the weights have been updated through the learning iteration at time $k-1$ and are applied for prediction at time k . In the hidden layer, $\mathbf{w}_j[k]$ is the weight vector from the input vector to the j th hidden neuron. In the output layer, $\mathbf{w}_o[k]$ is the weight vector from the hidden neurons to the output neuron. The input vector $\mathbf{u}[k]$ is linearly combined at the hidden neuron and then processed by the activation function $g(\cdot)$ which can be one of the continuous neuron models, e.g., logistic, hyperbolic tangent, linear threshold, exponential and Gaussian signal function [82]. For the j th hidden neuron, the output is

$$h_j[k] = g\left(\sum_{i=1}^m w_{ji}[k] \cdot u_i[k] + w_{jb}[k]\right) \quad (2.1)$$

where $w_{ji}[k]$ and $w_{jb}[k]$ are the components of $\mathbf{w}_j[k]$ and $\mathbf{w}_j[k] = [w_{j1}[k] \ w_{j2}[k] \ \dots \ w_{jm}[k] \ w_{jb}[k]]^T$. The output of the network is

$$y[k] = \sum_{j=1}^p w_{oj}[k] \cdot h_j[k] + w_{ob}[k] \quad (2.2)$$

where $w_{oj}[k]$ and $w_{ob}[k]$ are the components of $\mathbf{w}_o[k]$ and $\mathbf{w}_o[k] = [w_{o1}[k] \ w_{o2}[k] \ \dots \ w_{op}[k] \ w_{ob}[k]]^T$. From (2.1) and (2.2), the input-output relationship of NN could be further represented as

$$y[k] = f(\mathbf{v}[k], \mathbf{u}[k]) \quad (2.3)$$

where

$$\begin{aligned}\mathbf{v}[k] &= [\mathbf{w}_1^T[k] \ \mathbf{w}_2^T[k] \ \dots \ \mathbf{w}_p^T[k] \ \mathbf{w}_o^T[k]]^T \\ &= [v_1[k] \ v_2[k] \ \dots \ v_q[k]]^T\end{aligned}\quad (2.4)$$

where $q = (m+2) \cdot p + 1$. To train the network, the proposed on-line batch training algorithm is adopted to update the weight vector $\mathbf{v}[k]$ and will be described in Section 2.2.

2.1.2 First-Order Single Variable Grey Model

Consider a discrete data sequence of length $n \geq 4$ formed as the following column vector

$$\mathbf{x}^{(0)} = [x^{(0)}[1] \ x^{(0)}[2] \ \dots \ x^{(0)}[n]]^T \quad (2.5)$$

where each element has the same numeric sign. In general, GM(1,1) adopts three fundamental operations, given as

Accumulate Generating Operation (AGO):

$$x^{(1)}[k] = \sum_{l=1}^k x^{(0)}[l], \quad k = 1, 2, \dots, n \quad (2.6)$$

Mean Generating Operation (MGO):

$$z^{(1)}[k] = \alpha x^{(1)}[k] + (1 - \alpha)x^{(1)}[k - 1], \quad k = 2, 3, \dots, n \quad (2.7)$$

Inverse Accumulate Generating Operation (IAGO):

$$x^{(0)}[k] = x^{(1)}[k] - x^{(1)}[k - 1], \quad k = 2, 3, \dots, n \quad (2.8)$$

where α is often set as 0.5. According to GM(1,1) [3], its grey differential equation is presented as

$$x^{(0)}[k] + az^{(1)}[k] = b, \quad k = 1, 2, \dots, n \quad (2.9)$$

where a is the development coefficient and b is the grey input. Both a and b are

unknown and have to be determined first by rearranging (2.9) into the following matrix form

$$\mathbf{x}^{(0)} = \mathbf{X}^{(1)} \begin{bmatrix} a \\ b \end{bmatrix} \quad (2.10)$$

where

$$\mathbf{X}^{(1)} = \begin{bmatrix} -\frac{x^{(1)}[1]}{2} & -\frac{x^{(1)}[1]-x^{(1)}[1]}{2} & \dots & -\frac{x^{(1)}[n]-x^{(1)}[n-1]}{2} \\ 1 & 1 & \dots & 1 \end{bmatrix}^T \quad (2.11)$$

Then a and b could be solved by the least square method as below

$$\begin{bmatrix} a \\ b \end{bmatrix} = \left(\mathbf{X}^{(1)T} \mathbf{X}^{(1)} \right)^{-1} \mathbf{X}^{(1)T} \mathbf{x}^{(0)} \quad (2.12)$$

Based on GM(1,1), the solution of the grey first-order differential equation (2.9) is estimated as

$$\hat{x}^{(1)}[n+1] = \left(x^{(0)}[1] - b/a \right) e^{-an} + b/a \quad (2.13)$$

Further applying IAGO in (2.8) to (2.13) yields

$$\hat{x}^{(0)}[n+1] = \left(x^{(0)}[1] - b/a \right) e^{-an} (1 - e^a) \quad (2.14)$$

which is the so-called one-step-ahead predictive value after the original data sequence $\mathbf{x}^{(0)}$.

Since GM(1,1) performs the prediction via the data sequence with the same numeric sign, the preprocess is employed to transform the raw data in (2.5) into

$$x'^{(0)}[l] = x^{(0)}[l] - \min(\mathbf{x}^{(0)}) + \gamma, \quad l = 1, 2, \dots, n \quad (2.15)$$

where γ is a constant bias to avoid the output to be zero. Then the transformed value is used to estimate $\hat{x}'^{(0)}[n+1]$ by (2.6)–(2.14). As a result, the one-step-ahead predictive value is determined as

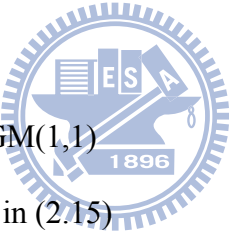
$$\hat{x}^{(0)}[n+1] = \hat{x}'^{(0)}[n+1] + \min(\mathbf{x}^{(0)}) - \gamma \quad (2.16)$$

Therefore, GM(1,1) can process the historical data sequence with different numeric signs.

2.1.3 Structure of FNAGM

This subsection presents an intelligent forecasting system, FNAGM, which is constructed by an NN and a GM(1,1). FNAGM consists of three phases: initialization phase, GM(1,1) prediction phase and FNAGM prediction phase. With FNAGM, the prediction error of GM(1,1) is further improved by the use of on-line batch training in the FNAGM prediction phase.

For the initialization phase, some parameters used in FNAGM are first defined as below:

N_s	the total step size	
n	the input length of GM(1,1)	
γ	the bias of GM(1,1) in (2.15)	
m	the input number of NN	
p	the number of hidden neurons of NN	
N	the maximum size of batch training pattern	
$\mathbf{v}[n+m]$	the initial weight vector of NN	
$\mu[n+m+1]$	the initially positive scalar for updating the weight vector	
β	the constant to adjust $\mu[k]$	

Note that $\mathbf{v}[n+m]$ is randomly chosen. Then, obtain $\{x^{(0)}[1], x^{(0)}[2], \dots, x^{(0)}[n]\}$ and get into the GM(1,1) prediction phase.

The GM(1,1) prediction phase is executed from $k = n$ to $k = n+m-1$ and generates the one-step-ahead predictive value $\hat{x}^{(0)}[k+1]$. For each step, the prediction error is

obtained as $e_{GM}[k+1] = x^{(0)}[k+1] - \hat{x}^{(0)}[k+1]$. This phase will be repeated to determine m prediction errors.

Based on these m prediction errors, the FNAGM prediction phase starts to further improve the prediction error for $k \geq n+m$. The configuration of the FNAGM prediction is depicted in Fig. 2.1. GM(1,1) calculates the one-step-ahead predictive value, $\hat{x}^{(0)}[k+1]$, and NN simultaneously adopts the past m prediction errors as the input vector to learn the prediction error $e_{GM}[k+1]$ on-line, estimated as $\hat{e}_{GM}[k+1]$. Then, the final one-step-ahead predictive value, $\hat{x}_{FNAGM}[k+1]$, is set to be the sum of $\hat{x}^{(0)}[k+1]$ and $\hat{e}_{GM}[k+1]$, which is better than $\hat{x}^{(0)}[k+1]$.

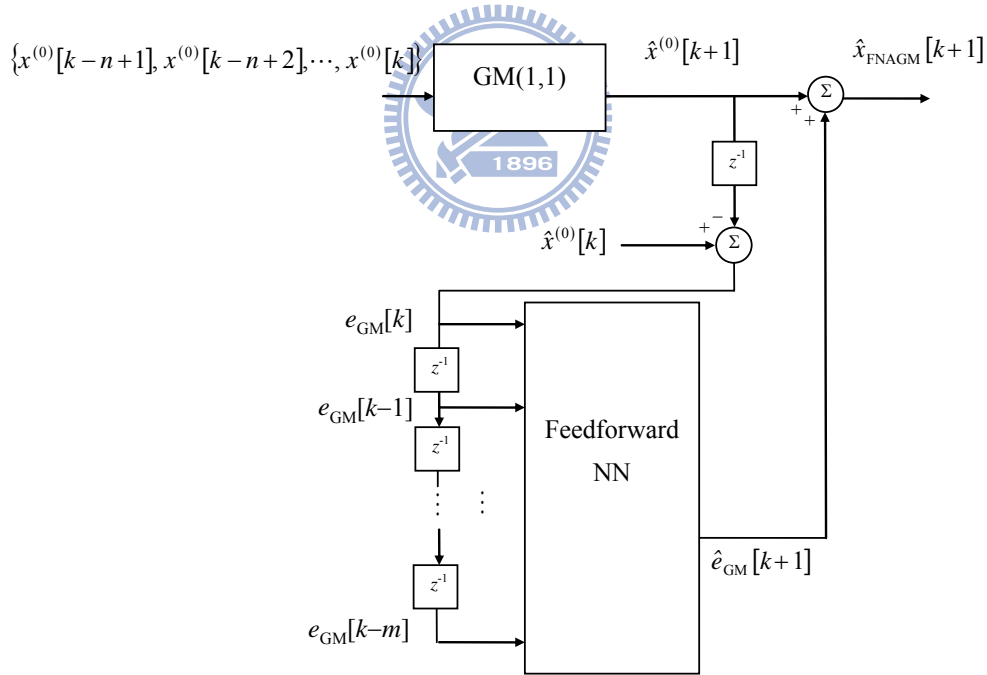


Fig. 2.1 Feedforward-neural-network-aided grey model.

2.2 On-Line Parameter Learning of FNAGM

This section presents the on-line parameter learning algorithm, called on-line batch training, which is used to continuously adapt FNAGM to the dynamical change of the

signal. The idea behind on-line batch training is to perform on-line training by more than one pattern. When a new data point is observed, FNAGM collects a batch of recently obtained training patterns and then performs on-line batch training by the batch training patterns at each time step.

2.2.1 On-Line Batch Training

The training process in the FNAGM prediction phase employs the on-line batch training to adjust NN because of its higher accuracy and less computation time. Let the batch training pattern at step k contain r recently observed training patterns denoted as

$$\mathbf{B}_k = \{\mathbf{P}_{k-r+1}, \mathbf{P}_{k-r+2}, \dots, \mathbf{P}_k\} \quad \text{for } k > n+m \quad (2.17)$$

where $r = \min\{k-(n+m), N\}$, $N > 1$ and $\mathbf{P}_j = \{\mathbf{u}[j], e_{\text{GM}}[j]\}$ is the j th training pattern related to the input vector $\mathbf{u}[j] = [e_{\text{GM}}[j-m], e_{\text{GM}}[j-m+1], \dots, e_{\text{GM}}[j-1]]^T$ and the target $e_{\text{GM}}[j]$. Note that the total number r of the batch training pattern \mathbf{B}_k is fixed and equal to N for $k \geq n+m+N$.

The on-line batch training modifies Levenberg-Marquardt algorithm [83] to update the weight vector $\mathbf{v}[k]$ for $k \geq n+m+1$, which is

$$\mathbf{v}[k+1] = \mathbf{v}[k] - [\mathbf{G}^T[k]\mathbf{G}[k] + \mu[k]\mathbf{I}]^{-1} \mathbf{G}^T[k]\boldsymbol{\varepsilon}[k] \quad (2.18)$$

where $\boldsymbol{\varepsilon}[k]$ is network error vector, $\mathbf{G}[k]$ is the Jacobian matrix and $\mu[k]$ is a positive scalar parameter. The j th component of the network error vector $\boldsymbol{\varepsilon}[k]$ is obtained as $\varepsilon_j[k] = e_{\text{GM}}[j] - f(\mathbf{v}[k], \mathbf{u}[j])$, $j = k-r+1, k-r+2, \dots, k$, corresponding to \mathbf{P}_j . After $\boldsymbol{\varepsilon}[k]$ is determined, the Jacobian matrix is calculated as

$$\mathbf{G}[k] = \frac{\partial \boldsymbol{\varepsilon}[k]}{\partial \mathbf{v}[k]} = \begin{bmatrix} \frac{\partial \varepsilon_{k-r+1}[k]}{\partial v_1[k]} & \frac{\partial \varepsilon_{k-r+1}[k]}{\partial v_2[k]} & \dots & \frac{\partial \varepsilon_{k-r+1}[k]}{\partial v_q[k]} \\ \frac{\partial \varepsilon_{k-r+2}[k]}{\partial v_1[k]} & \frac{\partial \varepsilon_{k-r+2}[k]}{\partial v_2[k]} & \dots & \frac{\partial \varepsilon_{k-r+2}[k]}{\partial v_q[k]} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_k[k]}{\partial v_1[k]} & \frac{\partial \varepsilon_k[k]}{\partial v_2[k]} & \dots & \frac{\partial \varepsilon_k[k]}{\partial v_q[k]} \end{bmatrix} \quad (2.19)$$

Note that the j th row is determined according to \mathbf{P}_j and the c th column is related to v_c . The on-line batch training adjusts $\mu[k]$ by comparing the previous and current errors based on $\varepsilon_j[k-1]$ and $\varepsilon_j[k]$, as shown by

$$\mu[k+1] = \begin{cases} \mu[k]/\beta, & \text{if } \sum_{j=k-s}^{k-1} \varepsilon_j^2[k] < \sum_{j=k-s}^{k-1} \varepsilon_j^2[k-1] \\ \mu[k] \cdot \beta, & \text{if } \sum_{j=k-s}^{k-1} \varepsilon_j^2[k] \geq \sum_{j=k-s}^{k-1} \varepsilon_j^2[k-1] \end{cases} \quad (2.20)$$

where $\beta > 1$, $s = \min\{N-1, k-n-m-1\}$ and $k \geq n+m+2$. Note that $\varepsilon_j[k-1]$ and $\varepsilon_j[k]$ are calculated via the same training pattern \mathbf{P}_j . Clearly, when the current error is decreased then $\mu[k]$ is reduced by β , otherwise $\mu[k]$ is increased. As a result, implementation of the on-line batch training requires past observations with maximum memory size N to update $\mathbf{v}[k]$ in batch mode and obtain more accurate solution at each time step.

The above intelligent forecasting system, FNAGM, is summarized as the following stepwise procedure:

Initialization phase:

- S1 Choose the total step size: N_s ;
- the input length of GM(1,1): n ;
- the bias of GM(1,1): γ ;
- the input number of NN: m ;
- the number of hidden neurons of NN: p ;
- the maximum size of batch training pattern: N ;
- the initial positive scalar: $\mu[n+m+1]$;

the constant to adjust $\mu[k]$: β ;

the initial weight vector: $\mathbf{v}[n+m]$ (randomly);

S2 Obtain $\{x^{(0)}[1], x^{(0)}[2], \dots, x^{(0)}[n]\}$ and set $k = n$.

GM(1,1) prediction phase for $n \leq k < n+m$:

S3 Compute $\hat{x}^{(0)}[k+1]$ by GM(1,1) in (2.6)–(2.16).

S4 Set $k = k+1$.

S5 Obtain $x^{(0)}[k]$ and then $e_{GM}[k]$.

S6 If $k < n+m$, then go to S3.

FNAGM prediction phase for $k \geq n+m$:

S7 Compute $\hat{x}^{(0)}[k+1]$ by GM(1,1) in (2.6)–(2.16).

S8 Calculate $\hat{e}_{GM}[k+1]$ by NN in (2.1), (2.2).

S9 Set the one-step-ahead predictive value of FNAGM as

$$\hat{x}_{FNAGM}[k+1] = \hat{x}^{(0)}[k+1] + \hat{e}_{GM}[k+1].$$

S10 Set $k = k+1$.

S11 Obtain $x^{(0)}[k]$ and then $e_{GM}[k]$.

S12 If $k < n+m+1$, then $\mathbf{v}[k+1] = \mathbf{v}[k]$ and go to S7.

On-line batch training (from S13 to S17)

S13 Construct $\mathbf{B}_k = \{\mathbf{P}_{k-r+1}, \mathbf{P}_{k-r+2}, \dots, \mathbf{P}_k\}$ in (2.17) where

$$r = \min\{k - (n+m), N\}.$$

S14 Compute the network error based on \mathbf{B}_k

for $j = k-r+1$ to k

$$\varepsilon_j[k] = e_{GM}[j] - f(\mathbf{v}[k], \mathbf{u}[j]);$$

end for

S15 Update $\mathbf{v}[k]$ by Levenberg-Marquardt algorithm in (2.18), (2.19).

S16 If $k < n + m + 2$, then $\mu[k+1] = \mu[k]$ and go to S7.

S17 Update $\mu[k]$ by (2.20).

S18 If $k < N_s$, then go to S7;

else stop.

With the above procedure, FNAGM can gradually learn to predict signal via the on-line batch training. The success of the prediction will be demonstrated in the Section 2.3 and 2.4.

2.2.2 Convergence Analysis

To discuss the convergence issue of FNAGM, the error difference could be approximately represented as [84]

$$\begin{aligned}\boldsymbol{\varepsilon}[k+1] &= \boldsymbol{\varepsilon}[k] + \Delta\boldsymbol{\varepsilon}[k] \\ &\cong \boldsymbol{\varepsilon}[k] + \frac{\partial\boldsymbol{\varepsilon}[k]}{\partial\mathbf{v}[k]} \Delta\mathbf{v}[k]\end{aligned}\quad (2.21)$$

where $\Delta\boldsymbol{\varepsilon}[k]$ and $\Delta\mathbf{v}[k]$ represent the error change and the weight vector change respectively. Note that (2.21) is the first order Taylor approximation which neglects the higher order terms. Equations (2.18) and (2.21) yield

$$\boldsymbol{\varepsilon}[k+1] = \boldsymbol{\varepsilon}[k] - \mathbf{G}[k](\mathbf{G}^T[k]\mathbf{G}[k] + \mu[k]\mathbf{I})^{-1} \mathbf{G}^T[k]\boldsymbol{\varepsilon}[k] \quad (2.22)$$

Then,

$$\begin{aligned}\|\boldsymbol{\varepsilon}[k+1]\| &= \left\| \boldsymbol{\varepsilon}[k] - \mathbf{G}[k](\mathbf{G}^T[k]\mathbf{G}[k] + \mu[k]\mathbf{I})^{-1} \mathbf{G}^T[k]\boldsymbol{\varepsilon}[k] \right\| \\ &= \left\| (\mathbf{I} - \mathbf{G}[k](\mathbf{G}^T[k]\mathbf{G}[k] + \mu[k]\mathbf{I})^{-1} \mathbf{G}^T[k]) \cdot \boldsymbol{\varepsilon}[k] \right\| \\ &\leq \left\| \mathbf{I} - \mathbf{G}[k](\mathbf{G}^T[k]\mathbf{G}[k] + \mu[k]\mathbf{I})^{-1} \mathbf{G}^T[k] \right\| \cdot \|\boldsymbol{\varepsilon}[k]\|\end{aligned}\quad (2.23)$$

where $\|\cdot\|$ denotes the Euclidean norm. Moreover, let g_j be the j th singular value of $\mathbf{G}[k]$

and assume that $\mathbf{G}[k]$ has full row rank, i.e., a \mathbf{B}_k with sufficient r is obtained. Thus [85]

$$\begin{aligned} \left\| \mathbf{I} - \mathbf{G}[k](\mathbf{G}^T[k]\mathbf{G}[k] + \mu[k]\mathbf{I})^{-1} \mathbf{G}^T[k] \right\| &\leq \max \left(1 - \frac{g_j^2[k]}{g_j^2[k] + \mu[k]} \right) \\ &\leq \max \frac{\mu[k]}{g_j^2[k] + \mu[k]} = 1 \end{aligned} \quad (2.24)$$

which implies that $\|\varepsilon[k]\|$ is monotonically decreasing. That means the output error between the prediction error of GM(1,1) and the output of the neural network converges to zero as $k \rightarrow \infty$. This fact completes the proof of the convergence.

2.3 Numerical Results

To verify the performance of the proposed FNAGM, two numerical examples are adopted for demonstration. The first example involves predicting an external disturbance that has been described in [27] and the second example involves predicting the Mackey-Glass chaotic time series [86]. The numerical simulations were executed in an Intel Pentium CPU at 1.5 GHz with 512 MBytes RAM.

2.3.1 Example 1: Disturbance Prediction

The disturbance to be predicted is

$$x[k] = 0.1 \cdot \cos(4kT) + 0.05 \cdot \sin(9.2kT) \quad (2.25)$$

where T is the sampling time and set as 0.05 second. In the step S1 of the initialization phase, the proposed FNAGM chooses $N_s = 200$, $n = 4$, $\gamma = 2$, $p = 4$, $\mu[n+m+1] = 0.001$ and $\beta = 4/3$. As for the weight vector $\mathbf{v}[n+m]$, it was randomly generated according to a normal distribution with zero mean and unit variance by the program. Note that the objective of this example is to predict the disturbance in real-time, so it is required to accomplish the prediction phase with sufficient accuracy during one sampling interval

by selecting an appropriate N , which was set to be 40 in the simulation.

First, let's determine a suitable input number m of NN of FNAGM by 100 independent runs of $m = 2, 3, 4$ and 5. The root mean square error (RMSE) calculated for $k > 100$ of each run is recorded. Table 2.1 shows the statistical means over these 100 runs of different input numbers. It is clear that the larger m achieved the better prediction accuracy. For instance, the prediction errors were reduced from 10^{-3} to 10^{-4} when m was changed from 2 to 3. However, the prediction errors were only improved a little from 4.29×10^{-5} of $m = 4$ to 4.05×10^{-5} of $m = 5$. Hence, the input number $m = 4$ was adopted for FNAGM in the following simulations. Several approaches exist in the literature, however, it is not feasible and necessary to perform an exhaustive comparison with all algorithms. The aim of our experimental comparison is to realize the advantage and disadvantage of the aid of NN and on-line batch training. Therefore, GM(1,1) [3] and Advanced GM(1,1) [27] are mainly considered for comparison.

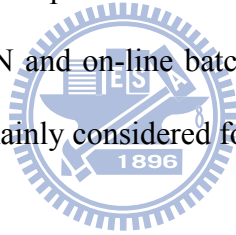


Table 2.1 Comparison of different input numbers of NN of FNAGM in Example 1.

Model	FNAGM $m = 2$	FNAGM $m = 3$	FNAGM $m = 4$	FNAGM $m = 5$
Statistical mean of RMSE over 100 runs	3.40×10^{-3}	6.16×10^{-4}	4.29×10^{-5}	4.05×10^{-5}
Computation time per prediction step	4.60×10^{-3}	4.70×10^{-3}	4.80×10^{-3}	4.90×10^{-3}

Based on the on-line batch training, FNAGM was applied to predict the disturbance (2.25) with $N = 40$ and $m = 4$. The statistical mean of RMSE of FNAGM over 100 runs for $k > 100$ is shown in Table 2.2, which also includes RMSEs of GM(1,1), Advanced GM(1,1), and NN for $k > 100$. The NN was trained by on-line learning and its number of hidden neurons was designed through a trial-and-error

method. As expected, Advanced GM(1,1) based on the Lagrange polynomial of third order improved the prediction error of GM(1,1) from 10^{-2} to 10^{-3} . Most significantly, the proposed FNAGM can achieve much better result than Advanced GM(1,1), highly reducing the prediction error from 10^{-3} to 10^{-5} .

In addition, Table 2.2 shows the computation time required for the prediction phase of FNAGM and the computation times per prediction step of GM(1,1) and Advanced GM(1,1). Although FNAGM requires longer computation time than the other two methods, increased from 10^{-4} to 10^{-3} second, FNAGM is still able to complete the prediction phase during one sampling interval. The proposed intelligent forecasting system FNAGM is, therefore, applicable for real-time prediction.

Table 2.2 Comparison of the prediction errors of GM(1,1), Advanced GM(1,1), NN and FNAGM in Example 1.

Model	GM(1,1)	Advanced GM(1,1)	NN	FNAGM $m = 4$
Statistical mean of RMSE over 100 runs	---	---	0.9883	4.29×10^{-5}
RMSE for $k > 100$ in Fig. 2.2	1.27×10^{-2}	1.60×10^{-3}	0.9996	3.59×10^{-5}
Computation time per prediction step	4.08×10^{-4}	4.15×10^{-4}	4.80×10^{-4}	4.80×10^{-3}

To further demonstrate the performance, Fig. 2.2 shows the prediction results of FNAGM, GM(1,1) and Advanced GM(1,1). Note that the curve of FNAGM is one of the 100 runs and it is randomly chosen from them. Evidently, after sufficient training iterations, i.e., $k > 60$ in this case, FNAGM learned the prediction error efficiently and obtains much better result than Advanced GM(1,1).

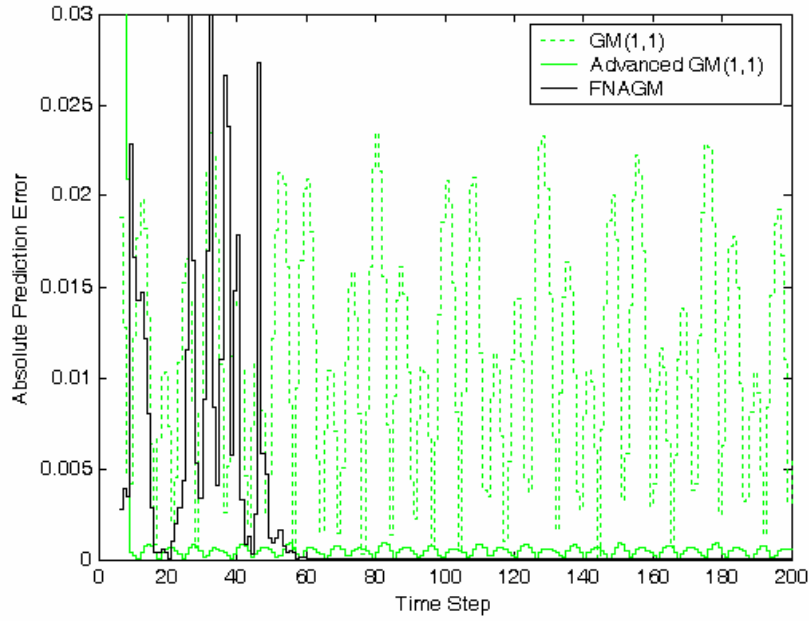


Fig. 2.2 Absolute prediction errors of GM(1,1), Advanced GM(1,1) and FNAGM with $m = 4$ in Example 1.

2.3.2 Example 2: Chaotic Time Series Prediction

The Mackey-Glass time series is generated from the following delay differential equation

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \quad (2.26)$$

where $\tau = 17$ and $x(0) = 1.2$ in the simulation. The data points were obtained based on the fourth-order Runge-Kutta method with sampling interval 0.1 second. In the initialization phase, all the parameters are the same as Example 1 except $N = 80$, i.e., $r = \min\{k-8, 80\}$, due to the use of sampling interval 0.1 second longer than 0.05 second in Example 1. The input number of NN of FNAGM was still selected as $m = 4$ by the same reason as Example 1.

The on-line batch training of FNAGM was performed for 100 independent runs. The RMSE for $k > 250$ of each run is recorded. The statistical mean over these 100 runs

are given in Table 2.3. From the performance comparison, Advanced GM(1,1) indeed improved the prediction error of GM(1,1) and the proposed intelligent forecasting system FNAGM achieved better performance than Advanced GM(1,1), where the prediction error was reduced from 10^{-3} to 10^{-4} . Furthermore, although FNAGM with $N = 80$ takes computation time 9.90×10^{-3} second in the prediction phase, which is longer than 4.80×10^{-3} second with $N = 40$, FNAGM is still an effective forecasting system for real-time prediction.

To demonstrate the performance of FNAGM, Fig. 2.3 shows that FNAGM outperforms GM(1,1) and Advanced GM(1,1) in prediction accuracy for $k > 250$. FNAGM indeed improved the prediction error of the time series which has chaotic, nonperiodic and nonconvergence natures.

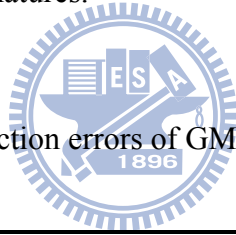


Table 2.3 Comparison of the prediction errors of GM(1,1), Advanced GM(1,1) and FNAGM in Example 2.

Model	GM(1,1)	Advanced GM(1,1)	NN	FNAGM $m = 4$
Statistical mean of RMSE over 100 runs	---	---	1.0269	6.08×10^{-4}
RMSE for $k > 250$ in Fig. 2.3	1.22×10^{-2}	1.30×10^{-3}	1.0336	5.39×10^{-4}
Computation time per prediction step	4.08×10^{-4}	4.15×10^{-4}	4.80×10^{-4}	9.90×10^{-3}

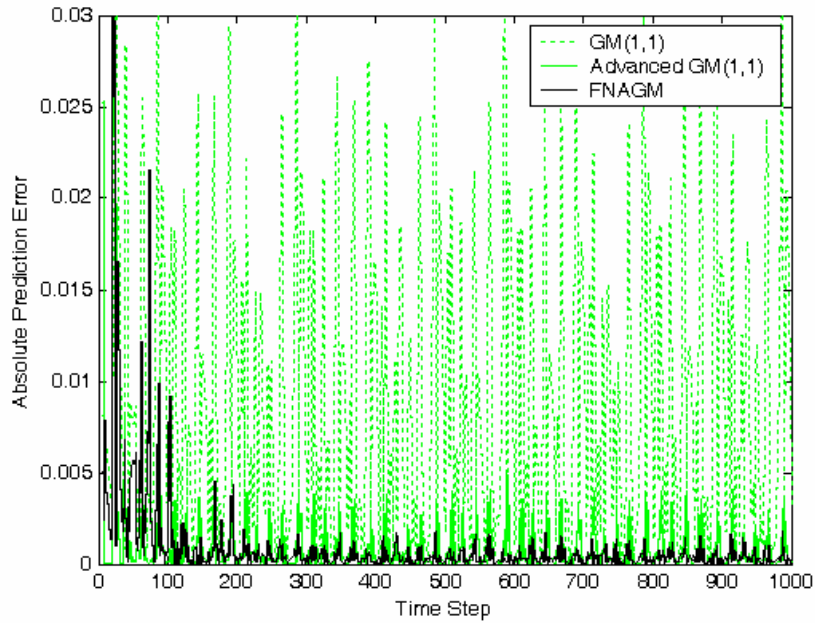


Fig. 2.3 Absolute prediction errors of GM(1,1), Advanced GM(1,1) and FNAGM with $m = 4$ in Example 2.

2.4 Experimental Results

The following experiment was conducted to verify the performance of the proposed FNAGM. The experiment involved predicting the trajectory of a moving object as performed by a binocular robot, called an Eye-Robot [87], [88]. The Eye-Robot shown in Fig. 2.4(a) was built using a parallel-axis camera with five motors to emulate the eye movement of humans. We adopted five FAULHABER DC servomotors to perform the panning movement of the eyes, the conjugated tilt movement of the eyes, and the pan and tilt movements of the head via an RS-232 interface. The range of panning is ± 120 degrees, while the range tilting is ± 60 degrees. The DC servomotors were controlled using a motion control card, MCDC 3006S, at a positioning resolution of 0.18° . The size of the Eye-Robot is $25 \times 25 \times 30 \text{ cm}^3$. The experiment was executed using an Intel Pentium CPU at 1.5 GHz with 512 MBytes RAM.

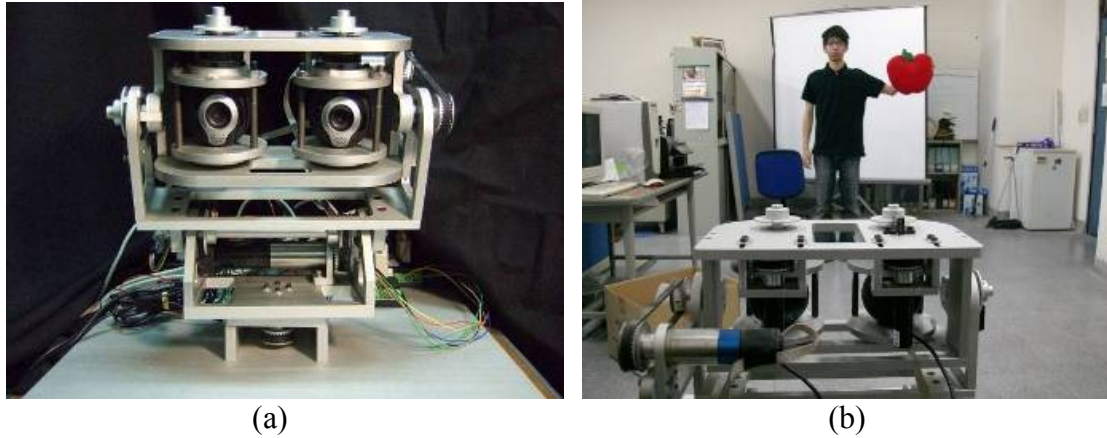


Fig. 2.4 (a) Eye-Robot. (b) Experimental environment where a participant held a red object.

2.4.1 Trajectory Prediction

The experiment was performed in an indoor environment, as shown in Fig. 2.4(b). The Eye-Robot captured 30 frames per second, for a sampling time of approximately 0.03 second. Once an image was obtained, Eye-Robot first extracted red object in the RGB color space, and then determined the center of gravity of the object. The trajectory of the object was further divided into x and y axes; therefore, one FNAGM was employed to make predictions for each of the two axes. Note that this experiment only performed a prediction of the trajectory, without controlling the motors to track the object.

In step S1 of the initialization phase, the proposed FNAGM selected $N_s = 500$, $n = 4$, $m = 4$, $p = 4$, $\mu[n+m+1]=0.001$ and $\beta = 4/3$ by a preliminary test. Weight vectors $\mathbf{v}[n+m]$ were randomly generated according to normal distribution with zero mean and unit variance using the program. Note that the objective of this experiment is to predict the trajectory in real-time; therefore, it is necessary to accomplish the prediction phase with sufficient accuracy during a single sampling interval by selecting an appropriate N ,

which was set at 40 in the experiment.

Fig. 2.5 shows the trajectory of the object obtained by Eye-Robot, and we applied FNAGM to predict the trajectory of the object. Similar to Section 2.3, GM(1,1) and Advanced GM(1,1) were performed for comparison. The statistical mean of RMSE of FNAGM over 100 runs for $k > 100$ is shown in Table 2.4, together with RMSEs of GM(1,1) and Advanced GM(1,1) for $k > 100$ where the unit of RMSE is pixel. As a result, the proposed intelligent forecasting system FNAGM can achieve much better result than GM(1,1) and Advanced GM(1,1), highly reducing the prediction error from 3.65 to 1.66 and from 5.01 to 2.13 for $x(k)$ and $y(k)$, respectively.

In addition, Table 2.4 shows the computation time required for the prediction phase of FNAGM and the computation times per prediction step of GM(1,1) and Advanced GM(1,1). Although FNAGM requires longer computation time than the other two methods, increased from 10^{-4} to 10^{-3} second, FNAGM is still able to complete the prediction phase during one sampling interval, i.e., 3.3×10^{-2} . The proposed intelligent forecasting system is, therefore, applicable for real-time prediction.

Table 2.4 Comparison of prediction error and computation time.

Model	GM(1,1)	Advanced GM(1,1)	FNAGM
RMSE of $x(k)$	3.7428	3.6512	1.6608
RMSE of $y(k)$	5.6125	5.0112	2.1343
Computation time per prediction step	4.08×10^{-4}	4.15×10^{-4}	4.80×10^{-3}

To further demonstrate the performance of FNAGM, Fig. 2.6 and Fig. 2.7 show the prediction results of $x(k)$ and $y(k)$ for FNAGM, GM(1,1), and Advanced GM(1,1). Note that the curve of FNAGM is one run randomly selected from 100 runs. It appears that

after a sufficient number of training iterations, FNAGM obtained smaller prediction error than GM(1,1) and Advanced GM(1,1).

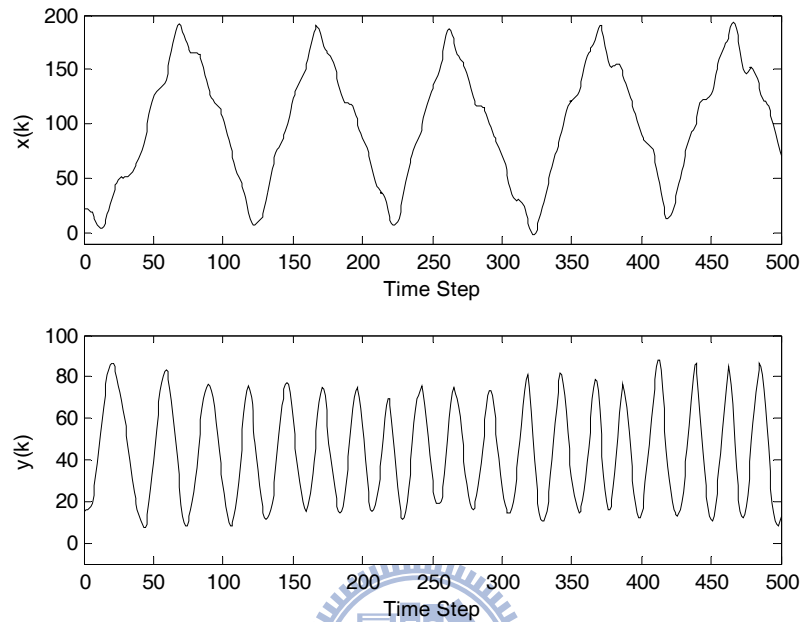


Fig. 2.5 Trajectory of the object.

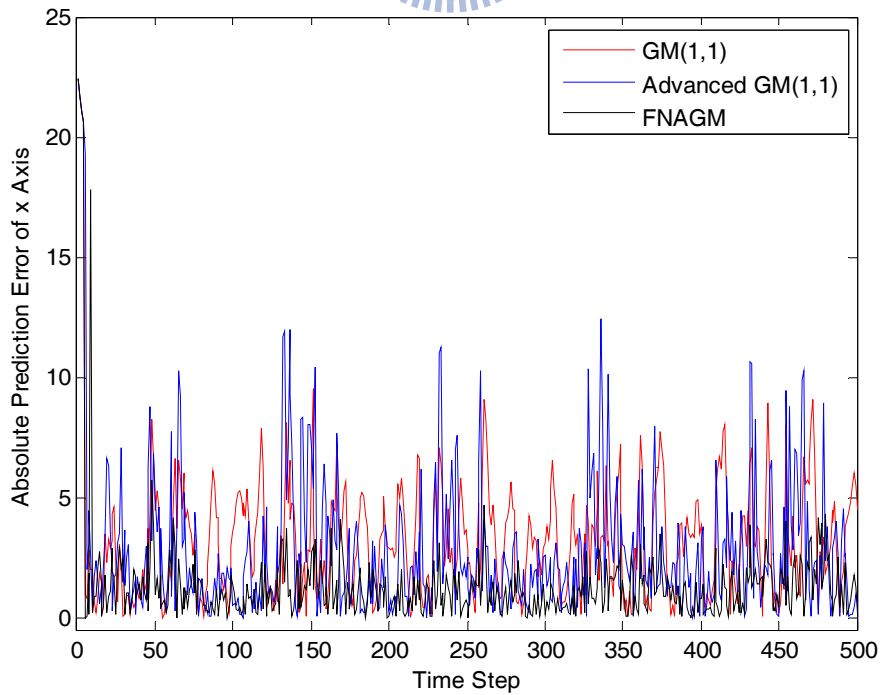


Fig. 2.6 Prediction error of $x(k)$ for GM(1,1), advance GM(1,1), and FNAGM.

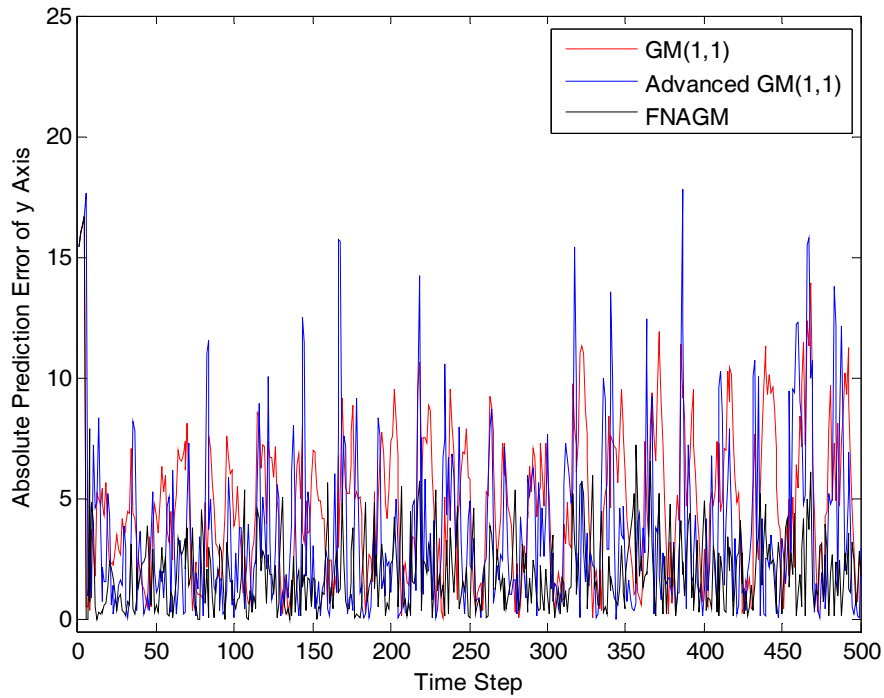


Fig. 2.7 Prediction error of $y(k)$ for GM(1,1), advance GM(1,1), and FNAGM.

2.4.2 Tracking Control

The purpose of this experiment is to control Eye-Robot to track a red object with the aid of FNAGM. The movement of the target in this experiment was manually achieved by the participant as shown in Fig. 2.4(b). To carry out fairly comparison, the movement of the target was performed as similar as possible in each run. The Eye-Robot employed the proportional-derivative (PD) controller for the tracking control. The PD controller first receives the tracking error which is the difference between the center of the image and the position of the object, and then controls the cameras to track the object such that the object could be located at the center of the image. In most control applications, the control signal is a function of the current and previous tracking errors. In this experiment, the predicted tracking error is used instead of current tracking error [89] to reduce tracking error. The setup of this experiment is precisely the same as that described in Section 2.4.1. Note that the purpose of this experiment is not to design

the PD controller, but to observe how FNAGM benefits the target tracking. For comparison, GM(1,1) and Advanced GM(1,1) were carried out to predict the tracking error of Eye-Robot.

Fig. 2.8-Fig. 2.10 show the tracking errors corresponding to GM(1,1), Advanced GM(1,1), and FNAGM, respectively. Note that the tracking error is equivalent to $x(k)$ whose unit is pixel in this case. From Fig. 2.8 and Fig. 2.9, it can be observed that GM(1,1) achieved a smooth tracking error while Advanced GM(1,1) did not always obtain smaller tracking error than GM(1,1). From Fig. 2.10, it can be seen that FNAGM achieved high tracking error for $k < 50$ and gradually obtained small tracking error for $k > 100$ through on-line batch learning. It shows the merit of FNAGM that the learning ability can adapt the intelligent forecasting system to dynamical changes and improve the prediction performance when comparing with GM(1,1) and Advanced GM(1,1). Most significantly, the proposed intelligent forecasting system FNAGM can carry out the on-line batch learning on the robotic application in real-time.

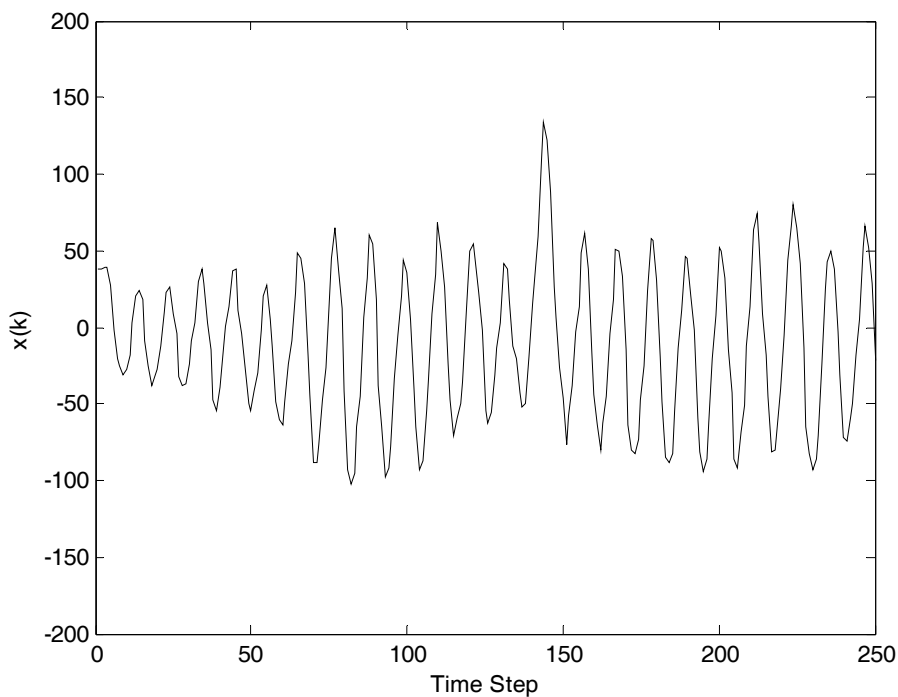


Fig. 2.8 Tracking error of $x(k)$ using GM(1,1).

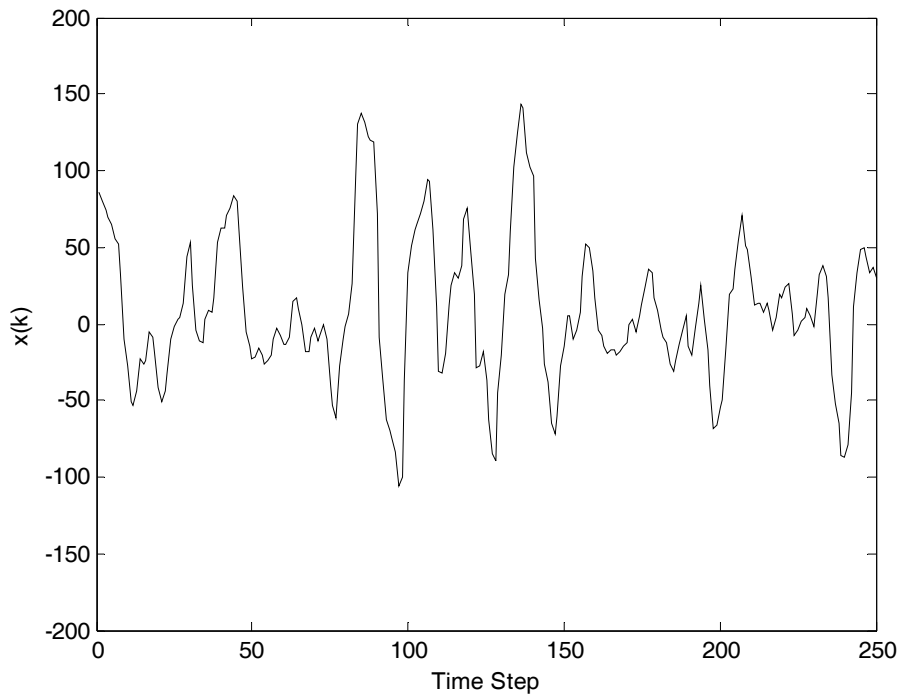


Fig. 2.9 Tracking error of $x(k)$ using Advanced GM(1,1).

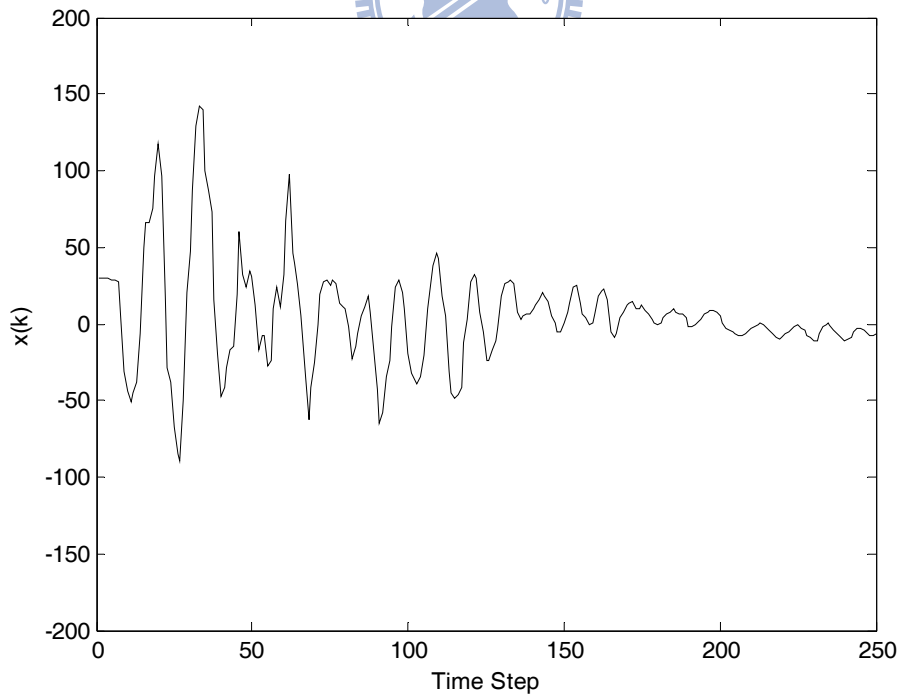
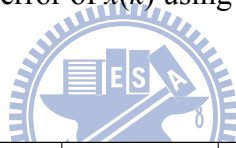
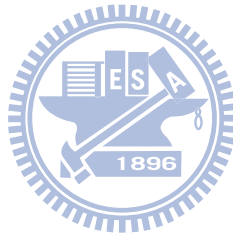


Fig. 2.10 Tracking error of $x(k)$ using FNAGM.

2.5 Summary

This chapter presents an FNAGM to predict signals in three phases, initialization phase, GM(1,1) prediction phase and FNAGM prediction phase. FNAGM adopts the on-line batch training to learn and then estimate the prediction error of GM(1,1) by the feedforward NN. Most importantly, the on-line batch training is applicable in real-time to accurately update the weight vector of NN and continually adapt FNAGM to the dynamical change. The simulation results demonstrate that the proposed intelligent forecasting system based on FNAGM is superior to the other existing methods. Experimental results demonstrate that FNAGM could achieve both trajectory prediction and target tracking in high accuracy for robotic application.



Chapter 3 Neuron-Based Structure Learning for Prediction

In Chapter 2, FNAGM has been developed to deal with prediction problem. However, the topology of NN used in FNAGM should be fully connected and the number of hidden neurons is determined through a trial-and-error method. Therefore, this chapter presents a neuron-based structure learning algorithm, called symbiotic structure learning algorithm (SSLA), to determine the topology of FNAGM. SSLA consists of three phases: initialization phase, evaluation phase, and reproduction phase. The initialization phase establishes a neuron population and a network population. The evaluation phase calculates the fitness of NN and then shares the fitness to the participating neurons. The reproduction phase performs neuron crossover and mutation on the neuron population based on their fitness. Finally, an FNAGM with appropriate topology is evolved by the neuron-based structure learning. Accordingly, a novel forecasting system, FNAGM-SSLA, is presented in this chapter where SSLA performs structure learning of FNAGM and the evolved FNAGM then predicts the signal and continuously learns to compensate the prediction error by NN with the on-line batch training.

3.1 Structure Learning Based on Symbiotic Evolution

Symbiotic evolution is an implicit fitness-sharing algorithm used in an immune system model [90], [91]. In general evolution algorithm, each individual represents a

complete solution of a problem. In symbiotic evolution, each individual in the population represents a partial solution to a given problem, and complete solutions are constructed from several individuals. The partial solutions can be considered specializations that ensure diversity and prevent a population from converging into suboptimal solutions. Furthermore, the fitness of an individual depends on other cooperated individuals. The work of [91] proposed symbiotic adaptive neuron evolution to develop NNs according to a population of neurons. The performance of a neuron is determined by how well it cooperates with the other neurons which it combines. This process shows that a neuron that cooperates well with one set of neurons may cooperate poorly with other sets of neurons. However, the number of hidden neurons must be assigned prior to the evolution process. Therefore, this algorithm is applicable for NN where the number of hidden neurons is known. Aside from NNs, many studies have applied symbiotic evolution to design fuzzy controller and neuro-fuzzy systems [92]-[98]. The results of these studies have demonstrated the efficiency and feasibility of symbiotic evolution in structure learning.

3.2 Symbiotic Structure Learning Algorithm

Section 2.1 provides a constraint that the topology of NN used in an FNAGM should be fully connected. Furthermore, the number of neurons is determined through a trial-and-error method [99]. This section presents a novel forecasting system which is composed of structure learning and on-line parameter learning as shown in Fig. 3.1. Briefly, the proposed forecasting system first evolves the structure of FNAGM by a proposed SSLA and then performs prediction and on-line batch training by the evolved FNAGM. Before the forecasting system starts, a small number of time series data is

acquired for the structure learning. The proposed SSLA determines the number of hidden neurons and the connected topology of FNAGM based on the training data. Once the structure learning is completed, the signal is continuously acquired for on-line prediction and parameter learning. Then the evolved FNAGM predicts the signal and continuously adapts itself to the dynamical change of the signal by on-line batch training. SSLA consists of three phases: initialization phase, evaluation phase, and reproduction phase which are described in the following subsections.

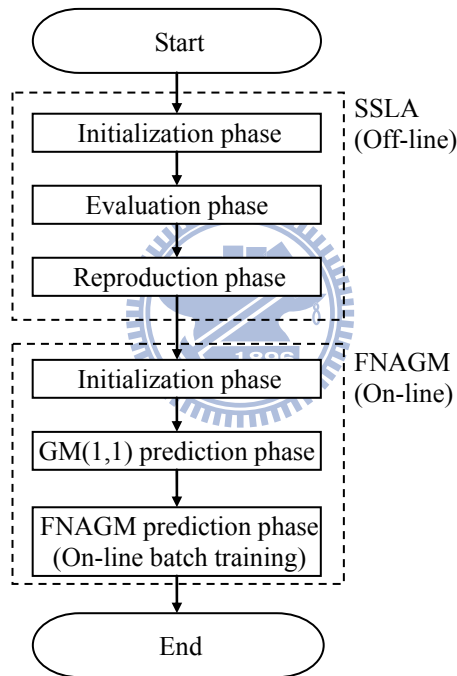


Fig. 3.1 Architecture of the proposed forecasting system.

3.2.1 Initialization Phase

A. Coding Step

Each neuron, the individual in the neuron population, consists of seven genes for four inputs problem as shown in Fig. 3.2. The seven genes are w_o , a , w_{hb} , w_{h1} , w_{h2} , w_{h3} , and w_{h4} which represent output weight, activation function type, and weights connected

from bias and four inputs to the neuron, respectively. The activation function type indicates which activation function the neuron uses where 1 represents hyperbolic tangent function and 0 represents linear function. Note that some weights of the neurons in Fig. 3.2 are zero which means that the weights are not connected to the neurons and the neurons are partially connected. Fig. 3.3 shows the graphical representation of neurons for three examples of Fig. 3.2.

Neuron	w_o	a	w_{hb}	w_{h1}	w_{h2}	w_{h3}	w_{h4}
Neuron A	1.5	1	0.9	1.3	2.1	0.2	0.1
Neuron B	0.7	0	0	0	0	2.5	-0.2
Neuron C	0.3	0	-0.4	0	0	0	0

Fig. 3.2 Coding of neuron and three examples.

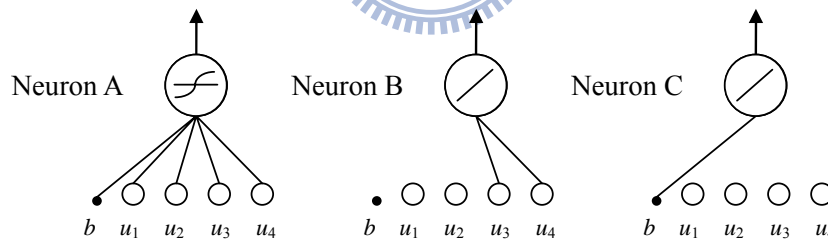


Fig. 3.3 Graphical representation of neurons for three examples.

The next step is to construct an NN from the three neurons and Fig. 3.4(a) shows the resulted NN whose output is the summation of the three neurons. It can be seen that NN has a partially connected topology and consists of different activation functions. Since the output of the neuron with linear activation function is the linear combination of the inputs, NN in Fig. 3.4(a) can be simplified as an equivalent cascade NN model in Fig. 3.4(b). Note that Neuron C can be simplified as the output bias and Neuron B

allows u_3 and u_4 to directly connect to the output neuron. Therefore, SSLA can result in not only feedforward NN but cascade NN without additional output weight coding step.

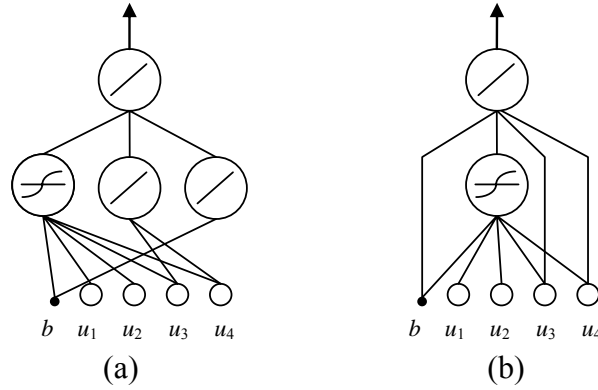


Fig. 3.4 (a) An NN constructed via three neurons. (b) Equivalent cascade NN model.

B. Create Network Population

SSLA creates neuron population by randomly generating N_p neurons where the weights and activation function type are randomly assigned. Note that some weights are randomly selected to be zero as the disconnected weights. Then, SSLA creates network population in four steps.

- Step 1) This step constructs P groups where each group consists of d empty individuals and thus the network population consists of $P \times d$ empty individuals.
- Step 2) Randomly select Group p where the reason of this step is further illustrated in Section 3.2.2.
- Step 3) Select one empty individual in Group p and build an NN by randomly selecting p neurons from the neuron population as shown in Fig. 3.5. For example, NN in Group 1 has one neuron and its length of chromosome is 7 while NN in Group P has P neurons and its length of chromosome is

7P. Note that the evaluation operator is performed on the chromosomes of network population while the reproduction operator is performed on the chromosomes of neuron population.

Step 4) Train the built NN via the approach in Section 3.2.2 so that the chromosome of NN would be updated expected for the activation function type. Go to Step 2) until the network population has no empty individual.

Each neuron can be selected at most one time for an NN and thus the neurons of an NN are different. It is expected that all neurons can be selected at least one time to construct all NNs in the network population so that the performance of all neurons can be evaluated. However, this mechanism can not be guaranteed according to the random selection in Step 3). Therefore, some neurons may not be selected in one generation. Nevertheless, this would not affect the evolution harmfully because a very few neurons are not selected via the observation of the experiment. Through the network population creation, SSLA generate numerous NNs with different number of hidden neurons and connection topology so that an NN with appropriate structure can be evolved.

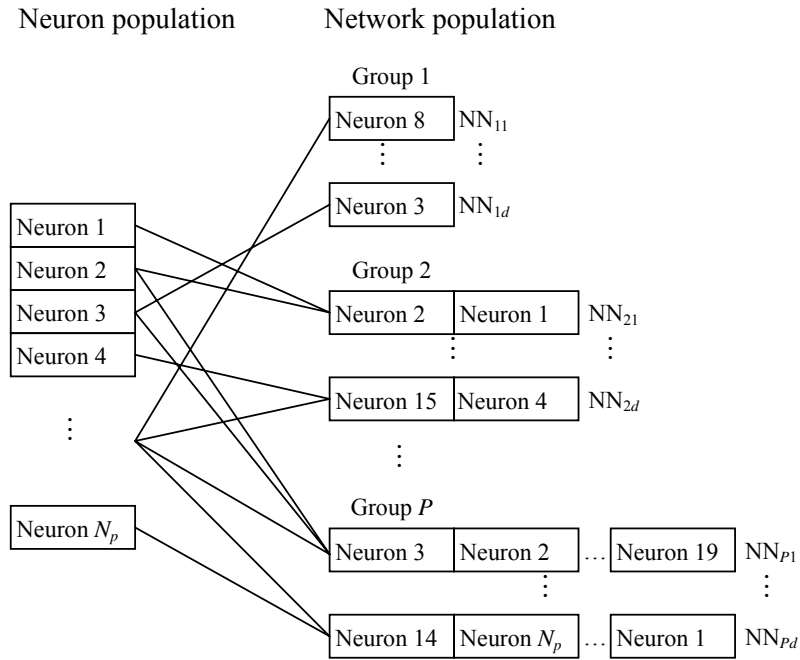


Fig. 3.5 Neuron population and network population.

3.2.2 Evaluation Phase

The evaluation phase mainly consists of two steps: weight training and fitness calculation of NN and neuron.

A. Weight Training

As mentioned in Section 1.2.2, collaboration between global search and local search could outperform individual ones. Furthermore, it is better to execute exploration and exploitation operations alternatively during evolution. Thus, SSLA performs local search to refine the weights after constructing an NN via symbiotic evolution which is a global search. SSLA then performs symbiotic evolution on the trained NNs for global search of NN structure. The global search and local search execute in a recurring fashion, one after another repeatedly. SSLA employs backpropagation algorithm (BP) [100] as local search method to train NN for φ epochs where φ is a user specified parameter. Note that BP is only performed on the connected weights, not on

disconnected weights. Once NN is trained, each trained neuron would replace the corresponding neuron in the neuron population. That means the weights of each neuron in the neuron population would be updated as participating an NN. For example, Neuron A and Neuron B are selected to construct an NN and then trained by BP for φ epochs. The two trained neurons, Neuron A' and Neuron B', would replace Neuron A and Neuron B in the neuron population, respectively. When Neuron A' and Neuron B' are selected again to construct another NN and trained by BP in the same generation, the two trained neurons, Neuron A'' and Neuron B'', replace Neuron A' and Neuron B' in the neuron population, respectively. It is expected that the latter constructed NNs might have better trained weights because the selected neurons may have been trained several times when participating in the former constructed NNs. Therefore, the latter constructed NNs possibly have better fitness than the former constructed NNs in one generation. If the NNs of Group 1 are trained earlier than the NNs of Group P , the NNs of Group P probably have better fitness than the NNs of Group 1. This may guide the evolution toward the solution with large number of hidden neurons which is not expected in SSLA. In order to achieve reasonable training process of the neuron population and network population, the Step 2) in Section 3.2.1.B, create network population, is performed by randomly selecting Group p and then constructing an NN in Group p .

B. Fitness Calculation of NN and Neuron

Once an NN is constructed and trained by BP, the inverse of RMSE is regarded as the fitness of NN, i.e., the smaller RMSE the larger fitness and vice versa. Since the purpose of SSLA is to evolve neurons in the neuron population, the fitness of the neuron should be determined. SSLA calculates the fitness of the neuron by sharing the fitness of NN to each participating neuron. For NN in the Group p , each participating neuron

can share fitness as $\frac{f}{p}$ where f is the fitness of NN. When all NNs in the network population are evaluated, the fitness of the neuron which participates in T NNs can be determined as

$$f_n = \frac{1}{T} \sum_{t=1}^T f_n^t \quad (3.1)$$

where f_n^t is the shared fitness in t th NN and $t = 1, 2, \dots, T$.

3.2.3 Reproduction Phase

This subsection presents the reproduction phase of SSLA which is performed only on the neuron population. The objective of this phase is to reproduce new neuron population from current neuron population so that various NNs can be constructed from new neuron population. The reproduction phase includes neuron crossover, neuron mutation, and survival selection. The neuron crossover exchanges the structure of two parent neurons to produce two offspring neurons and the neuron mutation modifies the structure of one offspring neuron. The survival selection attempts to probabilistically promote better solutions for the next generation from the parents and offspring of the current generation. According to the three operations, the reproduction phase mainly uses the current neuron population to evolve new neuron population for next generation and then guides the evolution to achieve near optimal solution, i.e., appropriate structure and weights of NN. Note that each parent neuron has the same length of chromosome, thus the offspring neuron also has the same length with the parent neuron. The detail concepts of the three operations are described as follows.

A. Neuron Crossover

The neuron crossover simultaneously exchanges the structure and weights of the

neurons and performs in three steps. First, select two parents according to the binary-tournament selection via their fitness. Second, randomly select the crossover point as shown in Fig. 3.6. Third, exchange the components starting from the crossover point to the end of the parents. Then two offspring, Neuron A' and Neuron B', are generated.

B. Neuron Mutation

The neuron mutation mainly attempts to modify the activation function and disconnect or connect the input for global searching of NN structure. It uses a mutation probability p_m to decide whether to perform neuron mutation on a gene, where p_m is a user specified parameter. For each gene excepted for w_o , a random number r with a uniform distribution between $[0, 1]$ is generated. If $r < p_m$, then the neuron mutation is performed on the gene. As shown in Fig. 3.7, three kinds of neuron mutation are presented. The first one shows the modification of activation function where the activation function type is modified from "0" to "1," i.e., from linear function to hyperbolic tangent function. The second one shows the disconnection of input where w_{hb} is modified from "-0.4" to "0," i.e., the bias is accordingly disconnected. The third one shows the connection of input where w_{h1} is modified from "0" to "2.9," i.e., the input is consequently connected. Note that the value of "2.9" is randomly assigned via a normal distribution with zero mean and unit variance after the input is connected.

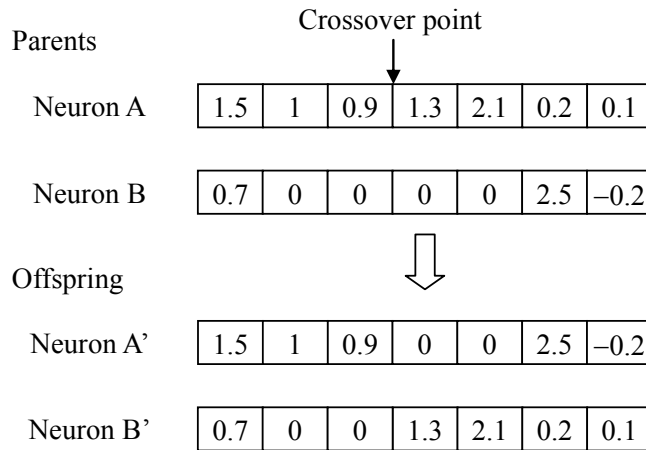


Fig. 3.6 Neuron crossover.

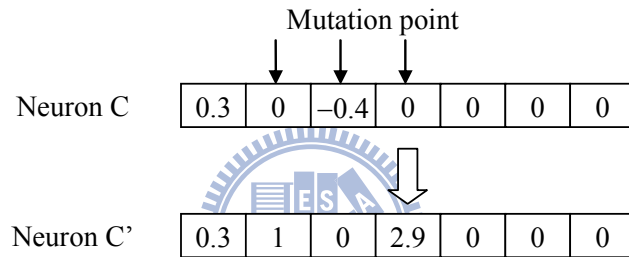


Fig. 3.7 Neuron mutation.

C. Survival Selection

The offspring neurons are generated in two ways as shown in Fig. 3.8. The parents are first rearranged in descending order of their fitness and then the best $N_p/2$ parents are copied as $N_p/2$ offspring. The rest $N_p/2$ offspring are generated via the neuron crossover and neuron mutation based on the whole parents. In Fig. 3.8, Neuron 7' is not necessarily generated from Neuron 7 and is indexed for convenience. In SSLA, the generational replacement is adopted as the survival selection, i.e., the offspring immediately replace all parents and become the parents in the next generation.

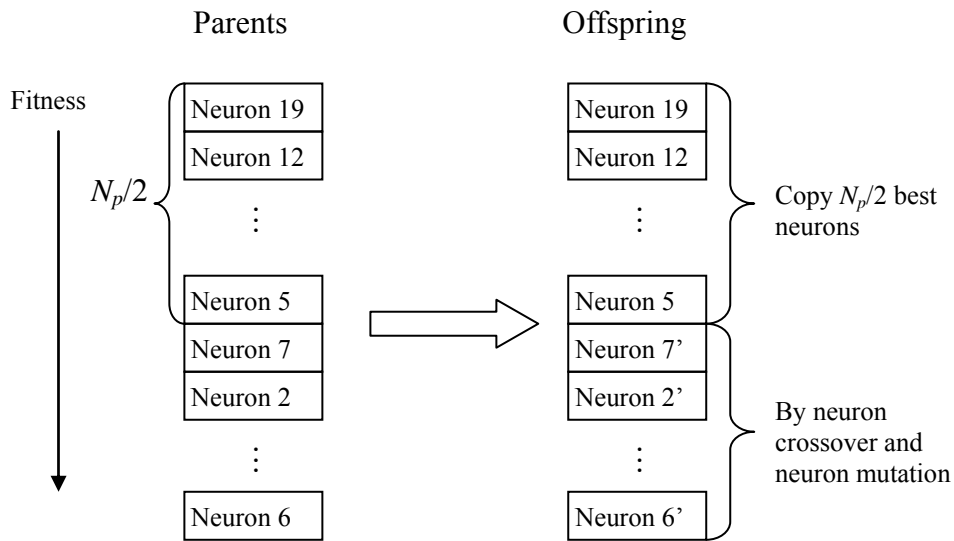


Fig. 3.8 Neuron reproduction.

Based on the three phases described above, SSLA is summarized in Fig. 3.9 as follows.

Initialization phase:

- Step 1) Create a neuron population which consists of N_p neurons whose lengths of the chromosome are the same.
- Step 2) Create a network population which consists of $P \times d$ empty individuals whose lengths of the chromosome are different.
- Step 3) Randomly select one group, Group p , and randomly select p neurons to construct an NN for an empty individual.

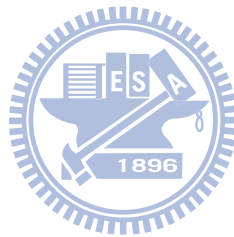
Evaluation phase:

- Step 4) Train the built NN by BP for φ epochs so that the chromosome of NN would be updated expected for the activation function type. Calculate the fitness of the NN to evaluate the chromosome of the network population.
- Step 5) Share the fitness to each participated neuron. Go to Step 3) until the network population has no empty individual.

- Step 6) Determine the fitness of all neurons in the neuron population via (3.1).
- Step 7) Preserve the NN with the best fitness.
- Step 8) If the maximum generation is achieved, then stop the algorithm; otherwise, go to Step 9).

Reproduction phase:

- Step 9) Copy the best $N_p/2$ parents as the offspring in the neuron population.
- Step 10) Generate the rest $N_p/2$ offspring by the neuron crossover based on the whole neuron population.
- Step 11) Perform the neuron mutation on the rest $N_p/2$ offspring.
- Step 12) Perform the generational replacement and go to Step 2).



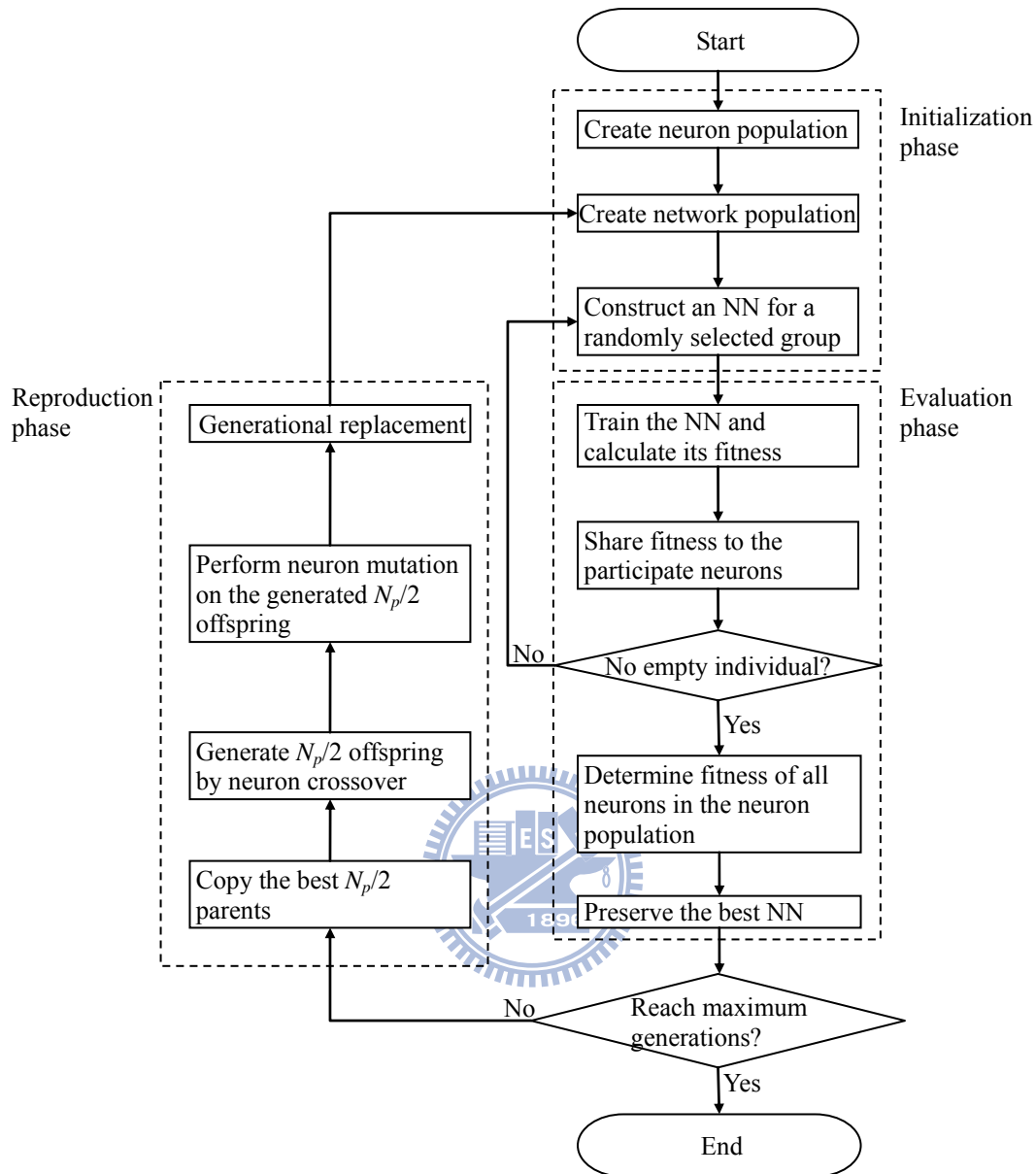


Fig. 3.9 Flowchart of SSLA.

3.3 Numerical Results

This section provides two examples to verify the performance of the proposed FNAGM-SSLA, where the structure of FNAGM is evolved by SSLA. The first example is to predict the chaotic time series and the second example is to predict the object trajectory acquired from Eye-Robot. The numerical simulations were executed by MATLAB software in an Intel Pentium CPU at 1.5 GHz with 512 MBytes RAM.

3.3.1 Example 1: Chaotic Time Series Prediction

The Mackey-Glass time series shown in Fig. 3.10 is generated from the following delay differential equation

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \quad (3.2)$$

where $\tau = 25$ and $x(0) = 1.2$ in the simulation. The objective involves using $[x(t-3) x(t-2) x(t-1) x(t)]$ to predict $x(t+1)$. Thus, the input of NN of FNAGM is $[e_{GM}(t-3) e_{GM}(t-2) e_{GM}(t-1) e_{GM}(t)]$ and the output is $e_{GM}(t+1)$. The first 250 pairs are used as the training data. The parameter of FNAGM was set as $\gamma = 2$ in (2.15). The parameters of SSLA were set as follows: probability of mutation $p_m = 0.1$, neuron population size $N_p = 50$, number of groups $P = 5$, number of group members $d = 3$, and maximum number of generations $G_{\max} = 250$.

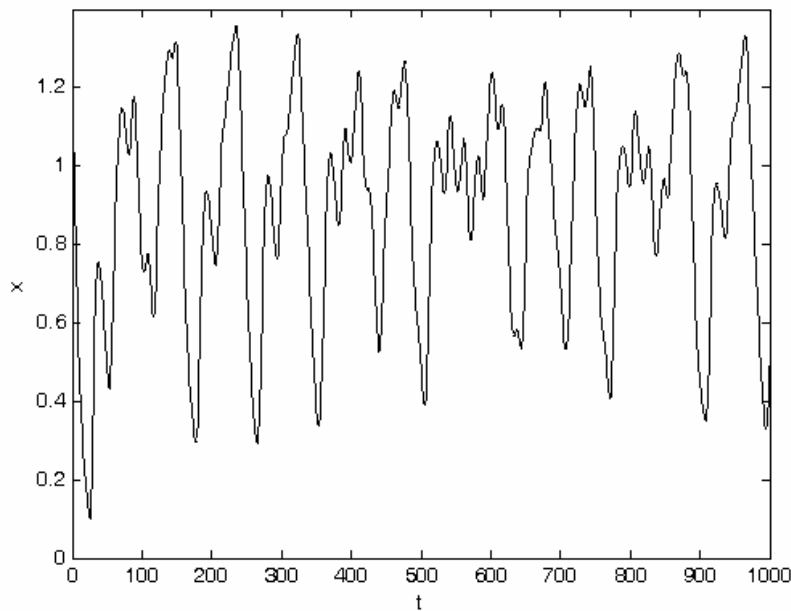


Fig. 3.10 Mackey-Glass time series.

Fig. 3.11 shows the evolved NN of FNAGM where the blue lines represent positive-valued weights and the red lines represent negative-valued weights. The input

u_1 , u_2 , u_3 , and u_4 represents $e_{GM}(t)$, $e_{GM}(t-1)$, $e_{GM}(t-2)$, and $e_{GM}(t-3)$, respectively. The evolved NN consists of two hidden neurons with hyperbolic tangent activation function and ten connections. Note that the bias does not connect to both two hidden neurons and output neuron. Furthermore, u_3 does not connect to the left hidden neuron and u_1 , u_2 , and u_4 do not connect to the right hidden neuron. Furthermore, u_1 , u_3 , and u_4 directly connect to the output neuron. Clearly, the evolved NN is a partially connected cascade NN.

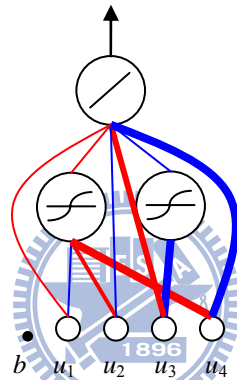


Fig. 3.11 Evolved NN of FNAGM-SSLA for Example 1.

Many approaches exist in the literature; however, it is not feasible and necessary to perform an exhaustive comparison with all algorithms. The purpose of our experimental comparison is to realize the advantages of FNAGM-SSLA. Because FNAGM-SSLA uses GM(1,1), NN, and SSLA, GM(1,1) [3], Advanced GM(1,1) [27], NN [8], and FNAGM [99] are primarily considered here for comparison. Advanced GM(1,1) is a former research work by us that is described in Section 1.2.1. NN is carried out in two ways: on-line and off-line parameter learning which are regarded as NN_{on} and NN_{off}, respectively. NN_{off} uses the first 500 and last 500 data points as the training data and the testing data, respectively. FNAGM is the work without structure learning which has been described earlier in Chapter 2. The number of hidden neurons of NN_{on}, NN_{off},

and FNAGM were selected by the preliminary tests which evaluate the performance of systems with different number of hidden neurons, from 1 to 20. As a result, the number of hidden neurons of NNNon, NNOff, and FNAGM with 8, 7, and 4 hidden neurons behave the highest performance, respectively. Note that NNNon, NNOff, and FNAGM have fully connected topologies.

Fig. 3.12 shows the absolute prediction errors of GM(1,1), Advanced GM(1,1), NNNon, NNOff, FNAGM, and FNAGM-SSLA. It can be seen that FNAGM performs better than GM(1,1), Advanced GM(1,1), NNNon, and NNOff. FNAGM-SSLA further improves FNAGM and has smaller prediction error. Although FNAGM-SSLA has larger prediction error in the beginning time steps, it can continuously perform learning and improve the prediction error.

To better illustrate the efficiency of the proposed FNAGM-SSLA, Table 3.1 shows the mean RMSE, number of hidden neurons (N_h), number of connections (N_c), and computation time of each prediction step (T_c) over 10 independent runs. It can be seen that FNAGM-SSLA has a better RMSE 6.92×10^{-4} than GM(1,1), Advanced GM(1,1), NNNon, NNOff, and FNAGM. Furthermore, FNAGM-SSLA has a more compact structure than NNNon, NNOff, and FNAGM in terms of N_h and N_c due to the use of SSLA. This implies that more connections do not necessarily result in superior fitness. In other words, appropriate connected topology is the key to improving fitness, not the number of connections. Since FNAGM-SSLA has averagely 2.9 hidden neurons and 13.6 weights (parameters) where the standard deviations are 1.34 and 3.75 respectively, it requires less computation time 1.90×10^{-3} second than FNAGM for both prediction and on-line batch training. Thus, it can be employed to real-time prediction application where the sampling time is larger than 2 ms.

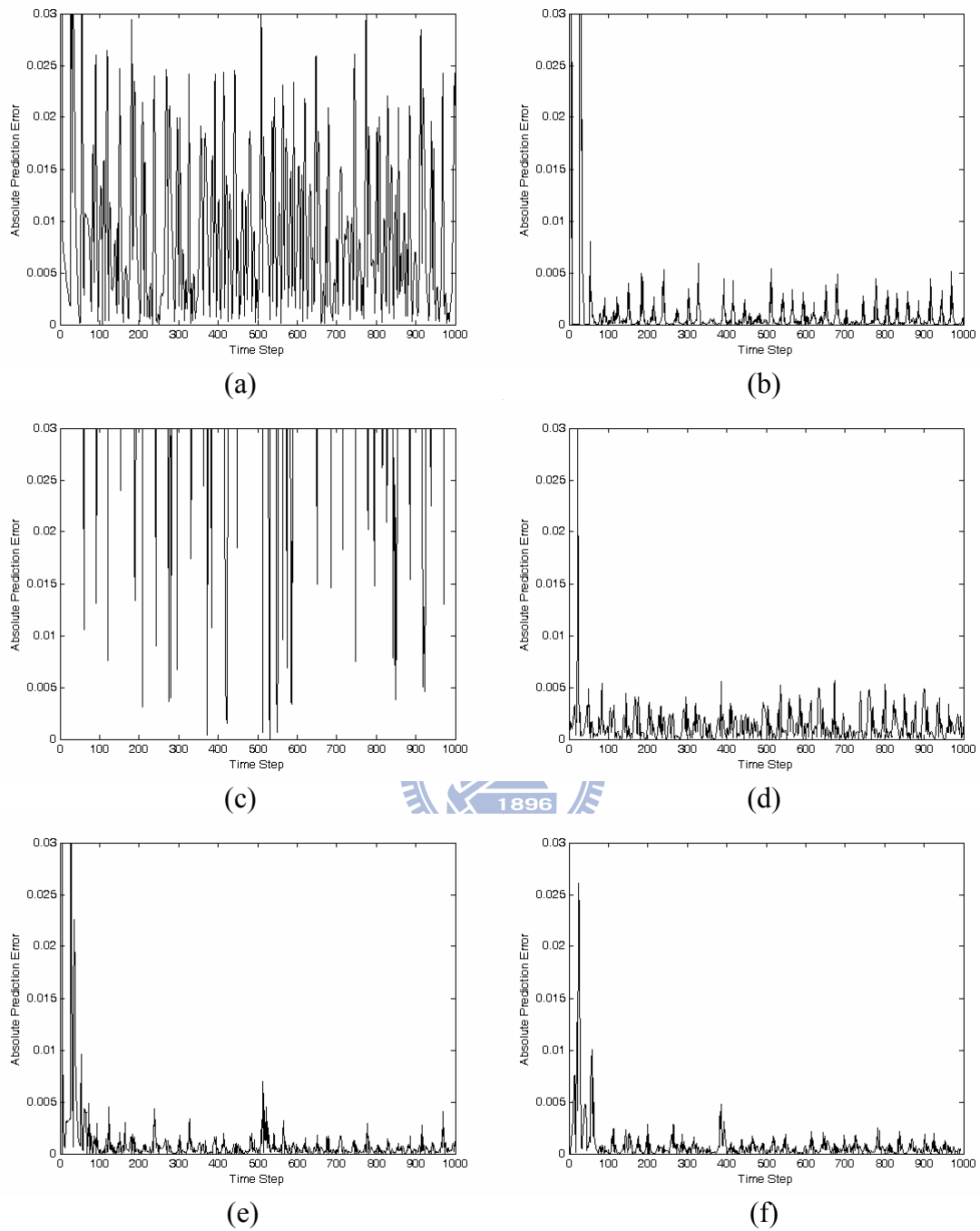


Fig. 3.12 Absolute prediction errors in Example 1. (a) GM(1,1) [3]; (b) Advanced GM(1,1) [27]; (c) NNon [8]; (d) NNoff [8]; (e) FNAGM; (f) FNAGM-SSLA.

Table 3.1 Comparison of prediction results for Example 1.

	GM(1,1) [3]	Advanced GM(1,1) [27]	NNon [8]	NNoff [8]	FNAGM [99]	FNAGM- SSLA
RMSE for $k > 250$	1.09×10^{-2}	1.10×10^{-3}	2.65×10^{-1}	1.41×10^{-3}	9.39×10^{-4}	6.92×10^{-4}
N_h	---	---	8	7	4	2.9(1.34)
N_c	---	---	49	43	25	13.6(3.75)
T_c	4.08×10^{-4}	4.15×10^{-4}	2.40×10^{-4}	1.81×10^{-4}	4.80×10^{-3}	1.90×10^{-3}

3.3.2 Example 2: Object Trajectory Prediction

The purpose of this example is to predict object trajectory during one sampling interval, 3.3×10^{-2} second, so that Eye-Robot can make decision in real-time according to the information. Fig. 3.13 shows the object trajectory where the data acquisition process is described in Section 2.4.1. The prediction goal of FNAGM is to predict $x(t+1)$ using the input $[x(t-3) \ x(t-2) \ x(t-1) \ x(t)]$. In this way, $[e_{GM}[k-3] \ e_{GM}[k-2] \ e_{GM}[k-1] \ e_{GM}[k]]$ is chosen as the input of NN of FNAGM and $e_{GM}[k+1]$ is the target value. The setup of FNAGM-SSLA is the same as Example 1 in Section 3.3.1. Furthermore, the training data were normalized to the range $[-1, 1]$.

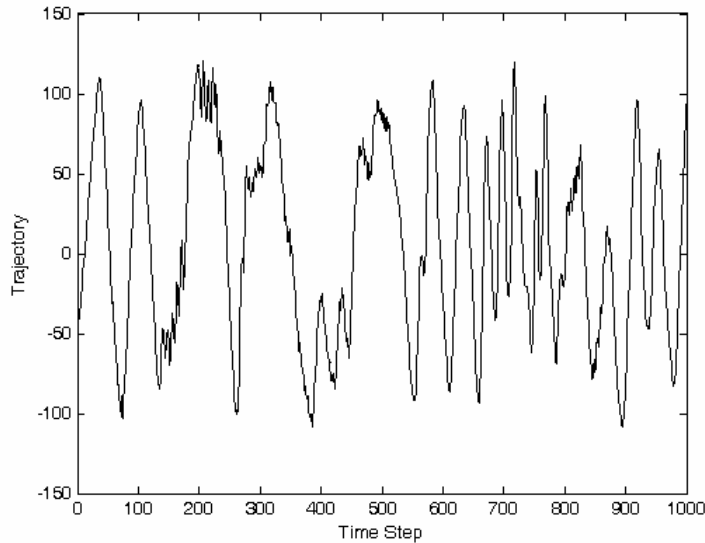


Fig. 3.13 Object trajectory captured by Eye-Robot.

Fig. 3.14 shows the evolved NN of FNAGM-SSLA. The evolved NN consists of one hidden neuron with hyperbolic tangent activation function and 9 connections. Note that the bias does not connect to the hidden neuron. Furthermore, u_2 does not connect to the hidden neuron. Moreover, the four inputs directly connect to the output neuron. Clearly, the evolved NN is a partially connected cascade NN.

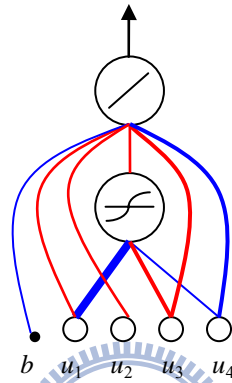
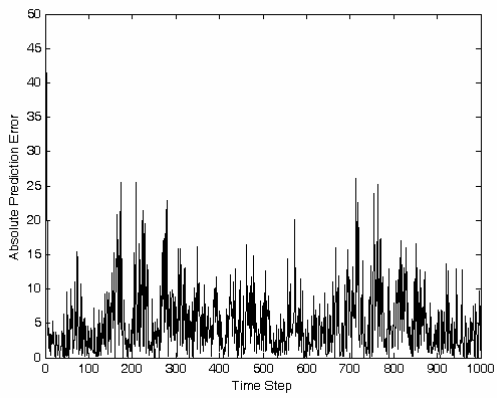
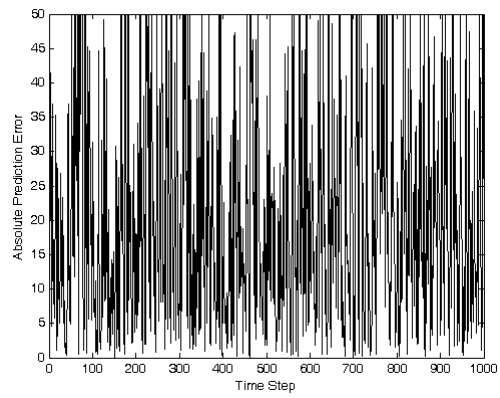


Fig. 3.14 Evolved NN of FNAGM-SSLA for Example 2.

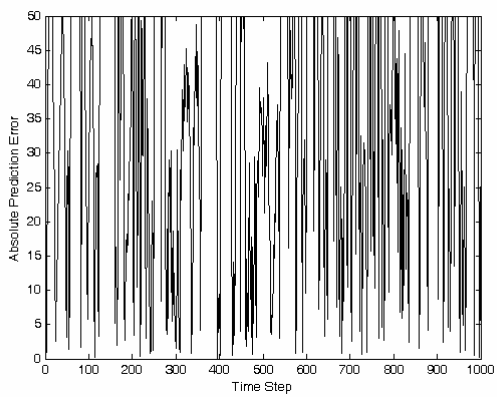
To evaluate the effectiveness of the proposed FNAGM-SSLA, Fig. 3.15 shows the prediction results of GM(1,1), Advanced GM(1,1), NN, FNAGM, and FNAGM-SSLA. The number of neurons of NN_{on}, NN_{off}, and FNAGM is chosen by the same procedure shown in Section 3.3.1. Consequently, NN_{on}, NN_{off}, and FNAGM with 6, 4, and 5 neurons have the best performance. Note that NN_{on}, NN_{off}, and FNAGM have fully connected topologies. It can be observed that GM(1,1) performs better than Advanced GM(1,1), NN_{on}, and NN_{off} in this example, while Advanced GM(1,1) and NN_{off} perform better than GM(1,1) in Example 1. FNAGM has better performance than GM(1,1) and FNAGM-SSLA further improves the prediction error of FNAGM.



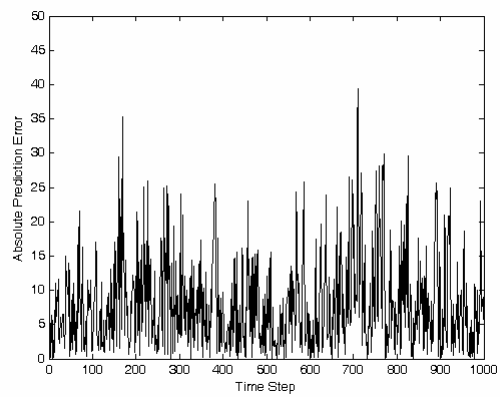
(a)



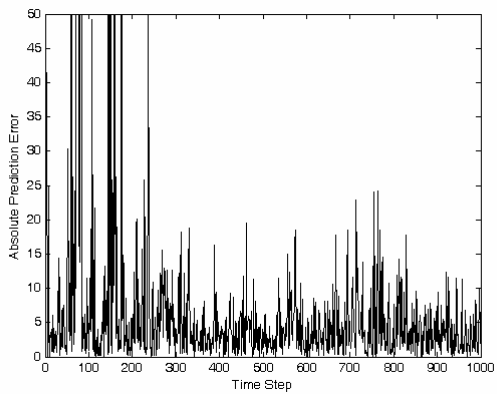
(b)



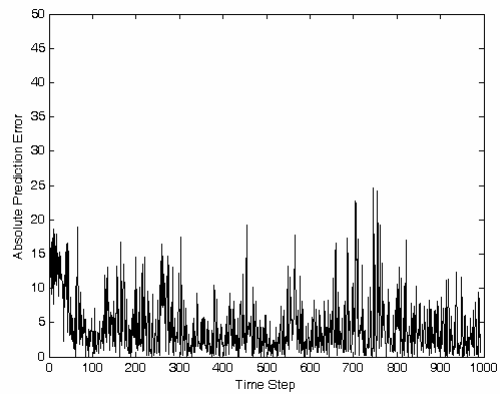
(c)



(d)



(e)



(f)

Fig. 3.15 Absolute prediction errors in Example 2. (a) GM(1,1) [3]; (b) Advanced GM(1,1) [27]; (c) NNon [8]; (d) NNoFF [8]; (e) FNAGM; (f) FNAGM-SSLA.

Table 3.2 Comparison of prediction results for Example 2.

	GM(1,1) [3]	Advanced GM(1,1) [27]	NNon [8]	NNoff [8]	FNAGM [99]	FNAGM- SSLA
RMSE for $k > 250$	6.6491	29.5957	72.5432	7.9706	6.0348	5.3806
N_h	---	---	6	4	5	1.3(1.28)
N_c	---	---	37	25	31	8.7(4.37)
T_c	4.08×10^{-4}	4.15×10^{-4}	2.40×10^{-4}	1.81×10^{-4}	8.00×10^{-3}	1.80×10^{-3}

Table 3.2 presents mean RMSE, N_h , N_c , and T_c of NNon, NNoff, FNAGM, and FNAGM-SSLA over 10 independent runs. FNAGM-SSLA achieves a better RMSE 5.5642 than GM(1,1), Advanced GM(1,1), NNon, NNoff, and FNAGM. In case of FNAGM-SSLA, the average N_h and N_c , 1.3 and 8.7 where the standard deviations are 1.28 and 4.37 respectively, are less than those of NNon, NNoff, and FNAGM. It can be seen that a better exploration in the structure search space is realized due to the use of SSLA which achieves compact structure of cascade NN and high prediction performance. Since FNAGM-SSLA has fewer hidden neurons and weights than FNAGM, it requires less computation time 1.80×10^{-3} second than FNAGM for both prediction and on-line batch training. The experiments show the same evidence as numerous studies [44]-[46] that partially connected NNs have better storage capability per connection than fully connected NNs. Furthermore, the results show that the evolved FNAGM can be used to real-time prediction application due to the less computation time.

3.4 Summary

This chapter presents a novel forecasting system, FNAGM-SSLA, which first facilitates structure learning of FNAGM by SSLA and then achieves prediction and

on-line batch training by the evolved FNAGM. The idea behind SSLA is to achieve neuron-based structure learning and develop neurons to construct NNs through symbiotic evolution. SSLA can construct cascade NNs and feedforward NNs by evolving the neurons with different activation functions including hyperbolic tangent and linear functions. It determines not only the number of hidden neurons, but also the connected topology between input and hidden layers. The experiments show that FNAGM-SSLA can obtain an appropriate FNAGM structure, with fewer neurons and connections, and a smaller prediction error than FNAGM designed in an empirical way. This implies that more connections do not necessarily result in superior fitness. In other words, appropriate connected topology is the key to improving fitness, not the number of connections. Furthermore, the experiments show that FNAGM-SSLA requires less computation time and can be used for real-time prediction application.



Chapter 4 Network-Based Structural Learning for Prediction

This chapter presents a method of designing NNs for prediction problems based on an evolutionary constructive and pruning algorithm (ECPA). The proposed ECPA begins with a set of NNs with the simplest possible structure, one hidden neuron connected to an input node, and employs crossover and mutation operators to increase the complexity of an NN population. Additionally, cluster-based pruning (CBP) and age-based survival selection (ABSS) are proposed as two new operators for NN pruning. The CBP operator retains significant neurons and prunes insignificant neurons on a probability basis and therefore prevents the exponential growth of an NN. The ABSS operator can delete old NNs with potentially complex structures and then introduce new NNs with simple structures; thus, NNs are less likely to be trapped in a fully connected topology. The ECPA framework incorporates constructive and pruning approaches in an attempt to efficiently evolve compact NNs. As a demonstration of the method, ECPA is applied to three prediction problems: the Mackey-Glass time series, the number of sunspots, and traffic flow. The numerical results show that ECPA makes the design of NNs more feasible and practical for real-world applications.

4.1 Basic Concept of Evolutionary algorithm

EA, which simulates Darwinian evolution, is a parameter optimization algorithm [101] that works through a simple cycle of stages. The EA begins with a randomly

generated population consisting of a number of feasible solutions for the problem. This population is referred to as the parents, and each solution is known as an individual. The next stage evaluates the fitness of each individual of the population. To improve the individuals of the current generation, the crossover operator is adopted to perform essential recombination of two or more individuals. Furthermore, to sample unknown regions, the mutation operator is employed to make a change or perturbation in a parameter with a random element. In other words, the crossover is considered to be an exploration operator, whereas the mutation is considered to be an exploitation operator. Therefore, EA creates a new population, known as the offspring, using the crossover and mutation operators. To maintain a constant population size over subsequent generations, the next stage performs a selection to determine whether the individuals in both the parent and offspring populations survive to the next generation. Ranking and tournament selections are frequently employed as selection strategies [102]. The EA stops when the maximum number of generations is reached.

4.2 Evolutionary Constructive and Pruning Algorithm

Based on the characteristics of EA, constructive, and pruning algorithms mentioned in Section 1.2.2, we propose ECPA to develop NNs in an attempt to balance the constructive and pruning manners in an evolutionary way. This approach starts from a group of NNs with the simplest possible structure, one hidden neuron connected to an input node. It then employs network crossover and network mutation to make NNs more complex, and adopts CBP and ABSS to prune NNs. As discussed in [81], theoretical work has shown that a single hidden layer is sufficient for forecasting purposes. Therefore, in this work, we designed a three-layer feedforward NN with an

input layer, a hidden layer, and an output layer. The major steps of ECPA are summarized in Fig. 4.1 and explained below.

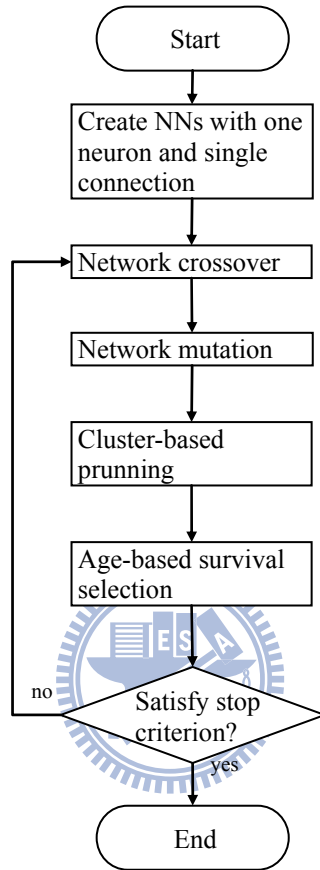


Fig. 4.1 Major steps performed in ECPA.

Initialization phase:

Step 1) Generate an initial population with N_p NNs, where N_p is the population size. The initial NN structure starts with a simplest possible network with one neuron and a single connection from one of the inputs, which is randomly selected.

Step 2) Train all NNs using BP for φ epochs, where φ is specified by the user, and determine their fitness.

Reproduction phase:

- Step 3) Select two parents by tournament selection. Produce one offspring from the two parents using network crossover.
- Step 4) Apply network mutation to the offspring.
- Step 5) Train the offspring NN using BP for φ epochs and determine its fitness.
- Step 6) Perform CBP on the offspring. Go to Step 3 until N_p offspring are generated.
- Step 7) Apply ABSS to the parents and their offspring to select the parents of the next generation. Go to Step 3 until the maximum number of generations is reached.
- Step 8) Select a single best NN among the final population.

4.2.1 Encoding Scheme and Design Mechanism

In order to encode an NN into a chromosome, NN is represented as a vector whose length depends on the size of NN such that the memory can be used efficiently. Fig. 4.2 shows the chromosome representation of two NNs and their corresponding graphical representations. The chromosome consists of the network connections and weights where w_1 , w_{b1} , and w_{12} indicate the output weight of the first hidden neuron, the weight connected from bias, and weight connected between first hidden neuron and second input node, respectively. Note that the weight with nonzero value represents the connected weight while that with zero value represents the disconnected weight. The initial population is a set of simplest possible networks whose initial weights are randomly generated by a uniform distribution in the range $[-1.0, 1.0]$ via the suggestion in [104].

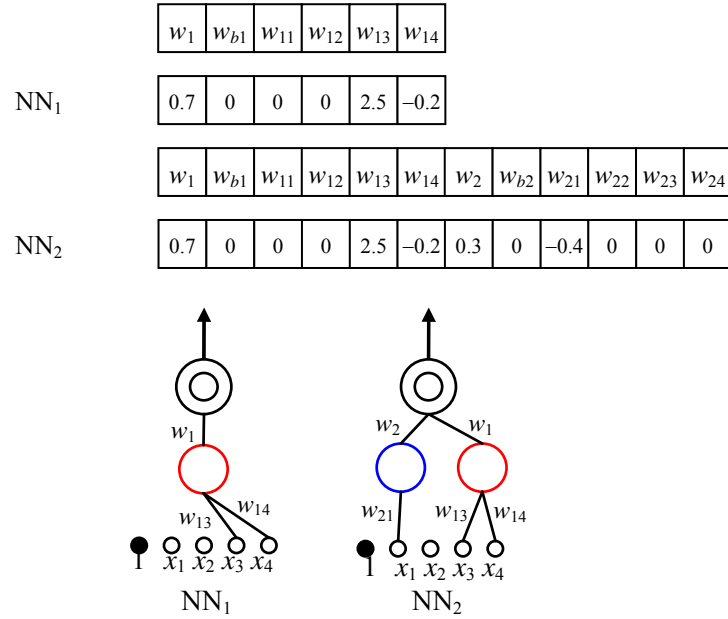


Fig. 4.2 Coding of NN and two examples.

The data set used in ECPA is partitioned into two sets: training set and testing set. In Step 2 and 5, the training set is used to train the weights of NN by local search operator and evaluate the fitness of NN. As mentioned in major steps, ECPA uses both global and local search to design NNs. The global operators in reproduction phase are used to explore the NN structures, and the local operator, BP, is used to enable a precise local search of weights. As shown in Section 1.2.2, it is better to carry out global search and local search alternatively during evolution. Therefore, ECPA performs weight training by BP after structure search by evolutionary operators, again and again. Basically, ECPA directs the evolution of NNs via four essential components: network crossover, network mutation, CBP, and ABSS. Details regarding each component of ECPA are provided in the following sections.

4.2.2 Network Crossover

The ECPA starts from a population of NNs with the simplest possible structures so

that the initialization phase can easily set the topology of NNs as one hidden neuron and a single connection from one input. However, these NNs may not be able to achieve enough and desired accuracy. In order to increase the processing capabilities of NNs, it is necessary to facilitate the exploration of the wider regions of a structural search space. For the sake of this objective, this stage executes constructive manner to add hidden neurons to each NN. To decide how many hidden neurons should be add to each NN, network crossover simply selects two NNs and combines them together. Hence, this operator does not necessarily use many heuristics and user-defined parameters, and require rich prior knowledge. The network crossover operation in ECPA produces an offspring NN by combining the substructures of two parent NNs. To clearly illustrate the network crossover, an example is shown in Fig. 4.3 for two parent NNs, NN_a and NN_b , and their offspring NN_c . The input-output relationship of NN_a is as follows:

$$y^a = w_{o1}^a \cdot h(w_{h11}^a \cdot x_1 + w_{h12}^a \cdot x_2) \quad (4.1)$$

where h is the hidden neuron activation function, w_{o1}^a is the output weight, and w_{h12}^a is the weight connected from x_2 to the hidden neuron. The hidden neuron activation function can be a linear, logistic or hyperbolic tangent function. The superscript of each weight represents its network index, and the subscript indicates the relationship between hidden and input neurons. For NN_b , the input-output relationship is as follows:

$$y^b = w_{o1}^b \cdot h(w_{h11}^b \cdot x_1) + w_{o2}^b \cdot h(w_{h22}^b \cdot x_2 + w_{h24}^b \cdot x_4) \quad (4.2)$$

where w_{o1}^b is the output weight of the first hidden neurons and w_{h24}^b is the weight from x_4 to the second hidden neuron.

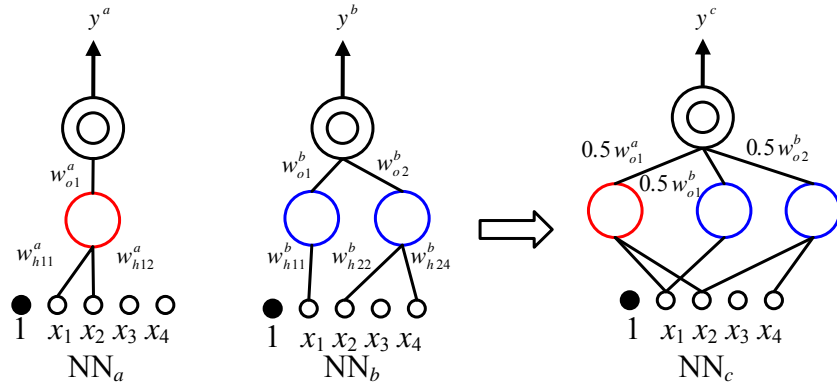


Fig. 4.3 An example of a network crossover.

The network crossover directly combines the substructures of NN_a and NN_b , and the offspring NN_c is subsequently obtained as:

$$\begin{aligned}
 y^c &= 1/2(y^a + y^b) \\
 &= 0.5 w_{o1}^a \cdot h(w_{h11}^a \cdot x_1 + w_{h12}^a \cdot x_2) \\
 &\quad + 0.5 w_{o1}^b \cdot h(w_{h11}^b \cdot x_1) + 0.5 w_{o2}^b \cdot h(w_{h22}^b \cdot x_2 + w_{h24}^b \cdot x_4)
 \end{aligned} \tag{4.3}$$

This result is shown in Fig. 4.3. The output weights of the offspring NN are half those of the parent NNs, and the hidden weights of the offspring retain the same weights as those of the parent NNs. As shown in Fig. 4.3, it is clear that the network crossover operator directs the evolution of NNs in a constructive manner. Furthermore, a crossover probability is chosen to determine whether or not to perform network crossover on two parent NNs. If the crossover probability is smaller than a random number, network crossover is performed on the two parent NNs; otherwise, the two parent NNs are copied as two offspring.

4.2.3 Network Mutation

When the network crossover is applied to parent NNs, some offspring NNs are likely to have more hidden neurons and thus possess much processing ability. However,

it may be inefficient to increase NN's performance by only adding hidden neuron with single connection generated by the initialization phase. It is possible to introduce more inputs into each hidden neuron to increase the prediction accuracy. EAs usually adopt mutation operator to achieve the perturbation and thus have a better exploitation capability. Hence, a small perturbation of structure is suitable for structural leaning. Since the initialization phase generates NNs with single connection, a small perturbation can be achieved via adding more connections to NNs and its capability is distinct from adding hidden neurons. For the sake of simple and small perturbation, network mutation is developed in ECPA and introduces a new connection into NN where the connection is built between randomly selected one input and one hidden neuron, and initializes its weight according to a normal distribution with mean = 0 and standard deviation = 0.01, by the program. A graphical representation of this operation, in which a new connection is added between x_2 and the first hidden neuron, is shown in Fig. 4.4. The input-output relationship of the mutated NN_b , $NN_{b'}$, is written as follows:

$$y^b = w_{o1}^b \cdot h(w_{h11}^b \cdot x_1 + w_{h12}^b \cdot x_2) + w_{o2}^b \cdot h(w_{h22}^b \cdot x_2 + w_{h24}^b \cdot x_4) \quad (4.4)$$

where $w_{h12}^b = 0$ and thus, $NN_{b'}$ retains the performance of NN_b .

4.2.4 Cluster-Based Pruning

ECPA employs network crossover and network mutation to design NNs in a constructive manner. However, it is well known that the constructive algorithms difficultly decide when to stop the addition process and may design an excessively large and complex NN with poor generalization performance. Thus, a pruning algorithm can be used to determine the relevance or significance of hidden neurons and delete insignificant ones. Nevertheless, it is not easy to determine the threshold value for

distinguishing insignificant hidden neurons from significant ones. Therefore, ECPA uses a different pruning scheme, called CBP, which simply separates the hidden neurons into two classes, good and worse, according to the best hidden neuron and the worst one without any user specified parameter. Then the hidden neurons in worse class are pruned in a stochastic way to avoid deleting excessive hidden neurons. Unlike conventional pruning algorithms, CBP proceeds in three steps.

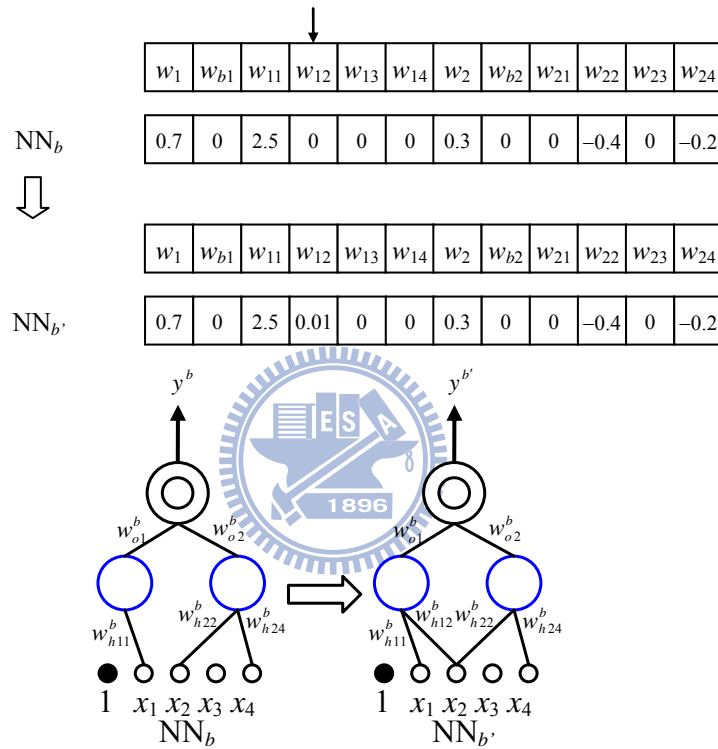


Fig. 4.4 An example of a network mutation.

In the first step, the significance of each hidden neuron is determined. For the i th hidden neuron, the significance is defined as

$$\sigma_i = \sqrt{S_i} \quad (4.5)$$

where S_i is obtained as follows [105]-[107]:

$$S_i = \sqrt{\frac{\sum_{p=1}^P (S_i^p)^2}{P}} \quad (4.6)$$

Thus, S_i is the root-mean-square of S_i^p , which is the sensitivity of the network output o^p to the output h_i^p of the i th hidden neuron for the p th pattern, expressed as

$$\begin{aligned} S_i^p &= \frac{\partial o^p}{\partial h_i^p} \\ &= w_i \end{aligned} \quad (4.7)$$

Here, w_i is the weight of the connection from the i th hidden neuron to the output neuron; this weight is constant because it is irrelevant to the patterns. Hence, the significance in (4.5) can be rewritten as

$$\sigma_i = \sqrt{|w_i|} \quad (4.8)$$

Thus, a hidden neuron with low significance has little influence on the network output and can be removed [51]. However, to avoid excessive pruning of the hidden neurons, the significance in (4.5) is purposely chosen as the square root of S_i , following the concept of the rootogram [108].

In the second step, the hidden neurons are categorized into two classes: good and worse. The prototypes of the good and worse classes are the hidden neurons with maximum and minimum significance, respectively. The remaining hidden neurons are categorized according to the difference between the good and worse prototypes with respect to their significance. If the significance of one hidden neuron is close to the good prototype, the hidden neuron is then categorized in the good class; otherwise, the hidden neuron is categorized in the worse class. The neurons in the good class are retained, whereas those in the worse class are deleted in a stochastic manner. For each neuron in the worse class, a random number r with a uniform distribution between $[0, 1]$ is generated. If r is smaller than 0.5, the neuron is deleted; otherwise, the neuron is retained.

4.2.5 Age-Based Survival Selection

After the network crossover, network mutation and CBP are completed, the individuals in the next generation are chosen through survival selection. If a general survival selection is adopted, the evolved NNs tend to have fully connected topologies due to network mutation, which add more inputs to the hidden neurons. As a result, hardware implementation costs are increased, and the generalization capabilities of the evolved NNs are reduced. To avoid this problem, we propose a different survival selection method, ABSS, to select younger NNs with partial connections, rather than full connections, for the next generation.

ABSS is performed in two steps. The first step involves traditional tournament selection to choose N_p candidates for the next stage. If the age of an NN is defined as the number of generations it survives in the population, then the N_p candidates may have different ages. For example, the age of a newborn NN is one, and its age increases by one if it survives to the next generation. The second step continues to delete the elder NNs from the N_p candidates according to the aging index, defined as follows:

$$A_j = \left(1 - \frac{1}{Age_j}\right)^2 \quad (4.9)$$

where Age_j is the age of the j th NN. Selection proceeds by generating a uniform random number r in the range $[0, 1]$. If $A_j > r$, the j th NN is deleted and replaced by a newborn NN produced by Step 1; otherwise, the j th NN is retained in the population. As a result, the population size N_p is unchanged after ABSS, and the average age of the NNs is potentially lower, which prevents the evolved NNs from adopting a fully connected topology.

In summary, the network crossover operator constructs an NN by adding hidden neurons so that the NN possesses more processing ability to accurately approximate the

target function. The network mutation operator adds one connection from the input to the hidden neuron so that the hidden neuron can process more input information. CBP prunes the worse hidden neurons from an NN to prevent overfitting of the training data. ABSS deletes elder NNs that are potentially fully connected. Thus, network crossover and mutation operations direct the evolution of NNs in a constructive manner that can improve their processing ability to accurately approximate the true function, whereas CBP and ABSS direct the evolution of NNs in a destructive way that can improve their generalization capabilities while reducing their hardware requirements.

4.3 Numerical Results

In this section, we demonstrate the performance of the proposed algorithm using three time series prediction problems: Mackey-Glass, sunspots, and vehicle count. The first time series is generated from the Mackey-Glass differential equation, the second series is recorded from the sunspots, and the third series is obtained from the hourly vehicle count for the Monash Freeway outside Melbourne in Victoria, Australia, beginning in August, 1995. In order to make a fair comparison with previous works, the first problem adopts RMSE as the fitness, which is calculated as follows:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{t=1}^N (x(t) - \hat{x}(t))^2} \quad (4.10)$$

where $\hat{x}(t)$ is the predicted value at time t and N is the number of data points. The second and third problems compare with previous works by normalized mean squared error (NMSE) and mean absolute percentage error (MAPE). The NMSE is defined as the ratio of the mean squared error to the variance of the time-series as follows:

$$\text{NMSE} = \frac{1}{N \sigma^2} \sum_{t=1}^N (x(t) - \hat{x}(t))^2 \quad (4.11)$$

where σ is the standard deviation of the time-series. The MAPE is determined as follows:

$$\text{MAPE} = \frac{1}{N} \sum_{t=1}^N \left| \frac{x(t) - \hat{x}(t)}{x(t)} \right| \cdot 100\% . \quad (4.12)$$

Furthermore, the number of hidden neurons N_h and the number of connections N_c are recorded to observe the evolutionary progress. The following parameters are used in each problem.

- 1) The population size N_p is 30.
- 2) The crossover probability is 0.8.
- 3) The mutation probability is 0.6.
- 4) The value of φ for training NN by BP is 15.
- 5) The maximum number of generations is 500.

As described in Section 3.5, the N_p , G , and φ would affect the computational complexity of ECPA. The larger N_p the less effect of genetic drift, the larger G the more chances to find better ANNs, and the larger φ the more prediction accuracy. However, the larger N_p , G , and φ lead to the longer computation time. To select suitable values, N_p is set as 30 according to the suggestion in [109]. In order to select appropriate φ and G , the values of φ were chosen as 5, 10, 15, 20, 25, and 30, and the values of G were chosen as 300, 400, 500, and 600 in the preliminary runs. As a result, $G = 500$ and $\varphi = 15$ were adopted in the following experiments due to the sufficient prediction accuracy and acceptable computation time. Because the parameters were chosen after some preliminary runs, the value was not meant to be optimal. The setting of $p_c = 0.8$ and $p_m = 0.6$ is to enhance the chance of increasing the number of hidden neurons than the chance of increasing the number of connections. It was expected that structures with more hidden neurons would

be found first, and these structure would then be pruned. We evaluate the performance of ECPA on the three examples over 10 independent runs.

4.3.1 Example 1: Chaotic Time Series Prediction

The Mackey-Glass time series prediction is recognized as a benchmark problem in the area of NNs. This chaotic time series prediction was considered to be a suitable way to evaluate the performance of the proposed ECPA. The Mackey-Glass time series is generated from the following delay differential equation:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \quad (4.13)$$

where $\tau = 17$ and $x(0) = 1.2$ in the simulation. The fourth-order Runge-Kutta method is used to generate 1,000 data points ranging from $t = 118$ to 1,117. The task involves predicting the value of $x(t+6)$ from the input vector $[x(t-18) x(t-12) x(t-6) x(t)]$ for any t . Therefore, the input-output data pairs for prediction are

$$[x(t-18), x(t-12), x(t-6), x(t); x(t+6)]$$

where the first 500 data pairs are used as training set and the later 500 data pairs are testing set.

The evolutionary progress of NNs for the Mackey-Glass time series prediction problem is illustrated in Fig. 4.5. The top panel of Fig. 4.5 shows the decrease in RMSE resulting from the evolution of NNs. The middle and bottom panels of Fig. 4.5 present N_h and N_c and demonstrate the structural evolution of NNs, respectively. Fig. 4.6 graphically illustrates how the topologies of NNs evolve in selected generations. The input vector $[u(4) u(3) u(2) u(1)]$ represents $[x(t-18) x(t-12) x(t-6) x(t)]$, and the output y represents $x(t+6)$. The blue lines represent positive-valued weights, and the red lines represent negative-valued weights. The widths of the lines indicate the relative strengths

of the weights. The NN structure produced in the 1st generation starts with a network of two neurons and a few connections from the inputs and thus has limited information processing ability for the task. As its evolution progresses, N_h gradually increases to 38 by the 25th generation, is reduced to 28 by the 30th generation, and then increases to 40 by the 490th generation. The NNs were observed to grow rapidly, but the growth did not always occur due to the use of CBP and ABSS. Note that the resulting NN structure does not have a fully connected topology; less than 85% of the synapses are connected.

Many approaches have been developed to design both the architecture and weights of NNs to address the same prediction problem. Table 4.1 presents the experimental results obtained using the proposed algorithm compared with other algorithms. The best (i.e., lowest) RMSE, N_h , and N_c values among the various approaches are shown in boldface type, and the RMSE, N_h , and N_c of ECPA are the average values over 10 independent runs. As shown here, although ECPA achieved a larger RMSE than that of Du and Zhang [110] with the training set, it obtained a lower RMSE than the other methods for the testing data. It is interesting that ECPA obtained a lower RMSE for the testing data than for the training data in this experiment, but this phenomenon has been observed previously [111]. In terms of the average number of hidden neurons over 10 independent runs, ECPA obtained a lower N_h than those of Du and Zhang [110] and Harpham and Dawson [112]. Although ECPA obtained a higher N_h than those of Rojas et al. [113], Chen et al. [114], and Cho and Wang [115], it achieved a lower RMSE. ECPA resulted in the evolution of an NN with training data RMSE, testing data RMSE, N_h , and N_c values of 6.76×10^{-4} , 6.30×10^{-4} , 40.5, and 203.2, respectively. Clearly, the evolved NN possessed a partially connected topology; our observations showed that ECPA can evolve NNs with a lower RMSE and more compact structure than the other methods.

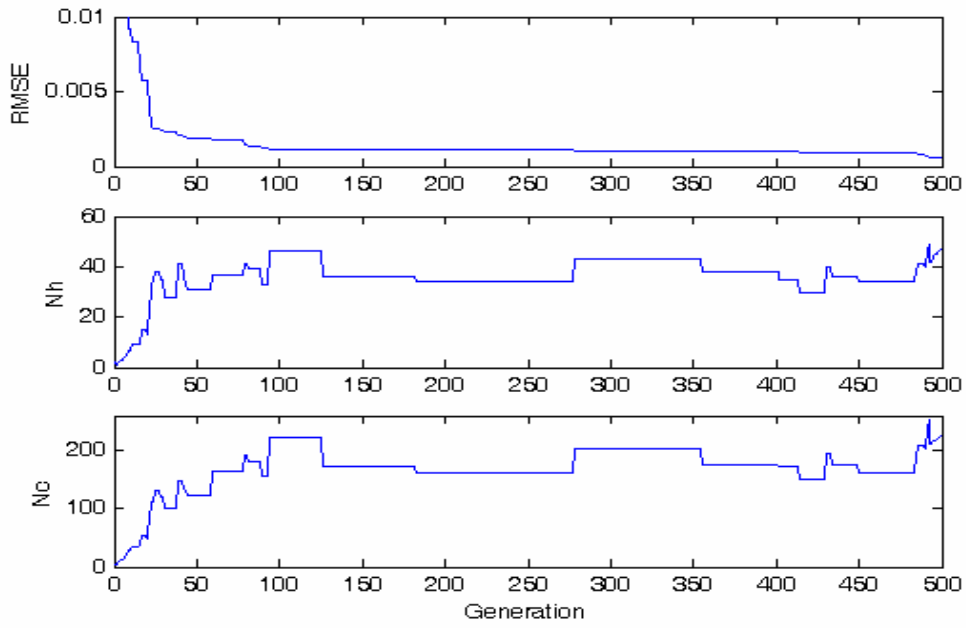


Fig. 4.5 Evolution progress for Example 1.

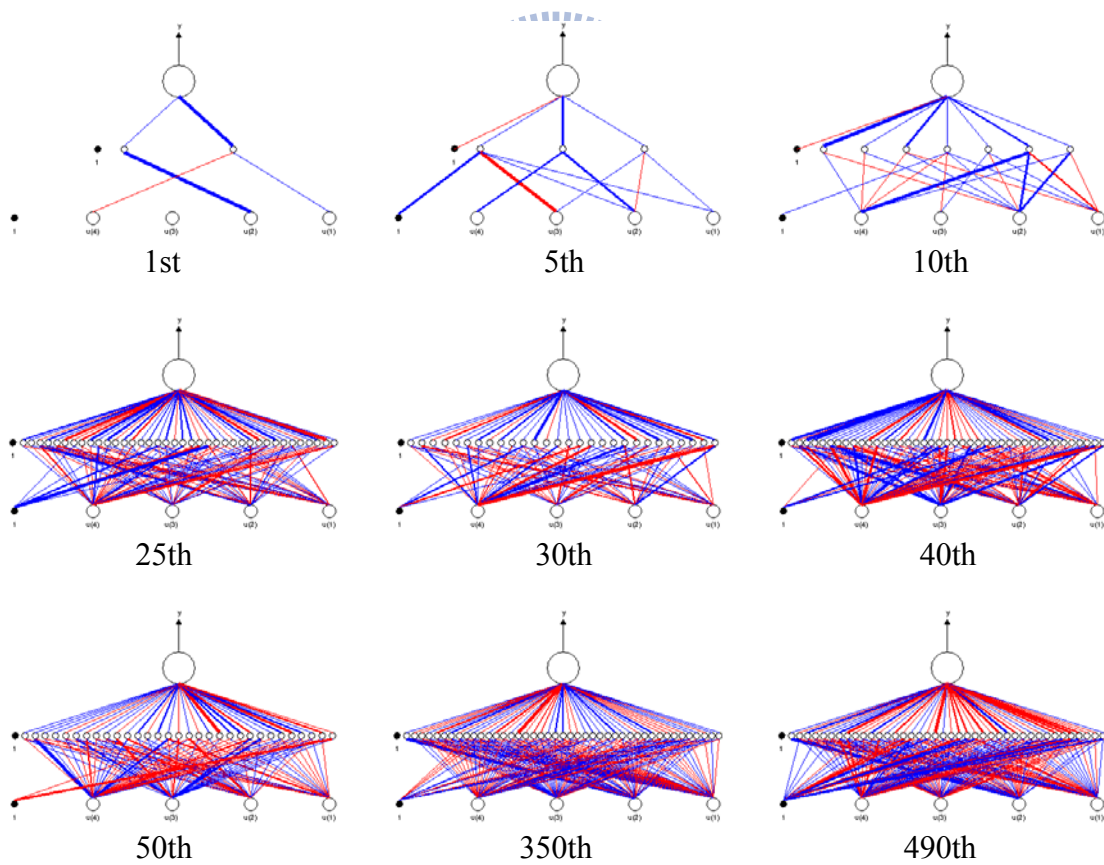


Fig. 4.6 Evolved NNs for Example 1..

Table 4.1 Prediction results for Example 1.

Method	t = 6		t = 84		N_h	N_c
	Train	Test	First 500 points	Last 500 points		
Du and Zhang [110]	2.87×10^{-4}	7.67×10^{-4}	1.93×10^{-2}	2.07×10^{-2}	294	---
Harpham and Dawson [112]	---	1.50×10^{-3}	---	---	116	---
Rojas et al. [113]	2.87×10^{-3}	---	2.63×10^{-2}	---	12	---
Chen et al. [114]	3.30×10^{-3}	3.60×10^{-3}	---	---	10	110
Cho and Wang [115]	9.60×10^{-3}	1.14×10^{-2}	---	---	23	---
ECPA	6.76×10^{-4}	6.30×10^{-4}	6.20×10^{-3}	3.10×10^{-3}	40.5	203.2

The prediction result for the one-step prediction of $x(t+6)$ is shown in Fig. 4.7. In addition to the one-step prediction of $x(t+6)$, the evolved NN was applied to another general testing case: the multiple-step prediction of $x(t+84)$ [111]. To perform a multiple-step prediction, the proposed algorithm iteratively predicts $x(t+6)$, $x(t+12)$, etc. until it reaches $x(t+84)$ after 14 such iterations. The prediction result for multiple-step prediction of $x(t+84)$ is shown in Fig. 4.8 where the NN was evolved based on the training data for the one-step prediction of $x(t+6)$. When compared with Fig. 4.7, the prediction error in Fig. 4.8 indicates an increase from 6.30×10^{-4} to 3.10×10^{-3} because multiple-step prediction is more complex than one-step prediction. As shown in Table 4.1, the prediction errors for multiple-step prediction of $x(t+84)$ for the first and last 500 points were 6.20×10^{-3} and 3.10×10^{-3} , respectively. Therefore, ECPA was superior to the other methods in the multiple-step prediction.

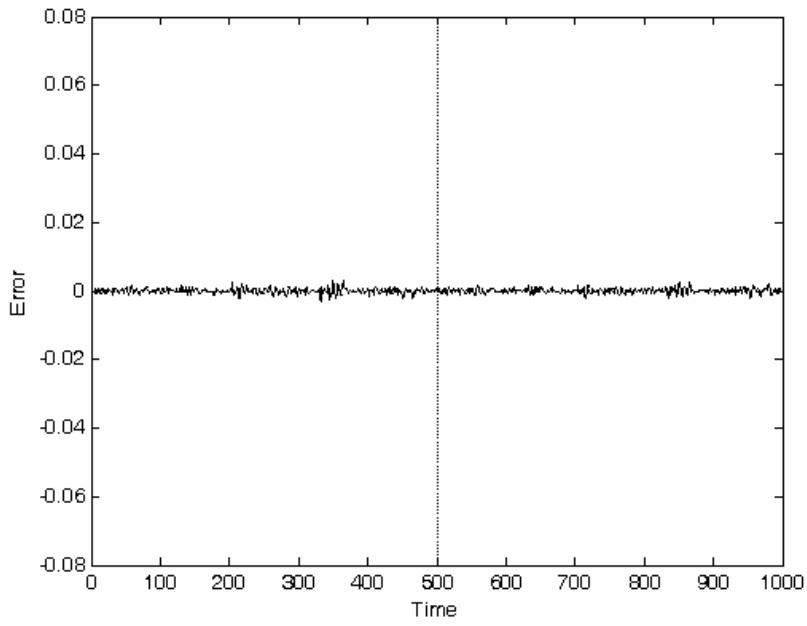


Fig. 4.7 Prediction error for Example 1.

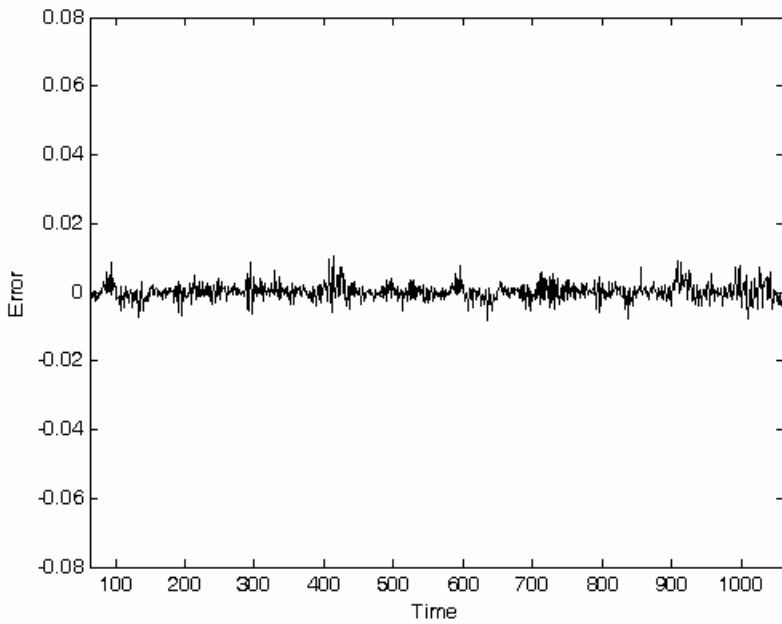


Fig. 4.8 Multiple-step prediction error for Example 1.

4.3.2 Example 2: Forecasting the Number of Sunspots

The number of sunspots varies nonlinearly in nonstationary and non-Gaussian cycles that are difficult to predict [116]. In this experiment, ECPA was used to predict the number of sunspots. The objective of this test involves using $[x(t-10) x(t-9) \dots x(t)]$ to predict $x(t+1)$, where t represents the year and $x(t)$ represents the number of sunspots in year t . For a fair comparison with the other methods, the data from 1700 to 1920 were the training set and the data from 1921 to 1955 were the testing set. The parameters used in this experiment are the same as those in Example 1.

The learning curves for ECPA in this example are shown in Fig. 4.9. The top panel shows that the training error gradually decreased as the evolution process progressed. The middle and bottom panels of Fig. 4.9 present the evolution of N_h and N_c , respectively. To illustrate the structural evolution of NNs in detail, Fig. 4.10 shows the topologies of NNs in selected generations. The input vector $[u(11) u(10) \dots u(1)]$ represents $[x(t-10) x(t-9) \dots x(t)]$, and the output y represents $x(t+1)$. We observed that N_h was always less than and equal to 5 during the evolutionary process and only a few inputs were connected to the hidden neurons. During the 1st and 40th generations, the structures of the evolved NNs grew rapidly, and more inputs were processed. The N_h increased to 4 in the 40th generation and N_c increased to 11. Finally, N_h and N_c converged to 5 and 24, respectively, at the end of the evolution process. Notably, all of the evolved NNs lack connections from the bias of the hidden layer to the output layer. The final evolved NN connected half of the inputs including $x(t)$, $x(t-2)$, $x(t-4)$, $x(t-8)$, $x(t-9)$, and $x(t-10)$, and each neuron connected an average of 3.8 inputs. Thus, the final evolved NN clearly has a partially connected topology.

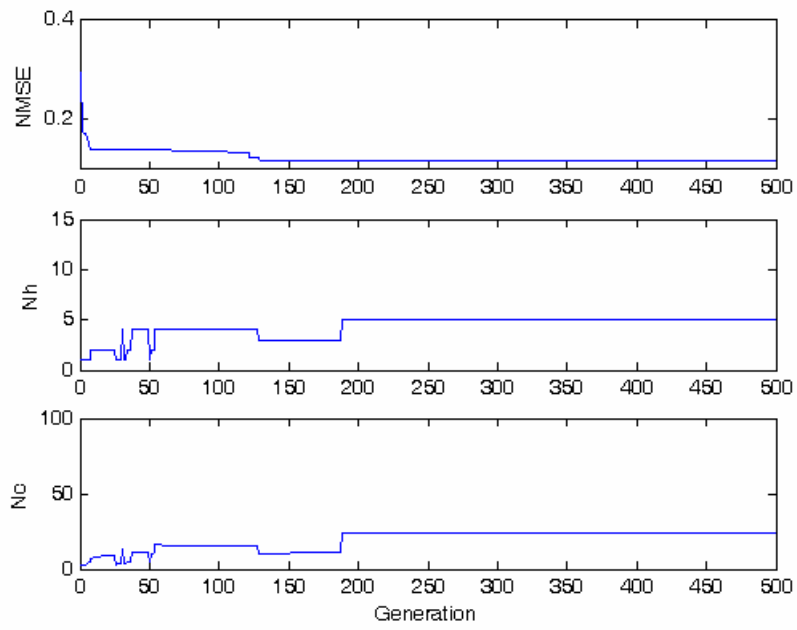


Fig. 4.9 Evolution progress for Example 2.

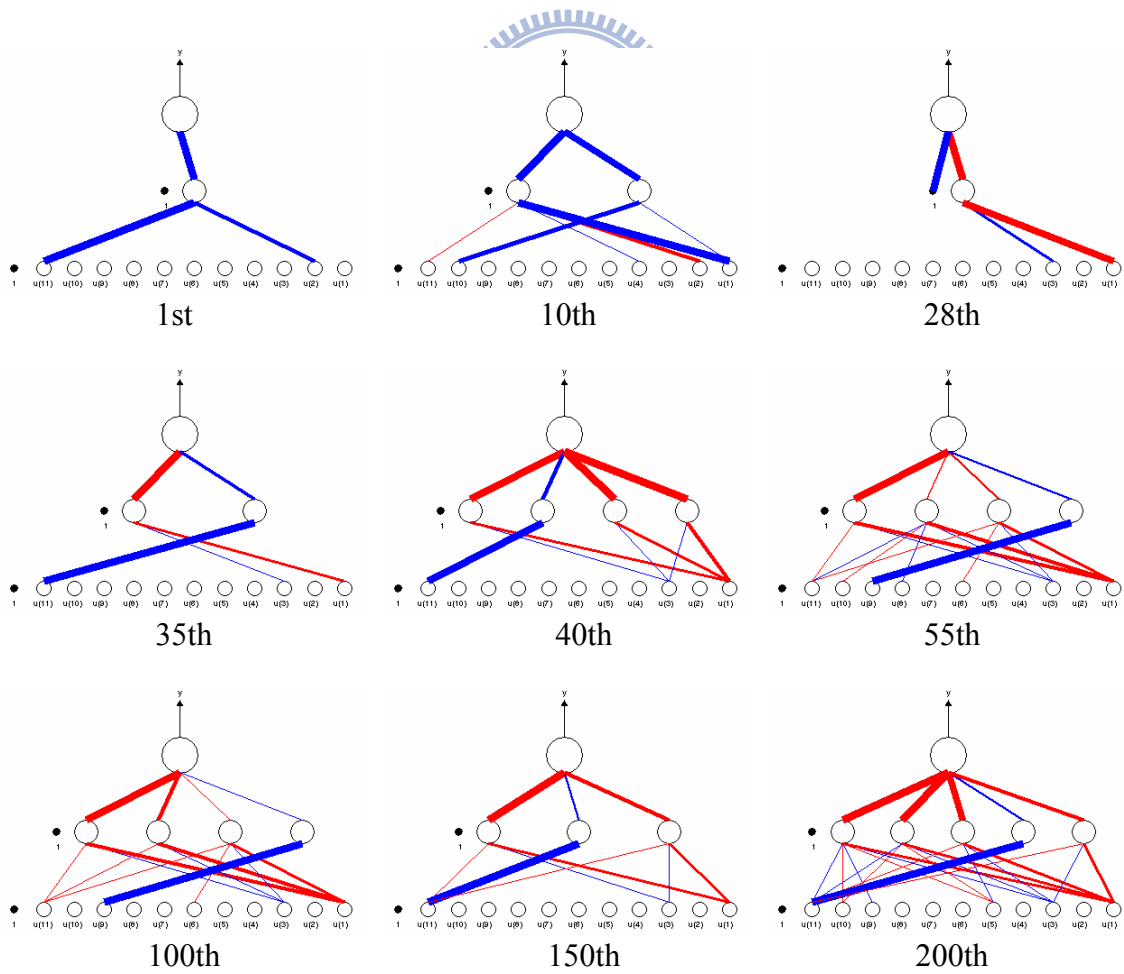


Fig. 4.10 Evolved NNs for Example 2.

The prediction result of the training data from the years 1700–1920 is shown in Fig. 4.11. The prediction result for the testing data for the years 1921–1955 is shown in Fig. 4.12. ECPA performance was further compared to those of an adaptive neural network (ADNN) [117], an artificial neural network (NN) [8], a hybrid methodology that combines an autoregressive integrated moving average with an artificial neural network (Hybrid) [118], and an adaptive k-nearest neighbors (AKN) [119]. In terms of average performance over 10 independent runs, Table 4.2 presents that ECPA obtained an NN with mean absolute percentage error (MAPE), normalized mean squared error (NMSE), N_h , and N_c values of 24.66, 0.0573, 5.0, and 46.9, respectively. These results indicate that the proposed ECPA can design an NN with a compact structure and a smaller prediction error than other methods.

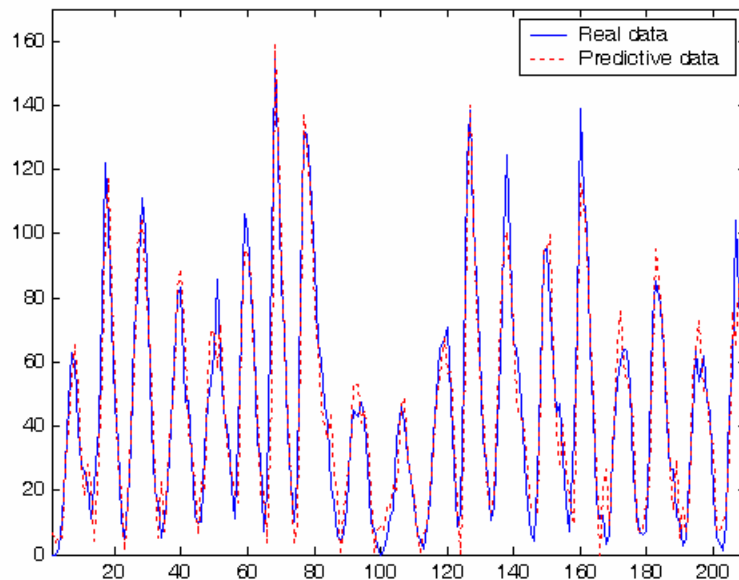


Fig. 4.11 Training results for Example 2.

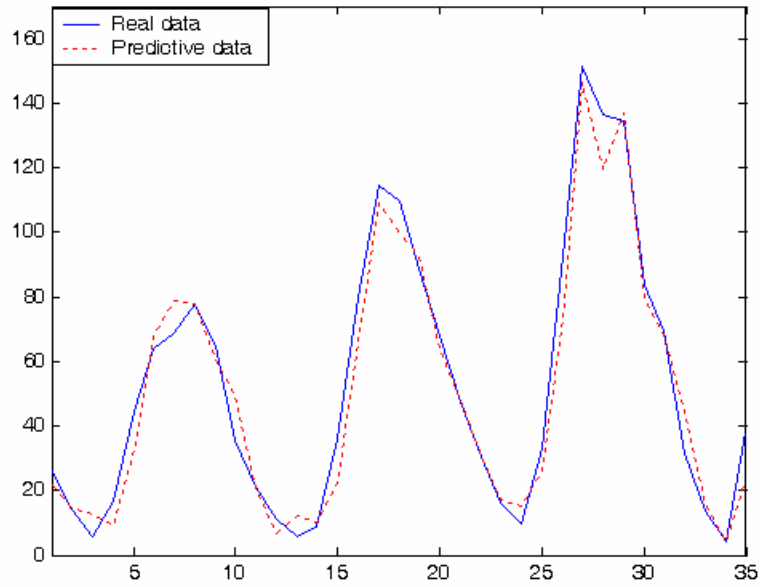


Fig. 4.12 Testing results for Example 2.

Table 4.2 Prediction results for Example 2.

Method	MAPE	NMSE	RMSE	N_h	N_c
ADNN [117]	28.45	0.068	9.233	6	--
NN [8]	30.8	0.078	9.888	6	--
Hybrid [118]	31.2	0.0852	10.334	--	--
AKN [119]	50.3	0.1833	15.158	--	--
ECPA	24.66	0.0573	8.475	5.0	46.9

4.3.3 Example 3: Vehicle Count Prediction

The vehicle count data set was obtained from the hourly vehicle count for the Monash Freeway outside Melbourne in Victoria, Australia, beginning in August 1995. The objective of this example involves using $[x(t-15) x(t-14) \dots x(t)]$ to predict $x(t+1)$. The setup of this experiment was identical to that described in Example 1.

The evolutionary progress of NNs is shown in Fig. 4.13. The top panel of Fig. 4.13 shows the NMSE of the evolved NNs in each generation. The NMSE was observed to

decrease rapidly in the first 50 generations and converged in the later generations. The middle panel of Fig. 4.13 shows the number of hidden neurons, N_h , of the evolved NNs. The bottom panel of Fig. 4.13 shows the total number of connections in the evolved NN N_c each generation. The structural evolution of NNs is presented in more detail in Fig. 4.14, which graphically illustrates the topologies of NNs in selected generations. The 16 inputs $[u(16) \ u(15) \ \dots \ u(1)]$ represent $[x(t-15) \ x(t-14) \ \dots \ x(t)]$, and the output y represents $x(t+1)$. We observed that N_h increased to 4 in the 4th generation and further increased to 9 in the 8th generation. After the 9th generation, N_h varied from 7 to 13. Finally, N_h converged to 7 in the 373rd generation. With respect to N_c , NNs have few connections in the early generations due to the single-connection topology presented in the initial population, and N_c does not always increase. N_c drops from 37 to 22 in the 20th generation and from 40 to 37 in the 240th generation, implying that more connections do not necessarily result in superior fitness. In other words, appropriate topology, not the number of connections, is the key to improving fitness. However, we observed that each neuron attempted to connect to more inputs as the number of generations was increased. The evolution of NNs almost converges, and no further improvement in NNs was observed after the 373rd generation. As a result, the highest-quality NN with well-trained weights has 7 hidden neurons and connects most of the inputs, except for $x(t-5)$ and $x(t-10)$. Thus, NN can automatically select the necessary inputs via ECPA. Clearly, the evolved NN is a partially connected network.

The prediction results of the training data and testing data are shown in Fig. 4.15 and Fig. 4.16, respectively. Table 4.3 summarizes the average performance of the evolved NNs for the testing set over 10 independent runs and compares these results to other methods, including ADNN [117], NN [8], Hybrid [118], and AKN [119]. The average NMSE using ECPA was 0.0182, which is less than that obtained using the other

methods. Furthermore, the evolved NNs obtained using ECPA have an average of 7.7 hidden neurons, which is less than the other methods.

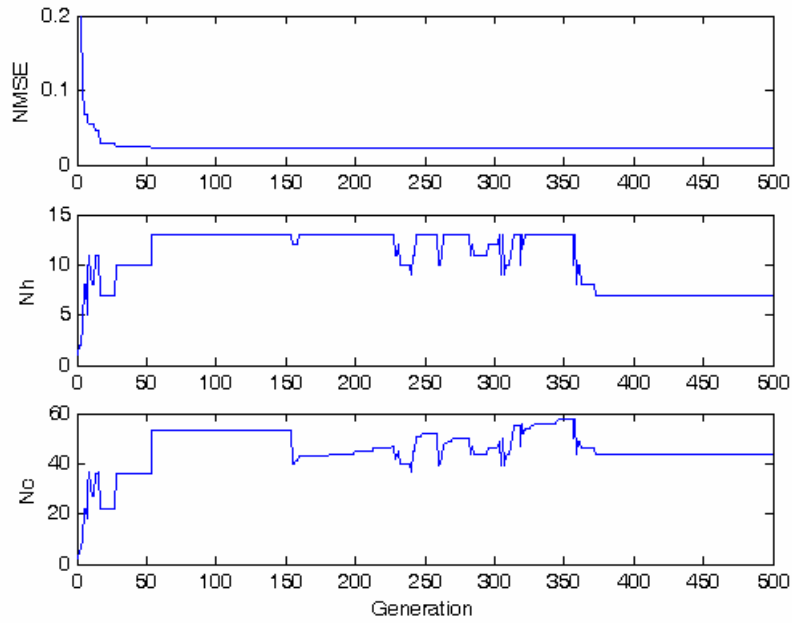


Fig. 4.13 Evolution progress for Example 3.

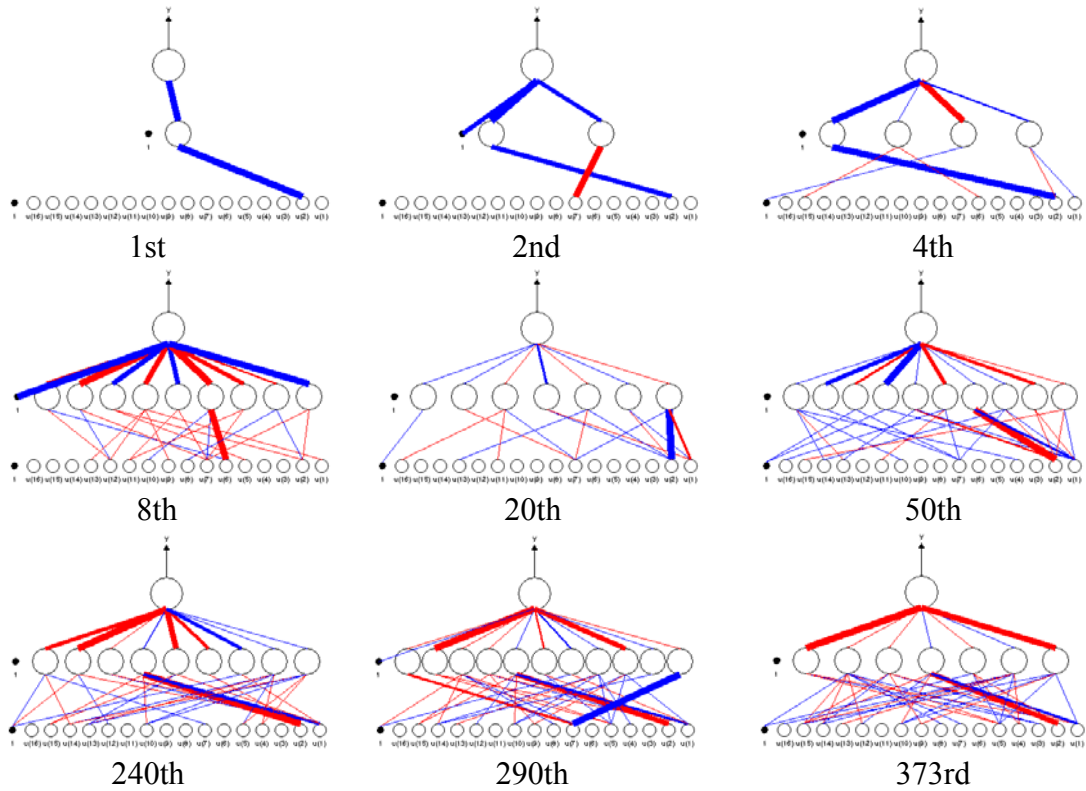


Fig. 4.14 Evolved NNs for Example 3.

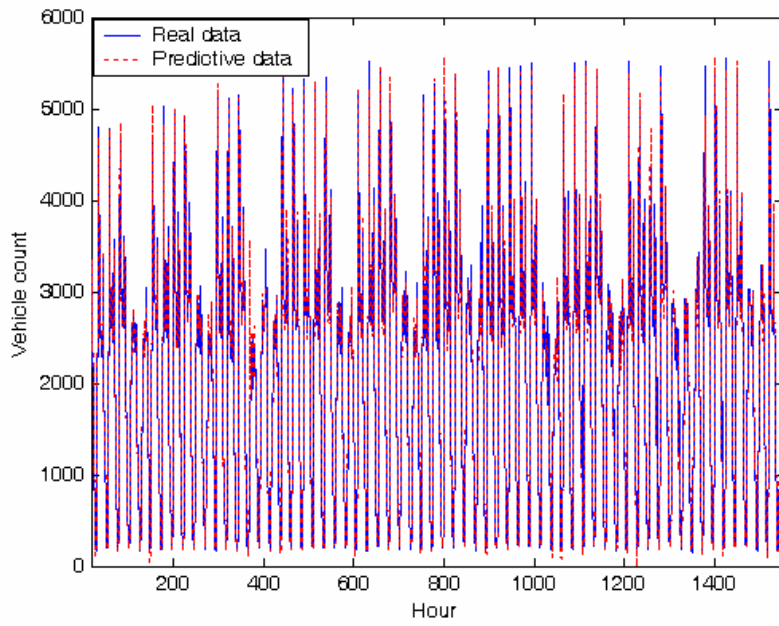


Fig. 4.15 Training results for Example 3.

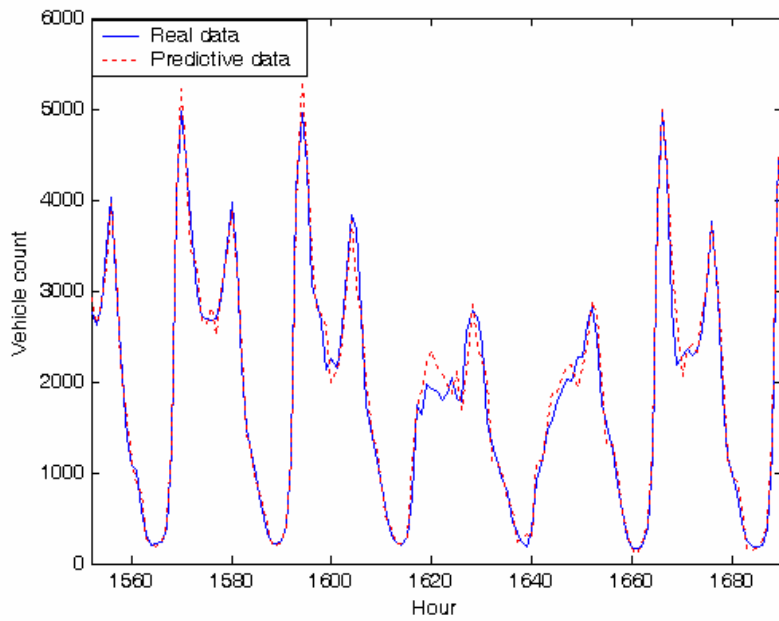


Fig. 4.16 Testing results for Example 3.

Table 4.3 Prediction results for the hourly vehicle count time series.

Method	MAPE	NMSE	RMSE	N_i	N_h	N_c
ADNN [117]	14.31	0.0193	186.096	180	12	--
NN [8]	17.97	0.0267	218.884	180	12	--
Hybrid [118]	26.98	0.0818	383.120	180	--	--
AKN [119]	17.39	0.0206	192.261	180	--	--
ECPA	11.35	0.0182	180.715	16	7.7	77.6

4.3.4 Effect of CBP and ABSS

The previous section discusses the performance of ECPA for different prediction problems. However, the effect of CBP and ABSS on evolution of NNs is unclear. To evaluate how CAP and ABSS affect NN evolution, two variants of ECPA which do not use CBP and ABSS, respectively, were used in repetitions of the above experiments. The variant of ECPA without CBP is referred to as ECPA/C and that without ABSS is referred to as ECPA/A. The setup of these experiments was identical to those in previous experiments.

In order to gain the deeper understanding of the performance difference between ECPA, ECPA/C, and ECPA/A in these three experiments, the three algorithms are compared in terms of NMSE, N_h , N_c , connection ratio, R_c , and computation time, T_c , whose unit is second. The R_c is determined as follows:

$$R_c = N_c / N_f \cdot 100\% \quad (4.14)$$

where N_f is the number of connections in an NN with a fully connected topology. An NN with N_h hidden nodes with a fully connected topology leads to the following relation:

$$\begin{aligned} N_f &= (N_i + 1) \cdot N_h + N_h + 1 \\ &= (N_i + 2) \cdot N_h + 1 \end{aligned} \quad (4.15)$$

where N_i is the number of input nodes. When $R_c < 100\%$, NN has a partially connected topology, and when $R_c = 100\%$, NN has a fully connected topology. The computational environment is Windows XP with Intel Core i7 870 2.93G CPU and 4GB RAM. These algorithms are implemented in MATLAB.

The results in Table 4.4 present that ECPA and ECPA/A produce different NNs in some aspects. For the three examples, the average N_h and N_c values over 10 independent runs returned by ECPA/A are much larger than those of ECPA which applies both CBP and ABSS. As comparing their prediction performance, ECPA/A yielded slightly smaller NMSE values than ECPA in the Mackey-Glass and vehicle count time series. This improvement may be yielded due to the great processing capability of a large number of hidden neurons. However, the NNs developed via ECPA/A for the sunspot time series have larger NMSE value than those via ECPA. This may be due to the overfitting property caused by too many hidden neurons. The results indicate that ECPA facilitates NNs with more generalization ability than ECPA/A.

Table 4.4 Performance of ECPA and ECPA/A in Mackey-Glass, sunspot, and vehicle count time series. All results are averaged over 10 independent runs, where * refers to RMSE.

Method	Experiment	NMSE	N_h	N_c	$R_c(\%)$	T_c
ECPA	Mackey-Glass	*6.3027×10 ⁻⁴	40.5	203.2	83.28	1112.7
	Sunspot	0.0573	5.0	46.9	71.06	375.1
	Vehicle count	0.0182	7.7	77.6	55.59	693.3
ECPA/C	Mackey-Glass	*1.3535×10 ⁻³	860.6	2357.7	45.65	60531.7
	Sunspot	0.5741	419.3	1145.6	21.01	5841.3
	Vehicle count	0.0358	224.3	605.1	14.98	5939.6
ECPA/A	Mackey-Glass	*6.3147×10 ⁻⁴	575.3	2173.9	62.96	12958.9
	Sunspot	0.7547	173.7	603.1	26.70	2237.1
	Vehicle count	0.0176	70.8	653.6	51.25	3358.3

In addition to NMSE, N_h , and N_c , Table 4.4 presents that R_c obtained by ECPA/C is lower than ECPA/A due to the use of ABSS. More specifically, elder NNs which are much likely to have more connections from inputs conducted by network mutation would be deleted by ABSS in ECPA/C. Although ECPA/C can produce sparsely connected topology of NN by the aid of ABSS, it would result in NNs with huge N_h . Thus, the NNs in ECPA/C face overfitting problem and have bad generalization ability, i.e., larger NMSE for testing set. When comparing ECPA, ECPA/C, and ECPA/A, ECPA/A can produce NNs with less hidden neurons than ECPA/C due to the use of CBP. ECPA further yields NNs with more compact structures and better generalization ability than ECPA/A due to the use of ABSS.

Furthermore, the computation time of ECPA, ECPA/C, and ECPA/A is compared. The computation time required by ECPA was less than that required by ECPA/C and ECPA/A since ECPA needs less computation time to process less hidden neurons while ECPA/C and ECPA/A require longer time to train and evaluate the NNs with large number of hidden neurons. According to the observation, both CBP and ABSS are beneficial for producing NNs with a compact structure and reducing computation time.

4.3.5 Discussion

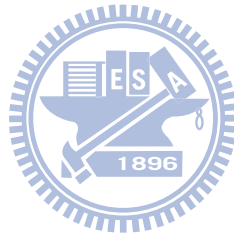
In this section, we summarize the observations in the three experiments described above, and discuss the experimental results. Fig. 4.5, Fig. 4.9, and Fig. 4.13 show that the NN structures developed using ECPA are simple in the first generations due to the use of initial NNs with one hidden neuron and a single connection to one of the inputs. As their evolution progresses, the NN structures grew rapidly in the beginning due to the addition of neurons via crossover and the addition of connections via mutation. However, the results show that NNs did not grow continuously in the later generations

due to the use of CBP and ABSS. CBP primarily preserves the significant neurons and prunes the insignificant neurons using a probability criterion. Pruning prevents the exponential growth of NNs and avoids long-term training for complex NNs. In addition, ABSS first deletes the old individuals likely to have complex structures and then provides an opportunity to introduce new individuals with simple structures generated via Step 1). Section 4.3.4 demonstrates that ABSS is useful for developing a compact NN architecture and avoiding the design of complex NNs. The highest-quality NN with well-trained weights is then attained using construction via crossover and mutation operations and destruction via CBP and ABSS. The resulting NNs do not have fully connected topologies; less than 80% of the synapses are connected in the Mackey-Glass time series, 50% are connected in the sunspot time series, and 40% are connected in the vehicle count time series. Furthermore, the evolved NNs do not connect to all the inputs, e.g., the evolved NN does not connect to $x(t-7)$ in the sunspot time series, and the evolved NN does not connect to $x(t-5)$ and $x(t-10)$ in the vehicle count time series. Thus, ECPA has the ability to select suitable inputs required to accurately perform predictions. These results imply that more connections do not necessarily result in superior fitness. In other words, an appropriately connected topology is the key to improving fitness, not the number of connections.

4.4 Summary

A novel structure learning algorithm, called ECPA, is proposed for the design of NNs based on an evolutionary constructive and pruning algorithm. ECPA evolves NNs starting with a minimal structure: one hidden neuron connected to an input node. The crossover and mutation operations make the NN structures more complex, whereas CBP

and ABSS make the NN structures more compact. The results of the numerical simulations show that the use of CBP and ABSS operations indeed generates compact NNs. Moreover, ECPA adopts variable-length chromosomes to represent the NNs so that memory is used efficiently. In the time series prediction problems, ECPA not only evolved partially connected NNs with sufficient prediction accuracy but also demonstrated the ability to select the proper inputs, i.e., input selection. The numerical results demonstrate that an appropriately connected topology, rather than the number of connections, is the key to improving NN performance.



Chapter 5 Conclusion and Future Work

This dissertation proposes an FNAGM based on GM(1,1) and NN for real-time prediction application. FNAGM uses GM(1,1) to predict signals and employs NN to compensate the prediction error of GM(1,1). Based on Levenberg-Marquardt algorithm, NN is adjusted in real-time by the proposed on-line batch training whose convergence property has been proven in this dissertation. Numerical results are also included to demonstrate that FNAGM has higher prediction accuracy than other methods and is applicable for real-time prediction. Furthermore, experimental results of a robotic application show that FNAGM can be successfully used for the trajectory prediction and object tracking.

In order to design the structure of FNAGM in an efficient way, a neuron-based structure learning algorithm, called SSLA, is proposed to construct the topology of FNAGM. The SSLA can flexibly construct cascade NNs and feedforward NNs by using the neuron population including neurons with hyperbolic tangent and linear activation functions. Therefore, a complex coding to represent these two classes of NNs is unnecessary. The neurons in the neuron population share the fitness from their participating NNs and then evolve according to neuron crossover, neuron mutation, and BP. Consequently, FNAGM with compact structure is obtained and applied to predict signal and further learn to compensate the prediction error by the on-line batch training. When comparing to the FNAGM designed by trial-and-error, numerical results show that FNAGM-SSLA can automatically determine the topology of FNAGM with more compact structure, higher prediction accuracy, and less computation time.

In addition to the neuron-based structure learning algorithm, ECPA, a

network-based structure learning algorithm, is proposed to determine the topology of NN by evolving network population. Different from SSLA which needs to define the minimum and maximum numbers of hidden neurons, ECPA simply starts from a minimum structure with one hidden neuron connected from one input node. ECPA incorporates the idea of constructive algorithm into the network crossover and mutation, and enables NNs to have more processing capabilities. Furthermore, CBP and ABSS, based on pruning algorithm, are adopted to make the structure of NN more compact. Numerical results show that ECPA can gradually evolve NNs from a minimum structure to an appropriate structure with suitable inputs. Moreover, the evolved NNs obtain higher prediction accuracy and more compact structures than other methods for prediction problems. Therefore, ECPA could be applied to extract the inputs which essentially have rare relationship to the original data, and thus reduce the problem dimensionality and eventually decrease the complexity of the generated NNs.

Future work will focus on developing cascade NNs with different activation functions, such as logistic, hyperbolic tangent, linear threshold, exponential and Gaussian signal function. Moreover, a large amount of computation time is required for the use of SSLA and ECPA. Approaches concerning the time reduction should be further investigated. In addition to prediction application, it would be of great interest to use reinforcement learning to develop both parameter and structure learning algorithms for nonlinear control problems. This dissertation adopts BP as the parameter learning algorithm; however, BP may be not applicable in certain control problems when gradient information of the plants is not available. In the future research, EAs will be considered as a substitute of BP to perform both parameter and structure learning by global searching techniques.

Bibliography

- [1] J. D. Markel and A. H. Gray, *Linear Prediction of Speech*. New York: Springer Verlag, 1976.
- [2] J. Makhoul, "Linear prediction: A tutorial review," *Proceedings of the IEEE*, vol. 63, pp. 561-580, Apr. 1975.
- [3] J. L. Deng, "Introduction to grey system theory," *The Journal of Grey System*, vol. 1, no. 1, pp. 1-24, 1989.
- [4] K. Jinno, S. Xu, R. Berndtsson, A. Kawamura, and M. Matsumoto, "Prediction of sunspots using reconstructed chaotic system equations," *J. Geophys. Res.*, vol. 100, pp. 14 773-14 781, 1995.
- [5] M. Han, J. Xi, S. Xu, and F.-L. Yin, "Prediction of chaotic time series based on the recurrent predictor neural network," *IEEE Trans. Signal Processing*, vol. 52, no. 12, pp. 3409-3416, 2004.
- [6] D. Massicotte, R. Morawski, and A. Barwicz, "Incorporation of a positivity constraint into a Kalman-filter-based algorithm for correction of spectrometric data," *IEEE Trans. Instrumentation and Measurement*, vol. 44, no. 1, pp. 2-7, 1995.
- [7] H. Yoo and R. L. Pimmel, "Short term load forecasting using a self-supervised adaptive neural network," *IEEE Trans. Power Systems*, vol. 14, no. 2, pp. 779-784, 1999.
- [8] M. Adya and F. Collopy, "How effective are neural networks at forecasting and prediction? A review and evaluation," *Journal of Forecasting*, vol. 17, pp. 481-495, 1998.
- [9] C. M. Zealand, D. H. Burn, and S. P. Simonovic, "Short term streamflow forecasting using artificial neural networks," *Journal of Hydrology*, vol. 214, pp. 32-48, 1999.

- [10] S. H. Yang and Y. P. Chen, "Intelligent forecasting system using grey model combined with neural network," *International Journal of Fuzzy Systems*, vol. 13, no. 1, pp. 8-15, 2011.
- [11] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Prentice-Hall, Upper Saddle River, NJ, USA, 1992, ch2.
- [12] S. Haykin, *Neural Network—A Comprehensive Foundation*. Prentice Hall PTR, 1994, ch. 6.
- [13] D. Saad, *On-Line Learning in Neural Networks*. Cambridge University Press, 1998, ch. 1.
- [14] A. Browne, *Neural Network Analysis, Architectures, and Applications*. CRC Press, 1997, ch6.
- [15] K. J. Hunt, G. W. Irwin, and K. Warwick, *Neural Network Engineering in Dynamic Control Systems*. Springer-Verlag, London, 1995, ch12.
- [16] R. Hoptroff, "The principles and practice of time series forecasting and business modelling using neural nets," *Neural Computing and Applications*, vol. 1, pp. 59-66, 1993.
- [17] F. Wong, "Time series forecasting using backpropagation networks," *Neurocomputing*, vol. 2, no. 4, pp. 147-159, 1991.
- [18] S. Kang and C. Isik, "Partially connected feedforward neural networks structured by input types," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 175-184, 2005.
- [19] F. Kamran, R. G. Harley, B. Burton, T. G. Habetler, and M. A. Brooke, "A fast on-line neural-network training algorithm for a rectifier regulator," *IEEE Trans. on Power Electronics*, vol. 13, no. 2, pp. 366-371, 1998.
- [20] L. Grippo, "Convergent on-line algorithms for supervised learning in neural networks," *IEEE Trans. on Neural Netw.*, vol. 11, no. 6, pp. 1284-1299, 2000.
- [21] G. L. Plett, "Adaptive inverse control of linear and nonlinear systems using dynamic neural networks," *IEEE Trans. on Neural Netw.*, vol. 14, no. 2, pp. 360-376, 2003.

- [22] D. Saad and S. A. Solla, "Dynamics of on-line gradient descent learning for multilayer neural networks," In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, vol. 8, pp. 302-308, Cambridge, MA. MIT Press, 1996.
- [23] D. Saad and M. Rattray, "Globally optimal parameters for on-line learning in multilayer neural networks," *Phys. Rev. Lett.*, vol. 79, pp. 2578-2581, 1997.
- [24] J. L. Deng, "Control problems of grey systems," *System & Control Letters*, vol. 1, no. 5, pp. 288-294, 1982.
- [25] W. L. Yao, S. C. Chi, and J. H. Chen, "An improved grey-based approach for electricity demand forecasting," *Electric Power Systems Research*, vol. 67, no. 3, pp. 217-224, 2003.
- [26] B. Ulutas, E. Erdemir, and K. Kawamura, "Application of a hybrid controller with non-contact impedance to a humanoid robot," in *International Workshop on Variable Structure Systems*, pp. 378-383, 2008.
- [27] H. C. Ting, J. L. Chang, C. H. Yeh, and Y. P. Chen, "Discrete time sliding-mode control design with grey predictor," *International Journal of Fuzzy Systems*, vol. 9, no. 3, pp. 179-185, 2007.
- [28] S. Fan, Y. Fang, W. Li, Y. Ma, and T. Xiao, "The combination of grey system and BP neural network," in *International Conference on Mechatronics and Automation*, pp. 1267-1271, 2007.
- [29] C. C. Chiang, M. C. Ho, and J. A. Chen, "A hybrid approach of neural networks and grey modeling for adaptive electricity load forecasting," *Neural Computing & Applications*, vol. 15, no. 3, pp. 328-338, 2006.
- [30] F. Wang and H. Xia, "Network traffic prediction based on grey neural network integrated model," in *International Conference on Computer Science and Software Engineering*, pp. 915-918, 2008.
- [31] C. Zhu and Q. Ju, "United grey system-neural network model and its application in prediction of groundwater level," in *International Conference on Industrial Mechatronics and Automation*, pp. 434-437, 2009.

- [32] P. Fu and Y. Li, "Application of combined model in forecasting logistic volume of a port," in *Intelligent Computation Technology and Automation*, pp. 742–745, 2010.
- [33] D. Zhang, Z. Ren, Y. Bi, D. Zhou, and Y. Bi, "Power load forecasting based on grey neural network," in *IEEE International Symposium on Industrial Electronics*, pp. 1885–1889, 2008.
- [34] S. H. Wang, R. L. Hao, Y. J. Chang, and Y. Zhao, "Research of short-term load forecasting based on combined grey neural network and phase space reconstruction," in *International Conference on Machine Learning and Cybernetics*, pp. 1194–1199, 2009.
- [35] X. Zhu, "Application of composite grey BP neural network forecasting model to motor vehicle fatality risk," in *International Conference on Computer Modeling and Simulation*, pp. 236–240, 2010.
- [36] B. R. Chang and H. F. Tsai, "Forecast approach using neural network adaptation to support vector regression grey model and generalized auto-regressive conditional heteroscedasticity," *Expert Systems with Applications*, vol. 34, no. 2, pp. 925–934, 2008.
- [37] C. C. Hsu and C. Y. Chen, "Applications of improved grey prediction model for power demand forecasting," *Energy Conversion and Management*, vol. 44, no. 14, pp. 2241–2249, 2003.
- [38] H. Liu, L. Cai, and X. Wu, "Grey-RBF neural network prediction model for city electricity demand forecasting," in *International Conference on Wireless Communications, Networking and Mobile Computing*, pp.1–5, 2008.
- [39] S. H. Yang and Y. P. Chen, "Intelligent forecasting system based on grey model and neural network," in *IEEE/ASME International Conf. on Advanced Intelligent Mechatronics*, pp. 699–704, 2009.
- [40] M. F. Yeh, C. T. Chang, and M. S. Leu, "Financial distress prediction model via GreyART network and grey model," *Advances in Neural Network Research and Applications*, vol. 67, no. 1, pp. 91–100, 2010.

- [41] J. Tanomaru and S. Omatu, "Process control by on-line trained neural controller," *IEEE Trans. on Industrial Electronics*, vol. 39, no. 6, pp. 511–521, 1992.
- [42] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Contr., Signals, Syst.*, vol. 2, no. 4, pp. 303-314, 1989.
- [43] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.
- [44] S. T. Chen, D. C. Yu, and A. R. Moghaddamjo, "Weather sensitive short-term load forecasting using nonfully connected artificial neural network," *IEEE Trans. on Power Systems*, vol. 7, no. 3, pp. 1098–1105, 1991.
- [45] A. Canning and E. Gardner, "Partially connected models of neural networks," *J. Phys. A*, vol. 21, pp. 3275–3284, 1988.
- [46] S. H. Yang and Y. P. Chen, "Symbiotic neuron evolution of a neural-network-aided grey model for time series prediction," *IEEE International Conference on Fuzzy Systems*, pp. 195-201, 2011.
- [47] D. Elizondo and E. Fiesler, "A survey of partially connected neural networks," *Int. J. Neural Syst.*, vol. 8, no. 5 and 6, pp. 535-558, 1997.
- [48] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 79- 88, Jan 2003.
- [49] T. Y. Kwok and D. Y. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 630–645, May 1997.
- [50] R. Reed, "Pruning algorithms—A survey," *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, Sep. 1993.
- [51] A. P. Engelbrecht, "A new pruning heuristic based on variance analysis of sensitivity information," *IEEE Trans. Neural Netw.*, vol.12, no.6, pp.1386–1399, Nov. 2001.
- [52] Jocelyn Sietsma, Robert J.F. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, no. 1, pp. 67–79, 1991.

- [53] Y.-C. Ho, "Perturbation analysis explained," *IEEE Trans. Automat. Contr.*, vol. 33, pp. 761–763, 1988.
- [54] J. M. Holtzman, "On using perturbation analysis to do sensitivity analysis: Derivatives versus differences," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 243–247, 1992.
- [55] M. Hagiwara, "Removal of hidden units and weights for back propagation networks," *International Joint Conference on Neural Networks*, pp. 351–354, Oct. 1993.
- [56] Y. Hirose, K. Yamashita, and S. Hijiya, "Back-propagation algorithm which varies the number of hidden units," *Neural Netw.*, vol. 4, no. 1, pp. 61–66, 1991.
- [57] Md. M. Islam and K. Murase, "A new algorithm to design compact twohidden-layer artificial neural networks," *Neural Netw.*, vol. 14, no. 9, pp. 1265–1278, 2001.
- [58] I. Rivals and L. Personnaz, "Neural-network construction and selection in nonlinear modeling," *IEEE Trans. Neural Netw.*, vol. 14, no. 4, pp. 804–819, 2003.
- [59] M. M. Islam, M. A. Sattar, M. F. Amin, X. Yao, and K. Murase, "A new adaptive merging and growing algorithm for designing artificial neural networks," *IEEE Trans. Syst., Man, Cybern. B*, vol. 39, no. 3, pp. 705–722, 2009.
- [60] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. New York: IEEE Press, 1995.
- [61] D. Whitley and C. Bogart, "The evolution of connectivity: Pruning neural networks using genetic algorithms," *International Joint Conference on Neural Networks*, pp. 134–137, 1990.
- [62] D. White and P. Ligomenides, "GANNet: A genetic algorithm for optimizing topology and weights in neural network design," in *International Workshop on Artificial Neural Networks, in New Trends in Neural Computation*, J. Mira, J. Cabestany, and A. Prieto, Eds. Berlin, Germany: Springer-Verlag, pp. 332–327, 1993.

- [63] C. Zanchettin, T. B. Ludermir, and L. M. Almeida, "Hybrid Training Method for MLP: Optimization of Architecture and Training," *IEEE Trans. Syst., Man, Cybern. B*, vol. 41, no. 4, pp. 1097-1109, 2011.
- [64] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
- [65] P. A. Gutiérrez, C. Hervás-Martínez, F. J. Martínez-Estudillo, "Logistic Regression by Means of Evolutionary Radial Basis Function Neural Networks," *IEEE Trans. Neural Netw.*, vol.22, no.2, pp.246-263, Feb. 2011.
- [66] T. H. Oong and N. A. M. Isa, "Adaptive Evolutionary Artificial Neural Networks for Pattern Classification," *IEEE Trans. Neural Netw.*, vol.22, no.11, pp.1823-1836, Nov. 2011.
- [67] J. C. F. Caballero, F. J. Martinez, C. Hervas, and P. A. Gutierrez, "Sensitivity versus accuracy in multiclass problems using memetic Pareto evolutionary neural networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 5, pp. 750–770, May 2010.
- [68] D. Mantzaris, G. Anastassopoulos, and A. Adamopoulos, "Genetic algorithm pruning of probabilistic neural networks in medical disease estimation," *Neural Netw.*, vol. 24, no. 8, pp. 831-835, October 2011.
- [69] B. Curry and P. H. Morgan, "Seasonality and neural networks: a new approach," *International Journal of Metaheuristics*, vol. 1, no. 2, pp. 181 - 197, 2010.
- [70] D.-S. Huang and J.-X. Du, "A Constructive Hybrid Structure Optimization Methodology for Radial Basis Probabilistic Neural Networks," *IEEE Trans. Neural Netw.*, vol.19, no.12, pp.2099-2115, Dec. 2008.
- [71] T. A.S. Masutti and L. N. de Castro, "Neuro-immune approach to solve routing problems," *Neurocomputing*, vol. 72, no. 10-12, pp. 2189-2197, June 2009.
- [72] A. Kaylani, M. Georgiopoulos, M. Mollaghasemi, and G.C. Anagnostopoulos, "AG-ART: An adaptive approach to evolving ART architectures," *Neurocomputing*, vol. 72, no. 10-12, pp. 2079-2092, June 2009.
- [73] C.-K. Goh, E.-J. Teoh, and K. C. Tan, "Hybrid Multiobjective Evolutionary Design for Artificial Neural Networks," *IEEE Trans. Neural Netw.*, vol.19, no.9, pp.1531-1548, Sept. 2008.

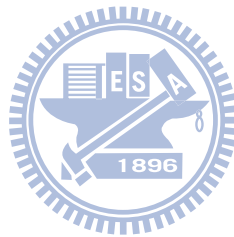
- [74] C. Hervás-Martínez, F. J. Martínez-Estudillo, and M. Carbonero-Ruz, “Multilogistic regression by means of evolutionary product-unit neural networks,” *Neural Netw.*, vol. 21, no. 7, pp. 951-961, September 2008.
- [75] P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.
- [76] P. P. Palmes, T. Hayasaka, and S. Usui, “Mutation-based genetic neural network,” *IEEE Trans. Neural Netw.*, vol. 16, no. 3, pp. 587–600, 2005.
- [77] T. B. Ludermir, A. Yamazaki, and C. Zanchettin, “An optimization methodology for neural network weights and architectures,” *IEEE Trans. Neural Netw.*, vol.17, no.6, pp.1452-1459, 2006.
- [78] J.-Y. Lin and Y.-P. Chen, “Analysis on the Collaboration Between Global Search and Local Search in Memetic Computation,” *IEEE Trans. Evol. Comput.*, vol.15, no.5, pp.608-623, 2011.
- [79] M. S. Alam, M. M. Islam, X. Yao, and K. Murase, “Recurring Two-Stage Evolutionary Programming: A Novel Approach for Numeric Optimization,” *IEEE Trans. Syst., Man, Cybern. B*, vol.41, no.5, pp.1352-1365, 2011.
- [80] W. E. Hart, “Adaptive global optimization with local search,” Ph.D. dissertation, Dept. Comput. Sci. Eng., Univ. California, San Diego, 1994.
- [81] G. Zhang, B. E. Patuwo, and M. Y. Hu, “Forecasting with artificial neural networks: The state of the art,” *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.
- [82] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Prentice-Hall, Upper Saddle River, NJ, USA, 1992, ch2.
- [83] M. T. Hagan and M. B. Menhaj, “Training feedforward networks with the Marquardt algorithm,” *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp.989–993, 1994.
- [84] Y. H. Pao, *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley Publishing Co., Reading, MA, USA, 1989.

- [85] D. Gorinevsky, “An approach to parametric nonlinear least square optimization and application to task-level learning control,” *IEEE Trans. on Automatic Control*, vol. 42, no. 7, pp.912–927, 1997.
- [86] M. C. Mackey and L. Glass, “Oscillation and chaos in physiological control systems,” *Science*, vol. 197, no. 4300, pp. 287–289, 1977.
- [87] S. H. Yang, C. Y. Ho, and Y. P. Chen, “Neural network based stereo matching algorithm utilizing vertical disparity,” in *IECON 2010*, pp.1155-1160, 2010.
- [88] S. H. Yang, C. Y. Ho and Y. P. Chen, “Intelligent stereo matching algorithm based on Hopfield neural network and genetic algorithm,” *Far East Journal of Experimental and Theoretical Artificial Intelligence*, vol. 6, no. 1-2, pp. 1-23, 2011.
- [89] E. Kayacan and O. Kaynak, “An adaptive grey PID-type fuzzy controller design for a non-linear liquid level system,” *Trans. Inst. Meas. Control*, vol. 31, no. 1, pp. 33-49, 2009.
- [90] R.E. Smith, S. Forrest and A.S. Perelson, “Searching for diverse, cooperative populations with genetic algorithms,” *Evol. Comput.*, vol. 1, no. 2, pp. 127–149, 1993.
- [91] D. E. Moriarty and R. Miikkulainen, “Efficient reinforcement learning through symbiotic evolution,” *Mach. Learn.*, vol. 22, pp. 11–32, 1996.
- [92] C. F. Juang, J. Y. Lin, and C. T. Lin, “Genetic reinforcement learning through symbiotic evolution for fuzzy controller design,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 30, no. 2, pp. 290–302, 2000.
- [93] C. J. Lin and Y. J. Xu, “A self-adaptive neural fuzzy network with groupbased symbiotic evolution and its prediction applications,” *Fuzzy Sets Syst.*, vol. 157, no. 8, pp. 1036–1056, 2006.
- [94] C. J. Lin, C. H. Chen, and C. T. Lin, “Efficient self-evolving evolutionary learning for neurofuzzy inference systems,” *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 6, pp. 1476–1490, 2008.
- [95] J.-J. Hu and T.-H. S. Li, “Genetic regulatory network-based symbiotic evolution,” *Expert Systems with Applications*, vol. 38, no. 5, pp. 4756-4773, 2011.

- [96] J.-J. Hu, T.-H. S. Li, and Y.-T. Su, "A novel particle swarm-based symbiotic evolutionary algorithm for a class of multi-modal functions," *International Journal of Innovative Computing, Information and Control*, vol. 7, no. 4, pp. 1905-1920, 2011.
- [97] Y.-C. Hsu and S.-F. Lin, "Reinforcement group cooperation-based symbiotic evolution for recurrent wavelet-based neuro-fuzzy systems," *Neurocomputing*, vol. 72, no. 10-12, pp. 2418-2432, 2009.
- [98] Y.-C. Hsu, S.-F. Lin, and Y.-C. Cheng, "Multi groups cooperation based symbiotic evolution for TSK-type neuro-fuzzy systems design," *Expert Systems with Applications*, vol. 37, no. 7, pp. 5320-5330, 2010.
- [99] S. H. Yang and Y. P. Chen, "Intelligent forecasting system using grey model combined with neural network," *International Journal of Fuzzy Systems*, vol. 13, no. 1, pp. 8-15, 2011.
- [100] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536, Oct. 9, 1986.
- [101] D. Dumitrescu, B. Lazzarini, L. C. Jain, and A. Dumitrescu, *Evolutionary Computation*. Boca Raton, FL: CRC Press, 2000, ch4.
- [102] K. Deb and R. B. Agrawal, Simulated binary crossover for continuous search space, in *Complex Syst.*, vol. 9, pp. 115-148, 1995.
- [103] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35-62, 1998.
- [104] C. Zanchettin, T. B. Ludermir, and L. M. Almeida, "Hybrid Training Method for MLP: Optimization of Architecture and Training," *IEEE Trans. Syst., Man, Cybern. B*, vol. 41, no. 4, pp. 1097-1109, 2011.
- [105] J. M. Zurada, A. Malinowski, and I. Cloete, "Sensitivity analysis for minimization of input data dimension for feedforward neural network," *International Symposium on Circuits and Systems*, pp. 447-450, 1994.

- [106] J. M. Zurada, A. Malinowski, and S. Usui, "Perturbation method for deleting redundant inputs of perceptron networks," *Neurocomputing*, vol. 14, no. 2, pp. 177–193, 1997.
- [107] A. P. Engelbrecht and I. Cloete, "A sensitivity analysis algorithm for pruning feedforward neural networks," *International Conference on Neural Networks*, pp. 1274–1278, 1996.
- [108] Tukey, J. W. *Exploratory Data Analysis*. Addison Wesley, Reading, MA, 1977.
- [109] A. D. Cioppa, C. D. Stefano, and A. Marcelli, "On the role of population size and niche radius in fitness sharing," *IEEE Trans. Evol. Comput.*, vol.8, no.6, pp. 580- 592, 2004.
- [110] H. Du and N. Zhang, "Time series prediction using evolving radial basis function networks with new encoding scheme," *Neurocomputing*, vol. 71, no. 7-9, pp. 1388-1400, 2008.
- [111] J.-S. R. Jang, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 3, pp. 665–685, 1993.
- [112] C. Harpham and C.W. Dawson, "The effect of different basis functions on a radial basis function network for time series prediction: a comparative study," *Neurocomputing*, vol. 69, no. 16-18, pp. 2161-2170, 2006.
- [113] I. Rojas, H. Pomares, J.L. Bernier, J. Ortega, B. Pino, F.J. Pelayo and A. Prieto, "Time series analysis using normalized PG-RBF network with regression weights," *Neurocomputing*, vol. 42, no. 1-4, pp. 267-285, 2002.
- [114] Y. Chen, B. Yang and J. Dong, "Time-series prediction using a local linear wavelet neural network," *Neurocomputing*, vol. 69, no. 4-6, pp. 449-465, 2006.
- [115] K. B. Cho and B. H. Wang, "Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction," *Fuzzy Sets Syst.*, vol. 83, no. 3, pp. 325-339, 1996.
- [116] S. H. Ling, F. H. F. Leung, H. K. Lam, Y. S. Lee and P. K. S. Tam, "A novel genetic-algorithm-based neural network for short-term load forecasting," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 4, pp. 793–799, 2003.

- [117] W. K. Wong, M. Xia, W. C. Chu, “Adaptive neural network model for time-series forecasting,” *European Journal of Operational Research*, vol. 207, no. 2, pp. 807–816, 2010.
- [118] G. P. Zhang, “Time series forecasting using a hybrid ARIMA and neural network model,” *Neurocomputing*, vol. 50, pp. 159–175, 2003.
- [119] M. Kulesh, M. Holschneider, and K. Kurennaya, “Adaptive metrics in the nearest neighbours method,” *Physics D*, vol. 237, pp. 283–291, 2008.



Vita

博士候選人學經歷資料

姓名：楊世宏 (Shih-Hung Yang)

性別：男

生日：民國 69 年 9 月 21 日

出生地：台北市

論文題目

中文： 具演化式結構學習能力之類神經網路及其預測之應用

英文： Neural Network with Evolutionary Structure Learning and Its Prediction
Application



學歷：

- 民國 91 年 6 月，國立交通大學機械工程學系畢業
- 民國 93 年 6 月，國立交通大學電機與控制工程研究所畢業
- 民國 100 年 10 月，國立交通大學電控工程研究所，提博士論文口試

Publication List

著作目錄

姓名：楊世宏 (Shih-Hung Yang)

Journal Paper

- [1] **S. H. Yang** and Y. P. Chen, “Intelligent forecasting system using grey model combined with neural network,” *International Journal of Fuzzy Systems*, vol. 13, no. 1, pp. 8-15, 2011. (2 點)
- [2] **S. H. Yang**, C. Y. Ho and Y. P. Chen, “Intelligent stereo matching algorithm based on Hopfield neural network and genetic algorithm,” *Far East Journal of Experimental and Theoretical Artificial Intelligence*, vol. 6, no. 1-2, pp. 1-23, 2011. (1.4 點)
- [3] **S. H. Yang** and Y. P. Chen, “An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications,” *Neurocomputing*, revised.
- [4] **S. H. Yang** and Y. P. Chen, “Symbiotic structure learning algorithm for feedforward-neural-network-aided grey model and its prediction application,” submitted to *IEEE Trans. Neural Netw.*

Conference Paper

- [1] **S. H. Yang** and Y. P. Chen, “Evolution strategies-based learning control system,” in *Proc. of 2007 CACS Int. Automatic Control Conf.*, pp. 230-235, 2007.
- [2] **S. H. Yang** and Y. P. Chen, “Intelligent forecasting system based on grey model and neural network,” in *IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics*, pp. 699-704, 2009.
- [3] **S. H. Yang**, C. Y. Ho and Y. P. Chen, “Neural network based stereo matching algorithm utilizing vertical disparity,” in *Proc. of the 36th Annual Conf. of the IEEE Industrial Electronics Society, IECON'10*, pp.1149-1154, 2010. (0.7 點)

- [4] **S. H. Yang** and Y. P. Chen, “Symbiotic neuron evolution of a neural-network-aided grey model for time series prediction,” in *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*, pp. 195-201, 2011.
- [5] **S. H. Yang**, C. H. Chou, W. P. Pai, T. H. Liu, Y. S. Chang, J. C. Li, H. C. Ting, and Y. P. Chen, “Grey neural network-based forecasting system for vision-guided robot trajectory tracking,” in *Int. Conf. on Control, Automation and Systems*, pp. 1512-1517, 2011.
- [6] **S. H. Yang**, J. C. Li and Y. P. Chen, “Integration of grey model and neural network for robotic application,” in *Proc. of the 37th Annual Conf. of the IEEE Industrial Electronics Society, IECON’11*, pp. 2307-2312, 2011. (0.7 點)

