TRAINING ISSUES AND LEARNING ALGORITHMS FOR FEEDFORWARD AND RECURRENT NEURAL NETWORKS

TEOH EU JIN
B.Eng (Hons., 1st Class), NUS

A THESIS SUBMITTED
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE

May 8, 2009

Abstract

An act of literary communication involves, in essence, an author, a text and a reader, and the process of interpreting that text must take into account all three. What then do we mean in overall terms by 'Training Issues', 'Learning Algorithms' and 'Feedforward and Recurrent Neural Networks?

In this dissertation, 'Training Issues' aim to develop a simple approach of selecting a suitable architectural complexity, through the estimation of an appropriate number of hidden layer neurons. 'Learning algorithms', on the other hand attempts to build on the method used in addressing the former, (1) to arrive at (i) a multi-objective hybrid learning algorithm, and (ii) a layered training algorithm, as well as to (2) examine the potential of linear threshold (LT) neurons in recurrent neural networks. The term 'Neural Networks', in the title of this dissertation is deceptively simple. The three major expressions of which the title is composed, however, are far from straightforward. They beg a number of important questions. First, what do we mean by a neural network? In focusing upon neural networks as a computational tool for learning relationships between seemingly disparate data, what is happening at the underlying levels? Does structure affect learning? Secondly, what structural complexity is appropriate for a given problem? How many hidden layer neurons does a particular problem require, without having to enumerate through all possibilities? Third and lastly, what is the difference between feedforward and recurrent neural networks, and how does neural structure influence the efficacy of the learning algorithm that is applied? When are recurrent architectures preferred over feedforward ones?

My interest in (artificial) neural networks (ANNs) began when in 2003 I embarked on an honor's project, as an undergraduate on the use of recurrent neural networks in combinatorial optimization and neuroscience applications. My fascination with the subject matter of this thesis was piqued during this period of time. Research, and in particularly, the domain of neural networks were a new beast that I slowly came to value and appreciate, then as it was – and now, almost half a decade later. While my research focus evolved during this period of time, the underlying focus has never wavered far from neural networks.

This work is organized into two parts, categorized according to the neural architecture under study: briefly highlighting the contents of this dissertation – the first part, comprising Chapters 2 to 4, covers mostly feedforward type neural networks. Specifically, Chapter 2 will examine the use of the singular value decomposition (SVD) in estimating the number of hidden neurons in a feedforward neural network. Chapter 3 then investigates the possibility of a hybrid population

based approach using an evolutionary algorithm (EA) with local-search abilities in the form of a geometrical measure (also based on the SVD) for simultaneous optimization of network performance and architecture. Subsequently, Chapter 4 is loosely based on the previous chapter – in that a fast learning algorithm based on layered Hessian approximations and the pseudoinverse is developed. The use of the pseudoinverse in this context is related to the idea of the singular value decomposition.

Chapters 5 and 6 on the other hand, focus on fully recurrent networks with linear-threshold (LT) activation functions – these form the crux of the second part of this dissertation. While Chapter 5 examines the dynamics and application of LT neurons in an associative memory scheme based on the Hopfield network, Chapter 6 looks at the possibility of extending the Hopfield network as a combinatorial optimizer in solving the ubiquitous Traveling Salesman Problem (TSP), with modified state update dynamics and the inclusion of linear threshold type neurons. Finally, this dissertation concludes with a summary of works.

Acknowledgements

This dissertation, as I am inclined to believe, is the culmination of a fortunate series of equally fortunate events, many of which I had little hand in shaping.

As with the genius clown who yearns to play Hamlet, so have I in desiring to attempt something similar and as momentous but in a somewhat different flavor - to write a treatise on neural networks. But the rational being in me eventually manifested itself, convincing the other being(s) in me that such an attempt would be one made in futility. Life as a graduate student rises above research, encompassing teaching, self-study and intellectual curiosity. All of which I have had the opportunity of indulging in copious amounts, first-hand. Having said that, I would like to convey my immense gratitude and heartfelt thanks to many individuals, all whom have played a significant role, however small or large a part, however direct or indirect, throughout my candidature.

My thanks, in the first instance therefore, go to my advisors, Assoc. Prof. Tan Kay Chen and Dr. Xiang Cheng for their time and effort in guiding me through my 46-month candidature, as well as for their immense erudition and scholarship – for which I've had the pleasure and respect of knowing and working with, as a senior pursuing my honors thesis during my undergraduate years.

Love to my family - for putting up with my very random eccentricities and occasional idiosyncrasies when at home, from the frequent late-night insomnia to the afternoon narcolepsies that have attached themselves to me. A particular word of thanks should be given to my parents and grandmother, for their (almost) infinite patience. This quality was also exhibited in no small measure by my colleagues, Brian, Chi Keong, Han Yang, Chiam, CY, CH and many others whose enduring forbearance and cheerfulness have been a constant source of strength, for making my working environment a dynamic and vivacious place to be in – and of course, as we would like to think, for the highly intellectual and stimulating discourses that we engaged ourselves in every afternoon. And to my 'real-life' friends, outside the laboratory for the intermittent ramblings, which never failed to inject diversity and variety in my thinking and outlook, and whose diligence and enthusiasm has always made the business of teaching and research such a pleasant and stimulating one for me.

Credit too goes to instant noodles, sliced bread, peanut butter and the occasional cans of tuna, my staple diet through many lunches and dinners. Much of who I am, what I think and how I look at life comes from the interaction I've had with all these individuals, helping me shape not only my thought process, my beliefs and principles but also the manner in which I have come to view and accept life. The sum of me, like this thesis, is (hopefully) greater than that of its individual parts.

Soli del Gloria.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides a broad overview of the field of artificial neural networks, from their classification or taxonomy to functional methodology to practical implementation. Specifically, this chapter aims to discuss neural networks from a few perspectives, particularly with respect to its architecture, weight or parameter optimization via learning algorithms and a few common application areas. This chapter then concludes with a highlight of subsequent chapters that is forms the content of this dissertation.

1.1 Artificial Neural Networks

In general, a biological neural system comprises of a group or groups of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network are almost always extensive. It is believed that the computational power of a biological network arises from its collective nature, where parallel arrangements of neurons are co-activated simultaneously. Connections, called synapses, are usually formed from axons to dendrites, though dendrodendritic microcircuits and other connections are possible. Apart from the electrical signaling, there are other forms of signaling that arise from neurotransmitter diffusion, which have an effect on electrical signaling. As such, biological neural networks are extremely complex. Whilst a detailed description of neural systems is nebulous, progress is being charted towards a better understanding of basic mechanisms.

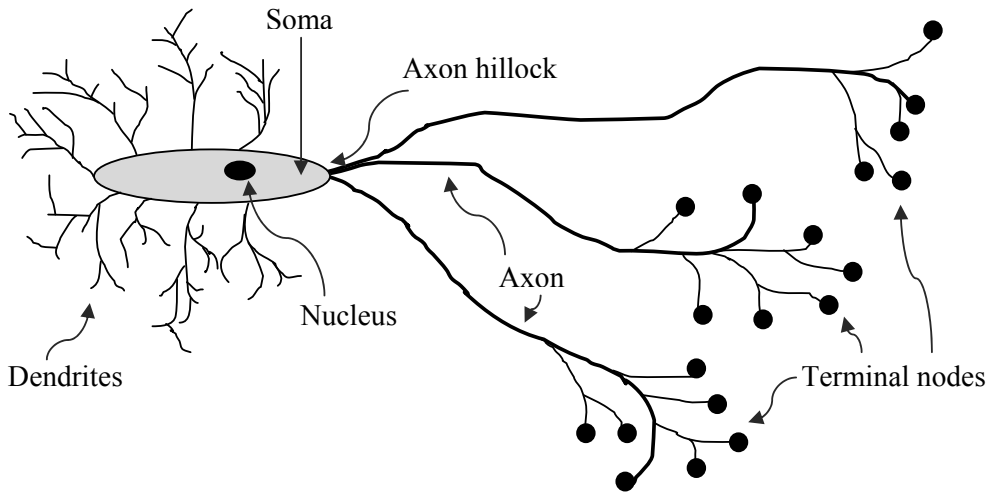

Figure 1.1: Simple biological neural network

On the other hand, an 'artificial' neural network (ANN) draws many parallels from its physiological counterpart, the animal brain. This (simplified) version attempts to mimic and simulate certain properties that are present in its biological equivalents. While largely inspired by the inner workings of the brain, many of the finer details of an artificial neural network, henceforth known simply as a neural network (or NN), arise more out of mathematical and computational convenience than actual biological plausibility, where an interconnected group of artificial neurons is built around a mathematical or computational model for information processing based on a what is known as a *connectionistic* approach to computation. In many cases a neural network is an adaptive system that changes its structure (through its topology and/or synaptic weight parameters) based on external or internal information that 'flows' through the network.

At a fundamental level, a neural network behaves much like a functional mapper between an input and an output space, where the objective of modeling is to 'learn' the relationship between the data presented at the inputs and the signals desired at the outputs. Neural networks are a particularly useful method of non-parametric data modeling because they have the ability to capture and represent complex input-output relationships between a sets of data via a learning algorithm based on an iterative optimization routine[1]. Having said that, from a taxonomical perspective,

[1]This is of course largely based on the assuption that we are dealing with a supervised learning algorithm, where a teaching signal in the form of a set of desired outputs being present at the output during the training phase. on the

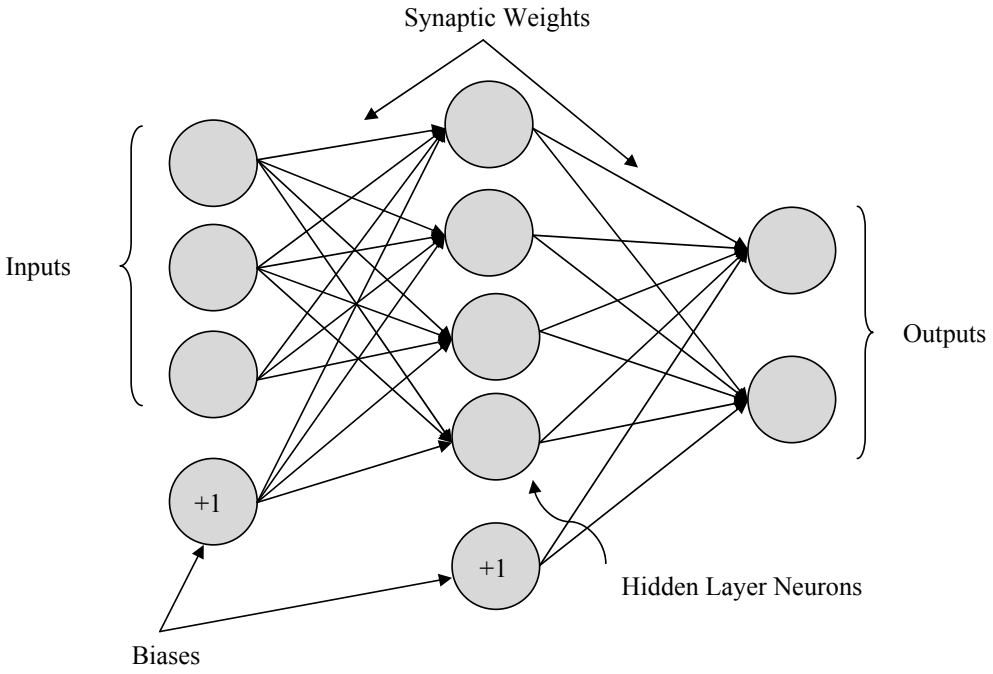

Figure 1.2: Simple feedorward neural network

neural networks are categorized under what is commmonly known as a 'computational intelligence' (CI) framework, which consists mostly of structured algorithmic and mathematical approaches that encompasses aspects of heuristics drawn from their biological analogue. Two other popular computational intelligence methods are *Evolutionary Computation* and *Fuzzy Logic*. Other approaches include, to a less popular extent, Bayesian networks, reinforcement learning (or dynamic programming), wavelets, agent-based modeling as well as other variants and hybrids.

The entire concept underlying a neural network can be deconstructed into two parts first, an architectural or structural model, and secondly, a separate somewhat independent learning mechanism, which is typically the back-propagation with gradient descent method (we term thiss SBP or standard backpropagation is described subsequently in this chapter). Under normal circumstances, the design of a neural network based approach to solving a problem would first require the determination of the architecture of a suitable structural complexity to meet problem-specific requirements. Structural complexity in this sense, can be quantified using a variety of measures but typically, the

other hand, in unsupervised learning, the network learns from similarities in the underlying distribution of data and hence no output data is required to train the network.

simplest method would the enumeration of the number of hidden layer neurons, number of synaptic weights or connections, as well as the degree of multiplicity of these connections to and from a neuron. And as will be highlighted later, the use of an appropriate training routine would be dependent on the type of neural architecture that has been constructed. The conceptually simple, 'black-box' nature of a neural network's learning ability is one of the attractive points of a neural network (as well as of its variants) that lends itself to such applications which require an adaptive, or machine-learning approach.

1.1.1 Learning Algorithms

The learning algorithms used for training a neural network is intimately tied with (i) the network topology or architecture, and (ii) the problem to be solved. The relationship between the leaning algorithm chosen and the network architecture and/or problem at hand is coupled in such a way as to almost make the two inseparable. For example, training a recurrent network is very different from training a feedforward network because of the presence of lateral and feedback connections in recurrent networks that render the backpropagation with gradient descent algorithm less effective than for its feedforward counterpart[2]; similarly, training a network for adaptive control usually requires an online adaptation of the synaptic weights, something which is not necessary when training a network for face or pattern recognition, where an offline batch training approach is acceptable and quite commonplace. Moreover, the application to which the neural network is applied to also affects the type of neural architecture considered – take for example time-series prediction such as power load forecasting, which might favor recurrent structures. A key difficulty thus would be to separate the parameters and functions of a given architecture from that of a learning rule.

The advantage of neural networks lie in their ability to represent both linear and nonlinear relationships and in their ability to learn these relationships directly from the data being presented – the set of input data as well as the set of corresponding (desired) outputs. In a neural network model simple nodes, which can be called variously 'neurons', 'interneurons', 'neurodes', 'processing elements' (PE) or 'units', are connected together to form a network of nodes – hence the term 'neural network'. While a neural network does not have to be adaptive per se, its practical use comes with

[2]This is attributed to the cyclic nature of signal flow, 'diluting' the error deltas during the backpropagation phase.

algorithms designed to alter the strength (weights) of the connections in the network to produce a desired signal flow.

To learn a mapping $\Re^d \rightarrow \Re$ between a set of input-output data, a training set $\mathbf{D}_I = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$ is presented to the network. $x_i \in \Re^d$ is assumed to be drawn from a continuous probability measure with compact support. Learning in this sense, involves the selection of a learning system $\mathbf{L} = \{\mathbf{H}, \mathbf{A}\}$, where the set $\mathbf{H}$ is the learning model and $\mathbf{A}$ is a learning algorithm. From a collection of candidate functions, $\mathbf{H}$ (assumed to be continuous) a hypothesis function $\mathbf{h}$ is chosen by learning algorithm $\mathbf{A}$: $\mathbf{D}_I \rightarrow \mathbf{H}$ on the basis of a performance criterion. This is known as supervised learning. Unsupervised learning, for example Hebbian learning (which is the focus of the second part of this dissertation, on fully recurrent neural networks), do not have a set of 'desired' output or training signals present at the output nodes.

The learning algorithm is a somewhat systematic way of modifying the network parameters (i.e. synaptic weights) in an iterative and automated manner, such that a pre-specified loss or error function is minimized. In most cases, the convention is to use a Sum-of-squared errors ($SSE = \sum_{i=1}^N (\mathbf{d}_i - \mathbf{y}_i)^2$), or a mean-squared-error ($MSE = \frac{1}{N}\sum_{i=1}^N (\mathbf{d}_i - \mathbf{y}_i)^2$ or simply $MSE = \frac{1}{N}SSE$). One of the most common algorithms used in supervised learning is the backpropagation algorithm based on a gradient-descent approach. Being simple and computationally efficient, the iterative gradient search here, has the possibility of local convergence – moreover, this method is also often criticized for being noisy and slow to converge.

That said, the traditional approaches to training neural networks for both feedforward and recurrent types are usually based on simple gradient-based methods. In such approaches, the input data is presented to the neural network and passes through the entire network, from which an output is then obtained this output is then compared with desired output (teaching signal). If they do not match, a corrective signal (which is essentially based on the gradient of this error term) is then passed in a reverse manner into the same network, but from the converse direction, to which corrective modifications to the synaptic weights are then made this is the well-known back-propagation algorithm of error derivatives. The degree of correction would largely depend on the size of deviation between the actual and the desired outputs. This correction can be carried out after every presentation of an input pattern (online, or sequential learning), or made when all the inputs patterns have been presented (batch learning). Such an approach is also known as supervised

learning because there is a set of desired outputs (teaching signal) that corresponds to the set of input patterns. Unsupervised learning on the hand is another class of learning algorithms that attempts to classify or arrange the inputs that are available in the training set of data, purely based on the similarity of features of the input data.

Neural networks have been applied to solve various real-world problems due to its well-documented advantages – adaptability, capability of learning through examples (good for data intensive problems where availability of reliable data is not a problem) and ability to generalize (under appropriate training conditions). To efficiently use the model to various applications, the optimization approaches of ANNs for each specific problem is critical, with numerous search-optimization algorithms for weight and/or architecture optimization, such as evolutionary algorithms (EAs) [45], simulated annealing (SA) [119], tabu search (TS) [61], ant colony optimization (ACO) [41], particle swarm optimization (PSO) [114] and genetic algorithms (GAs) [1, 91] [3]. Among these searching-optimization techniques, some of them have been applied to simultaneous connection weights adjustment and/or architecture optimization of ANNs in a multiobjective scheme [62, 2].

As a case-in-point, a genetic algorithm was hybridized with local search gradient methods for the process of ANN training via weight adjustment of a fixed topology in [8]. Ant colony optimization was used to optimize a fixed topology ANN in [26]. In [175], tabu search was used for training ANNs. Simulated annealing and genetic algorithms were compared for the training of ANNs in [176], where the GA-based method was proven to perform better than simulated annealing. Simulated annealing and the backpropagation variant Rprop [163] were combined for MLP training with weight decay in [201].

The current emphasis is to integrate neural networks within a comprehensive interpretation scheme instead of as a stand-alone application. Neural network studies have evolved from one that was largely theoretical to one that is now predomoninantly application specific through the incorporation of heuristical and *a priori* information, as well as merging the neural network approach with other methods in a hybridized scheme. As domains within science and engineering progresses, neural networks will play an increasingly vital role in helping researchers and practitioners alike in finding relevant information in the vast streams of data under the constraints of lower costs, less time, and fewer people.

[3] All of these are mainly for feedforward neural network architectures

1.1.2 Application Areas

Over the last 2-3 decades, neural networks have found widespread use in myriad applications ranging from pattern classification, recognition and forecasting to modeling various problems in many industries[4] and domains from biology and neuroscience to control systems and finance-economics. The tasks to which artificial neural networks are applied tend to fall within the following broad categories:

1. Function approximation: regression analysis, including time series prediction/forecasting and modeling.

2. Classification: pattern and sequence recognition, novelty detection and sequential decision making.

3. Data processing: filtering, clustering, blind signal separation and compression.

Specific application areas include system identification and control (vehicle control, process control), game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition, etc.), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications, data mining (or knowledge discovery in databases or "KDD"), visualization and e-mail spam filtering.

1.2 Architecture

As mentioned, a taxonomical classification of neural networks can be made on the basis of the direction of signal or informational flow in a neural network where from an architectural perspective, neural networks can be categorized into either feedforward or recurrent networks. As their names suggest, a feedforward network processes information or signal flow strictly in a single direction from input to output; a recurrent network on the other hand, has less restrictive connections signals and information from one neuron to another can be connected between layers of neurons laterally, or even with feedback. The beauty of a recurrent network is only truly understood when dealing with

[4] This includes more exotic domains such as the prediction of food freezing and thawing times [72].

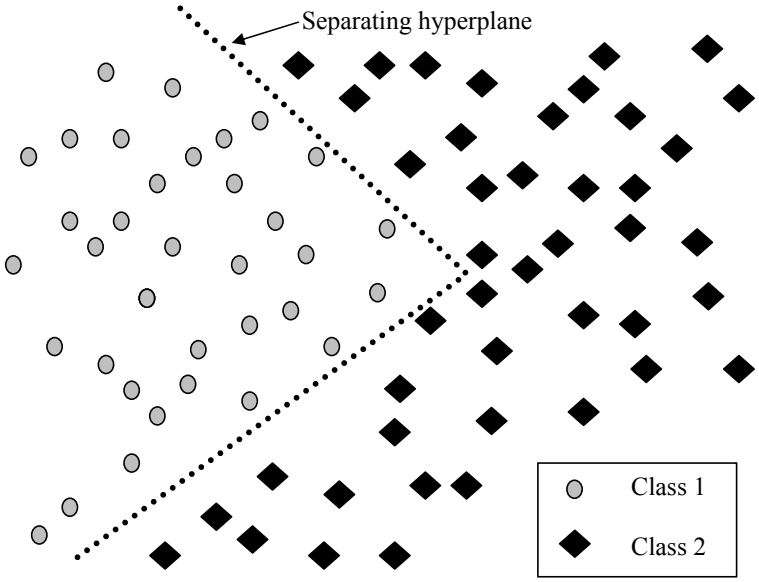

Figure 1.3: A simple, separable, 2-class classification problem.

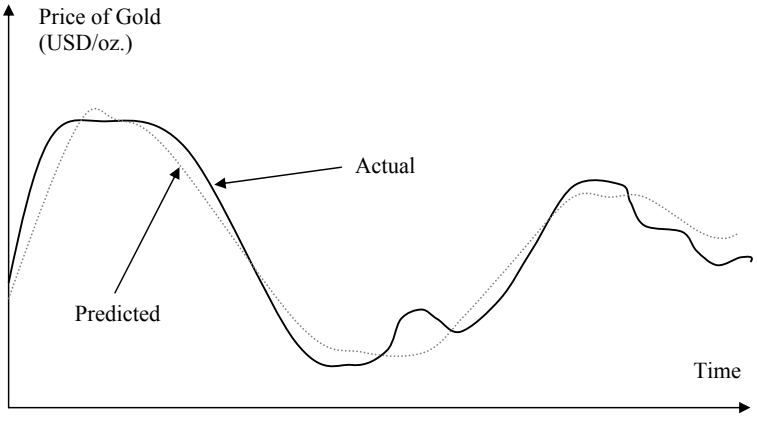

Figure 1.4: A simple one-factor time-series prediction problem.

time-dependent problems as often the difficulty in training a recurrent network using conventional learning algorithms far outweighs its benefits.

A feedforward network, as its name suggests only allows the flow of information in a single or unidirectional manner (in a forward pass, although corrective signals are made in a backpass when using the backpropagation with gradient descent, those signals are error deltas). Recurrent networks on the other hand, allow a very general interpretation of informational flow – there are no limits being placed on the direction of signal flows – structurally, there are feedforward, lateral and feedback connections. Assuming the mathematical nomenclature of a synaptic weight connection to be w_{ij}, where i and j represent the layer index of the network (i.e. the signal flows from layer i to j), the following holds for the different type of synaptic weight connections – feedforward: $i < j$, lateral $i = j$ and feedback $i > j$.

Learning, given a set of input-output examples, is essentially the computation of a mapping from an input to output space, which in turn can be cast as an optimization problem where the minimization of a suitable cost function (such as the ubiquitous sum-of-squared-loss error) is desired. Having said that, it comes as no surprise that the learning strategies for feedforward and recurrent networks are necessarily different.

The selection of an appropriate training algorithm for a neural network, whether recurrent or otherwise, is largely dependent on its overall architecture. Therefore the critical issue is to find, or adapt and evolve a 'sufficient' architecture and correspondingly, the appropriate set of weights to solve a given task – all of which is done in an iterative manner. The weights are thought to be variable parameters that are subjected to adaptation during the learning process. The network is initialized with some random weights and it is run on a set of training examples. Dependent on the response of the network to these training examples, weights are adjusted with respect to some learning rule.

Typically, the more complex the architecture, the likelier that many local minima exists. This is particularly relevant in the dynamics of recurrent neural network. As such, gradient descent based approaches (besides being computationally expensive) usually result in sub-optimal solutions when applied to complex problems being solved by recurrent neural networks. Moreover, the learning time increases substantially when there are time lags between relevant inputs and desired outputs become longer due to the fact that the error decays exponentially as it is propagated through the network.

Long term dependencies are hard to learn using gradient based methods – this is called the vanishing gradient problem. There are a few areas that can be identified to work upon. Two of them are the characterization of the structure of the weight space and the location of the minima of the error.

1.2.1 Feedforward Neural Networks

Multilayer feedforward neural networks (FNN), also equivalently known as multilayer perceptrons (MLP), have a layered structure which processes informational flow in a unidirectional (or feedforward, as its name suggests) manner: an input layer consisting of sensory nodes, one or more hidden layers of computational nodes, and an output layer that calculates the outputs of the network. By virtue of their universal function approximation property, multilayer FNNs play a fundamental role in neural computation, as they have been widely applied in many different areas including pattern recognition, image processing, intelligent control, time series prediction, etc. From the universal approximation theorem, a feedforward network of a single hidden layer is sufficient to compute a uniform approximation for a given training set and its desired outputs, hence this chapter is restricted to discuss the single hidden layer FNN, unless otherwise specified.

Error Back-propagation Algorithm with Gradient Descent

In the standard back-propagation (SBP) algorithm, the learning of a FNN is composed of two passes: in the forward pass, the input signal propagates through the network in a forward direction, on a layer-by-layer basis with the weights fixed; in the backward pass, the error signal is propagated in a backward manner. The weights are adjusted based on an error-correction rule. Although it has been successfully used in many real world applications, SBP suffers from two infamous shortcomings, i.e., slow learning speed and sensitivity to parameters. Many iterations are required to train small networks, even for a simple problem. The sensitivity to learning parameters, initial states and perturbations was analyzed in [220].

Typical learning algorithms for training feedforward neural architectures for a variety of applications such as classification, regression and forecasting utilize well-known optimization techniques.

These numerical optimization methods usually exploit the use of first-order (Jacobian) and second-order (Hessian) methods [5].

The standard back-propagation algorithm for example, utilizes first-order gradient descent based methods to iteratively correct the weights of the network. Learning using second-order information such as those based on based on the Newton-Raphson framework offer faster convergence, but at the cost of increased complexity. Typically, the Jacobian or gradient of the cost function can be computed quite readily and conveniently; however, the same cannot be said of the Hessian, particularly for larger-sized networks as the number of free parameters (synaptic weights) increase. As such, second-order approaches are not popular primarily because of the additional computational complexity that is introduced in calculating the Hessian of the cost function with respect to the weights.

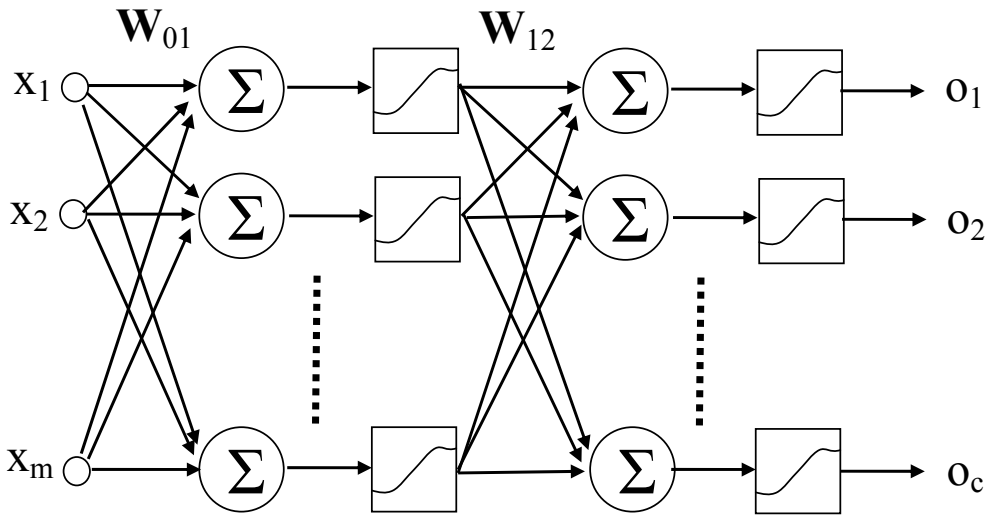

Figure 1.5: Typical FNN architecture

The error signal at the output node $k, k = 1, \cdots, C$, is defined by

$$e_k^p = d_k^p - o_k^p,$$

where the subscript p denotes the given pattern, $p = 1, \cdots, P$. The mean squared error (given the

[5] Respectively, the Jacobian and the Hessian refer to the first and second derivatives of the cost function with respect to the weights.

p-th pattern) is written as

$$E = \frac{1}{C}\sum_{k=1}^{C}\frac{1}{2}(e_k^p)^2 = \frac{1}{C}\sum_{k=1}^{C}\frac{1}{2}(d_k^p - o_k^p)^2. \tag{1.1}$$

In the batch mode training, the weights are updated after all the training patterns are fed into the inputs and thus the cost function becomes:

$$E = \frac{1}{PC}\sum_{p=1}^{P}\sum_{k=1}^{C}\frac{1}{2}(d_k^p - o_k^p)^2. \tag{1.2}$$

For the output layer, compute the derivative of E with respect to (w.r.t.) the weights w_{kj}:

$$\begin{aligned}\frac{\partial E}{\partial w_{kj}} &= \frac{\partial E}{\partial e_k}\cdot\frac{\partial e_k}{\partial o_k}\cdot\frac{\partial o_k}{\partial \alpha_k}\cdot\frac{\partial \alpha_k}{\partial w_{kj}} \\ &= e_k\cdot(-1)\cdot\varphi_k'(\alpha_k)\cdot y_j\end{aligned}$$

Similarly, we compute the partial derivative w.r.t. the hidden layer weights v_{ji}:

$$\begin{aligned}\frac{\partial E}{\partial v_{ji}} &= \sum_k \frac{\partial E}{\partial e_k}\cdot\frac{\partial e_k}{\partial \alpha_k}\cdot\frac{\partial \alpha_k}{\partial y_j}\cdot\frac{\partial y_j}{\partial \alpha_j}\cdot\frac{\partial \alpha_j}{v_{ji}} \\ &= \sum_k e_k\cdot(-1)\cdot\varphi_k'(\alpha_k)w_{kj}\cdot\varphi_j'(\alpha_j)\cdot x_i \\ &= -x_i\cdot\varphi_j'(\alpha_j)\cdot\sum_k e_k\varphi_k'(\alpha_k)w_{kj}\end{aligned}$$

The functions φ_j, φ_k are called *activation functions* which are continuously differentiable. The activation functions commonly used in feed-forward neural networks are described below:

1. *Logistic function.* Its general form is defined by

$$y = \varphi(x) = \frac{1}{1+\exp(-ax)}, \quad a > 0, x \in \mathbf{R}. \tag{1.3}$$

The output value lies in the range $0 \le y \le 1$. Its derivative is computed as

$$\varphi'(x) = ay(1-y). \tag{1.4}$$

2. *Hyperbolic tangent function.* This type of activation functions takes the following form

$$y = \frac{1 - \exp(-2x)}{1 + exp(-2x)}. \tag{1.5}$$

The output lies in the range $-1 \leq y \leq 1$. Its derivative takes the simple form

$$\varphi'(x) = (1 + y)(1 - y). \tag{1.6}$$

3. *Linear function.* It is simply defined as

$$y = \varphi(x) = x. \tag{1.7}$$

The kind of activation functions to be used is dependent on the applications. The former two types of activation functions are called *sigmoidal nonlinearity.* The hidden layer and output layer can take either form 1 or 2. In particular, for function approximation problem, the output layer often employs the linear activation function.

The optimization of the error function over the weights w_{kj} and v_{ji} is typically taking the *steepest descent algorithm*, that is, the successive adjustments applied to the weights are in the direction of steepest descent (a direction opposite to the gradient):

$$\begin{aligned} \Delta w_{kj} &= -\eta \frac{\partial E}{\partial w_{kj}} \\ &= \eta e_k \cdot \varphi_k'(\alpha_k) \cdot y_j \qquad (1.8a) \\ \Delta v_{ji} &= -\eta \frac{\partial E}{\partial v_{ji}} \\ &= \eta x_i \cdot \varphi_j'(\alpha_j) \cdot \sum_k e_k \varphi_k'(\alpha_k) w_{kj} \qquad (1.8b) \end{aligned}$$

where η is a positive constant and called the *learning rate.* The steepest descent method has a zig-zag problem when approaching the minimum, and in order to remedy this drawback, a momentum term

is often added into the above equations:

$$\Delta w_{kj}(t) = -\eta \frac{\partial E(t)}{\partial w_{kj}} + \beta \Delta w_{kj}(t-1) \tag{1.9a}$$

$$\Delta v_{ji}(t) = -\eta \frac{\partial E(t)}{\partial v_{ji}} + \beta \Delta v_{ji}(t-1) \tag{1.9b}$$

where t denotes the iteration number, and β is a positive constant between 0 and 1 called the *momentum constant*.

1.2.2 Recurrent Neural Networks

Recurrent neural networks, through their unconstrained synaptic connectivity and resulting state-dependent nonlinear dynamics, offer a greater level of computational ability when compared with regular feedforward neural network (FFNs) architectures. A necessary consequence of this increased capability is a higher degree of complexity, which in turn leads to gradient-based learning algorithms for RNNs being more likely to be trapped in local optima, thus resulting in sub-optimal solutions.

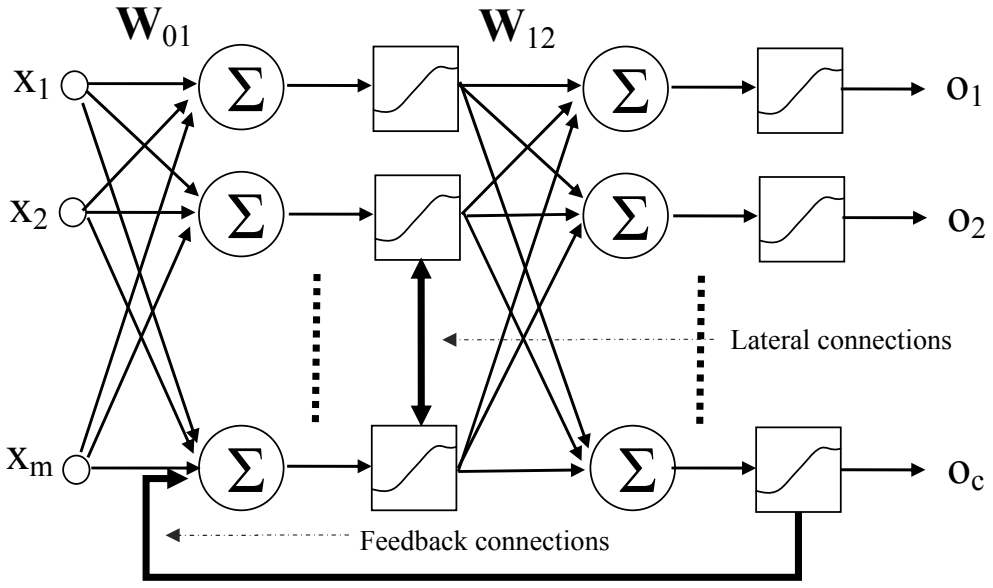

Figure 1.6: Typical RNN architecture: compare with the FNN structure in Fig. 1.5. Note the inclusion of both lateral and feedback connections.

As described in the previous subsection, error backpropagation in feedforward neural network

models is a well-known learning algorithm that has its roots in nonlinear estimation and optimization. It is being used routinely to calculate error gradients in nonlinear systems with hundreds of thousands of parameters. However, the conventional architecture for backpropagation has severe restrictions [155].

Both the BPTT (backpropagation through time) and the batch version of RTRL (real-time recurrent learning) are equivalent (they perform gradient descent on the same cost function). The online version of RTRL, however, introduces some additional complexities, most notable of which is that the RTRL, being an epoch-based algorithm, resets the state of the network to the initial conditions of the trajectory periodically, such that the online version may move very far from the desired trajectory and never return. This is especially true if the network moves into a region where the neurons are saturated, because the gradients go to zero. To alleviate this problem, Williams and Zipser [211] introduced the idea of teacher forcing, where visible units of the network are clamped to the desired trajectory, preventing the actual trajectory from getting too far off course. In this version of the algorithm, we take a single Euler integration step of the original algorithm, including our weight updates, and then enforce the clamps.

Back-propagation through time (BPTT)

As with static back-propagation, fixed-point learning maps a static input with a static output. The difference is that the mapping is not instantaneous. When data is fed to the input of the network, the network cycles the data through the recurrent connections until it reaches a fixed output. Training a network using fixed-point learning can be more difficult than with static back-propagation, but the added power of these networks can result in much smaller and more efficient implementations. In recurrent back-propagation, activations are fed forward until a fixed value is achieved. After this relaxation period, the error is computed and propagated backwards. The error activations must be stable before the weights can be updated, so relaxation of the error is also needed.

Instead of mapping a static input to a static output, BPTT maps a series of inputs to a series of outputs. This provides the ability to solve temporal problems by extracting how data changes over time. Examples of temporal problems are digital signal processing, speech recognition, and time-series prediction. In back-propagation through time, the goal is to compute the gradient over

the trajectory. Since the gradient decomposes over time, this can be achieved by computing the instantaneous gradients and summing the effect over time. During BPTT the activation is sent through the network and each processing element stores its activation locally for the entire length of the trajectory. At each step the network output is also computed and stored. At the end of the trajectory, the errors are generated at the output and a vector of errors is used to update the network weights.

[150] introduced an extension of the back-propagation algorithm for RNNs [6]. The idea is to unfold the recurrent network in time and then to treat it as a feedforward network. [150] extends this analogy to continuous time, and readers are directed to that article for further details.

Real-time recurrent learning (RTRL)

The real-time recurrent learning (RTRL) [211] algorithm is an on-line training algorithm for RNNs. It is a gradient descent based algorithm in which the weights of the network are determined by minimizing the MSE between the desired output and the actual output at the current time step. Given an RNN, the corresponding error gradient at the current time step is calculated, and the weights are changed according to the error gradient to minimize the MSE.

Backpropagating through time requires that the network maintains a trace of its activity for the duration of the trajectory. This becomes very inefficient for long trajectories. Using Real-Time Recurrent Learning, RTRL [211], the temporal credit assignment problem is solved during the forward integration step by keeping a running estimate of the total effect that parameters W_{ij} and b_i have on the activity of neuron x_k. However, the RTRL algorithm is computational intensive because it has a time complexity of for each time step, where is the number of processing nodes. [211] overviews this approach in more detail.

This algorithm is computationally expensive, because we have to store $O(N^3)$ variables in memory and process as many differential equations at every time step. On the other hand, we do not have to keep a trace of the trajectory, so the memory footprint does not depend on the duration of the trajectory. The major advantage, however, is that we do not have to execute the backward dynamics – the temporal credit assignment problem is solved during the forward pass.

[6] See [151] for a survey of techniques that have been used for gradient learning in dynamical RNNs.

1.3 Overview of This Dissertation

Neural networks have thus far demonstrated their effectiveness in both modeling and optimization problems, casting itself as an interesting and particularly worthy subject of study. The primary motivation underlying this work, which will be presented over the following chapters, is to examine two specific areas of personal and professional (in an academic sense) interest: firstly, for feedforward networks, how structural complexity can be studied using a simple geometrical measure and from there, exploited and incorporated in a learning algorithm; and secondly, for recurrent networks, how a particular type of neuron with a linear-threshold type activation function can be utilized and its dynamics analyzed in a Hopfield-type fully recurrent architectures for solving optimization problems.

In Chapter 2, the idea underlying the use of a geometric informational measure to quantify the significance and contribution of separating hyperplanes constructed in the hidden layer space lies, is presented – specifically in attempting to provide a simple yet useful method to describe the contribution of each additional neuron that is added to the hidden layers of a neural network. While the approach that is described here is based on a feedforward architecture, extending this framework to recurrent or more general structure would be similar in thinking. The idea here stems from the understanding of how hidden layer neurons construct hyperplanes in hidden layer space. For a simple two-dimensional classification problem (with two features in input space), each hidden neuron construct a separating hyperplane – how these hyperplanes are arranged and laid out with respect to each other is largely a function of the learning algorithm, which essentially attempts to maximize the linear independency of te final positions of the separating hyperplanes such that they meet, as best as possible, the distribution of the underlying classes of data (by minimizing some performance metric which, commonly for neural networks is the sum, or mean of squared errors).

Building on this idea, Chapter 3 then introduces a geometrical measure based on the SVD operator to estimate the necessary number of neurons to be used in training a single hidden layer feedforward neural network (SLFN). In addition, we develop a new hybrid multi-objective evolutionary approach which includes the features of a variable length representation that allow for easy adaptation of neural networks structures, an architectural recombination procedure based on the geometrical measure that adapts the number of necessary hidden neurons and facilitates the exchange of neuronal information between candidate designs, and a micro-hybrid genetic algorithm

with an adaptive local search intensity scheme for local fine-tuning. In addition, the performances of well-known algorithms as well as the effectiveness and contributions of the proposed approach are analyzed and validated through a variety of dataset types.

Subsequently, Chapter 4 looks at the possibility of decomposing both feedforward and recurrent neural networks into layered stages such that each layer is trained using a different learning algorithm – a layer-by-layer training approach then attempts to build upon the idea of developing a learning paradigm that is able to learn at a fraction of the computational and structural complexity of conventional training algorithms. Specifically, we present two simple, yet effective method to learning for both feedforward and recurrent neural networks based on a 'layered' training mechanism – first, this is done for an MLP that is based on approximating the Hessian using only local information, specifically, the correlations of output activations from previous layers of hidden neurons, and second, for a recurrent MLP structure that is based on a hybrid Evolutionary Strategy (ES) and pseudoinverse approach together with an adaptive linear observer (the pseudoinverse and adaptive linear observer acting as local search operators), as a simple layered learning mechanism for general RNN applications.

In the second part of this dissertation, recurrent architectures are examined – specifically those with a linear-threshold activation function. Unlike feed-forward neural networks, recurrent neural networks (RNN) are described by a system of differential equations that define the exact evolution of the model dynamics as a function of time. The system is characterized by a large number of coupling constants represented by the strengths of individual junctions, and it is believed that the computational power is the result of the collective dynamics of the system. Two prominent computation models with saturating transfer functions, the Hopfield network and cellular neural network, have stimulated a great deal of research efforts over the past two decades because of their great potential of applications in associative memory, optimization and intelligent computation [94, 95, 194, 27, 134, 224, 195, 223].

As a nonlinear dynamical system, intrinsically, the stability is of primary interest in the analysis and applications of recurrent networks, where the Lyapunov stability theory is a fundamental tool and widely used for analyzing nonlinear systems [74, 203, 221, 160]. Based on the Lyapunov method, the conditions of global exponential stability of a continuous-time RNN were established and applied to bound-constrained nonlinear differentiable optimization problems [129]. A discrete-time recurrent

network solving strictly convex quadratic optimization problems with bound constraints was analyzed and stability conditions were presented [153]. Compared with its continuous-time counterpart, the discrete-time model has its advantages in digital implementation. However, there is lack of more general stability conditions for the discrete-time network in the previous work [153], which deserves further investigation.

Solving NP-hard optimization problems, especially the traveling salesman problem (TSP) using recurrent networks has become an active topic since the seminal work of [95] showed that the Hopfield network could give near optimal solutions for the TSP. In the Hopfield network, the combinatorial optimization problem is converted into a continuous optimization problem that minimizes an energy function calculated by a weighted sum of constraints and an objective function. The method, nevertheless, faces a number of disadvantages. Firstly, the nature of the energy function causes infeasible solutions to occur most of the time. Secondly, several penalty parameters need to be fixed before running the network, while it is nontrivial to optimally set these parameters. Besides, low computational efficiency, especially for large scale problems, is also a restriction.

It has been a continuing research effort to improve the performance of the Hopfield network [7, 3, 152, 148, 185]. The authors in [7] analyzed the dynamic behavior of a Hopfield network based on the eigenvalues of connection matrix and discussed the parameter settings for TSP. By assuming a piecewise linear activation function and by virtue of studying the energy of the vertex at a unit hypercube, a set of convergence and suppression conditions were obtained [3]. A local minima escape (LME) algorithm was presented to improve the local minima by combining the network disturbing technique with the Hopfield network's local minima searching property [152]. Most recently, a parameter setting rule was presented by analyzing the dynamical stability conditions of the energy function [185], which shows promising results compared with previous work, though much effort has to be paid to suppress the invalid solutions and increase convergence speed.

In recent years, the linear threshold (LT) network which underlies the behavior of visual cortical neurons has attracted extensive interests of scientists as the growing literature illustrates [83, 42, 21, 167, 76, 77, 209, 222]. Differing from the archetypical neurons used in Hopfield-type networks, the LT network possesses nonsaturating transfer functions of neurons, which is believed to be more biologically plausible and has more profound implications in the neurodynamics. For example, the

network may exhibit multistability and chaotic phenomena, which might give provide insights into the underlying processes of associative memory and sensory information processing [214].

The LT network has been observed to exhibit one important property, which is multistability; this feature allows the network to possess multiple steady-states, or equilibrium points coexisting under certain synaptic weights and external inputs. This then allows LT networks to exhibit characteristics suitable for decision-making, digital selection and analogue amplification [77]. It was proven that local inhibition is sufficient to achieve nondivergence of LT networks [210]. Most recently, several aspects of LT dynamics were studied and the conditions were established for boundedness, global attractivity and complete convergence [222]. Nearly all the previous research efforts were devoted to stability analysis, thus the cyclic dynamics has yet been elucidated in a systematic manner. In the work of [76], periodic oscillations were observed in a multistable WTA (Winner-Take-All) network when slowing down the global inhibition. He reported that the epileptic network switches endlessly between stable and unstable partitions and eventually the state trajectory approaches a limit cycle (periodic oscillation) which was shown by computer simulations. It was suggested that the appearance of periodic orbits in linear threshold networks was related to the existence of complex conjugate eigenvalues with positive real parts. However, there was lack of theoretical proof about the existence of limit cycles. It also remains unclear what factors will affect the amplitude of the oscillations. Studying recurrent dynamics is also of crucial concern in the realm of modeling the visual cortex, since recurrent neural dynamics is a basic computational substrate for cortical processing. Physiological and psychophysical data suggest that the visual cortex implements preattentive computations such as contour enchancement, texture segmentation and figure-ground segregation.

This recurrent architecture of LT neurons is further investigated in Chapters 5 and 6, specifically for associative memories and combinatorial optimization (Traveling Salesman Problem) respectively.

Chapter 2

Estimating the Number of Hidden Neurons Using the SVD

In this chapter, I attempt to quantify the significance of increasing the number of neurons in the hidden layer of a feedforward neural network architecture using the singular value decomposition (SVD). Although the SVD has long been utilized as a method of off-line or non-real-time computation, parallel computing architectures for its implementation in near real time have begun to emerge. Through this, I extend some well-known properties of the SVD in evaluating the generalizability of single hidden layer feedforward networks (SLFNs) with respect to the number of hidden layer neurons. The generalization capability of the SLFN is measured by the degree of linear independency of the patterns in hidden layer space, which can be indirectly quantified from the singular values obtained from the SVD, in a post-learning step. A pruning/growing technique based on these singular values is then used to estimate the necessary number of neurons in the hidden layer. More importantly, this chapter describes in detail properties of the SVD in determining the structure of a neural network particularly with respect to the robustness of the selected model.

2.1 Introduction

The ability of a neural network to generalize well on unseen data depends on a variety of factors, most important of which is the matching of the network complexity with the degree of freedom or information that is inherent in the training data. A network that is too small is unable to learn the inherent or salient characteristics of the data ('underfitting'), while a network that is too large captures an excessive amount of information, most of which is redundant, such as noise and properties that are only limited to the training data ('overfitting'). Matching of this information with complexity is critical for networks that are able to generalize well. With this in mind, growing and pruning methods have been the focus of numerous approaches in the neural network literature [126, 207, 84]. A measure of the complexity of a structure is its number of free, or adjustable parameters, which for a feed-forward neural network is the number of synaptic weights. Clearly, the capacity of a feed-forward neural network in learning the samples of the training set is proportional to its complexity [101, 103]. For a simple SLFN architecture, the number of weights in the structure is a function of the number of hidden layer neurons; each hidden neuron added increases the number of connections by a factor of $(M+1)+C$ where M and C are the number of input and output nodes respectively (the additional dimension accounts for the bias for the hidden layer neurons). See [13] for an overview of the complexity of learning in neural networks. An SLFN with a large number of adjustable parameters, corresponding to a high model capacity and complexity tends to over-fit the training data and thus poor generalization on the testing set. Conversely, a classifier with insufficient capacity will provide a poor learning performance.

The objective here is to determine a parsimonious model that provides a reliable performance for the given (training) data, with the smallest structural complexity – as the model complexity increases, with all other factors (such as the training data) held constant, the performance of this network improves up to a certain limit in the complexity, after which the model perofrmance on the unseen testing data then deteriorates. Clearly, there is a trade-off in the approximation obtained in the training set and the generalization of the unseen testing data. The aim of structural selection is to find this compromise value. If the model complexity is below it, underfitting occurs; conversely overfitting occurs when the model complexity is above this compromise value. Determining this value is difficult, depending not only on our definition of an 'optimal' value but also on the amount

and quality of data. A computationally simple approach would thus be best, as will be put forward in this chapter.

The problem of estimating the number of neurons in the hidden layers of SLFNs (also known as multi-layer perceptrons, MLPs) is difficult; conventional techniques are often based on a trial-and-error approach or some heuristics; a typical method involves partitioning the available data into three sets – training, validation and testing sets. Usually, this approach involves training the MLP while increasing number of neurons in the hidden layer [1] and subsequently comparing the performance (accuracy) of the resulting SLFN with respect to both the training and validation set, up till a point where the SLFN's performance peaks on the validation set – after which the introduction of additional hidden neurons will result in an increase in the training accuracy at the expense of generalization (there is a corresponding decrease in the SLFN's accuracy on the validation set). This is similar in principle to the *early stopping method of training* [87]; this technique however, emphasizes a statistical approach, and does not provide much insight to the geometry of the problem at hand.

In this chapter, I examine an approach of estimating the necessary number of hidden layer neurons using the Singular Value Decomposition (SVD). The SVD essentially factorizes a matrix of any dimension into a product of 3 constituent matrices – 2 square, orthonormal bases and a diagonal, rectangular matrix consisting of singular values. Its properties and corresponding application in the context of estimating the number of hidden layer neurons will be highlighted in further detail subsequently. I discuss more in detail, the significance of these singular values in the construction of hyperplanes in hidden layer space, as well as suggest some heuristical thresholds for pruning and growing the network model complexity based on these singular values.

The outline of this chapter is as follows: preliminaries are given in Section 2, describing some previous efforts in estimating bounds on the number of neurons in the hidden layer, as well as establishing notations that will be used throughout this chapter. Section 3 then introduces the SVD while Section 4 describes the construction of hyperplanes in hidden layer space by introducing hidden neurons, after which I explore the concept of actual (effective) and numerical rank of matrices. A pruning/growing technique using the singular values obtained from the SVD is then used to estimate the necessary number of hidden neurons – this is addressed in Section 5. Simulations and illustrative

[1] This approach gradually increases the network size, and is in this sense, a 'growing' algorithm.

examples together with a discussion on some of the ideas and concepts that are proposed will be presented in Section 6. A summary highlighting some possible areas for future work then concludes this chapter.

2.2 Preliminaries

2.2.1 Related work

Previous work on estimating the number of hidden layer neurons have often focused on the learning capabilities of the SLFN on a training set without accounting for the possibility of the network being over-trained [100, 169, 186, 188, 101, 103]. Projection onto increasingly higher-dimensional (hidden layer) space is achieved by implementing increasingly larger number of neurons in the hidden layer. Cover's seminal work using concepts from combinatorial geometry rigorously showed that patterns projected onto higher dimensions are likelier to be linearly separable [34]. This work was followed by Huang [100] and Sartori and Antsaklis [169], demonstrating that a simple SLFN can achieve perfect accuracy on an N-size problem (there are N samples in the training set) when N hidden neurons are used – this creates N hyperplanes that partition the N samples into N distinct regions. This approach however, was achieved using a simple matrix inverse without any use of any iterative training algorithms. When $N-1$ hidden neurons are used – this creates $N-1$ hyperplanes that partition the N samples into N distinct regions. This approach however, was achieved using a simple matrix inverse without any use of any iterative training algorithms. Moreover, perfect reconstruction of the training set is usually not desirable as over-fitting usually occurs, particularly in many practical examples where noise is typically present in the training set – perfect reconstruction would mean that the mapping (look-up table approach) learnt by the SLFN fits the noise rather than the actual features of the training samples. The generalization ability of such large capacity SLFNs are hence very poor.

The use of the SVD in neural networks have been briefly highlighted by [85, 187, 159]. [206] used the SVD to demonstrate that the effective ranks of the hidden unit activations and the hidden layer weights are equal to each other as the solution converges thus indicating that the learning algorithm (the backpropagation using gradient descent was used) can be stopped. On the other

hand, geometrical methods such as in [213] have been used to great effect in providing greater insight into the nature of the problem, however such methods are only applicable to problems in which the dimensionality of the problem is in either 2- or 3-dimension for the problem is thus visualizable.

Traditionally, the simplest possible approach in network pruning would be to first train the network and to subsequently remove weights that are of small magnitudes – clearly this method, though simple, has obvious flaws. Other more sophisticated pruning techniques such as *Optimal Brain Damage* (OBD) [126] and *Optimal Brain Surgeon* (OBS) [84] make use of the Taylor approximation of the error functional, focusing on the second-order Hessian matrix in determining the sensitivities of the weights of the trained network to remove redundant weights. While [126] uses a diagonal approximation of the Hessian to determine the saliency (importance) of the removal of weights, [84] explores the off-diagonal entries of the Hessian to obtain improved results on generalization, at the cost of increase complexity as a near-exact computation of the Hessian is now being used. The Hessian, however, is computationally intensive to obtain and despite advances that have been made in computing the Hessian [25, 28], a simpler approach is always desired. While the OBD and OBS focuses on the pruning of weights, the approach that is proposed here is focused on the estimation of the appropriate number of neurons to be used in the hidden layer – in this sense, full connectivity between the layers of the feedforward network is assumed.

The SVD computes directly on a data matrix, rather than on the corresponding estimated correlation or covariance matrix. This might lead to one questioning the similarity between the use of SVD and another equivalent approach, namely *principal components analysis* (PCA). PCA, while similar in approach to the SVD, can be viewed as the application of the SVD operator on a set of mean-centered data. This is the key difference between SVD and PCA – the centering of the data, which in turn ensures that the origin is in the middle of the data set. SVD will result in vectors that pass through the origin. In many problems this is desired. On the other hand, centering destroys sparseness, and in some problems it might affect other structure in the data as well. In addition, centering is sensitive to outliers. Alternatively, PCA is seen to work on the covariance matrix of the data while the SVD works on the original matrix. Both SVD and PCA are however, similar in the sense that they are related to standard eigenvalue-eigenvector problems, as well as aim to remove noise or correlation and obtain the most significant information, or factors, from the data. Moreover, singular values can be obtained more efficiently with a greater degree of numerical and

computational stability as compared to through the computation of the eigenvalues [120] [2].

2.2.2 Notations

For the purpose of clarity and uniformity, I define some symbols and abbreviations that will be used in this chapter. Consider a single hidden layer feedforward network (SLFN) with n hidden neurons trained on a set of N training patterns, X of dimension $M+1$ (an added dimensionality is included to account for the bias input). The nonlinearity of the hidden layer activation functions is $h(\cdot)$. The output activations of these hidden neurons are represented by the matrix H, where $H = h(XW)$ with $X \in \mathbb{R}^{\mathbb{N}\times(\mathbb{M}+\not{\mu})}$, $W \in \mathbb{R}^{(\mathbb{M}+\not{\mu})\times\ltimes}$ and $H \in \mathbb{R}^{\mathbb{N}\times\ltimes}$. Typically, we have an over-determined system where $N \geq M$ (H is *tall and thin*). The discussion that follows throughout this chapter will center about the singular value decomposition (SVD) of the hidden layer activation matrix H.

2.3 The Singular Value Decomposition (SVD)

The ordinary Singular Value Decomposition (SVD) is widely used in numerous statistical and signal processing computation applications, both for the insight it provides of the structure of a linear operator, and as a technique for reducing the computational word length required for least-squares solutions and certain Hermitian eigensystem decompositions, by roughly a factor of two. The SVD is one of many widely known matrix factorizations, but its primary function is as a *rank-revealing decomposition*, providing an insight of the *actual rank* (also known as *effective rank*) of the matrix that is being analyzed. Alternatively, the SVD can also be seen as a robust method of determining the null space of a particular matrix. The interested reader is directed to [69, 120, 182] (and references therin) for a more thorough treatment on the theory and concepts underlying the SVD. In this section, I motivate the use of the SVD as a tool to estimate the necessary number of neurons to be used in training an SLFN.

[2] As Klema and Laub ([120]) asserts, it is generally not advisable to compute the singular values from the square-root of the eigenvalues of auto-correlation of a real, rectangular matrix H, that is H^TH – [120] provides an illustrative example, which is reproduced here for completeness. Given a real number μ ($|\mu| < \sqrt{\epsilon}$), such that $fl(1+\mu^2)$ (fl denoting a floating-point computation) and $H = [1\ 1;\ \mu\ 0;\ 0\ \mu]$, then $fl(H^TH) = [1\ 1;\ 1\ 1]$. Hence, because of the lack of computational precision, $\hat{\sigma}_1 = \sqrt{2},\ \hat{\sigma}_2 = 0$ leading to the incorrect conclusion that the rank of H is 1. This would not occur is we have infinite computational precision, for then $H^TH = [1+\mu^2\ 1;\ 1\ 1+\mu^2]$ with $\sigma_1 = \sqrt{2+\mu}$ and $\sigma_2 = |\mu|$ leading to the correct conclusion that the rank$(H) = 2$.

If this 'contamination' were absent, the matrix H would clearly be rank deficient, but due to the noise and other observational oddities, H will have cluster of small nonzero singular values [82]. Typically the individual elements of E is unknown (but its general properties may be known – this will be explained subsequently), and applying a rank-revealing decomposition such as the SVD to the 'contaminated' matrix $\tilde{X}$ would provide a quantification of the 'robustness' of a number k such that $k \leq n$. In most cases, $\tilde{X}$ will be full-ranked and hence the decomposition would not reveal the actual rank of $\tilde{X}$ through the structure of its zero elements [182]; however it does provide detail into its actual rank by allowing us knowledge of its the 'small' elements – what constitutes a 'small' number will be examined subsequently. The SVD is now described. Given a real matrix $H \in \mathbb{R}^{\mathbb{N}\times\aleph}$, applying the SVD results in the orthogonal transformation.$U \in \mathbb{R}^{\mathbb{N}\times\mathbb{N}}$ is known as the left singular vectors of H and $V \in \mathbb{R}^{\aleph\times\aleph}$ is known as the right singular vectors of H. Both U and V are orthonormal. $\Sigma \in \mathbb{R}^{\mathbb{N}\times\aleph} = \text{diag}(\sigma_{\not\aleph}, \ldots, \sigma_{\aleph})$ where $(\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0 = \sigma_{n+1} = \ldots = \sigma_N)$, is a diagonal matrix with unique, nonnegative entries ordered in decreasing magnitude. This decoupling technique of the SVD allows the expression of the original matrix as a sum of the first n columns of u and v^T, weighted by the singular values, i.e. $H = \sum_{i=1}^{n} \sigma_i u_i v_i^T$. The matrix $M_i = u_i v_i^T$ is known as the ith mode of H, where $H = \sigma_1 M_1 + \sigma_2 M_2 + \ldots + \sigma_n M_n$. The rank of H is determined by observing that the n largest singular values are non-zero, whereas the $N - n$ smallest singular values are zero. The SVD also factorizes the original matrix H into two orthonormal bases for range and null spaces associated with H [182]. Note that $UU^T = I_N$ and $VV^T = I_n$. By partitioning U and V as

$$U = (U_1, U_2) \text{ and } U = (V_1, V_2) \tag{2.1}$$

we have:

i. The columns of U_1 form an orthonormal basis for the column space of H.

ii. The columns of U_2 form an orthonormal basis for the null space of H^T.

iii. The columns of V_1 form an orthonormal basis for the column space of H^T.

iv. The columns of V_2 form an orthonormal basis for the null space of H.

Supposing $\tilde{H} = H + E$, let the singular values of H and $\tilde{H}$ be $(\sigma_1, \ldots, \sigma_k)$ and $(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_k)$ respectively. Likewise, let U_1, U_2, V_1, V_2 and $\tilde{U}_1, \tilde{U}_2, \tilde{V}_1, \tilde{V}_2$ be the orthonormal bases of H and $\tilde{H}$ respectively. From Schmidt's *Subspace Theorem* [182], we have (with $\|\cdot\|_F$ denoting the Frobenius

norm),

$$\tilde{\sigma}_{k+1}^2 + \ldots + \tilde{\sigma}_n^2 \leq \|E\|_F^2 \tag{2.2}$$

With this, the SVD of $\tilde{H}$ reveals the rank, in a manner that its $n-k$ smallest singular values are bounded by the Frobenius norm of E [182]. While each of the $n-k$ smallest singular values are bounded by the spectral norm of E, this result is less useful than the above bound. This also allows us to conclude that the spaces spanned by $\tilde{U}_1, \tilde{U}_2, \tilde{V}_1, \tilde{V}_2$ are approximations to the original space spanned by U_1, U_2, V_1, V_2 to about $\sigma_k^{-1}\|E\|$. Hence, if σ_k is reasonably large as compared to E (the problem has a favorable Signal-to-Noise Ratio, SNR), the SVD provides a good approximation to the desired subspaces [182]. The use of the SVD, specifically its spectral properties through its singular values (the diagonals of Σ), provides us with an indication of the degree of linear independency of the matrix H. While each of the $n-k$ smallest singular values are bounded by the spectral norm of E, this result is less useful than the above bound.

Algorithms for computing the SVD can be found in [70], where its derivations, concepts and applications are discussed further in depth. The most widely used algorithm for computing the SVD is by Golub and Reinsch ([68]) which is both stable and efficient. This chapter will focus on the use of the SVD as a robust tool in estimating the number of hidden layer neurons without focusing on the algorithmic details of the SVD, which can be found in the literature highlighted above.

2.4 Estimating the number of hidden layer neurons

2.4.1 The construction of hyperplanes in hidden layer space

Every neuron in the hidden layer constructs a hyperplane in the input feature space [100]. The additional separating capability of the system in view of this additional hyperplane created by an additional hidden neuron depends on its 'uniqueness', an indication of which is given by the rank of the matrix H. Quite clearly, and in a geometrical sense, the rank of H denotes the space in which the columns of H occupy, representing the number of (unique) separating hyperplanes of the system. In the case where $n = N-1$ hidden neurons are used (with a suitable nonlinear activation function) such that H is full-ranked, this satisfies the rank requirement of [100, 169, 101], giving rise to $N-1$ separating hyperplanes for the N training samples. Using the simple matrix inverse [100, 169, 101],

we are guaranteed perfect reconstruction for the training set [3]. Intuitively, if H is of higher rank, H better fits the training data, at the expense of generalization when $\lim_{n \to N} \text{rank}(H)$. This full-rank condition also ensures that the patterns projected onto the hidden layer space are linearly independent; accordingly this is also known as *ϕ-general position* as described in Cover ([34]). h is equivalent to ϕ in the context of this chapter. [4] It can be surmised that hidden neurons of the first layer form the (separating) hyperplanes, those of the second layer form the regions, and finally the neurons of the output layer form the classes [199].

2.4.2 Actual rank (k) versus numerical rank (n): H_k vs. H_n

The actual rank (say k) of H may be different from its numerical rank (say n), where $k \leq n$. Such a situation usually arises when the original matrix X is 'contaminated' by E, resulting in a matrix $\tilde{H}$, such that $\tilde{H} = X + E$ [182], with $\text{rank}(H) = k$ and $\text{rank}(\tilde{H}) = n$. These 'contaminations', which are commonly referred to as *noise*, obstructs the characterization of the true properties of the system or problem given the observed training samples. As will be highlighted subsequently, this 'noise' (when taken in our context), corresponds to the marginal role that is played by additional hidden layer neurons that tends to fit the features of the training samples which are not truly representative of the intrinsic underlying distribution of observations. Aside from noise, these 'contamination' can also consist of environment and measurement errors or features that is only present in the training but not the testing set.

Knowledge of the rank of H is often the initial stage in determining further information that can be derived from the column and null spaces of H. The use of the SVD, as a rank-revealing decomposition, allows us to obtain approximations to the subspaces spanned by H_n and H_k. Thus, even if the transformed patterns produced in the hidden layer space are in general position, some of these patterns may be degenerate, in the sense that certain projected patterns can 'almost' be represented as a linear combination of other projected patterns. But is this additional hidden neuron contributing to the actual separability of the samples, or it it merely compensating for the noise that

[3] A reviewer suggested that the validity of [100] and [169] is based on the condition that the same value may be chosen for all the hidden biases – however, in many practical applications, the hidden biases may be adjusted with different values. In fact, as pointed out by [101], N hidden neurons are actually used in [100] and [169].

[4] Too much nonlinearity will produce many non-unique outputs while too little nonlinearity may produce activations which are integral multiples of one another – both will give rise to a matrix H which is low-ranked.

is present in the observations? Clearly, there is a limit, for which introducing additional hidden neurons will tend to over-fit the training data. The *numerical rank* of H is defined as the number of linearly independent columns of H. Equivalently, the rank of a matrix H is also the minimum n which satisfies $\sigma_n > \sigma_{n+1} = 0$ with σ_i $(i = \{1, 2, \ldots, N\})$ being the singular values of H. However, a value k which describes the *actual rank* of the matrix H is more useful for us in estimating the appropriate number of hidden neurons to be used in the SLFN for a given problem.

Often, the presence of degeneracy in H is subtle and not readily apparent – this is to say that even if H is full-ranked, it may be numerically rank-deficient or unstable, in that it has nonzero singular values that are close to zero. Under many practical circumstances, the presence of noise and error will give rise to a matrix $\tilde{H}$ that, from a mathematical perspective is full-ranked and linearly independent, yet is *almost* linearly dependent. The SVD allows us to estimate the number of columns of H (corresponding to the number of hidden layer neurons that should be used for the problem) that are most linearly independent. The closeness of H to a matrix that is of defective rank will cause it to behave erratically when subjected to many statistical and numerical algorithms [69].

This suggests that the column space of H_k spans a region that is almost similar to that spanned by the column space of H_n. Conceptually, we may arrive at the conclusion that H_n is the result of a perturbation of a rank-k matrix H_k. Thus a more general concept of the rank of a matrix is used, by specifying a tolerance below which, it is of full rank. In the context of representing the input patterns in hidden layer space, we can think of additional hidden layer neurons as causing degeneracy in this hidden layer space, for increasing the number of hidden layer neurons is similar to introducing noise into the system – thus perturbing the hidden layer space (which could have been reasonably represented using k hidden neurons instead) such that the hidden layer space is now being represented by n hidden neurons which are of marginal benefit.

Theorem 1. *Define the* numerical ϵ-rank k_ϵ *of the matrix* H *with respect to some tolerance* ϵ *[82] by:*

$$k_\epsilon = k_\epsilon(H, \epsilon) \equiv \min_{\|E\|_2 \leq \epsilon} \{rank\ (H + E)\} \tag{2.3}$$

Theorem 1 states that if there is a 'gap' between the k_ϵ-th and the $k_{\epsilon+1}$-th singular values of size ϵ (i.e., $\sigma_{k_\epsilon} > \epsilon > \sigma_{k_\epsilon+1}$), then H has actual rank (ϵ-rank) k_ϵ. The larger this gap ϵ is, the more 'robust' the matrix H' is to perturbation (the more 'confident' we are that the matrix H can be truncated

at k) – Hansen ([82]) further elucidates that the ϵ-rank of a matrix H is equal to the number of columns of H that are guaranteed to be linearly independent for any perturbation of H with norm less than or equal to the tolerance ϵ, i.e. (robust to perturbations of E such that $\|E\|_2 \leq \epsilon$). The sensitivity to the errors E is now influenced by the ratio of σ_1/σ_k instead of the usual conditional number of H, cond(H)= σ_1/σ_n. More specifically, the numerical rank (δ, ϵ, r) of a matrix if r if and only if, $\sigma_r \geq \delta > \epsilon \geq \sigma_{r+1}$ with δ being defined as, $\epsilon < \delta \leq \sup\{\eta : \|H - H'\| \leq \eta \Rightarrow \text{rank}(H') \geq r\}$. Golub et. al. [69], from which the following definition originates, provides an excellent treatment of the concept of numerical rank. To avoid possible problems when ϵ is itself perturbed, the definition of actual rank is refined by introducing δ as an upper bound for ϵ for which the numerical rank remains at least equal to k.

Theorem 2. *The matrix H has a numerical rank of (δ, ϵ, r) with respect to the norm $\|\cdot\|$ if δ, ϵ and r satisfies the following:*
i) $k = \inf\{rank\ (\tilde{H}) : \|H - \tilde{H}\| \leq \epsilon\}$
ii) $\epsilon < \delta \leq \sup\{\eta : \|H - \tilde{H}\| \leq \eta \Rightarrow rank\ (\tilde{H}) \geq k\}$
σ_k provides an upper bound for δ while σ_{k+1} provides a lower bound for ϵ, i.e. $\delta \leq \sigma_r$ and $\epsilon \geq \sigma_{k+1}$. This is the best ratio that can be obtained for ϵ/γ is σ_{k+1}/σ_k. Note that the norms used above are either the L_2 or Frobenius norms.

The above definitions are equivalent to saying that the matrix H is linearly-independent when perturbed by E up to a threshold determined by $\|E\|_2 \leq \epsilon$. This result also means that the singular values of H satisfies $\sigma_{k_\epsilon} > \epsilon > \sigma_{k_{\epsilon+1}}$. This concept will be used in the next section as a robustness measure of the estimate that is made of the number of hidden layer neurons. As described in [82] and originally in [69], a *well-determined gap* between the singular values of σ_{k_ϵ} and $\sigma_{k_\epsilon+1}$, represented by ϵ should exist in order for the above definition to make much sense; k_ϵ should be, in other words, well-defined for small perturbations of the threshold ϵ and the singular values. Alternatively, the numerical ϵ-rank is the smallest integer k for which $\sum_{j=k+1}^{n} \sigma_j^2 \leq \epsilon^2$ (again, Schmidt's *Subspace Theorem*).

This result suggests that as more neurons are added to the hidden layer, the contributory effect of each additional hidden neuron decreases after a certain threshold. In economic theory, this could be seen as *diminishing marginal returns*. From a geometric point of view, additional hyperplanes

constructed by these newly introduce hidden neurons are not *that* unique, or different as compared to existing hyperplanes (these new hyperplanes may be almost parallel to existing ones). The 'significance' of these new hyperplanes can be quantified by examining the singular values of the matrix H, as more hidden layer neurons are added.

These singular values indicate the degree of mutual correlation between features in the hidden layer space with column degeneracy resulting when these hidden space features are highly correlated, which in turn leads to the conclusion that these additional neurons are redundant. While the singular values do not provide information on which of these features are correlated, the presence of small singular values would indicate that these additional hidden neurons can be removed without affecting the performance of the SLFN significantly. However, it is the relative and not the absolute magnitude of the singular values that interests us – this is described in further detail in the next section.

2.5 A Pruning/Growing Technique based on the SVD

In the previous sections, our notion of 'small' singular values was purely arbitrary. In this section, I define the 'small' singular values purely in a relative sense. Here I describe some possible methods in selecting the threshold γ. These methods are largely heuristical in nature, as is common in the determination of model order in system identification problems [131].

2.5.1 Determining the threshold

A long-standing problem in the use of the SVD as a tool in determining the actual, or effective rank of a (perturbed) matrix is in the distinguishing of (significantly) small and (insignificantly) large singular values [123]. Supposing $H_n \in \mathbb{R}^{\mathbb{N}\times\aleph}$ represents the hidden layer output activation matrix of a SLFN with n neurons in the hidden layer. As n increases, the input-output space mapping that is discovered by the MLP better approximates the training data. Increasing n can be seen as increasing the complexity (and hence capacity) of the network, but all problems would have an 'inherent' degree of complexity, which is essentially unknown – estimating this complexity, characterized by k that is in some sense close to the inherent complexity of the problem, is our objective. However, as n is increased, the better fitting of the training data by the MLP gives rise to a lesser ability to generalize

on unseen examples (i.e. the training set). Let $\sigma_i(H_n)$ denote the ith singular value of the SLFN with n hidden neurons. A way to measure the contribution of the i singular value to say, the separability of classes, is to relate its value to other singular values in Σ.

Using the singular values obtained from the SVD, the *condition number* of any matrix H by definition, is given by $\kappa(H_n) = \sigma_1(H_n)/\sigma_n(H_n) = \|H_n\| \cdot \|H_n^\dagger\|$ where σ_i is the ith singular value of Σ. The condition number is a measure of how far the matrix H_n is from a matrix H_k of lesser rank ($k < n$) [120]. Adopting a similar approach, we can limit the number of hidden neurons to k, where we attempt to find a value of k which satisfies the following: $\min_k\{\sigma_{k+1}(H_n)/\sigma_1(H_n)\} \leq \gamma$, with γ as some threshold.

Alternatively, we could also take into account the sensitivity of the singular values ($\epsilon = \sigma_k - \sigma_{k+1}$), as an indication of the 'robustness' in the selection of a number k ($< n$). This sensitivity value is also used to determine the numerical ϵ-rank r_ϵ [82] of the matrix H (this was explained in the previous section),

$$r_\epsilon = r_\epsilon(H, \epsilon) \equiv \min_{\|E\|_2 \leq \epsilon} \{\text{rank}(H + E)\} \tag{2.4}$$

Since the L_2-norm of H is equivalent to the sum-of-squares of the singular values of H (U and V are orthogonal and their norms are unity), another approach would be to find k such that selection of the k largest singular values retain the 'spectral energy' of H, i.e. $\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2 \geq \gamma$ with γ being some pre-specified threshold.

If ϵ is large, it can be assumed that the matrix H is relatively robust to perturbations (such as rounding, measurement or observation errors); conversely, if ϵ is selected to be small (but not too small such that the numerical ϵ-rank k_ϵ does not make sense [82]), external noise that is introduced to the system may cause the matrix H to be rank-degenerate. Sometimes, there is no clear value for k where $\sigma_k - \sigma_{k+1}$ is obvious. If the SLFN has been well-trained, and has converged (there is little change in its weight values), in some problems, the decay of the singular values is gradual and not very distinct, and hence cannot conclude confidently that the numerical rank of matrix H is less than its actual rank.

In what follows, both σ_i and ϵ_i ($i = \{1, \ldots, n\}$) are used to estimate the necessary number (k) of hidden layer neurons to implement. Recall that $\epsilon_k = \sigma_k - \sigma_{k+1}$. We associate the ϵ_i with σ_i. Ideally, we want to preserve all $i = \{1, 2, \ldots, k\}$ such that σ_i is large and ϵ_k is large. Put simply,

we want to retain singular values that contribute to the overall spectral energy (this corresponds to large $\sigma_i,\ i = \{1, 2, \ldots, k\}$) and yet remain sufficiently robust (this corresponds to a large ϵ_k).

Let $\hat{\sigma}_i$ denote the normalized, or scaled singular value, i.e. $\hat{\sigma}_i = \sigma_i / \sum_{j=1}^{n} \sigma_j$. The simple measure ($\rho_i$) that is used is a λ-weighted (or regularized) linear combination of σ_i and ϵ_i, i.e. $\rho_i = \lambda\sigma_i + (1-\lambda)\epsilon_i$. Since $\epsilon_i = \sigma_i - \sigma_{i+1}$, $\rho_i = \lambda\sigma_i + (1-\lambda)(\sigma_i - \sigma_{i+1}) = \sigma_i + (\lambda - 1)\sigma_{i+1}$, where $\lambda \in [0, 1]$.

To summarize the possible measures or criteria that can be used in determining the appropriate, or necessary value for $k = R(H, \gamma)$:

(1) $\min_k\{\big(\sigma_{k+1}/\sigma_1\big) \leq \gamma\}$ or $\min_k\{\big(\sigma_k/\sigma_1\big) < \gamma\}$

(2) $\min_k\{\epsilon_k \geq \gamma\}$

(3) $\min_k\{\big(\sum_{i=1}^{k} \sigma_i / \sum_{i=1}^{n} \sigma_i\big) \geq \gamma\}$ or $\min_k\{\big(\sum_{i=k+1}^{n} \sigma_i / \sum_{i=1}^{n} \sigma_i\big) < \gamma\}$

(4) $\min_k\{\big(\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2\big) \geq \gamma\}$ or $\min_k\{\big(\sum_{i=k+1}^{n} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2\big) < \gamma\}$

(5) $\min_k\{\left[\sum_{i=1}^{k} \sigma_i^2 / \sum_{i=1}^{n} \sigma_i^2\right]^{\frac{1}{2}} \geq \gamma\}$

(6) $\min_k\{\big(\lambda\sigma_k + (1-\lambda)\epsilon_k\big) \leq \gamma\}$

(7) $\min_k\{\big(\lambda\hat{\sigma}_k + (1-\lambda)\hat{\epsilon}_k\big) \leq \gamma\}$

Criterion (1-5) have been previously considered in [123, 70]. Here, in criteria (6) and (7), I propose using both the singular values σ and its associated sensitivity ϵ in determining the actual rank k of the matrix H_n. Presently, the threshold value γ is chosen heuristically and is largely problem-dependent – there is a lack of available theoretical or analytical expressions – however [123] provides some statistical techniques in constructing suitable threshold values for γ, particularly for rank determination in signal processing problems. By varying the threshold γ I am able to 'control' the confidence that I have in the selected model complexity of the network – a higher γ would reflect a greater level of confidence that $k = n$. Further work, using more advanced measures of structural complexity such as the Akaike or Bayesian information criterion is possible.

Typically, the feedforward network is trained on the training set using a tuning-based learning algorithm such as the well-known backpropagation with steepest-gradient descent (scaled-conjugate variant) algorithm, using a sufficiently large number of hidden layer neurons (in our simulations, 20 hidden layer neurons were often used as a starting value). A sufficiently large number of hidden neurons is required because too few hidden neurons often give a large first singular value, which,

together with a high threshold, provide an incorrect indication of the rank of the output activations of the hidden layer neurons (H). The network is then trained to convergence, until a minimum in the training error is obtained where there is little or no change in the error (approximately 5000-10000 epochs).

Pruning, as well as growing of the network is applied *after* the learning phase, after all observations (training samples) have been presented. The SVD is then applied on the output activations of the hidden layer neurons (H), and the network subsequently pruned after an appropriate threshold (γ) and selection criteria (see above) is chosen, thus obtaining an estimated rank of H as k. After the rank k is obtained, the network is retrained using the same learning algorithm, but with the number of hidden layer neurons in the network set to k. Our results demonstrate improved generalization performance on the testing set with insignificant change in the training error.

Conversely, in developing a growing method using the SVD, additional neurons are incorporated into the hidden layer only *if the decay of singular values of H does not show a distinct gap.* In this case, geometrically, the construction of the hyperplanes are more or less distinct from one another, and perhaps the solution to the problem would benefit from the incorporation of additional neurons into the hidden layer. The notion of a 'distinct' gap can be approached from the same perspective as that of the pruning method used above, where $\epsilon = \sigma_k - \sigma_{k+1}$ is the measure of robustness used. Unlike OBD and OBS, this approach is significantly faster – as this criterion is based on the geometric nature of the hyperplanes in hidden layer space, and hence do not make use of the error function in determining the number of redundant neurons.

2.6 Simulation results and Discussion

In this section, the performance of the pruning/growing method using the singular values and the associated sensitivity values from performing the SVD on H (in a post-learning step) is investigated on two types of classification datasets: (i) 2 toy datasets, and (ii) well-known datasets. The use of the toy datasets are studied as these are of low-dimension (2-D) and are illustrative examples that would thus offer better visual insight of the problem and its proposed solution (particularly in the construction of hyperplanes in hidden layer space), while the real-life datasets provide some

justification and indications for use in practical circumstances [5]. The SLFN for each dataset was trained for 20 runs for 5000 epochs (to ensure that it is sufficiently well-trained). The average training and testing accuracies were then obtained. The back-propagation algorithm was implemented using a batch scaled-conjugate gradient (SCG) variant [142]. The algorithm is based upon a class of optimization techniques well known in numerical analysis as the Conjugate Gradient Methods. SCG uses second order information from the neural network but requires only $O(N)$ memory usage, where N is the number of weights in the network. Batch learning in this case is preferred over stochastic learning as the conditions of convergence are well understood, in addition to the availability of many acceleration techniques such conjugate gradient that only operate in a batch learning mode. Also, theoretical analysis of the weight dynamics and convergence rates are simpler [127].

Prior to training the SLFN, some preprocessing is carried out on the dataset samples. All input features are scaled such that the resulting input features have a mean of 0 and a variance of 1. Convergence is usually faster if the average of each input variable over the training set is close to zero [127]. For the outputs, since classification problems are considered, binary target values are used, with a 1-out-of-C encoding – where for a C-class problem, the largest output i is assigned to class i, with $i = \{1, 2, \ldots, C\}$. For the jth training sample, the desired class output c is $d_i(j) = 1 \in \{0, 1\}$ for $i = c$ and 0 otherwise. Hidden layer neurons use a hyperbolic tangent nonlinearity, while the output nodes use a linear output activation function. For the real-life classification datasets, the proposed criteria (6) and (7) are used from Section 2.5. However, for the toy datasets, criteria (4) with a threshold of $\gamma = 0.95$ is used to demonstrate our ideas. Note that the classification accuracies are measured using a normalized scale, $\in [0, 1]$.

2.6.1 Toy datasets

Banana dataset: See Figures 2.2, 2.3 and 2.4. This is an artificially created dataset (*banana* dataset) consisting of two classes and 200 samples of which 140 samples were used as the training set and the remaining for the testing set. Geometrically, the use of 2-3 hyperplanes constructed from 2-3 corresponding hidden neurons would be sufficient (see Figure 2.2). Expectedly, the first three singular values are dominant based on criteria (4), as shown in Figure 2.4. Setting the threshold

[5] These datasets were obtained from the UCI Machine Learning database at *http://www.ics.uci.edu/~mlearn/MLRepository.html*

$\gamma = 0.95$, $k = 3$.

Lithuanian dataset: See Figures 2.5, 2.7 and 2.8. This is an artificially created dataset (*lithuanian* dataset) consisting of two classes and 200 samples of which 140 samples were used as the training set and the remaining for the testing set. Geometrically, the use of 3-4 hyperplanes constructed from 3-4 corresponding hidden neurons would be sufficient (see Figure 2.5). Expectedly, the first four singular values are dominant based on criteria (4), as shown in Figure 2.8. Setting the threshold $\gamma = 0.95$, $k = 4$.

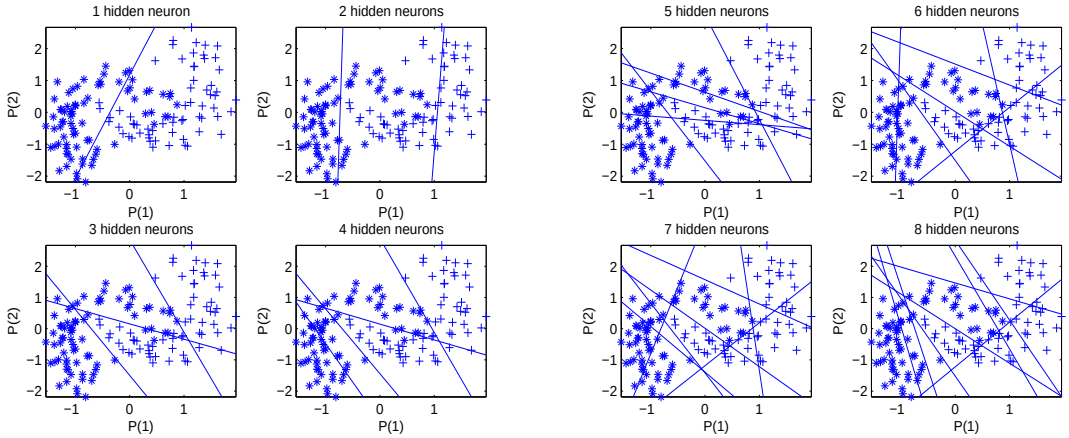

Figure 2.1: *Banana* dataset: 1-8 hidden neurons

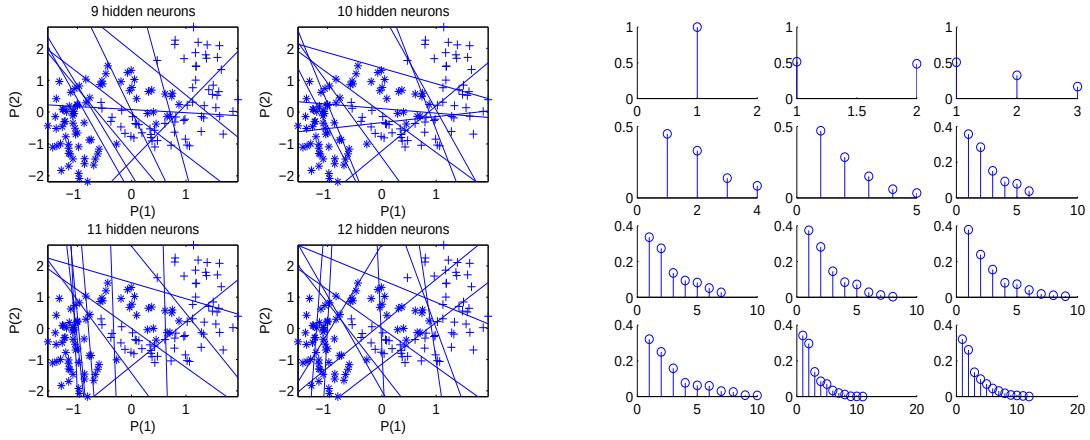

Figure 2.2: *Banana* dataset: 9-12 hidden neurons and corresponding decay of singular values

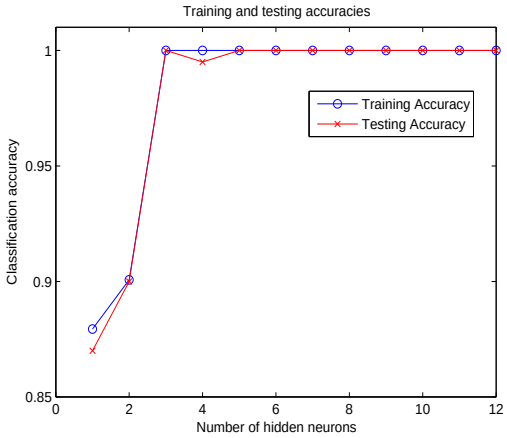

Figure 2.3: *Banana*: Train/Test accuracies

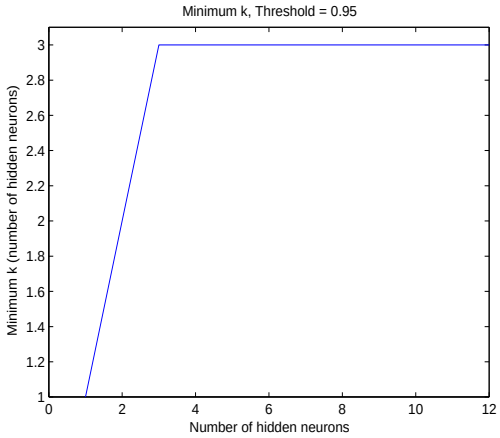

Figure 2.4: *Banana*: Criteria (4)

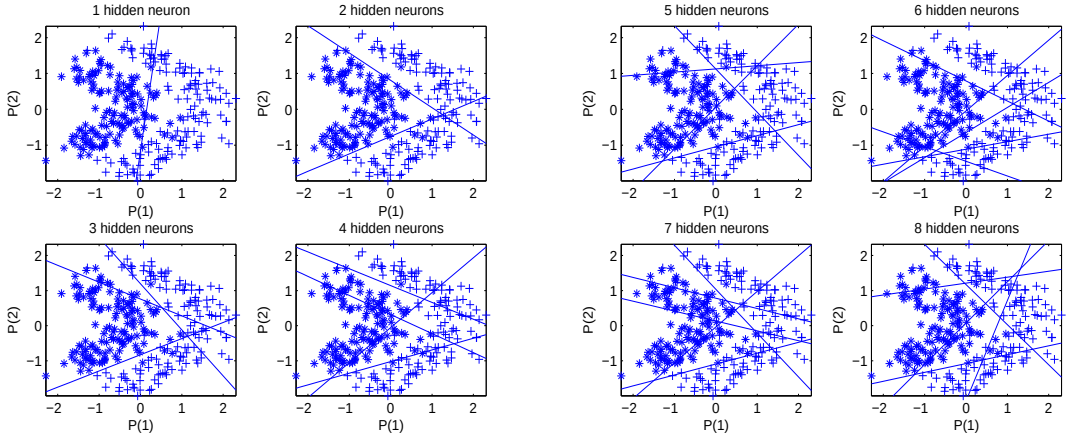

Figure 2.5: *Lithuanian* dataset: 1-8 hidden neurons

2.6.2 Real-life classification datasets

Figures 2.13 and 2.16 show the mean training and testing accuracies, and the suggested number of hidden neurons using criteria (7) of Section 2.5 on two datasets – the Iris and Heart datases from the UCI machine learning repository.

2.6.3 Discussion

From the simulation results obtained, increasing the number of hidden layer neurons after a certain threshold of neurons has been reached has a marginal effect on the resulting performance of the

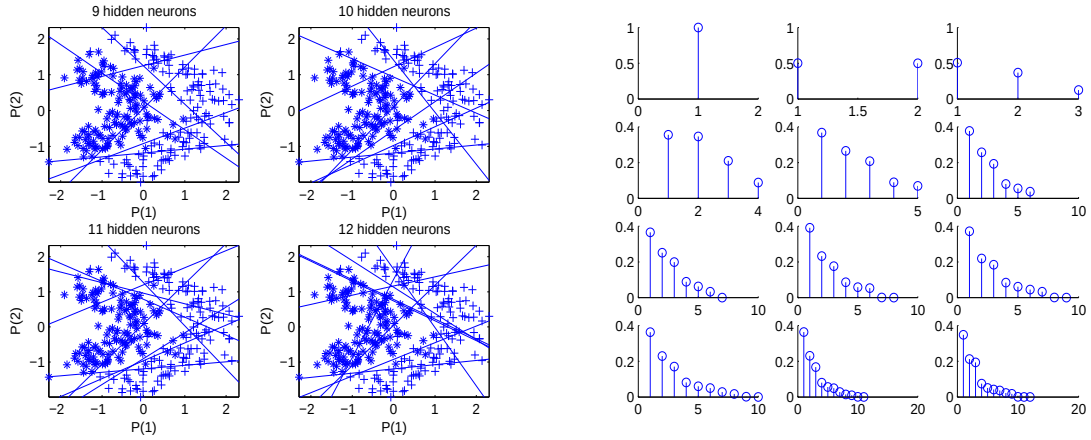

Figure 2.6: *Lithuanian* dataset: 9-12 hidden neurons and corresponding decay of singular values

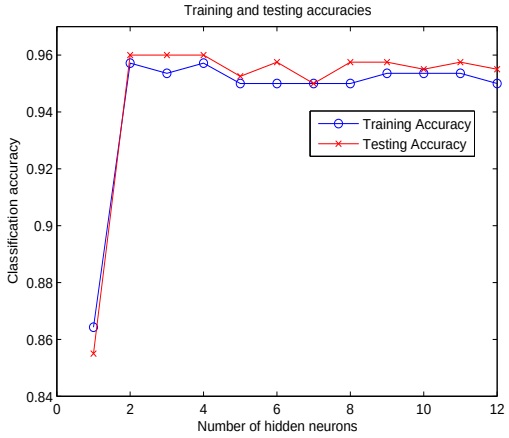

Figure 2.7: *Lithuanian*: Train/Test accuracies

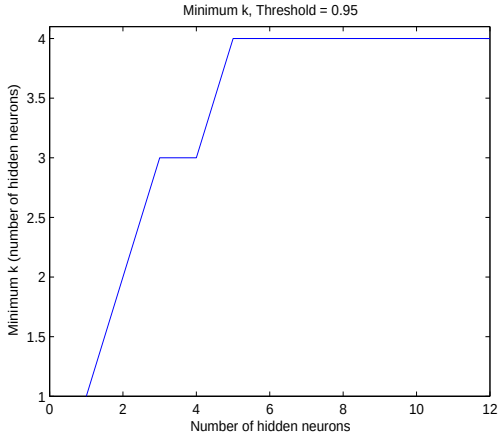

Figure 2.8: *Lithuanian*: Criteria (4)

classifier, mostly on the training set. Generally, using more hidden neurons than that which is necessary has a detrimental effect on the classifier's performance on the testing set, as the capacity of this more complex SLFN will exploit the additional free parameters of the network in over-fitting the samples in the training set. This can be observed from the increased classification accuracy on the training set at the expense of classification accuracy on the testing set.

For the problems described in the previous section, instead of using the singular values or the sensitivity of the singular values in isolation, a λ-weighted (or regularized) linear combination of σ_i and ϵ_i is used. This is criteria (6) and (7) of Section 2.5 is used in displaying the decay of singular values shown in the right columns of the figures in Section 2.6. Crues (σ_i) and sensitivities (ϵ_i). For

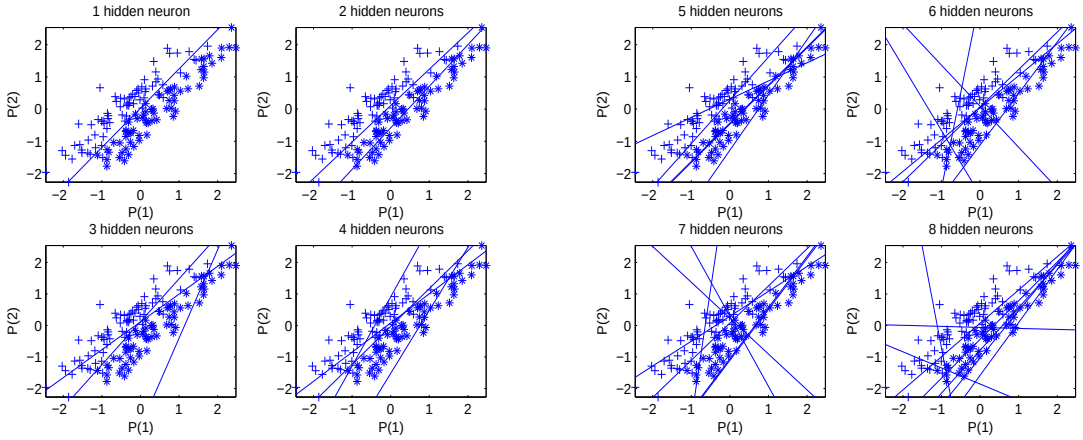

Figure 2.9: *Difficult* dataset: 1-8 hidden neurons

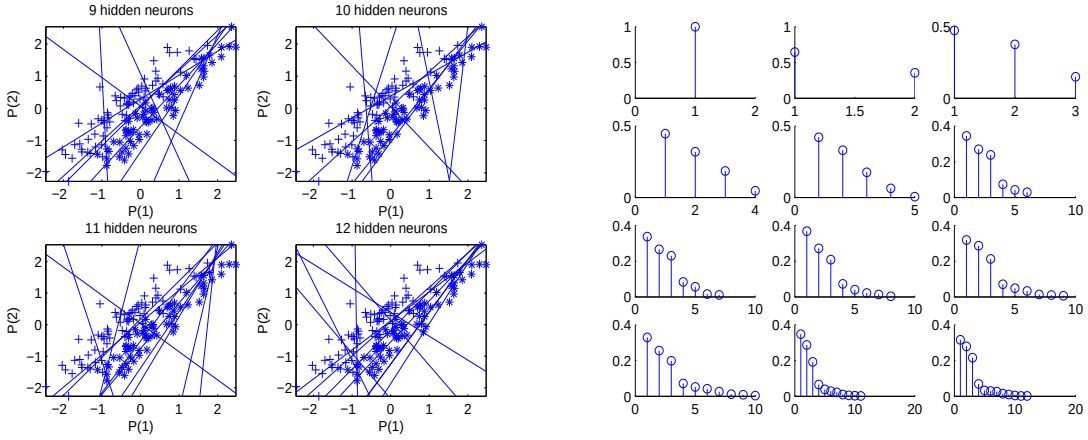

Figure 2.10: *Difficult* dataset: 9-12 hidden neurons and corresponding decay of singular values

our simulations, $\lambda = 0.5$. A possible approach is to manually determine the actual rank k from a visual observation of the proposed criteria, which is graphically similar to that obtained from these right columns of the aforementioned figures.

It has to be noted that there is no ideal threshold for every problem, and unless prior information is available of the problem, the fine-tuning of the threshold involves a rather qualitative than analytical approach, and is to be expected from system identification problems [131]. Visual inspection of the singular values and their associated sensitivities provide an interesting (though manual) approach in determining the appropriate number of hidden neurons to be used for the given problem. This decay shows a range of hidden neurons that should be used in solving a given problem, such

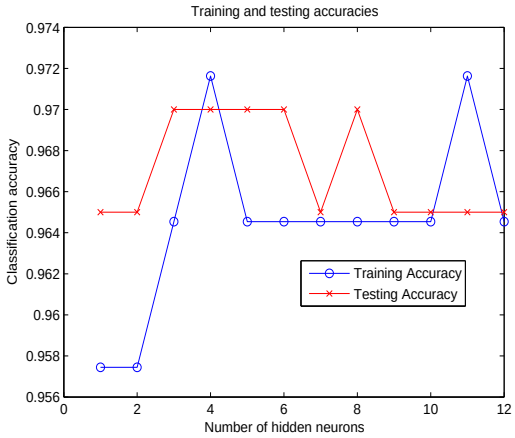

Figure 2.11: *Lithuanian*: Train/Test accuracies

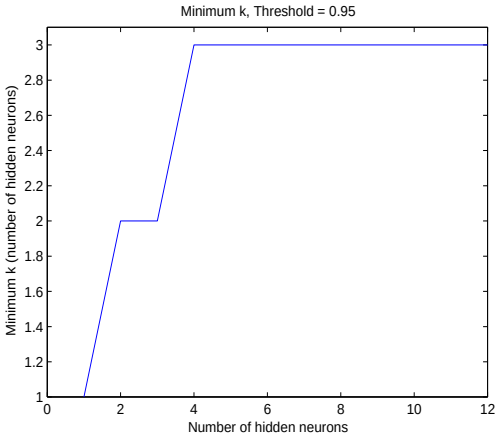

Figure 2.12: *Lithuanian*: Criteria (4)

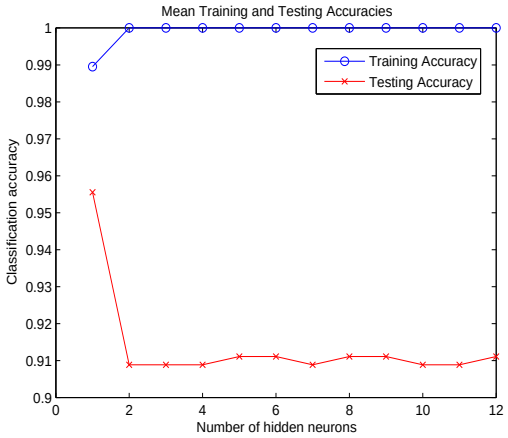

Figure 2.13: Iris: Classification accuracies (2 neurons, criteria (7))

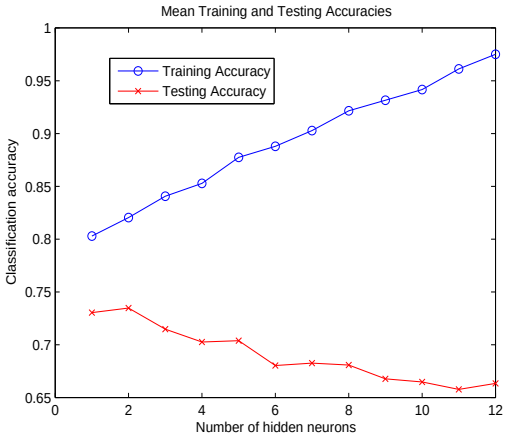

Figure 2.14: Diabetes: Classification accuracies (3 neuron, criteria (7))

that the hyperplanes constructed from these neurons are as 'unique' as possible – redundant *number* of neurons are identified (though not exact knowledge of the identities of the redundant neurons). Although *subset selection* [70] is a possible technique where the k most linearly independent columns of $H \in \Re^{N \times n}$ are picked, such an approach is quite complex, requiring the construction of a permutation matrix, and is hence much slower than retraining the network using the reduced number k of hidden neurons.

We thus argue that exact identification of redundant neurons is not feasible nor useful because

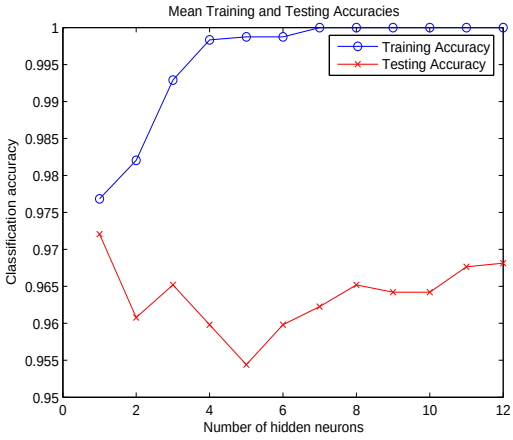

Figure 2.15: Breast cancer: Classification accuracies (2 neurons, criteria (7))

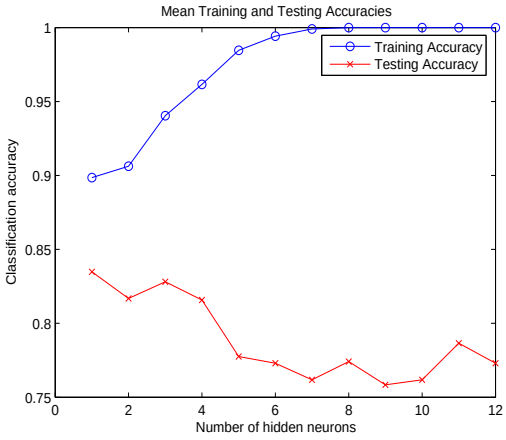

Figure 2.16: Heart: Classification accuracies (3 neurons, criteria (7))

the training algorithm constructs the separating hyperplanes (mapping features from input space to hidden layer space) based on the number of free weights available, fitting the distribution of training samples. The training algorithm would, to the best of its ability, construct these hyperplanes such that they are as linearly independent as possible, as a function of the number of hidden neurons. Note that the addition of every new hyperplane alters the positioning of previous hyperplanes as the training algorithm attempts to place this new set of hyperplanes to maximize the linear independency of the set of hyperplanes while improving the class separation. Retraining the network using the same training algorithm would result in better accuracy and is more efficient from a computational perspective as the training algorithm attempts to maximize the linear independency of the reduced set (k instead of n) of hidden layer neurons. This happens until a certain number of hidden layer neurons is used, after which additional hidden neurons introduced cannot be constructed such that they are linearly independent (for an illustrative example, refer to the hyperplanes constructed in the toy datasets – when increasingly larger number of hidden neurons are used, the hyperplanes tend to lie closely parallel to each other), and this manifests itself as the decay in the singular values of the matrix H.

With this said, a technique that can be used is to first train the SLFN on the given set of training data using a reasonably large number of hidden layer neurons. The SVD is then applied on the matrix of hidden layer activations (H), in a post-learning step. Upon visualization of the decay

of singular values using the various criterion specified in Section 2.5, we estimate the appropriate number of hidden layer neurons to be used for that particular problem; subsequently the training algorithm is reapplied to the same set of training data using the reduced number (k instead of n) of hidden layer neurons.

Although most systems use nodes based on dot products and sigmoids, many other types of units or layers can be used. A common alternative is the radial basis function (RBF) network. In such networks, the SVD, when applied to the hidden layer output activation matrix (H) of a well-trained RBF network would, provide an estimate of the number of basis functions to be implemented in the hidden layer. This is conceptually similar to the SLFN case for the MLP as discussed in this chapter. In a similar vein, this approach can also be used in the context of estimating the number of hidden neurons (interneurons) in a recurrent neural network setting.

2.7 Chapter Summary

This chapter has presented an empirical approach in estimating the necessary number of hidden layer neurons for a single hidden layer feedforward network (SLFN) given a set of training data, through the use of the singular values via the singular value decomposition (SVD) in a post-learning step. These singular values provide a quantification of the degree of degeneracy, or equivalently, the level of linear independency of the training patterns in hidden layer space. The weights learnt during the training stage of the SLFN projects the features from input space onto a hidden layer space. However, while increasing the number of hidden layer neurons increases the capacity and thus the complexity of the system on the *training set*, the generalizability of the resulting SLFN is compromised as over-fitting occurs. We believe the use of the SVD can be extended to the estimation of hidden neurons in feedforward neural networks with multiple hidden layers, as well as in recurrent neural networks. Another possible direction for future work could be a study into the use of the proposed approach on networks that have been trained using constructive learning algorithms that are non-tuning based, such as the ELM paradigm [104].

The main contribution of this chapter is to present a simple framework for the use of the singular values as well as their corresponding sensitivities of the hidden layer output activation matrix H, in providing an indication of the necessary or appropriate number of hidden layer neurons (as well

as a robustness measure of this estimate) that should be used in a SLFN for a given training set. While the current approach involves a rather qualitative and manual approach in determining the thresholds and number of hidden layer neurons, the use of the rank-revealing SVD allows us to gain a better practical and empirical insight into the geometry of hidden layer space, which would otherwise be difficult, if not outright impossible given that many of today's practical problems involve features that are of high-dimension and models that are of high-complexity.

Chapter 3

Hybrid Multi-objective Evolutionary Neural Networks

Evolutionary algorithms are a class of stochastic search methods that attempts to emulate the biological process of evolution, incorporating the 'borrowed' concepts of selection, reproduction and mutation. In recent years, there has been an increase in the use of evolutionary approaches in the training of artificial neural networks (ANNs). While evolutionary techniques for neural networks have shown to provide superior performance over conventional training approaches, the simultaneous optimization of network performance and architecture will almost always result in a slow training process due to the added algorithmic complexity. In this chapter, I present a geometrical measure based on the singular value decomposition (SVD) operator to estimate the necessary number of neurons to be used in training a single hidden layer feedforward neural network (SLFN). In addition, a new hybrid multi-objective evolutionary approach is developed, which includes the features of a variable length representation that allow for easy adaptation of neural networks structures, an architectural recombination procedure based on the geometrical measure that adapts the number of necessary hidden neurons and facilitates the exchange of neuronal information between candidate designs, and a micro-hybrid genetic algorithm with an adaptive local search intensity scheme for local fine-tuning. In addition, the performances of well-known algorithms as well as the effectiveness

and contributions of the proposed approach are analyzed and validated through a variety of dataset types.

3.1 Evolutionary Artificial Neural Networks

In the context of artificial neural network (ANN) design, evolutionary optimization have led to the development of evolutionary artificial neural networks (EANNs) in which adaptation is performed primarily by means of evolution. EANNs have been shown to possess several advantages over conventional methods of training [51, 130, 217]. More significantly, it provides a platform for the simultaneous optimization of synaptic weights and network architecture, and eliminates the daunting task faced by researchers in the selection of appropriate network architecture. According to Stanley and Miikkulainen [181], it is necessary to evolve network architecture and weights simultaneously to achieve a desirable performance. Given that the intrinsic relationship between the architecture and the associated synaptic weights can be quite complex, the design methodology would be flawed if these two properties are decoupled during the training phase of the network [138, 219, 226, 227].

Additionally, the design of ANN involves the optimization of two competing objectives, namely the maximization of network capacity and the minimization of neural architecture complexity. Therefore, it is not surprising that multi-objective evolutionary algorithms (MOEAs) have been applied with great success to the concurrent optimization of both architecture and connection weights recently [2, 1, 46, 57, 60]. Abbass [2] presents a multiobjective (MO) memetic Pareto ANN (MPANN), which is based on differential evolution and applies the backpropagation algorithm as the local search (LS) operator. The author further extends his original approach by suggesting an alternative self-adaptive algorithm that is claimed to be computationally less intensive. While Abbass [2, 1] sought to achieve a good generalization by optimizing the objectives of training accuracy number of hidden neurons simultaneously, Giustolisi and Simeone [60] considers the additional objective of model input dimension. Garcia-Pedrajas et al [57] investigated the influence of 10 different objectives and results indicated the advantages of the MO approach over single objective (SO) approaches.

However, the simultaneous evolution of both architecture and network weights will inevitably result in a large increase in the size of the search space. While evolutionary algorithms (EAs) are capable of exploring and identifying promising regions of the search space, they require a relatively

longer time to locate the local optima. Support from experimental studies has shown that EA-LS hybrids or hybrid EAs are capable of more efficient search capabilities [139, 145]. Therefore, one of the goals of this chapter is to present a hybrid multiobjective EANN (HMOEN) which entails the design of appropriate representation, genetic and LS operators. One of the core design issues is the capability to evolve architecturally parsimonious networks as measured by the structural complexity of the evolved network. The ability of an ANN to generalize well depends on the matching of the network complexity with the degree of information that is inherent in the training data. Thus, in order to achieve this goal, the informational or to be more specific, the geometrical relationship between the hidden layer neurons in the evolved network needs to be obtained such that a pruning mechanism can be developed to identify in an approximate manner, the geometrically irrelevant hidden neurons to be removed.

Similar to [2, 1], the proposed HMOEN is based upon the concepts of Pareto optimality, which accounts for the inherent tradeoffs in capacity and complexity. Most of the previous work investigating optimization of network structure involves stochastic and performance-driven adaptation of network architecture which will inevitably result in larger network complexities. In contrast to these approaches, an information measure based on the singular value decomposition (SVD) is developed to quantify the contribution of the neurons in the hidden layer, for an archetypical neural network classifier. Subsequently, SVD-based architectural recombination (SVAR) is designed for the purpose of facilitating the exchange of neuronal information between candidate ANN designs and adaptation of the number of neurons for each individual. The role of the SVD operator is important, particularly in obtaining the geometrical relevancy of neurons in hidden layer space when attempting to prune a neural network through the proposed SVAR scheme to finally arrive at a parsimonious network architecture.

Together with the SVAR, two other problem specific operators including the variable length representation and the micro-hybrid genetic algorithm (μHGA) are proposed to handle the issues of architectural adaptation and local fine-tuning. The variable length representation encodes the entire ANN design with each neuron and its associated weights as the basic building block, and is designed to allow for easy adaptation of ANN architecture. The μHGA is then applied for the exploitation of connection weights and incorporates the pseudoinverse to find the optimal output weights in the least squares sense.

The organization of the chapter is as follows: some background information is provided in Section 2, while an overview of the matrix decomposition operator, namely the singular value decomposition (SVD) as well as the Moore-Penrose generalized inverse (pseudoinverse) is given in Section 3. The SVD is used here as a rank-revealing decomposition to adapt the network structure, as well as to compute the pseudoinverse. The details of the proposed features in the HMOEN are described in Section 4. Experimental studies including a comparative study with well-known approaches upon medical data sets, analysis of parameter sensitivity and individual contribution of the proposed features are conducted in Section 5. Conclusions are subsequently drawn in Section 6.

3.2 Background

This section provides the necessary background required to appreciate the work presented in this chapter. Fundamental multi-objective optimization concepts and definitions are presented in Section 3.2.2 while a brief overview of MOEAs is given in Section 3.2.1. Section 3.2.3 then introduces preliminary ANN concepts.

3.2.1 Multi-objective Optimization

At present, many real-world applications involve complex optimization problems with various objectives that are often non-commensurable, and conflicting in nature; this, together with the presence of numerous constraints makes such problems difficult, if not infeasible to solve without the support of powerful and efficient optimization algorithms. Here, without any loss of generality, a minimization problem is considered, with a decision space X, a subset of real numbers. For the minimization problem, we intend to solve for a parameter (variable) set $\mathbf{P}$ that optimizes the following objective,

$$\min_{\mathbf{P} \in X} \mathbf{F}(\mathbf{P}), \mathbf{P} \in \Re^n \tag{3.1}$$

where $\mathbf{P} = \{p_1, p_2, \ldots, p_n\}$ is a vector with a set of n decision variables and $\mathbf{F} = \{f_1, f_2, \ldots, f_m\}$ is the objective vector with m objectives to be minimized.

The concepts of Pareto dominance and Pareto optimality are fundamental in MO optimization, with Pareto dominance forming the basis of solution quality. There are three possible relationships between the solutions that are defined by Pareto dominance.

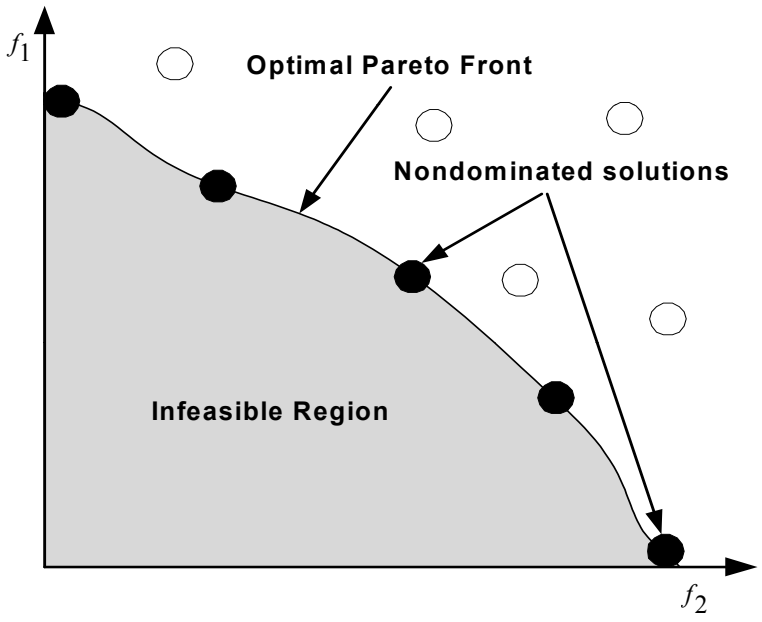

Figure 3.1: Illustration of the optimal Pareto front and the relationship between dominated and non-dominated solutions).

Definition 2.1 (Pareto Dominance): Given the objective vectors $X_1, X_2 \in \Re^m$. Then X_1 dominates X_2, denoted as $X_1 \prec X_2$ iff $x_{1i} \leq x_{2i} \forall i \in \{1, 2, \ldots, m\}$ and $x_{1j} < x_{2j} \exists j \in \{1, 2, \ldots, m\}$. Definition 2.2 (Optimal Pareto Front): The optimal Pareto Front (PF) denoted by Z^* is the set of individuals $Z^* = \{Z_j^* | Z_j^* \prec Z_i, \forall Z_i \in Z\}$.

In contrast to SO optimization, the solution to MO optimization problem exists in the form of alternate tradeoffs known as Pareto optimal set. The different dominance relationship is illustrated in Figure where the solutions denoted by closed circles form the optimal Pareto front and dominated the solutions represented by open circles. Note that each objective component of any non-dominated solution in the Pareto optimal set can only be improved by degrading at least one of its other objective components [180].

3.2.2 Multi-Objective Evolutionary Algorithms

Traditional operational research approaches to MO optimization typically entails the transformation of the original problem into a SO problem and employs point-by-point algorithms such as

branch-and-bound to iteratively obtain a better solution. Such approaches have several limitations including the requirement of the MO problem to be well-behaved, i.e. differentiability or satisfying the Kuhn-Tucker conditions, sensitivity to the shape of the Pareto-front and the generation of only one solution for each simulation run. On the other hand, meta-heuristical approaches that are inspired by biological or physics phenomena such as evolutionary algorithms and simulated annealing have been gaining increasing acceptance as a much more flexible and effective alternative to complex optimization problems in the recent years.

Among these meta-heuristics, MOEA is one of the more popular stochastic search methodologies to solve MO problems and it has been applied to solve real world problems [32,122,189,205] involving multiple non-commensurable and often competing criteria. Emulating the DarwinianWallace principle of 'survival-of-the-fittest' in natural selection and adaptation, MOEAs have the distinct advantage of being able to sample multiple solutions simultaneously. Such a feature provides the MOEA with a global perspective of the MO problem as well as the capability to find a set of Pareto-optimal solutions in a single run.

Based on the different cost assignments and selection strategies, EAs for MO optimization can be broadly classified into two classes: non-Pareto approaches and Pareto-based approaches [31]. Notable examples of non-Pareto approaches include VEGA [171], HLGA [79] and MO genetic local search algorithm [108]. Pareto-based approaches include PAES [121], SPEA2 [228] and NSGA [180]. Despite these differences, the general MOEA framework can be represented in the pseudocode shown below. The optimization process starts with the initialization and evaluation of candidate solutions. After which, good solutions are updated into an external population or archive. The selection process typically involves the archive of non-dominated solutions to improve convergence. Common genetic operators used in the literature include simulated binary crossover and Gaussian mutation for real-number MOEAs while uniform crossover and bit-flip mutation is applied for binary MOEAs. The iterative process of evaluation, archive update and genetic variation will be repeated until a stopping criterion is satisfied.

```
INITIALIZE Population
EVALUATE  Candidate solution fitness
     Update Archive
```

```
REPEAT UNTIL stopping criteria satisfied  DO
     SELECTION
     GENETIC VARIATION
     EVALUATE  Candidate solution fitness
     Update Archive
END
```

3.2.3 Neural Network Design Problem

This chapter considers the problem of multilayer feedforward neural network (MLFN) design for pattern recognition or regression. Each training sample X_i, is drawn from the set $X(= x_1 \times x_2 \times \cdots \times x_N)$ is associated with a corresponding desired output label Y_i, from $Y(= y_1 \times y_2 \times \cdots \times y_N)$. The MLFN can be described by its topological structure, activation function and connection weights. Let L denote the number of layers in the network. Layer 0 corresponds to the input layer of neurons. NH_k represents the number of neurons in (hidden) layer k. W_k refers to the set of weights from layer $k-1$ to layer k, where $k = \{1, 2, \ldots, L\}$. In a similar vein, b_k represents the biases of the neurons in layer k.

Note that $W_k \in \Re^{NH_{k-1} \times NH_k}$ and $b_k \in \Re^{NH_k}$. However, for purpose of specificity, in our subsequent discussions, a single hidden layer feedforward network (SLFN) is used, as a single hidden layer approach is justified (over a multiple hidden layer approach) lies in the fact that the Universal Approximation Capability (UAC) theorem asserts that a single hidden layer of neurons would be sufficient for approximating any given function [33, 55, 98]. As such, the use of additional layers of hidden neurons might (perhaps) lead to fewer neurons being required, but the number of connections between these layers would necessarily increase, which in turn would lead to a more complex, unwieldy structure being evolved. Hence, while it is possible to develop our approach towards handling multi-layer feedforward networks, the benefits might be marginal compared to the additional computational resource and complexity required in doing so.

3.3 Singular Value Decomposition (SVD) for Neural Network Design

As mentioned in the introduction, the adaptation of network architectures is mainly stochastic or performance-driven (by the classification accuracies, in our case), which will inevitably result in larger network complexities, as measure by the architectural size, or otherwise known as structural complexity. The primary difficulty in this area is the formulation of an appropriate measure to quantify the contribution of the 'information' that emerges during the process of evolution while the learning, or training mechanism takes place. At present, it is difficult to quantify the contribution of additional neurons in the hidden layer without the use of an independent validation set of data. As such, a simple, yet robust information measure based on the SVD operator is used in the framework of EANNs, to achieve this purpose, in removing neurons in the hidden layer of the evolved single hidden layer feedforward neural network. For further insight into the use of the SVD in investigating structural complexity of neural networks, refer to [198]. This has also been discussed in the previous chapter (Chapter 2).

Computationally, the SVD is very robust and allows the discrimination against noise contamination. Typically, the SVD is utilized in computing the pseudoinverse (Moore-Penrose generalized inverse) of a rectangular, possibly singular, matrix. The SVD has also been extensively applied in problems of least squares, spectral estimation and system identification. In signal processing, the SVD plays a central role in subspace modeling or low-rank approximation (similar to our problem of estimating the number of hidden layer neurons) of signals. Please refer to [120, 182] for further technical details.

These singular values indicate the degree of mutual correlation between features in the hidden layer space with column degeneracy resulting when these hidden space features are highly correlated, which in turn leads to the conclusion that these additional neurons are redundant. While the singular values do not provide information on which of these features are correlated (the identity of the neurons are not explicitly known), the presence of small singular values would indicate that these additional hidden neurons can be removed without affecting the performance of the SLFN significantly. In application of the pseudoinverse, or otherwise known as the *Moore-Penrose Generalized Pseudoinverse*, in resolving a linear system of the general form $H\beta = T$, the approach is fairly straightforward if the

matrix H is square and non-singular. However, under many practical circumstances, the matrix H is usually singular and likely to be rectangular. The Moore-Penrose generalized inverse simplifies the treatment by providing the solution to the linear system in a least-squares sense. The pseudoinverse of H is defined differently depending on the rank and dimensionality of H.

In most practical problems, the system is over-determined and hence would want find the least-square error of $||H\beta - T||_2$ in the presence of the inconsistencies introduced by the additional equations. Thus, β is obtained from $||H\beta - T||_2$. The pseudoinverse can be shown to be the minimum norm least-squares solution of the system, i.e. the pseudoinverse of β, which is $\beta^{\dagger}$, minimizes $||H\beta - T||_2$. For further details on the pseudoinverse, readers are directed to [174].

Again, these issues have been further explained in the previous chapter (Chapter 2) as well as in [198].

3.4 Hybrid MO Evolutionary Neural Networks

3.4.1 Algorithmic flow of HMOEN

To design an EANN that is capable of evolving the architecture and weights of the ANN simultaneously, a few features such as variable-length chromosome representation, specialized genetic operator in the form of the SVD-based Architectural Recombination (SVAR), and micro-hybrid genetic algorithm (μHGA) for effective local search are incorporated in HMOEN. The flow of the HMOEN is shown in Figure 3.2. The design process begins with the initialization of population. All individuals will be evaluated according to the objective functions and ranked according to their dominance relationship in the population. The objective functions and ranking scheme will be described in Section 3.4.2.

After the ranking process, non-dominated solutions will be updated into the archive. This chapter applies a fixed-sized archive. Elitism is implemented by a two-stage process; 1) the archive and evolving population are combined and 2) binary tournament selection of this combined population is conducted to fill up the mating pool. In binary tournament selection, two individuals are selected randomly from the combined population and compared based on the Pareto rank. The individual

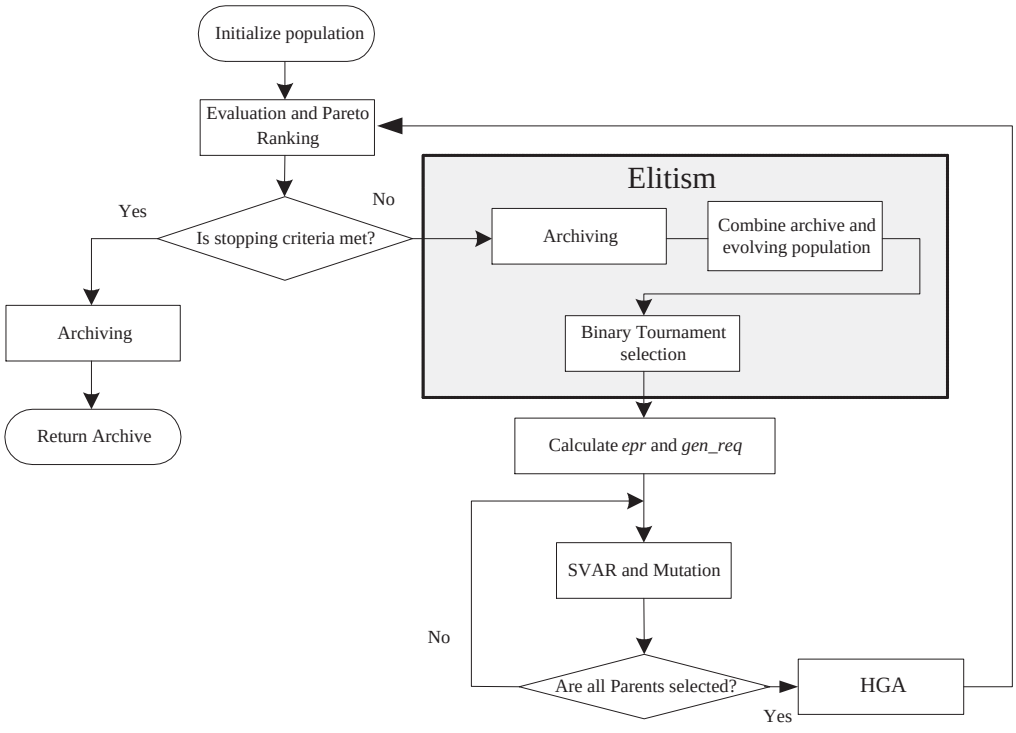

Figure 3.2: Algorithmic Flow of HMOEN.

with a lower rank, i.e, better is selected for genetic variation. In the event of a tie in Pareto ranks, niche count is used as the tie-breaker.

The selected ANNs will then undergo the process of SVAR, which adapts the network architecture, and the mutation process. In order to reduce the noise presented by the change in network architecture as well as to improve convergence, the offspring are allocated to the for local exploitation. The evolution process repeats until the stopping criterion is satisfied. The mechanisms of SVAR and μHGA are described in Section 3.4.4 and 3.4.5 respectively.

3.4.2 MO Fitness Evaluation

The ANN design is cast as a MO optimization problem where a number of objectives such as training accuracy and degree of complexity can be specified. The conflicting objectives of maximizing network capacity and minimizing network complexity is manifested in the tradeoffs between training and test

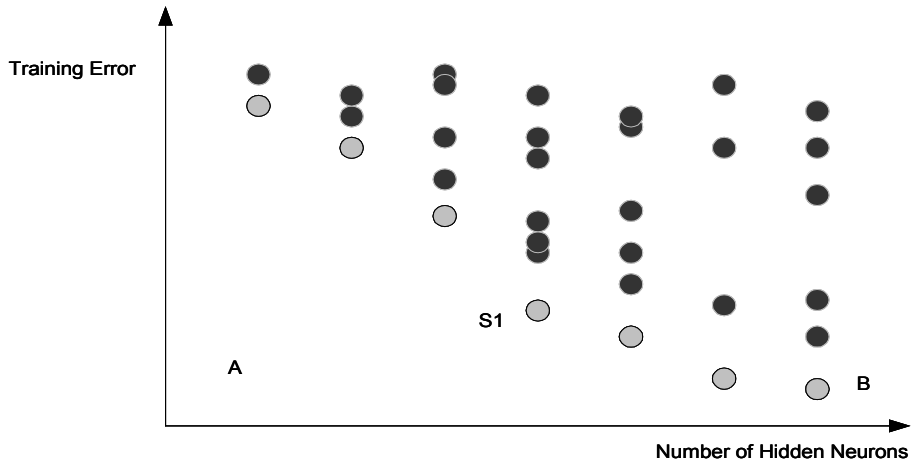

Figure 3.3: Tradeoffs between training error and number of hidden neurons.

accuracy. As before, the Pareto ranking scheme [52] which assigns the same smallest cost for all non-dominated individuals. One of the primary reasons why a weighted objective is not favored is due to the fact that it is difficult to properly apportion the weights that should be associated with each objective, in converting a MO problem into a SO problem. Most objectives that are considered, such as training accuracy and size of neural network weights are not commensurable (not of the same dimensional quantity) which makes it rather difficult to place these two objectives on a similar platform for comparison.

Figure 3.3 illustrates the concept of Pareto optimality in ANN design where each closed circle represents an ANN design evolved by the MOEA. The gray and black circles denote non-dominated and dominated solutions respectively. Point A represents the utopian point which may not exist. Solution A represents the compromise between training error and number of neurons. In the context of single objective optimization of ANN where training error is the sole concern, the evolutionary process will inevitably drive the solutions towards point B.

1) Pareto Ranking: The Pareto ranking scheme [52] is adopted in this chapter based on the objectives of minimizing training error and network complexity. This scheme assigns the same smallest cost for all non-dominated individuals, while the dominated individuals are ranked according to how many individuals in the population dominate them. So, the rank of an individual in the

population will be given by

$$rank(i) = 1 + n_i \tag{3.2}$$

where n_i is the number of individuals dominating the individual i in the objective domain.

2) Optimization Objectives: One of the primary reasons why a weighted objective is not favored is due to the fact that it is difficult to properly apportion the weights that should be associated with each objective, in converting a MO problem into a SO problem. Most objectives that are considered, such as training accuracy and size of neural network weights are not commensurable (not of the same dimensional quantity) which makes it rather difficult to place these two objectives on a similar platform for comparison.

In this chapter,the simultaneous evolution of both the neural architecture as well as the synaptic weights is considered. Further, this problem is distinguished from previous work by formulating the problem as a MO problem where the twin objectives of classification accuracy and network complexity are conflicting in nature. Therefore, the optimization problem for the ANNs generalization on unseen data can be written as

$$\begin{aligned} f_1 &= \min\Big\{\sum_{\theta=1}^{N}\sum_{k=1}^{C}\big(y_k(\theta) - d_k(\theta)\big)^2\Big\} \\ f_2 &= \min\{NH_k\} \\ f_3 &= \min\{||W_k||_2\} \end{aligned} \tag{3.3}$$

where f_1 refers refers to the sum-of-squared (SSE) errors of the classification errors. In our problem formulation, only one hidden layer of neurons (k=1) is used, as dictated by the Universal Approximation Capability (UAC) theorem for neural networks. N is the number of samples, C is the number of class and d_k is the desired output.

The two other objectives that are considered in the proposed MO framework are firstly, the minimization of the number of neurons in the hidden layer (f_2), and secondly the minimization of the L_2-norm of the hidden layer weights (f_3). f_2 and f_3 are different methods of controlling the size of the network complexity. f_2 is the conventional manner in which MO methods approach the (direct) control of structural complexity in ANNs, while f_3 has its roots in [16]. With that in mind, we intended to analyze and compare empirically, the performance of these two structural complexity control methods.

3) Diversity Preservation: The approximation of the Pareto optimal front requires the MOEA to perform a multi-directional search simultaneously to discover multiple, widely different solutions. This requires a substantial amount of diversity in the evolving population. According to Mahfoud [135], a simple elitist EA tends to converge towards a single solution and often loses solutions due to the effects of selections pressure, selection noise, drifting, and operator disruption.

The use of niche sharing was proposed by Goldberg and Richardson [64] to prevent genetic drift and to promote the sampling of the whole Pareto front by the population. The basic idea is that individuals in a particular niche should share the available resources and the fitness value of an individual is degraded as the number of individuals in its neighbourhood increases. Niching can be performed in either the decision variable domain or the objective domain. The choice between niching in decision variable domain or objective domain depends on what is desired for the underlying problem. In general, niche sharing is achieved using a sharing function. Let d be the Euclidean distance between x and y. The neighbourhood size is specified by the so-called niche radius σ_{share}. The sharing function is defined as follows,

$$sh(d) = \begin{cases} 1 - (d/\sigma_{share} & \text{if } d < \sigma_{share} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

And the niche count function is defined with the help of sharing function,

$$nc(x) = \sum_{y} sh(d(x, y)) \tag{3.5}$$

The niche radius σ_{share} is a key parameter that affects MOEA's effectiveness. Improper setting of the value will result in bad distribution of the population. In practice, the niche radius is difficult to estimate because there is no a priori knowledge about the shape of the Pareto front for many problems. Fonseca and Fleming [52] gave some bounding guidelines of appropriate σ_{share} values for multi-objective problems when the number of individuals in the population and the minimum/maximum values in each objective dimension are given. In this chapter, the objective space is normalized online based on the evolved tradeoffs and the sharing distance is set as $\sigma_{share} = \frac{1}{archive\ size}$. This niche count approach will be used in the selection and archive updating process.

3.4.3 Variable Length Representation for ANN Structure

EAs process a set of encoded parameters, providing its designers with the flexibility to design an appropriate representation of the potential solutions (the candidates). Appropriate representation implies that it fulfils some criteria such as ease of implementation or exploitation of the problem structure. For simplicity, the chromosome is often represented as a fixed-structure and the embedded variables are usually assumed to be independent and context insensitive. In EANNs, a hybrid structure between binary and real-number representation is commonly used for the simultaneous optimization of weights and architecture. However, such a representation is not suitable for the ANN design problem where flexibility is essential for the simultaneous evolution of both architecture and connection weights.

In this chapter, a variable-length chromosome representation is adopted to represent the ANN topology including the number of neurons in the hidden layers and the connection weights linking the input, hidden and output layer, as illustrated in Figure 3.4(a). Figure 3.4(b) is the instantiation of the representation in Figure 3.4(a). Each neuron is coupled with its associated weights, thus allowing easy manipulated by search operators for the addition or deletion of neurons. The chromosome may consist of different number of neurons which reflects the complexity of the ANN but the number of connections is fixed by the number of input attributes. Such a representation is efficient and facilitates the design of problem-specific genetic operators.

3.4.4 SVD-based Architectural Recombination

In EANNs, the recombination process between two ANNs is unlikely to produce a 'good' offspring due to the lack of a clear definition of what defines a 'building block' in the framework of ANNs [218]. However, the lack of recombination to facilitate the exchange of information between candidate solutions implies that each individual is expected to adapt independently by making best use of all available 'local' information (confined within its own genetic pool). This motivates the development of the SVAR approach which is based on the fact that each neuron constructs a hyperplane in the input feature space and hence contributes to the resulting separating capability of the ANN. It follows that each neuron and its associated connections defines a (somewhat coarse-grain) 'building block' which contributes to the capacity of the ANN.

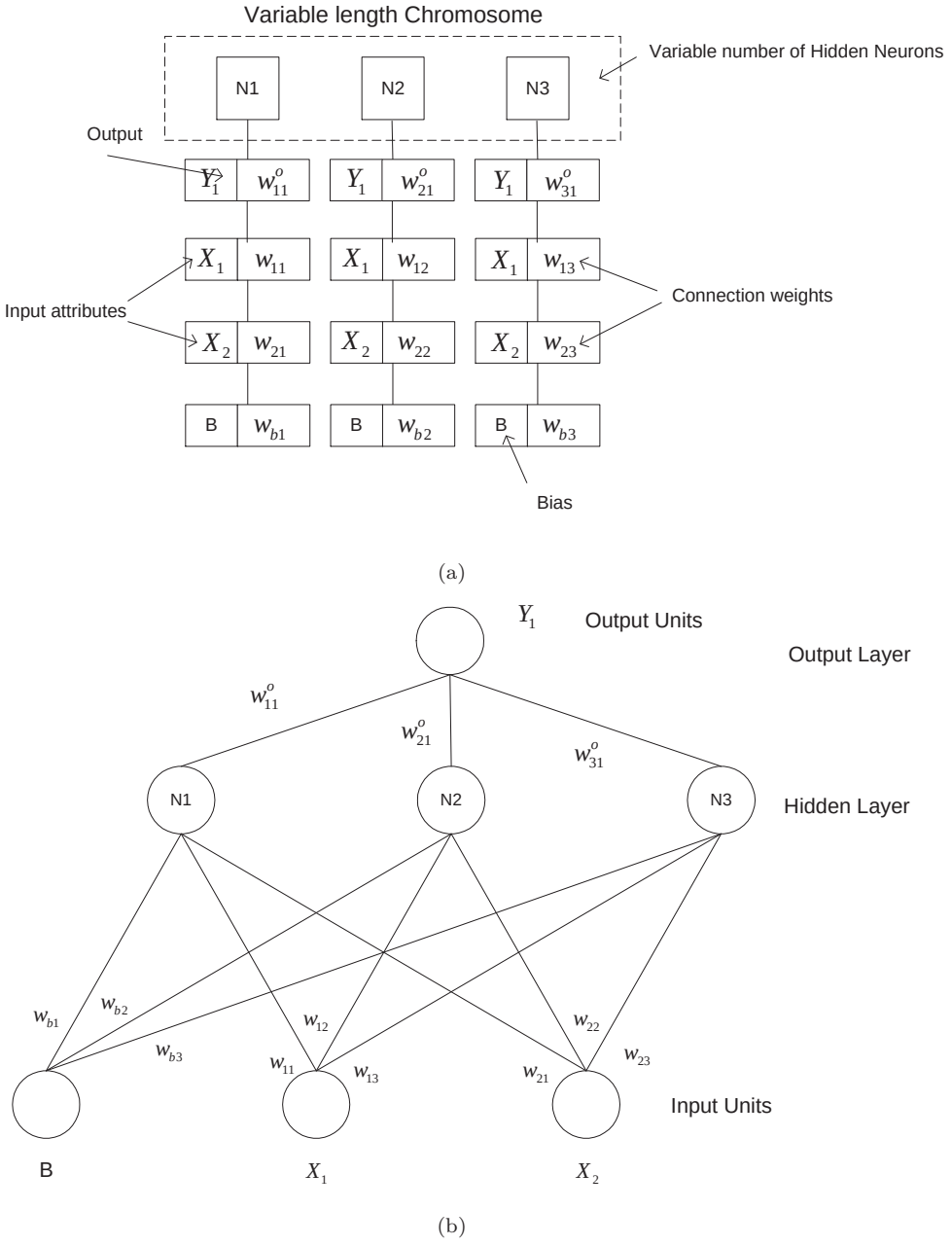

Figure 3.4: An instance of the variable chromosome representation of ANN and (b) the associate ANN.

The issues considered in the design of the SVAR operator include the selection of the appropriate neuron and its associated weights for recombination as well as the decision to remove or add an appropriate neuron to the candidate ANN design. SVAR is performed between two parent ANNs and the procedure is outlined in the following pseudocode (Figure 3.5).

```
SVD Determine presence of redundant neurons for both parents – the SVD
operator is applied, together with a corresponding threshold, to determine
the number of (redundant) hidden layer neurons to 'prune', or remove.
SELECTION Calculate intra- and inter- subspace angle between neurons.
Mark neurons with smallest intra-subspace angle and largest inter-subspace
angle for removal and exchange respectively.
FOR both parents DO
    IF U(0,1) < 1/3
        EXCHANGE neurons marked for exchange
    ELSEIF redundant neuron is TRUE
        REMOVE neurons marked for removal
    ELSE
        ADD neurons marked for exchange
    END
END
```

Figure 3.5: SVAR pseudocode.

For our proposed approach, the building blocks of each network are the set of neurons (together with its incoming weights from the previous layer). These are known as 'building blocks', as they are the 'smallest' units for which are operated on (such as performing crossover). The SVD is used as the tool to determine the presence of redundant neurons while the calculation of inter- and intra-subspace angles is used for the selection of neurons to be removed or exchanged the SVD operator (together with the corresponding threshold) will decide the *number* of hidden layer neurons to be removed.

The number of neurons in the hidden layers that are deemed redundant by the SVD operator is in effect, a function of the threshold that is used, with ε assuming the role of the SVD threshold. In deciding *which* hidden layer neuron to remove or prune, a geometrical approach is used, where the algorithm examines the subspace spanned between the hidden layer neurons such that the neuron(s) which is (are) most linearly correlated with the other hidden layer neurons are consequently removed. This is to prevent unnecessary removal of a neuron at the initial stages where the weights are not yet adapted to the problem.

The rationale for utilizing subspace angles as the selection criterion for pruning and exchange is to encourage the linear independency between the neurons. In [198], the authors use the SVD operator to first determine the appropriate, or necessary number of hidden layer neurons on an initially large structured feedforward neural network, after which, the network is retrained using the same learning algorithm, but using the reduced set of hidden layer neurons. In our approach, however, an online method is adopted, in that during the evolutionary process, the identity of the hidden layer neuron(s) to be removed are determined geometrically, by the subspace approach as described above. From the flowchart, it is observed that it is possible for a candidate ANN with redundant neurons to retain the same structure via the exchange of neurons from the other parent.

3.4.5 Micro-Hybrid Genetic Algorithm

As mentioned in the introduction, the optimization of neural network structure is an inherently noisy problem, i.e., the immediate neural network fitness after the recombination process may not be a good indicator of the new network structure due to an inappropriate set of weights. Thus, it is necessary to optimize the synaptic weights with respect to the new ANN structure after any structural changes. The mutation operator offers a simple option for local fine-tuning [109]. However, domain information cannot be easily incorporated and its stochastic nature tends to render the search operation inefficient. Intuitively, the change in genetic structure should be ordered instead of being left to chance in order for the local search to be robust. While the well-known back-propagation (BP) algorithm is a directed by means of gradient descent, it is prone to being trapped in local optima. In view of these concerns, the EANN is hybridized with the μHGA .

μHGA

THe μHGA exploits the synergy between a μGA [113] and the pseudoinverse operator to decompose the large and complex search space. Specifically, the μGA performs local fine-tuning of the hidden layer weights while the pseudoinverse optimizes output weights in the least squares sense based on the weights found by the μGA. The pseudocode of the μHGA is shown in Figure 3.6 where POP_SIZE_{LS} is the population size of the μHGA. In this chapter, simulated binary crossover (SBX) and uniform mutation (UM) is applied to evolve the desired set of connection weights. The description of SBX

and GM can be found in [39]. The mutation strength of UM is adapted as,

$$s = 0.1 \cdot (uppbd_w - lowbd_w) \tag{3.6}$$

where $uppbd_w$ and $lowbd_w$ corresponds to the minimum, maximum of the associated weights in the population.

INITIALIZE *POP* by create POP_SIZE_{LS} variants of selected ANN
EVALUATE POP_SIZE_{LS} ANNs
STORE best ANN
REPEAT UNTIL *gen_req* reached **DO**
  SELECTION: Insert best ANN into mating pool. Binary tournament POP_SIZE_{LS} ANN from *POP*
  CROSSOVER: Perform SBX on selected neurons
  MUTATION: Perform UM on selected neurons
  PSEUDOINVERSE
  EVALUATE ANNs
  STORE best ANN
END

Figure 3.6: μHGA pseudocode.

Balance between Evolution and Learning

While the incorporation of local search can accelerate convergence of the evolutionary optimization process, hybrid EAs also give rise to issues pertinent to the tradeoffs between evolution and learning. Apart from the obvious the challenge posed by limited computational resources, balance between exploration and exploitation is necessary to maintain diversity in the evolving population for the approximation of the Pareto optimal front. Consequently, these concerns have lead to the recent development of resource utilization schemes such as local search probability [109] and simulated heating [15].

While local search probability can reduce the computational time utilized for local fine-tuning, the *exploration-exploitation* dilemma is not explicitly considered. The fundamental idea behind simulated heating is based on simulated annealing, where the intensity of local search increases with time. Although, it is intuitive that more computational time for local search should be allocated in

the later stages, online search requirements are not considered in simulated heating. In contrast to existing methods which allocate resources based on a predetermined schedule, this chapter adapts the allocation of resources based on the feedback of an online performance measure, the evolutionary progress rate [192]. The evolutionary progress rate (epr(n)) can be defined as the ratio of the number of new non-dominated solutions discovered in generation n, $new_n ondomsol(n)$, to the total number of non-dominated solutions, $total_n ondomsol(n)$,

$$\text{epr}(n) \quad = \quad \frac{new\_nondomsol(n)}{total\_nondomsol(n)} \tag{3.7}$$

The set of new non-dominated individuals discovered at each generation is basically composed of individuals that dominate the non-dominated individuals of the previous generation and individuals that contribute to the diversity of the solution set.

In this adaptive scheme, the number of individuals allocated for LS is adapted based on the epr(n) in every generation. Mathematically, the adaptation of computational resource allocation can be written as,

$$gen\_req(n) \quad = \quad \big(1 - \text{epr}(n)\big) \cdot \big(upp\_bd_{com} - low\_bd_{com}\big) + low\_bd_{com} \tag{3.8}$$

where gen_req is the number of generations performed by μHGA while upp_bd_{com} and low_bd_{com} denote the upper and lower limits of available computational resource. The rationale is that a high value of epr(n) means that the algorithm is still in the exploratory stage and the need for local fine-tuning is low. Likewise, a low value of epr(n) is an indication of convergence and more resources are required to meet the requirements of local fine-tuning. In this chapter, upp_bd_{com} and low_bd_{com} are set as 20 and 10 of the total population size respectively.

Archiving

In our algorithm, elitism is implemented in the form of a fixed-size archive to prevent the loss of good particles due to the stochastic nature of the optimization process. The size of the archive can be adjusted according to the desired number of individual distributing along the tradeoff in the objective space. The archive is updated at each cycle, e.g., if the candidate solution is not dominated by any

members in the archive, it will be added to the archive. Likewise, any archive members dominated by this solution will be removed from the archive. In order to maintain a set of uniformly distributed nondominated individuals in the archive, the dynamic niche sharing scheme is employed. When the predetermined archive size is reached, a recurrent truncation process [115] based on niche count is used to eliminate the most crowded archive member.

3.5 Experimental Study

3.5.1 Experimental Setup

In order to evaluate the effectiveness of the proposed methods, a detailed empirical study is carried out on seven different datasets. HMOEN is implemented using the MATLAB technical computing platform, and corresponding simulations are performed on an Intel Pentium 4 2.8 GHz computer. Sixty independent runs are performed for each of the dataset to obtain statistical information such as consistency and robustness of the algorithms. The various parameter settings of HMOEN are tabulated in Table 3.1. For all experiments, evolution of the ANNs is terminated once the mean training accuracy of the archived solutions stops improving for three consecutive generations.

In the training phase for the classifiers, 30-fold cross-validation is used, partitioning the data into two independent training and testing sets. 60% of the available samples are randomly selected as training data, with the remaining 40% as testing data. Prior to training, pre-processing is carried on the samples of each dataset. All input features are scaled and transformed such that the resulting input features have a mean of 0 and a variance of 1, as it has been shown that convergence is usually faster if the average of each input variable over the training set is close to zero [127]. For the outputs, as classification datasets are the focus, binary target values with a 1-out-of-C encoding are used - where for a C-class problem, the largest output i is assigned to class i, with $i = \{1, 2, , C\}$. For the μth training sample, the desired class output c where $d_k(\theta) = \{0, 1\}$ is 1 for $k = c$ and 0 otherwise. This is essentially a winner-take-all approach for the output layer neurons, and is a common approach used for classification purposes. Hidden layer neurons use a hyperbolic tangent nonlinearity, while the output nodes use a linear output activation function.

Table 3.1: Parameter settings of HMOEN for the simulation study

Parameter	Settings
Population	Main Population size: 20; Archive size: 20
	μHGA: 4
	Archive (secondary population) size: 20
Chromosome	HMOEN: Variable Length real number representation;
	μHGA: Real number representation;
Selection	Binary tournament selection
Crossover operator	HMOEN: SVAR
	μHGA: SBX
Crossover rate	0.9
Distribution index	10
Threshold (SVAR)	0.995
Mutation operator	Normally (Gaussian) distribution
Mutation rate	0.1
Mutation strength	Adaptive
Niche radius	Dynamic

The real-world datasets used in this chapter, represent some of the most challenging problems in machine learning, were obtained from the UCI machine learning database [1]. Many researchers have used these datasets in validating the performances of their algorithms, and thus these datasets provide a good test suite of problem for evaluation of the proposed approach. The key characteristics of these problems and their associated learning tasks are summarized in Table 3.2.

3.5.2 Analysis of HMOEN Performance

Experimental Results

Table 3 and Table 4 show the results of HMOEN over 60 independent runs on the seven problems. HMOEN_HN denotes that ANNs are evolved based on the criteria of f_1 and f_2 while HMOEN_L2 denotes ANNs are evolved based on the criteria of f_1 and f_3. The test and training results are based on the ANN with the best training accuracy on the dataset from each run. Network size refers

[1] (http://www.ics.uci.edu/$\tilde{}$mlearn/MLRepository.html)

Table 3.2: Characteristics of Data Set

Dataset	Samples	Attributes	Classes	Remarks
Cancer	699	9	2	Determine the patients for whom the tumour is benign or malignant
Pima	768	8	2	Determine whether a patient shows sign of diabetes according to World Health Organization Criteria
Heart	297	13	2	The learning task is to predict the presence or absence of heart disease given the results of various medical tests carried out on a patient.
Hepatitis	155	19	2	The hepatitis problem is a complex and noisy dataset as it contains a large number of missing data (there are 167 missing values in total in this dataset). The learning task is to predict whether a patient with hepatitis will live or die.
Horse	368	27	2	The objective here is to determine, based on the physical ailments and attributes of a particular horse, if it should have surgery performed on it.
Iris	150	4	3	This dataset is perhaps the best-known database to be found in pattern recognition literature. One class is linearly separable from the other two; the latter are NOT linearly separable from each other.
Liver	345	7	2	The learning task for this dataset is to determine, if the adult male that is tested using blood tests suffer from liver disorders that might arise from excessive alcohol consumption

to the mean of the number of hidden layer neurons of the associated ANNs. With the exception of Hepatitis, the networks evolved by HMOEN_HN and HMOEN_L2 have comparable sizes. For instance, the mean network size evolved by HMOEN_HN and HMOEN_L2 is 9.8667 and 9.6833 percent respectively. This is probably because architectural adaptation is governed by the same mechanism of SVAR. On the other hand, apart from Liver, the paired T-test reveals that the different optimization criteria have a significant impact on test accuracies. In particular, HMOEN_L2 performs significantly better for the problems of Pima, Hepatitis, Horse and Iris while HMOEN_HN only fares slightly better for Cancer and Heart.

Method		Data Set						
		Cancer	Pima	Heart	Hepatitis	Horse	Iris	Liver
HMOEN_HN								
TRAINING	Mean	0.9816	0.8075	0.9021	0.9668	0.9991	1	0.797
	Std	0.0015	0.0053	0.0098	0.0113	0.002	0	0.0098
TESTING	Mean	0.9682	0.7536	0.8106	0.7551	0.977	0.9103	0.6894
	Std	0.0058	0.0182	0.0261	0.0441	0.0171	0.0313	0.0271
NETWORK SIZE	Mean	4.8	8.0833	9.6833	10.7	9.15	2.6833	6.7667
	Std	1.8485	1.8712	3.1218	3.5668	5.3673	1.0813	1.1552

Figure 3.7: HMOEN_HN Performance on the Seven Different Datasets. The Table Shows the Mean Classification Accuracy and Mean Number of Hidden Neurons for all Datasets

Method		Data Set						
		Cancer	Pima	Heart	Hepatitis	Horse	Iris	Liver
HMOEN_L2								
TRAINING	Mean	0.9825	0.7954	0.9035	0.9498	0.9991	0.9839	0.8059
	Std	0.0017	0.0052	0.0088	0.0164	0.0022	0.0056	0.0119
TESTING	Mean	0.9626	0.7845	0.7969	0.803	0.9838	0.98	0.68
	Std	0.011	0.0122	0.0294	0.048	0.0131	0.0184	0.0294
NETWORK SIZE	Mean	4.6667	7.5167	9.8667	11.3833	7.2	3.0833	6.8333
	Std	1.5367	2.446	3.5865	3.2682	5.0583	0.8693	1.2374

Figure 3.8: HMOEN_L2 Performance on the Seven Different Datasets. The Table Shows the Mean Classification Accuracy and Mean Number of Hidden Neurons for all Datasets.

Effects of proposed features

In this section, the effects and contribution of multiobjectivity, and the proposed features of SVAR and are examined for the different datasets. Note that only HMOEN_L2 is used in the study here since it has been observed in previous section that f_1 and f_3 are generally the better optimization criteria. Here, different case setups are used to represent different combinations of features in HMOEN_L2. The different cases are summarized in Table 5. In Case 1, f_1 and f_3 are aggregated with the weight vector $[0.8 \;\; 0.2]$, i.e. $F = 0.8f_1 + 0.2f_3$. HMOEN_L2 is represented as Case 5. The variable-length chromosome, Gaussian mutation operator and archive mechanism remain fixed for the five cases.

The distributions of the classification accuracy and network size are represented by box-plots in Figure 9 and Figure 10 respectively. By comparing Case 1 (which is the SO version of HMOEN) and Case 5 in Figure 9, is can be noted that the MO approach is generally better with similar structural complexities. The paired-T test conducted also indicates that the performance between Case 1 and

Case	Settings		
	Optimization Goal	SVAR	μHGA
1	Weighted	Yes	Yes
2	MO	No	No
3	MO	Yes	No
4	MO	No	Yes
5	MO	Yes	Yes

Figure 3.9: Different Case Setups to Examine Contribution of the Various Features.

Case 5 is statistically different in all problems except Cancer.

The purpose of conducting variants of HMOEN_L2 with and without SVAR is to ascertain the contribution of the proposed operator. The effects of SVAR can be observed by comparing the performances between Cases 2 and 3, and Cases 4 and 5 in Figure 9. Clearly, without the use of the SVD as a form of capacity control in SVAR, the performance demonstrated in Case 2 and Case 4 is inferior to Case 3 and Case 5 respectively for most of the problem. In addition, comparable, if not better, classification accuracies are achieved with smaller networks as evident in Figure 10. These results substantiate our earlier hypothesis that each (hidden) neuron, together with its corresponding hidden layer weights (leading from the input layer to the hidden layer), functions as a building block for an EANN. The specialized recombination operator acts specifically on these neuronal building blocks. This notion is intuitively appealing because when viewed from the perspective of hidden layer space, each hidden neuron and its input set of weights (for the single hidden layer) constructs a separating hyperplane in hidden layer space; thus, each hidden neuron together with its corresponding set of input weights, which are treated as a set of building blocks, are accountable for determining the separation of the training samples in hidden layer space.

By comparing the performance between Case 2 and 4, and Case 3 and 5, it is clear that the introduction of the μHGA provides improvements to the classification performance on the testing sets of all the datasets. It can be noted that the local search ensures that the final network is sufficiently well-trained such that the SVD is able to operate on the hidden layer activation matrix effectively. Recall that the use of the SVD, as described earlier requires the network to be 'well-trained'. In other words, without the μHGA, the SVAR tend to remove neurons excessively as reflected in the

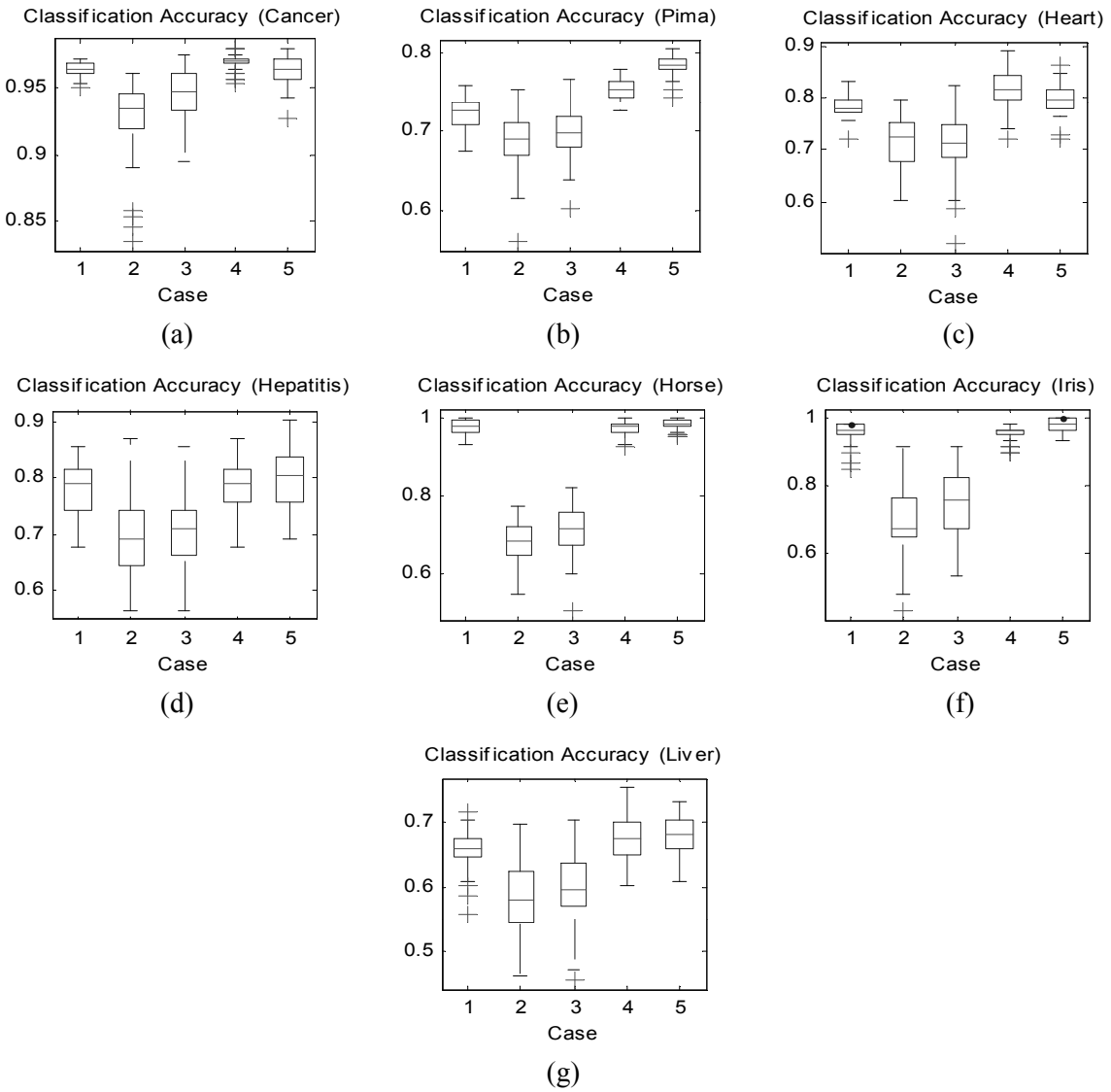

Figure 3.10: Test Accuracy of the Different Cases for (a) Cancer, (b) Pima, (c) Heart, (d) Hepatitis, (e) Horse, (f) Iris, and (g) Liver.

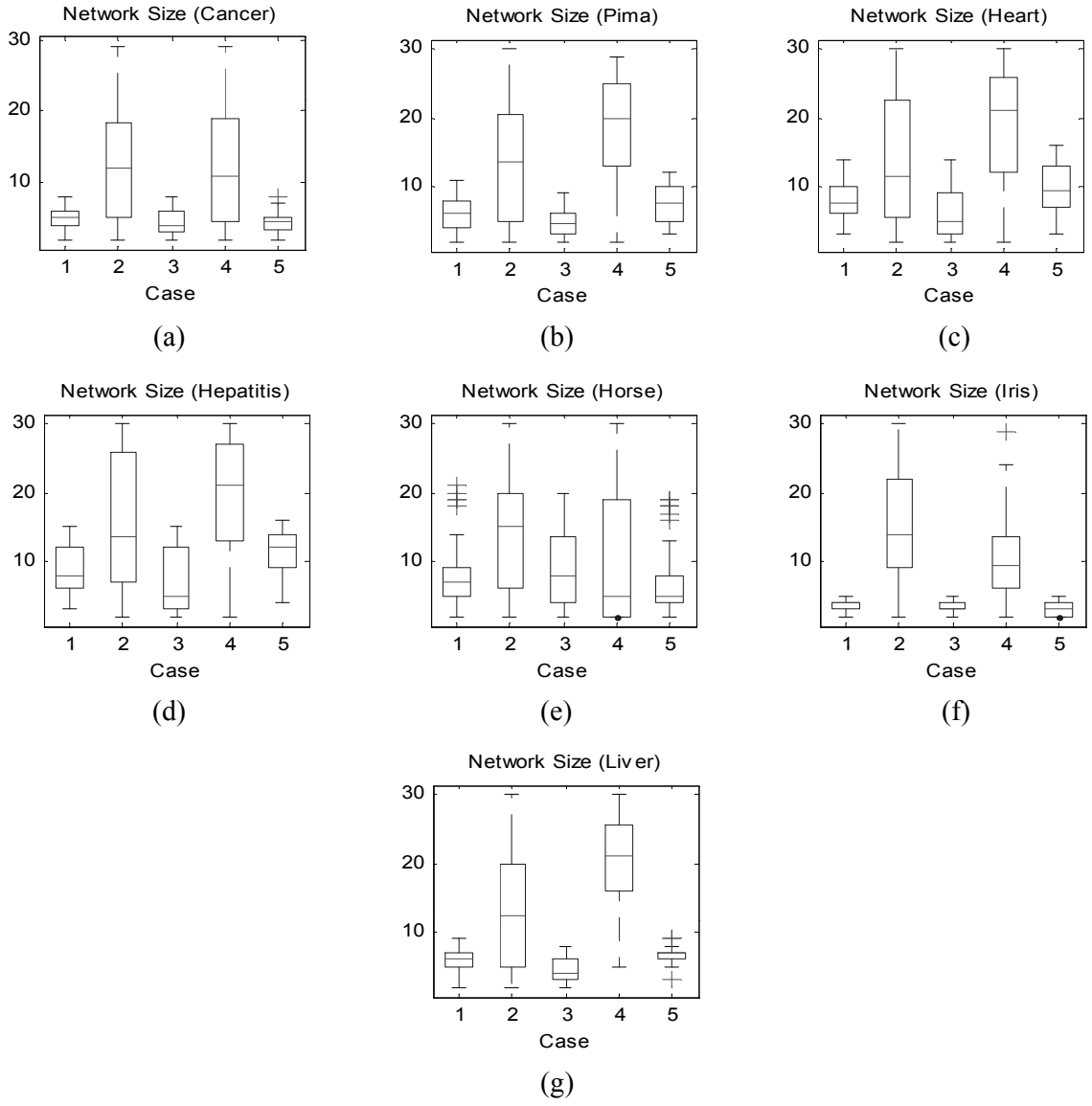

Figure 3.11: Test Accuracy of the Different Cases for (a) Cancer, (b) Pima, (c) Heart, (d) Hepatitis, (e) Horse, (f) Iris, and (g) Liver.

generally lower number of hidden neurons (large number of neurons is pruned). Therefore, it is also evident that that SVAR and the μHGA are complementary mechanisms in HMOEN.

Effects of SVD threshold settings

It can be observed in the previous section that the SVAR allows HMOEN to evolve smaller networks with comparable, if not better, classification accuracies. In this section, experiments are conducted for the various datasets over SVD threshold settings of $\{0.9, 0.95, 0.98, 0.99\}$ to investigate its effects on network structure and classification performance. Trends of testing accuracies and network sizes over the four threshold settings for the different datasets are plotted in Figure 11 and Figure 12 respectively. Theoretically, the size of the network is expected to increase as the SVD threshold becomes larger - this can be explained from the fact that the 'pruning' mechanism, implemented through the proposed SVD-based crossover, becomes stricter so as to maintain more of the spectral energy of the singular values, hence requiring that more hidden layer neurons need to be kept. This conjecture is reinforced from the Figure 12 shown where a monotonically increasing trend is seen for the architectural or structural complexity (as measured by the number of hidden layer neurons) as the SVD threshold is progressively increased from 0.95 to 0.99.

From a conventional and theoretical perspective, the trend for the training and testing accuracy is usually strongly positively correlated up to a certain point, beyond which the performance of the classifier on the test set degrades where the correlation between these 2 sets becomes negative, in that the training accuracy steadily increases while the testing accuracy drops. Continuing the training process beyond this point would result in the classifier being overtrained, and any subsequent training, while increasing the training accuracy, will always degrade the performance of the classifier on the testing set. Overtraining occurs in two ways either through an overly strict learning algorithm (with excessively many training epochs) or when the architecture of the network is overly complex (with excessively many hidden layer neurons) the former applies to the traditional learning route for neural networks, while the latter applies to our approach, where the SVD threshold is increased during the pruning mechanism which in turn is implicitly implemented via the SVD-based crossover. The SVD threshold can be understood from the perspective of architectural complexity - a higher SVD threshold means larger networks with more hidden layer neurons are evolved. However, using the proposed approach (based on Figure 11), using a larger SVD threshold does not necessarily

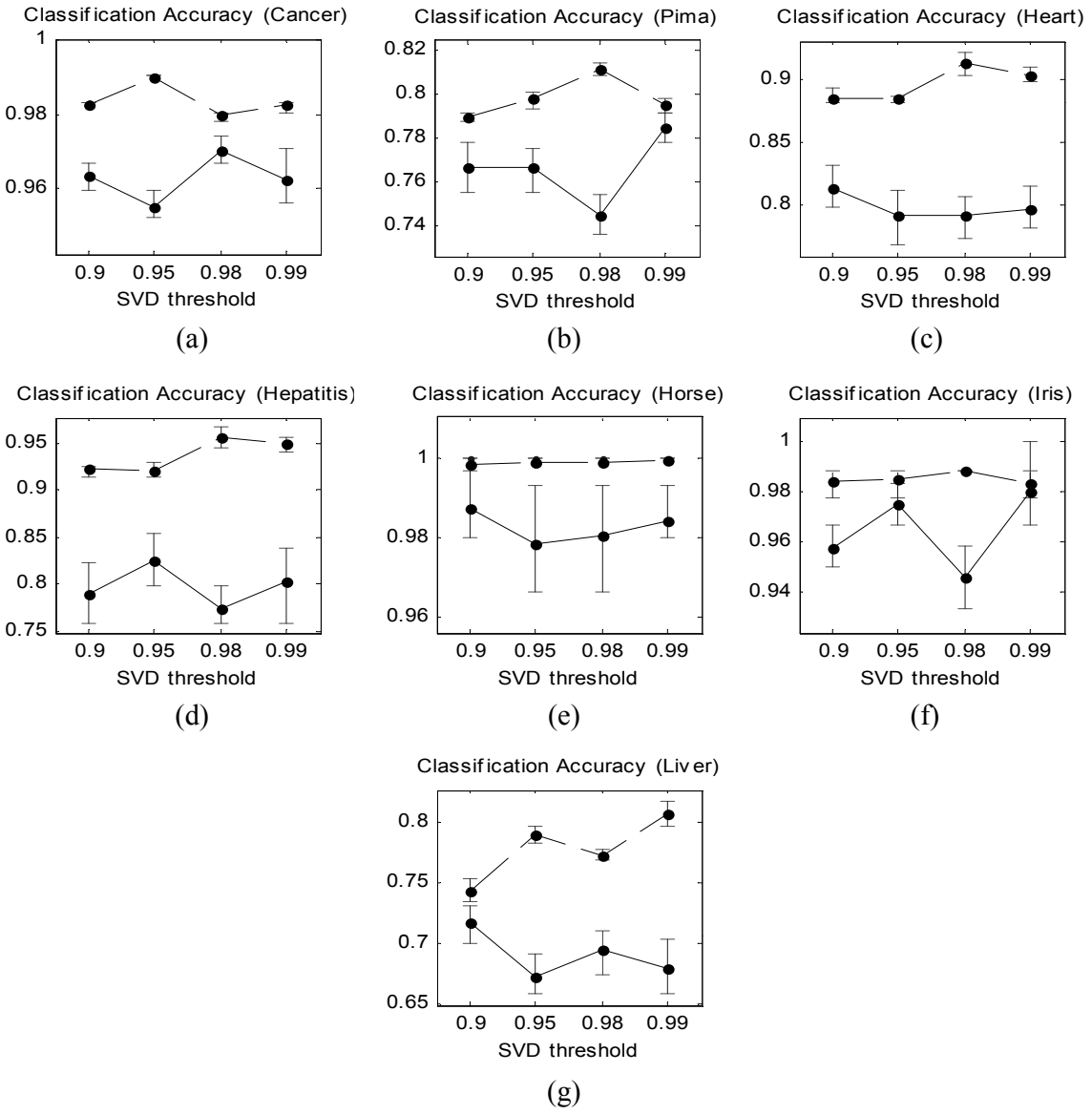

Figure 3.12: Trend of Training Accuracy (–) and Testing Accuracy (-) over different SVD threshold settings for (a) Cancer, (b) Pima, (c) Heart, (d) Hepatitis, (e) Horse, (f) Iris, and (g) Liver. The trend is connected through the mean while the upper and lower edges represent the upper and lower quartiles respectively.

lead to a lower performance in generalization, as measured by the accuracy on the testing set. Alternatively, this can be understood from the perspective of the proposed method being able to 'prevent' overtraining from occurring.

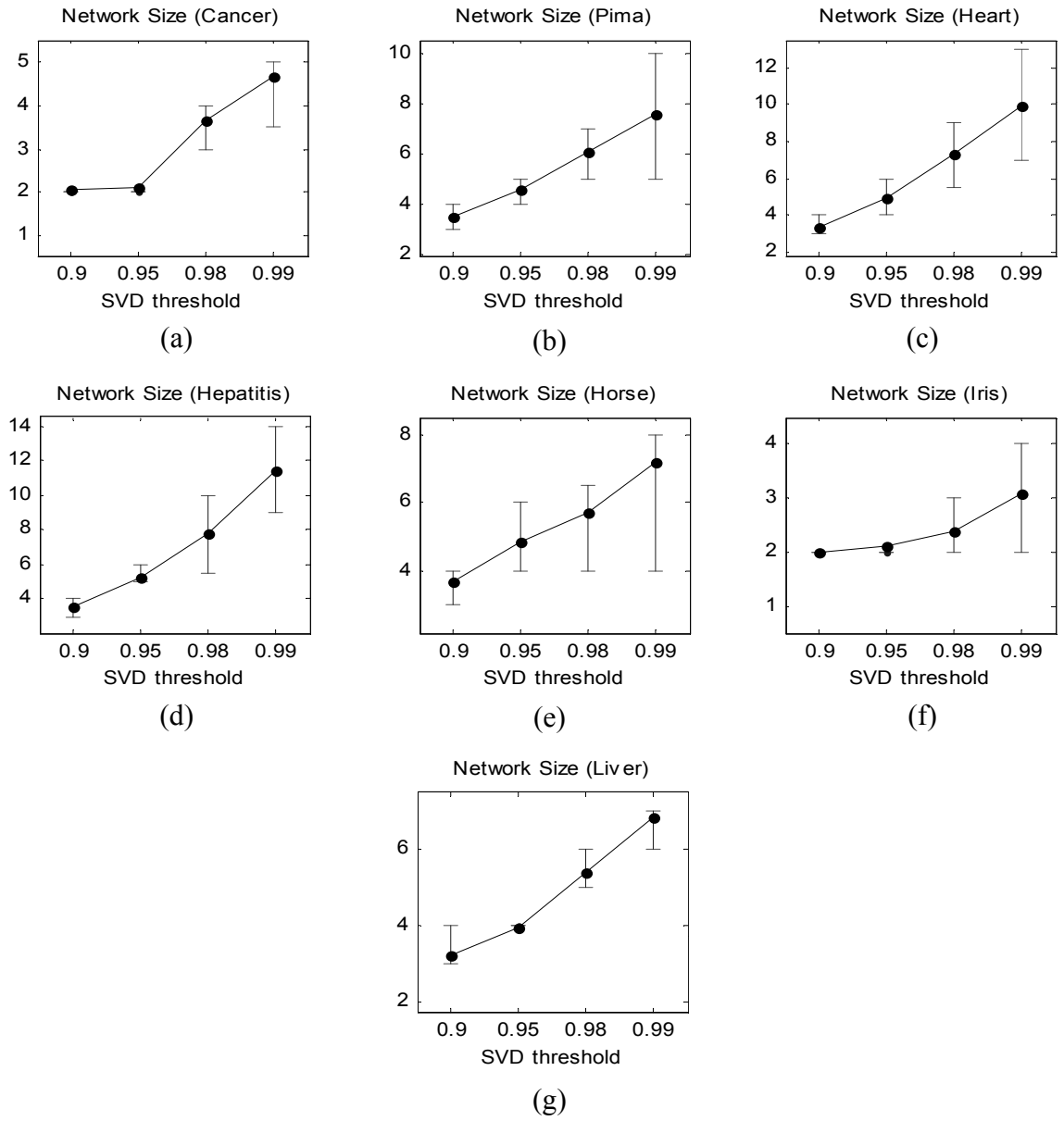

Figure 3.13: Trend of Network Size over different SVD threshold settings for (a) Cancer, (b) Pima, (c) Heart, (d) Hepatitis, (e) Horse, (f) Iris, and (g) Liver. The trend is connected through the mean while the upper and lower edges represent the upper and lower quartiles respectively.

3.5.3 Comparative Study

Having validated the effectiveness of the μHGA and SVAR, the performance of HMOEN_L2 and HMOEN_HN are compared against other works in the literature using these datasets. These works includes some well-known rule-based machine-learning algorithms as well as recent EANN approaches. The discussion is limited specifically to methods which is largely centered upon evolutionary algorithms, namely MOBNET [58] and MGNN [146]. MGNN [146] utilizes a mutation-based genetic operation in the evolutionary process in adapting the weights of the neural network to replace the traditional back-propagation algorithm during the training process - this is achieved via the mutation strategy of local adaptation of evolutionary programming (EP) to affect weight learning. The mutation process allows the network to dynamically evolve its structure and adapt its weights simultaneously. MOBNET [58] on the other hand evolves subcomponents in an independent manner, such that they are subsequently combined to form a network, unlike traditional approaches which focuses on the development of an entire structure at once. Clearly, these subcomponents when evolved separately in a cooperative co-evolution model must meet predefined criteria such that the sum of these parts is useful, even though these criteria might be in conflict with each other, which is quite commonly the case in many practical circumstances. This dilemma in evaluating the fitness on an individual based on multiple criteria that are to be optimized together is almost always approached as from a multi-criteria optimization perspective. [2]

The summary of results is shown in Table 6. Note that comparisons between the results obtained from different approaches have to be made cautiously, as there are numerous ways in which the experimental and simulation setups are done, for example, the training/testing ratio, the pre/post-processing, the cross-validation runs, etc. The results that are presented here are not fine-tuned in any manner, i.e., the same parameter and experimental settings are used for all the datasets. Nonetheless, it can be observed that the proposed approach is better or at least competitive for Pima, Hepatitis, Horse, Iris and Liver. Cancer results are outperformed by MPANN and GABE while Heart result is outperformed by GABE. On the other hand, GABE and MPANN perform poorly for Pima with respect to our proposed HMOEN_L2 and HMOEN_HN.

[2] An additional point to note is that the improvement in performance comes with additional requirements in terms of computational complexity – where compared to 'unhybridized' approaches, the proposed approach would result in longer simulation times.

Method /Reference	Data Set						
	Cancer	Pima	Heart	Hepatitis	Horse	Iris	Liver
HMOEN_L2	0.963	**0.785**	0.797	0.803	**0.9838**	**0.98**	0.68
HMOEN_HN	0.9682	0.7536	0.8106	0.7551	0.977	0.9103	**0.6894**
C4.5 [3], [47], [56]	0.947	0.7313	0.7661	0.7925	0.8504	0.94	-
CART [43], [56]	0.745	0.745	0.808	**0.827**	-	0.96	-
PART [15], [56]	0.7278	0.7278	0.7797	-	-	-	-
NB [30], [56]	0.975	0.7509	-	-	-	-	-
MLP/ RBF	0.963	0.733	0.7817	-	-	0.9453	-
MSDD	0.9515	0.7133	-	0.8077	-	-	-
SONG [26]	0.974	0.764	-	-	-	0.973	0.685
SNG[1][17]	0.9722	0.7378	0.8103	-	0.6538	-	-
MPANN [1]	0.981	0.749	-	-	-	-	-
GABE[2] [6]	**0.9883**	0.7383	**0.8858**	-	-	0.9133	-
MGNN [46]	0.9695	-	-	-	-	0.9383	-
MOBNET [18]	-	0.7785	-	-	-	-	-

Figure 3.14: 1) Results recorded from [17] are based on the performance of a single ANN (SNG) as opposed to an ensemble; 2) Results recorded from [6] are based on the performance of the ANNs using genetic algorithm with Baldwinian evolution (GABE).

Abbass [2] reported that the average network sizes of the ANN with the lowest classification error for MPANN for the Cancer and Pima datasets were 4.125 and 6.6 percent respectively. In the case of single network in [57], the mean network sizes for the datasets of Cancer, Pima, Heart, and Horse are 5.89, 7.9, 7.28, and 20.3 percent respectively. GABE [29] fixes the number of neurons in the hidden layer to be 5. Using our proposed approach, the size of the networks that were evolved are correspondingly, for the datasets of Cancer, Pima, Heart, and Horse: (1) 4.8, 8.0833, 9.6833, and 9.15 hidden layer neurons respectively when the L_2-norm is used as the second objective, and (2) 4.667, 7.517, 9.8667, and 7.2 percent respectively when the number of hidden layer neurons is used as the second objective.

3.6 Chapter Summary

In this chapter, a hybrid multi-objective evolutionary approach to artificial neural network design is proposed. To address the issue of network architectural development, the use of a simple, but robust information measure based on the singular value decomposition, is used to estimate the necessary number of neurons to be used in training a single hidden layer feedforward network. Subsequently, an SVD-based architectural recombination is presented for the purpose of facilitating the exchange

of neuronal information between candidate neural network designs and adaptation of the number of neurons for each individual, based on a geometrical approach in identifying hidden layer neurons to prune.

In addition, two other problem specific operators comprising a variable length representation and a micro-hybrid genetic algorithm with adaptive local search intensity are also proposed to handle the fundamental issues of structural adaptation and local fine-tuning. It has been shown that neural network classifiers evolved by the proposed approach provides competitive, if not better, performances over the set of datasets employed in the comparative study as compared to existing approaches. Experimental studies were also performed to examine the effectiveness of the proposed methods with respect to real-life datasets to illustrate that the both SVAR and models assume different, but nonetheless significant roles in the evolution of effective ANN designs.

While the effectiveness of our proposed approach have been demonstrated for classification problems, it is believed that the methods that employed in this chapter are sufficiently flexible and robust to be extended to handle a variety of problem domains, such as regression, prediction as well as system identification problems, all of which are possible directions for future research.

Chapter 4

Layer-By-Layer Learning and the Pseudoinverse

In this chapter, two simple, yet effective methods to learning are presented, for both feedforward and recurrent neural networks based on a 'layered' training mechanism – first, this is done for an MLP that is based on approximating the Hessian using only local information, specifically, the correlations of output activations from previous layers of hidden neurons, and second, for a recurrent MLP structure that is based on a hybrid Evolutionary Strategy (ES) and pseudoinverse approach together with an adaptive linear observer (the pseudoinverse and adaptive linear observer acting as local search operators), as a simple layered learning mechanism for general RNN applications.

For training feedforward networks, this approach of training the hidden layer weights with the Hessian approximation combined with the training of the final output layer of weights using the pseudoinverse (from the ELM paradigm [[104]) yields improved performance at a fraction of the computational and structural complexity of conventional learning algorithms. On the other hand, the training approach used for a recurrent neural network uses an Evolutionary Strategy (ES) to learn the real-valued weights of the recurrent stage of the network, and a combination of the pseudoinverse and adaptive linear observer implemented using gradient descent (both as local search operators) for learning the weights of the feedforward stage. Essentially, underlying vein of the methods used to train the feedforward and recurrent networks in this chapter is the partitioning of the networks,

whether feedforward or recurrent, into a cascade of layers or stages to which different, or specific learning algorithms can be used for each target layer.

4.1 Feedforward Neural Networks

4.1.1 Introduction

Learning, given a set of input-output examples, is essentially the computation of a mapping from an input to output space, which in turn can be cast as an optimization problem where the minimization of a suitable cost function (such as the ubiquitous sum-of-squared-loss error) is desired.

Typical learning algorithms for training feedforward neural architectures for a variety of applications such as classification, regression and forecasting utilize well-known optimization techniques. These numerical optimization methods usually exploit the use of first-order (Jacobian) and second-order (Hessian) methods. [1] The standard back-propagation algorithm for example, utilizes first-order gradient descent based methods to iteratively correct the weights of the network. Learning using second-order information such as those based on based on the Newton-Raphson framework offer faster convergence, but at the cost of increased complexity. Typically, the Jacobian or gradient of the cost function can be computed quite readily and conveniently; however, the same cannot be said of the Hessian, particularly for larger-sized networks as the number of free parameters (synaptic weights) increase. As such, second-order approaches are not popular primarily because of the additional computational complexity that is introduced in calculating the Hessian of the cost function with respect to the weights.

With this in mind, approximation of the Hessian matrix, commonly found in numerous quasi-Newton approaches for optimization, is of great interest. Although more efficient methods in computing the Hessian exactly have been suggested [25, 28], exact computation of the Hessian is costly - moreover, in many practical applications, such accurate computation is not required. For example, the Hessian is often numerically ill-conditioned or even singular for many real-life problems due to the limited discrimination ability of certain nonlinear functions, hence requiring some form

[1] Respectively, the Jacobian and the Hessian refer to the first and second derivatives of the cost function with respect to the weights.

of regularization to be included. Some of the more well-known approximation techniques include only using the diagonal entries of the Hessian, omission of certain terms (such as the second partial derivatives) [158] or through numerical differentiation [28].

While various approximations to the Hessian have been proposed, computation of the Hessian matrix based solely on local information, such as (input and output) signals available to weights of a particular layer, is often desirable, as this may allow the partitioning of the training of a multi-layered neural architecture into layers, each of which can be trained separately from each other. This avoids global computation methods (such as of the Hessian) which are inherently computationally intensive. However, in approximating the Hessian, it is important to consider those which maintains the symmetry and positive definiteness such that the one-dimensional minimization step remains unaltered [17]. On the other hand, non-gradient based approaches such as evolutionary algorithms (EAs), and the ELM paradigm [104] which trains only the output layer weights while leaving the hidden layer weights initialized randomly with a uniform probability distribution provide interesting alternative methods of training a neural network. Non-gradient based techniques are attractive particularly because they are less likely to converge to a local minima, which holds true for increasingly complex problems. However, EAs are largely stochastic optimizers and are hence slow in discovering solutions; the ELM approach is fast in obtaining solutions but at the cost of the need to use an architecturally more complex (as measured by the number of hidden layer neurons used) structures.

In [104], the Moore-Penrose generalized inverse (or more commonly known as the pseudoinverse) was used to train the output weights of the feedforward network (for either a conventional MLP or an RBF network) while leaving the hidden layer weights (initialized randomly using a uniform probability distribution) untrained. While this method is extremely fast in the training stage, relatively larger number of hidden layer neurons are needed – some form of training if utilized, to learn the weights of the hidden weights, is always able to obtain a smaller, or minimal network architecture using fewer hidden neurons. The number of iterations (or epochs) that is additionally used to train the hidden layer weights need not be exhaustive, as will be subsequently demonstrated.

Depending on the order of the model is chosen, and applying the stationarity condition, the standard versions of the steepest-gradient and Newton's approach is obtained as follows. Neglecting

terms of order two and above, the standard steepest-gradient weight update rule is as follows,

$$\Delta w = -J(w_0) \tag{4.1}$$

$$w(t+1) = w(t) - \eta(t)J(w_0) \tag{4.2}$$

One of the critical drawbacks of the steepest descent approach is its rate of convergence, which is slow and at best, linear. However, if additional information of the approximation of the cost function is included by incorporating the second-order term, and neglecting terms of order three and above to obtain a quadratic function, the standard Newton's formula for weight updating (after imposing the stationarity condition $J(w_0) + H(w_0)(w - w_0) = 0$) is obtained,

$$H(w)\Delta w = -J(w_0) \tag{4.3}$$

$$w(t+1) = w(t) - \eta(t)H(w_0)^{-1}J(w_0) \tag{4.4}$$

This solves the optimization problem in a single step if and only if the cost function is the second-order surface. Otherwise, as is usually the case, this second-order weight update procedure is iteratively applied. The direction $-H(w_0)^{-1}J(w_0)$ is known as Newton's direction, or the Newton step at $w = w_0$. Unlike the steepest gradient method, Newton's approach does not guarantee that the $E(w, t+1) < E(w, t)$ as this is dependent on the positive-definite nature of $H(w_0)$. Furthermore, the use of the Hessian in Newton's approach introduces added complexity (assuming N is the size of $H(w_0)$), in the form of storage ($O(N^2)$) and inversion ($O(N^3)$). Like most second-order optimization methods, the use of Newton's approach converges fast but only in a local sense, that is when it is used (initialized) sufficiently close to a solution. As noted by [17], the main idea when developing a general-purpose learning algorithm is to combine a fast *tactical* local method with a robust *strategic* method that is able to assure global convergence.

4.1.2 The proposed approach

Given a cost function $E(w) : \Re^n \rightarrow \Re$, such as the sum-of-squares cost function, [2] $E(w)$ is constructed as a localized model using the Taylor series approximation by constructing an expansion about a

[2] $\sum_p^N \sum_c^C (d_c(p) - y_c(p))^2$

nominal point w_0 and subsequently analyze the effect of perturbing the parameter vector (w) wbout w_0 (the local minimum). Thus, $E(w) = E(w_0) + J(w_0)(w - w_0) + \frac{1}{2}(w - w_0)^T H(w_0)(w - w_0) + O(\|w - w_o\|^3)$, where $J(w)$ and $H(w)$ are the Jacobian (gradient) and Hessian of the cost function respectively.

The essential idea that underlies the proposed approach is the layer-wise (iterative) estimation of the Hessian, which together with the gradient (Jacobian) information forms a weight update rule for hidden layer weights. This, coupled with the least-squares or pseudoinverse approach for the weights of the final (output) layer is the basis of our proposed training algorithm which offers faster convergence and better generalizability. Is is further shown through empirical simulations, for the specific instance of a feedforward network with a single hidden layer, the improvement in the convergence as well as performance of the resulting network compared to conventional training using the back-propagation algorithm. This special case of a single hidden layer uses the inverse of the correlation matrix of the training data (incorporating a regularization term) in an iterative update of the hidden layer weights (from the input to hidden layer).

Instead of manipulating the weights of the network directly, the approach does so in an indirect manner, through the modification of the desired activations of each layer, and is derived by noticing the separability of the layers of the network into linear (given by the affine form of the locally induced field) and a nonlinear (given by the activation function of the nodes) blocks. Consider the activation of a layer k at time step (iteration) t as $Y_k(t)$, where

$$Y_k(t) = X_k(t)W_k(t) \tag{4.5}$$

$$X_{k+1}(t) = [f(Y_k(t)) \quad 1] \tag{4.6}$$

The new estimate for Y_k at the next time step is $Y_k(t+1) = Y_k(t) + D_k(t)$ where $D_k(t)$ is the estimate for the direction in which the new activation for layer k should proceed. $D_k(t)$ can be derived using a variety of methods, but for simplicity, is found using the gradient descent on the cost function with respect to the output activations of the hidden layers ($Y_k(t)$), i.e. $D_k(t) = -\eta(t)\nabla_{Y_k(t)}E(w,t)$.

Thus,

$$Y_k(t+1) = Y_k(t) - \eta(t)\nabla_{Y_k(t)}E(w,t) \tag{4.7}$$

$$Y_k(t+1) = X_k(t)W_k(t+1) \tag{4.8}$$

$$X_k(t)W_k(t+1) = X_k(t)W_k(t) - \nabla_{Y_k(t)}E(w,t) \tag{4.9}$$

$$W_k(t+1) = X_k(t)^\dagger X_k(t)W_k(t) - \eta(t)X_k(t)^\dagger\nabla_{Y_k(t)}E(w,t) \tag{4.10}$$

$$W_k(t+1) = W_k(t) - \eta(t)X_k(t)^\dagger\nabla_{Y_k(t)}E(w,t) \tag{4.11}$$

Clearly, $\nabla_{W_k(t)}E(w,t) = X_k(t)^T\nabla_{Y_k(t)}E(w,t) \Rightarrow \nabla_{Y_k(t)}E(w,t)$ and thus $\nabla_{W_k(t)}E(w,t) = (X_k(t)^T)^{-1}\nabla_{W_k(t)}E(w,t)$. By virtue of the fact that $X_k(t)^\dagger = (X_k(t)^T X_k(t))^{-1}X_k(t)^T$, this then leads to

$$W_k(t+1) = W_k(t) - \eta(t)X_k(t)^\dagger\nabla_{Y_k(t)}E(w,t) \tag{4.12}$$

$$W_k(t+1) = W_k(t) - \eta(t)(X_k(t)^T X_k(t))^{-1}X_k(t)^T\nabla_{Y_k(t)}E(w,t) \tag{4.13}$$

$$W_k(t+1) = W_k(t) - \eta(t)(X_k(t)^T X_k(t) + \lambda_1 I)^{-1}\nabla_{W_k(t)}E(w,t) \tag{4.14}$$

$$W_k(t+1) = W_k(t) - \eta(t)(\Phi_k(t) + \lambda_1 I)^{-1}\nabla_{W_k(t)}E(w,t) \tag{4.15}$$

From this, $\Phi_k(t)$ represents the correlation matrix of the activation outputs of the preceding layer and λ_1 being the regularization parameter for the hidden layer weights. On the other hand, the gradient or the Jacobian of the cost function with respect to the weights in layer k, $\nabla_{W_k(t)}E(w,t)$ can be recursively computed layer-by-layer. This approach was previously suggested by [170, 149] (without involving the use of the pseudoinverse), which offers a simpler method to building an approximation of the Hessian in a second-order approach.

The use of the pseudoinverse in the layer-by-layer computation of the weights of each layer affords us the ability to introduce regularization terms to 'control' the resulting size of the weights in each layer as measured by its L_2-norm. This prevents overtraining of the network, and is similar in idea to that of 'early-stopping'. As was rigorously discussed by Bartlett, the size of the weights of the network is important for the generalization performance of the network [16]. The computational advantage obtained from the use of the proposed approach however, is dependent on the number of neurons that is used in the hidden layer(s). Another point worthy to note is that this layered estimation

of the Hessian assumes the independency of weights on different layers – such as assumption is generally acceptable in the case when the learning is close to convergence [149]. The convergence rate using only the pseudoinverse as a learning algorithm is extremely fast - however the use of the pseudoinverse in determining the weights of a layer in a multi-layer feedforward network is only optimal with respect to the weights of the other layers. This is to say that global optimality is only guaranteed for that layer. In a global context, the weights found using the pseudoinverse is locally optimal.

Based on Cover's 1965 function counting theorem [34], projecting samples or observations from a low dimensional input space to a higher dimensional feature space would make these transformed features more likely to be linearly separable. Placed in our context, a single hidden layer with a large number of neurons (with randomly initialized hidden layer weights) is instead used. To find the output layer weights, the pseudoinverse (also known as the Moore-Penrose generalized inverse)is used. Note that the output nodes need not be a linear activation function, for if $f(y) = d$, then $y = f^{-1}(d)$ where d is the desired signal, $f^{-1}(\cdot)$ is the functional inverse. The pseudoinverse is then used with y as the desired signal.

Like other numerical minimization algorithms, the method being suggested here is an iterative procedure. The gradient (Jacobian) of the cost function (sum-of-squares) with respect to the weights in the different layers can be found quite readily. The estimate of the Hessian is based on a layer-wise approximation [149] that only uses local information (in essence, using only the correlation of the activation signals from the previous layer). Clearly, this Hessian approximation is a function of the layer index (k) and the iteration step (t). The output layer, using a linear activation function is computed using the pseudoinverse.

Let k denote the layer index, and t the iteration step.

$$\hat{H}(k,t)\Delta w(k) = \eta(t)\nabla(E(\mathbf{w})) \tag{4.16}$$

$$w^k(t+1) = w^k(t) - \eta(t)\hat{H}(k,t)^{-1}\nabla(E(\mathbf{w})) \tag{4.17}$$

Appropriate settings of the learning rate, η were shown to lead to good performance, and is generally insensitive to small variations. For the simulations carried out, $\eta(0) = 1$ and is made to decrease in an exponential manner as learning progresses (i.e. $\eta(t) \propto \frac{\eta(0)}{1+\ln(t)}$).

For the final (output) layer ($k = L$), with the superscript † denoting the pseudoinverse and $\mathbf{d}$ the desired signals,

$$X_{L-1}(t)w_L(t+1) = \mathbf{d} \tag{4.18}$$

$$w_L(t+1) = X_{L-1}(t)^{\dagger}\mathbf{d} \tag{4.19}$$

$$w_L(t+1) = \left(X_{L-1}(t)^T X_{L-1}(t) + \lambda_2 I\right)^{-1} X_{L-1}(t)^T \mathbf{d} \tag{4.20}$$

$X_{L-1}(t)$ is the set of activation (output) values from the previous layer in the current time step. The system is usually overdetermined as there are more samples than hidden layer neurons in layer $L-1$. The use of this training process, in a way, allows a layer-by-layer training approach in determining the appropriate set of weights for each hidden layer – by utilizing different learning algorithms for learning the weights on different layers, the properties and characteristics of each layer can be exploited.

4.1.3 Experimental results

The datasets that were used to demonstrate the effectiveness and efficiency of the proposed approach are taken from the UCI Machine Leaning Repository. I then compare the proposed approach with the ELM paradigm [104] and a MLP-based neural network (using the NETLAB toolbox). All initial simulation parameters (such as the weights) of the network used were randomly initialized but are the same for the different approaches. The comparison of the proposed algorithm with the back-propagation algorithm (scaled conjugate gradient variant) is based on the CPU time. The ratio of the improvement of the proposed algorithm over the BP algorithm (the speed-up) is approximately 3-5 times faster – notably it is difficult to compare the speeds of the algorithms based solely on the CPU time as different implementation and coding techniques may bias the results and hence the run times. Moreover, the convergence of the algorithms are different, for example the ELM approach converges in a single step ('one-shot' learning) as no iterative correction of the weights are required since the training on the linear output layer of weights is achieved via the use of the pseudoinverse. The MLP on the other hand takes a longer time to converge while the proposed algorithm's convergence speed is between the two, depending on the problem complexity and network size. All weights were

randomly initialized to be within the range of $[-1, 1]$ (kept the same for the MLP, ELM and the proposed approach). The *tansig* function is used as the nonlinearity for the hidden layers.

Results obtained, comparing the performances of the MLP, the proposed approach (using 2 different regularization parameters) and the ELM are as in Table 4.1. 60% of available data was used as the training data while the remaining 40% was used as the testing data – these were randomly selected for each of the 50 runs. 10 hidden neurons were used for the MLP, ELM and the proposed approach.

4.1.4 Discussion

The 'size' of the weights, as measured by the norm of the weights at each layer is smaller using the proposed approach than that obtained from using the back-propagation. For the output layer weights this can be attributed to the operation of the pseudoinverse as the pseudoinverse obtains the minimum norm of possible weights that solves an overdetermined problem in a least-squares sense. For better generalization performance, clearly, fitting the network to the data of the training set is not desirable. Some form of regularization is advantageous in preventing the network from being overtrained. However, determining the appropriate amount of regularization is an art in itself. Regularization is introduced explicitly through the use of the regularization parameter λ, and through the approximation of the Hessian.

Approximations made during the weight update in the training phase is akin to regularization, since little attempt is made to fit the samples in the training data in an exact manner. In the approach presented here, the Hessian, which in the usual sense is a global computation and is expensive to obtain, is approximated on a layer-by-layer basis using the correlation matrix of the output activations from the preceding layer. Moreover, as was highlighted previously in this chapter, exact computation of the Hessian is not only computationally expensive but may also result in an ill-conditioned and even singular matrix. The regularization term that is introduced in the computation of the pseudoinverse has an effect on the classification performance of the resulting feedforward network – generally a larger regularization term (clearly, only up to a certain point) leads to poorer training performance but better testing performance. Geometrically, this is seen as the construction of increasingly smoother separating hyperplanes in hidden layer space as larger

Table 4.1: Performance comparisons – Mean accuracies and standard deviations (50 runs, 10 epochs, 10 hidden neurons)

Dataset	**MLP** training	**MLP** testing	**Fast** training ($\lambda = 0.001$)	**Fast** testing ($\lambda = 0.001$)	**Fast** training ($\lambda = 1$)	**Fast** testing ($\lambda = 1$)	**ELM** training	**ELM** testing
Diabetes	0.7777 (0.0146)	0.7517 (0.0238)	0.8184 (0.0178)	0.7403 (0.0245)	0.7945 (0.0132)	0.7629 (0.0193)	0.7690 (0.0188)	0.7530 (0.0256)
Breast cancer	0.9680 (0.0072)	0.9659 (0.0106)	0.9789 (0.0110)	0.9668 (0.0072)	0.9790 (0.0054)	0.9666 (0.0078)	0.9666 (0.0087)	0.9640 (0.0110)
Biomedical	0.9166 (0.0199)	0.8861 (0.0329)	0.9285 (0.0333)	0.8671 (0.0380)	0.9178 (0.0157)	0.8821 (0.0298)	0.8963 (0.0170)	0.8871 (0.0309)
Sonar	0.8695 (0.0409)	0.7395 (0.0492)	0.9924 (0.0091)	0.7388 (0.0392)	0.9729 (0.0126)	0.7507 (0.0358)	0.7270 (0.0451)	0.6776 (0.0626)
Ionosphere	0.8884 (0.0237)	0.8357 (0.0340)	0.9886 (0.0072)	0.8736 (0.0270)	0.9744 (0.0097)	0.8664 (0.0245)	0.8227 (0.0383)	0.7967 (0.0388)
Heart	0.8646 (0.0211)	0.8088 (0.0357)	0.9404 (0.0174)	0.7837 (0.0350)	0.8981 (0.0163)	0.8295 (0.0251)	0.8087 (0.0369)	0.7963 (0.0462)
Glass	0.9617 (0.0127)	0.9184 (0.0275)	0.9913 (0.0097)	0.8635 (0.0948)	0.9710 (0.0117)	0.9219 (0.0224)	0.9340 (0.0182)	0.9073 (0.0272)
Liver	0.7351 (0.0282)	0.6736 (0.0369)	0.7836 (0.0406)	0.6581 (0.0341)	0.7539 (0.0199)	0.7032 (0.0377)	0.7069 (0.0252)	0.6706 (0.0404)
E.Coli	0.9503 (0.0165)	0.8039 (0.1232)	0.9656 (0.0123)	0.9250 (0.0318)	0.9670 (0.0084)	0.9467 (0.0173)	0.9543 (0.0125)	0.8150 (0.1677)
Iris	0.9498 (0.0269)	0.9450 (0.0342)	0.9682 (0.0318)	0.9420 (0.0461)	0.9587 (0.0149)	0.9423 (0.0327)	0.9431 (0.0269)	0.9307 (0.0421)
IMOX	0.9214 (0.0217)	0.8695 (0.0432)	0.9674 (0.0186)	0.8976 (0.0322)	0.9552 (0.0158)	0.9126 (0.0283)	0.8864 (0.0249)	0.8453 (0.0414)
Wine	0.9498 (0.0298)	0.9373 (0.0388)	0.9687 (0.0322)	0.9320 (0.0457)	0.9598 (0.0145)	0.9417 (0.0309)	0.9491 (0.0365)	0.9263 (0.0427)

regularization is used. Clearly, the magnitude of the regularization term assists in the generalization ability of the resulting network only up to a certain point, beyond which any increase would be detrimental to both the training and testing performance.

While the ELM is fast in the training stage, from the perspective of architectural or structural complexity, it requires more hidden neurons than is typically expected for a network that is trained on all its weights. In other words, the output layer weights found from the use of the pseudoinverse in the ELM method is dependent (or 'slaved' [132]) to the randomly initialized weights of the hidden layer. The output layer weights is globally optimum, but only with respect to the hidden layer weights that was found during the training process (as is in the proposed approach), or that was which was initially randomized (as in the ELM paradigm). The use of the pseudoinverse in solving the output layer weights in a least-squares sense quickens convergence of the algorithm. As such, fewer training epochs are required for the proposed algorithm, which essentially combines the speed of the ELM and second-order gradient descent methods with the minimal network requirement of MLPs which are trained on all layers. The strength of the proposed approach lies in (1) its rate of convergence, (2) its speed resulting from its simplicity of implementation, and (3) minimal network structure requirement.

4.1.5 Section Summary

Faster convergence, or lengthier training on the training set is not beneficial if performance on the testing set is poor – for it is not desirable for the learning algorithm to converge onto the less salient features of the training set, but rather only up to a point where it would be able to provide a sufficiently high level of discrimination accuracy on the unseen observations in the testing set. The approach proposed here is less complex on each iteration (epoch) while still retaining the fast convergence of the pseudoinverse used in [104] yet using a smaller network size (as measured by the number of hidden layer neurons). The proposed algorithm, exploiting Cover's 1965 theorem [34], together with the pseudoinverse and an approximation to the Hessian, demonstrate a significant increase in the performance on both the training and testing set, in addition to a faster convergence rate. The psudoinverse is used to train the output layer weights while a second-order method using a Hessian approximation is used to train hidden layer(s) weights. Regularization terms introduced

allow fine-tuning of both the weights in all layers of weights. The approach proposed here can possibly be used for networks with multiple hidden layers, which is a direction for future exploration. As the structural complexity of the network increases [3], or if noise is known to be present in the training examples, it makes little sense to impose a 'strong' or 'strict' learning algorithm for which it will tend to overtrain the network, thus leading to a decreased ability of the resulting network to generalize on unseen data. Approximate methods incorporating regularization terms allow better and faster training techniques to be applied.

4.2 Recurrent Neural Networks

Recurrent neural networks, through their unconstrained synaptic connectivity and resulting state-dependent nonlinear dynamics, offer a greater level of computational ability when compared with regular feedforward neural network (FFNs) architectures. A necessary consequence of this increased capability is a higher degree of complexity, which in turn leads to gradient-based learning algorithms for RNNs being more likely to be trapped in local optima, thus resulting in sub-optimal solutions. This motivates the use of evolutionary computational methods which center about the use of population-based global-search techniques as an optimization scheme. In this chapter, I propose the use of a hybrid evolutionary strategy (ES) approach together with an adaptive linear observer, acting as a local search operator, as a learning mechanism for general RNN applications. Illustrative examples, though largely preliminary in nature, in solving a few system identification problems, are encouraging.

4.2.1 Introduction

Feedforward neural networks (FNNs) have been extensively used for stationary/static input-output mapping due to their powerful approximation capabilities. As [36] and [98] have shown, using real and functional analysis, multi-layer perceptrons with at least one single hidden layer possesses what is known as the Universal Approximation Capability (UAC), where the network has the ability to approximate any given function to any specified level of accuracy, provided that a sufficient number

[3] Akin to increasing the number of adjustable free parameters of the network, or equivalently, the number of hidden layer neurons.

of hidden layer neurons is used. However, the use of FNNs is limited by its inability to trace or incorporate time-dependant dynamics. RNNs are clearly a more general architecture for neural networks – RNNs can have arbitrary topologies depending on the connectivity of its synaptic weights. RNNs have often been used to model systems that exhibit time-dependent behavior, such as those found in controller systems and chemical processes, as well as to represent the cortical dynamics of the brain – in addition to many other natural systems. For over two decades, much work has focused around RNNs because of their capability in exhibiting *dynamical behavior*, having been used to model oscillators, associative memories, finite automata and control systems [23].

In this section, I present a hybrid evolutionary strategy (ES) learning approach together with a local search operator, in the form of an adaptive linear observer, in simultaneously learning the parameters (weights) and observer gains and structure of a recurrent neural network (RNN) that is applied to a system identification problem, based on a predefined structural complexity. The organization of this section is as follows: preliminaries are given in Section 2 – specifically I introduce traditional learning algorithms for RNNs, as well as describe previous efforts that have been made in the area of learning for RNNs. Section 3 subsequently overviews some previous work, general concepts as well as distinguishing features of evolutionary strategies and (adaptive linear) observer systems. Section 4 then introduces the proposed method, using a cascaded recurrent neural network structure consisting of a recurrent stage and a feedforward stage both of which are trained independently of each other, first with an ES, the second with the pseudoinverse, coupled with an adaptive linear observer. Subsequently I discuss its corresponding characteristics and properties in terms of its representation and corresponding operators. Simulations and some illustrative examples are presented in Section 5. Section 6 then provides a discussion on some of the ideas and concepts that are proposed, as well as highlighting some possible areas for future work. A summary then concludes this chapter.

4.2.2 Preliminaries

System Identification

System identification is a well-researched domain that involves, as its name suggests, the identification of a particular unknown system through either the use of model-based, or model-free structures, either the time or frequency domain. Typically, this involves the construction of mathematical models of

dynamical systems using empirical, or measured input-output data, that are found in all types of applications, from chemical plants and processes to engine and aircraft dynamics to economic and financial processes.

In many cases, first principles are not used as they may lead to overly complex structures. Instead, a more common approach is to begin from measurements of the behavior of the system and the external influences (inputs to the system) and attempt to determine a mathematical relation between them and the corresponding outputs without going into the details of what is actually happening inside the system. Two paradigms are common in the field of system identification: (a) model-based: although the peculiarities of what is going on inside the system are not entirely known, a certain model is already available. This model does however still have a number of unknown free parameters which can be estimated using system identification, and (b) model-free: No prior model is available – most system identification algorithms are of this type. The use of neural networks in system identification tasks fall under this latter category.

Evolutionary Algorithms

The most significant advantage of using an evolutionary search lies in the additional flexibility and adaptability to the task at hand, in combination with robust performance and global search characteristics [14]. In fact, evolutionary computation should be understood as a general adaptable concept for problem solving, especially well suited for solving difficult optimization problems [14] such as those for which the system is largely unknown and/or for multi-modal problems with many local optima.

The majority of current implementations of evolutionary algorithms (EAs) originate from three related but independently developed approaches: evolutionary programming (EP), evolution strategies (ES) and genetic algorithms (GAs) – all of which operate on a population of candidate solutions and rely on a set of variation operators to generate new offspring. Selection is then used to promote (occasionally in a probabilistic manner) better solutions to subsequent generations and eliminate less fit solutions. The different EAs emphasize the use of the different representations, operators as well as their corresponding interactions. For example, conventional implementations of EP and ES for continuous real-valued parameter optimization focus on the use Gaussian mutations to generate

offspring; GAs on the other hand emphasize the role of the crossover operator in exploring the search space.

It should also be noted that classical GAs on the other hand are based on three distinguishing features: the representation used *(bitstrings)*, the method of selection *(proportional selection)*, and the primary method of producing variations *(crossover)*. Of these three features, however, it is the emphasis placed on crossover which makes GAs distinctive [80]. Genetic algorithms (GAs) are stochastic search algorithms based on the mechanics of natural selection and natural genetics. GAs can be and have been used in training neural networks (for a review, see [218]). The proposed algorithm that will be described later in this chapter builds upon the synergistic interaction between GAs and ESs in evolving both the architecture as well as synaptic weights of a RNN.

4.2.3 Previous work

4.2.4 Gradient-based Learning algorithms for RNNs

Both the BPTT (backpropagation through time) and the batch version of RTRL (real-time recurrent learning) are equivalent (they perform gradient descent on the same cost function). The online version of RTRL, however, introduces some additional complexities, most notable of which is that the RTRL, being an epoch-based algorithm, resets the state of the network to the initial conditions of the trajectory periodically, such that the online version may move very far from the desired trajectory and never return. This is especially true if the network moves into a region where the neurons are saturated, because the gradients go to zero. To alleviate this problem, Williams and Zipser [211] introduced the idea of teacher forcing, where visible units of the network are clamped to the desired trajectory, preventing the actual trajectory from getting too far off course. In this version of the algorithm, a single Euler integration step of the original algorithm is taken, including the weight updates, and then enforce the clamps.

Various attempts have been made to combine EAs and learning paradigms for the optimization of the weights and/or topologies of neural networks – in particular, evolution has been used in constructing neural networks at three different levels [218]: *connection weights, architectures and learning rules.* Petridis *et. al.* [154] proposed a simple hybrid GA that evolved the low-level weights of

a RNN, where the actual real-valued weights of the RNN were converted to binary strings. The evolutionary parameters were typical of a canonical GA (at that moment of time), where roulette wheel (proportional) selection, one-point crossover and bit inversion mutation were used. The interesting approach that the authors used was the fitness-dependent adaptive probability parameters that were associated with the crossover and mutation operators. However, current theory from both genetic algorithms and connectionism suggests that GAs are not well-suited for evolving networks [11]. On the other hand, Mandischer [136] suggested the use of a high-level representation (consisting of layer size, network parameters, training parameters such as learning rates and momentum terms) of the neural network (both feedforward and recurrent structures were considered), in evolving the architecture of the network. This approach was partly motivated by the large search space of representations encoded with low-level information. Readers are encouraged to refer to [218] (and references therein), which provides an interesting and comprehensive survey on *how* evolutionary techniques have been applied to neural networks. For a basic introduction into evolutionary computation in general, the interested reader is directed to [14, 80].

Evolutionary Strategies (ES)

Evolution and adaptation are the key mechanisms of EAs, not only for finding candidate solutions for a given problem, but also the parameter values and operator probabilities of the algorithm [89]. This property of self-adaptation is the hallmark of an ES (from which the idea of an adaptive mechanism was derived from) and is intrinsically a dynamical search process that self-adapts at the level of its individuals and components (most commonly the mutation step-sizes). In this sense, the control parameters that undergo adaptation are known as the *strategy parameters* (as opposed to the *object parameters* that are directly optimized) – not unlike the idea of meta- or hyper- parameters. These strategy parameters are encoded into the chromosomes, and like object parameters, they undergo recombination and mutation – but unlike object parameters, they do not contribute directly to the fitness of the corresponding individual. However, better strategy parameters are more likely to correspond to fitter individuals and hence survive to produce offspring for propagation into subsequent generations.

While I have motivated the use of evolutionary computational methods in training a RNN by rationalizing the cause from the perspective that complex RNNs are likelier to be trapped in local

optima when a gradient-based technique is used, I will now address why ES is preferred – it is believed that an ES is one of the more appropriate evolutionary computation techniques for evolving a *direct, low-level representation* of neural network weights [218] because of the indeterminate nature of the genotype-to-phenotype mapping of neural network representations in chromosomes (possibly, even many-to-many). In other words, for a particular problem solved using a neural network, there are myriad possible combinations of network parameters (weights), as well as architectures, that will lead to a solution. This is one of the defining characteristics of neural networks – two different networks might exhibit similar qualitative behavior. While I have motivated the use of evolutionary computational methods in training a RNN by rationalizing the cause from the perspective that complex RNNs are likelier to be trapped in local optima when a gradient-based technique is used, I will now address why ES is preferred – it is believed that an ES is one of the more appropriate evolutionary computation techniques for evolving a *direct, low-level representation* of neural network weights [218] because of the indeterminate nature of the genotype-to-phenotype mapping of neural network representations in chromosomes (possibly, even many-to-many). In other words, for a particular problem solved using a neural network, there are myriad possible combinations of network parameters (weights), as well as architectures, that will lead to a solution. This is one of the defining characteristics of neural networks – two different networks might exhibit similar qualitative behavior.

There is less similarity between ES and GA (than say between, ES and EP), as GAs emphasize simulating specific mechanisms that apply to natural genetic systems whereas ES emphasizes the behavioral, rather than genetic, relationships between parents and their offspring [80]. The population size need not be kept constant and there can be a variable number of offspring per parent, much like the $(\mu + \lambda)$ methods offered in ES. In contrast to these methods, selection is often made probabilistic in GAs, giving lesser-scoring solutions some probability of surviving as parents into the next generation. In contrast to GAs, no effort is made in ES to support (some say maximize) schema processing, nor is the use of random variation constrained to emphasize specific mechanisms of genetic transfer, perhaps providing greater versatility to tackle specific problem domains that are unsuitable for genetic operators such as crossover. Crossover, for example, tends to be most effective in environments where the fitness of a member of the population is reasonably correlated with the expected ability of its representational components [66, 11]. Environments where this is not true are called *deceptive* [66]. See [80][Ch A.2] for a historical overview.

Cascaded/Layered Recurrent Neural Network

A possible method in circumventing the local minima convergence problem usually found in gradient-based approaches for learning in recurrent neural networks is to avoid the use of gradient information in updating the weights of the network. Here, the recurrent neural network is decomposed into two parts: a recurrent stage using a nonlinear activation function (for hidden neurons, or interneurons), and a feedforward stage using a linear activation function for the output nodes. While the recurrent stage accounts for the *temporal* evolution of the system dynamics, the feedforward stage is responsible for the *spatial* variation of the overall dynamics of the network [225].

Our proposed hybrid algorithm uses ES and a local search operators for training of the RNNs – based on our empirical results thus far, I find that it is less likely that the network will be trapped in local minima using this algorithm. However, performance tends to converge slowly as the networks evolve. This is because the network has only one RNN layer, so all the training efforts are concentrated on that layer, which is exactly opposite to the situation of the cascaded DNNs structure, whereas all the training occur in feedforward layer. Therefore, it is natural to consider integrating the two methods so that the total learning effort is shared between the feed forward layer and the recurrent layer, while the advantages of these two methods can be extracted to improve the performance to a higher level.

Local Search Operators

The use of local search operators is aimed to complement the global optimization ability of evolutionary algorithms, and is biased towards the discovery of solutions which are better, or as good as other solutions in a certain neighborhood - a local optimum. In a conventional sense, this entails the generation of an initial solution, subsequently making small, controlled changes to this solution over a length of time to continuously improve the solution until a point for which no further improvement is possible.

Typically, in training the weights of a neural network, an evolutionary algorithm functions as a primary search operator, over which the well-known backpropagation using a gradient descent algorithm acts as the local search operator. In effect, the weights found using the EA is used as the initial, or starting points of the training using backpropagation. In this chapter, the local search

operators that are implemented as a result of the use of two operators: (1) a pseudoinverse method to train the output layer weights (since the activation function for the output neuron is linear, the optimal set of weights with respect to the hidden layer weights in a least-squares sense can be found using the pseudoinverse), (2) an adaptive linear observer system, which is trained using a simple first-order gradient based method.

Least-Squares Approach Training the recurrent layer of weights in the RNN can be quite complex because of the presence of lateral and feedback connections. It is also difficult to relate the error to a set of weights of a certain neuron (the classical credit assignment problem). In training RNNs, it is also important to see how the weight changes affect the stability of the NN. In [104], the Moore-Penrose generalized inverse (or more commonly known as the pseudoinverse) was used to train the output weights of the feedforward network (for either a conventional MLP or an RBF network) while leaving the hidden layer weights (initialized randomly using a uniform probability distribution) untrained. While this method is extremely fast in the training stage, relatively larger number of hidden layer neurons are needed – some form of training if utilized, to learn the weights of the hidden weights, is always able to obtain a smaller, or minimal network architecture using fewer hidden neurons. The number of iterations (or epochs) that is additionally used to train the hidden layer weights need not be many, as will be subsequently demonstrated.

Adaptive Linear Observer

Adaptive observers are schemes that simultaneously estimate the plant state variables and parameters by processing the plant input and output measurements in an on-line manner. Clearly, there is a growing need in many industries for handling complex systems, and the capacity of the neural networks (NN) for approximating functions and dynamical systems, has motivated the neural network-based identification and control approach. Mainly because this allows the identification and control of highly uncertain dynamical systems that can present unknown nonlinearities, unmodeled dynamics and disturbances. Essentially, this approach allows the replacement of parts of the unknown plant dynamics by neural network (plus small approximation error terms), that have known structures but unknown weights, hence the parameterization problem is simplified and the adaptive observer design is reduced to the choice of (appropriate robust) adaptation laws for the weights. It is assumed that the states of the system are inaccessible and hence unknown and thus would require the construction of an observer, which then attempts to estimate these unknown states using only

input and output information over a certain time interval. A (linear) system to be controlled is considered to be of the form

$$x(k+1) = Ax(k) + Bu(k) \tag{4.21}$$

$$y(k) = Cx(k) \tag{4.22}$$

A more general system, possibly nonlinear can be represented as

$$x(k+1) = f(x(k), u(k)) \tag{4.23}$$

$$y(k) = g(x(k), u(k)) \tag{4.24}$$

In a simple case, a stochastic gradient descent algorithm is used to learn the 'correct' value of the observer gain in an adaptive manner, for a linear system. The error function is the difference between the actual output and the previously estimated value. If the weights are updated in an online manner, then the error function should be weighted such that more recent error terms contribute more significantly to the weight adaptation process. State estimation drew much attention from the control and signal processing community following Kalman's seminal work [112]. Modern approaches on state estimation are based on Luenberger's design [133]. The classical Luenberger observer deals with linear systems. The dynamics of a nonlinear system can be cast into a form that is identical to the dynamics of a recurrent neural network. First consider an estimated linear system of the form,

$$\hat{x}(k+1) = A\hat{x}(k) + Bu(k) + L(y(k) - \hat{y(k)}) \tag{4.25}$$

$$\hat{y}(k) = C\hat{x}(k) + Du(k) \tag{4.26}$$

Rearranging, we obtain

$$\hat{x}(k+1) = (A - LC)\hat{x}(k) + (B - LD)u(k) + Ly(k) \tag{4.27}$$

$$\hat{x}(k+1) = (A - LC)\hat{x}(k) + \begin{bmatrix} B - LD & 0 \\ 0 & L \end{bmatrix} \begin{bmatrix} u(k) \\ y(k) \end{bmatrix} \tag{4.28}$$

This can be compared to an equivalent linear recurrent neural network of the form $z(k+1) =$

$W_1 z(k) + W_2 b(k)$. Let $W_1 = A - LC$ and $W_2 = \begin{bmatrix} B - LD & 0 \\ 0 & L \end{bmatrix}$ as well as $b(k) = \begin{bmatrix} u(k) \\ y(k) \end{bmatrix}$. If L, the observer gain, is adaptive (as is later shown), stability of the resulting network can be verified by treating the system as that of one containing time-varying parameters, of the form

$$\hat{x}(k+1) = W_1(k)\hat{x}(k) + W_2(k)z(k) \tag{4.29}$$

$$z(k) = [u(k) \;\; y(k)]^T \tag{4.30}$$

$$\hat{y}(k) = C\hat{x}(k) + Du(k) \tag{4.31}$$

As can be seen, the above can be used to map a conventional linear (or even a nonlinear) system into that of a RNN, such is the similarity of the structure(s). Likewise, a nonlinear system, can be approximated with a linear adaptive observer, of the form,

$$x(k+1) = f(x(k), u(k)) + L(y(k) - \hat{y(k)}) \tag{4.32}$$

$$y(k) = g(x(k), u(k)) \tag{4.33}$$

Clearly, we could choose instead to evolve $W_A(k) \simeq A$, $W_B(k) \simeq B$ and $W_C(k) \simeq C$ directly as separate components of the overall system. This is suitable for problems where there is an approximate knowledge of the system to be identified, for instance, its system order – and, as will be shown later, an ES is used to evolve W_A and W_B for a second-order system, the local search operator (pseudoinverse) to compute W_C and then another local search operator, in the form of an adaptive linear observer to find the observer gains $L(k)$. For more complex problems, for example, systems where the order is unknown, and/or the system is nonlinear, a neural network representation is used, with a sigmoidal (*tanh*) nonlinearity. Essentially, this means that the state update of the RNN now becomes, with $f(\cdot) = \tanh(\cdot)$: $x_{rnn}k+1) = f(x_{rnn}(k), u(k)) + L(k)(y(k) - \hat{y_{rnn}}(k))$. x_{rnn} and $\hat{y_{rnn}}$ are the states and output of the RNN, respectively. The output, $\hat{y_{rnn}} = W_{output} f(x_{rnn}(k), u(k))$ is then computed as a linear sum of the states. W_{output} and $L(k)$ can then be found using the pseudoinverse and a simple first-order gradient descent algorithms respectively, both of which acts as local search operators.

Table 4.2: Notations, symbols and abbreviations

Symbol	Type	Remarks
N	$\mathbb{N}$	Number of neurons
k	$\mathbb{N}$	Number of output neurons
$N-k$	$\mathbb{N}$	Number of hidden neurons
$\mathbf{w}$	$\mathbb{R}^{\mathbb{N}\times\mathbb{N}}$	Network weights
σ	$\mathbb{R}_{+}^{\mathbb{N}}$	Mutation step-size/variance
L_{ES}	$\mathbb{N}$	Length of genome encoding strategy parameters
L_{GA}	$\mathbb{N}$	Length of genome encoding masking bits (active/inactive neurons)

4.2.5 Proposed Approach

In the following subsections, I describe the salient features of the approach and the corresponding algorithm, particularly the representation of individuals, the operators (mutation, crossover-recombination, selection) and the fitness measure. As in almost all EAs, we aim for a balance, or compromise between exploration (local/global diversification of the search space) and exploitation (of the best individual(s)) to prevent the algorithm from degenerating into a slow, exhaustive random search at one end of the spectrum, or a search process that prematurely converges at the other end of the spectrum. In describing our proposed approach, I will restrict our discussion to the training procedure for RNNs with hidden units, that is, units which have no particular desired behavior and are not directly involved in the input or output of the network. Biologically, they can be thought of as *interneurons* that perform such form of intermediary computations. Hidden units make it possible for networks to discover and exploit regularities of the task at hand, such as symmetries or replicated structure [173]. Let N be the number of neurons present in the system. For purpose of clarity and uniformity, I first establish the notations that will be used throughout this section in describing the proposed algorithm.

Representation

In attempting to find a suitable representation and structure to represent the *object parameters* and *strategy parameters* of the RNN, it is necessary and perhaps more elegant to use the natural encoding of the RNN. This is to say, rather briefly, that the low-level encoding of the weights of the RNN should be continuously real-valued, as is the mutation step-size (this suggests the use of an ES-based

approach).The first genome consists of the exogenous object parameters – the RNN weights (and biases), represented in the form of a real-valued (floating-point) matrix $\mathbf{W} \in \mathbb{R}^{\mathbb{N}\times(\mathbb{N}+\not{\mathbb{K}})}$ with the number of entries being equal to the number of weights in the RNN and is in turn equal to the number of genes in the genome. The $(N+1)$th column accounts for the bias (afferent inputs). w_{ij} represents the weights leading from the jth neuron to the ith neuron (corresponding to the matrix cell in the ith row and jth column) – the weights of the network are not necessarily symmetrical, i.e. $w_{ji} \neq w_{ij}$. Nevertheless, it is obvious that one of the critical shortcomings of using a direct low-level encoding as a representation is the lack of scalability when extending the algorithm to larger-size problem, since the size of the chromosome scales quadratically ($O(N^2)$) with the size of the network (N). This problem is highlighted in [218] (Section III. A). A high-level representation of a neural network might alleviate this problem, at the expense of a realistic representation of the evolution and learning process.

The second genome encodes the endogenous strategy parameters, namely the mutation variance of the $(N+1)$ neurons, that is $\Sigma = \{\sigma_1, \ldots, \sigma_{N+1}\}$. Again, the additional dimensionality accounts for the biases (afferent inputs). The jth value of Σ, i.e. σ_j corresponds to the jth column of $\mathbf{W}$ – specifically, σ_j is responsible for the mutation variance of the set of weights leading from neuron j to neuron i. This is biologically plausible, and is particularly attractive because the set of weights leading from a particular neuron to all other neurons (including, possibly itself) should correspond, or at least be proportional to its output activation. Hence the mutation variance for the set of output weights leading from any given neuron should be the same.

Local Search Operators

The global-local search algorithm proposed here is a result of the structure of the proposed neural network model, which consists of a cascaded recurrent and feedforward layer as well as a separate observer system. While the hidden layer weights are optimized via a global optimization process (ES), the feedforward weights and observer gains are trained using a pseudoinverse and a first-order gradient descent algorithm, respectively. The local search operators are not updated at every generation, as is done with the training of the network weights using the ES, but rather only at *every multiple of a predefined number of generations*. In out simulations, 20 generations are used in between the application of the local search operators. I believe that the use of global-local search

algorithms combine the best of both broad-based searching abilities as well as a focused, fine-tuning capability, further described below.

Local search operator: Least-squares (pseudoinverse)

For the final (output) layer ($k = L$), with the superscript $\dagger$ denoting the pseudoinverse and $\mathbf{d}$ the desired signals,

$$X_{L-1}(t)w_L(t+1) = \mathbf{d} \tag{4.34}$$

$$w_L(t+1) = X_{L-1}(t)^{\dagger}\mathbf{d} \tag{4.35}$$

$$w_L(t+1) = \left(X_{L-1}(t)^T X_{L-1}(t) + \lambda_2 I\right)^{-1} X_{L-1}(t)^T \mathbf{d} \tag{4.36}$$

$X_{L-1}(t)$ is the set of activation (output) values from the previous (hidden) layer in the current time step. The system is usually overdetermined as there are more samples than hidden layer neurons in layer $L-1$. The use of this training process, in a way, allows a layer-by-layer training approach in determining the appropriate set of weights for each hidden layer – by utilizing different learning algorithms for learning the weights on different layers, we are able to exploit the properties and characteristics of each layer. Furthermore, the use of the pseudoinverse for computing the feedforward weights in a layer-wise computation of the weights of the network affords us the ability to introduce regularization terms (represented by λ) to 'control' the resulting size of the weights as measured by its L_2-norm. This prevents overtraining of the network, and is similar in idea to that of 'early-stopping'. As was rigorously discussed by Bartlett, the size of the weights of the network is important for the generalization performance of the network [16].

An interesting property of the pseudoinverse is that it can be computed in an online manner based on the availability of newer data, as noted by Poggio and Girosi [156] and originally by Albert [9]. This method can be contrasted with the 'batch' update mode that uses all the available samples in the training set to find the pseudoinverse. The batch versus online approach is similar in principle to that of gradient-descent back-propagation technique used in neural networks, and we can justifiably extend the arguments on the merits and drawbacks of both the batch and online method to the pseudoinverse case. This recursive method for calculating the pseudoinverse, as noted by Poggio and Girosi [156] and originally by Albert [9], is beneficial in applications where more recent data would assist in better characterizing the problem, cumulatively. For example, with the current

knowledge of the first $n-1$ data points, and supposing the nth input-output pair $\{h_n, t_n\}$ arrives ($h_n = h(\langle x_n \cdot \hat{W_{n-1}} \rangle)$), the value of H_n can be recursively computed using,

$$T = \beta H \tag{4.37}$$

$$\beta_n = \beta_{n-1} + \big(t_n - \beta_{n-1} h_n\big) \frac{h_n^T (H_{n-1} H_{n-1}^T)^{-1}}{1 + h_n^T (H_{n-1} H_{n-1}^T) h_n} \tag{4.38}$$

While batch updates is shown to give a better estimate of the error surface, perform more reliably in the presence of noise, as well as allow more sophisticated variants to be applied, the increased computational cost can be quite demanding. On the other hand, a sequential update rule is believed to allow the network training to escape local minima in the presence of noise due to the increased randomness inherent during the training phase [199]; for the pseudoinverse approach, the sequential approach comes at the expense of additional time required for training.

Local search operator: Adaptive linear observer

For an observable pair (A, C), if there exists a gain matrix L such that $A_o = A - LC$ is stable with positive definite matrices P and Q verifying the discrete-time Lyapunov equation $A_o^T P A_o - P = -Q$, then the dynamics of the recurrent network of the form

$$\hat{x}_{nn}(k+1) = A\hat{x}_{nn}(k) + Bu(k) + D\sigma(\hat{x}_{nn}(k)) \tag{4.39}$$

$$\hat{y}_{nn}(k) = C\hat{x}_{nn}(k) \tag{4.40}$$

the nonlinear observer is asymptotically stable, and the hidden layer dynamics is state convergent, i.e. $lim_{k\to\infty} x_{nn}(k) - \hat{x}_{nn}(k) = 0$, if and only if

$$D\tau \leq -A_o = \sqrt{A_o^2 + \frac{\lambda_{min}(Q)}{\lambda_{min}(P)}} \tag{4.41}$$

with τ being the Lipschitz constant where $\|f(x_1) - f(x_2)\| \leq \tau \|x_1 - x_2\|$, $f(\cdot)$ is a Lipschitz nonlinearity. This method, however, does not provide any direction for designing a stable observer and is merely used as an procedure to verify the stability of the resulting observer. A standard approach to obtain a stable matrix L would be the use of a linear matrix inequality (LMI) technique.

Consider the nonlinear system in (4.23) and (4.24). Subsequently, we derive a simple gradient-

based approach (based on a least-means squares method) in computing the adaptive observer gain,

$$J(k) = (y(k) - \hat{y}(k))^T (y(k) - \hat{y}(k)) \tag{4.42}$$

$$\frac{\partial J(k)}{\partial L(k)} = -2(y(k) - \hat{y}(k))^T h_x(\hat{x}(k), u(k)) \frac{\partial \hat{x}(k)}{\partial L} \tag{4.43}$$

$$\frac{\partial \hat{x}(k)}{\partial L} = \left[f_x(\hat{x}(k-1), u(k-1) \quad - L(k) h_x(\hat{x}(k-1), u(k-1)) \right] \frac{\partial \hat{x}(k-1)}{\partial L} + (y(k-1) - \hat{y}(k-1)) \tag{4.44}$$

where $J(k)$ is the instantaneous error at time index k, $f_x(\cdot)$ and $g_x(\cdot)$ representing the Jacobian matrices of the nonlinearities in (4.23) and (4.24) respectively. Subsequently, the iterative update to the observer gains are simple, in a first-order manner,

$$\mathbf{L}(k) = \mathbf{L}(k-1) - \eta \frac{\partial J(k)}{\partial L(k)} \tag{4.45}$$

η is the learning rate, and is usually chosen to be a small value (≈ 0.001) to prevent divergence of the state dynamics. Here, the recurrent network is regarded as a dynamical operator implementing a mapping (possibly nonlinear) between input and output pairs of functions. I now treat the overall network as a system with time-varying parameters, for as L is continuously adapted (at least, until convergence), then both W_1 and W_2 are continuously being modified. Stability can then be investigated by, as I have suggested, assuming this resulting network to be a system with time-varying parameters (with W_1 and W_2 being the time-varying parameter) where the input-output stability can be studied. While a more direct method of investigating the (input-output) stability is the use of Lyapunov methods to find a globally asymptotic stable equilibrium in state space, these assumptions may not hold particularly when the parameters of the network (weights) are time-varying. This will be investigated subsequently as a future research direction.

Standard Operators

Fitness function The objective of the approach used here is to *minimize* the network error function which is defined as the mean squared error (MSE) between the actual and desired system outputs, over a time period bounded by $[0, T_{traj}]$. t_{traj} denotes the time steps of the simulation. The use of

the subscript *traj* is to avoid confusion with the *generation index* t and the *maximum generation number* T that will be used in subsequent sections. Since (as will be described later) a proportional selection strategy is not used, the requirement that the fitness values be non-negative is not necessary. I then aim to *maximize* the negated MSE value, i.e. minimizing E is equivalent to maximizing ($-E$ or $1/E$). This is to say that the fitness F_i of each individual i in the population is defined as the negative of the error between the desired and the actual trajectory learnt, i.e. $F_i = -E_i$ This error is in turn defined as, $E_i(T_{traj}) = \sum_{t_{traj}=1}^{T_{traj}} (\mathbf{d}(t_{traj}) - \mathbf{y}^i(t_{traj}))^2$.

Mutation The core of the evolutionary process used in ES centers about the mutation operator, which perturbs the original value by an amount determined by (i) the type of mutation probability distribution, and (ii) the mutation step-size (variance). The step-size of the mutation undertaken is based on the observation that smaller changes in a genotype is more likely than a large change – this naturally suggests the use of 'bell-shaped' probability distribution from which the mutation variance is sampled from. With this in mind, the most popular and widely-known probability distribution which meets this criteria is the Gaussian, or Normal distribution $f_t(x) = \frac{1}{\sqrt{2\pi}} \exp \frac{-(x-\mu)^2}{2\sigma^2}, \quad -\infty < x < \infty$; an alternative is the Cauchy distribution $f_t(x) = \frac{1}{\pi} \frac{t}{t^2+x^2}, \quad -\infty < x < \infty$, which essentially is similar to the Gaussian distribution except that it has a smaller peak and a much longer tail. Recently, Cauchy mutations have been proposed for use with EP and ES [215, 216] inspired by fast simulated annealing [183]. The lognormal self-adaptation scheme [172, 168] was extended for evolving scale parameters for these Cauchy mutations. Qualitatively, the fatter tails of the Cauchy distribution generate a greater probability of taking large steps, which could help in escaping local optima, and this has been offered as a possible explanation for its enhanced performance [172], particularly for multi-modal problems.

Recombination The primary drawback of crossover and recombination is its destructive effect, which is detrimental to the real-valued encoding used for representing the weights and mutation step-sizes of the resulting RNN architecture. In order to mitigate this (destructive) effect yet retain some form of variation, a discrete recombination through the sharing of genotypic material from ρ ($\leq \mu$) parents is applied – unlike conventional GAs where a two parents produce two offspring, the standard ES recombination operator to a parent family of size ρ produces only one offspring. While there are two possible types of recombination used in ES (discrete (dominant) recombination, and continuous (intermediate) recombination), I focus on the former, where the kth feature of the

child chromosome is determined exclusively by the kth feature of a randomly (in a uniform manner) chosen parent from the (mating) parent population of size ρ [4].

Selection While selection methods in GAs typically use a stochastic selection operator (proportional, rank-based and tournament being the more popular), ES uses a deterministic selection procedure (usually an elitist $(\mu+\lambda)$ or a non-elitist (μ, λ) selection strategy). The degree of selection pressure imposed in the population decreases the diversity of the population rapidly, often resulting in premature convergence (stasis) – this necessitate the use of highly disruptive search/variation (recombination and mutation) operators; alternatively, a selection scheme with high selection pressure is associated with high recombination and mutation probabilities. Tournament selection [67] is a well-known and popular ranking method of performing selection [43]. A binary tournament in turn implies that only two individuals compete directly for selection, and this can be done with, or without reinsertion of competing individuals into the population. In our proposed selection strategy, a probabilistic approach is implemented by decreasing the probability that a less fit individual is chosen over a more fit individual as the search process progresses, i.e.

$$P_{tour} = p_o \frac{c}{1 + \ln(t)} \in (0, 1) \tag{4.46}$$

where $c > 0$ $(= 1)$ and p_o $(= 0.25)$ denoting a positive constant and initial probability (values within brackets indicate simulation values), respectively. This attempts to maintain population diversity for the initial stages of the search process while on the other hand, increasing the probability of retaining the best solution as the search advances.

As was previously highlighted, the complexity of the RNN dynamics usually leads to the increased likelihood that many local minima may exist on the error surface. With this in mind, the approach avoids using a deterministic as well as an elitist strategy, preferring to focus instead on a probabilistic non-elitist selection strategy. To be more specific, a probabilistic binary tournament selection (without reinsertion) is used on a (μ, λ) strategy. This naturally allows the selection strategy to accept temporary deteriorations that may assist the algorithm in leaving attraction regions of local minima and ultimately arrive at a better optimum (the progression of the fitness function is no longer monotonic as would have been the case if a deterministic elitist strategy had been adopted).

[4] The alternative continuous, or intermediate recombination creates the kth feature of a child as the weighted average of ρ parents. This is discussed with more detail in [22, 43].

Table 4.3: Evolutionary parameters

Parameter	Value	Remarks
μ	10	Parent size
λ	7μ	Offspring size
τ	$\frac{1}{L_{ES}} = \frac{1}{N}$	Mutation learning rate
ρ	$\frac{\mu}{2}$	Discrete recombination parent pool size
L_{ES}	$\mathbb{N}$	Length of genome
P_{tour}	$\frac{0.25}{1+\ln(t)}$	Probabilistic binary tournament
T	$\in [40, 200]$	Number of generations per time step

Evolutionary parameters

A deterministic non-elitist (μ, λ) selection strategy (as opposed to an elitist $(\mu+\lambda)$ strategy) is used. This naturally allows the selection strategy to accept temporary deteriorations that may assist the algorithm in leaving attraction regions of local minima and ultimately arrive at a better optimum (the progression of the fitness function is no longer monotonic as would have been the case if a deterministic elitist strategy had been adopted). This has been suggested by Beyer [22] particularly for real-valued object parameters. While the possibility of the loss of good individuals is present, non-elitist selection strategies is able to avoid local optima more readily than elitist selection strategies. Avoiding being trapped in local minima is crucial for RNN learning. In our simulations, $\mu = 15$ and $\lambda = 7\mu$ with the number of generations in the range of $[40, 200]$, increasing as time progresses (since the neural network model needs to be fitted to a larger number of input, or sample observations). Table 4.3 summarizes the evolutionary parameters used in the simulation.

General algorithm

Figure 4.1 illustrated the flow of the proposed ES-local search approach.

1. *Initialization*: Generate an initial population of μ individuals, and set the generation number t to one. Each individual is taken as a pair of real-valued vectors, $(\mathbf{w}(k), \sigma(k)),\ \forall k \in \{1, 2, \ldots, \mu\}$. $\mathbf{w}(k)$ give the kth member's object variables (network weights) and$\sigma(k)$ the associated strategy parameters (mutation step size). The object variables typically are the real parameters to be optimized and the strategy parameters represent standard deviations of the associated Gaussian mutations.

2. *Produce offspring*: For each parent in the parental pool (μ), spawn $\lambda = 7\mu$ offspring. Each child

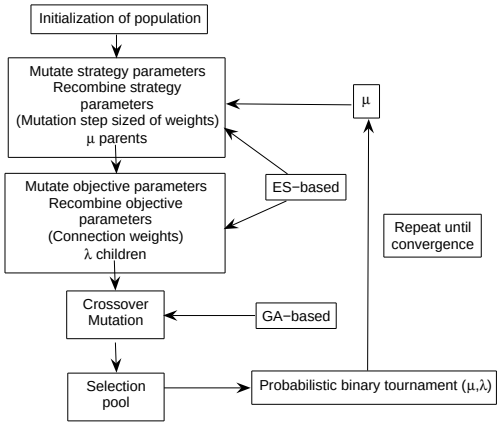

Figure 4.1: Flow-diagram of the proposed ES-local search approach

takes on the genotypical attributes of the parents, but is mutated and recombined, as described in Steps 3 and 4.

3. *Mutate and recombine strategy parameters*: The mutation operator used here sampled from a Gaussian probability distribution. Although a Cauchy probability distribution was attempted, the results were not very encouraging perhaps due to the fact that the weights often grow without bounds due to the higher probability that larger mutation step-sizes are undertaken. Subsequently, a discrete recombination operator is then applied, where the parent pool (of size $\frac{\mu}{2}$ is randomly chosen.

4. *Mutate and recombine object parameters*: The object parameters are subsequently mutated and recombined, using the values found from Step 2. Mutation and recombination of strategy parameters is carried out before that of the objective parameters, for if the converse was done, 'bad' strategy parameters (and correspondingly 'bad' object parameters) would be preserved while individuals having 'good' object parameters are eliminated.

5. *Create mating pool*: The mating pool for which the subsequent generation is created, from the λ offspring (the μ parents are removes since the use of a deterministic elitist strategy is avoided).

6. *Selection*: A probabilistic binary tournament selection without reinsertion (no replacement) is then carried out on the mating pool. An alternative approach would be to rank these individuals and select the top μ individuals for propagation into the next generation. However, it was found that a probabilistic approach demonstrated improved results over a deterministic method, as it is likelier that local minima are overcome.

7. *Convergence criteria*: Repeats Steps 2-7 until some convergence criteria is met. The convergence criteria can be specified based either on some resource (fitness, diversity, progress ratio, etc.) or computational aspect (maximum time, maximum number of generations reached). In our simulations, a maximum generation number of between 50 to 200 per time step (increasing with time as more observations are used in the training data) is used.

4.2.6 Simulation results

All prediction or tracking of the output of the unknown system is done in an online manner, where new data is continuously being used, to iteratively update the parameters (network weights, biases and observer gains) of the neural network model.

The first simulation was carried out on a second-order linear system of the following form:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}u(k) \tag{4.47}$$

$$y(k) = \mathbf{C}\mathbf{x}(k) \tag{4.48}$$

where $\mathbf{A} = \begin{bmatrix} -0.6539 & 0.8938 \\ -0.4248 & -0.1347 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$ and $\mathbf{C} = [0.5 \;\; 0.3]$. With the implementation of the adaptive linear observer, the system structure now becomes,

$$\hat{\mathbf{x}}(k+1) = \mathbf{A}\hat{\mathbf{x}}(k) + \mathbf{B}u(k) + \mathbf{L}(k)e(k) \tag{4.49}$$

$$\hat{y}(k) = \mathbf{C}\hat{\mathbf{x}}(k) \tag{4.50}$$

where the error term $e(k)$, is $e(k) = y(k-1) - \hat{y}(k-1)$. $y(k)$ is the actual observed output, $\hat{y}(k)$ is the estimated, or predicted output, and $u(k)$ is the input. $\mathbf{L}(k)$ is the adaptive observer gains, which is modified in an iterative manner using a local search operator in the form of a simple first-order gradient descent. The input is defined as $u(t) = 0.8[\sin(2t)^2 - 2\cos(3t)] + 0.1U(0,1)$, with $U(0,1)$ being a standard uniform probability distribution. See Figure 4.2.

The second system that was simulated had the following system properties (matrices) $\mathbf{A} = \begin{bmatrix} 0.0386 & 0.9084 \\ -0.6514 & -0.9531 \end{bmatrix}$, $\mathbf{B} = \begin{bmatrix} 0.2 \\ 0.1 \end{bmatrix}$ and $\mathbf{C} = [0.2 \;\; 0.2]$. The resulting performance is shown in Figure 4.3.

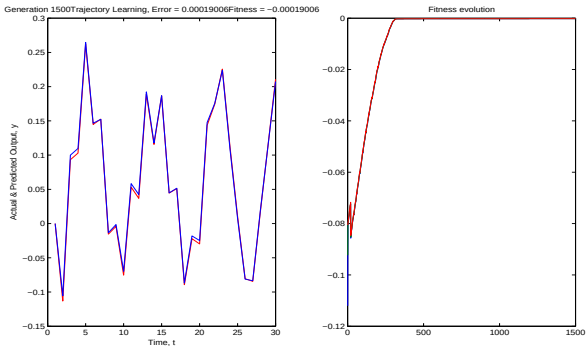

Figure 4.2: *Left:* Actual and predicted output (SSE $= 1.9 \times 10^{-4}$); *Right:* Fitness evolution

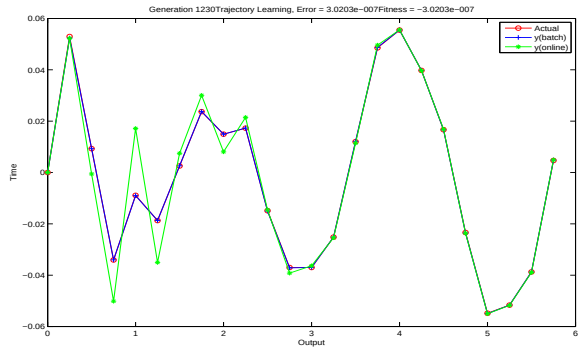

Figure 4.3: Actual and predicted (both online, and batch) output – by 'online' it is meant that the next step output is based solely on the last previous states, output and present input while in 'batch' mode, at the end of the simulation, the system is simulated again using the found weights, biases and observer gains during the training process. (SSE $= 3.02 \times 10^{-7}$)

4.2.7 Discussion

Self-adaptive update schemes, in conjunction with the selection process, favor strategy parameters that yield a higher probability of improving the quality of the parent. This is the crux of the proposed ES-local search approach. ES (like EP), differs philosophically from other evolutionary computational techniques such as GAs in a crucial manner, in that it is a top-down versus bottom-up approach to optimization. It is important to note that (according to neo-Darwinism theories) selection operates only on the phenotypic expressions of a genotype; the underlying coding of the phenotype is only affected indirectly. The realization that *a sum of optimal parts rarely leads to an optimal overall solution* is key to this philosophical difference. GAs rely on the identification,

combination, and survival of 'good' building blocks (schemata) iteratively combining to form larger 'better' building blocks [80]. Generally, a less complex structure would require a more sophisticated learning algorithm, while a more complex structure might require a less complex learning algorithm since there would be a greater degree of freedom inherent in the network structure.

In a canonical GA, the coding structure (genotype) is of primary importance as it contains the set of optimal building blocks discovered through successive iterations. The building block hypothesis is an implicit assumption that the fitness is a separable function of the parts of the genome. This successively iterated local optimization process is different from ES, which is an entirely global approach to optimization. Individuals in an ES are judged solely on their fitness with respect to the given environment. *No attempt is made to partition credit to individual components of the solutions.* This observation is imperative in training an RNN because of the indeterminate nature of the genotype-to-phenotype mapping of neural network representations in chromosomes (possibly, even many-to-many) – what constitutes a building block for a neural network representation? From empirical observations, it would be a hidden neuron together with its associated weights. In an ES, the variation operator allows for simultaneous modification of all variables at the same time. Fitness, described in terms of the behavior of each population member, is evaluated directly, and is the sole basis for survival of an individual in the population [5]. Thus, a crossover operation designed to recombine building blocks is not utilized in the general forms of ES, where in our proposed approach, a discrete recombination operator was instead applied between individuals with similar degrees of connectivity.

Despite some fairly elegant algorithms like BPTT and RTRL, most are unable to solve more complex trajectories without having to increase the structural complexity (i.e. number of hidden neurons) of the network. Essentially, a balance needs to be maintained between a *network structure* and its corresponding *training algorithm.* For less complex problems, where in this case, simple trajectories like the circle limit cycle demonstrate that gradient-descent methods may perform better that an EA approach. Nevertheless, it is also difficult to compare, directly, the performance of the proposed method against conventional gradient-descent methods – while the proposed approach 'works' on a population of individuals in a parallel manner, conventional algorithms operate sequentially; moreover, these gradient-based techniques are essentially localized operators unlike EAs which

[5] This is described in more detail in [80][Ch. B.1].

are more global in character.

Apart from that, these gradient-based learning methods are computationally expensive for large networks, componding the fact that the search space seems to have a number of spurious local minima – there are also very few results describing the stability/robustness of the trained trajectories. What this means is that the resulting set of network parameters found by the algorithms (i.e. weights and biases) are sensitive to perturbations of their starting positions – once the network has been trained, and the initial state $x(0)$ is subsequently perturbed by a small amount, the resulting trajectory will not be able to track the desired trajectory. This issue was not rigorously pursued in this chapter, but is an interesting direction for future work. Furthermore, at present, very little is known about the capacity of these networks. The effectiveness of the various approaches that have been presented in this chapter ultimately depends on the degree of interaction of representations (of the individuals), the variation and selection operators as well as the configuration (coupling) of the ES together with the adaptive linear observer system that was proposed. A key property here is adaptation, which provides the opportunity to modify the resulting hybrid ES-local search algorithm to the problem at hand as well as to alter the configuration and strategy parameters that are used while solutions to the problem are being evolved.

A promising direction for future work would be to allow a more 'natural' representation of the architecture and weights of the network, such that the upper bound of the number of hidden neurons used need not be specified *a priori* – this is beneficial in problems of unknown complexity, where the algorithm would then be allowed to freely add or remove hidden neurons as necessary, depending on the problem difficulty. To further increase the flexibility of the approach, the algorithm should also be allowed to specify the type of nonlinear activation function (for each neuron) – this can be incorporated using one of two ways: (i) allow modification of the parameters of the nonlinearity (such as the scaling, or gradient of the sigmoid), and/or (ii) allow the algorithm to select from a set of predefined nonlinear activation functions (such as the hyperbolic tangent, binary threshold, sigmoidal, or even higher-order polynomial functions). Through this, the versatility of the algorithm is enhanced because for some problems, the dynamical trajectory that is being traced cannot be followed using the current architecture due to the inherent capacity of the structure. Future work would also entail a more detailed comparative study of the performance of the proposed approach against standard system identification schemes, in terms of tracking or prediction accuracy and

architectural or model complexity.

4.2.8 Section Summary

This section presents a hybrid ES-local search algorithm as an adaptive learning algorithm for RNNs in addressing system identification problems, through the simultaneous evolution of connection weights, its mutation step-sizes as well as a separate observer system, which in turn facilitates the exploration of interactions between the structure and weights of the network. Preliminary results obtained highlight the ability of the proposed ES-local search approach in accurately tracking the trajectory or output of a particular system – all achieved using an online, or real-time manner. This gives us evidence that a hybridized technique, when approached correctly, can solve complex problems that would otherwise be unsolvable by standard gradient-based methods. An emerging possibility in this context would be to strengthen the coupling between the ES and the local search technique used here such that the resulting synergistic interaction between them would complement the strengths of these two methods leading to a better performance – present search dynamics, indicate the possibility of constructing hybrid EA-based learning algorithms for neural networks (both feedforward and recurrent) that are extremely versatile. Future work would focus on constructing a more general representation for the evolutionary process to undertake, as well as to perform further comparative analysis on a wider, and more complex range of problem sets.

Chapter 5

Dynamics Analysis and Analog Associative Memory

The additive recurrent network structure of linear threshold neurons represents a class of biologically-motivated models, where nonsaturating transfer functions are necessary for representing neuronal activities, such as that of cortical neurons. This chapter attempts to extend existing results of the dynamics analysis of such linear threshold neurons by establishing new and milder conditions for boundedness and asymptotical stability (while allowing for multistability). As a condition for the asymptotical stability, it is found that boundedness does not require a deterministic matrix to be symmetric or possess positive off-diagonal entries. The conditions put forward an explicit way to design and analyze such networks. Based on the established theory, an alternative approach to study such networks is through permitted and forbidden sets. An application of the LT network includes that of analog associative memory – where a simple design method describing associative memory is suggested. The proposed design method is similar to a generalized Hebbian approach; however, with distinctions of additional network parameters for normalization, excitation and inhibition, both on a global and local scale. Furthermore, a brief outline of the dependence of the computational abilities of the network with respect to its nonlinear dynamics is also made, which is in turn is reliant upon the sparsity of the memory vectors.

5.1 Introduction

Although recurrent networks with linear threshold (LT) neurons were first examined in 1958 in a study of the limulus in the anatomical make-up of the eye [83], only in recent years has interest in the biologically-inspired models been growing [42, 21, 167, 18]. The lack of an upper saturating state in neurons has been supported by experiments of cortical neurons that rarely operate close to saturation [42], which suggests that the upper saturation may not be involved in actual computations of the recurrent network with LT neurons. However, this non-saturating characteristic of a neuron increases the likelihood that the network dynamics may be unbounded, as well as the possibility that no equilibrium point might exist [53]. Therefore, it is worthy of investigating and establishing conditions of boundedness and convergence of LT dynamics to design or exploit the recurrent networks.

Hahnloser (1998) analyzed the computational abilities and dynamics of linear threshold networks, particularly in explaining winner-take-all (WTA) multistability of the network with self-excitation and global inhibition, as well as oscillating behavior of the dynamics when the inhibition delay of the network is increased in an asymmetrical network. Of more recent interest, a functional silicon-based circuit design for a linear threshold network [77] has demonstrated to be able to verify the theory of digital selection (groups of active and inactive neurons) and analog amplification (active neurons are amplified in magnitude, as represented by the gain in neuronal activity). A convergence condition was given by constructing a Lyapunov function for asynchronous neural networks with non-differentiable input-output characteristics [48]. Conditions for generalized networks of linear threshold neurons that ensure boundedness and complete convergence, for symmetrical and asymmetrical weight matrices were established [210,222]. Meanwhile, the studies [214,76] put forward the underlying framework for analyzing networks of linear threshold neurons by examining the system eigenvalues and classifying neurons according to permitted and forbidden sets, where subsets of permitted groups of neurons are likewise permitted, and supersets of forbidden groups of neurons are likewise forbidden. This is a consequence of the digital selection abilities of the network. This chapter places an emphasis on the dynamics analysis for the LT networks, to extend existing results by providing some new and milder conditions for boundedness and asymptotical stability.

It has been shown that complex perceptual grouping tasks, such as feature binding and sensory segmentation, can be accomplished in an LT network composed of competitive layers [209]. Yet the

computational ability of associative memory (also called content-addressable memory) has not been made clear. Recurrent neural network architectures have long been the focus of dynamical associative memory studies since Hopfield's seminal work [93]. However, while the majority of efforts made since then have centered about the use of binary associative memory with discrete-state and saturating neural activities, few in the literature have generalized the original idea to the non-binary case, and even less so to the continuous case. Another focus of this chapter is on the associative memory (especially analog, real-valued auto-associative memory of gray-level images). A design method for the LT network's analog associative memory is proposed and discussed extensively.

5.2 Linear Threshold Neurons

The analog of the firing-rates of actual biological neurons in an artificial neuron is represented by an activation function. Various types of activation functions have been used in modeling the behavior of artificial neurons, and it is from these (nonlinear) activation functions that a network of neurons is able to derive its computational abilities. In recent years, a class of neurons which behaves in a linear manner above a threshold, known as linear threshold (LT) or threshold linear neurons, have received increased attention in the realm of neural computation. Previously, saturation points were modeled in activation functions to account for the refractory period of biological neurons when it 'recharges', and hence cannot be activated.

Computational neuroscientists believe that artificial neurons based on an LT activation function are more appropriate for modeling actual biological neurons. This is quite a radical departure from the traditional line of thought which assumed that biological neurons operate close to their saturating points, as conventional approaches to neural networks were usually based on saturating neurons with upper and lower bounded activities. Studies have demonstrated that cortical neurons rarely operate close to their saturating points, even in the presence of strong recurrent excitation [42]. This suggests that although saturation is present, most (stable) neural activities will seldom reach these levels.

It is believed that there are three generations of artificial neuron models. First-generation artificial neurons were discrete and binary-valued (*digital representation*), such as the classical binary threshold *McCulloch-Pitts* neuron. Second-generation models were based on neurons with continuous activation functions (*analog representation*), with sigmoidal neurons being archetypical. A

third-generation neuron model is the *spiking neuron model*, which is represented by an LT neuron – a non-differentiable, unbounded activation function that behaves linearly above a certain threshold, similar in functionality to (half-wave) rectification in circuit theory. The general form of the LT activation function is defined as

$$\sigma(x) = [x]^+ = k \times max(\theta, x) \tag{5.1}$$

where k and θ respectively denotes the *gain* (usually 1) and *threshold* (usually 0), of the activation function. This form of nonlinearity is also known as threshold, or rectification nonlinearity.

5.3 Linear Threshold Network Dynamics

The network with n linear threshold neurons is described by the additive recurrent model,

$$\dot{x}(t) = -x(t) + W\sigma(x(t)) + h, \tag{5.2}$$

where $x, h \in \Re^n$ are the neural states and external inputs, $W = (w_{ij})_{n\times n}$ the synaptic connection strengths. $\sigma(x) = (\sigma(x_i))$ where $\sigma(x_i) = \max\{0, x_i\}, i = 1, \cdots, n$.

There is another equivalent model which is described by

$$\dot{y}(t) = -y(t) + \sigma(Wy(t) + h). \tag{5.3}$$

The former one is more dynamically plausible than the latter one as suggested in [222], because system (5.3) can be transformed to system (5.2) by a simple linear transformation while it is not this case when transforming (5.2) to (5.3).

The nonlinearity of an LT network is a consequence of (i) the rectification nonlinearity of the LT neurons on an individual level, and (ii) the switching between active and inactive partitions of neurons on a collective level. Although (i) is apparent from the structure of the activation function, (ii) is a more subtle effect, yet contributes significantly to the computational abilities of the network. Specifically, the latter arises from the fact that *an inactive partition of LT neurons do not contribute to the feedback within the recurrent neural network.*

Such a network is said to exhibit hybrid analog-digital properties; while most digital circuits display some form of multistability (recall that feedback in digital flip-flops promotes some form of inherent differential instability), analog circuits in contrast, demonstrates linear amplification. Together, both characteristics co-exist in an LT network, and have been shown to manifest in an actual silicon circuit [77], and is suggested as an underlying mechanism of the neocortex in response to sensory stimulation.

Since at any time t, each LT neuron is either firing (active) or silent (inactive), the whole ensemble of the LT neurons can be divided into a partition P^+ of neurons with positive states $x_i(t) \geq 0$ for $i \in P^+$, and a partition P^- of neurons with negative states $x_i(t) < 0$ for $i \in P^-$. Clearly $P^+ \cup P^- = \{1, \cdots, n\}$.

Define a matrix $\Phi = \text{diag}(\phi_1, \cdots, \phi_n)$, where $\phi_i = 1$ for $i \in P^+$ and $\phi_i = 0$ for $i \in P^-$. The matrix Φ carries the digital information about which neurons are active. It is obviously

$$W\sigma(x) = W\Phi x. \tag{5.4}$$

Let $W^e = W\Phi$, called the *effective recurrence matrix* [76], which describes the connectivity of the operating network, where only active neurons are dynamically relevant.

Theorem 3. *Let Φ define a digital carry matrix. If there exists a matrix $\hat{W} \geq W$ satisfying one of the conditions*

$$w_{ii} + \sum_j |\hat{w}_{ij}|\phi_j - 1 < 0, \tag{5.5a}$$

$$Re\{\lambda_{max}\{\hat{W}\Phi - I\}\} < 0, \tag{5.5b}$$

$$Re\{\lambda_{max}\{\hat{W}\Phi\}\} < 1. \tag{5.5c}$$

for all i, then the LT dynamics (5.2) is bounded in the partitions defined by Φ.

Proof. It holds that

$$\begin{aligned} \dot{x} &= -x + W\sigma(x(s)) + h \\ &\leq -x + \hat{W}\sigma(x(s)) + h \\ &= -(\mathrm{I} - \hat{W}\Phi)x(s) + h. \end{aligned} \tag{5.6}$$

Let $y(t) = x(t) - h$, then

$$\dot{y} \leq -(\mathrm{I} - \hat{W}\Phi)y(t). \tag{5.7}$$

Define $\dot{z}(t) = -(\mathrm{I}-\hat{W}\Phi)z(t)$. Let λ_i and $\mathbf{v}_i, i = 1, ..., n$ be the eigenvalues and n linearly independent eigenvectors of $-(\mathrm{I} - \hat{W}\Phi)$, respectively. According to the linear system theory [19], then $z(t)$ can be decomposed as $z(t) = \sum_i \xi_i(t)\mathbf{v}_i$, where $\xi_i(t), i = 1, ...n$ are functions of time. Then

$$z(t) = \sum_{i=1}^{n} e^{\lambda_i(t-t_0)}\mathbf{v}_i\xi_i(0). \tag{5.8}$$

Obviously $z(t)$ has an upper bound $m > 0$ if $\lambda_i < 0$. According to the comparison principle, $y(t) \leq z(t) \leq m$. Thus

$$x(t) \leq m + h$$

for all $t \geq 0$. This shows that $x_i(t)$ is upper bounded. Furthermore, the trajectory $x(t)$ of network (5.2) satisfies

$$x_i(t) = x_i(t_0)e^{-(t-t_0)} + \int_{t_0}^{t} e^{-(t-s)} \left(\sum_{j=1}^{n} w_{ij}\sigma(x_j(s)) + h_i \right) ds, \tag{5.9}$$

for all $t \geq t_0$. Then

$$|x_i(t)| \leq |x_i(t_0)|e^{-(t-t_0)} + (m + h)\sum_{j=1}^{n} |w_{ij}| + |h_i|, \quad i = 1, ..., n. \tag{5.10}$$

This proves that the LT dynamics is bounded in the partition defined by Φ if (5.5b) holds. It is obviously to derive (5.5c). Furthermore, by applying Gerschgorin's theorem [63], (5.5a) is easily obtained from (5.5b). The proof is completed. □

A set of active neurons is called a *permitted set* if they can be coactivated at a stable steady

state by some input; otherwise, it is termed a *forbidden set*. Permitted set and forbidden set were firstly studied in [77], then further discussed in [214] and [78]. The assertion given for the superset of a forbidden set and the subset of a permitted set, nevertheless, was addressed in a more likely intuitive manner. Herein, from the above established theorem, such results are proven as follows.

Corollary 4. *Any subset of a permitted set is also permitted; any superset of a forbidden set is also forbidden.*

Proof. Let $\Gamma = \{\gamma_i\}$ denote a set of m active neurons, $\gamma_i \in \{1, ..., n\}, i = 1, ..., m$. Firstly, suppose Γ is a forbidden set. Since $\phi_j = 1$ if $j \in \Gamma$ and $\phi_j = 0$ otherwise, according to Theorem 1, the following condition is met,

$$w_{ii} + \sum_{j \in \Gamma} |\hat{w}_{ij}| \phi_j > 1.$$

Then its superset $\hat{\Gamma}$, with more active neurons, obviously results in

$$w_{ii} + \sum_{j \in \hat{\Gamma}} |\hat{w}_{ij}| \phi_j > w_{ii} + \sum_{j \in \Gamma} |\hat{w}_{ij}| \phi_j > 1.$$

Thus $\hat{\Gamma}$ is also forbidden. It is similarly to prove for permitted set. □

Next, the conditions which ensure that the LT dynamics is bounded regardless of partitions, are presented. Firstly, the following lemma is introduced.

Lemma 5. *Let $y(t)$ denote the solution of the following equation*

$$\dot{y}(t) = -y(t) + \sigma\left(Wy(t) + h + (x(0) - Wy(0) - h)e^{-t}\right).$$

Then for any $x(0) \in \Re^n$, the solution $x(t)$ of network (5.2) starting from $x(0)$ can be represented by

$$x(t) = Wy(t) + h + (x(0) - Wy(0) - h)\, e^{-t}.$$

This proof can be referred to [222].

Theorem 6. *If there exists a matrix $\hat{W} \geq W$ satisfying one of the following:*

$$w_{ii} + \sum_j |\hat{w}_{ij}| - 1 < 0, \tag{5.11a}$$

$$Re\{\lambda_{max}\{\hat{W} - I\}\} < 0, \tag{5.11b}$$

$$Re\{\lambda_{max}\{\hat{W}\}\} < 1, \tag{5.11c}$$

for all i, then the LT dynamics (5.2) is bounded.

Proof. Consider the following dynamical system

$$\dot{y}(t) = -y(t) + \sigma(Wy(t) + h), \ t \geq 0, \tag{5.12}$$

of which the solution is calculated as

$$y(t) = e^{-t}y(0) + \int_0^t e^{-(t-s)}\sigma(Wy(s) + h)ds. \tag{5.13}$$

By taking $x(0) = Wy(0) + h$, recall Lemma 1, the dynamics (5.2) are associated with (5.12) as

$$x(t) = Wy(t) + h. \tag{5.14}$$

It means that any trajectory $y(t)$ starting from $y(0)$ can be transformed to the trajectory $x(t)$ starting from $Wy(0) + h$. Consider the trajectory formulation (5.13), the dynamics (5.12) possess a property of nonnegativity: if $y(0) \geq 0$ then $y(t) \geq 0$ and if $y(0) < 0$ then $y(t_1) \geq 0$ after some transient time $t_1 > 0$. Consequently, in (5.12) it is assumed that $y_i(t) \geq 0$ for all i. Now we have either $\dot{y}(t) = -y(t) \leq 0$ or

$$\begin{aligned} \dot{y}(t) &= -y + Wy + h \\ &\leq -y + \hat{W}y + h \\ &= -(\mathrm{I} - \hat{W})y + h. \end{aligned} \tag{5.15}$$

Thus if $Re\{\lambda_{max}\{\mathrm{I} - \hat{W}\}\} > 0$, following the method in the proof of Theorem 1, $y(t)$ is bounded

for all $t \geq 0$. Therefore, in light of (5.14), $x(t)$ is also bounded. The rest conditions are easily derived. □

In contrast to that of Theorem 1, the condition of Theorem 2 implies that the dynamics (5.2) is bounded in any of partitions, thus it results in global boundedness.

Note that the components of $\hat{W}$ need not be positive. Define a matrix $W^+ = (w_{ij}^+)$, where

$$w_{ij} = \begin{cases} w_{ii}, & i = j, \\ \max(0, w_{ij}), & i \neq j. \end{cases}$$

Clearly, W^+ provides a straightforward choice of $\hat{W}$ with nonnegative off-diagonal elements. As a consequence, all the analytical results for $\hat{W}$ are applicable to W^+, which give rise to equivalent conditions as those of [210] for nonsymmetric networks. Unlike their argument that required W and $\hat{W}$ to be symmetric, furthermore, note that Theorem 2 where $\hat{W}$ need not be symmetric is applicable to both symmetric and nonsymmetric networks. Hence it gives a wider rule.

Lemma 7. *For any $x, y \in \Re^n$, the linear threshold transfer function $\sigma(\cdot)$ has the following properties:*

$$\begin{aligned} &i). \quad \|\sigma(x) - \sigma(y)\|^2 \leq (x-y)^T[\sigma(x) - \sigma(y)], \\ &ii). \quad [\sigma(y) - y]^T[\sigma(y) - u] \leq 0, \quad u_i \geq 0 (i = 1, ...n), \\ &iii). \quad x^T[\sigma(u-x) - u] \leq -\|\sigma(u-x) - u\|^2, \quad u_i \geq 0 (i = 1, ...n). \end{aligned}$$

Proof. (i) can be concluded from Lemma 2 of [222]. To prove (ii),

$$\begin{aligned} [\sigma(x) - x]^T[\sigma(x) - u] &= [\sigma(x) - x]^T[\sigma(x) - \sigma(u)] \\ &= [\sigma(x) - \sigma(u)]^T[\sigma(x) - \sigma(u)] + [\sigma(u) - x]^T[\sigma(x) - \sigma(u)] \\ &= \|\sigma(x) - \sigma(u)\|^2 - [x-u]^T[\sigma(x) - \sigma(u)] \\ &\leq 0. \end{aligned}$$

The last inequality is resulted from (i). Let $y = u - x$ and substitute into (ii), it follows

$$[\sigma(u-x) - u + x]^T[\sigma(u-x) - u] \leq 0,$$

which is equivalent to

$$[\sigma(u-x)-u]^T[\sigma(u-x)-u]+x^T[\sigma(u-x)-u]\leq 0.$$

Thus

$$x^T[\sigma(u-x)-u]\leq -\|\sigma(u-x)-u\|^2.$$

This completes the proof. □

Theorem 8. *If W is symmetric and there exists a $\hat{W}$ satisfying one of conditions (5.11), then the LT dynamics (5.2) is Lyapunov asymptotically stable.*

Proof. The LT dynamics (5.3) admits an energy-like function

$$E(t)=\frac{1}{2}y^T(t)(\mathrm{I}-W)y(t)-h^Ty(t) \tag{5.16}$$

for all $t\geq 0$. Obviously, $E(0)=0$. According to Theorem 2, $y(t)$ is bounded. It is easy to show that $E(t)$ is also bounded. It is derived from Lemma 1 that

$$\begin{aligned}\dot{E}(t) &= [y(t)(\mathrm{I}-W)-h]^T\dot{y}(t)\\ &= [y(t)-(Wy(t)+h)]^T(-y(t)+\sigma(Wy(t)+h))\\ &= -[y(t)-(Wy(t)+h)]^T[\sigma(y(t))-\sigma(Wy(t)+h)]\\ &\leq -\|y(t)-\sigma(Wy(t)+h)\|^2\\ &= -\|\dot{y}(t)\|^2. \end{aligned}\tag{5.17}$$

Hence $E(t)$ is monodecreasing. Since it is bounded below, $E(t)$ must approach a limit as $t\to\infty$ and thus $\dot{E}(t)=0$ as $t\to\infty$. So $\dot{y}(t)\to 0$ as $t\to\infty$. It follows that

$$\lim_{t\to\infty}\dot{x}(t)=W\lim_{t\to\infty}\dot{y}(t)=0.$$

Eventually, based on an analogous analysis as [222], it is concluded that any trajectory of the network approaches one of its equilibria, that is to say the dynamics is Lyapunov asymptotically stable. □

In contrast to global asymptotical stability that admits exactly one equilibrium, the above theorem does obey such restriction and thus allows for *multistable* dynamics stated in [76, 210].

While the synaptic weight matrix W determines the stability of the dynamics, the input vector h determines the analog responses of the active neurons at steady state. By replacing the rectification function $\sigma(x)$ with Φ (only when the identities of the active neurons are known *a priori*), the steady state is computed by

$$(\mathrm{I} - W\Phi)x = h.$$

Thus $x = (\mathrm{I} - W\Phi)^{-1}h$. Using a power series expansion,

$$(\mathrm{I} - W\Phi)^{-1}h = (\mathrm{I} + W\Phi + (W\Phi)^2 + ...)h. \tag{5.18}$$

This can be interpreted as the signal flowing through the partition of active neurons in a first synapse pass, a second synapse pass, and so on. Mathematically, the power series expansion allows the determination of the analog responses of the active neurons without explicitly computing the inverse of (I-W), which may not be possible in large networks.

Together, W and h determines the identity and response of the active neurons at steady state, thus the LT network can perform computations involving both digital selection and analog amplification, as realized in a silicon circuit [77].

5.4 Analog Associative Memory and The Design Method

5.4.1 Analog Associative Memory

From a physiological or biological viewpoint, memory can be divided into two main categories – short-term and long-term memory. Short-term memory, it is believed, arises from persistent activity of neurons, where the time-frame involved is in the order of seconds to minutes. On the other hand, long-term memory involves storage of memory patterns through actual physical/neuronal modifications of synaptic strengths. Such changes, which usually persist for period from tens of minutes or longer [37] are generally known as long-term potentiation (LTP) and long-term depression (LTD) – and of these

types, the longest lasting forms of LTP and LTD require some kind of protein synthesis to modify the synaptic connections.

Another categorical measure of associative memory is the input stimulus used [117]. *Auto-associative memory* refers to the presentation of a partial or noisy version of the desired memory state which is to be recalled. An example would be presentation of a noisy version of A, say A' to recall A. *Hetero-associative memory*, in contrast, is said to occur when an input stimulus that is different from the memory to be recalled is presented to the network. An example would be presentation of an input stimulus B to recall A. The focus of this chapter is centered on long-term auto-associative memory.

The dynamics of early associative memory models involved the use of input patterns (in a partial or approximate form) h as the starting conditions for the network dynamics. The most recognizable type of network based upon such a principle is the Hopfield network [93] of binary neurons, which incorporate recurrent connections between neurons, thus allowing the current state of the network to directly influence its state in the future. These states thus evolve systematically over time, tracing out a trajectory in the space of possible values. The dynamics of the recurrent connections of neurons leverage upon the shape of these trajectories to perform computations, allowing stored patterns (the desired or intended patterns to be recalled) to be encoded as fixed-points of the network, with the 'noisy' or corrupted versions presented at the input as the probe vector – these initial conditions that are started with then 'flow' into a state-space trajectory that will converge to their equilibrium points. Error correction or pattern retrieval is achieved by these initial states or probes being attracted to the stored fixed-points.

However, which of these fixed equilibrium points that are arrived at, largely depends on where these initial conditions begin: only if these starting points lie within the basin of attraction of the desired pattern (for binary patterns, the Hamming distance measures the difference between the initial probe and the desired pattern) will these starting points converge to an intended pattern. Useful pattern matching will in turn require each fixed-point to have a sufficiently large basin of attraction.

Table 5.1: Nomenclature

m	*no. of patterns*
δ	*normalization term*
$\nu(\mu)$	*contribution of each pattern,* $\mu = 1, ..., m$.
κ	*removal of self-correlation term*
ξ^μ	μ*th pattern vector* ξ
$\mathbf{I}$	*identity matrix*
$\mathbf{1}\mathbf{1}^T$	*matrix of ones*
α	> 1 :*global excitation;* < 1 : *divisive inhibition*
β	*global inhibition*
ω	*self-coupling for eigenvalue 'placement'*
W_c	*analog correlation matrix*

5.4.2 The Design Method

Essentially, the key proposition of the Hebbian rule is that, during the learning phase, pre-synaptic signals and post-synaptic activity that fire at the same time or within a short time period of each other are more closely correlated, which is reflected in the synaptic weight connection between these two neurons. There should then be a greater (excitatory) connection between neurons i and j. Such adaptation method is described as follows ($\eta > 0$ denotes the learning rate)

$$\Delta\ w_{ij} \propto\ \xi_i \xi_j \tag{5.19a}$$

$$\Delta\ w_{ij} = \eta \xi_i \xi_j \tag{5.19b}$$

During the training phase, the network learns by potentiating the synapse between two neurons if both neurons are active (or inactive), and depressing the synapse if one neuron is active (inactive) and the other is inactive (active). The inclusion of depressive action in reducing the synaptic strength is a more general form of the Hebb's original postulate. Other, more general form of this rule is that synapses should change according to, and proportionally to the degree of correlation or covariance of the activities of both the pre-synaptic and post-synaptic neurons [37].

For gray-level image storage of an N-dimensional pattern/memory vector, the synaptic weight

matrix is set-up using the following design approach (which is not unlike a generalized Hebbian rule for learning)

$$\mathbf{W} = \{\alpha\{\frac{\delta}{m}\sum_{\mu=1}^{m}\{\nu(\mu)\sqrt{\xi^{\mu}\xi^{\mu T}} - \kappa\mathbf{I}\}\} - \beta\mathbf{1}\mathbf{1}^{T}\} - \omega\mathbf{I} \tag{5.20a}$$

$$\mathbf{W} = \{\alpha W_c - \beta\mathbf{1}\mathbf{1}^{T}\} - \omega\mathbf{I} \tag{5.20b}$$

The proposed approach uses small, positive (excitatory) synaptic weights mediated by inhibition for non-divergence of the linear threshold network dynamics. Inhibitive self-coupling further allows the fine-tuning of the stability margin of the system. Because self-couplings are denoted as the diagonal entries in the synaptic weight matrix $\mathbf{W}$, control of the eigenvalues can be done directly (however, without knowledge of the actual values). Suppose $\lambda_i, i = 1, ..., n$, are eigenvalues of $\mathbf{W}$. By introducing an exogenous variable ω that only shifts the diagonals of $\mathbf{W}$, the eigenvalues of the system are shifted uniformly by $(\lambda_i + \omega)$. In a further step, after scaling by a factor α, i.e., applying a transformation to $\mathbf{W}$ as $\alpha\mathbf{W} + \omega\mathbf{I}$, the eigenvalues are now changed to $\alpha\lambda_i + \omega$ for all i, where ω can be either positive or negative.

The three parameters (termed as 'external' parameters, as opposed to the 'internal' parameters as those found within the correlation matrix W_c) are α, β and ω. These three parameters are considered to be 'external' by virtue that they are not dependent on the input patterns (images).

The nomenclature for the terms used in the proposed design approach is listed in Table I. The primary difficulties with using the Hebbian rule for storing analog patterns, are firstly, the possibility of instability, and secondly, excessive, unbalanced inhibition. The first problem arises because the possibility of instability is caused by the method used to set up the synaptic weights (as correlations between pixel gray levels). If saturating neurons were used instead of LT neurons, neural activities would be driven to their saturation values. On the other hand, the second problem occurs because of the large range of possible gray level values: leaving darker pixels inhibited and brighter pixels excited, in effect, a partitioning occurs between active (very large activities) neurons and silent neurons (at threshold).

To further elucidate, consider the following: for gray scale images, possible gray-level values reside in the range [0,255]. The idea that is being put forward here is that neurons corresponding to

higher gray-level values should have, accordingly greater excitatory connections with other neurons. However, the problem with the preceding approach is that neurons corresponding to lower gray level values have less excitatory connections with other neurons. Thus some of these neurons are inhibitive (negative connective weights) after introducing inhibition. This in turn leads to the (perpetual) state at which the group of neurons with lower gray level values will always be inhibited by those more active neurons (because of the inhibitive connections), and hence are subsequently silent at steady state.

In view of the aforementioned problem, the key idea in the proposed design method to be used in a network of LT neurons for continuous, real-valued associative memory is to limit the degree of inhibition introduced, such that most synaptic connections are excitatory ($w_{ij} > 0$) only to a small extent. Divisive inhibition is preferably used ($\alpha < 1$), as opposed to subtractive inhibition (β). However, the self-couplings of the neurons in the network are usually made inhibitive to prevent runaway excitation in the dynamics of the network (in simpler terms, this means that the diagonals of the network matrix are negative), though this condition is not always necessary; small, finitely positive values can also be used for self-excitation, and yet retain stability. Moreover, the off-diagonals are also small but finite values, which can either be excitatory or inhibitory.

A modulating term (δ) that is introduced to the weight matrix $\mathbf{W}$ assists in normalizing the synaptic weights that connects a neuron i to all other neurons and vice-versa. It is a global parameter, as it is applied to all synaptic connections in the weight matrix $\mathbf{W}$. Supposing there are N neurons, and assuming the maximum gray level value in the stored image/pattern is $I_{\max}$. In the worst case scenario, all N neurons would consist of the maximum gray level value ($I_{\max}$). To approximate the value of δ to be used,

$$\delta N I_{\max} < 1 \Rightarrow \delta < \frac{1}{N I_{\max}} \tag{5.21}$$

5.4.3 Strategies of Measures and Interpretation

A measure of the difference between two analog patterns, can be quantified by comparing the absolute differences in the original pattern and the distorted pattern, to the difference in the original pattern

and the corrected/retrieved pattern, namely, the signal-to-noise ratio (SNR) computed as

$$SNR = \frac{\sum_i^N \sum_j^N (I_{original}(i,j) - I_{noise}(i,j))^2}{\sum_i^N \sum_j^N (I_{original}(i,j) - I_{retrieval}(i,j))^2}. \tag{5.22}$$

The greater the SNR value, the better the correction (retrieval) process is, as performed by the associative memory network. An SNR value larger than unity indicates that error-correction has occurred - however, the tasks that are performed by an image filter and that of an associative memory system are, at a conceptual level, quite different. While a filter is expected to remove noise from any signal, the associative memory system is designed to only filter the noise on prototype/probe vectors only [141].

The SNR value measures the absolute differences in gray-level values by quantifying the degree to which error correction has been made. However, this measure may not be particularly good for measuring the difference in content, or structure of the retrieved image with the original stored image (where relative difference is more important).

Interpretation of the neural activities at steady-state is not possible unless some form of quantization is imposed on the final neural activities. The final image that is obtained from simulating the network dynamics can be interpreted by linearly scaling these neural activities according to the minimum and maximum gray level value ($I_{\min}$ and $I_{\max}$ respectively)

$$\bar{x} = I_{\min} + x \times \frac{I_{\max} - I_{\min}}{x_{\max} - x_{\min}} \tag{5.23}$$

This process if less of a biological mechanism, but is useful in simulations to better visualize the readout process of the final neural activities because of the large range of their values.

5.5 Simulation Results

The dynamics of an LT network is capable of amplifying arbitrarily small differences in the inputs – the explicit presence of the input (h_i) in the state update equation also introduces some interesting behavior into the dynamics of the network, and as [210] has illustrated, multistability allows a system to exhibit symmetry-breaking, where the dynamics allows for nonlinear amplification of small input

differences. Moreover, assuming that the state (x^*) is a fixed-point in the network (or a memory state), (ρx^*) is also a fixed-point (memory state) of the network, so that uniform scaling of active neurons is possible while leaving all inactive neurons unchanged. Care, however, has to be taken in ensuring that the input vector h_i is approximately of the same order as $\sum_j^N w_{ij}x_j$ to prevent either term from dominating.

The update of neuron states can be achieved either synchronously (all neurons in parallel at once), or asynchronously (one neuron at a time in a serial manner and at any time step, each neuron has a uniform probability of being selected for updating). From the previous analysis for the dynamics, the energy function of an LT network assumes the form

$$E(x) = \frac{1}{2}x(\mathbf{I} - \mathbf{W})x^T - h^T x \tag{5.24}$$

A system which is stable and convergent would tend to a certain energy limit after some transient time or a number of iterations. In the case of an unstable system, the energy would increase exponentially as the individual neural activities of neurons experience runaway excitation. An oscillatory system is characterized by fluctuations (which may be upper and lower bounded) without tending to a fixed energy value. The trend of the energy function is more important than the final value, unless the problem is that of optimization, such as quadratic optimization [196].

5.5.1 Small-Scale Example

In this example, a smaller network of *N=9* neurons was used. Smaller networks are easier to analyze in terms of its eigen-structure. A 2-dimensional analog pattern was stored in the network, and subsequently recalled using a noisy input. The network parameters are: $\alpha = 4.5, \beta = 0.6, \omega = -0.2$ and $\max(w_{ii} + \sum_{j\neq i}^N w_{ij}^+) = 0.9365$. The N eigenvalues of the system matrix W are

$$-3.8805, -1.7750, -1.7750, -1.7750, -1.7750, -1.7750, -1.7750, -1.7750, 1.2638,$$

of which the largest eigenvalue is above unity. Since $\max(w_{ii} + \sum_{j\neq i}^N w_{ij}^+) < 1$ and W is symmetric, according to Theorem 3, the network is asymptotically stable. The original pattern and the noisy

input (probe vector) are respectively,

$$\begin{bmatrix} 7 & 7 & 3 \\ 1 & 5 & 9 \\ 0 & 4 & 2 \end{bmatrix} \text{ and } \begin{bmatrix} 6.9158 & 7.9138 & 3.8590 \\ 0.0871 & 5.2176 & 8.8599 \\ -0.8009 & 3.4935 & 1.0980 \end{bmatrix}.$$

After convergence, the final neural activities x and the retrieved pattern vector (after linear scaling) are respectively,

$$\begin{bmatrix} 27.9295 & 27.9600 & 7.7582 \\ 0 & 18.5496 & 35.5272 \\ 0 & 13.7720 & 1.1145 \end{bmatrix} \text{ and } \begin{bmatrix} 8 & 8 & 2 \\ 1 & 5 & 9 \\ 0 & 4 & 1 \end{bmatrix}.$$

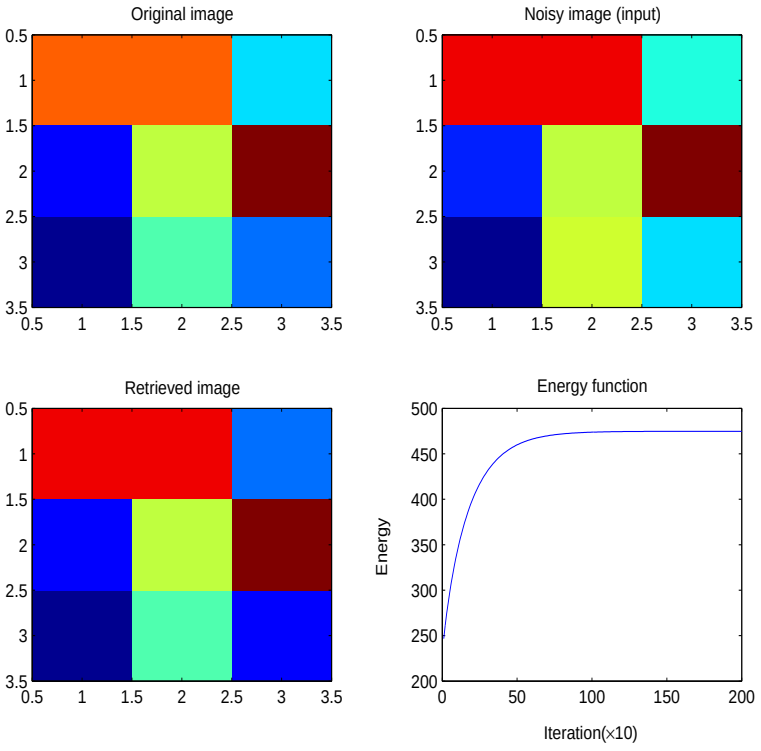

Figure 5.1: Original and retrieved patterns with stable dynamics

Fig. 5.1 shows the retrieval process and the convergence of energy function (for the sake of illustration, the energy function takes its reverse sign herefrom). Fig. 5.2 depicts the convergence of individual neuron activity. Note that if $\omega = -0.1$ was to be shifted to $\omega = 0.1$, the eigenvalues

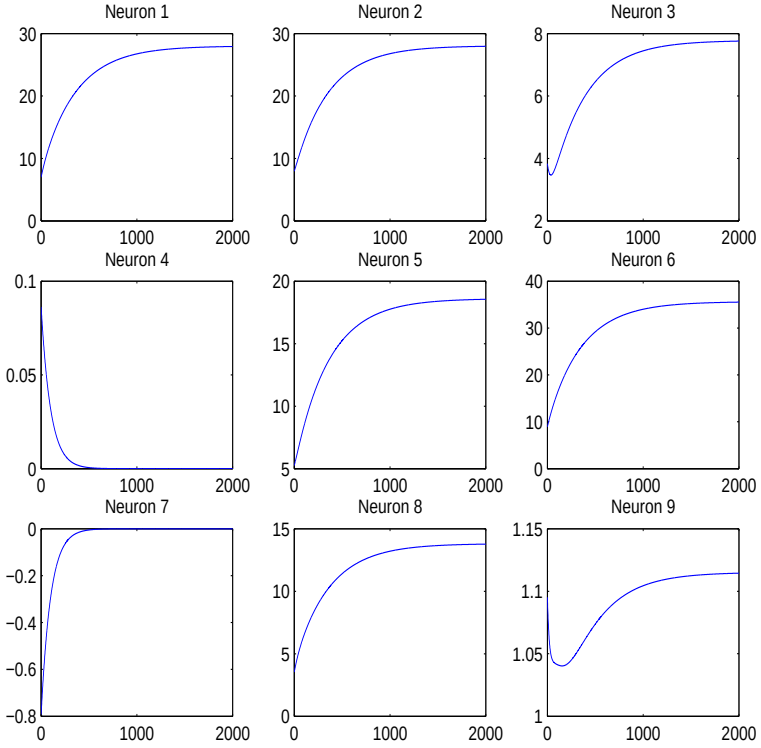

Figure 5.2: Illustration of convergent individual neuron activity

would all be shifted by +0.2, making the largest eigenvalue of W^+ to be larger than unity (also $\max(w_{ii} + \sum_{j \neq i}^{N} w_{ij}^{+}) = 1.2365 > 1$) and hence create an unstable system. On the contrary, if ω to be shifted to a more negative value, it would result in a faster convergence. *Self-coupling can therefore be used to tune the stability margins of the system.*

5.5.2 Single Stored Images

Four different gray scale images of resolution 32×32 with 256 gray-levels were stored in the network independently of each other (Fig. 5.3). The scaling parameter α proves to be an interesting parameter for consideration because its role is two-fold: if $\alpha > 1$, it controls the excitation; if $\alpha < 1$, it controls the amount of (divisive) inhibition in the network. Because excitation is inversely related to divisive inhibition, increasing α is similar to increasing the instability (excitation) while decreasing the amount of (divisive) inhibition. The input (probe) vectors were noisy versions of the original gray scale images (Gaussian noise with mean 0 and variance of 10% or 50% salt-&-pepper noise).

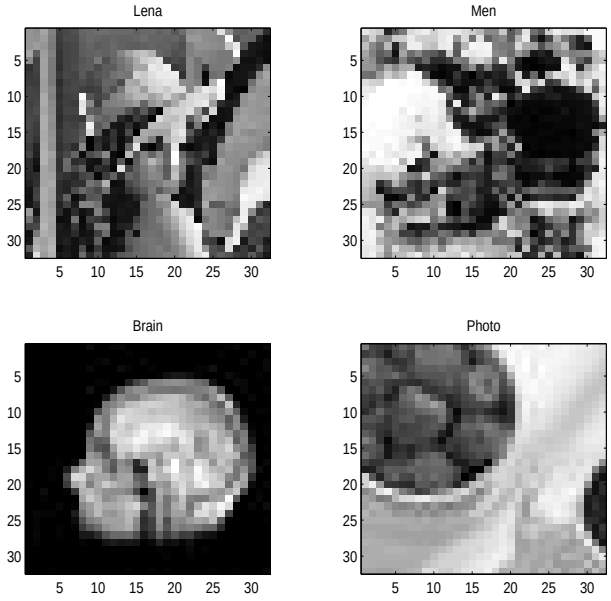

Figure 5.3: Collage of the 4, 32×32, 256 gray-level images used

Because of the input dependency of the LT dynamics, certain types of noise will not result in reliable retrieval because neurons with zero inputs will remain inactive at steady-state. Unless otherwise specified, the network parameters used in the simulations are, $\nu = 1$, $\kappa = 0.35$; m and σ are pattern-dependent. As to the other parameters, α was varied such that the excitation measured by $w_{ii} + \sum_{j \neq i}^{N} w_{ij}^{+}, i = 1, ..., N$ was increased; β and ω were kept at zero and the resulting SNR value calculated.

Figs. 5.4–5.5 illustrate the variation in the quality of recall measured by the SNR value in the retrieval process of the four gray scale images as the excitation is increased (correspondingly, divisive inhibition is decreased), where the excitation strength, denoted by MaxW^{+}, is calculated as $\max_i\{w_{ii} + \sum_{j \neq i}^{N} w_{ij}^{+}\}$. From the resulting plots, it is interesting to note the range of values of α that would result in the best recalled image, as measured by the SNR. These figures give a general trend of how the SNR measure varies: as α increases up to a certain point (competition/instability likewise becomes greater), the SNR will correspondingly increase. After this peak, greater competition will result in overall instability in the network; consequently, divergence of individual neural activity will occur resulting in a marked decrease in the SNR. The optimal α value is image-dependent, closely related to the overall composition and contrast of the image. More precise tuning of the network

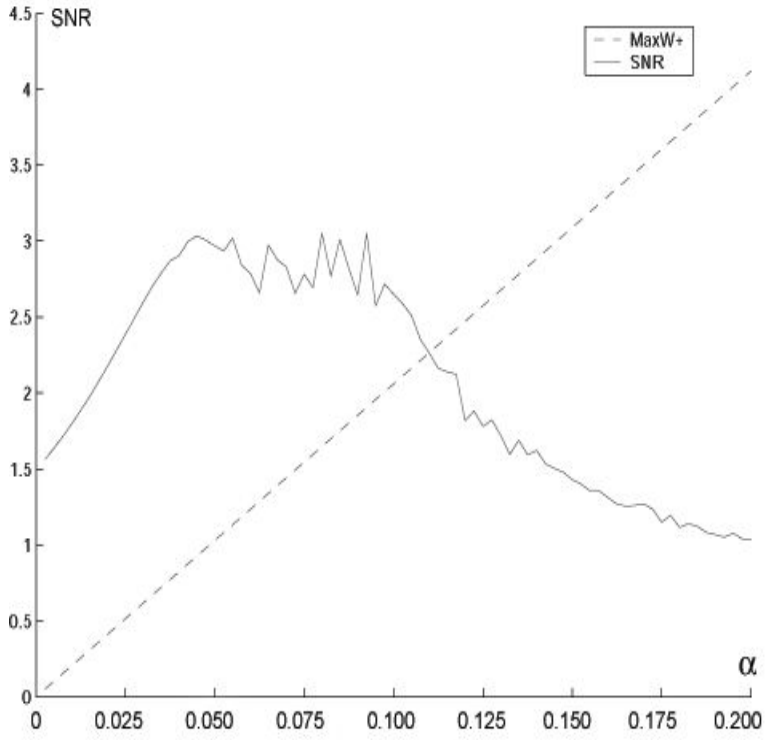

Figure 5.4: Lena: SNR and MaxW$^+$ with α in increments of 0.0025

parameters (α, β, and ω) will result in better quality memory retrieval as shown in Figs. 5.6–5.8.

5.5.3 Multiple Stored Images

Now the four images (Fig. 5.3) were stored together in the same network (not independently, as in the previous approach). Expectedly, the quality of recall decreases, yet the retrieved image is still quite apparent (see Fig. 5.9). 50% Salt-&-Pepper noise was used (Gaussian noise was also used with similar results). The network parameters were set as $\alpha = 0.24, \beta = 0.0045, \omega = 0.6$, with $SNR = 1.8689$.

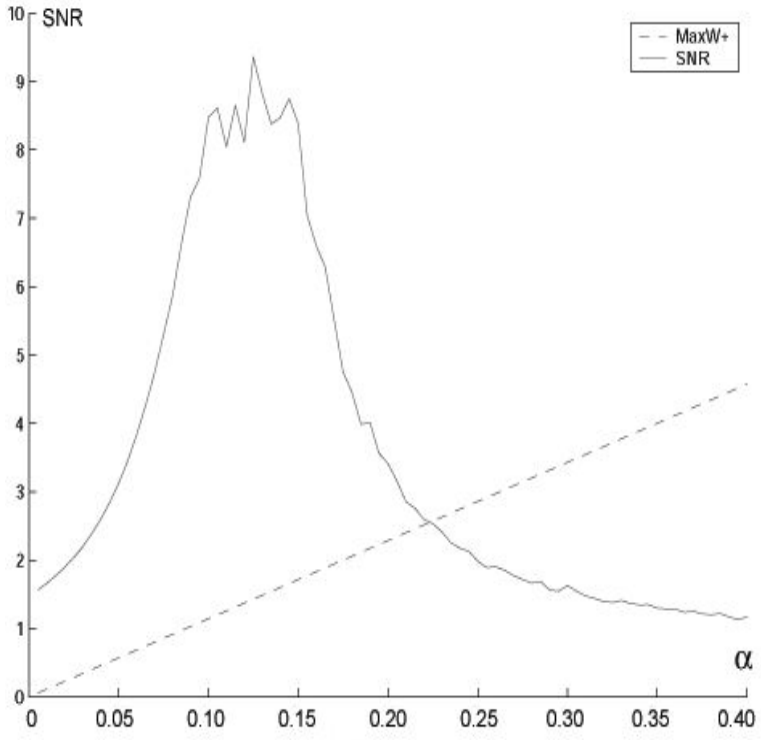

Figure 5.5: Brain: SNR and MaxW$^+$ with α in increments of 0.005

5.6 Discussion

5.6.1 Performance Metrics

A typical performance metric of associate memory is the ability of the network to recall *noisy* or *partial* memory vectors presented at the inputs. This ability is in turn, dependent on the sparsity of the stored memory vectors [93]. This requirement of sparsity places restrictive constraints on the types of images that are stored in an associative memory network, which is even more severe when dealing with gray scale images because large regions of gray scale images have positive gray-levels. Likewise, the *information capacity per neuron* for an LT neuron is difficult to quantify because of its unbounded, continuous characteristic (unlike discrete-valued binary neurons). The performance of the network using the proposed approach, cannot be compared vis-á-vis with that of the Hopfield network – primarily because the approach outlined here is concerned mainly with a range of graylevel values, much unlike the Hopfield model which focuses on discrete binary values.

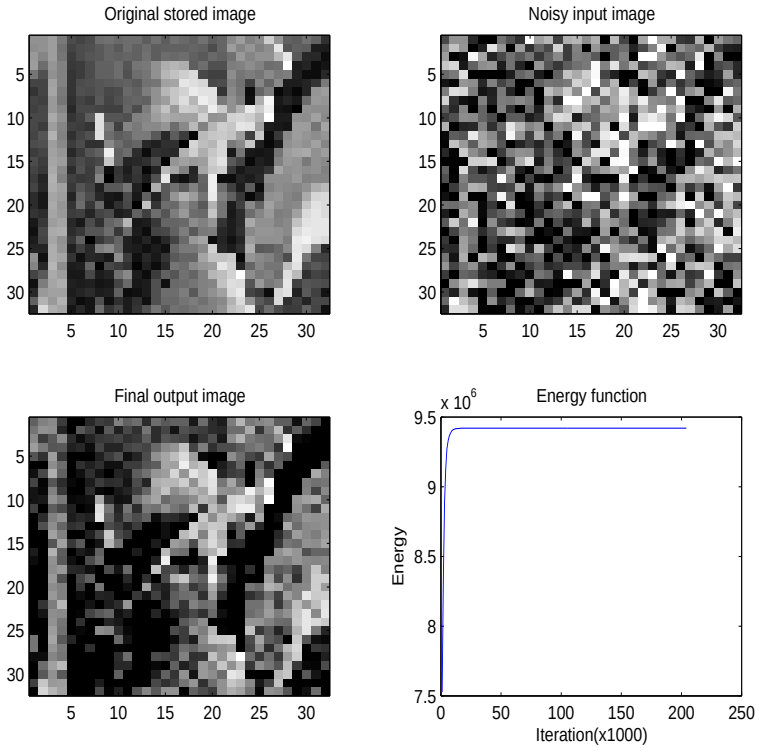

Figure 5.6: Lena: $\alpha = 0.32, \beta = 0.0045, \omega = -0.6, SNR = 5.6306$; zero mean Gaussian noise with 10% variance

5.6.2 Competition and Stability

Competition is a corollary of instability, which promotes separation of neural activities in a recurrent neural network. It can be observed from Figs. 5.4–5.5 that the SNR increases as $w_{ii} + \sum_{j \neq i}^{N} w_{ij}^{+}$ (regarded as a measure of competition/instability) increases, until it results in divergent neural activities and hence a decrease in the SNR. The use of a saturating neuron in competitive dynamics would result in activities that are close to the saturation points of the activation function, the LT neuron, in a stable LT network, however, allows its neural activities to run to arbitrarily high value, until convergence.

Multistability is a manifestation of instability in an LT network that arises from the existence of unstable differential-mode eigenvalues [77]. The LT dynamics results in nonlinear switching between active and inactive partitions of neurons until neurons corresponding to unstable eigenvalues are eventually inactivated. Convergence then occurs after the nonlinear switching selects a permitted

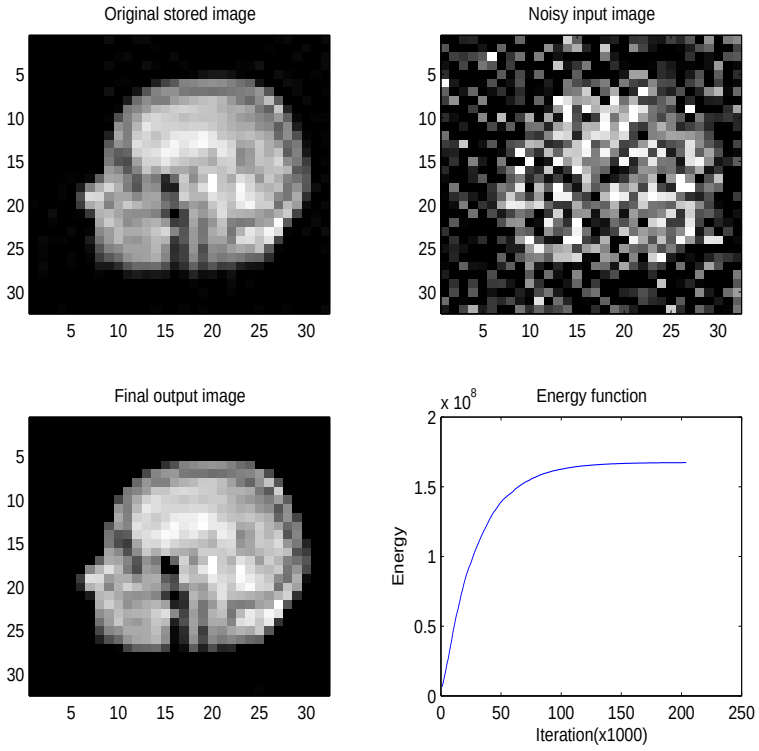

Figure 5.7: Brain: $\alpha = 0.43, \beta = 0.0045, \omega = -0.6, SNR = 113.8802$; zero mean Gaussian noise with 10% variance

set of neurons and the linear dynamics of the LT network then takes over to provide for the analog responses of these active sets of neurons.

5.6.3 Sparsity and Nonlinear Dynamics

The computational abilities of any neural network is a result of the degree of nonlinearity present in the network. Because the nonlinear dynamics of the LT network is a consequence of the switching between active and inactive partitions of neurons, since inactive neurons do not contribute to the strength of feedback in the network. This form of *dynamic feedback* depends on the ratio of the number of inactive neurons to active neurons, which is a measure of the *sparsity* of the memory vectors that are stored. Recall that an LT neuron behaves linearly above threshold, hence if the pattern vector that is stored in the associative memory is dense (less sparse), it can be expected that the network to behave more like a linear system, and in a way, lose its computational abilities derived from its nonlinear switching activities.

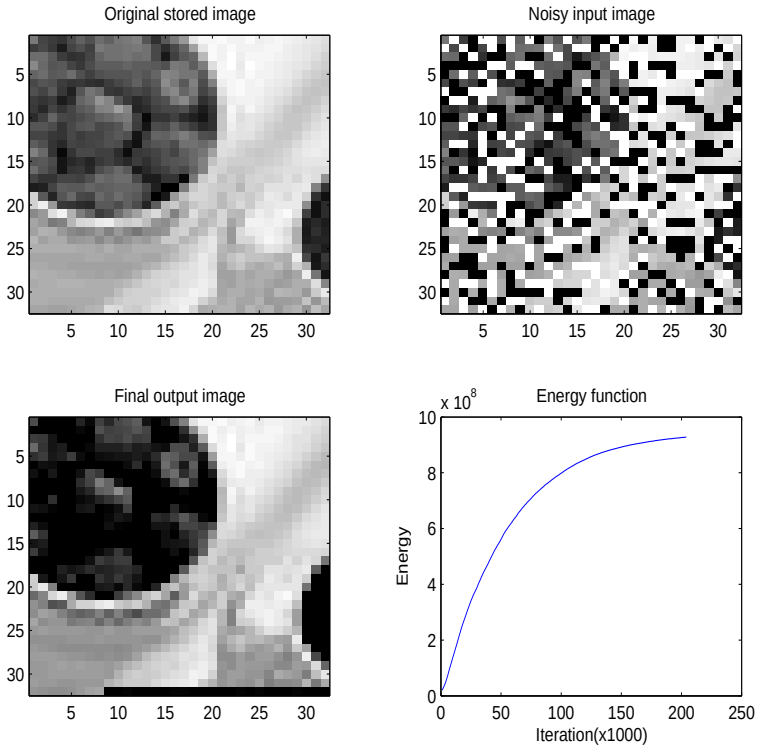

Figure 5.8: Strawberry: $\alpha = 0.24, \beta = 0.0045, \omega = 0.6, SNR = 1.8689$; 50% Salt-&-Pepper noise

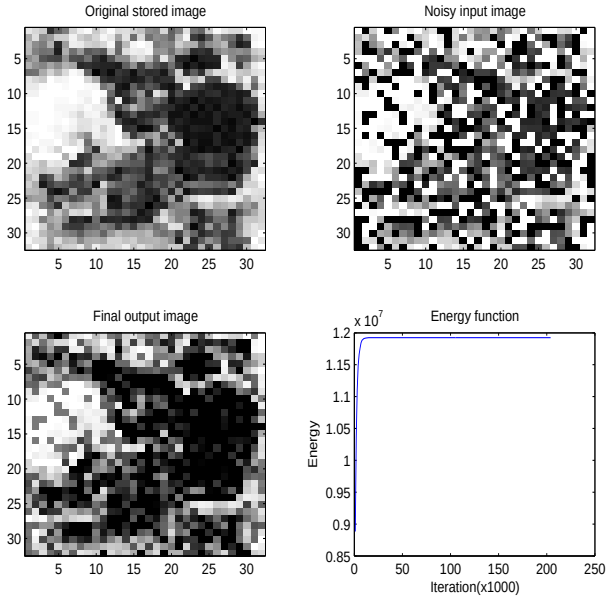

Figure 5.9: Men: $\alpha = 0.24, \beta = 0.0045, \omega = 0.6, SNR = 1.8689$; 50% Salt-&-Pepper noise

Some effort was made to determine the optimal sparsity for random patterns in a specialized network of LT neurons [214]. In determining the maximum storage capacity of an associative memory network, it is necessary to know the optimal sparsity of the memory vectors to be stored. The reason for this is mostly related to information theory; the network has a upper limit on the amount of information that it will be able to store. If each pattern was to contain more information (low sparsity), the number of patterns that can be stored and retrieved reliably decreases. In effect, a network would be able to store more sparse patterns than dense patterns. However, the difficulty in analyzing the storage capacity using conventional methods is the continuous and unbounded nature of an LT neuron. It might be interesting to derive the optimal sparsity of pattern vectors for an associative memory network of LT neurons, much like the classical Hopfield network, where it was calculated that $p = 0.138N$ [88]. A possible direction for future work might include developing better, more efficient algorithms for learning the stored patterns as well as methods to retrieve them.

5.7 Conclusion

LT networks, because of its continuous, real-valued elements, allow a greater representational scheme which in turn is more expressive than discrete, binary-state neurons (see [144] for a discussion on the relative benefits of real-valued vectors over binary vectors). While the unbounded nature of LT networks may cause instability, when properly inhibited, such networks allows more competitive dynamics between groups of neurons as the activation states of these neurons will not be driven to their saturating values.

This chapter has extensively analyzed the LT dynamics, presenting new and milder conditions for its boundedness and stability, which extended the existing results in the literature such that symmetric and nonnegative conditions for a deterministic matrix are not required to assert bounded dynamics. Based on a fully-connected single-layered recurrent network of linear threshold neurons, this chapter also proposed a design method for analog associative memory. In the design framework, with the established analytical results and the matrix theory that underly LT networks, fine-tuning of network stability margins was realized in a simple and explicit way. The simulation results illustrated that the LT network can be successful to retrieve both single stored and multiple stored real-valued images.

Measures to quantify the performance of associative memory were also suggested, and its relationship with network competition and instability was also discussed. Because the computational abilities of the LT network is a consequence of its switching between active and inactive partitions, this is consistent with the behavior of typical associative memory systems, with dense patterns result in low storage capacity. Although the performance of the associative memory capacity was largely limited to a few stored patterns of high density (low sparsity), LT networks have shown to exhibit interesting dynamics that can be further analyzed and addressed.

Chapter 6

Asynchronous Recurrent LT Networks: Solving the TSP

In this chapter, an approach to solving the classical Traveling Salesman Problem (TSP) using a recurrent network of linear threshold (LT) neurons is proposed. It maps the classical TSP onto a single-layered recurrent neural network by embedding the constraints of the problem directly into the dynamics of the network. The proposed method differs from the classical Hopfield network in the update of state dynamics as well as the use of network activation function. Furthermore, parameter settings for the proposed network are obtained using a genetic algorithm, which ensure a stable convergence of the network for different problems. Simulation results illustrate that the proposed network performs better than the classical Hopfield network for optimization.

6.1 Introduction

This chapter is primarily focused on the mapping of combinatorial optimization problems [162] onto a linear threshold (LT) network, where an approach to solving the classical Traveling Salesman Problem (TSP) using a recurrent network of LT neurons is proposed. Although LT-type neurons were first examined in Hartline and Ratliff's Nobel prize-winning work in 1958 during a study of the limulus in the anatomical make-up of the eye [83], only in recent years has interest in such

neurons been growing. The lack of an upper saturating state in neurons has been shown to exclude spurious ambiguous states by Feng and Hadeler [47], as well as to guarantee the presence of a unique stationary state. However, this non-saturating characteristic of a neuron increases the likelihood that the network dynamics may be unbounded, as well as the possibility that no equilibrium point might exist [53].

Nevertheless, further work by Feng [48] using the theory of supermartingales in developing a Lyapunov-based function for asynchronous neural networks with non-differentiable input-output characteristics, has demonstrated the existence of a convergence condition by proving that a Lyapunov function for such a network is equivalent to one that is differentiable. Hahnloser [76] subsequently analyzed the computational abilities and dynamics of linear threshold networks, particularly in explaining Winner-Take-All (WTA) multistability of the network with self-excitation and global inhibition, as well as oscillating behavior of the dynamics when the inhibition delay of the network is increased in an asymmetrical network.

Of more recent interest, Hahnloser et.al. [77] demonstrated a functional silicon-based circuit design for a linear threshold network that is able to verify the theory of digital selection (groups of active and inactive neurons) and analog amplification (active neurons are amplified in magnitude, as represented by the gain in neuronal activity). Necessary and sufficient conditions for generalized networks of linear threshold neurons, for both symmetrical and asymmetrical weight matrices were established in works by Wersing et.al. [210] and Yi et.al. [222]. Meanwhile, Xie [214] and Hahnloser [78] put forward the underlying framework for analyzing networks of linear threshold neurons by examining the system eigenvalues and classifying neurons according to permitted and forbidden sets, where subsets of permitted groups of neurons are permitted, and supersets of forbidden groups of neurons are forbidden. This is a consequence of the digital selection abilities of the network. Further analysis of the dynamics of LT neurons in a recurrent network is addressed in Tan [190] with application in analog associative memories pursued in Tang [197].

The original TSP requires that each city and hop taken is passed through exactly once. This means that there can only be a single active neuron in each row and column. Any city cannot be in more than one hop in a valid tour. Conversely, in any hop (column), only one city can be visited. The TSP can be formulated as follows: Given a finite number of n cities, together with the distance (cost) between city x and y (denoted by d_{xy}), find the optimal sequence of cities that the salesman

should visit such that the total distance taken to travel to all cities exactly once (and returning to the starting city) is the minimum. For the problem being considered here, and consistent with the focus of this chapter on symmetrical networks, the symmetrical-TSP is considered, i.e. $d_{xy} = d_{yx}$. A more general formulation is to treat the cost of traversing a pair of cities unequally, i.e. the cost of traveling from city x to y is different from that of traveling from city y to x, ($d_{xy} \neq d_{yx}$). Mathematically, this is similar to a quadratic 0-1 programming problem with linear constraints, i.e.

$$\text{Minimize} \quad \frac{1}{2}\sum_{x}^{n}\sum_{y\neq x}^{n}\sum_{i}^{n}\delta(v_{x,i})d_{xy}(v_{y,i-1}+v_{y,y+1}) \tag{6.1}$$

subject to

$$\sum_{x}^{n}\delta(v_{x,i}) = 1 \;\forall\; i \in \{1,2,\ldots,n\} \tag{6.2}$$

$$\sum_{i}^{n}\delta(v_{x,i}) = 1 \;\forall\; x \in \{1,2,\ldots,n\} \tag{6.3}$$

and a redundant constraint $\sum_x \sum_i \delta(v_{x,i}) = n$, where $\delta(v_{x,i}) = 1 \quad \forall v_{x,i} > 0$ and $\delta(v_{x,i}) = 0$ otherwise. The matrix $\mathbf{v} \in \mathbb{R}^{\ltimes\times\ltimes}$ is the solution, or assignment matrix consisting of the entries $v_{x,i} \in [0,\infty)$, which in turn represents the activation of the neuron in the xth row and ith column. An active neuron, defined as a non-negative activation in the xth row and ith column corresponds to city x being assigned to the ith hop.

(Eqn. 6.1) defines the *tour length*, the quantity which is to be minimized since it is the length of the tour a salesman would make when visiting the cities in the order specified by the permutation, returning at the end to the initial city. For n cities, this *tour* would involve n hops, where each hop is the route taken (a straight line for a *Euclidean* problem) between one city and the next. Alternatively, using nomenclature from graph theory, each hop is also known as an edge or arc, and the cities known as a vertices (singular: vertex) or nodes. The final sequence of cities is known as a *tour*. A valid tour solution satisfies the constraints of (Eqn. 6.2) and (Eqn. 6.3). A sufficiently good solution would in turn be a tour which minimizes (Eqn. 6.1). See Fig. 6.1 for a simple illustration of the weight connections between cities in a 6-city example.

For a symmetrical network, where the distance from city x to city y is equal in both directions, i.e. $d_{xy} = d_{yx}$, there are $\frac{1}{2}\frac{n!}{n} = \frac{1}{2}(n-1)!$ possible tours (a specified tour describes the order in which the n cities are visited, thus a problem with n cities would have n tours of equal length in

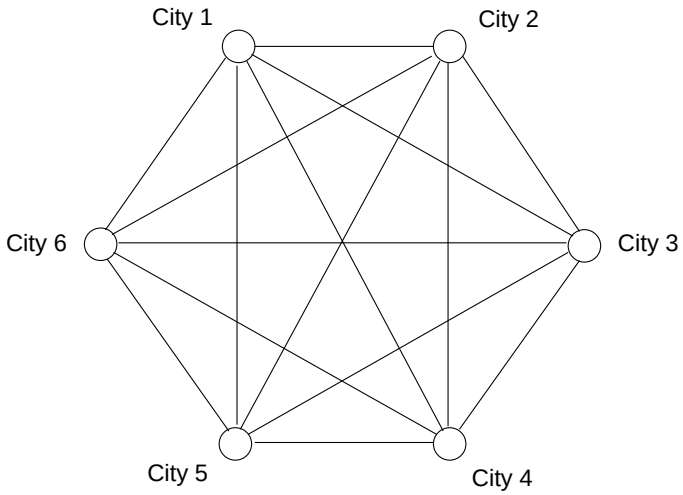

Figure 6.1: 2-D topological view of a simple 6-city TSP illustrating the connections between all $n = 6$ cities (no self-coupling or connections (diagonals of $\mathbf{W}$ are set to 0).

one direction, and another n tours of equal length in the other direction, leaving $2n$ degenerate tours). Conversely, for an asymmetrical network, ($d_{xy} \neq d_{yx}$), there are $(n-1)!$ possible tours. An optimum solution would be a tour solution of the least (minimum) length, while an arbitrarily good tour solution is one whose distance traversed is within a range of the optimum distance.

It is apparent that an exhaustive search is not possible for large n in practice. It is interesting, however, to note that near-optimal tours can be vastly different, especially for larger-sized problems. To improve on a good solution, many modifications to the problem approach need to be done, hence the intractibility of the TSP. To many, solving the TSP seems simple enough, but the key difference here is that humans perceive the problem differently, from a 2-dimensional point of view that provides a perspective that allows a rather trivial solution (which might be significantly harder for large n). Notions of distances vary according to how the problem is set-up. It is thus not necessary for distances to be defined strictly in a *Euclidean* sense. Any appropriate measures of cost such as time, economical cost, etc. are possible. Practical applications of the TSP include constraint satisfaction, packing, assignment, scheduling, and routing problems [178, 88].

The use of neural networks in solving combinatorial optimization problems have been largely inspired by Hopfield and Tank's implementation of a single-layered recurrent neural network [95, 96], which was in turn drawn from Hopfield's earlier work on content-addressable memory (CAM) [93,94]. The TSP is one of the more representative polytopes of problems in combinatorial optimization, and is often used as a test-bed for evaluating many proposed solutions and approaches for combinatorial optimization [88], making the TSP a particularly interesting optimization problem for neural networks. The iterative nature of solving such problems necessitates the use of recurrency in the network architecture, in which Hopfield and Tank's original approach was based on the use of penalty terms with an inhibitory weight matrix, to solve the mapped constrained optimization problem of the TSP. Constraints exist in the form of permissible numbers of active neurons per row and column of the solution matrix. The dynamics of the network is such that its energy function is minimized. Analogously, the energy function is equivalent to the cost or objective function of the original problem. In the TSP, the single objective is to minimize the cost represented by the total distance of the resulting tour. While the initial method proposed by Hopfield and Tank was fraught with some difficulties particularly with the number of invalid solutions found, many improvements using a variety of techniques have been used to obtain improved results.

Previous neural network methods for solving TSPs often involved the use of a network of saturating neurons with bounded activities [95]. However, it is believed that the use of a class of continuous-valued, neurons without a upper saturation state is able to exclude the problem of spurious ambiguous states in Winner-Take-All (WTA) dynamics [76, 164, 210]. Self-organizing neural networks inspired by Kohonen [116], using an elastic net approach [44] have also been employed.

In solving the TSP using a neural network approach, there is an inherent trade-off between a network that has parameters tuned to obtain very good quality solutions and one that has been tuned to obtain mostly valid solutions [128, 191]. For the interested reader, Smith (and references therein) [178] provides a fairly comprehensive overview on the evolution of neural networks in solving a variety of combinatorial optimization problems such as the TSP, while Ramamujam [162] discusses the general approach of mapping combinatorial type problems onto neural networks. As Smith has concluded, research is still continuing to improve the many approaches that have been inspired from Hopfield's seminal work [178], and further asserts that hybridization of these approaches, such as that between neural networks and genetic algorithms attempted by this chapter, could possibly yield

better results.

The organization of this chapter is as follows: Section 2 will present the proposed approach to solving the TSP using a recurrent linear threshold network as well as its corresponding state update dynamics. Optimization of the network parameters using a genetic algorithm is then described in Section 3. A comparative study of the proposed method, together with its GA-optimized parameters is subsequently drawn with the classical Hopfield network [95] in Section 4. Consequently in Section 5, a discussion on the network dynamics of the proposed approach, pertinent to the scalability and setting of the parameters, is presented. A summary, in Section 6 then concludes this chapter.

6.2 Solving TSP using a Recurrent LT Network

For an n-city problem, there are n^3 possible connections in the network; n^2 connections between neurons in a single layer, and connections between neurons between the n layers. In other words, an n-city problem has n^2 neurons with a weight matrix of n^3 connections. According to [88], the use of binary threshold neurons does not provide good results as the dynamics will rapidly converge onto a local minimum with a poor tour length; hence either stochastic units with simulated annealing, or continuous-valued neurons should be used. This can be explained in terms of granularity of binary neurons versus that of analog neurons. While binary neurons are more nonlinear in the sense that they arrive at decisions more readily (either 1 or 0 states with no values in-between), they do not represent transitionary states of the network well. Analog neurons, on the other hand are continuous-valued, hence provide more informative and thus useful values for intermediary computation.

Often, the optimization problem at hand needs to be *mapped* onto a neural network, such that the dynamics or activities of the network can interpreted as the result, or solution to the originally posed problem. This mapping process usually involves the setting up of an appropriate cost, or objective function which in turn can be represented as an energy function in the dynamics of the neural network. The solution in turn is most often a discrete answer, although the neurons perform analog computations [95]. In the original formulation of the TSP [95], negative weights (inhibitive connections) were used between neurons in the network. However, in the proposed network of LT neurons, a positive weight matrix $\mathbf{W} \in \mathbb{R}^{\aleph \times \aleph}$ (with zero entries along the diagonal) and zero inputs ($b_{x,i} = 0 \;\; \forall x, i \in n$) is used.

6.2.1 Linear Threshold (LT) Neurons

In recent years, a class of neurons which behaves in a linear manner above a threshold, known as linear threshold (LT) or threshold linear neurons, have received increased attention in the realm of neural computation. Previously, saturation points were modeled in activation functions to account for the refractory period of biological neurons when it 'recharges', and cannot be activated. Computational neuroscientists believe that artificial neurons based on an LT activation function are more appropriate for modeling actual biological neurons. This is quite a radical departure from the traditional line of thought which assumed that biological neurons operate close to their saturating points, as conventional approaches to neural networks were usually based on saturating neurons with upper and lower bounded activities. Studies have demonstrated that cortical neurons rarely operate close to their saturating points, even in the presence of strong recurrent excitation [42]. This suggests that although saturation is present, most (stable) neural activities will seldom reach these levels.

An LT neuron is a non-differentiable, unbounded activation function that behaves linearly above a certain threshold, similar in functionality to (half-wave) rectification in circuit theory. The general form of the LT activation function is defined as

$$\sigma(x) = [x]^{+} = k \times max(\theta, x) \tag{6.4}$$

where k and θ respectively denotes the *gain* (usually 1) and *threshold* (usually 0), of the activation function (see Fig. 6.2). This form of nonlinearity is also known as threshold, or rectification nonlinearity.

6.2.2 Modified Formulation with Embedded Constraints

Unlike the Hopfield model [95] where continuous, bounded neurons were used, this chapter proposes an approach using a network of LT neurons. It is believed that using LT neurons in a WTA arrangement reduces the possibility of invalid tours as there is more granularity, or resolution in the neural activities; since the neural activities are not bounded, stable network dynamics will ultimately converge to a set of active neurons with a range of analog values. Hence, the criteria for convergence can be stricter to allow for the possibility that the neural activities can assume any real nonnegative

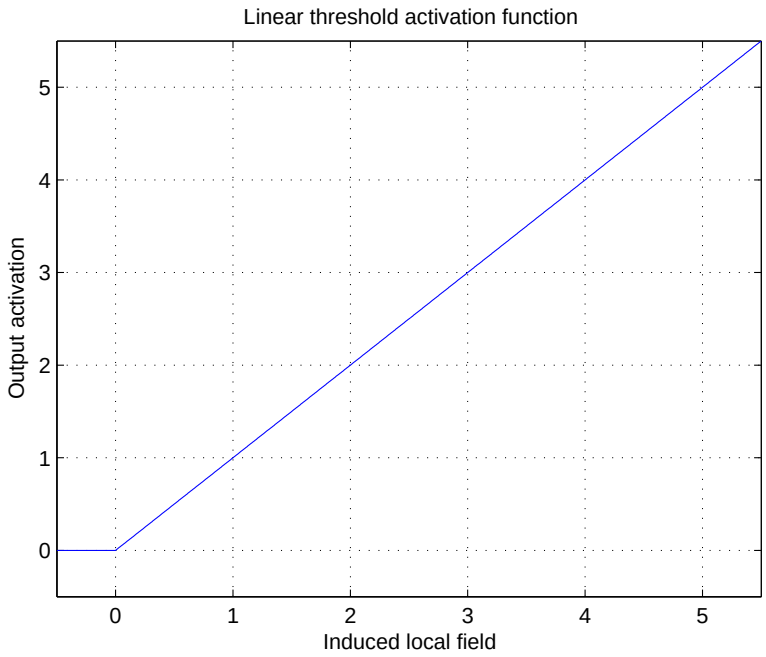

Figure 6.2: LT Activation Function with Gain $k = 1$, Threshold $\theta = 0$, relating the neural activity output to the induced local field.

value, instead of requiring that the activities of neurons be represented by the neural activity of 0 or 1. See subsection (5.4) on *Conditions for Convergence*.

Let $\mathbf{v}$ denote the set of $n \times n$ neurons indicating the tour specified by the network. For an n-city problem, $\mathbf{W} \in \mathbb{R}^{\ltimes \times \ltimes}$ and $\mathbf{v} \in \mathbb{R}^{\ltimes \times \ltimes}$; a valid tour solution would mean that exactly n neurons are active in $\mathbf{v}$, with the row and column constraints specified by Eqn. (6.2) and Eqn. (6.3). For example, a 6-city problem that has the final tour solution as shown in Figure 6.3 has the tour path as follows: $1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 3 \rightarrow 1$. Note that each column and row has exactly one active neuron.

An important stage involved in attempting to map a constrained optimization problem onto a neural network implementation is to determine appropriately, how to embed the given problem constraints into the dynamics (motion equation) of the network. Specifically, in addressing this issue for the TSP using a network of LT neurons, the implied idea is to restrict the activation of neurons along a row or column of $\mathbf{v}$ which is already active, unless conditions are favorable (e.g. the support it receives from neighboring hops is greater than the inhibition it receives from active neurons along

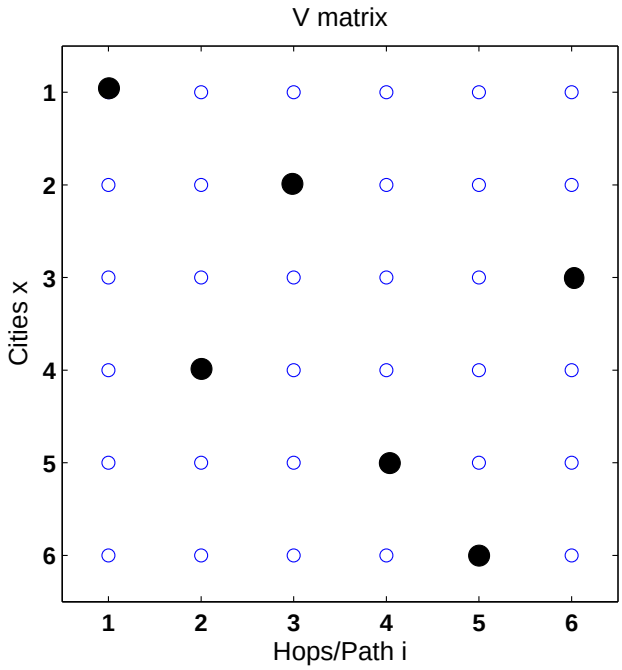

Figure 6.3: A valid tour solution for a simple 6-city TSP with a tour path of $1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 3 \rightarrow 1$.

the same row or column). The next subsection further elucidates this idea.

6.2.3 State Update Dynamics

The constraints of the problem is directly embedded into the state dynamics of the network. Let x and y denote the identities of the n cities, and i and j represent the n possible hops taken ($x,\ y,\ i,\ j \in n$). $[x]^+ = x$ if $x > \theta$ (where θ is the threshold and is usually taken to be 0) and 0 otherwise. This form of an activation function is linear threshold (or threshold linear) and is said to have rectification nonlinearity (linear behavior only occurs above the threshold). Let $\mathbf{d} \in \mathbb{R}^{\aleph \times \aleph}$ represent the square, symmetrical distance matrix between all n nodes of the network. Then the weight matrix $\mathbf{W}$ of the network is set-up as follows ($\mathbf{W}$ is a nonnegative matrix, where all off-diagonal entries are strictly positive while the components are zero along the diagonal):

$$\mathbf{W} = max(\mathbf{d}) + min(\mathbf{d}) - \mathbf{d} \tag{6.5}$$

The equation of motion of the LT network dynamics is as follows (one can assume that the time constant of the dynamics $\tau = 1$), where a neuron x (in hop/path i) randomly selected for asynchronous update. Recall that $u_{(x,i),(y,j)} \in \mathbb{R}$ (all real values) is the pre-activation, and $v_{(x,i),(y,j)} \in \mathbb{R}^{\not\vdash,+\infty)}$ (all real, non-negative values) is the activation. Also, $k_0,\ k_1,\ k_2, k_3\ \in \mathbb{R}^+$ are all real, positive values:

$$u_{x,i} = k_0\xi_0 - k_1\xi_1 - k_2\xi_2 + k_3\xi_3 \tag{6.6a}$$

$$v_{x,i} = [u_{x,i}]^+ \tag{6.6b}$$

$$\frac{du_{x,i}}{dt} = -\frac{u_{x,i}}{\tau} + v_{x,i} \tag{6.6c}$$

where

$$\xi_0 = \sum_{y\neq x}^{n} w_{xy}(v_{y,j-1} + v_{y,j+1}) \tag{6.7a}$$

$$\xi_1 = \sum_{y\neq x}^{n} v_{y,i} \tag{6.7b}$$

$$\xi_2 = \sum_{j\neq i}^{n} v_{x,j} \tag{6.7c}$$

$$\xi_3 = \Big(2 - \big(\delta(\sum_{y}^{n} v_{y,j}) + \delta(\sum_{j}^{n} v_{y,j})\big)\Big) \tag{6.7d}$$

$\delta(z)$ represents a step function where $\delta(z) = 1$ if $z > 0$ and 0 otherwise. The first term (ξ_0) represent the support a randomly selected neuron in hop i receives from neighboring hops ($i-1$ and $i+1$); the second (ξ_1) and third term (ξ_2) denote the row and columnar constraints respectively, while the fourth term (ξ_3) forces the network to have exactly n active neurons. The inhibitive terms (Eqn. 6.7.b) and (Eqn. 6.7.c) penalizes neurons in the same row or column that are active. Neurons having positive activations at convergence are then selected as the active neurons. This is collectively represented in the $\mathbf{v}$ matrix. As was previously highlighted, feasible or valid solutions require that only one neuron per row and column are active at steady-state.

In an ideal situation, we would like to avoid specifying the weighted penalty terms, and let the network attempt to solve the problem to the best of its ability without human intervention solely based on the dynamical update and states of the network itself. However, in our formulation,

these penalty terms are often derived heuristically, based on the size of the problem (it becomes substantially more difficult to determine appropriate settings of these penalty terms for larger-sized problems). In particular, a genetic algorithm is used to assist in the process of obtaining a set of penalty terms that is valid for a particular problem size. This approach is outlined in the next section. Note, however, that these penalty terms (also known as network parameters in this chapter) are not unique.

6.3 Evolving network parameters using Genetic Algorithms

Determining the appropriate values for $k_0,\ k_1,\ k_2,\ k_3$ in (Eqn. 6.6.a) is a non-trivial task. A 'pre-processing' stage using a simple genetic algorithm approach was used to obtain these values. Genetic algorithms (GAs) are stochastic search methods that simulate the process of biological evolution. The principle of "natural selection" forms the core of such algorithms, enabling it to maintain the necessary evolutionary force to make progressive improvement towards the optima. In contrast to traditional operation research and heuristical methods, GAs are capable of sampling multiple potential solutions simultaneously. Furthermore, operations such as crossover encourage the exchange of information between individuals, giving the GA a global perspective of the optimization process. Intuitively, GA is more effective in dealing with local optimal traps. In addition, the exploitation of both global and local information allows the GA to perform a much more effective optimization process.

Thus, it is not surprising that GAs have found increasing applications and interest in many practical problems as the tuning of parameters for process controllers, drug scheduling, optimization of neural network weights and design set up in the industry. In fact, GAs have been applied to different problems of neural network design such as the optimization of network parameters, network structures as well as to the training of known architectures. As was highlighted in the previous section, the network parameters of $k_0,\ k_1,\ k_2,\ k_3$ are presently determined using a trial-and-error approach. Such an approach becomes infeasible when extended to larger-size problems. With this in mind, a GA-based approach is used to determine the appropriate values of $k_0,\ k_1,\ k_2,\ k_3$ for different problems.

This section is solely meant to illustrate the suitability of the parameters that were selected (k_i $\forall i = \{1, 2, 3, 4\}$). In other words, the optimization of the parameters through the use of a simple GA is not meant to be part of the proposed approach, but rather to 'verify' the appropriateness of the parameters that were used in the simulation runs. The use of a GA in evolving candidate solutions for solving the TSP has long been a research topic in the literature, and is thus not our objective to solve the TSP using this approach.

6.3.1 Implementation Issues

In this section, a GA approach is used to determine the appropriate values of $k_0,\ k_1,\ k_2,\ k_3$, which ensures a stable convergence of the network for different problems. By stability, it is meant that the activation values of the $\mathbf{v}$ matrix do not 'explode' (recall that linear threshold activation functions is continuous and unbounded). The rationale of such a goal is intuitive since stable activation is a necessary condition before feasible solutions can be found and feasibility is in turn a necessary condition for optimality.

6.3.2 Fitness Function

Considering the implementation of GA, it should be noted that the GA processes a set of encoded parameters and not the parameters itself. Hence, it provides us with the flexibility to design an appropriate representation of the potential solutions. By *appropriate representation*, it is meant that it fulfills some criteria such as ease of implementation or exploitation of the problem structure. In this case, the parameters to be optimized are represented directly by real-number coded chromosomes. The fitness function attempts to minimize the number of epochs required to attain a stable solution. Suppose epoch is the number of epochs required by a NN to converge. Then minimizing epoch is akin to finding the best set of parameter that ensures stability. Then any reasonable solution will be able to provide a reasonable upper bound to the number of epochs required. The fitness function is given as:

$$\text{epoch} = min\{num\_epoch, (MAX\_EPOCH - state\_stop)\} \tag{6.8}$$

num_epoch is the actual number of epoch required to attain stability, while MAX_EPOCH is the predefined upper bound on the number of epochs and *state_stop* is period of stable state conditions

prior to the end of NN training. Due to the stochastic nature of neural networks, each solution is evaluated and averaged over sample times.

6.3.3 Genetic Operators

Crossover facilitates the exchange of information between selected individuals, allowing the evolutionary process to explore new solutions while retaining past information. In this chapter, simulated binary crossover [38] is implemented to produce offspring. The mutation operator employed is the normally distributed mutation, which is a popular method. In addition, the niching mechanism proposed by Goldberg [65] to prevent genetic drift and to promote the sampling of the whole Pareto front by the population is also employed.

6.3.4 Elitism

The use of the elitist strategy was first introduced by De Jong [40] to preserve the best individuals found into the next generation. The objective of this policy is to prevent the lost of good individuals due to the stochastic nature of the evolution process. De Jong found that elitism could improve the performance of GAs although there is a potential danger of premature convergence. Since then, several variants of the elitist scheme have been employed in GA and showed that elitism can indeed improve the performance of the GA greatly.

In this chapter, elitism is implemented using the evolving population and an archive. All the good solutions found from the evolutionary process are copied into the archive while previously archived solutions that are found to be inferior due to the introduction of better solutions are deleted. Binary tournament selection is then performed on a combined population of the evolving population and archive to fill the mating pool. Tournament selection is chosen because it eliminates the need to rank and sort the population.

6.3.5 Algorithm Flow

The algorithm flow of the implemented GA is summarized below. In the experiments, 10 runs are performed for the design problem so as to study the statistical performance, such as consistency and

Table 6.1: Genetic Algorithm Parameters

Parameter	Description
Pop_size	Population size
Arc_size	Archive size
$genNum$	Maximum number of generations
P_c	Crossover rate
P_m	Mutation rate

Table 6.2: Genetic Algorithm Parameter Settings

Parameter	Settings
Chromosome	Real-number representation
Populations	Population size: 20; Archive size: 10
Selection	Binary tournament selection
Crossover operator	Simulated Binary Crossover
Crossover rate	0.95
Distribution Index	20
Mutation rate	0.25
Mutation σ	0.1
Diversity operator	Niche count with radius 0.1 in the normalized objective space
Sample	5
Evaluation number	2000

robustness of the methods. Note that a random initial population is created for each of the 10 runs, and for each test problem. See Table 6.1 for a brief description of the simulation parameters. Actual parameter settings are listed in Table 6.2.

- *Step* 1: Initialization – Generate initiate population, and empty archive. Set $gen = 0$.
- *Step* 2: Fitness assignment – Calculate fitness values of individuals in $population(gen)$.
- *Step* 3: Update archive-Copy all good individuals into the archive. If current archive $size > arc(size)$, niching is employed for archive truncation.
- *Step* 4: Reinsertion – Combine archive solutions and evolving population.
- *Step* 5: Selection- Select individuals according to binary tournament selection scheme and add to mating pool.
- *Step* 6: Crossover – Select two individuals each time and perform simulated binary crossover. Add offspring to $population(gen + 1)$. Continue until mating pool is empty.
- *Step* 7: Mutation – Perform normally distributed mutation on individuals in $population(gen + 1)$ with P_m.
- *Step* 8: if $gen < genNum$, go to *Step* 2.

6.4 Simulation Results

The following results show that the performance of the proposed LT network solves the TSP more efficiently than the classical Hopfield network for optimization. A particularly interesting result is the worst, or maximum distance of the tour solutions solved using the respective approaches, where the proposed LT network produces results that are significantly better (shorter in distance). *Good* tour solutions are defined to be tours whose distances are within 150% of the optimal tour distance (this performance measure was similarly used in [195, 184]), which in the 10-city case is, $d_{optimal} = 2.58325$ units and $d_{150\%} = 3.87488$ units. For the 12-city double-circle problem, the optimum distance is $d_{optimal} = 12.3003$ and $d_{150\%} = 18.045045$. The performance ratio metric measures the consistency of the obtained solutions with respect to the minimum distance obtained, i.e. Performance Ratio $= \frac{\text{Minimum Distance}}{\text{Average Distance}}$.

6.4.1 10-City TSP

A 10-city problem was solved using the proposed LT approach, with the optimal solution of 2.58325 units shown in Figure 6.4. Empirical results obtained of the proposed LT network and the classical Hopfield network are summarized in Table 6.3. For all runs, stopping condition of $\Delta v_{x,i} = 0 \; \forall x, i \in n$ for 80 iterations or $\Delta\delta(v_{x,i}) = 0 \; \forall x, i \in n$ for 2000 iterations is used.

Random valid tours are solution tours where at each hop, the destination city is randomly selected, without taking into account the distance matrix $\mathbf{d}$. This is *pseudo-random*, as purely random tours are not considered because of the possibly large set of invalid solution tours. The use of more complex problem sets, in terms of size (n) and/or positioning of cities (see the 12-city double-circle TSP in the next subsection) would further differentiate the results obtained using a random approach (as explained above), the Hopfield network method and the proposed LT network. Throughout the simulations carried out for the 10-city TSP, the default values for the LT network parameters are $k_0 = \frac{1}{2}, k_1 = 1, k_2 = 1, k_3 = 1$. The values obtained for these parameters using a GA was found to be $k_0 = 0.6183, k_1 = 1.244, k_2 = 0.8531, k_3 = 1.3311$. Note the closeness of these values to the values that were found using a trial-and-error method. These GA-evolved parameter settings were shown to obtain solutions substantially faster. The difference between the LT and LT+GA approaches is that for the latter, the LT network parameters are optimized using the GA.

Table 6.3: Simulation Results for the 10-city TSP

Performance Metric	Random Valid Tours	Hopfield Network	Proposed LT Network	Proposed LT Network w/ GA evolved k
Average Distance	4.6534	3.8601	3.2608	3.3955
Maximum Distance	5.6504	5.2216	4.0468	4.1489
Minimum Distance	3.3153	2.8978	2.5832	2.6018
Valid Tours	100	100	100	100
Invalid Tours	n/a	6	1	0
Good Tours	6	64	95	91
Performance Ratio	0.7124	0.7507	0.7979	0.7662
Time taken (secs.)	6.219	54.375	156.078	82.531

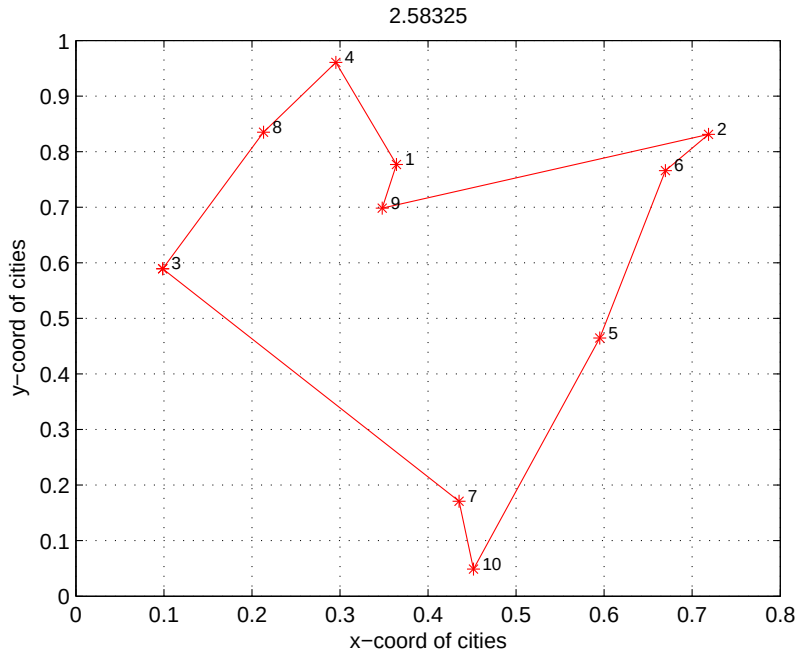

Figure 6.4: Optimal solution for the 10-city TSP (2.58325 units).

Histograms of the distribution of tour distances, for the three cases: the LT network approach, the random and the Hopfield methods are shown in Figures 6.7, 6.9 and 6.10 respectively, illustrating the noticeable spread of tour distances using the various approaches.

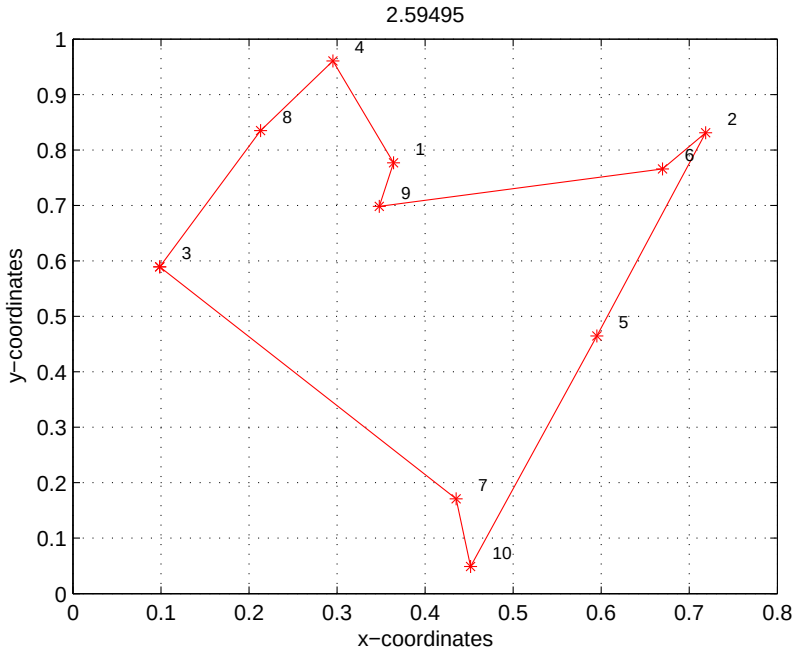

Figure 6.5: A near-optimal solution for the 10-city TSP (found using the proposed LT network with parameters found using a trial-and-error approach).

Figure 6.11 shows a comparison of the total distance resulting from the different methods used to solve the 10-city TSP (random, Hopfield, LT, LT+GA). The distribution of the results is represented in boxplot format to visualize the distribution of the simulated results efficiently (the results being the distances of the solutions found by the different methods). The vertical axis represents the distances while the horizontal axis denotes the solution approach used. Each box plot represents the distribution of the 100 results (runs), where a thick horizontal line within the box encodes the median while the upper and lower ends denote the upper and lower quartile respectively. Dashed appendages illustrate the spread and shape of distribution and dots represent outside values. This boxplot clearly illustrates the improvement in the results obtained for the 10-city TSP using the proposed LT and LT+GA approach. In this particular example, the results obtained using the LT+GA method is similar to the results obtained using the LT approach, the only improvement being the shorter time required for obtaining solutions using the GA optimized parameters in the LT+GA method (see Table 6.3).

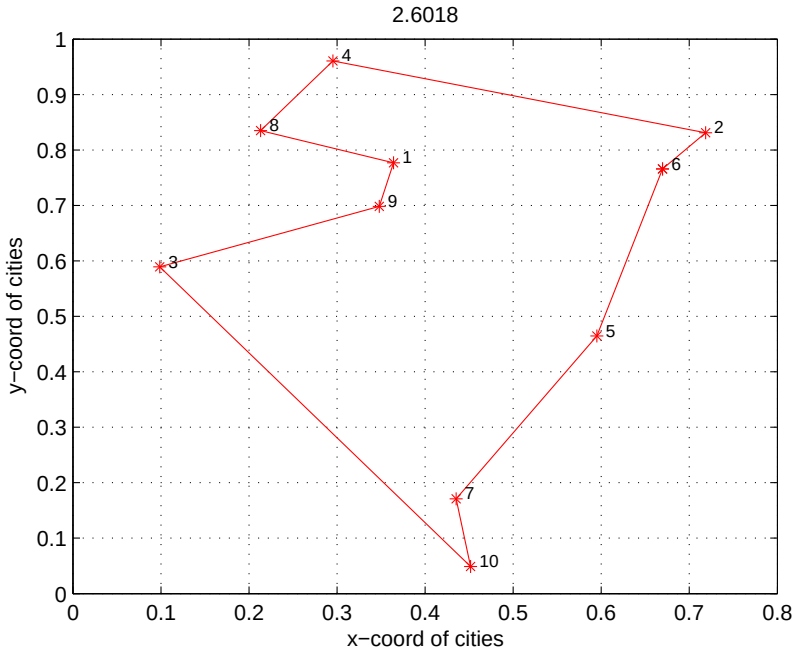

Figure 6.6: A near-optimal solution for the 10-city TSP (found using the proposed LT network with GA-evolved parameters).

6.4.2 12-City Double-Circle TSP

The solutions to a double-circle TSP, though perceptually is easy to solve, is very difficult for the original Hopfield network to solve. For this case, 12 cities were arranged in a circle such that there are 6 cities on the inside circle and another 6 cities located just on the outside (see Figure 6.12 for the optimal tour solution). Throughout the simulations carried out for the 12-city double-circle TSP, the default values for the LT network parameters are $k_0 = \frac{1}{2}, k_1 = \frac{3}{2}, k_2 = \frac{3}{2}, k_3 = 1$. The values obtained for these parameters using a GA was found to be $k_0 = 0.8555, k_1 = 1.5002, k_2 = 1.3072, k_3 = 1.6078$. Again, note the closeness of these values to the values that were found using a trial-and-error method. Recall that the difference between the LT and LT+GA approaches is that for the latter, the LT network parameters are optimized using the GA. Similar to the 10-city problem, these GA-evolved parameter settings were shown to obtain solutions substantially faster (and in this case, slightly better results were obtained). The weight matrix $\mathbf{W}$ is also scaled by a factor of $\frac{1}{4}$ to ensure that the network dynamics is stable, since unlike the previous 10-city TSP, the positions (x, y-coordinates) of the cities are not normalized within the unit box $[0, 1]$. The definitions for the various performance

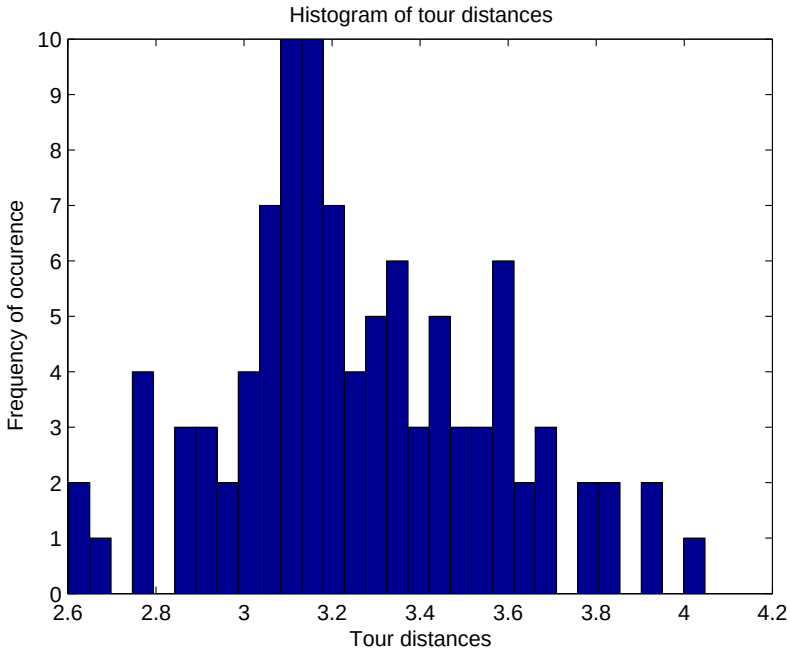

Figure 6.7: Histogram of tour distances for the 10-city TSP from the proposed LT network.

measures are the same as for the 10-city TSP explained in the previous section. Empirical results obtained of the proposed LT network and the classical Hopfield network are summarized in Table 6.4. For all runs, stopping condition of $\Delta v_{x,i} = 0 \; \forall x, i \in n$ for 80 iterations or $\Delta\delta(v_{x,i}) = 0 \; \forall x, i \in n$ for 2000 iterations is used.

Histograms of the distribution of tour distances, for the three cases: the LT network approach, the random and Hopfield methods are shown in Figures 6.13, 6.15 and 6.16 respectively, again illustrating the noticeable spread of tour distances using the various approaches. Figure 6.14 shows the noticeable improvement obtained from using the parameters obtained from the GA.

Figure 6.17 shows a comparison of the total distance resulting from the different methods used to solve the 12-city double-circle TSP (random, Hopfield, LT, LT+GA). This boxplot clearly illustrates the improvement in the results obtained for the 12-city double-circle TSP using the proposed LT and LT+GA approach. In this case, the results obtained using the LT+GA method is significantly better than the results obtained using the LT approach, in all measured performance metrics (see Table 6.4).

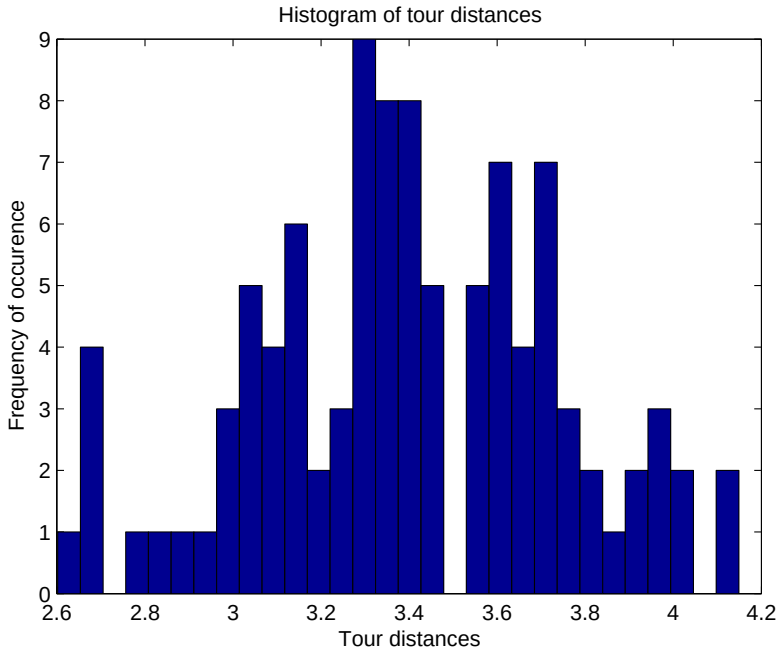

Figure 6.8: Histogram of tour distances for the 10-city TSP from the proposed LT network with GA-evolved parameters.

6.5 Discussion

6.5.1 Energy Function

The objective of any mapped optimization problem onto a neural network is to minimize the computational energy function of the network. The energy function not only determines the number of neurons used in the system but also the strength of the synaptic connections (weights) between these neurons. Such an energy function is constructed based on the constraints and the cost function of the original problem. For the TSP, the constraints are specified by (Eqn. 6.2) and (Eqn. 6.3), while the cost function is given by (Eqn. 6.1). The change in the (input) state of a neuron ($u_{x,i}$) is given by the partial derivatives of the computational energy function E with respect to the (output) activity of the particular neuron ($v_{x,i}$) in question. Observe that E is a $n \times n$-variable function of the $n \times n$ neurons present in the system. The motion equation of the xth neuron in the ith hop is

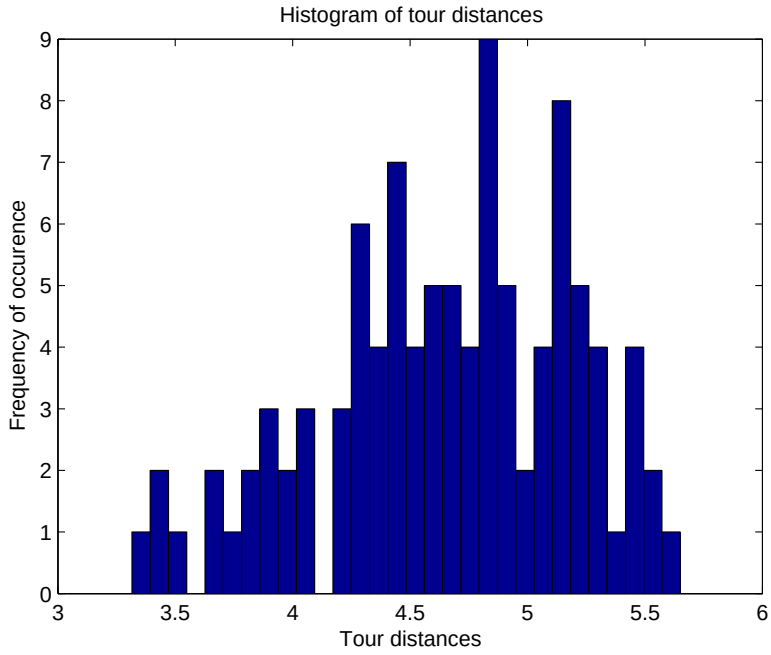

Figure 6.9: Histogram of tour distances for the 10-city TSP from the random case.

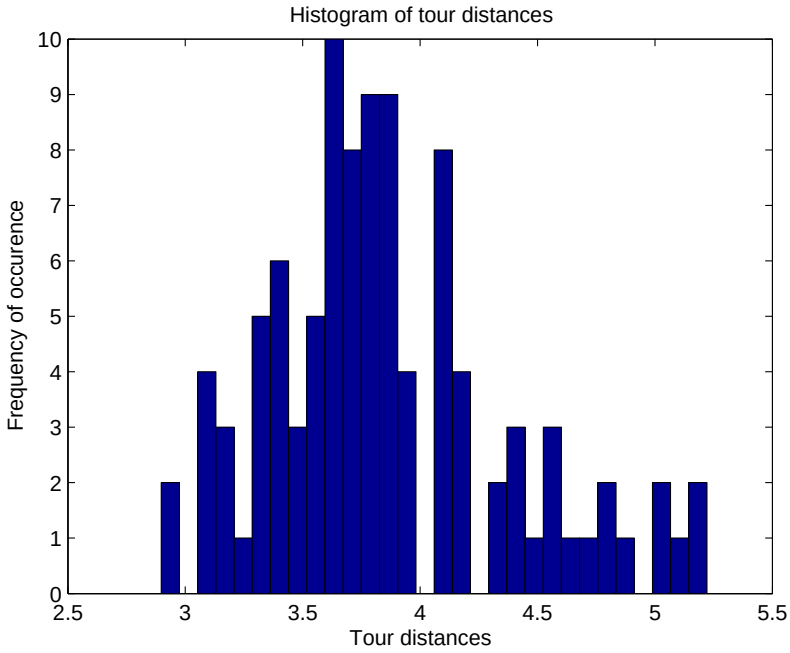

Figure 6.10: Histogram of tour distances for the 10-city TSP from the Hopfield case.

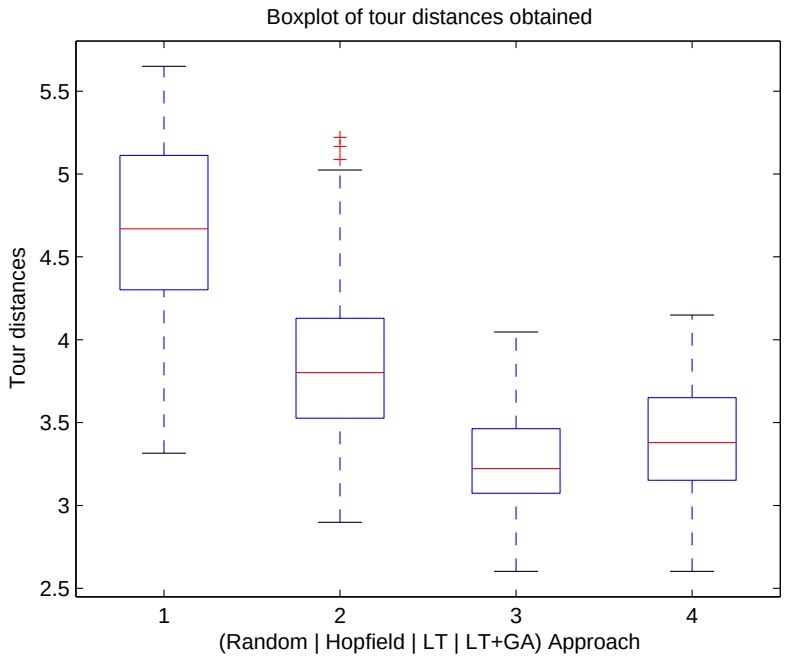

Figure 6.11: Boxplot of the tour distances obtained for the 10-city TSP, comparing the (1) Random, (2) Hopfield, (3) Proposed LT, (4) Proposed LT + GA approaches.

given by

$$\frac{du_{x,i}}{dt} = -u_{x,i} + \left[-\frac{\partial E}{\partial v_{x,i}} + u_{x,i} \right]^{+} \tag{6.9}$$

The total energy of the system, denoted by E, can be separated into four terms,

$$E = E_0 + E_1 + E_2 + E_3 \tag{6.10}$$

Table 6.4: Simulation Results for the 12-city double-circle TSP

Performance Metric	Random Valid Tours	Hopfield Network	Proposed LT Network	Proposed LT Network w/ GA evolved k
Average Distance	31.6149	22.9828	18.6660	16.1568
Maximum Distance	41.7054	32.1454	24.4208	25.4873
Minimum Distance	21.7492	12.3040	12.3040	12.3003
Valid Tours	100	100	100	100
Invalid Tours	n/a	6	5	6
Good Tours	0	13	42	74
Performance Ratio	0.6879	0.5354	0.6386	0.7613
Time taken (secs.)	10.906	47.062	770.406	95.094

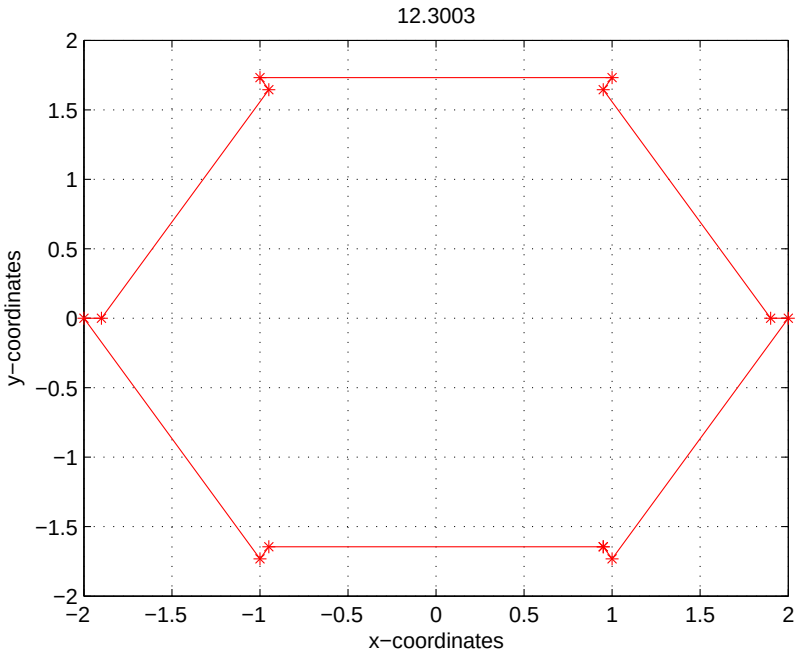

Figure 6.12: Optimal solution for the 12-city double-circle TSP (12.3003 units).

Individually, these computational energy terms correspond to

$$E_0 = -\frac{k_0}{2}\sum_{x}^{n}\sum_{i}^{n}\sum_{y\neq x}^{n} w_{xy}v_{x,i}(v_{y,i-1}+v_{y,i+1}) \tag{6.11a}$$

$$E_1 = \frac{k_1}{2}\sum_{x}^{n}\sum_{i}^{n} v_{x,i}\sum_{j\neq i}^{n} v_{x,j} \tag{6.11b}$$

$$E_2 = \frac{k_2}{2}\sum_{x}^{n}\sum_{i}^{n} v_{x,i}\sum_{y\neq x}^{n} v_{y,i} \tag{6.11c}$$

$$\frac{k_3}{2}\sum_{x}^{n}\sum_{i}^{n}\left(\sum_{j}^{n} v_{x,j} - \sum_{y}^{n} v_{y,i}\right)$$

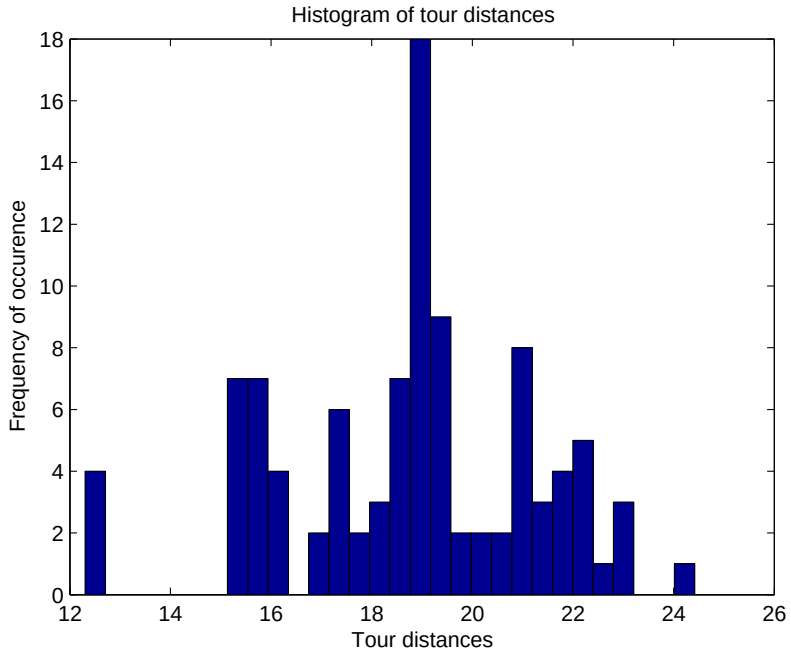

Figure 6.13: Histogram of tour distances for the 12-city double-circle TSP from the proposed LT network.

The first term (E_0) is always nonpositive for a valid tour solution since $v_{x,i} \forall x, i \in n$ is nonnegative and w_{xy} is positive (with zero diagonals). E_1 and E_2 is, in a similar manner, nonnegative following the same argument that $v_{x,i} \forall x, i \in n$ is nonnegative. The last term, E_3 is zero if exactly n neurons are active in the whole network. The inclusion of such terms in the energy function gives rise to dynamics that is inclined towards (i) valid tour solutions, and (ii) tours with short distances.

$$\frac{\partial E}{\partial v_{x,i}} = \frac{\partial E_0}{\partial v_{x,i}} + \frac{\partial E_1}{\partial v_{x,i}} + \frac{\partial E_2}{\partial v_{x,i}} + \frac{\partial E_3}{\partial v_{x,i}} \tag{6.12}$$

$$\frac{\partial E}{\partial v_{x,i}} = k_0\xi_1 + k_1\xi_2 + k_2\xi_3 + k_0\xi_3 \tag{6.13}$$

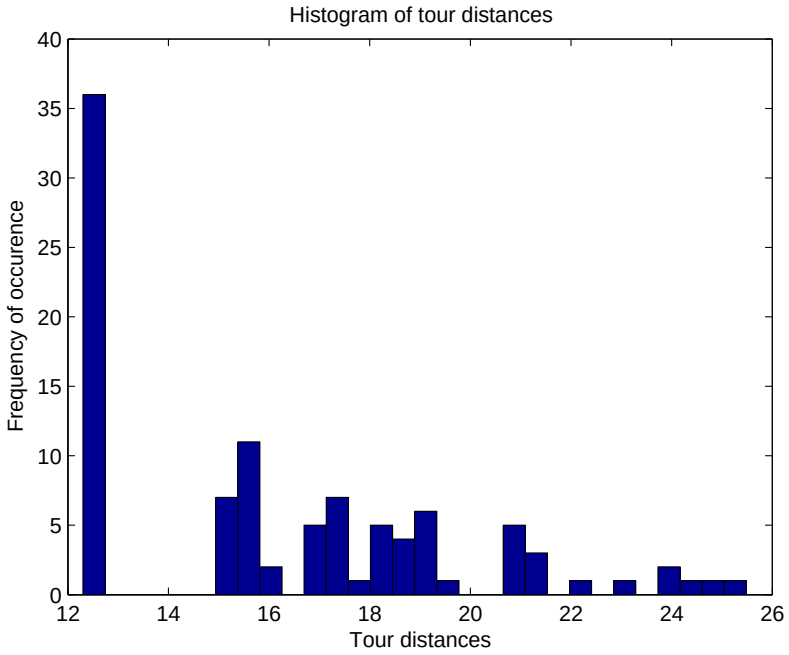

Figure 6.14: Histogram of tour distances for the 12-city double-circle TSP from the proposed LT network with GA-evolved parameters.

More specifically,

$$\frac{\partial E_0}{\partial v_{x,i}} = -\frac{k_0}{2}\sum_{y\neq x}^{n} w_{xy}(v_{y,i-1} + v_{y,i+1}) \tag{6.14a}$$

$$\frac{\partial E_1}{\partial v_{x,i}} = k_1\sum_{j\neq i}^{n} v_{x,j} \tag{6.14b}$$

$$\frac{\partial E_2}{\partial v_{x,i}} = k_2\sum_{y\neq x}^{n} v_{y,j} \tag{6.14c}$$

$$\frac{\partial E_3}{\partial v_{x,i}} = -k_3\Big((2 - u(\sum_{y\neq x}^{n} v_{y,i}) - u(\sum_{j\neq i}^{n} v_{x,j})\Big) \tag{6.14d}$$

Because (Eqn. 6.14.b) is always a nonpositive term, and (Eqn. 6.14.e) is zero when exactly n neurons are active, while Eqns. (6.14.c, 6.14.d) are always nonpositive, insofar that (Eqn. 6.14.b) is stable (non-divergent), one can expect the total energy of the system to be likewise stable.

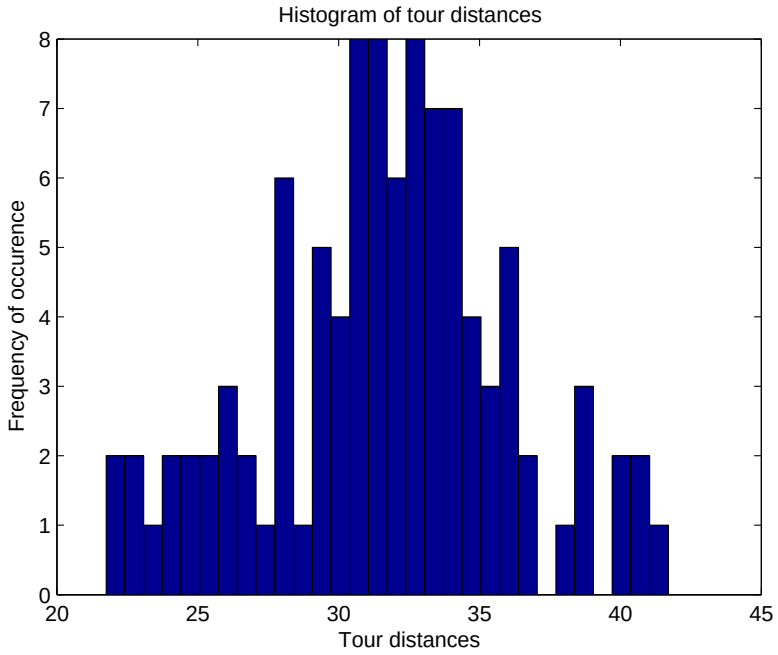

Figure 6.15: Histogram of tour distances for the 12-city double-circle TSP from the random case.

The evolution of the energy dynamics is represented as,

$$\frac{d^{\dagger}E}{dt} = \sum_i \sum_j \frac{\partial E}{\partial v_{ij}} \frac{\partial v_{ij}}{\partial u_{ij}} \frac{\partial u_{ij}}{\partial t} \tag{6.15}$$

The LT function results in a non-negative, monotonically increasing function (see Fig. 6.2). The dagger superscript (†) indicates the derivative of a discontinuous function (recall that the linear threshold function has a discontinuity at its threshold). This gives $\frac{\partial v_{ij}}{\partial u_{ij}} \geq 0$, let $\frac{\partial v_{ij}}{\partial u_{ij}} = a$, where a is some arbitrary non-negative value.

$$\frac{d^{\dagger}E}{dt} = \sum_i \sum_j \left(\frac{\partial E}{\partial v_{ij}}\right)\left(\frac{\partial u_{ij}}{\partial t}\right) a \tag{6.16}$$

$$\frac{d^{\dagger}E}{dt} = \sum_i \sum_j \left(\frac{\partial E}{\partial v_{ij}}\right)\left(-u_{ij} + \left[-\frac{\partial E}{\partial v_{ij}} + u_{ij}\right]^{+}\right) a \tag{6.17}$$

$$\tag{6.18}$$

Now, two cases are possible, depending on whether the LT function (in square brackets) results

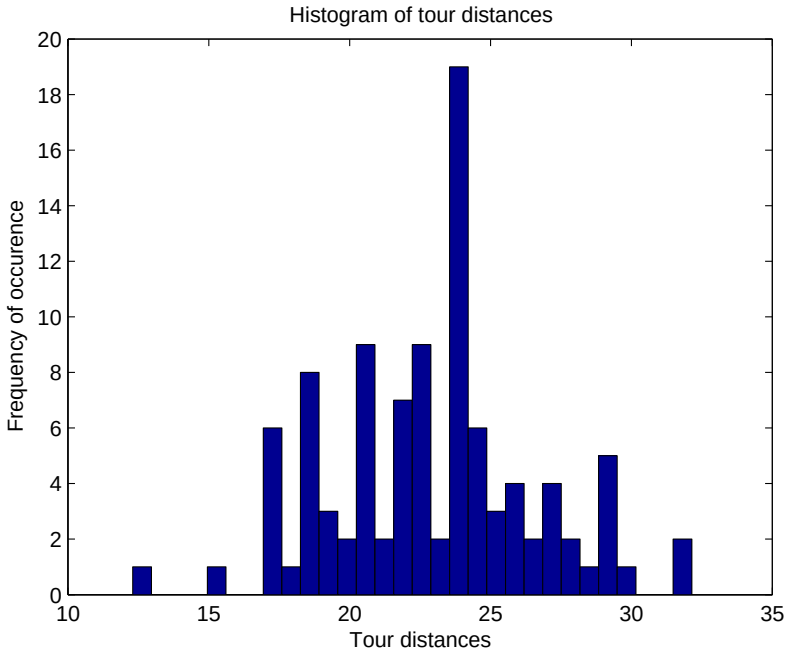

Figure 6.16: Histogram of tour distances for the 12-city double-circle TSP from the Hopfield case.

in a positive activation, or because of its rectification, thresholds the activation at zero.

$$u_{ij} > \frac{\partial E}{\partial v_{ij}} : \quad \frac{\partial u_{ij}}{\partial t} = -\frac{\partial E}{\partial v_{ij}} \Rightarrow \frac{dE}{dt} = -\sum_i \sum_j \left(\frac{\partial E}{\partial v_{ij}}\right)^2 a \tag{6.19}$$

$$u_{ij} \leq \frac{\partial E}{\partial v_{ij}} : \quad \frac{\partial u_{ij}}{\partial t} = -u_{ij} \Rightarrow \frac{dE}{dt} = -\sum_i \sum_j \left(\frac{\partial E}{\partial v_{ij}}\right) u_{ij} a \tag{6.20}$$

In the first case, when the activation is positive, results in $\frac{dE}{dt} \leq 0$. However, for the second case, when the activation is thresholded at zero, $\frac{dE}{dt}$ can be either positive or negative; this in turn depends on the value of the pre-activation u_{ij}. Since u_{ij} obeys the state update equation (Eqn. 6.15), u_{ij} will always tend to a positive value after some time, allowing the assumption that u_{ij} is non-negative. Because $u_{ij} \leq \frac{\partial E}{\partial v_{ij}}$, $\frac{\partial E}{\partial v_{ij}} \geq 0$ for $u_{ij} \geq 0$. Then $\frac{\partial E}{\partial v_{ij}} \geq 0$. Thus, $\frac{d^\dagger E}{dt} \leq 0$. Essentially, this means that as u_{ij} tends to a non-negative value, the energy of the system remains stable. While $\frac{dE}{dt}$ initially increases (this can be seen as escaping a local minimum), $\frac{dE}{dt}$ will eventually tend to zero for appropriate settings of $k_0,\ k_1,\ k_2,\ k_3$. Typically, $k_0, k_3 \gg k_1, k_2$ is chosen such that $E(> 0)$ is bounded from below. This has been empirically verified from the simulation results obtained.

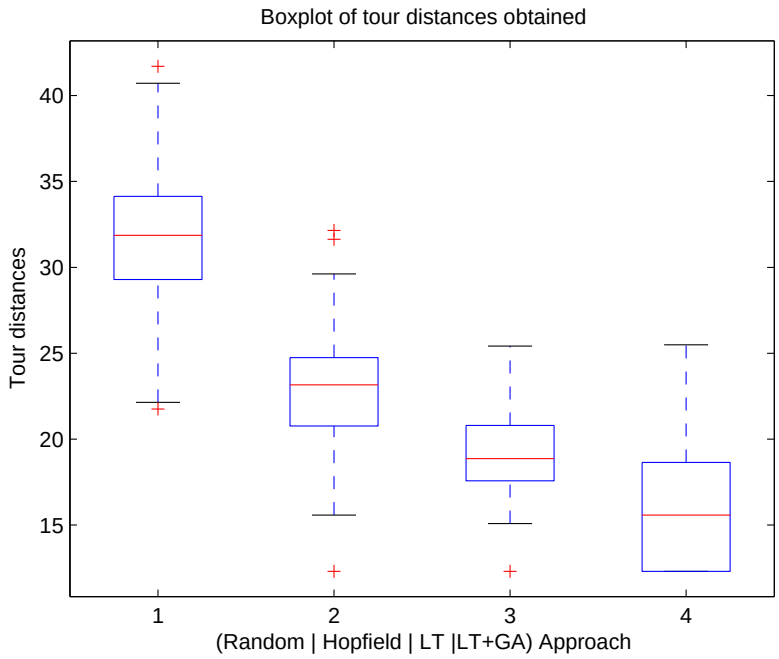

Figure 6.17: Boxplot of the tour distances for the 12-city double-circle TSP obtained, comparing the (1) Random, (2) Hopfield, (3) Proposed LT, (4) Proposed LT + GA approaches.

However, to guarantee boundedness of E such that it does not tend to $-\infty$, one needs to restrict the *support* term ($\xi_0 = \sum_{y \neq x}^{n} w_{xy}(v_{y,j-1} + v_{y,j+1})$) of the dynamics. This is turn is dependent on the values of the weight matrix $\mathbf{W}$, which is a matrix with positive entries and zero diagonals. Sufficient conditions for stability of $\frac{dE_0}{dt}$ is based upon the original requirements for stability in a linear threshold network, that is $\sum_{y}^{n} w_{xy} < 1$. This is thus discussed subsequently. This then leads to us to consider conditions for which the dynamics of (Eqn. 6.14.a) will be stable. To be more specific, one needs to consider the conditions for which w_{xy} in (Eqn. 6.14.b) will result in a stable value of ξ_0. Such sufficient conditions can be derived from

$$w_{xx} + \sum_{y \neq x}^{n} w_{xy}^{+} < 1 \tag{6.21}$$

But since $w_{xx} = 0$, and $w_{xy} > 0, \forall x, y \in n$, this condition is reduced to

$$\sum_{y \neq x}^{n} w_{xy} < 1 \tag{6.22}$$

In other words, the network dynamics of the proposed approach can be guaranteed to be stable (but not necessarily result in valid tour solutions) if the weight matrix **W** derived from the distances between the cities are *normalized*. Hence when different scales (feet vs. meters) or magnitude of measure (1 meter vs. 1 kilometer) are used in calculating the distances between the cities prior to creating the weight matrix **W**, attention needs to be given to ensure that **W** does not give rise to unstable dynamics. It is surmised that normalization would aid in scalability of the problem particularly in determining appropriate values of the network parameters. It is not the absolute values but the relative values of these distances that are important. A simple method of synaptic weight normalization is to divide the individual weight entries in **W** with the sum of the weights leading from, or to a particular neuron. Mathematically, this corresponds to

$$w'_{xy} = \frac{w_{xy}}{\sum_{y \neq x}^{n} w_{xy}} \tag{6.23}$$

A caveat to this point is that not all the entries in **W** contribute to the stability (instability) of the network dynamics. Only the components of **W** that correspond to active neurons in **v** at any time step are significant to the dynamics. For example if a candidate tour solution at a time step for a 5-city TSP is $1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 1$, this means neurons in **v** that correspond to $v_{1,1}, v_{3,2}, v_{2,3}, v_{5,4}, v_{4,5}$ are active (of which the non-zero magnitudes are unknown but positive). Hence, only the entries of **W** that correspond to (x, y) values of $(1, 3), (3, 2), (2, 5), (5, 4), (4, 1)$ as well as $(3, 1), (2, 3), (5, 2), (4, 5), (1, 4)$, and because of symmetry, contribute to the stability (instability) of the network dynamics. Thus, to further extend (Eqn. 6.22),

$$\sum_{y \neq x}^{n} \sigma(x, y) w_{xy} < 1 \tag{6.24}$$

where $\sigma(x, y) = 1$ if a path exists between city x and y and $\sigma(x, y) = 0$ otherwise. The issue of stability and convergence will be a focal point for future work.

6.5.2 Constraints

The problem constraints, mapped onto the recurrent LT network manifests itself as constraints upon the dynamics of the network, usually in the form of inhibition. These embedded constraints in the

proposed dynamics pose an interesting problem. A tradeoff exists between the selection of a small inhibitory term with that of a large inhibitory term. If the constraints, mapped as an inhibitory term in the neural network implementation are excessively large, an optimum solution is less likely to be found, as previously activated neurons are more likely to inhibit the activation of later neurons. Intuitively, this means that the search space, or candidate sets of neurons is limited and that the possible candidates are less diverse. Conversely, too small an inhibition would result in the network perpetually switching between active and inactive neurons, since the support a neuron receives from previously activated neurons may be greater than the inhibition it receives from active neurons in the same row or column. A corollary of this is that convergence of the network dynamics will be much slower as neuron activity is less likely to settle to a steady-state. Invalid tour solutions would then result.

6.5.3 Network Parameters

In the proposed LT approach, the 4 network parameters (k_0, k_1, k_2, k_3) can be modified to alter the relative contribution of each term ($\xi_0, \xi_1, \xi_2, \xi_3$ respectively) to the state update dynamics of the LT network. These parameters are critical in determining the stability, feasibility and optimality of the network dynamics, particularly because the architecture is based on a recurrent structure, which, together with the use of unbounded LT activation function can easily lead to unstable dynamics.

Under most circumstances, k_0 is modified when the distances of the cities are not normalized (i.e. the Cartesian (x, y) coordinates of the cities are not within the unit box with range $[0, 1]$. This is to prevent the divergence of network activity since ξ_0 is always nonpositive. k_1 and k_2 controls the magnitude of inhibition of the rows and columns of neurons. We believe that prior normalization of the weight matrix $\mathbf{W}$ would aid in scalability of the problem particularly in determining appropriate values of the network parameters. As mentioned in the previous sections, a greater inhibitory value restricts the search space and the number of possible candidate solutions in the network dynamics. Lastly, k_3 accounts for the magnitude of the penalty term whenever the sum of active neurons is less than n (positive penalty) or more than n (negative penalty).

These parameter values (k_0, k_1, k_2, k_3) influences the size of the search space that is being considered by the dynamics of the LT network. Too strict values will lead to a small search space

being considered and is heavily dependent on the initial conditions of the network due to premature convergence, while values which are too relaxed will often lead to non-convergence of the dynamics. Proper setting of these values (the use of stochastic optimization approaches such as GA, as was outlined in Section 6.3) will enable better results to be obtained.

6.5.4 Conditions for Convergence

In a similar manner, the conditions for convergence are also of considerable significance. A requirement that is too strict would lead to long convergence times, while on the other hand, convergence conditions that are not sufficiently strong would result in solutions that are not particularly good (longer distances are obtained), although the corresponding convergence times might be favorable. Worse yet, invalid tour solutions occur more frequently when weaker conditions for stopping are used. This can be interpreted as the search space being stuck in a local minimum.

It is for this reason that analog neurons are used instead of binary neurons, as the lack of granularity in a binary threshold neuron does not provide a good basis for intermediary computation (since only one of two states are stored). The finer resolution in an analog neuron in this sense allows greater versatility in calculation of transitory state values prior to convergence. The use of an LT neuron, which is not only analog-valued but also non-saturating, allows a greater range of values to be represented, without having to be overly concerned with the possibility of intermediate state values saturating. As long as the LT network is stable (using the previously mentioned sufficient conditions), the dynamics are guaranteed to converge.

In the simulations that were carried out, unless otherwise specified, the condition requirements were that there is no change in the energy function (see next section) for successive iterations, i.e. $\Delta E = 0$ for at least k time steps. Alternatively, other stopping conditions can be one that requires that there be no change in either (i) the analog values of the neurons (i.e. the activations of the neurons remain steady or constant), or (ii) the states of the neurons (i.e. active neurons remain active, inactive neurons remain likewise inactive).

Condition (i) is equivalent to requiring that $\Delta v_{x,i} = 0 \ \ \forall x, i \in n \Rightarrow \sum_x^n \sum_i^n (v_{x,i}^{new} - v_{x,i}^{old})^2 = 0$ or $\sum_x^n \sum_i^n |v_{x,i}^{new} - v_{x,i}^{old}| = 0$. Condition (ii), on the other hand, is a less strict criteria, only requiring that the set of active neurons remain active, and the set of inactive neurons remain inactive for a

certain number (k) of time steps, i.e. $\sum_x^n \sum_i^n (\delta(v_{x,i}^{new}) - \delta(v_{x,i}^{old})) = 0$ for (k) time steps. Again, $\delta(z) = 1$ for $z > 0$ and $\delta(z) = 0$ otherwise.

Figure 6.18 shows a typical Pareto front for the solutions (tour distances), as a trade-off between convergence time (a strict stopping criteria takes a longer time to converge; conversely, a more relaxed stopping criteria would converge more rapidly) and the tour distances obtained. The Pareto front in this case is based on the 10-city TSP where the optimal tour distance is 2.59 units. The solutions are represented by the circles lying on a curved line (since two competing objectives are being examined here). In an ideal situation, both these objectives are minimized, that is the tour distance that is obtained, as well as the time required to arrive at the solution. A particular solution is a member of the Pareto front if there are no other solutions that are better than this solution both in tour distance and time taken. These are known as the *non-dominated* solutions. On the other hand, solutions that do not lie on this Pareto front (not shown) are the *dominated* solutions.

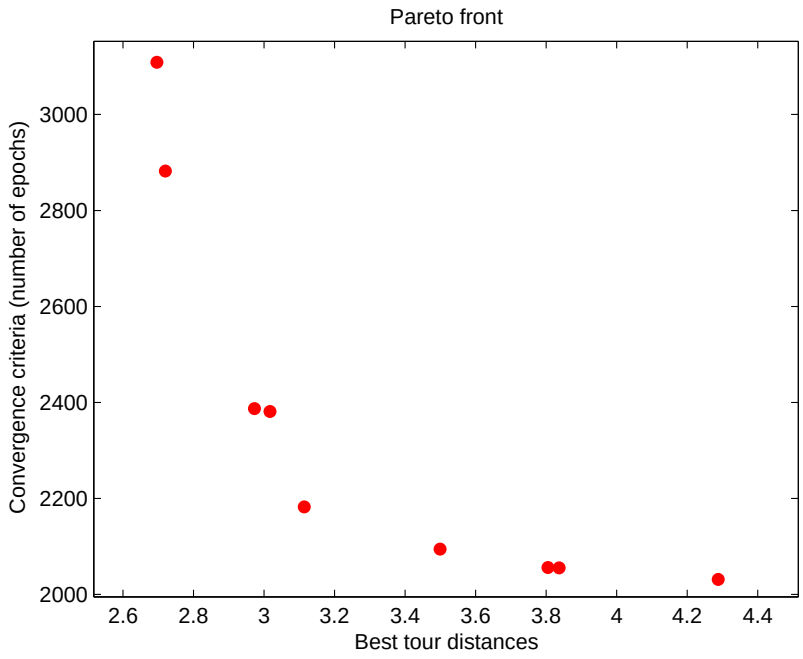

Figure 6.18: Pareto front illustrating the tradeoff between a stricter convergence criteria and the tour distance produced by the LT network.

6.5.5 Open Problems

In combinatorial optimization, or for any problems in general optimization theory, linear threshold networks are an interesting possibility particularly because of its non-saturating activation function which allows greater competition and hence selectivity to be achieved as compared to archetypical classes of neurons with discrete or bounded activation functions. The study into identifying appropriate parameters when mapping combinatorial optimization problems onto neural networks of linear threshold neurons warrants further investigation. Better insight into the role played by these parameters would be expected to result in better quality solutions. The determination of the network parameters, can be analogously compared to as determining a balance between the size of the search space and restricting the search within the confines of the problem constraints. The use of a GA that was used here provide a more appropriate setting of the network parameters ($k_0,\ k_1,\ k_2,\ k_3$) that was shown to obtain better results. This issue of stability and convergence of the energy dynamics will be a focal point for future work. Aside from the above parameters, the solution approach also involves the prior determination of various settings, such as convergence conditions, network weights; all of which are less explicit but nonetheless critical in the set of solutions that are obtained using the proposed approach. A drawback however, as is typical of such approaches to solving similiarly posed problems is the increased computational complexity ($O(n^2)$) as the size of the problem (n cities) increase. The manner in which we define the convergence criteria also increases the computational requirements of the proposed approach.

6.6 Conclusion

This chapter has presented an alternative formulation to solving the TSP, where a recurrent network of non-saturating linear threshold (LT) neurons has been used, instead of the classical Hopfield model. Results that were obtained from utilizing the proposed network of linear threshold neurons using asynchronous update dynamics consistently produced better results than its Hopfield equivalent in solving certain instances of the *Traveling Salesman Problem (TSP)*. Particularly important is that the worst (maximum) distances obtained for the TSP using the proposed LT network is significantly better (lesser in distance) than the Hopfield network (see Tables 6.3 and 6.4). The improved results are believed to be attributed to the ability of the LT neurons to avoid saturating

levels, characteristic of many archetypical neuronal activating functions. Larger-size problems were much slower to converge because of the difficulty in determining the appropriate parameters, which in turn is due to the exponentially greater number of iterations and candidate solutions, and hence were not quantitatively pursued. However, the dynamics for higher dimensional problems exhibit qualitatively similar dynamics.

Clearly, heuristical and mathematical approaches using *a priori* information would give rise to better performance and scalability. Notwithstanding, the objective of this chapter is not to demonstrate improved results for the TSP, but rather, to put forward the viability of using neurons with non-saturating properties, specifically of the linear threshold (LT) type, as a candidate activation function for solving such combinatorial optimization problems using Hopfield-like recurrent networks. We believe that further fine-tuning of the architecture and parameter settings would increase the practicability of such networks in solving a more general class of optimization problems.

Chapter 7

Conclusion

Neural networks refer to 'simplified' mathematical models of how the human brain functions in tasks such as perception, computation and memory – in some ways, modeling systems using neural networks and their variants facilitates the investigation of how information processing occurs in the animal brain from a mathematical and computational perspective, where the complexity and capabilities of such networks rely on our present understanding and knowledge of similar biological neural systems.

7.1 Contributions and Summary of Work

This work attempts a two-fold contribution: firstly, towards the design and learning of feedforward neural networks based on the singular value decomposition (SVD), and secondly, towards the use of linear threshold (LT) neurons in recurrent neural networks. In particular, the first part of this thesis centered largely about feedforward neural architectures, first adopting a theoretical stance before taking on a more application-oriented perspective.

- Chapter 2 lays the foundation for the first part of this dissertation by examining the singular value decomposition as an operator and a measure to estimate the appropriate number of hidden layer neurons in a single hidden layer feedforward neural network. The motivation for this chapter stems from the observation of singular values as an indicator of 'stability' of a

system, which in this context allows a simple quantification of the degree of degeneracy, or equivalently, the level of linear independency of the training patterns in hidden layer space. Becase the weights learnt during the training stage of the SLFN projects the features from an input space onto a hidden layer space, increasing the number of hidden layer neurons increases the capacity and thus the complexity of the system on the *training set*, the generalizability of the resulting SLFN is compromised as over-fitting occurs.

The main contribution of this chapter was to present a practical and simple framework in the use of singular values as well as their corresponding sensitivities of the hidden layer output activation matrix H, in providing an indication of the necessary or appropriate number of hidden layer neurons for a given problem set – where the use of the rank-revealing SVD allows us to gain a better practical and empirical insight into the geometry of hidden layer space, which would otherwise be difficult, if not outright impossible given that many of today's practical problems involve features that are of high-dimension and models that are of high-complexity. Results, first demonstrated on simple datasets, and subsequently extended to real-world datasets illustrated the effectiveness of this approach.

- Building upon the framework laid out in Chapter 2, Chapter 3 then investigated the possibility of using the SVD, together with two other problem specific operators comprising a variable length representation and a micro-hybrid genetic algorithm with an adaptive local search intensity scheme (both as local search operators) to improve the performance of a multi-objective evolutionary algorithm approach to training neural networks. Similar to the previous chapter, but based on a different learning scheme and context, I consider the use of a information measure based on the SVD to estimate the necessary number of neurons to be used in training a single hidden layer feedforward network – this, to address the issue of network architectural development in an evolutionary learning approach. In this approach, the SVD was used in a recombination scheme that was termed as an SVD-based architectural recombination (SVAR). This idea here is for the purpose of facilitating the exchange of neuronal information between candidate ANN designs and adaptation of the number of neurons for each individual, in a population-based approach.

- Subsequently, Chapter 4 proposed a learning algorithm for both feedforward and recurrent

neural architectures based on a 'layered' training mechanism. For feedforward networks (first section), this was done using approximations of the Hessian based on only local information, specifically, the correlations of output activations from previous layers of hidden neurons. The pseudoinverse is then used to train the output layer weights while a second-order method using a Hessian approximation is used to adapt the hidden layer(s) weights. Regularization terms introduced allow fine-tuning of both the weights in all hidden layers. This is partly motivated on the understanding that as the structural complexity of the network increases, or if noise is known to be present in the training examples, it makes little sense to impose a 'strong' or 'strict' learning algorithm for which it will tend to overtrain the network, thus leading to a decreased ability of the resulting network to generalize on unseen data. With that in mind, approximate methods incorporating regularization terms would in turn allow better and faster training techniques to be applied.

On the other hand, for a recurrent MLP structure (second section) this layered training was achieved via a hybrid Evolutionary Strategy (ES) and pseudoinverse approach together with an adaptive linear observer (the pseudoinverse and adaptive linear observer both acting as local search operators), through the simultaneous evolution of connection weights, its mutation step-sizes as well as a separate observer system, which in turn facilitates the exploration of interactions between the structure and weights of the network. Results indicated a faster rate of convergence with performance matching or exceeding benchmarks. This gives us evidence that a hybridized technique, when approached correctly, can solve complex problems that would otherwise be unsolvable by standard gradient-based methods.

In the second part of this dissertation, I examined the use of linear-threshold type neurons in recurrent neural networks addressing both theoretical and application-oriented aspects – because of its continuous, real-valued elements, allow a greater representational scheme which in turn is more expressive than discrete, binary-state neurons. Specifically,

- Chapter 5 puts forward an extension of existing results in the dynamics analysis of linear threshold neurons in recurrent networks by establishing new and milder conditions for boundedness and asymptotical stability while allowing for multistability, where the typical requirement for symmetric and nonnegative conditions for a deterministic matrix are not necessary to assert

bounded dynamics. As an example of a possible application of the proposed LT network, a simple analog associative memory scheme was demonstrated – where a design method describing associative memory is suggested similar to a generalized Hebbian approach but with distinctions of additional network parameters for normalization, excitation and inhibition, both on a global and local scale – this was based on a fully-connected single-layered recurrent network of linear threshold neurons.

Because the computational abilities of an LT network is the consequence of its switching between active and inactive partitions, this is consistent with the behavior of typical associative memory systems, with dense patterns resulting in low storage capacity. Although the performance of the associative memory capacity was largely limited to a few stored patterns of high density (low sparsity), LT networks have shown to exhibit interesting dynamics that can be further analyzed and addressed as a future direction of research.

- Lastly, Chapter 6 investigated the use of a recurrent network of linear threshold (LT) neurons (similar in structure to the previous chapter), but with modified asynchronous state update dynamics to solve the classical Traveling Salesman Problem (TSP). Results that were obtained from utilizing the proposed network of LT neurons using asynchronous update dynamics consistently produced better results than its Hopfield equivalent in solving certain instances of the TSP. The improved results are believed to be attributed to the ability of the LT neurons to avoid saturating levels, characteristic of many archetypical neuronal activating functions. The objective of this chapter was not to demonstrate improved results for the TSP, but rather, to put forward the viability of using neurons with non-saturating properties, specifically of the linear threshold (LT) type, as a candidate activation function for solving such combinatorial optimization problems using Hopfield-like recurrent networks.

7.2 Some Open Problems and Future Directions

The computational abilities of a neural network, either feedforward or recurrent, stems from a confluence of factors – two of the most critical being its (i) learning algorithms, used to optimize a set of parameters, usually the weights of the neural network; and (ii) structure, from its architectural

complexity. While much of the research done thus far has focused on the former, the latter has seen fewer contributions. That said, the advancement of one of the two factors cannot proceed far without an equivalent development in the other – in other words, the structure affects the learning algorithm, and vice-versa.

In feedforward neural networks, much of the emphasis has been placed on the aspect of its learning algorithms, which is quite rightly so as the architecture of such networks are assumed to be fairly simple. However, the same does not hold for recurrent neural networks, as given the additional degrees of freedom in how the synaptic weights are connected, in terms of lateral and feedback connections instead of just the usual feedforward connections, requires a greater level of analysis and understanding to fully comprehend how architecture-specific learning algorithms can be developed to exploit the additional structural complexity and freedom of the recurrent network. As has been argued earlier in this thesis – the computational power of a neural network arises from both the architecture as well as learning algorithm used. As a case in point, using a feedforward structure as an example – increasing the complexity of the network via the number of hidden layers/neurons would require fewer training epochs to reach a particular level of performance. The same performance level can be achieved by using a more parsimonious architecture but with increased complexity (training epochs). Of course this is a simple example but nevertheless serves to underline the close and often causal relationship between the network structure and the learning algorithm.

Looking back, the use of the SVD, as was done in Chapter 2, can be extended to the estimation of hidden neurons in feedforward neural networks with multiple hidden layers, as well as in recurrent neural networks. The use of the SVD does not need to be confined to a particular architecture – I believe that an important and particularly interesting aspect of analysis would be to investigate the hidden layer space of neural networks, for both feedforward and recurrent types as the hidden layer(s) are the single most important contributor to the computational abilities of the network. Through the decomposition of the hidden layer space, a better understanding of how the hidden layer neurons operate is obtained. in fact be used during the learning process to examine the geometric implications as well as understanding the behavior of how the modification of the weights during the training process affects the computational ability of the neural network. Another possible direction for future work would be to study the use of the proposed approach on networks that have been trained using constructive learning algorithms that are non-tuning based, such as the ELM paradigm [104].

Similarly, in Chapter 3, while the effectiveness of our proposed approach for classification problems was demonstrated on feedforward-type neural networks, it is believed that the methods that were employed are sufficiently flexible and robust to be extended to handle a variety of problem domains, such as regression, prediction as well as system identification problems, all of which can be further investigated as a future direction for research. In a likewise manner, the approach proposed in the first part of Chapter 4 can be used for networks with multiple hidden layers, which is a possible direction for future exploration. This would be beneficial for (feedforward) neural networks particularly those with many hidden layers, where each layer would be specifically trained for a certain task (in applications such as hardware replication of the human visual system). In such a situation, each layer can adopt a particular learning algorithm, with task specificity.

An emerging possibility, based on the second part of Chapter 4 would be to strengthen the coupling between the Evolutionary Strategy and the local search technique used such that the resulting synergistic interaction between them would be able to complement the strengths of these two methods leading to a better performance [1] – present search dynamics indicate the possibility of constructing hybrid EA-based learning algorithms for neural networks (both feedforward and recurrent) that are extremely versatile. Future work could focus on constructing a more general representation for the evolutionary process to undertake, as well as to perform further comparative analysis on a wider, and more complex range of problem sets.

Lastly, looking at Chapters 5 and 6, recurrent neural networks using Linear Threshold (LT) neurons are emerging as a potentially interesting field of study. Given their rich dynamics, partly due to their non-saturating behavior means possible application in a wide variety of problem domains. An avenue for possible study would be the use of LT networks not only in replicating and modeling the dynamics of biological systems (LT neurons were in fact inspired by the anatomical make-up of the eye [83]), but also in solving suitably mapped combinatorial optimization problems.

In concluding, the past two to three decades have seen research in neural networks achieve many notable accomplishments in the areas of theoretical analysis, mathematical modeling and have found widespread acceptance and use in a variety of domains from an application perspective. In writing this dissertation, the author hopes it will be helpful for readers who share similar interests in these few areas of neural network research, and to inspire new ideas in related topics.

[1] While evolutionary algorithms are suited for a first-stage, global-based search, gradient-descent searches work well as a fine-tuning mechanism

List of Publications

This is a list of publication, which largely contributed to the development of this dissertation.

Journals

1. Goh, C. K., Teoh, E. J., Tan, K. C. and Liu, D. K., "A hybrid evolutionary approach for heterogeneous multiprocessor scheduling", *Soft Computing*, accepted.
2. Goh, C. K., Teoh, E. J. and Tan, K. C., "Hybrid multiobjective evolutionary design for artificial neural networks", *IEEE Transactions on Neural Networks*, accepted.
3. Teoh, E. J., Tan, K. C., Tang, H. J., Xiang, C. and Goh, C. K., "An asynchronous recurrent linear threshold network approach to solving the traveling salesman problem", *Neurocomputing*, vol. 71, issue 7-9, pp. 1359-1372, March 2008.
4. Teoh, E. J., Tan, K. C. and Xiang, C., "Estimating the number of hidden neurons in a feed-forward network using the Singular Value Decomposition", *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1623-1629, 2006.
5. Tang, H. J., Tan, K. C. and Teoh, E. J., "Dynamics analysis and analog associative memory of networks with LT neurons", *IEEE Transactions on Neural Networks*, vol. 17, no. 2, pp. 409-418, 2006.

Conferences

1. J.H. Ang, E.J. Teoh and K.C. Tan, "A Hybrid Evolutionary Algorithm for Attribute Selection", Proc. IEEE International Joint Conference on Evolutionary Computation, World Congress on Computational Intelligence (WCCI), June 1-6, Hong Kong, 2008.
2. E.J. Teoh and C. Xiang, "Feature Selection and Classification via a GA-SVM Hybrid", International Conference on Genetic and Evolutionary Methods, WORLDCOMP'08, Las Vegas, July 14-17, 2008.

3. E.J. Teoh and C. Xiang, "Index Prediction Using Multiple Classifiers", Fifth International Symposium on Neural Networks (ISNN), September 24-28, Nanjing, China, 2008.

4. K.C. Tan, C.K. Goh, E.J. Teoh, and D. Liu, "A Hybrid Evolutionary Approach for Heterogeneous Multiprocessor scheduling", The 8th International Conference on Intelligent Technologies, Australia, December 12-14, 2007.

5. E.J. Teoh and C. Xiang, "A Hybrid Evolutionary Strategy (ES) – Least-Squares Approach for Time-Dependent Nonlinear Trajectory Learning in Recurrent Neural Networks (RNNs)", Proc. of IEEE Congress on Evolutionary Computation, Singapore, September 25-28, 2007.

6. K.C. Tan, C.K. Goh, J.H. Ang and E.J. Teoh, "Designing recurrent neural network-based controller for gyro-mirror line-of-sight stabilisation system using an artificial immune algorithm", The Third International Conference on Autonomous Robots and Agents, Palmerston North, New Zealand, 12-14 December, pp. 255-260, 2006.

7. E.J. Teoh, H.J. Tang and K.C Tan, "A Columnar Competitive Model with Simulated Annealing for Solving Combinatorial Optimization Problems", Proc. IEEE International Joint Conference on Neural Networks, Vancouver, BC, Canada, July 16-21, pp. 3254-3259, 2006.

8. E.J. Teoh, C. Xiang and K.C. Tan, "Estimating the Number of Hidden Neurons in a Feedforward Network Using the Singular Value Decomposition", Advances in Neural Networks – ISNN 2006: Third International Symposium on Neural Networks, Chengdu, China, May 28 – June 1, Proceedings, Part I: pp. 858-865, 2006.

9. E.J. Teoh, C. Xiang and K.C. Tan, "A Fast Learning Algorithm Based on Layered Hessian Approximations and the Pseudoinverse", Advances in Neural Networks – ISNN 2006: Third International Symposium on Neural Networks, Chengdu, China, May 28 – June 1, Proceedings, Part I: pp. 530-536, 2006

10. E.J. Teoh, S.C. Chiam, C.K. Goh and K.C. Tan, "Adapting evolutionary dynamics of variation for multi-objective optimization", IEEE Congress on Evolutionary Computation, Edinburgh, UK, 2-5 September, vol. 2, pp. 1290-1297, 2005.

11. C.K. Goh, H.Y. Quek, E.J. Teoh and K.C. Tan, "Evolution and incremental learning in the iterative prisoner's dilemma", IEEE Congress on Evolutionary Computation, Edinburgh, UK, 2-5 September, vol. 3, pp. 2629-2636, 2005.

Book Chapters

1. J.H. Ang, C.K. Goh, E.J. Teoh, and K.C. Tan, "Designing recurrent neural network-based controller for gyro-mirror line-of-sight stabilization system using an artificial immune algorithm", Advances in Evolutionary Computing for System Design, Springer-Verlag, pp. 187-208, 2007.

Bibliography

[1] H.A. Abbass, "A Memetic Pareto Evolutionary Approach to Artificial Neural Networks," in *Proceeding of the Australian Joint Conference on Artificial Intelligence, Lecture Notes in Artificial Intelligence (LNAI 2256)*, Springer-Verlag, pp. 1–12, 2001.

[2] H.A. Abbass, "Speeding up backpropagation using multiobjective evolutionary algorithms," *Neural Computation*, vol. 15, pp. 2705-2726, 2003.

[3] A. Abe, "Global convergence and suppression of spurious states of the Hopfield neural networks", *IEEE Trans. Circuits and Systems I*, vol. 40, no. 4, pp. 246–257, 1993.

[4] S. Aeberhard, D. Coomans and O. de Vel", "Comparison of Classifiers in High Dimensional Settings,"Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland (no. 92-02), 1992.

[5] S. Aeberhard, D. Coomans and O. de Vel", "The classification performance of RDA,"Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland (no. 92-01), 1992.

[6] C.W. Ahn and R.S. Ramakrishna, "Elitism-based compact genetic algorithms", *IEEE Trans. Evolutionary Computation*, 7(4):367–385, 2003.

[7] S.V.B. Aiyer, M. Niranjan and F. Fallside, "A theoretical investigation into the performance of the Hopfield model", *IEEE Trans. Neural Networks*, vol. 1, no. 2, pp. 204-215, 1990.

[8] E. Alba and J.F. Chicano, "Training Neural Networks with GA Hybrid Algorithms", K. Deb(ed.), *Proceedings of GECCO'04*, Seattle, Washington, LNCS 3102, pp. 852–863, 2004.

[9] A. Albert, "Regression and the Moore-Penrose Pseudoinverse,"Academic Press, New York and London, 1972.

[10] J.A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, 1988.

[11] P.J. Angeline, G.M. Saunders, and J.B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks,"*IEEE Transactions on Neural Networks*, 5(1):54-65, 1994.

[12] M.-L. Antonie, O.R. Zaiane, R.C. Holte, "Learning to Use a Learned Model: A Two-Stage Approach to Classification", in *Proceedings of the Sixth International Conference on Data Mining*, pp. 33–42, 2006.

[13] M. Anthony, "Probabilistic Analysis of Learning in Artificial Neural Networks: The PAC Model and its Variants," NC-TR-94-3, London, UK, 1994, url = " citeseer.ist.psu.edu/article/anthony97probabilistic.html"

[14] T. Back, U. Hammel, , and H.-P. Schwefel. "Evolutionary computation: Comments on the history and current state," *IEEE Transactions on Evolutionary Computation*, 1(1):3-17, 1997.

[15] N.K. Bambha, S.S. Bhattacharyya, J. Teich, and E. Zitzler, "Systematic Integration of Parameterized Local Search Into Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 137–154, 2004.

[16] P.L. Bartlett, "The sample complexity of pattern classication with neural networks: The size of the weights is more important than the size of the network," *IEEE Trans. Information Theory*, vol. 44(2), pp. 525–536, 1998.

[17] R. Battiti, "First- and second-order methods for learning: between steepest descent and Newton's method," *Neural Computation*, vol. 4, pp. 141–166, 1992.

[18] U. Bauer, M. Scholz, J.B. Levitt, K. Obermayer, and J.S. Lund, "A biologically-based neural network model for geniculocortical information transfer in the primate visual system," *Vision Research*, vol. 39, pp. 613–629, 1999.

[19] J.S. Bay, *Fundamentals of Linear State Space Systems*, McGraw-Hill, 1999.

[20] R. Bellman, "Combinatorial processes and dynamic programming", in: *Combinatorial Analysis* (R. Bellman and M. Hall, Jr., eds.), American Mathematical Society, pp. 217–249, 1960.

[21] R. Ben-Yishai, R. Lev Bar-Or, and H. Sompolinsky, "Theory of orientation tuning in visual cortex," *Prof. Nat. Acad. Sci. USA*, vol. 92, pp. 3844–3848, 1995.

[22] H.G. Beyer and H.P. Schwefel, "Evolution strategies: A comprehensive introduction," *Natural Computing*, 1:3-52, 2002.

[23] M. Bianchini, M. Gori, and M. Maggini, "On the problem of local minima in recurrent neural networks," *IEEE Transactions on Neural Networks*, Special Issue on Dynamic Recurrent Neural Networks:167-177, 1994.

[24] G.L. Bilbro, W.E. Snyder, S.J. Garnier, and J.W. Gault, "Mean Field Annealing: A Formalism for Constructing GNC-like Algorithms," *IEEE Transactions on Neural Networks*, 3(1):131–138, 1992.

[25] C.M. Bishop, "Exact calculation of the Hessian matrix for the multi-layer perceptron," *Neural Computation*, vol. 4(4), pp. 494–501, 1992.

[26] C. Blum and K. Socha, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification", *Fifth International Conference on Hybrid Intelligent Systems (HIS05)*, pp. 233–238, 2005.

[27] A. Bouzerdoum and T.R. Pattison, "Neural network f or quadratic opmization with bound constraints", *IEEE Trans. Neural Networks*, vol. 4, no. 2, pp. 293–303, 1993.

[28] W.R. Buntine and A.S. Weigend, "Computing second derivatives in feed-forward networks: a review," *IEEE Trans. Neural Networks*, vol. 5(3), pp. 480–488, 1994.

[29] E. Cant-Paz, and C. Kamath, "An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, pp. 915–927, 2005.

[30] Z. Chen and S. Haykin, "On Different Facets of Regularization Theory," *Neural Computation*, vol. 14, pp. 2791–2846, 2002.

[31] C.A. Coello Coello, "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques," *Knowledge and Information Systems: An International Journal*, vol. 1, no. 3, pp, 269–308, 1999.

[32] C.A. Coello Coello and A. H. Aguirre, "Design of Combinational Logic Circuits through an Evolutionary Multiobjective Optimization Approach", *Artificial Intelligence for Engineering, Design, Analysis and Manufacture*, Cambridge University Press, vol. 16, no. 1, pp. 39-53, 2002.

[33] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, no. 4, pp. 303-314, 1989.

[34] T.M. Cover, "Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition," *IEEE Trans. Electronic. Comput.*, vol. 14, pp. 326–34, 1965.

[35] L.H. Cox, M.M. Johnson and K. Kafadar, "Exposition of Statistical Graphics Technology," ASA Proc Stat. Comp Section, pp. 55–56, 1982.

[36] G. Cybenko, "Continuous valued neural networks with two hidden layers are sufficient," Department of Computer Science, Tufts University, Medford, Massachusetts:Technical report, 1988.

[37] P. Dayan and L.F. Abbott, *Theoretical Neuroscience*, MIT Press, 2001.

[38] K. Deb and R.B. Agrawal, "Simulated Binary Crossover for Continuous Search Space", *Complex Systems*, 9:115–148, 1995.

[39] K. Deb, *Multi-objective Optimization Using Evolutionary Algorithms*, John Wiley & Sons, New York, 2001.

[40] K. De Jong, "An analysis of the behaviour of a class of genetic adaptive systems", Ph.D thesis, University of Michigan, 1975.

[41] M. Dorigo, V. Maniezzo and A. Colorni, "Ant System: optimization by a colony of cooperating agents", IEEE Transactions on Systems, Man and Cybernetics - Part B, vol. 26, no. 1, pp. 29–41, 1996.

[42] R. Douglas, C. Koch, M. Mahowald, K. Martin and H. Suarez, "Recurrent excitation in neocortical circuits," *Science*, vol. 269, pp. 981–985, 1995.

[43] D. Dumitrescu, B. Lazzerini, L.C. Jain and A. Dumitrescu, *Evolutionary Computation*, The CRC Press International Series on Computational Intelligence, 2000.

[44] R. Durbin and D. Willshaw,, "An analogue approach to the traveling salesman problem using an elastic net method", *Nature*, 326:689–691, 1987.

[45] E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, Natural Computing Series, MIT Press, Springer, Berlin, 2003.

[46] J.E. Fieldsend and S. Singh, "Pareto evolutionary neural networks," *IEEE Transactions on Neural Networks*, vol. 16, no. 2, pp. 338–354, 2005.

[47] J. Feng, and K.P. Hadeler, "Qualitative behavior of some simple networks", *Journal of Physics A*, 29:5019–5033, 1996.

[48] J. Feng, "Lyapunov functions for neural nets with nondifferentiable input-output characteristics," *Neural Computation*, vol. 9, pp. 43–49, 1997.

[49] R.A. Fisher, "The use of multiple measurements in taxonomic problems," *Annual Eugenics*, vol. 7(2), pp. 179–188, 1936.

[50] R.A. Fisher, "The use of multiple measurements in taxonomic problems," *Contributions to Mathematical Statistics*, vol. 7(2), pp. 179–188, John Wiley, NY, 1950.

[51] D.B. Fogel, E.C. Wasson, and E.M. Boughton, "Evolving neural networks for detecting breast cancer," *Cancer Letters*, vol. 96, no. 1, pp. 49-53, 1995.

[52] C.M. Fonseca, and P.J. Fleming, "Genetic algorithm for multiobjective optimization, formulation, discussion and generalization," in *Proceeding of the Fifth International Conference on Genetic Algorithms*, pp. 416–423, 1993.

[53] M. Forti and A. Tesi, "New conditions for global stability of neural networks with application to linear and quadratic programming problems," *IEEE Trans. Circuits Systems*, vol. 42, pp. 354–366, 1995.

[54] E. Frank and I.H. Witten, Generating accurate rule sets without global optimization, in *Proceedings of the Fifteenth International Conference Machine Learning*, vol. 22, pp. 144-151, 1998.

[55] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183-192, 1989.

[56] J. Gallier, *Geometric Methods and Applications For Computer Science and Engineering*, Texts in Applied Mathematics, Vol.38, Springer-Verlag, New York, 2000.

[57] N. Garcia-Pedrajas, C. Hervas-Martinez and D. Ortiz-Boyer, "Cooperative Coevolution of Artificial Neural Networks Ensembles for Pattern Classification," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 271–302, 2005.

[58] N. Garcia-Pedrajas, C. Hervas-Martinez and J. Munoz-Perez, "Multiobjective cooperative coevolution of artificial neural networks," *Neural Networks*, vol. 15, no. 10, pp. 1255–1274, 2002.

[59] R. Ghosh and B. Verma, "Finding Optimal Architecture and Weights Using Evolutionary Least Square Based Learning", in *Proceedings of Neural Information Processing*, vol. 1, pp. 528–532, 2002.

[60] O. Giustolisi and V. Simeone, "Optimal design of artificial neural networks by a multi-objective strategy: groundwater level predictions," *Hydrological Sciences Journal*, vol. 51, no. 3, 2006.

[61] F. Glover, "Future paths for integer programming and links to artificial intelligence", *Computers and Operation Research*, Vol. 13, pp. 533–549, 1986.

[62] C.K. Goh, E.J. Teoh and K.C. Tan, "Hybrid Multiobjective Evolutionary Neural Networks", *IEEE Trans. Neural Networks*, accepted.

[63] J.L. Goldberg, *Matrix Theory With Applications*, McGraw-Hill, 1992.

[64] D.E. Goldberg, and J. Richardson, "Genetic algorithms with sharing for multi-modal function optimization," in *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 41–49, 1987.

[65] D.E. Goldberg, "Genetic algorithms and walsh functions: Part 2 – deception and its analysis," *Complex Systems*, 3:153-171, 1989.

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

[67] D.E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of Genetic Algorithms*, G.J.E. Rawlins (Ed.), Morgan-Kaufmann, San Mateo, CA:6993, 1991.

[68] G.H. Golub and C. Reinsch, "Singular Value Decomposition and Least Squares Solutions," *Numer. Math.*, vol. 14, pp.403–420, 1970.

[69] , G.H. Golub, V. Klema and G.W. Stewart, "Rank Degeneracy and the Least Squares Problem," *Technical Report (STAN-CS-76-559), Computer Science Department, School of Humanities and Sciences, Stanford University*, 1976.

[70] G.H. Golub and C.G. Van Loan", *Matrix Computations (3rd ed.)*, John Hopkins University Press, Baltimore, 1999.

[71] G.H. Golub, P.C. Hansen and D.P. O'Leary, "Tikhonov regularization and total least squares," *SIAM J. Matrix Anal. Appl.*, vol. 21(1), pp. 185–194, 1999.

[72] S.M. Goni, S. Oddone, J.A. Segura, R.H. Mascheroni, V.O Salvadori, "Prediction of foods freezing and thawing times: artificial neural networks and genetic algorithm approach", Journal of Food Engineering, vol. 84, no. 1, pp. 164–178, 2008.

[73] R.P. Gorman and T.J. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," *Neural Networks*, vol. 1(1), pp. 75–89, 1988.

[74] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms and architectures", *Neural networks*, vol. 1, pp. 17–61, 1988.

[75] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[76] R.H.R. Hahnloser, "On the piecewise analysis of networks of linear threshold neurons," *Neural Networks*, vol. 11, pp. 691–697, 1998.

[77] R.H.R. Hahnloser, R. Sarpeshkar, M.A. Mahowald, R.J. Douglas, and H.S. Seung, "Digital selection and analog amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, pp. 947–951, 2000.

[78] R.H.R. Hahnloser, H.S. Seung, and J.J. Slotine, "Permitted and forbidden sets in symmetric threshold-linear networks," *Neural Computation*, vol. 15, no. 3, pp. 621–638, 2003.

[79] P. Hajela and C.Y. Lin, "Genetic search strategies in mulitcriterion optimal design," *Journal of Structural Optimization*, 4:99–107.

[80] K. De Jong, L. Fogel, and H-P. Schwefel, *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, 1997.

[81] P.C. Hansen, "The 2-norm of random matrices," *J. Comp. Appl. Math*, vol. 23, pp. 185–199, 1988.

[82] P.C. Hansen, "Rank-Deficient and Discrete Ill-Posed Problems: Numerical Aspects of Linear Inversion," *SIAM*, 1998.

[83] H.K. Hartline and F. Ratliff, "Spatial summation of inhibitory influence in the eye of limulus and the mutual interaction of receptor units," *Journal of General Physiology*, vol. 41, pp. 1049–1066, 1958.

[84] B. Hassibi, D.G. Stork and G.J. Wolff, "Optimal brain surgeon and general network pruning," *Proc. IEEE Int. Conf. Neural Networks*, vol. 1, pp. 293-299, 1992.

[85] M. Hayashi, "A Fast Algorithm for the Hidden Units in a Multilayer Perceptron," *Proc. Int. Joint Conf. on Neural Networks*, vol. 1, pp. 339–342, 1993.

[86] S. Haykin, *Adaptive Filter Theory (3rd ed.)*, Prentice Hall, Eaglewood Cliffs, NJ, 1996.

[87] S. Haykin, *Neural networks: A comprehensive foundation (2nd ed.)*, Prentice Hall, Upper Saddle River, NJ, 1999.

[88] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the theory of neural computation*, New York: Addison-Wesley, 1991.

[89] R. Hinterding, Z. Michalewicz, and A.E. Eiben. "Adaptation in evolutionary computation: A survey," *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, pages 65-69, 1997.

[90] A.E. Hoerl and R.W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems,"emphTechnometrics, vol. 12(3), pp. 501-506, 1970.

[91] J.H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, MI, 1975.

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

[93] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci. USA*, vol. 79, pp. 2554–2558, 1982.

[94] J.J. Hopfield, "Neurons with graded responses have collective computational properties like those of two-state neurons", *Proc. Natl. Acad. Sci. USA*, 81:3088–3092, 1984.

[95] J.J. Hopfield and D.W. Tank, "'Neural' computation of decisions in optimization problems", *Biol. Cybern.*, 52:141–152, 1985.

[96] J.J. Hopfield and D.W. Tank, "Computing with neural circuits: A model", *Science*, 233:625–633, 1986.

[97] R.A. Horn and C.R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, UK, 1985.

[98] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[99] P. Horton and K. Nakai, "A Probablistic Classification System for Predicting the Cellular Localization Sites of Proteins," *Intelligent Systems in Molecular Biology*, pp. 109–115, 1996.

[100] S.C Huang and Y.F. Huang, "Bounds on Number of Hidden Neurons of Multilayer Perceptrons in Classification and Recognition," *IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 2500–2503, 1990.

[101] G.B Huang and H.A. Babri, "Upper Bounds on the Number of Hidden Neurons in Feedforward Networks with Arbitrary Bounded Nonlinear Activation Functions," *IEEE Trans. Neural Networks*, vol. 9(1), 1998.

[102] G.B. Huang, and Y.Q. Chen and H.A. Babri, "Classification ability of single hidden layer feedforward neural networks," *IEEE Trans. on Neural Networks*, vol. 11(3), pp. 799–801, 2000.

[103] G.B. Huang, "Learning Capability and Storage Capacity of Two-Hidden-Layer Feedforward Networks," *IEEE Trans. on Neural Networks*, 14(2), pp. 274–281, 2003.

[104] G.B. Huang, Q.Y. Zhu and C.K. Siew, "Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks," *Proc. Int. Joint Conf. on Neural Networks*, 2004.

[105] D.R. Hush, "Learning from examples: from theory to practice," *Proc. Tutorial 4 Proc. IEEE Int. Conf. Neural Networks*, 1997.

[106] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol.4, pp. 251–257, 1991.

[107] H. Inoue and H. Narihisa, "Self-Organizing Neural Grove and Its Applications", in *Proceedings of International Joint Conference on Neural Networks*, pp. 1205–1210, 2005.

[108] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling", *IEEE Transaction on Systems, Man, and Cybernetics - Part C*, vol. 28, no. 3, pp. 392-403, 1998.

[109] H. Ishibuchi, T. Yoshida, and T. Murata, "Balance between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop", *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 204–223, 2003

[110] A. Jaszkiewicz, "On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment," *IEEE Transaction on Evolutionary Computation*, vol. 6, no. 4, pp.402–412, 2002.

[111] G.H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345, 1995.

[112] R.E. Kalman and R.S. Bucy, "New Results in Linear Filtering and Prediction Theory", *Transactions of the ASME - Journal of Basic Engineering*, Vol. 83: pp. 95–107, 1961.

[113] S.A. Karzrlis, S.E. Papadakis, J.B. Theocharis, and V. Petridis, "Microgenetic Algorithms as Generalized Hill-Climbing Operators for GA Optimization," *IEEE Transactions On Evolutionary Computation*, vol. 5, no. 3, pp. 204–217, 2001.

[114] J. Kennedy and R. Eberhart, "Particle Swarm Optimization", *Proc. IEEE Intl. Conf. on Neural Networks* (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV:1942–1948, 1995.

[115] E.F. Khor, K. C. Tan, T. H. Lee, and C. K. Goh, "A study on distribution preservation mechanism in evolutionary multi-objective optimization," *Artificial Intelligence Review*, vol. 23, no. 1, pp. 31–56, 2005.

[116] T. Kohonen, "Self-organized formation of topologically correct feature maps", *Biological Cybernetics*, 43:50–69, 1982.

[117] T. Kohonen, *Self-organization and associative memory*, 3rd ed. Springer-Verlag, Berlin, 1989.

[118] W. Kinnebrock, "Accelerating the standard backpropagation method using a genetic approach," *Neurocomputing*, vol. 6, no. 5-6, pp. 583-588, 1994.

[119] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671–680, 1983.

[120] V.C. Klema and A.J. Laub", "The Singular Value Decomposition: Its Computation and Some Applications," *IEEE Trans. Automatic Control*, vol. 2, pp. 164–176, 1980.

[121] J.D. Knowles, and D.W. Corne, "Approximating the non-dominated front using the Pareto archived evolution strategy," *Evolutionary Computation*, vol. 8, no. 2, pp. 149–172, 2000.

[122] E. M. Koper, W. D. Wood, and S. W. Schneider, "Aircraft antenna coupling minimization using genetic algorithms and approximations," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 2, pp. 742–751, 2004.

[123] K. Konstantinides and K. Yao, "Statistical Analysis of Effective Singular Values in Matrix Rank Determination," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. 36(5), pp. 757–763, 1988.

[124] V. Kurkov'a, "Kolmogorov's Theorem and Multilayer Neural Networks," *Neural Networks*, vol. 5, pp. 501–506, 1992.

[125] V. Kurkov'a, "Learning from data as an inverse problem,"In J. Antoch, editor, COMPSTAT2004, Springer-Verlag, pp. 1377–1384, 2004.

[126] Y. Lecun, J.S. Denker and S.A. Solla, "Optimal brain damage," *Adv. Neural Inform. Process. Syst.*, vol. 2, pp. 598–605, 1990.

[127] Y. LeCun, L. Bottou, G.B. Orr and K.-R. Miller, "Efficient backprop,"In G. B. Orr and K.-R. M/iller, editors, Neural Networks: Tricks of the Trade, Number 1524 in LNCS, chapter 1, Springer-Verlag, 1998.

[128] S.Z. Li, "Improving convergence and solution quality of Hopfield-type neural network with augmented lagrange multipliers", *IEEE Trans. Neural Networks*, 7(6):1507–1516, November, 1996.

[129] X.B. Liang and J. Wang, "A recurrent neural network for nonlinear continuously differentiable objective function and bound constraints", *IEEE Trans. Neural Networks*, vol. 11, no. 6, pp. 1251–1262, 2000.

[130] Y. Liu, X. Yao, and T. Higuchi, "Evolutionary Ensembles with Negative Correlation Learning," *IEEE Transactions On Evolutionary Computation*, vol. 4, no. 4, pp. 380–387, 2000.

[131] L. Ljung, *System Identification: Theory for the User*, Eaglewood Cliffs, NJ, Prentice-Hall, 1987.

[132] D. Lowe, "Adaptive radial basis function nonlinearities, and the problem of generalization," *Proceedings of the First IEE International Conference on Artificial Neural Networks*, pp. 171–175, 1989.

[133] D.G. Luenberger. *Introduction to Dynamic Systems: Theory, Models and Applications*, John Wiley and Sons, Inc. New York, 1979.

[134] C.Y. Maa and M. Shanblatt, "Linear and quadratic programming neural network analysis", *IEEE Trans. Neural Networks*, vol. 3, no. 4, pp. 580–594, 1992.

[135] S.W. Mahfoud, Niching Methods for Genetic Algorithms, Ph.D thesis, University of Illinous at Urbana-Champaign, 1995.

[136] M. Mandischer, "Evolving recurrent neural networks with non-binary encoding," *IEEE International Conference on Evolutionary Computation*, 2:584-589, 1995.

[137] O.L. Mangasarian and W.H. Wolberg, "Cancer diagnosis via linear programming," *SIAM News*, vol. 23(5), pp. 1–18, 1990.

[138] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 39-53, 1994.

[139] P. Merz and B. Freisleben, "A comparison of memetic algorithms, Tabu search, and ant colonies for the quadratic assignment problem," in *Proceedings of the 1999 Congress on Evolutionary Computation*, vol. 1, pp. 2063–2070, 1999.

[140] D. Michie, D.J. Spiegelhalter and C.C. Taylor, *Machine Learning, Neural and Statistical Classification*, London: Ellis Horwood, 1994.

[141] M.K. Muezzinoglu, C. Guzelis, and J.M. Zurada, "A new design approach for the complex-valued multistate Hopfield associative memory," *IEEE Trans. Neural Networks*, vol. 14, no. 4, pp. 891–899, 2003.

[142] M.F. Moller, "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning," *Neural Networks*, vol. 6, pp. 525–533, 1993.

[143] V.A. Morozov, *Methods for solving incorrectly posed problems*, Springer-Verlag, New York, 1984.

[144] D.C. Noelle, G.W. Cottrell, and F.R. Wilms, *Extreme attraction: The benefits of corner attractors*, Technical Report CS97-536, Department of Computer Science & Engineering, University of California, San Diego.

[145] Y.S Ong and A.J. Keane, "Meta-Lamarckian Learning in Memetic Algorithms", IEEE Transactions on Evolutionary Computation, vol. 8, no. 2, pp. 99–110, 2004.

[146] P.P. Palmes, T. Hayasaka, and S. Usui, "Mutation-Based Genetic Neural Network", IEEE Transactions on Neural Networks, Vol. 16, No. 3, pp. 587–600, May 2005.

[147] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[148] G. Papageorgiou, A. Likas and A. Stafylopatis, "Improved exploration in Hopfield network state-space through parameter perturbation driven by simulated annealing", *European Journal of Operational Research*, vol. 108, pp. 283–292, 1998.

[149] R. Parisi, E.D. Di Claudio, G. Orlandi and B.R. Rao, "A generalized learning paradigm exploiting the structure of feedforward neural networks," *IEEE Trans. Neural Networks*, vol. 7(6), pp. 1450–1460, 1996.

[150] B.A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, 1: 263-9, 1989.

[151] B.A. Pearlmutter, "Gradient calculation for dynamic recurrent neural networks: a survey," *IEEE Transactions on Neural Networks*, 6(5):1212-1228, 1995.

[152] M. Peng, K. Narendra and A. Gupta, "An investigation into the improvement of local optima of the Hopfield network", *Neural Networks*, vol. 9, pp. 225–233, 1993.

[153] M.J. Pérez-Ilzarbe, "Convergence analysis of discrete-time recurrent neural networks to perform quadratic real optimization with bound constraints" *IEEE Trans. Neural Networks*, vol. 9, no. 6, pp. 1344–1351, 1998.

[154] R. Petridis, S. Kazaplis, and A. Papaikonomou, "A genetic algorithm for training recurrent neural networks," *Proceedings of 1993 International Joint Conference on Neural Networks*, 3:2706-2709, 1993.

[155] F.J. Pineda, "Recurrent backpropagation and the dynamical approach to adaptive neural computation", Neural Computation, vol. 1, no. 2, pp. 161–172, 1989

[156] T. Poggio and F. Girosi, "A Theory of Networks for Approximation and Learning", AI memo 1140, MIT Artificial Intelligence Laboratory, July, 1989.

[157] W.H. Press, S.A. Teukolsky, W.T. Vetterling and B.P. Flannery", *Numerical Recipes in C*, Cambridge University Press, Cambridge, UK, 1992.

[158] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in C Example Book : The Art of Scientific Computing, 2nd. Ed.*, Cambridge University Press, 1994.

[159] D.C. Psichogios and L.H. Ungar, "SVD-NET: An Algorithm that Automatically Selects Network Structure," *IEEE Trans. on Neural Networks*, vol. 5(3), pp. 513–515, 1994.

[160] H. Qiao, J. Peng, Z.B. Xu and B. Zhang, "A reference model approach to stability analysis and of neural networks", *IEEE Trans. Systems, Man and Cybernetics (B)*, vol. 33, no. 6, pp. 925–936, 2003.

[161] J. R. Quinlan, *C4.5: Programs for Machine Learning*, San Mateo, CA: Morgan Kaufmann, 1993.

[162] J. Ramanujam and P. Sadayappan, "Mapping Combinatorial Optimization Problems onto Neural Networks", *Information Sciences* 82:239–255, 1995.

[163] M. Riedmiller, "Rprop - description and implementations details", Technical report, University of Karlsruhe, 1994.

[164] H. Ritter, "A spatial approach to feature linking", *Int. Neur. Netw. Conf.*, Paris, France, 1990.

[165] B.D. Ripley, "Neural networks and related methods for classification," *J. Roy. Statist. Soc. B.*, vol. 56(3), pp. 409–456, 1994.

[166] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing* vol. 1(8), pp. 318–362, MIT Press, Cambridge, 1986.

[167] A. Salinas and L.F. Abbott, "A model of multiplicative responses in parietal cortex," *Prof. Nat. Acad. Sci. USA*, vol. 93, pp. 11956–11961, 1996.

[168] N. Saravanan, D.B. Fogel and K.M. Nelson, "A comparison of methods for self-adaptation in evolutionary algorithms," *Biosystems*, vol. 36, pp. 157–166, 1995.

[169] M.A Sartori and P.J. Antsaklis, "A Simple Method to Derive Bounds on the Size and to Train Multi-Layer Neural Networks," *IEEE Trans. on Neural Networks*, vol. 2(4), pp. 467–471, 1991.

[170] R.S. Scalero and N. Tepedelenlioglu, "A fast new algorithm for training feedforward neural networks," *IEEE Trans. Signal Processing*, vol. 40(1), pp. 202–210, 1992.

[171] J. D. Schaffer, "Multiple-objective optimization using genetic algorithm," in *Proceedings of the First International Conference on Genetic Algorithms*, pp. 93–100, 1985.

[172] H.-P. Schwefel. *Evolution and optimum seeking*. New York: Wiley, pages 1423-1447, 1995.

[173] T.J. Sejnowski, P.K. Kienker, and G. Hinton. "Learning symmetry groups with hidden units: Beyond the perceptron," *Physica D*, pp. 22:260-275, 1986.

[174] D. Serre, *Matrices: Theory and Applications*, Springer-Verlag, New York, 2002.

[175] R.S. Sexton, B. Alidaee, R.E. Dorsey and J.D. Johnson, "Global optimization for artificial neural networks: a tabu search application", *European Journal of Operational Research*, (106)2-3, pp.570–584, 1998.

[176] R.S. Sexton, R.E. Dorsey and J.D. Johnson, "Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing", *European Journal of Operational Research*, (114), pp.589–601, 1999.

[177] V.G. Sigillito, S.P. Wing, L.V. Hutton and K.B. Baker, "Classification of radar returns from the ionosphere using neural networks," *Johns Hopkins APL Technical Digest*, vol. 10, pp. 262–266, 1989.

[178] K.A. Smith, "Neural networks for combinatorial optimization: A review of more than a decade of research", *INFORMS J. Comput.*, 11(1):15–34, 1999.

[179] C.M. Soukoulis, K. Levin, and G.S. Grest, "Irreversibility and Metastability in Spin-Glasses. I. Ising Model", *Physical Review B*, 28:1495–1509, 1983.

[180] N. Srinivas, and K. Deb, "Multiobjective optimization using non-dominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[181] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, No. 2, pp. 99-127, 2002.

[182] G.W. Stewart, "Determining Rank in the Presence of Error," *Technical Report (TR-92-108, TR-2972)* Institute for Advanced Computer Studies, Department of Computer Science, University of Maryland, College Park, Oct. 1992.

[183] H.H. Szu, R.L. Hartley, "Nonconvex Optimization by Fast Simulated Annealing", Proc. of IEEE, vol. 75, pp. 1538–1540, 1987.

[184] Y. Tachibana, G.H. Lee, H. Ichihashi and T. Miyoshi, "A simple steepest descent method for minimizing Hopfield energy to obtain optimal solution of the TSP with reasonable certainty", in *Proc. IEEE Int. Conf. Neural Networks*, 1995, pp. 1871–1875.

[185] P.M. Talaván and J. Yáñez, "Parameter setting of the Hopfield network applied to the TSP, *Neural Networks*, vol. 15, pp. 353–373, 2002.

[186] S. Tamura, "Capabilities of a Three-Layer Feedforward Neural Network," *Proc. Int. Joint Conf. on Neural Networks*, 1991.

[187] S. Tamura, M. Tateishi, M. Matsumoto and S. Akita, "Determination of the Number of Redundant Hidden Units in a Three-Layered Feedforward Neural Network," *Proc. Int. Joint Conf. on Neural Networks*, vol. 1, pp. 335–338, 1993.

[188] S. Tamura and M. Tateishi, "Capabilities of a Four-Layered Feedforward Neural Network: Four Layers Versus Three," *IEEE Trans. on Neural Networks*, vol. 8(2), pp. 251–255, 1997.

[189] K.C. Tan, E.F. Khor, J. Cai, C.M. Heng, and T.H. Lee, "Automating the drug scheduling of cancer chemotherapy via evolutionary computation," *Artificial Intelligence in Medicine*, vol. 25, pp. 169–185, 2002.

[190] K.C. Tan, H.J. Tang, and W.N. Zhang, "Qualitative analysis for recurrent neural networks with linear threshold transfer functions", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 5, pp. 1003–1012, 2005.

[191] K.C. Tan, H.J. Tang, and S.S. Ge, "Dynamical stability analysis for parameter settings of Hopfield neural networks applied to TSP", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 5, pp. 994–1002, 2005.

[192] K.C. Tan, C.K. Goh, Y.J. Yang, and T.H. Lee, "Evolving better population distribution and exploration in evolutionary multi-objective optimization," *European Journal of Operational Research*, vol. 171, no. 2, pp. 463–495, 2006.

[193] K.C. Tan, Q. Yu and J.H. Ang, "A coevolutionary algorithm for rules discovery in data mining," *International Journal of Systems Science*, vol. 37, no. 12, pp. 835–864, 2006.

[194] D.W. Tank and J.J. Hopfield, "Sinple neural optimization networks: an a/d converter, signal decision circuit, and a linear programming circuit", *IEEE Trans. Circuits and Systems*, vol. 33, no. 5, pp. 533–541, 1986.

[195] H. Tang and K.C. Tan and Z. Yi, "A columnar competitive model for solving combinatorial optimization problems", *IEEE Trans. Neural Networks*, 15(6):1568–1573, 2004.

[196] K. C. Tan, H. J. Tang, and Z. Yi, "Global exponential stability of discrete-time neural networks for constrained quadratic optimization," *Neurocomputing*, vol. 56, pp. 399–406, 2004.

[197] H.J. Tang, K.C. Tan, and E.J. Teoh, "Dynamics analysis and analog associative memory of networks with LT neurons", *IEEE Trans. Neural Networks*, vol. 17, no. 2, pp. 409–418, 2006.

[198] E.J. Teoh, K. C. Tan and C. Xiang, "Estimating the number of hidden neurons in a feedforward network using the Singular Value Decomposition," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1623–1629, 2006.

[199] S. Theodoridis and K. Koutroumbas, *Pattern recognition (2nd. ed.)*, Academic Science, 2003.

[200] A.N. Tikhonov and V.Y. Arsenin, *Solution of ill-posed problems*, V.H. Winston, Washington, DC, 1977.

[201] N.K. Treadgold and T.D. Gedeon, "Simulated annealing and weight decay in adaptive learning: the SARPROP algorithm", *IEEE Transactions on Neural Networks*, vol. 9:662–668, 1998.

[202] B. Verma and R. Ghosh, "A novel evolutionary Neural Learning Algorithm," in *Proceedings of the 2002 Congress on Evolutionary Computation*, vol. 2 , pp. 1884-1889, 2002.

[203] , M. Vidyasagar, *Nonlinear systems analysis*, 2nd. ed., Prentice Hall, New Jersey, 1992.

[204] J. Wang, "Recurrent neural networks for computing pseudoinverses of rank-deficient matrices," *SIAM J. Sci. Comput.*, vol. 18(5), pp. 1479–1493, 1997.

[205] N. Weicker, G. Szabo, K. Weicker, and P. Widmayer, "Evolutionary Multiobjective Optimization for Base Station Transmitter Placement with Frequency Assignment," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 189-203, 2003.

[206] A. Weigend and D.E. Rumelhart, "The Effective Dimension of the Space of Hidden Units,"*IEEE International Joint Conference on Neural Networks*, pp. 2069–2074, 1991.

[207] A.S. Weigend, D.E. Rumelhart and B.A. Huberman, "Generalization by weight elimination with application to forecasting,"*Adv. Neural Inform. Process. Syst.*, vol. 3, pp. 875–882, 1991.

[208] P. Werbos, "Backpropagation through time: What does it do and how to do it", *Proc. IEEE*, 78:1550-1560, 1990.

[209] H. Wersing, J.J. Steil, and H. Ritter, "A competitive layer model for feature binding and sensory segmentation," *Neural Computation*, vol. 13, no. 2, pp. 357–387, 2001.

[210] H. Wersing, W.J. Beyn, and H. Ritter, "Dynamical stability conditions for recurrent neural networks with unsaturating piecewise linear transfer functions," *Neural Computation*, vol. 13, no. 8, pp. 1811–1825, 2001.

[211] R.J. Williams and D. Zipser, "Experimental analysis of the real-time recurrent learning algorithm," Connection Science, pp. 87-111:1212-1228, 1989.

[212] W.H. Wolberg and O.L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology,"emphProceedings of the National Academy of Sciences, vol. 87, pp. 9193–9196, 1990.

[213] C. Xiang, S.Q. Ding and T.H. Lee, "Geometrical Interpretation and Architecture Selection of MLP,"*IEEE Trans. Neural Networks*, vol. 16(1), pp. 84–96, 2005.

[214] X. Xie, R.H.R. Hahnloser, and H.S. Seung, "Selectively grouping neurons in recurrent networks of lateral inhibition," *Neural Computation*, vol. 14, no. 11, pp. 2627–2646, 2002.

[215] X. Yao and Y. Liu, "Fast evolutionary programming," in *Proc. Fifth Annual Conf. Evolutionary Programming (EP96)*, L.J. Fogel, P.J. Angeline, and T. Back, Eds. Cambridge, MA: MIT Press: pp. 451-460, 1996.

[216] X. Yao and Y. Liu, "Fast evolution strategies," in *Evolutionary Programming VI: Proc. of the Sixth Int. Conf. Evolutionary Programming (EP97)*, P.J. Angeline, R.G. Reynolds, J.R. McDonnell, and R. Eberhart, Eds. Berlin, Germany: Springer: pp. 151-161, 1997.

[217] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694-713, 1997.

[218] X. Yao, "Evolving Artificial Neural Networks", in *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[219] X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Transaction on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 28, pp. 417-425, 1998.

[220] D.S. Yeung and X. Sun, "Using function approximation to analyze the sensitivity of MLPs with antisymmetrix squashing activation function, *IEEE Trans. NEural Networks*, vol. 13, no. 1, pp. 34–44, 2002.

[221] Z. Yi, P.A. Heng and A.W. Fu, "Estimate of exponential convergence rate and exponential stability for neural networks", *IEEE Trans. Neural Networks*, vol. 6, no. 6, pp. 1487–1493, 1999.

[222] Z. Yi, K.K. Tan, and T.H. Lee, "Multistability analysis for recurrent neural networks with unsaturating piecewise linear transfer functions," *Neural Computation*, vol. 15, no. 3, pp. 639–662, 2003.

[223] Z. Yi and K.K. Tan, *Convergence analysis of recurrent neural networks*, vol. 13 of *Network Theory and Applications*, Kluwer Academic Publishers, Boston, 2004.

[224] S.H. Zak, V. Upatising and S. Hui, "Solving linear programming problems with neural networks: A comparative study", *IEEE Trans. Neural Networks*, vol. 6, no. 1, pp. 94–104, 1995.

[225] P. Zegers and M.K. Sundareshan. "Periodic motions, mapping ordered sequences, and training of dynamic neural networks to generate continuous and discontinuous trajectories". In *Proceedings of the IJCNN*, Como, Italy, pp. 9–14, 2000.

[226] Q. Zhao and T. Higuchi, "Evolutionary learning of nearest-neighbor MLP," *IEEE Transactions on Neural Networks*, vol. 7, pp. 762-767, 1996.

[227] Q. Zhao, "Stable on-line evolutionary learning of NN-MLP," *IEEE Transactions on Neural Networks*, vol. 8, pp. 1371-1378, 1997.

[228] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: improving the strength Pareto evolutionary algorithm," Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, May 2001.