Non-Linear Problem Solving Using Neural Networks and Genetic Algorithms

by

Junaid Ali

MS Electronic Engineering

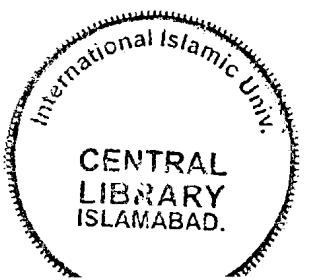
A dissertation submitted to FET IIUI in partial fulfillment of requirement for degree of Master of Science in Electronics Engineering

Department of Electronic Engineering

Faculty of Engineering and Technology

International Islamic University, Islamabad.

(2008)




بسم الله الرحمن الرحيم

Certificate of Approval

17-

It is certified that we have read the project report submitted by **Junaid Ali [~~181-FAS/MS-CS/F-04~~** FET/PHDEE/F04 **]** transferred in Electronics Engineering. It is our judgment that this report is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad for degree of MS Electronic Engineering (MSEE).

Supervisor
Dr. Ijaz Mansoor Qureshi
Dean, FET, IIU Islamabad.

External Examiner
Dr. Abdul Jalil
Associate Professor
PIEAS, Islamabad.

Internal Examiner
Dr. Tanweer Ahmed Cheema
Assistant Professor, IIU, Islamabad.

A dissertation submitted to the Department of Electronics Engineering, Faculty of Engineering and Technology, International Islamic University, Islamabad, Pakistan, as a partial fulfillment of the requirements for the award of the degree of

MS in "Electronic Engineering"

To

The Holiest man Ever Born,

PROPHET MUHAMMAD (PEACE BE UPON HIM)

&

To

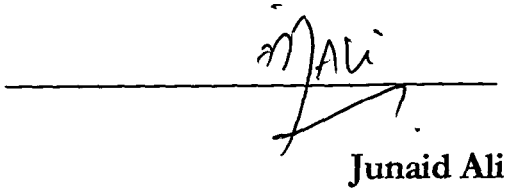
MY PARENTS

I am most indebted to my parents, whose affection has been the source of encouragement for me, and whose prayers have always been key to my success.

Declaration

I hereby declare that this research and simulation, neither as a whole nor as a part thereof, has been copied out from any source. It is further declared that I have developed this research, simulation and the accompanied report entirely on the basis of my personal effort made under the guidance of my supervisor and teachers.

If any part of this report to be copied or found to be reported, I shall standby the consequences. No portion of this work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.



Junaid Ali

~~**181-FAS/MS-CS/F-04**~~

(Transferred in MSEE)

17-FET/PHDEE/F-04

Acknowledgements

With the glorious name of **ALLAH** almighty who bestowed His blessings upon me and has made me worthy to make this report to utmost effort. I am extremely thankful to my God for the moral help, patience and courage to complete my thesis.

I acknowledge with deep appreciation, the utmost cooperation, constant encouragement and inspiring guidance of **Dr. Ijaz Mansoor Qureshi** supervisor of my thesis. Guidance and valuable suggestions by **Dr. Tanweer Ahmed Cheema** helps a lot. I am grateful to my friend **Raja Asif** who has helped me to improve my work through insightful and critical comments.

At this state of my life I don't forget the biggest moral support of my **Parents** whose prays have always remained with me.

Project in Brief

Project Title:	Non Linear Problem Solving Using Artificial Neural Network and Genetic Algorithms
Organization:	International Islamic University H-10, Islamabad
Under Taken By:	Mr . Junaid Ali
Supervised By:	Dr.Ijaz Mansoor Qurashi Dean Faculty of Engineering &Technology IIUI
External Examiner:	Dr. Abdul Jalil Associate Professor PIEAS, Islamabad.
Tool Used:	Matlab 7
System Used:	Pentium IV

Table of Contents

Chapter 1

Introduction

1.1 Introduction

Artificial neural networks (ANN) are among the newest signal-processing technologies in the electronics engineering. An Artificial Neural Network is an adaptive, most often nonlinear system that learns to perform a function (an input/output mapping) from data. The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits.

A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural property for storing experimental knowledge and making it available for use. It resembles the brain in two respects

1. The Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weight, are used to store the acquired knowledge.

The use of the neural networks offers the following useful properties and capabilities:

1. **Nonlinearity:** An artificial neuron can be linear and non linear. A neural network made from the no linear neurons interconnection is itself non linear. Nonlinearity is a highly important property particularly if the underlying physical mechanism responsible for generation of the input signals (speech signal, face recognition signal etc) are inheritally non linear.
2. **Input/ Output Mapping:** Supervised learning is one of the most popular paradigm in learning that involves modification of the synaptic weights of neural network by applying a set of labeled training samples, it consist of unique input signal and desired corresponding output signal. Supervised learning paradigm suggests a close analogy between the input output mapping performed by the neural network and nonparametric statistical interference.
3. **Adaptively:** Neural network have a built-in capability to adopt their synaptic weights to change in the surrounding environment, if neural network is trained on a specific environment can be trained by itself with the minor changes. Their synaptic weights are modified by itself with the changes in the environment like it does in real time. This adaptability does not always robust; I mean it can go very opposite as well.
4. **Evidential Response:** Neural networks particularly when implemented of pattern recognition systems can be designed to provide information not only about which particular

pattern to select, but also about confidence in the decision making. This latter information can reject the ambiguous patterns.

5. **Contextual Information:** Knowledge is represented by the very structure and activation state of neural network. Every neuron in the network is potentially affected by the global activity of all other neurons in the network. Consequently, contextual information is dealt with naturally by a neural network.
6. **Fault Tolerance:** Implementing a neural network on the hardware has the potential to be inherently fault tolerant or capable of robust computation, in the sense that its performance degrades gracefully under adverse operating conditions. In principle a neural network exhibits a graceful degradation in performance rather than catastrophic failure.

From the various existed activation function; I use the linear activation function on the input and output layer of Differential Equation (DE) neural network, while log sigmoid activation function is used for mathematical linear mapping of non linear differential equation. The architecture is simple feed forward with the real numbers as the bias from the range -5 to 5, with some hidden layers. Details are their in rest of the chapters. Weights on the communication links are adjusted and upgraded by the genetic algorithms and pattern search which forms it as hybrid intelligent algorithm.

Unsupervised learning is used to train the system is discussed in chapter 3. Learning has a primary significance in the neural networks which is supposed to be trained according to the environment. The performance can be improved by the learning of the neural net. A neural network is trained due to the synaptic weights on the communication links and due to the bias, which keep on performing better by the learning on each iterative process about its environment that is the key point in an intelligent system.

"Learning is a process by which the free parameters of a neural network are adopted through a process of simulation by the environment in which the network is embedded. This type of the learning is determined by the manner in which the parameter changes take place."

Genetic algorithms are global search, an optimization technique model from the natural genetics, Exploring search space by incorporating a set of candidate solution in parallel. A genetic algorithm maintains a population of candidate solutions where each of the solution is usually coded as a binary string called a chromosome. A chromosome also referred as a genotype – encode a parameter set i.e. a candidate solution. For a set of variables being

optimized, each encoded parameter in a chromosome is also called as gene. A decode parameter set is called phenotype. Our chromosome set contains the real values. A set of chromosome forms a population which is evaluated and ranked by a fitness evolution function. The evolution form from one generation to the next one evolves mainly three steps.

i) Creation

ii) Reproduction

iii) Mutation & recombination

Moreover the genetic algorithms alone are not so helpful for the searching for a global minimum; for this purpose I have incorporated the pattern search to form the hybrid intelligent algorithm whose convergence towards the answer is remarkable.

1.2 Literature Survey

Solutions of differential equations arise in a wide variety of engineering applications in electromagnetic, signal processing, computational field dynamics, embedded systems, digital communication etc. These equations are typically solved using either analytical methods or numerical methods. Analytical solution method are however feasible only for simple geometries, which limit their applicability, in most practical problems with complex boundary conditions, numerical methods are required in order to obtain a reasonable solution. As the artificial neural network can approximate any continuous function more efficiently than any linear combination of fixed basic functions. However their learning structure is not yet classified due to the non linearity, abnormality and complexity.

1.3 Previous Work

In nineteen sixties various scientists used unsuccessfully the feed forward neural network in the field of signal processing and computation. In 1971 people use successfully the feed forward neural network as an application in digital computers with the KBM methods. Previously people have applied the neural network on motion control methods for bipedal humanoid locomotion a nd their rectilinear path. Robot locomotion is a multi-objective

problem in motion controls. To optimization of the parameters simultaneously cause explosion of search space and timings as well. The same techniques are used to measure the braking torques in the motion of the bodies; these applications involve solution of complex ordinary differential equations. Hydro system scheduling involves complex non linear equations that were solved using the artificial neural networks (ANN).

1.4 Recent Research

Artificial neural network have been recently shown that they can be successfully incorporated into solution methods for ordinary differential equations both for ordinary differential equation and partial differential equations. These solution methods rely on the function approximation capabilities of the feed forward neural network. A recent method for solving differential equation using feed forward neural network was applied to a non steady catalyst solid gas reactor. Due to ANN universal capabilities of approximation, it is possible to postulate them as a solution for a given DE problem that defines an unsupervised error. These problems are also solvable by traditional numerical methods which are slow and have more space complexity. Recently Riccati differential equation has been solved by using multilayer back propagation neural network.

Unsupervised neural network is also used to solve non linear complex Schrodinger's equation in 2001. Lots of recent research has been done in this regard from which the versatility and scope of the thesis is clear as a reference, I have mentioned in the above passage.

1.5 Problem statement

Analytical methods to solve the non linear differential equation are very complex and tedious so analytical methods are used to solve these equations but the time and space complexity is very high and slow as well. Previously researchers solve the non linear problems by transforming into the linear samples. The problem comes for the computation of parameters to be trained for particular task, different strategies are used previously like LMS, sparse tree etc. But no one has calculated these parameters by direct method as the optimal parameters are unique and can be directly calculated. A function approximate the differential equation automatically from the learning samples, but the learning methodology will be unsupervised

in order to make our system versatile. Feed forward DENN (differential equation neural network) will produce the better results quantitatively with a MSE $\sim 10^{-5}$.

1.6 Objective of the project

This research project aims at solving the non linear differential equation using neural network and hybrid intelligent algorithms. The objective is to process and analyze the complex non linear systems. The theme of the project is to design and develop new and advance algorithms and techniques to tackle various problems occur during the adjustment of the link weights, during the learning and their up gradation. The focus is to get the best chromosome set which approximate the smallest possible MSE or number of generations, which ever comes earlier. This project aims to the advancement in the field of signal processing in digital electronics.

1.7 Scope and Plan of the Project

Artificial neural network is one of the growing fields in digital electronics which can tackle the 80% of the non linear problems of the world, due to its versatility and compatibility of ANN using the heuristics and non sequential procedures intelligently. The system based on the research done in this thesis will play its major role in providing the help to mathematicians and scientists to calculate/approximate the solution of non linear complex differential equations like Bernoulli, Van der Pol, Weissinger's etc.

The whole project is divided into four categories:

- Neural Network Architecture

  Our NN architecture is feed forward neural network

- Learning

  We perform the unsupervised learning

- Hybrid Intelligent Algorithm

  To avoid the clasping in the local minima and to search global minima we use genetic algorithm with pattern search

- Exact Solution

To clarify over analytical method we find the complex exact solutions provided in the standard texts.

1.8 Organization of Thesis

Chapter 1 have a very brief introduction of the neural networks, learning and intelligent algorithms, with a survey about the project is taken in this regard is also the part of this chapter. Chapter2 is related to the neural network architecture techniques and activations function in detail. Supervised and unsupervised learning to train the network is in the chapter 3. Genetic algorithms and hybrid intelligent algorithms are one of the important segments in the learning of synaptic weights on the communication links of neural network is in chapter 4. Discussion of Non linear differential equations is the part of chapter 5. Proposed system is one of the important part of thesis in which the architecture and layers of the non linear differential equation have been discussed with the unsupervised learning rules. These non linear problems are solved and results are approximated with the neural network approach incorporated with genetic algorithms is discussed in chapter 6. Last chapter contain the results with the best chromosomes and MSE obtained by proposed method run for 400 generations on a finite real domain of time.

Chapter 2

Artificial Neural Network

2.1 What is a Neural Network?

An artificial neural network is a system based on the operation of biological neural networks, in other words, an emulation of biological neural system. The key elements of this paradigm are the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems.

Artificial neural networks (ANN) are among the newest signal-processing technologies in the electronics engineering. An artificial neural network is an adaptive, most often nonlinear system that learns to perform a function (an input/output mapping) from data. Adaptive means that the system parameters are changed during operation, normally called the training phase. After the training phase the Artificial Neural Network parameters are fixed and the system is deployed to solve the problem at hand (the testing phase). The Artificial Neural Network is built with a systematic step-by-step procedure to optimize a performance criterion or to follow some implicit internal constraint, which is commonly referred to as the learning rule

2.2 Historical Background:

Neural network simulations appear to be a recent development. However, this field was established before the advent of computers, and has survived at least one major setback and several eras. Many important advances have been boosted by the use of inexpensive computer emulations. Following an initial period of enthusiasm, the field survived a period of frustration and disrepute. During this period when funding and professional support was minimal, important advances were made by relatively few researchers. These pioneers were able to develop convincing technology which surpassed the limitations identified by Minsky and Papert. Minsky and Papert, published a book (in 1969) in which they summed up a general feeling of frustration (against neural networks) among researchers, and was thus accepted by most without further analysis. Currently, the neural network field enjoys a resurgence of interest and a corresponding increase in funding. The first artificial neuron was produced in 1943 by the neurophysiologist Warren McCulloch and the logician Walter Pits. But the technology available at that time did not allow them to do too much in this field.

2.3 The Biological Model

Artificial neural networks emerged after the introduction of simplified neurons by McCulloch and Pitts in 1943 (McCulloch & Pitts, 1943). These neurons were presented as models of biological neurons and as conceptual components for circuits that could perform computational tasks. The basic model of the neuron is founded upon the functionality of a biological neuron. "Neurons are the basic signaling units of the nervous system" and "each neuron is a discrete cell whose several processes arise from its cell body".

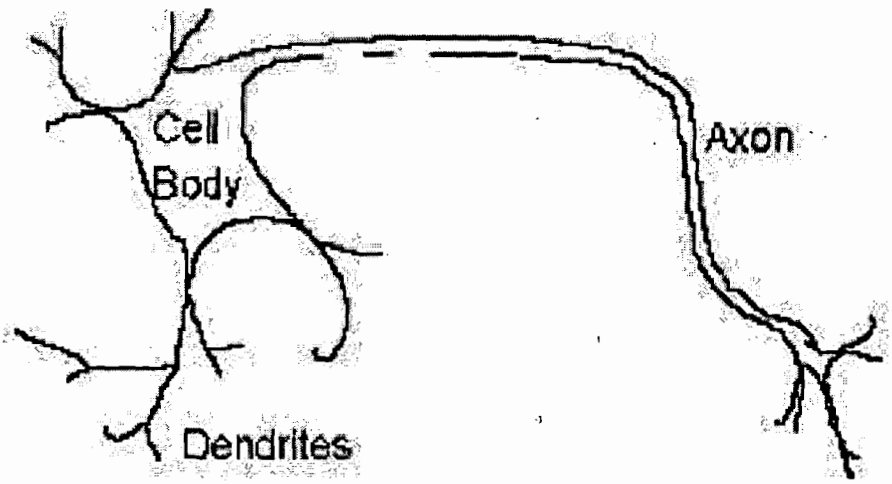


Figure 2.1 Biological model of neuron

The neuron has four main regions to its structure. The cell body, or soma, has two offshoots from it, the dendrites, and the axon, which end in presynaptic terminals. The cell body is the heart of the cell, containing the nucleus and maintaining protein synthesis. A neuron may have many dendrites, which branch out in a treelike structure, and receive signals from other neurons. A neuron usually only has one axon which grows out from a part of the cell body called the axon hillock. The axon conducts electric signals generated at the axon hillock down its length. These electric signals are called action potentials. The other end of the axon may split into several branches, which end in a presynaptic terminal. Action potentials are the electric signals that neurons use to convey information to the brain. All these signals are

identical. Therefore, the brain determines what type of information is being received based on the path that the signal took. The brain analyzes the patterns of signals being sent and from that information it can interpret the type of information being received. Myelin is the fatty tissue that surrounds and insulates the axon. Often short axons do not need this insulation. There are uninsulated parts of the axon. These areas are called Nodes of Ranvier. At these nodes, the signal traveling down the axon is regenerated. This ensures that the signal traveling down the axon travels fast and remains constant (i.e. very short propagation delay and no weakening of the signal). The synapse is the area of contact between two neurons. The neurons do not actually physically touch. They are separated by the synaptic cleft, and electric signals are sent through chemical 13 interaction. The neuron sending the signal is called the presynaptic cell and the neuron receiving the signal is called the postsynaptic cell. The signals are generated by the membrane potential, which is based on the differences in concentration of sodium and potassium ions inside and outside the cell membrane. Neurons can be classified by their number of processes (or appendages), or by their function. If they are classified by the number of processes, they fall into three categories. Unipolar neurons have a single process (dendrites and axon are located on the same stem), and are most common in invertebrates. In bipolar neurons, the dendrite and axon are the neuron's two separate processes. Bipolar neurons have a subclass called pseudo-bipolar neurons, which are used to send sensory information to the spinal cord. Finally, multipolar neurons are most common in mammals. Examples of these neurons are spinal motor neurons, pyramidal cells and Purkinje cells (in the cerebellum). If classified by function, neurons again fall into three separate categories. The first group is sensory, or afferent, neurons, which provide information for perception and motor coordination. The second group provides information (or instructions) to muscles and glands and is therefore called motor neurons. The last group, interneuronal, contains all other neurons and has two subclasses. One group called relay or projection interneurons have long axons and connect different parts of the brain. The other group called local interneurons are only used in local circuits.

2.4 The Mathematical Model

When creating a functional model of the biological neuron, there are three basic components of importance. First, the synapses of the neuron are modeled as weights. The strength of the connection between an input and a neuron is noted by the value of the weight. Negative weight values reflect inhibitory connections, while positive values designate excitatory connections [Haykin]. The next two components model the actual activity within the neuron cell. An adder sums up all the inputs modified by their respective weights. This activity is referred to as linear combination. Finally, an activation function controls the amplitude of the output of the neuron. An acceptable range of output is usually between 0 and 1, or -1 and 1.

Mathematically, this process is described in the figure

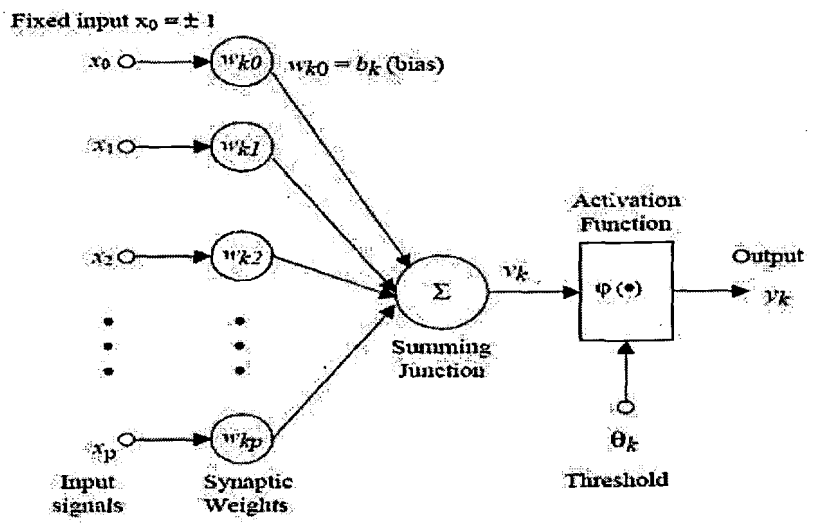


Figure2.2 Mathematical Model

From this model the interval activity of the neuron can be shown to be:

$$v_k = \sum_{j=1}^{p} w_{kj} x_j \qquad 2.1$$

The output of the neuron y_k would therefore be the outcome of some activation function on the value of v_k.

2.5 Activation Functions

There are a number of common activation functions in use with neural networks. This is not an exhaustive list.

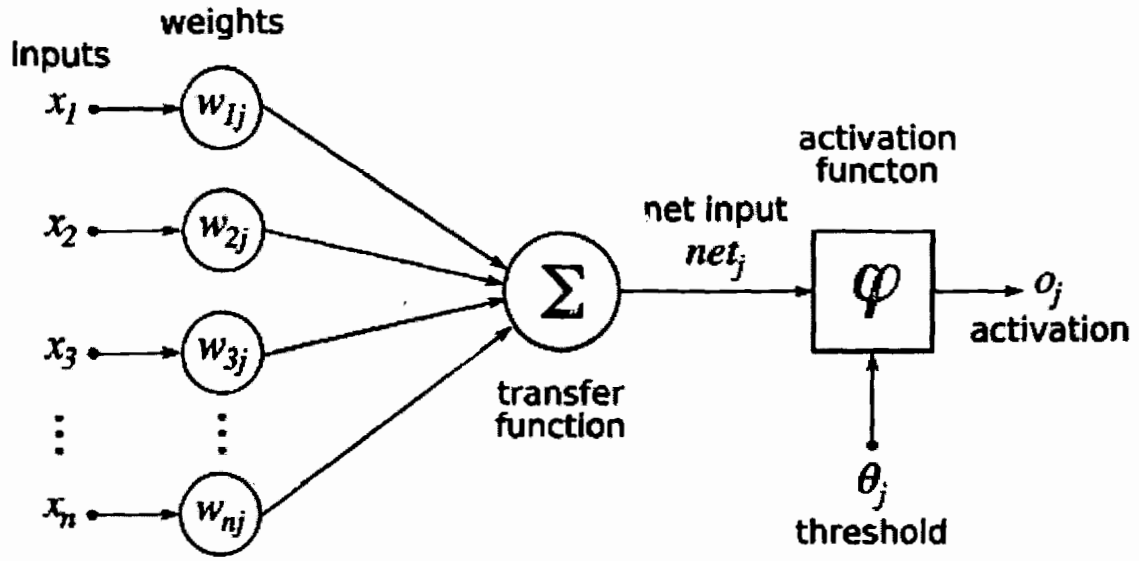


Figure2.3 A simple neural network

2.5.1 Step Function

A step function is a function like that used by the original Perceptron. The output is a certain value, A_1, if the input sum is above a certain threshold and A_0 if the input sum is below a certain threshold. The values used by the Perceptron were $A_1 = 1$ and $A_0 = 0$. These kinds of step activation functions are useful for binary classification schemes. In other words, when we want to classify an input pattern into one of two groups, we can use a binary classifier with a step activation function. Another use for this would be to create a set of small **feature identifiers**. Each identifier would be a small network that would output a 1 if a particular input feature is present and a 0 otherwise. Combining multiple feature detectors into a single network would allow a very complicated clustering or classification problem to be solved.

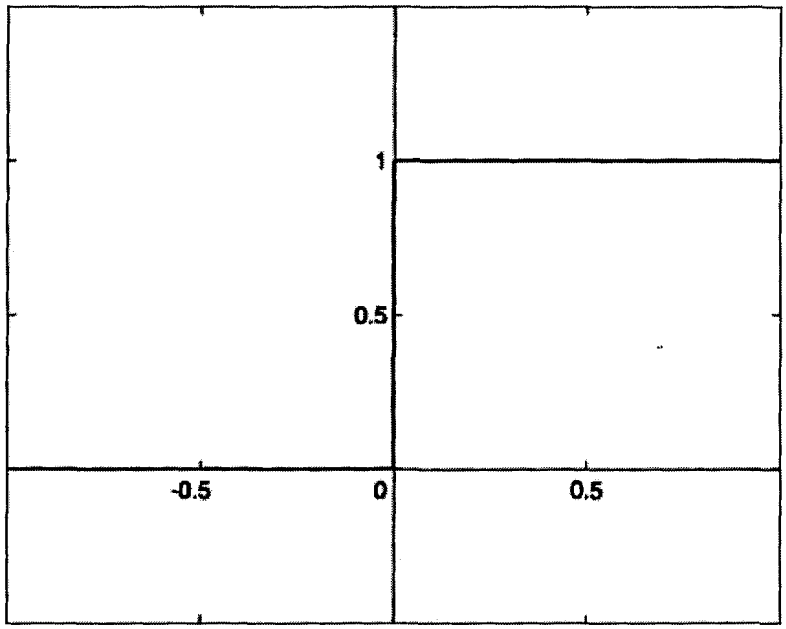


Figure2.4 Step function

2.5.2 Linear Combination

A linear combination is where the weighted sum input of the neuron plus a linearly dependant bias becomes the system output. Specifically:

$$y = \zeta + b \qquad 2.2$$

In these cases, the sign of the output is considered to be equivalent to the 1 or 0 of the step function systems, which enables the two methods, be to equivalent if

$$\theta = -b \qquad 2.3$$

2.5.3 Continuous Log-Sigmoid Function

A log-sigmoid function, also known as a logistic function, is given by the relationship:

$$\prod(t) = \frac{1}{1+e^{-\beta t}} \qquad 2.4$$

Where β is a slope parameter. This is called the log-sigmoid because a sigmoid can also be constructed using the hyperbolic tangent function instead of this relation. In that case, it would be called a tan-sigmoid. Here, we will refer to the log-sigmoid as simply "sigmoid". The sigmoid has the property of being similar to the step function, but with the addition of a region of uncertainty. Sigmoid functions in this respect are very similar to the input-output relationships of biological neurons, although not exactly the same. Below is the graph of a sigmoid function.

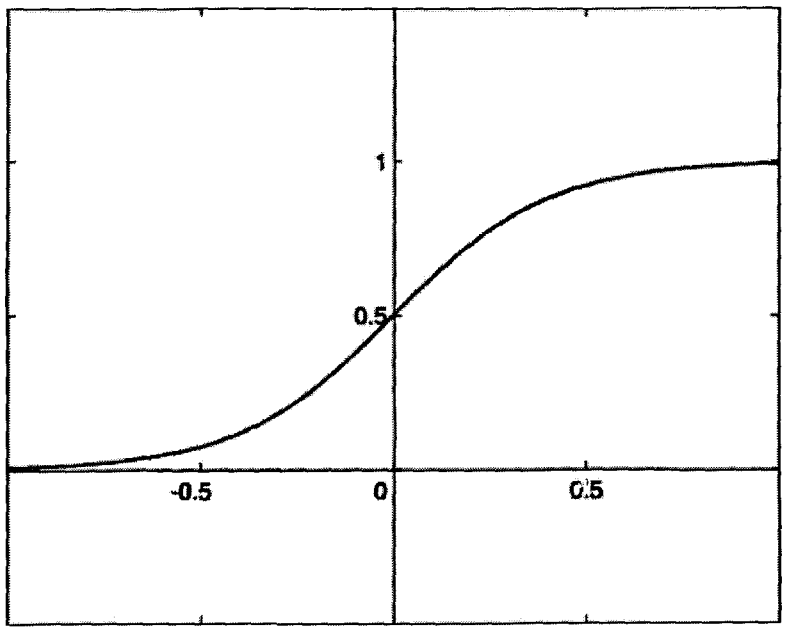


Figure 2.5 Log sigmoid activation function

Sigmoid functions are also prized because their derivatives are easy to calculate, which is helpful for calculating the weight updates in certain training algorithms. The derivative is given by:

$$\frac{d\prod(t)}{dt} = \prod(t)[1-\prod(t)] \qquad 2.5$$

2.5.4 Continuous Tan-Sigmoid Function

2.5.5 Softmax Function

The softmax activation function is useful predominantly in the output layer of a clustering system. Softmax functions convert a raw value into a posterior probability. This provides a measure of certainty. The softmax activation function is given as:

$$y_i = \frac{e^{\varsigma_i}}{\sum_{j \in L} e^{\varsigma_j}} \qquad 2.6$$

L is the set of neurons in the output layer.

2.6 Architecture of Neural Networks

Artificial Neural Networks are classified as either Feed-Forward, or Feedback networks.

2.6.1 Single Layer Feed Forward Network:

A Feed-Forward network allows traffic in one direction only, i.e. from input to output. There are no loops, and each output from a particular layer does not affect other neurons on the same layer.

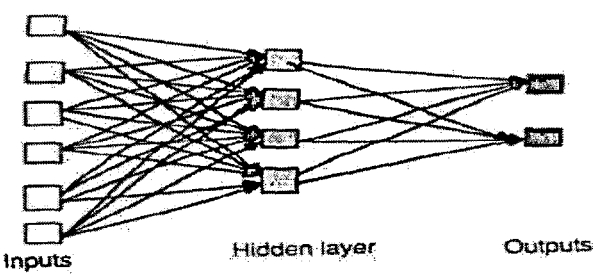


Figure 2.6 Single layer feed forward network

2.6.2 Multilayer Feed Forward Networks

The package supports FF neural networks with any number of hidden layers and any number of neurons (hidden neurons) in each layer. In Figure 2.6 a multi-output FF network with two hidden layers is shown.

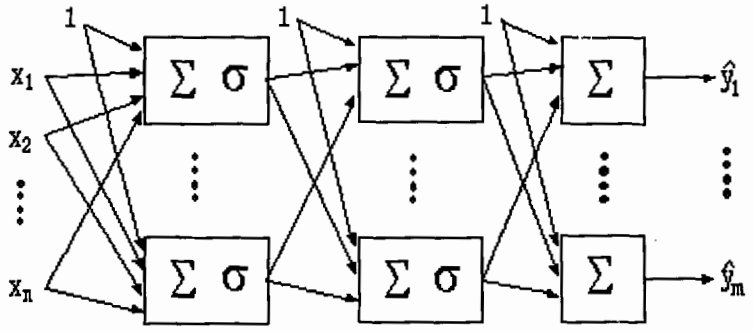


Figure 2.7 A multi-output feed forward network with two hidden layers.

The number of layers and the number of hidden neurons in each hidden layer are user design parameters. The general rule is to choose these design parameters so that the best possible model with as few parameters as possible is obtained. This is, of course, not a very useful rule, and in practice you have to experiment with different designs and compare the results, to find the most suitable neural network model for the problem at hand. For many practical applications, one or two hidden layers will suffice.

2.6.3 Feed Back Neural Networks:

Feedback architectures allow loops in the network. This means that the output from one neuron can be sent to neurons that have already fired. Obviously, this architecture is much more complex, but it is also a more accurate representation to biological models. It is also much more powerful than the existed feed forward neural networks and recurrent neural networks.

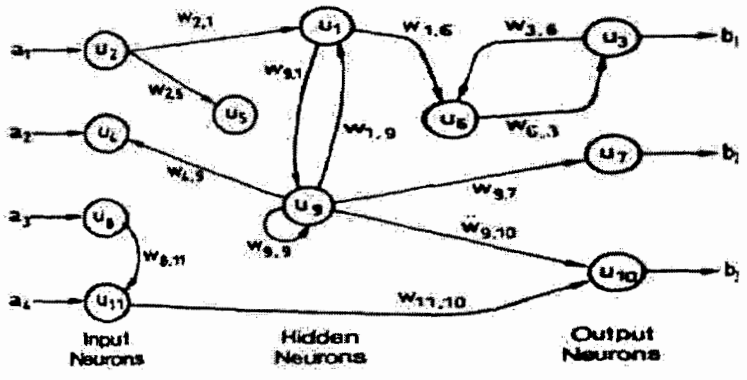


Figure 2.8 Feed back neural network

In either structure, 3 distinct layers exist: an input layer, a "hidden" layer, and an output layer.

The input layer represents the raw data that is fed into the system. The hidden layer does the "work" in a neural network and its output is generated based on the weights of the links it receives input from, and the threshold of each neuron. This output in turn is interpreted based on the links between the hidden layer and the output layer and creates a final output. In a single layer system, the process ends here. However, there are multiple layer systems which then use this output as input into another layer, essentially nesting several neural networks amongst each other. Again, this is a more complex implementation, but more closely resembles natural neural networks.

To learn more about the history of neural networks

2.6.4 Recurrent Neural Networks

A **recurrent neural network** is a neural network where the connections between the units form a directed cycle. Recurrent neural networks must be approached differently from feed forward neural networks, both when analyzing their behavior and training them. Recurrent neural networks can also behave chaotically.

2.7 Applications of Neural Networks:

ANN are also used in the following specific paradigms: recognition of speakers in communications; diagnosis of hepatitis; recovery of telecommunications from faulty software; interpretation of multi meaning Chinese words; undersea mine detection; texture analysis; three-dimensional object recognition; hand-written word recognition; and facial recognition.

2.7.1 Neural Networks in Medicine

Artificial Neural Networks (ANN) is currently a 'hot' research area in medicine and it is believed that they will receive extensive application to biomedical systems in the next few

years. At the moment, the research is mostly on modeling parts of the human body and recognizing diseases from various scans (e.g. cardiograms, CAT scans, ultrasonic scans, etc.).

Neural networks are ideal in recognizing diseases using scans since there is no need to provide a specific algorithm on how to identify the disease. Neural networks learn by example so the details of how to recognize the disease are not needed. What is needed is a set of examples that are representative of all the variations of the disease. The quantity of examples is not as important as the 'quantity'. The examples need to be selected very carefully if the system is to perform reliably and efficiently.

2.7.2 Modeling and Diagnosing the Cardiovascular System

Neural Networks are used experimentally to model the human cardiovascular system. Diagnosis can be achieved by building a model of the cardiovascular system of an individual and comparing it with the real time physiological measurements taken from the patient. If this routine is carried out regularly, potential harmful medical conditions can be detected at an early stage and thus make the process of combating the disease much easier.

A model of an individual's cardiovascular system must mimic the relationship among physiological variables (i.e., heart rate, systolic and diastolic blood pressures, and breathing rate) at different physical activity levels. If a model is adapted to an individual, then it becomes a model of the physical condition of that individual. The simulator will have to be able to adapt to the features of any individual without the supervision of an expert. This calls for a neural network.

Another reason that justifies the use of ANN technology is the ability of ANNs to provide sensor fusion which is the combining of values from several different sensors. Sensor fusion enables the ANNs to learn complex relationships among the individual sensor values, which would otherwise be lost if the values were individually analyzed. In medical modeling and diagnosis, this implies that even though each sensor in a set may be sensitive only to a specific physiological variable, ANNs are capable of detecting complex medical conditions by fusing the data from the individual biomedical sensors.

2.7.3 Electronic Noses

ANNs are used experimentally to implement electronic noses. Electronic noses have several potential applications in telemedicine. Telemedicine is the practice of medicine over long distances via a communication link. The electronic nose would identify odours in the remote surgical environment. These identified odours would then be electronically transmitted to another site where a door generation system would recreate them. Because the sense of smell can be an important sense to the surgeon, telesmell would enhance telepresent surgery.

2.7.4 Instant Physician

An application developed in the mid-1980s called the "instant physician" trained an auto associative memory neural network to store a large number of medical records, each of which includes information on symptoms, diagnosis, and treatment for a particular case. After training, the net can be presented with input consisting of a set of symptoms; it will then find the full stored pattern that represents the "best" diagnosis and treatment.

2.8 Advantages

- A neural network can perform tasks that a linear program can not.
- When an element of the neural network fails, it can continue without any problem by their parallel nature.
- A neural network learns and does not need to be reprogrammed.
- It can be implemented in any application.
- An ability to learn how to do tasks based on the data given for training or initial experience.
- It can be implemented without any problem.

2.9 Disadvantages

- The neural network needs training to operate.
- The architecture of a neural network is different from the architecture of microprocessors therefore needs to be emulated.

- Requires high processing time for large neural network

2.10 Perceptrons

Perceptrons are neural nets that change with "experience," using an error-correction rule designed to change the weights of each response unit when it makes erroneous responses to stimuli that are presented to the network. A simple perceptron is one in which the associator units are not interconnected, which means that it has no short-term memory. (If such connections are present, the perceptron is called cross coupled. A cross-coupled perceptron may have multiple layers and loops back from an "earlier" to a "later" layer.) If the associator units feed the pattern $x = (x_1 + x_2 + x_3 \ldots\ldots\ldots\ldots x_n)$ to the output unit, then the response of that unit will be to provide the pattern discrimination.

2.11 Neural Coding

There are various existed methods for neural coding.

- Adaptive Spike Coding
- Integrated-And-Fire Neurons and Networks
- Localized Versus Distributed Representation
- Motor Cortex: Coding And Decoding of Directional
- Operations
- Optimal Sensory Encoding
- Population Codes
- Rate Coding And Signal Processing
- Sensory Coding and Information Transmission
- Sparse Coding in the Primary Cortex
- Synchronization, Binding and Expectancy
- Synfire Chains

Chapter 3

Learning

A **neural network** has to be configured such that the application of a set of inputs produces (either 'direct' or via a relaxation process) the desired set of outputs. Various methods to set the strengths of the connections exist. One way is to set the weights explicitly, using a priori knowledge. Another way is to **'train' the neural network** by feeding it teaching patterns and letting it change its weights according to some learning rule.

We can categories the learning situations in two distinct sorts. These are:

3.1 Supervised learning

Supervised learning is also called associative learning in which the network is trained by providing it with input and matching output patterns. These input-output pairs can be provided by an external teacher, or by the system which contains the neural network (self-supervised). During the learning process global information may be required. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning.

An important issue concerning supervised learning is the problem of error convergence, i.e. the minimization of error between the desired and computed unit values. The aim is to determine a set of weights which minimizes the error. One well-known method, which is common to many learning paradigms, is the least mean square (LMS) convergence.

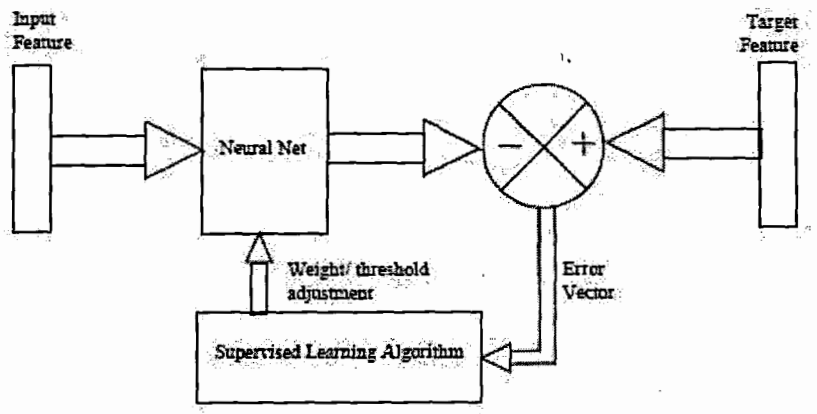


Figure 3.1 Supervised learning procedure

3.2 Unsupervised learning

It is also called Self-organization in which an (output) unit is trained to respond to clusters of pattern within the input. In this paradigm the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which the patterns are to be classified; rather the system must develop its own representation of the input stimuli. uses no external teacher and is based upon only local information. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. Paradigms of unsupervised learning are Hebbian learning and competitive learning. Ano2.2 From Human Neurones to Artificial Neuronesther aspect of learning concerns the distinction or not of a separate phase, during which the network is trained, and a subsequent operation phase. We say that a neural network learns off-line if the learning phase and the operation phase are distinct. A neural network learns on-line if it learns and operates at the same time. Usually, supervised learning is performed off-line, whereas unsupervised learning is performed on-line.

3.3 Reinforcement Learning

This type of learning may be considered as an intermediate form of the above two types of learning. Here the learning machine does some action on the environment and gets a feedback response from the environment. The learning system grades its action good (rewarding) or bad (punishable) based on the environmental response and accordingly adjusts its parameters. Generally, parameter adjustment is continued until an equilibrium state occurs, following which there will be no more changes in its parameters. The self organizing neural learning may be categorized under this type of learning.

3.4 Delta Rule

Delta rule is also known as three other name which are as following:

- Adaline Rule
- Widrow-Hoff Rule
- Least Mean Squares (LMS) Rule

3.4.1Change from Perceptron:

- Replace the step function in the with a continuous (differentiable) activation function, e.g linear
- For classification problems, use the step function only to determine the class and not to update the weights.
- Note: this is the same algorithm we saw for regression. All that really differs is how the classes are determine

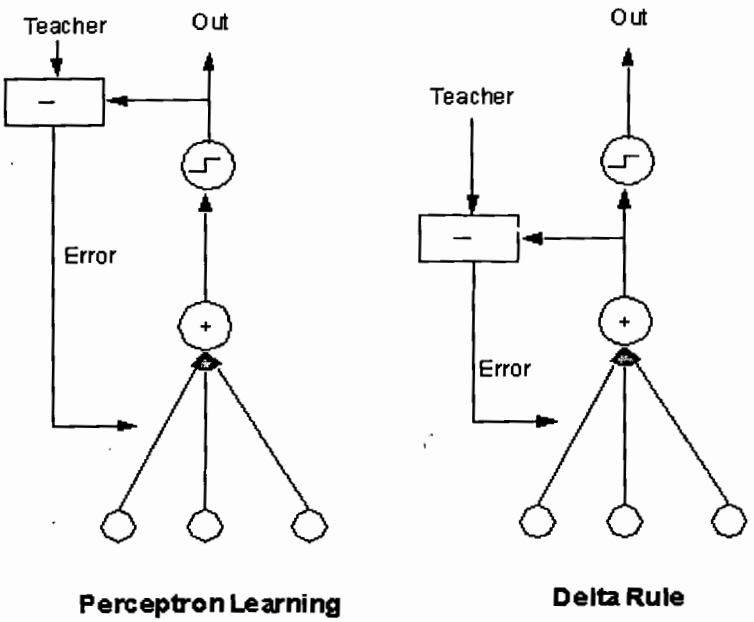


Figure 3.2 Learning through delta rule

Delta Rule: Training by Gradient Descent Revisited

Construct a cost function E that measures how well the network has learned. For example

$$E = \frac{1}{2}\sum_{i=1}^{n}(t_i - y_i)^2 \qquad 3.1$$

(one output node)

where

n = number of examples

t_i = desired target value associated with the i-th example

y_i = output of network when the i-th input pattern is presented to network

- To train the network, we adjust the weights in the network so as to decrease the cost (this is where we require differentiability). This is called gradient descent.

3.4.2 Algorithm

- Initialize the weights with some small random value
- Until E is within desired tolerance, update the weights according to
  where E is evaluated at W(old), μ is the learning rate
  and the gradient is

$$w(new) = w(old) - \mu\frac{\partial E}{\partial w} \qquad 3.2$$

$$E = \frac{1}{2}\sum_{i=1}^{n}(t_i - y_i)^2 \qquad 3.3$$

$$\frac{\partial E}{\partial W} = \sum_{i=1}^{n}(t_i - y_i)x_i \qquad 3.4$$

3.5 Nonlinear Compression Techniques

Two layer networks perform a projection of the data onto a linear subspace. In this case, the encoding and decoding portions of the network are really single layer linear networks. This works well in some cases. However, many datasets lie on lower dimensional subspaces that are not linear.

3.5.1 Example

A helix is 1-D, however, it does not line on a 1-D linear subspace.

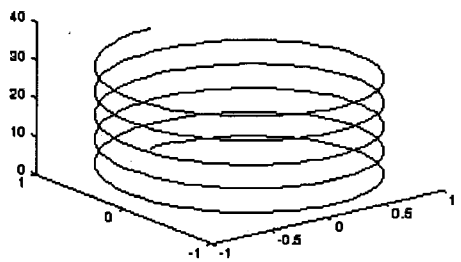


Figure 3.3 1-D linear subspace

To solve this problem we can let the encoding and decoding portions each be multilayer networks. In this way we obtain nonlinear projections of the data.

3.5.2 5-Layer Networks

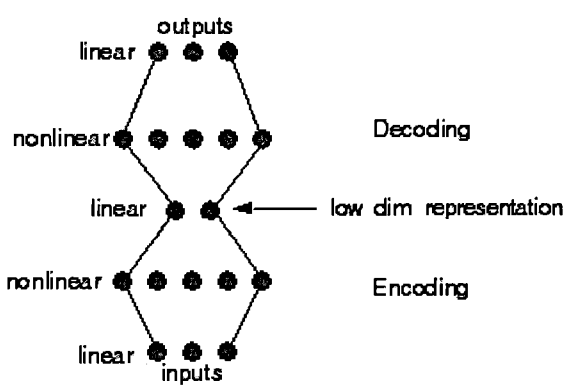


Figure 3.4 5-layer networks

3.5.3 Example: Hemisphere

(From *Fast Nonlinear Dimension Reduction*, Nanda Kambhatla,NIPS93)

Compressing a hemisphere onto 2 dimensions

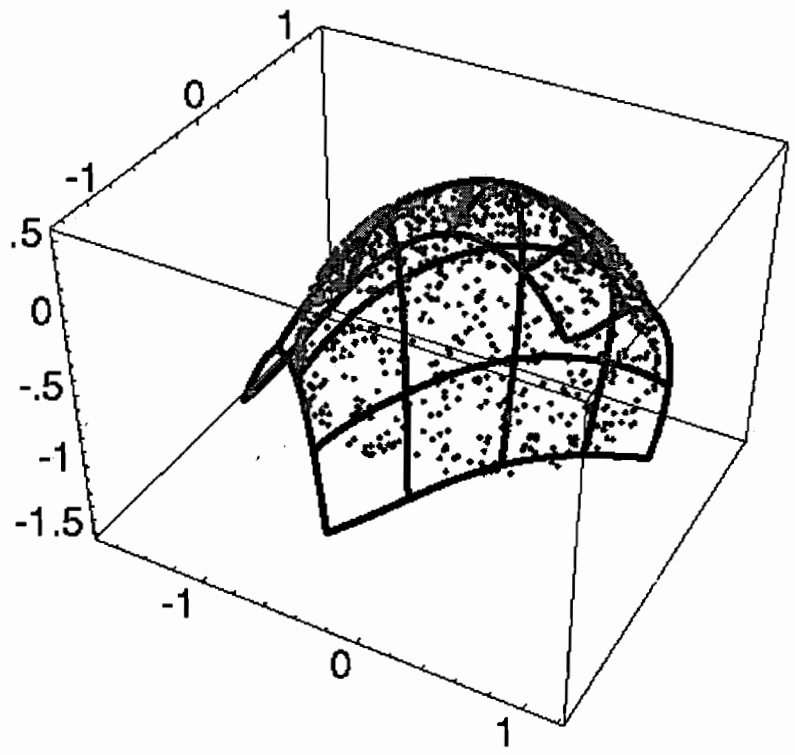


Figure 3.5 Hemisphere on 2-dimension

4 Example: Faces

(from *Fast Nonlinear Dimension Reduction*, Nanda Kambhatla,NIPS93)

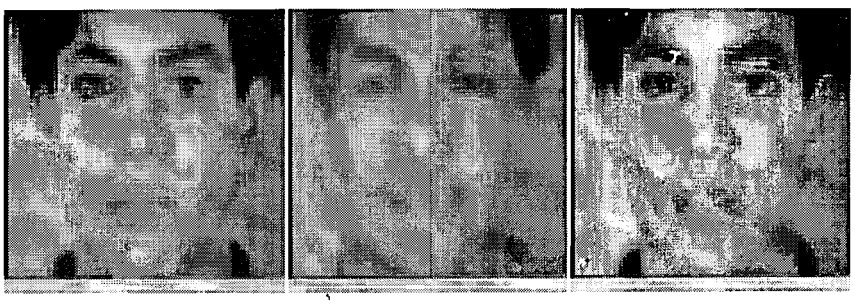
In the examples below, the original images consisted of 64x64 8-bit/pixel grayscale images. The first 50 principal components were extracted to from the image you see on the left. This was reduced to 5 dimensions using linear PCA to obtain the image in the center. The same

imageon the left was also reduced to 5 dimensions using a 5-layer (50-40-5-40-50) network to produce the image on the right.

Figure 3.6 Face1

50 principal components 5 principal components 5 nonlinear components

Figure 3.7 Face 2



3.6 Bayesian Learning for Neural Networks:

Artificial ``neural networks" are now widely used as flexible models for regression and classification applications, but questions remain regarding what these models mean, and how they can safely be used when training data is limited. *Bayesian Learning for Neural Networks* shows that Bayesian methods allow complex neural network models to be used without fear of the ``overfitting" that can occur with traditional neural network learning methods. Insight into the nature of these complex Bayesian models is provided by a theoretical investigation of the priors over functions that underlie them. Use of these models in practice is made possible using Markov chain Monte Carlo techniques. Both the theoretical and

computational aspects of this work are of wider statistical interest, as they contribute to a better understanding of how Bayesian methods can be applied to complex problems

3.7 Kohonen's Self-Organizing Map (SOM):

Kohonon's SOMs are a type of unsupervised learning. The goal is to discover some underlying structure of the data. However, the kind of structure we are looking for is very different than, say, PCA or vector quantization.

Kohonen's SOM is called a topology-preserving map because there is a topological structure imposed on the nodes in the network. A topological map is simply a mapping that preserves neighborhood relations

3.7.1Algorithm for Kohonon's Self Organizing Map:

- Assume output nodes are connected in an array (usually 1 or 2 dimensional)
- Assume that the network is fully connected - all nodes in input layer are connected to all nodes in output layer.
- Use the competitive learning algorithm as follows:

- Randomly choose an input vector x
- Determine the "winning" output node i, where w_i is the weight vector connecting the inputs to output node.
- Note: the above equation is equivalent to $w_i x \geq w_k x$ only if the weights are normalized.

$$|w_i - x| \leq |w_k - x| \forall k \qquad 3.5$$

- Given the winning node i, the weight update is

$$w_k(new) = w_k(old) + \mu N(i,k)(x - w_k) \qquad 3.6$$

where, $N(i,k)$ is called the neighborhood function that has value 1 when $i = k$ and falls off with the distance $|r_k - r_i|$ between units i and k in the output array. Thus, units close to the winner as well as the winner itself, have their weights updated appreciably. Weights associated with far away output nodes do not change significantly. It is here that the topological information is supplied. Nearby units receive similar updates and thus end up responding to nearby input pattems.

The above rule drags the weight vector w_i and the weights of nearby units towards the input x.

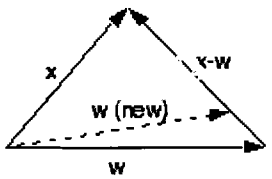


Figure 3.8 Nearby input pattems

Up until now we have discussed how to train nets given a training set of input and target values. The target value is often called the teacher signal because it represents the "right answer". i.e. what the output of the net should be. Training with a teacher signal is called"**Supervised learning**".

We can also train nets on inputs where there is no teacher signal. The purpose might be to

- discover underlying structure of the data
- encode the data
- compress the data
- transform the data

This kind of learning is called **unsupervised learning** because there is no explicit teacher signal.

3.7.2 Examples of unsupervised learning

- Hebbian learning

$$w(t+1) = w(t) + \eta y(t) x(t) \qquad 3.7$$

This moves w toward infinity in the direction of the eigenvector with largest Eigen value of the correlation matrix

A more stable version is Oja's rule

$$w(t+1) = w(t) + \eta(x(t) - y(t)w(t))y(t) \qquad 3.8$$

- principal component analysis
- competitive learning
- vector quantization

3.8 Active Learning

There are situations in which unlabeled data is abundant but labeling data is expensive. In such a scenario the learning algorithm can actively query the user/teacher for labels. This type of supervised learning is called active learning. Since the learner chooses the examples, the number of examples to learn a concept can often be much lower than the number required in normal supervised learning. With this approach there is a risk that the algorithm might focus on unimportant or even invalid examples.

3.9 Approaches and algorithms

- Analytical learning
- Artificial neural network
- Back propagation
- Boosting
- Bayesian statistics
- Case-based reasoning
- Decision tree learning
- Inductive logic programming

- Gaussian process regression
- Learning Automata
- Minimum message length (decision trees, decision graphs, etc.)
- Naive bayes classifier
- Nearest Neighbor Algorithm
- Probably approximately correct learning (PAC) learning
- Ripple down rules, a knowledge acquisition methodology
- Symbolic machine learning algorithms
- Sub symbolic machine learning algorithms
- Support vector machines
- Random Forests
- Ensembles of Classifiers
- Ordinal Classification
- Data Pre-processing
- Handling imbalanced datasets

Chapter 4

Genetic algorithms

4.1 Genetic Algorithms

"Genetic Algorithms are good at taking large, potentially huge search spaces and navigating them, looking for optimal combinations of things, solutions you might not otherwise find in a lifetime."

4.2 Introduction:

This is an introduction to genetic algorithm methods for optimization. Genetic algorithms were formally introduced in the United States in the 1970s by John Holland at University of Michigan. The continuing price/performance improvements of computational systems have made them attractive for some types of optimization. In particular, genetic algorithms work very well on mixed (continuous *and* discrete), combinatorial problems. They are less susceptible to getting 'stuck' at local optima than gradient search methods. But they tend to be computationally expensive.

To use a genetic algorithm, you must represent a solution to your problem as a *genome* (or *chromosome*). The genetic algorithm then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s).

This presentation outlines some of the basics of genetic algorithms. The three most important aspects of using genetic algorithms are: (1) definition of the objective function, (2) definition and implementation of the genetic representation, and (3) definition and implementation of the genetic operators. Once these three have been defined, the generic genetic algorithm should work fairly well. Beyond that you can try many different variations to improve performance, find multiple optima (species - if they exist), or parallelize the algorithms.

4.3 Hierarchy of GA's

The hierarchy diagram of the genetic algorithms is shown below to have a quick view on the evolution of genetic algorithms in the searching techniques.

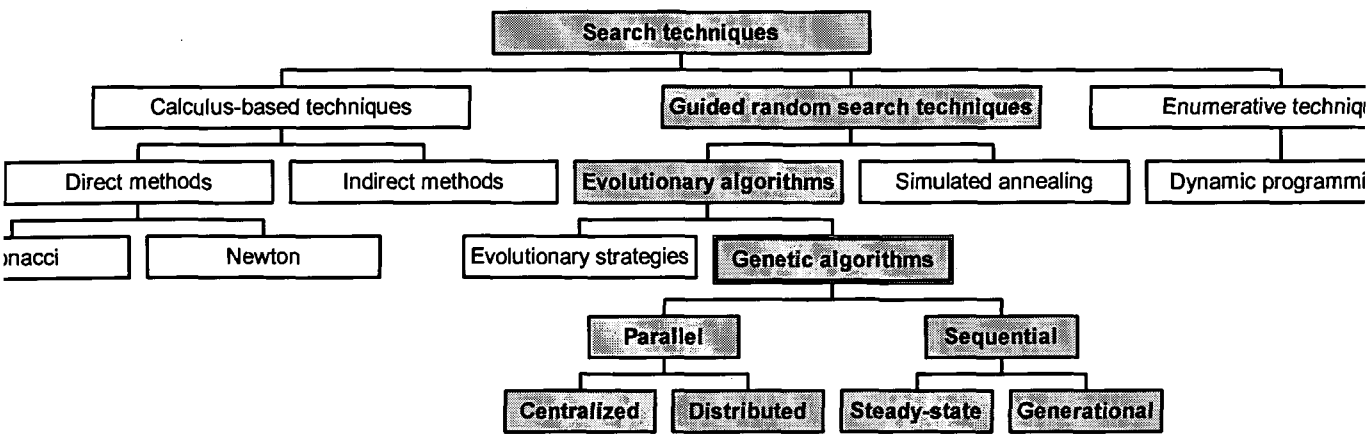


Figure 4.1 Genetic algorithm hierarchy

Many human inventions were inspired by nature. Artificial neural networks is one example. Another example is *Genetic Algorithms* (GA). GAs search by simulating evolution, starting from an initial set of solutions or hypotheses, and generating successive "generations" of solutions. This particular branch of AI was inspired by the way living things evolved into more successful organisms in nature. The main idea is *survival of the fittest*, a.k.a. *natural selection.*

A chromosome is a long, complicated thread of DNA (deoxyribonucleic acid). Hereditary factors that determine particular traits of an individual are strung along the length of these chromosomes, like beads on a necklace. Each trait is coded by some combination of DNA (there are four bases, A (Adenine), C (Cytosine), T (Thymine) and G (Guanine). Like an alphabet in a language, meaningful combinations of the bases produce specific instructions to the cell.

Changes occur during reproduction. The chromosomes from the parents exchange randomly by a process called **crossover**. Therefore, the offspring exhibit some traits of the father and some traits of the mother.

A rarer process called **mutation** also changes some traits. Sometimes an error may occur during copying of chromosomes (mitosis). The parent cell may have -A-C-G-C-T- but an accident may occur and changes the new cell to -A-C-T-C-T-. Much like a typist copying a book, sometimes a few mistakes are made. Usually this results in a nonsensical word and the

cell does not survive. But over millions of years, sometimes the accidental mistake produces a more beautiful phrase for the book, thus producing a better species. The following basics term are used in genetic algorithms.

Chromosome: A set of genes. Chromosome contains the solution in form of genes.
Gene: A part of chromosome. A gene contains a part of solution. It determines the solution. E.g. 16743 is a chromosome and 1, 6, 7, 4 and 3 are its genes.
Individual: Same as chromosome.
Population: No of individuals present with same length of chromosome.
Fitness: Fitness is the value assigned to an individual. It is based on how far or close a individual is from the solution. Greater the fitness value better the solution it contains.
Fitness function: Fitness function is a function which assigns fitness value to the individual. It is problem specific.
Breeding: Taking two fit individuals and intermingling there chromosome to create new two individuals.
Mutation: Changing a random gene in an individual.
Selection: Selecting individuals for creating the next generation.

4.4 Components of a GA:

A problem to solve, and...

- Encoding technique *(gene, chromosome)*
- Initialization procedure *(creation)*
- Evaluation function *(environment)*
- Selection of parents *(reproduction)*
- Genetic operators *(mutation, recombination)*
- Parameter settings *(practice and art)*

4.4.1 Simple Genetic Algorithm

{

```
    initialize population;
    evaluate population;
        while TerminationCriteriaNotSatisfied
        {
            select parents for reproduction;
            perform recombination and mutation;
            evaluate population;
        }
}
```

4.5 Natural Selection

In nature, the individual that has better survival traits will survive for a longer period of time. This in turn provides it a better chance to produce offspring with its genetic material. Therefore, after a long period of time, the entire population will consist of lots of genes from the superior individuals and less from the inferior individuals. In a sense, the fittest survived and the unfit died out. This force of nature is called **natural selection.**

The existence of competition among individuals of a species was recognized certainly before Darwin. The mistake made by the older theorists (like Lamarck) was that the environment had an effect on an individual. That is, the environment will force an individual to adapt to it. The molecular explanation of evolution proves that this is biologically impossible. The species does not adapt to the environment, rather, only the fittest survive.

4.6 Simulated Evolution

To simulate the process of natural selection in a computer, we need to define the following:

- A representation of an individual at each point during the search process we maintain a "generation" of "individuals." Each individual is a data structure representing the "genetic structure" of a possible solution or hypothesis. Like a chromosome, the genetic structure of an individual is described using a fixed, finite alphabet. In GAs, the alphabet {0, 1} is usually used. This string is interpreted as a solution to the problem we are trying to solve.

  For example, say we want to find the optimal quantity of the three major ingredients in a recipe (say, sugar, wine, and sesame oil). We can use the alphabet {1, 2, 3 ..., 9} denoting the number of ounces of each ingredient. Some possible solutions are 1-1-1, 2-1-4, and 3-3-1.

  As another example, the traveling salesperson problem is the problem of finding the optimal path to traverse, say, 10 cities. The salesperson may start in any city. A solution is a permutation of the 10 cities: 1-4-2-3-6-7-9-8-5-10.

  As another example, say we want to represent a rule-based system. Given a rule such as "If color=red and size=small and shape=round then object=apple" we can describe it as a bit string by first assuming each of the attributes can take on a fixed set of possible values. Say color={red, green, blue}, size={small, big}, shape={square, round}, and fruit={orange, apple, banana, pear}. Then we could represent the value for each attribute as a substring of length equal to the number of possible values of that attribute. For example, color=red could be represented by 100, color=green by 010, and color=blue by 001. Note also that we can represent color=red or blue by 101, and any color (i.e., a "don't care") by 111. Doing this for each attribute, the above rule might then look like: 100 10 01 0100. A set of rules is then represented by concatenating together each rule's 11-bit string.

  For another example see page 620 in the textbook for a bit-string representation of a logical conjunction.

- **Fitness function**

Given an individual, we must assess how good a solution it is so that we can rank individuals. This is usually a real number. For example, say we have individuals that are represented as a length-30 binary number. We can then use this individual as an integer, *i*, in the range 0 to 2^{30} - 1. A possible fitness function is $Fitness(i) = (i/2^{30} - 1)^{10}$. This function has a value between 0 and 1 and is monotonically increasing. Note that fitness functions need not be monotonic and frequently have multiple local maxima.

For example, one can give a subjective judgment from 1 to 5 for the dish prepared with the recipe 2-1-4.

Similarly, the length of the route in the traveling salesperson problem is a good measure, because the shorter the route, the better the solution.

For classification problems, the fitness function could be the percent correct classification on a given training set. For example, $Fitness(i) = (correct(i))^2$.

- **Reproduction methods**

There are two basic methods of reproduction, called mutation and crossover:

i. **Mutation**

Randomly change one or more digits in the string representing an individual. For example, the individual 1-2-3 may be changed to 1-3-3 or 3-2-3, giving two new offspring. How often to do mutation, how many digits to change, and how big a change to make are adjustable parameters.

ii. **Crossover**

Randomly pick one or more **pairs of individuals** as parents and randomly swap segments of the parents. For example, the individuals 1-3-3 and 3-2-3 may be chosen as parents. Suppose we select a crossover point after the first digit, then the above will generate two offspring: 3-3-3 and 1-2-3. As another example, given two parents 1011010 and 1100010, if the crossover point is between the third and fourth digits, then the two offspring are 1010010 and 1101010. This method is called *1-point crossover*. Similarly, we could define *2-point crossover*, which would select two points in each individual defining three intervals; the middle intervals are swapped to

produce the two offspring. The rate of crossover, the number of parent pairs, the number of crossover points, and the positions of the crossover points are adjustable parameters.

- **Selection**

  From a population of individuals, we wish to give the fitter individuals a better chance to survive to the next generation. We do *not* want to use the simple criterion "keep the best *n* individuals." It turns out nature does not kill all the unfit genes. They usually become recessive for a long period of time. But then they may mutate to something useful. Therefore, there is a tradeoff for better individuals and diversity.

  A simple selection method is each individual, *i*, has the probability *Fitness(i) / sum_over_all_individuals_j Fitness(j)*, where *Fitness(i)* is the fitness function value for individual *i*. This method is sometimes called **fitness proportionate selection**. Other selection methods have also been used, e.g., **rank selection**, which sorts all the individuals by fitness and the probability that an individual will be selected is proportional to its rank in this sorted list.

  One potential problem that can be associated with the selection method is called **crowding**. Crowding occurs when the individuals that are most fit quickly reproduce so that a large percentage of the entire population looks very similar. This reduces diversity in the population and may hinder the long-run progress of the algorithm.

If only mutation is used, the algorithm is very slow. Crossover makes the algorithm significantly faster.

With the above defined, one way to define a Genetic Algorithm is as follows:

```
proc GA(Fitness, theta, n, r, m)
  ; Fitness is the fitness function for ranking individuals
  ; theta is the fitness threshold, which is used to determine
  ;  when to halt
  ; n is the population size in each generation (e.g., 100)
```

; r is the fraction of the population generated by crossover (e.g., 0.6)

; m is the mutation rate (e.g., 0.001)

P := generate n individuals at random

; initial generation is generated randomly

while max Fitness(h_i) < theta **do**

i

; define the next generation S (also of size n)

Reproduction step: Probabilistically select

(1-r)n individuals of P and add them to S intact, where

the probability of selecting individual h_i is

Prob(h_i) = Fitness(h_i) / SUM Fitness(h_j)

j

Crossover step: Probabilistically select rn/2 pairs

of individuals from P according to Prob(h_i)

foreach pair (h_1, h_2), produce two offspring by applying

the crossover operator and add these offspring to S

Mutate step: Choose m% of S and randomly invert one

bit in each

P := S

end_while

Find *b* such that Fitness(b) = max Fitness(h_i)

i

return(b)

end_proc

4.7 Benefits of Genetic Algorithms

There are the following advantages to use the genetic algorithms.

- Concept is easy to understand
- Modular, separate from application
- Supports multi-objective optimization
- Good for "noisy" environments
- Always an answer; answer gets better with time
- Inherently parallel; easily distributed
- Many ways to speed up and improve a GA-based application as knowledge about problem domain is gained
- Easy to exploit previous or alternate solutions
- Flexible building blocks for hybrid applications
- Substantial history and range of use

4.7.1 When to Use a GA

- Alternate solutions are too slow or overly complicated
- Need an exploratory tool to examine new approaches
- Problem is similar to one that has already been successfully solved by using a GA
- Want to hybridize with an existing solution
- Benefits of the GA technology meet key problem requirements

4.8 Conclusion

Genetic Algorithms are easy to apply to a wide range of problems, from optimization problems like the traveling salesperson problem, to inductive concept learning, scheduling, and layout problems. The results can be very good on some problems, and rather poor on others. Genetic algorithms **are** rich - rich in application across a large and growing number of disciplines. If only mutation is used, the algorithm is very slow. Crossover makes the algorithm significantly faster. GA is a kind of **hill-climbing** search; more specifically it is very similar to a **randomized beam search**. As with all hill-climbing algorithms, there is a problem of local maxima. Local maxima in a genetic problem are those individuals that get stuck with a pretty good, but not optimal, fitness measure. Any small mutation gives worse fitness. Fortunately, crossover can help them get out of a local maximum. Also, mutation is a random process, so it is possible that we may have a sudden large mutation to get these individuals out of this situation. (In fact, these individuals never get out. It's their offspring that get out of local maxima.) One significant difference between GAs and hill-climbing is that, it is generally a good idea in GAs to fill the local maxima up with individuals. Overall, GAs has fewer problems with local maxima than back-propagation neural networks.

Chapter 5

Non Linear ODE

5.1 Differential Equation:

A differential equation is an equation that involves the derivatives of a function as well as the function itself. If partial derivatives are involved, the equation is called a partial differential equation; if only ordinary derivatives are present, the equation is called an ordinary differential equation. Differential equations play an extremely important and useful role in applied math, engineering, and physics, and much mathematical and numerical machinery has been developed for the s olution of differential equations.

A differential equation is an equation relating some function f to one or more of its derivatives. An examples is

$$\frac{d^2}{dx^2}f(x)+2x\frac{d}{dx}f(x)+f^2(x)=\sin x \qquad 5.1$$

This particular equation involves a function f together with its first and second derivatives. Any given differential equation may or may not involve f or any particular derivative of f. But, for an equation to be a differential equation, at least some derivative of f must appear.

A few additional examples of differential equations are

$$(1-x^2)\frac{d^2}{dx^2}y-2x\frac{d}{dx}y+p(p+1)y=0 \qquad 5.2$$

$$x^2\frac{d^2}{dx^2}y+x\frac{d}{dx}y+(x^2-p^2)y=0 \qquad 5.3$$

$$\frac{d^3}{dx^3}y+\left(\frac{d}{dx}y\right)^2=y^3+\sin x \qquad 5.4$$

Here equation 1 is called Legendre's equation, equation2 is Bessel's equation and 3 is third order non linear differential equation.

5.2 The Nature of Solutions

An ordinary differential equation of order n is an equation involving an unknown function f together with its derivatives

$$\frac{d}{dx}f, \frac{d^2}{dx^2}f, \ldots\ldots\ldots\ldots, \frac{d^n}{dx^n}f \qquad 5.5$$

In a formal manner the above equation is written as following.

$$F\left(x, f, \frac{d}{dx}f, \frac{d^2}{dx^2}f, \frac{d^3}{dx^3}f, \ldots\ldots\ldots\ldots \frac{d^n}{dx^n}f\right) = 0 \qquad 5.6$$

The solution of this equation will be a function f in term of x.

5.3 Separable Equation

A general class of equations with the property that

i) We can immediately recognize members of this class of equations

ii) We have a simple and direct method for solving such equations

This is the class of separable equation.

A first order ordinary differential equation is separable if it is possible; by elementary algebra manipulation to arrange the equation so hat all the dependant variables are on one side and independent variables are on the other side.

5.4 First Order Linear Differential Equation

This is another class of differential equation that is recognizable easily, an equation is said to be first order differential equation if it is in the form of

$$\frac{dy}{dt} + a(x)y(t) = b(x) \qquad 5.7$$

Because it contain the first derivative that's why it is called as first order differential equation. The linear aspect depends upon the fact that the left hand side involves a differential operator that acts linearly on the space of differential functions.

5.4.1 Examples

Consider the differential equation

$$y' + 2xy = x \tag{5.8}$$

After the solution we got the following exact solution

$$y = \frac{1}{2} + Ce^{-x^2} \tag{5.9}$$

Similarly for another simple first order differential equation

$$x^2 y' + xy = x^3 \tag{5.10}$$

Has the solution as

$$y = \frac{x^2}{3} + \frac{C}{x} \tag{5.11}$$

5.5 Second Order Linear Differential Equation

Second order linear differential equation is frequently used in physics and electronics for signal processing, for instance acceleration given by second derivative and force is mass time acceleration. A function that involves the second derivative in the equation is called second order linear differential equation.

$$y'' - 5y' + 9y = 0 \tag{5.12}$$

Is a second order linear while

$$\sin(y'') - 2y' + 2y = 0 \tag{5.13}$$

and

$$y.\ddot{y} + 5\dot{y} + 2y = 0 \tag{5.14}$$

are not.

5.5.1 Examples

The differential equation

$$2\ddot{y} + 6\dot{y} + 2y = 0 \tag{5.15}$$

Has the following exact solution

$$y = Ae^{(-3+\sqrt{5})x/2} + Be^{(-3-\sqrt{5})x/2} \tag{5.16}$$

We have number of practical applications of first order and second order linear ODEs like Bessel ,Legendre equation and Schrodinger equations which has complex solutions although the equations are linear.

5.6 Non Linear Differential Equation:

Non linear ordinary differential equations and non linear partial differential equations have many applications in mechanics, circuit theory, computational analysis, reaction kinetics, mathematical biology, economics and many other areas. In fact, in applications the most of the systems are non linear. I am here to introduce only the simple non linear differential equation and non linear coupled differential equation; however for detail studies of subject matter reader is referred to the reference books, nonlinear partial differential equations 1990 Ams Chen Dibenedetto.djvu and introduction to numerical methods in differential equations (Springer, 2007.pdf).

5.6.1 Examples

Consider the following Newton's second law of motion for a particle of mass m moving in one dimension,

$$m\ddot{x} = F(x, \dot{x}) \tag{5.17}$$

Where F is a force depending upon the position and the velocity.

Convert the above equation in linear first order system

$$\dot{x} = y \tag{5.18}$$

$$\dot{y} = \frac{1}{m} F(x, y) \qquad 5.19$$

Because the above system is autonomous initial value problem (IVP) consists of the solving the system subject to the initial condition.

$$x(t_o) = x_o$$
$$y(t_o) = y_o$$

Thus equilibrium solutions are found as solutions of the algebraic, simultaneous system of equations

$$f(x, y) = 0, g(x, y) = 0 \qquad 5.20$$

If an equilibrium point in the phase plane has the property that there is small neighborhood about the point where there are no other equlibira, then we say the equilibrium point is isolated.

Non linear dynamics is most common in nature and almost 80% problems of the world are non linear in nature. These non linear systems are very complex than linear problems.

Non linear differential equations are one of the hot issues in signal processing piratical examples, like Weissinger's non linear equations is used in mearning the turning effects in the wings of aeroplane. The solutions of the non linear differential equation are very difficult with respect to calculation, so researchers have made the numerical methods like Euler method, modified Euler method, Simpson method and Runge Kutta method. All these methods have its own strengths and weaknesses whose detail can be found in standard numerical computing reference books.

These numerical methods are designed for the non linear system in which non linear systems are rarely be resolved analytically by finding solution formulas. So along with qualitative methods numerical method come to the front.

I have produced a modern numerical technique for the continuous time solution for famous non linear differential equations such that,

1) Bernoulli
2) Weissinger's
3) Van der Pol

Detail procedure for the solution of above differential equation is provided in next session.

Chapter 6

Proposed Numerical Method for Solution of Non Linear ODE

6.1 Bernoulli Equation:

The network is called as DE-feed forward neural network. The first two layers have a bias; the last four layers have no bias. The biases in first two layers are b_i. The activation functions for the first two layers are log sigmoid and the first derivative of the log sigmoid function. While linear function is used in the last four layers as the activation function. The numbers of neuron in the first two layers is m and the number of neuron in he four three layers is 1. The input t is connected two first two layers. The connection weight of the input to the first layer to the second layer is equal to w_i. The first layer is connected to the third layer with the connection weights α_i and the second layer is connected to the fourth layer with the connection weights $w_i\alpha_i$. Third layer is connected to the fifth layer with the square factor as the connection weight. Third, fourth and fifth layers is connected to the six layer with a constant connection weight and the value of this constant is 1.

The three networks have to train simultaneously as a consequence of the inter-relationship. It is a specific point of intention how to adjust the value of the weights of the DE-feed forward neural network. Although we can use the well known gradient decent method but as the weights of the DE-feed forward neural network are highly correlated, which feature is not easily incorporated into these gradient decent methods. So in order to find the values simultaneously we formed a highly powerful evolutionary algorithm called as hybrid intelligent algorithm to compute the connection weights.

$$\frac{dy}{dt} + y(t) - y^2(t) = 0 \qquad 6.1$$

6.2 Weissinger's Equation:

$$ty^2(t)\left(\frac{dy}{dt}\right)^3 - y^3(t)\left(\frac{dy}{dt}\right)^2 + t(t^2+1)\frac{dy}{dt} - t^2 y(t) = 0 \qquad 6.2$$

With following known conditions

6.3

$$y(4) = \sqrt{\frac{33}{2}} \qquad 6.4$$

Weissinger's equation has the following exact solution.

$$y = \sqrt{t^2 + \frac{1}{2}} \qquad 6.5$$

I form another DE feed forward network for a well known Weissinger's equation whose numerical solution is not existed but has a complex mathematical analytical solution. In the network cumulatively there are nine layers. In which layer one and layer2 have a bias; the last seven layers have no bias. The biases in first two layers are b_i. The activation functions for the first two layers are log sigmoid and the first derivative of the log sigmoid function. While linear function is used in the last seven layers as the activation function. The numbers of neuron in the first two layers is m and the number of neuron in the last seven layers is 1. The input t is connected two first two layers. The connection weight of the input to the first layer to the second layer is equal to w_i. The first layer is connected to the third layer with the connection weights α_i and the second layer is connected to the fourth layer with the connection weights $w_i\alpha_i$. Third layer and fourth layer connected to the fifth layer with the square and cube of the previous values simultaneously. Layer three and layer four is connected to the layer six by the cube and square values of the previous connection link simultaneously. Layer five, six, seven and eight are connected to the layer nine with a constant connection weight and the value of this constant is 1,-1,1 and -1 respectively. The three networks have to train simultaneously as a consequence of the inter-relationship. It is a specific point of intention how to adjust the value of the weights of the DE-feed forward neural network. Although we can use the well known gradient decent method but as the weights of the DE-feed forward neural network are highly correlated, which feature is not easily incorporated into these gradient decent methods. Weight sharing technique is also a good option for the parameters setting but due to the highly correlation in the adaptive parameters and the architecture so genetic algorithms are used for the optimization of the weights, in order to not struck in local minima I have used the pattern search so the exact minima is obtained from the whole solution set space.

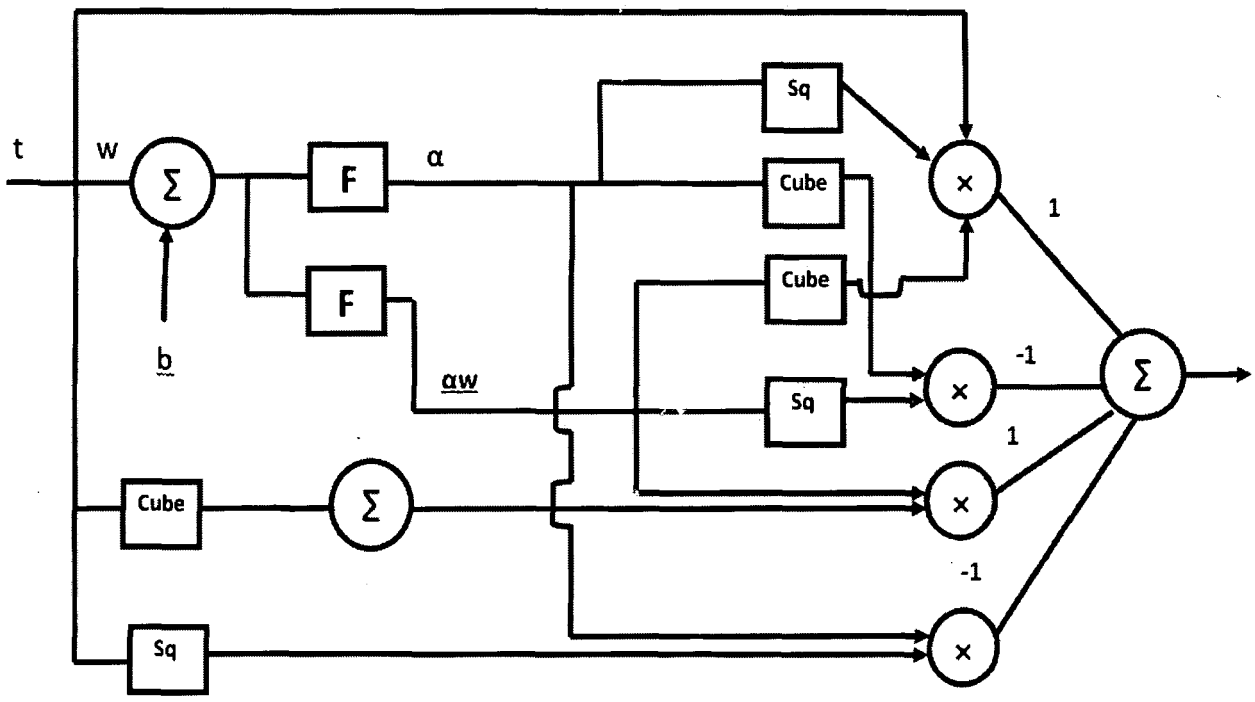


Figure 6.4 Weissinger's Neural Network Architecture

6.3 Van Der Pol Equation

$$\frac{d^2y}{dt^2} - \mu(1-y^2(t))\frac{dy}{dt} + y(t) = 0 \qquad 6.6$$

Suppose $\mu = 1$

Here we take the non stiff problem in order to reduce the complexity because in a stiff problem the behavior is abnormal with respect to time. In the van der pol equation we take the constant μ as 1. The method I adopt to solve this is vary effective as compared to the existed analytical mathematical method and MATLAB method to solve it. Because DE-feed forward neural network can handle both of the stiff and non stiff problem easily without giving the large number of constraints.

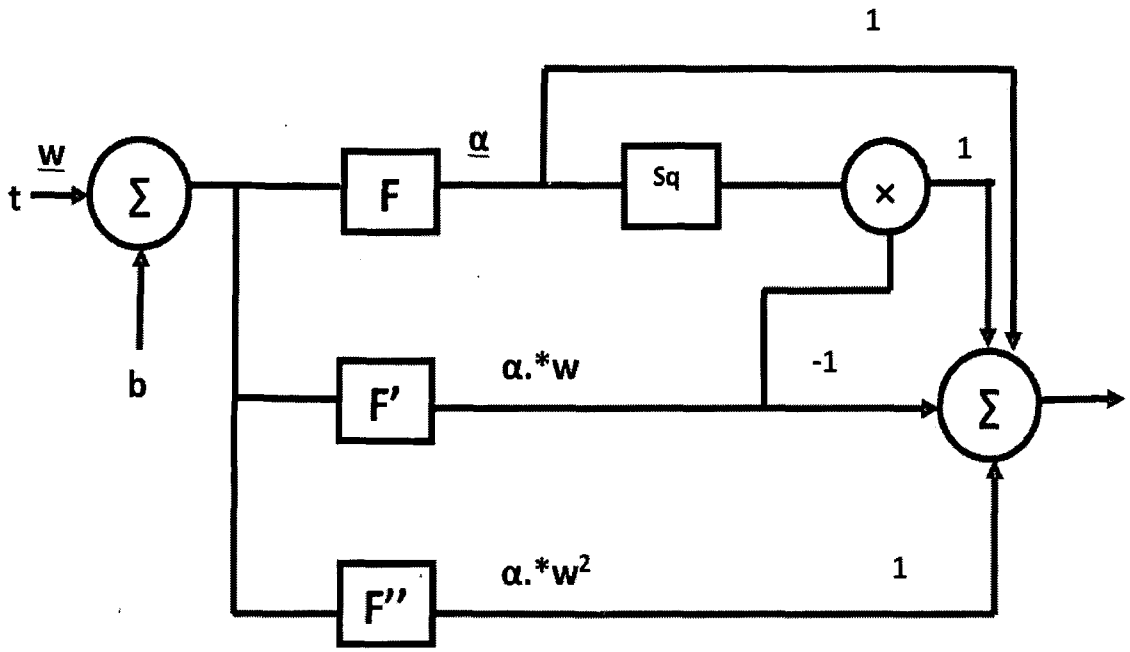


Figure 6.4 Van der Pol architecture

DE feed forward network for Van Der Pol equation consist of eight layers. In which layer one, layer two and layer three have a bias; the last five layers have no bias. The biases in first three layers are b_i. The activation functions for the first three layers are log sigmoid, first derivative of the log sigmoid function and second derivative of log sigmoid function. While linear function is used in the remaining layers as the activation function. The numbers of neuron in the first three layers is m and the number of neuron in the last five layers is 1. The input t is connected to the first three layers. The connection weight of the input to the first layer to the third layer is equal to w_i. The first layer is connected to the fourth layer with the connection weights α_i; the second layer is connected to the fifth layer with the connection weights $\alpha_i w_i$ and third layer is connected to the layer six with the connection weight $\alpha_i w_i^2$. Layer four l and five connected to the layer seven. Layer four, five, six, seven are connected to the layer eight with a constant connection weight and the values of the constants are 1,-1, 1 and 1 respectively.

6.4 Mapping for DENN

DENN has a approximation mapping that is the linear combination of functions here, I take the log sigmoid function as the activation function and its first and second derivative according to the non linear problem. This is the general mapping for the function, its first derivative and second derivative respectively. Here $\phi(t)$ is the log sigmoid function, α_i, w_i are the weights on the communications links which are trained by the genetic algorithm incorporated with the pattern search called as hybrid intelligent algorithm. The values of these variable are vary from [-5 to 5] and are real in the finite whole domain. While b_i is the values of the bias which also have the range as the range of alphas and weights. In feed forward DE neural network I have taken some fixed biases as 1 or -1 as well depending upon the architecture of the noon linear differential equation. The number of neurons is m with hidden layers.

$$\phi(x) = \frac{1}{1+\exp(-x)} \tag{6.7}$$

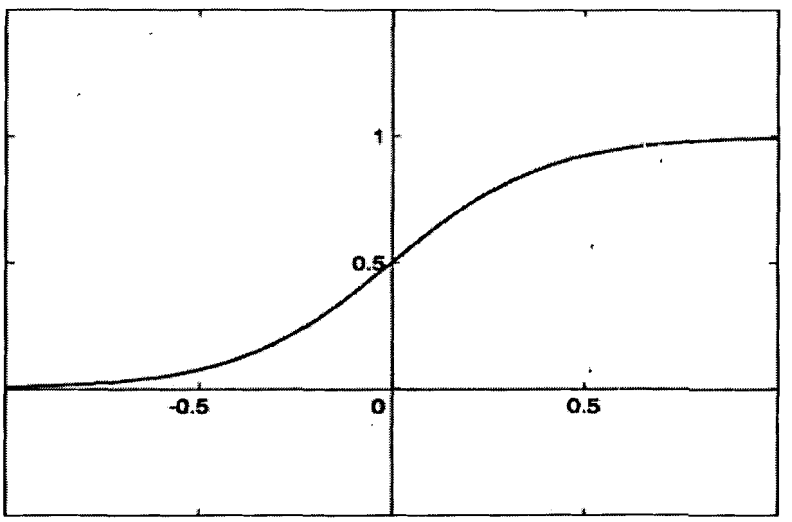


Figure 6.4 Log sigmoid function

$$f(x) = \sum_{i=0}^{m} \alpha_i \phi(w_i x + b_i) \tag{6.8}$$

$$\frac{d}{dx}(f(x)) = \sum_{i=0}^{m} \alpha_i w_i \frac{d}{dx}\phi(w_i x + b_i) \tag{6.9}$$

$$\frac{d^2}{dx^2} f(x) = \sum_{i=0}^{m} \alpha_i w_i^2 \frac{d^2}{dx^2} \phi(w_i x + b_i) \qquad 6.10$$

There x is the running time taken in a finite real domain $x \in \{0.1,0.2,0.3,.....4.0\}$.

Our genetic algorithm conducts a randomized, parallel, hill-climbing search for hypothesis that optimizes our pre defined fitness function (fit_fun).

GA (Fitness, Fitness_threshold, p, r, m)

Where

Fitness: A function that assigns an evaluation score, given a hypothesis.

Fitness_threshold: A threshold specifying the terminate criteria that is 0.0001.

p: The number of the hypothesis inclined in the population.

r: The fraction of the population to be replaced by he crossover at each step.

m: The mutation rate.

- Initialize population: P ⟵ generate 'p' hypotheses at random
- Evaluate: For each h in p, compute Fitness(h)
- While(max Fitness(h)) < Fitness_threshold do

Create a new generation, Ps:

1. Select: Probabilistically select (1-r)p members of P to add to Ps. The probability Pr(hi)of selecting hypothesis hi from P is given by

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_i)} \qquad 6.11$$

2. Crossover: Probabilistically select $\frac{r-p}{2}$ pairs of hypothesis from P, according to Pr(hi) given above. For each pair $(h1, h2)$, produce two offspring by applying the crossover operator. Add all offspring to Ps.
3. Mutation: Choose m percent of members of Ps with uniform probability.
4. Update: P ⟵ Ps.
5. Evaluate: For each h in P, compute Fitness(h)

Return the hypothesis from P that has the highest fitness.

Appendix A

Simulation and Results

Bernoulli

Main_simple_bernulee

% Main function provides the graphical comparison between exact analytical solution and solution generated from DD-Neural Network incorporating the function fit_fun_sb and output function

% Input is randomly selected vector of length 15 between -5 to 5

% W1: the input vector of length fifteen randomly selected between -5 to 5.

% a: Initial population is taken 160 and is divided into eight sub populations each of length 20

%Program runs for 400 generation or $MSE < 10^{-6}$ which comes earlier

```
W1=rand(1,15);
a=[20,20,20,20,20,20,20,20];
options = gaoptimset('FitnessLimit',0.000001,'Generations',400,'PopulationSize',a);
[W1, fval]=ga(@fit_fun_sb,15,options);
   x=0:-0.1:-3.9;
aa=output(W1);
plot(x,aa,'o');
hold;
syms z;
y=1/(1-exp(z)/2);
ezplot(y,[-4,0]);
```

fit_fun_bernoulli

% This function defines the fit function which has to be satisfied for the correct approximation of the Bernoulli equation by the proposed method.

% e1: Defines the approximation of functional equation

%e2: Defines the initial condition for the Bernoulli equation

```
function e=fit_fun_sb(W)
t=-rand()*4;
[output,derivative,dderivative,in_value_zero,der_value_zero]=eq_parametersloop(t,W);
e1=(output-(output^2)+derivative)^2;
t=0;
[output,derivative,dderivative,in_value_zero,der_value_zero]=eq_parametersloop(t,W);
e2=(in_value_zero-2)^2;
   e=e1+e2;
```

Eq_parameter_bernullee

% equation parameter function defines the derivative and double derivative of the log sigmoid function with the initial conditions/ boundary value condition of the equation

%w: Defines the weights on the connection links between the neurons

%Alpha: these are the real constant in the neurons

%Beta: Defines the real values of the bias

```
function [output,derivative,dderivative,in_value_zero,der_value_zero]=eq_parametersloop(t,W)
% W=rand(1,15);
[m,n]=size(W);
w=W(1:(n/3));
Alpha=W((n/3)+1:(2*n/3));
beta=W((2*n/3)+1:n);
Mul=Alpha.*w;
Mul2=Alpha.*w.^2;
x=w*t+beta;
for k=1:n/3
   log_s(k)   =1/(1+exp(-x(k)));
   dlog_s(k)  =1/(2+exp(x(k))+exp(-x(k)));
   ddlog_s(k) =(exp(-x(k))-exp(x(k)))/(2+exp(x(k))+exp(-x(k)))^2;
end
output=sum(Alpha.*log_s);
derivative=sum(Mul.*dlog_s);
dderivative=sum(Mul2.*ddlog_s);
in_value_zero=output;
der_value_zero=derivative;
```

Output_bernullee

```
function our_output=output(W)
[m,n]=size(W);
for j=1:m
   i=0;
   for t=0:-0.1:-3.9
      w=W(j,1:(n/3));
      Alpha=W(j,(n/3)+1:(2*n/3));
      beta=W(j,(2*n/3)+1:n);
      x=w*t+beta;
      for k=1:n/3
         log_s(k)    =1/(1+exp(-x(k)));

      end

      output1(j,i+1)=sum(Alpha.*log_s);
      i=i+1;
   end
end
our_output=output1;
```

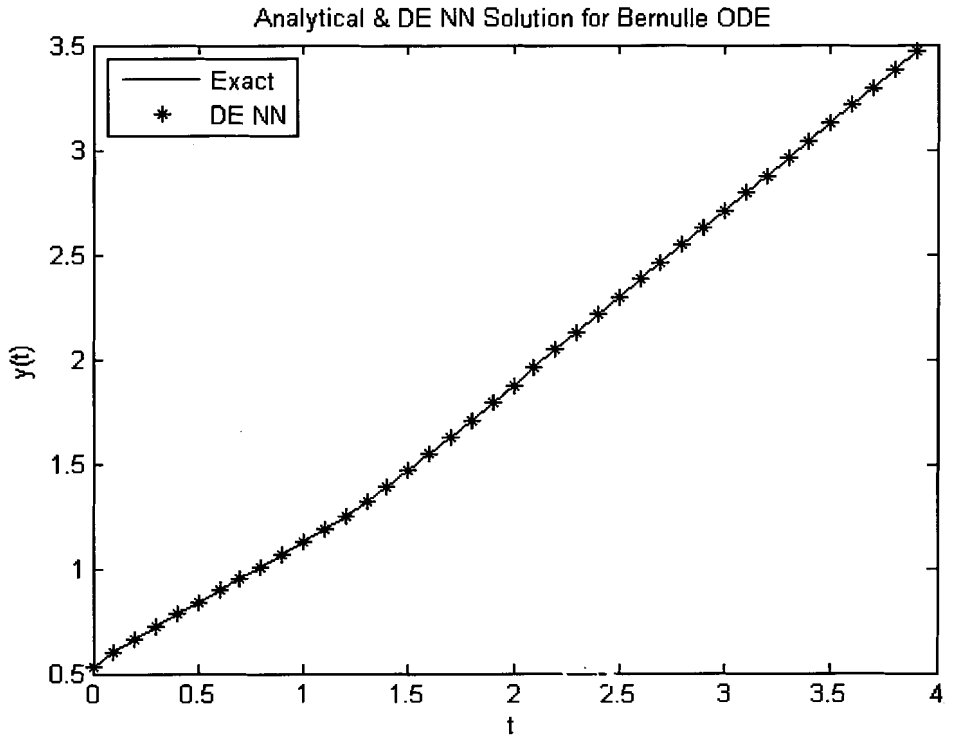


Figure A1: Analytical & DENN solution for Bernoulli equation

Chromosomes:

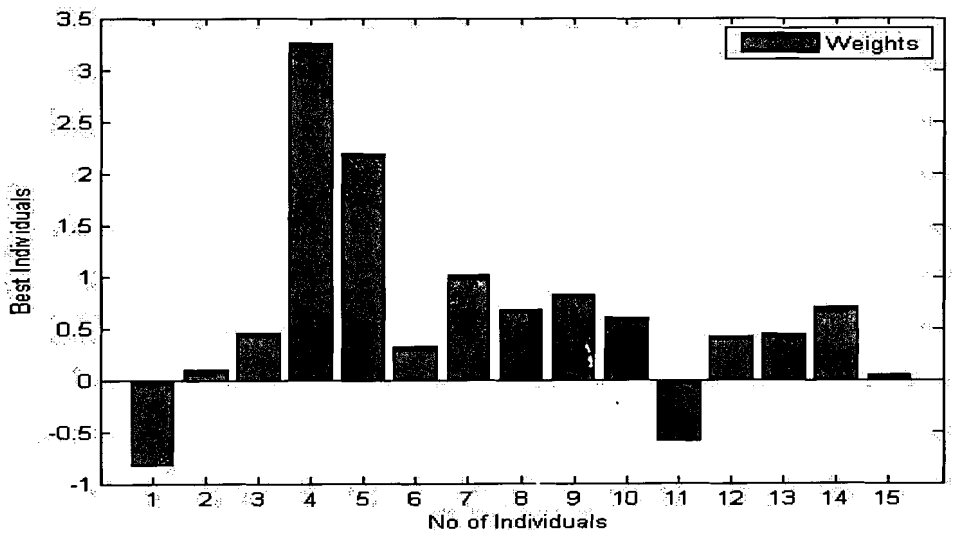


Figure A2: Beast individuals in Bernoulli equation

MSE Result

$MSE = 1.23e^{-6}$

Generations:

400

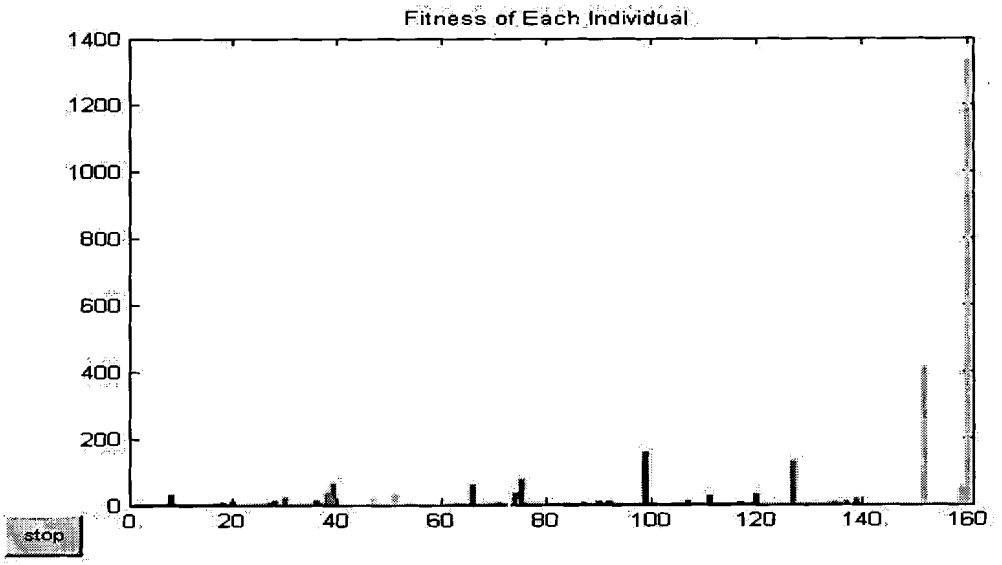


Figure A3: Fitness behavior after 100 generations

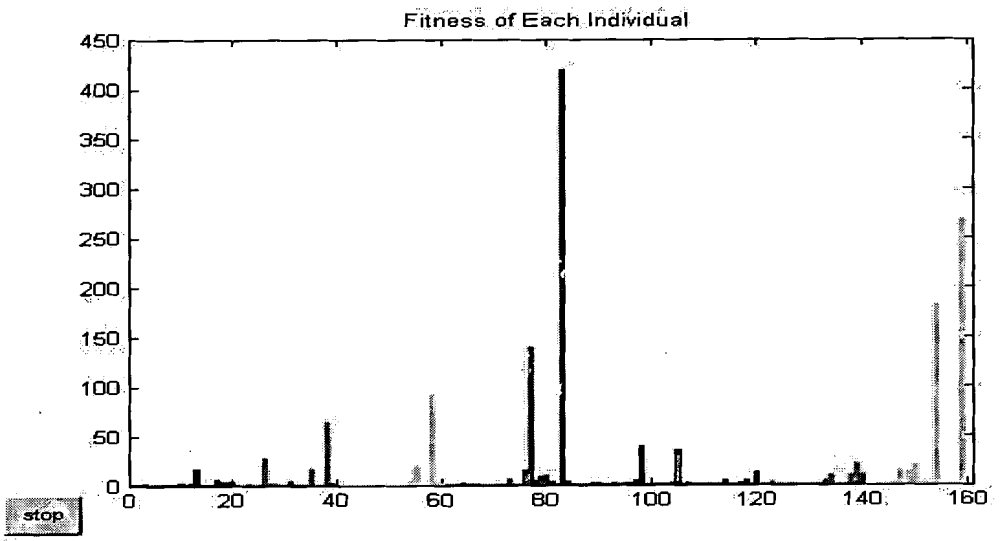


Figure A4: Fitness behavior after 200 generation

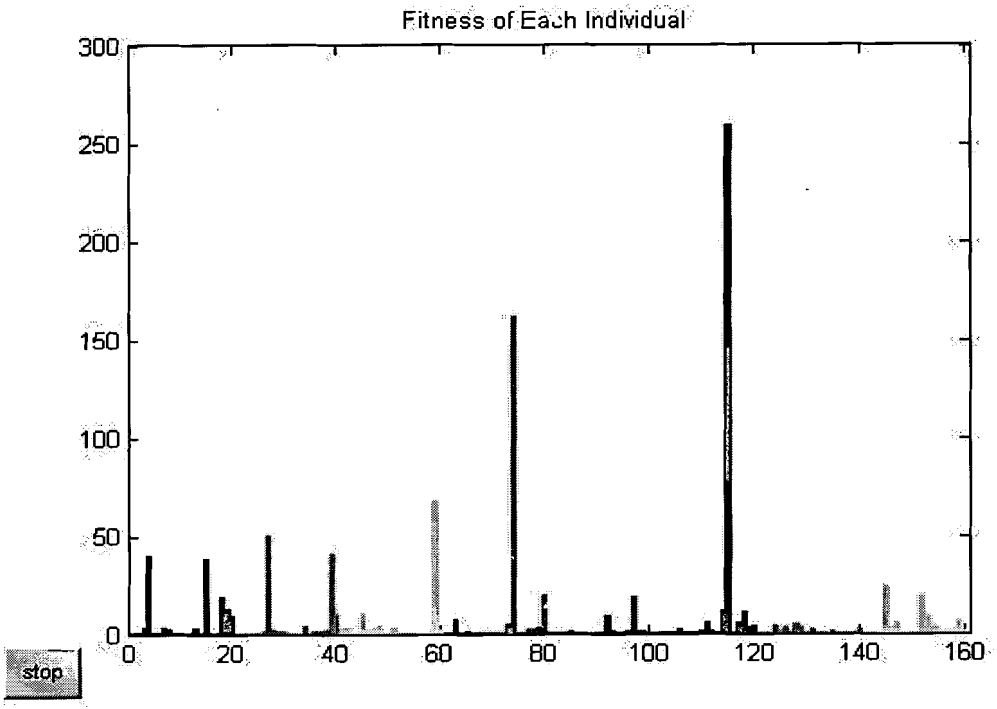


Figure A5: Fitness behavior after 300 generations

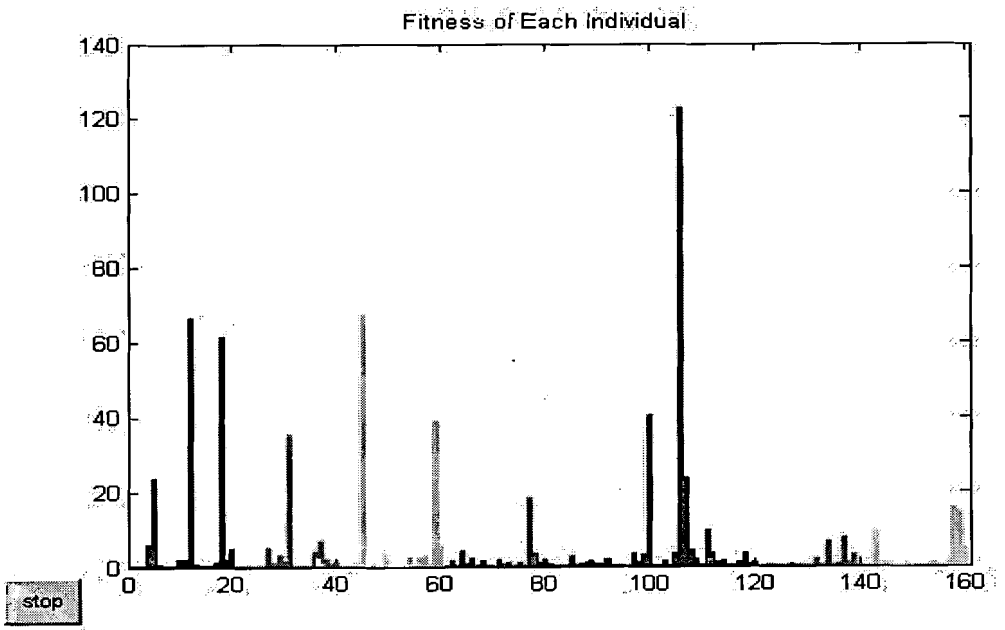


Figure A6: Fitness behavior after 400 generations

Van Der Pol

% Main function provides the graphical comparison between existed numerical method and solution generated from DD-Neural Network incorporating the fitness function and output function

% Input is randomly selected vector of length 15 between -5 to 5

% W1: the input vector of length fifteen randomly selected between -5 to 5 real numbers.

% a: Initial population is taken 160 and is divided into eight sub populations each of length 20

%Program runs for 400 generation or $MSE < 10^{-6}$ which comes earlier

Main_Van_dar_pol

```
W1=rand(1,33);
a=[20,20,20,20,20,20,20,20];
options = gaoptimset('FitnessLimit',0.000001,'Generations',2000,'PopulationSize',a);
[W1, fval]=ga(@fit_fun_vdp,33,options);


x=0:0.1:3.9;
y1=output(W1);
plot(y1,'*','color','black');
```

% Fitness function contain the values of the error which have the initial conditions

Fit_fun_van_dar_Pol

```
function e=fit_fun_vdp(W)
t=rand()*20;
[output,derivative,dderivative,in_value_zero,der_value_zero]=eq_parametersloop(t,W);
e1=(dderivative-(1-output^2)*derivative+output)^2;
t=0;
[output,derivative,dderivative,in_value_zero,der_value_zero]=eq_parametersloop(t,W);
e2=(in_value_zero-2)^2;
e3=(der_value_zero)^2;
   e=e1+e2+e3;
```

% Alpha: Contains the real variables from -5 to 5

%beta: Are the bias to the DE-Neural network

% Equation Parameter computes the Logsigmoid, first derivative of logsigmoid and second derivative of logsigmoid respectively

eq_parameter_van_dar_pol

```
function [output,derivative,dderivative,in_value_zero,der_value_zero]=eq_parametersloop(t,W)
 [m,n]=size(W);
w=W(1:(n/3));
Alpha=W((n/3)+1:(2*n/3));
beta=W((2*n/3)+1:n);
Mul=Alpha.*w;
Mul2=Alpha.*w.^2;
x=w*t+beta;
for k=1:n/3
   log_s(k)   =1/(1+exp(-x(k)));
   dlog_s(k)  =1/(2+exp(x(k))+exp(-x(k)));
   ddlog_s(k) =(exp(-x(k))-exp(x(k)))/(2+exp(x(k))+exp(-x(k)))^2;
end
output=sum(Alpha.*log_s);
derivative=sum(Mul.*dlog_s);
dderivative=sum(Mul2.*ddlog_s);
in_value_zero=output;
der_value_zero=derivative;
```

Output_van_dar_pol

```
function our_output=output(W)
[m,n]=size(W);
for j=1:m
   i=0;
   for t=0:0.1:19.5
      w=W(j,1:(n/3));
      Alpha=W(j,(n/3)+1:(2*n/3));
      beta=W(j,(2*n/3)+1:n);
      x=w*t+beta;
      for k=1:n/3
         log_s(k)    =1/(1+exp(-x(k)));
      end
      output1(j,i+1)=sum(Alpha.*log_s);
      i=i+1;
   end
end
our_output=output1;
```

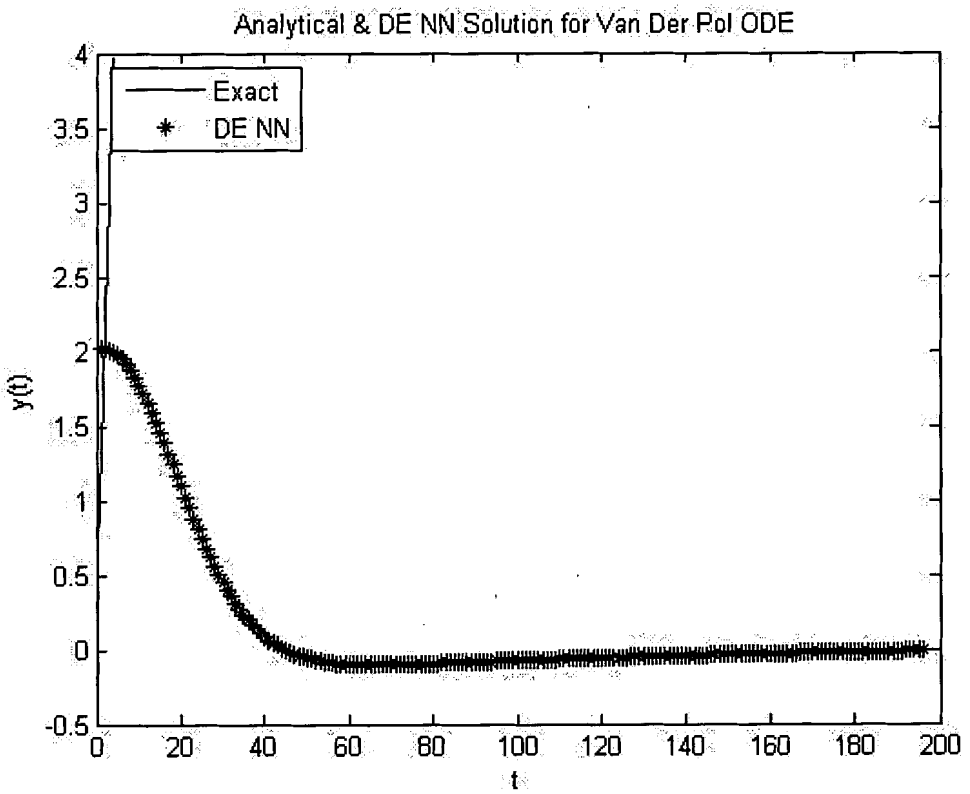


Figure A7: Analytical & DENN solution for Van der Pol ODE

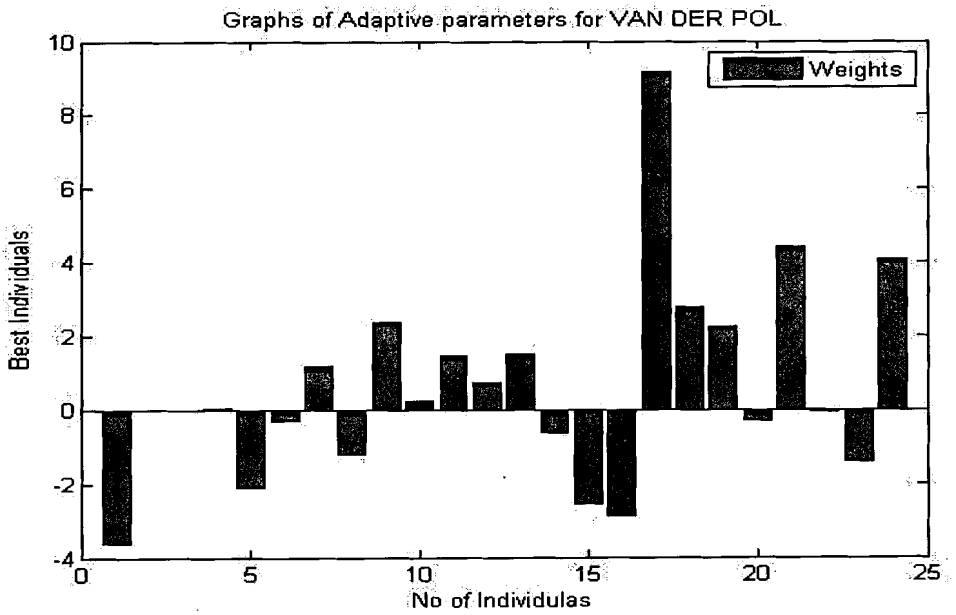


Figure A8: Beast individuals in Van Der Pol equation non Stiff problem

MSE Result:

$MSE = 1.49e^{-5}$

Generations:

400

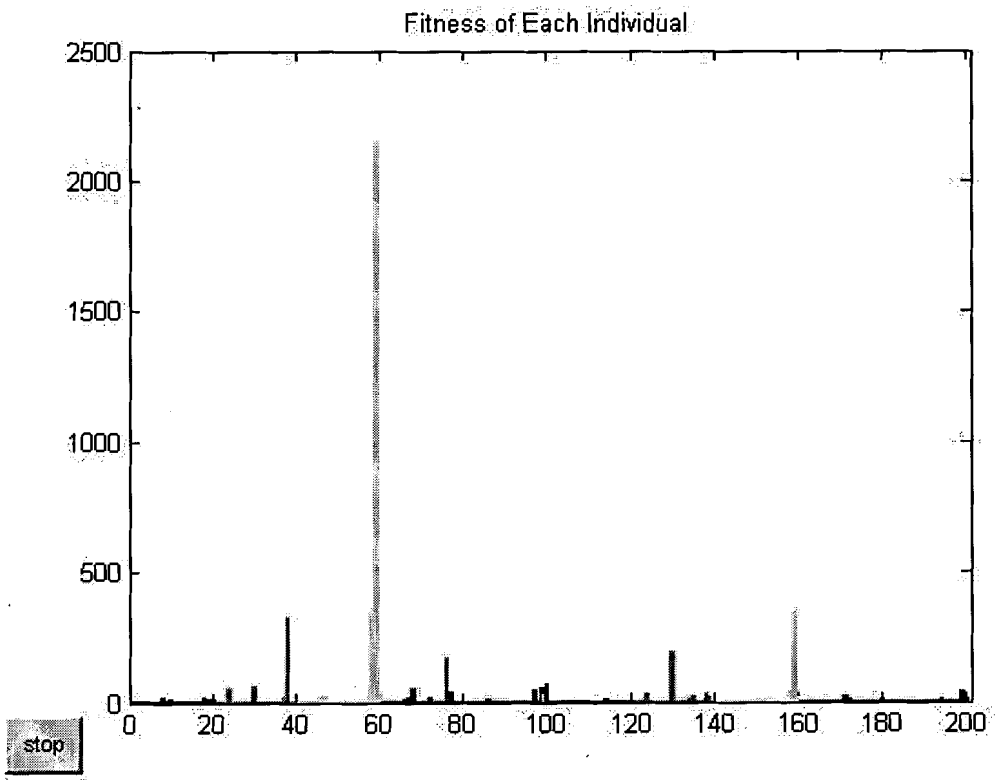


Figure A9: Fitness behavior after 50 generations

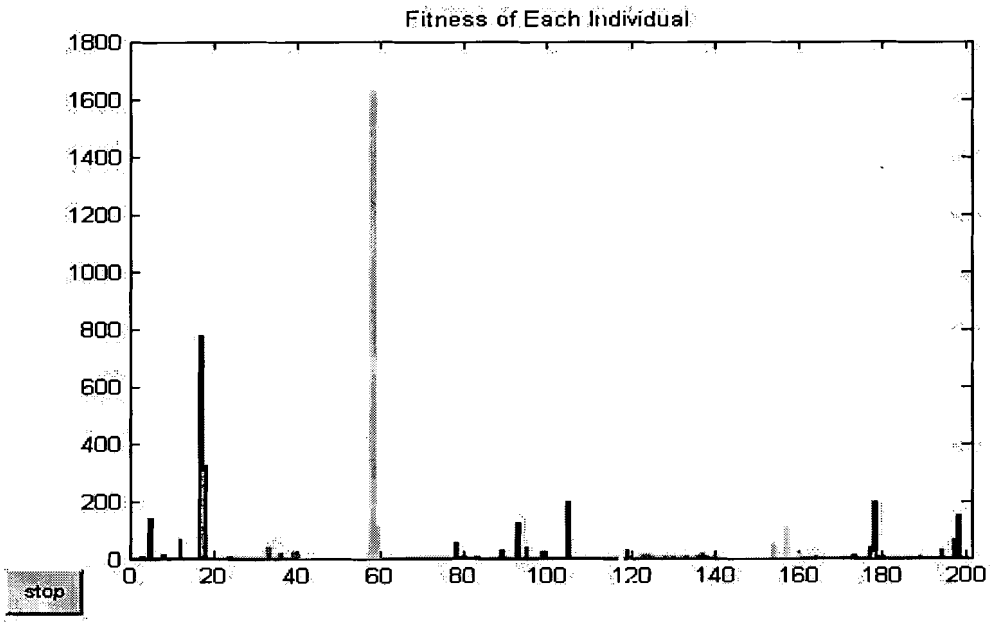


Figure A10: Fitness behavior after 100 generations

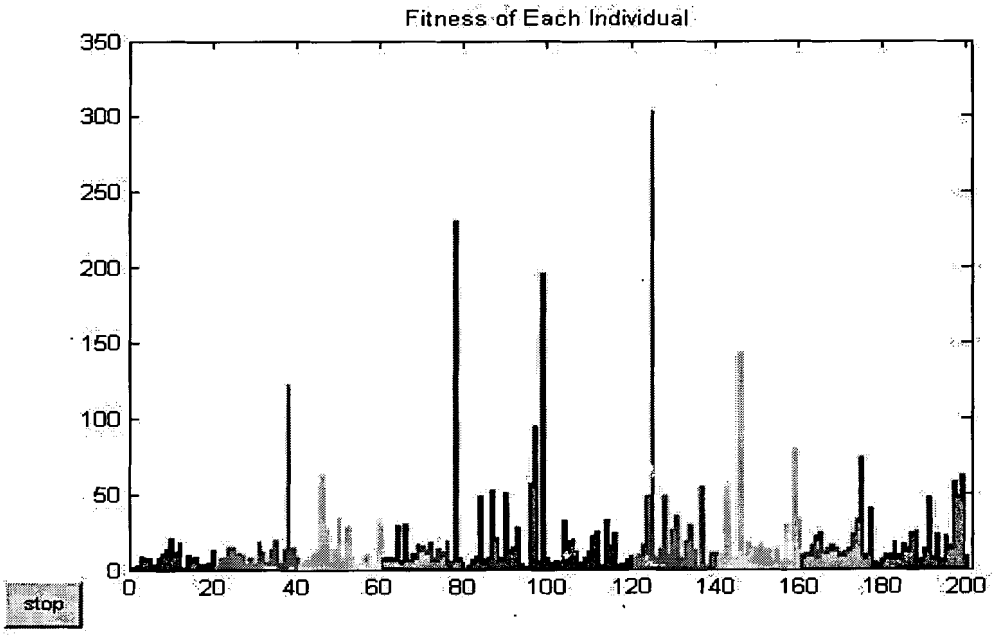


Figure A11: Fitness behavior after150 generations

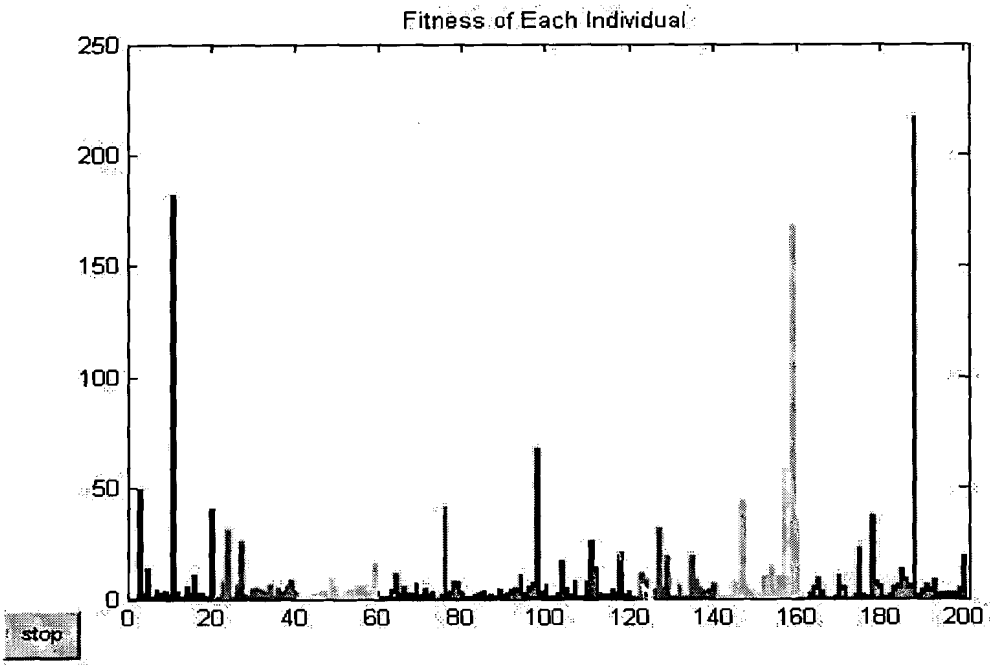


Figure A12: Fitness behavior after 200 generations

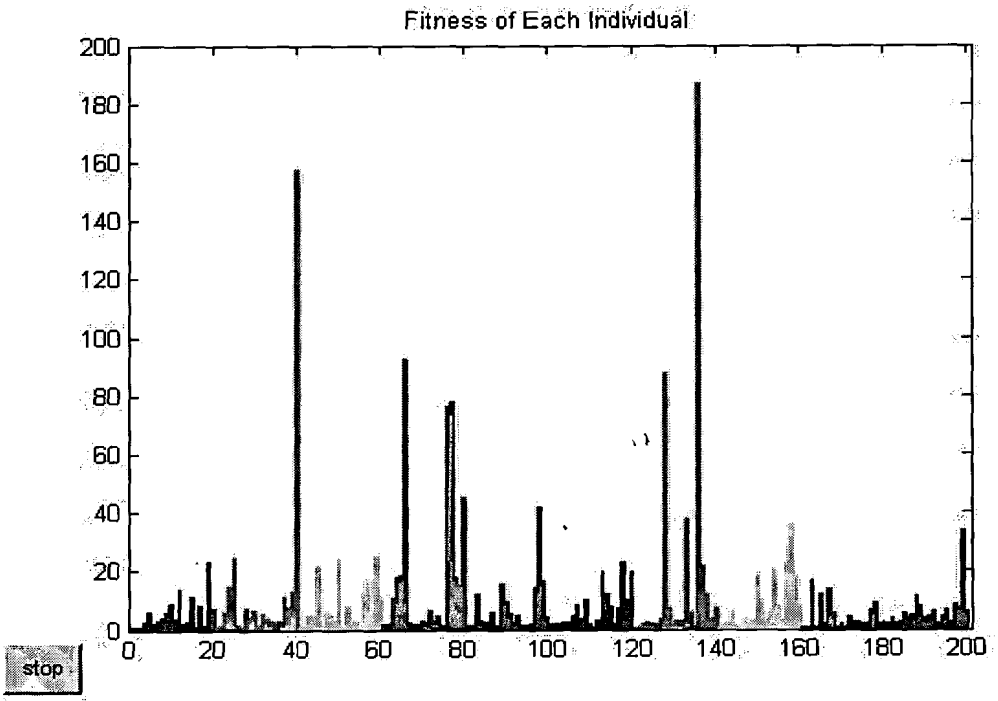


Figure A13: Fitness behavior after 300 generations

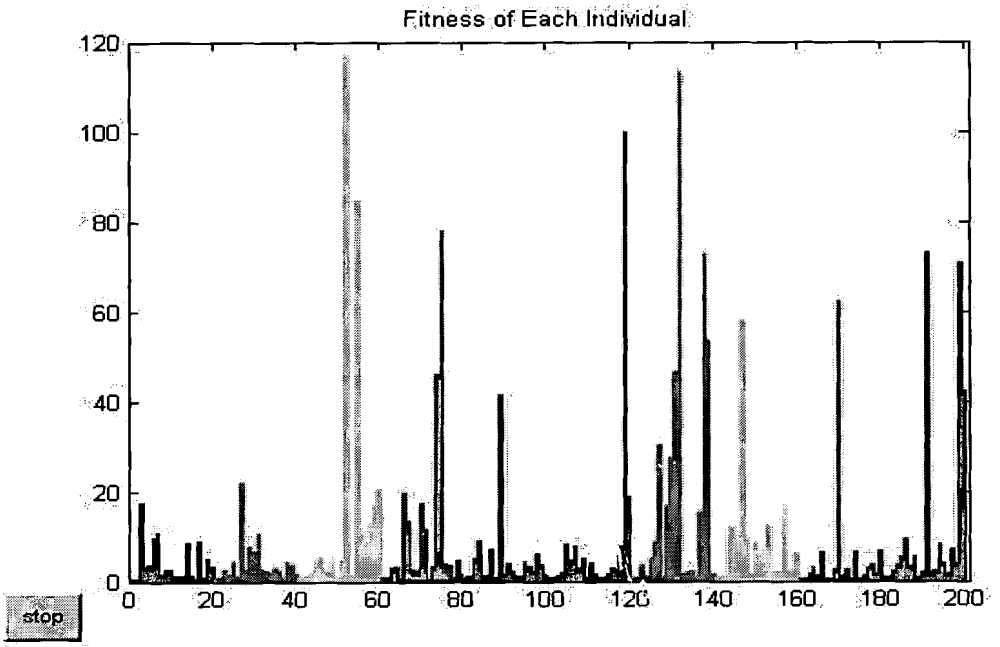


Figure A14: Fitness behavior after 350 generations

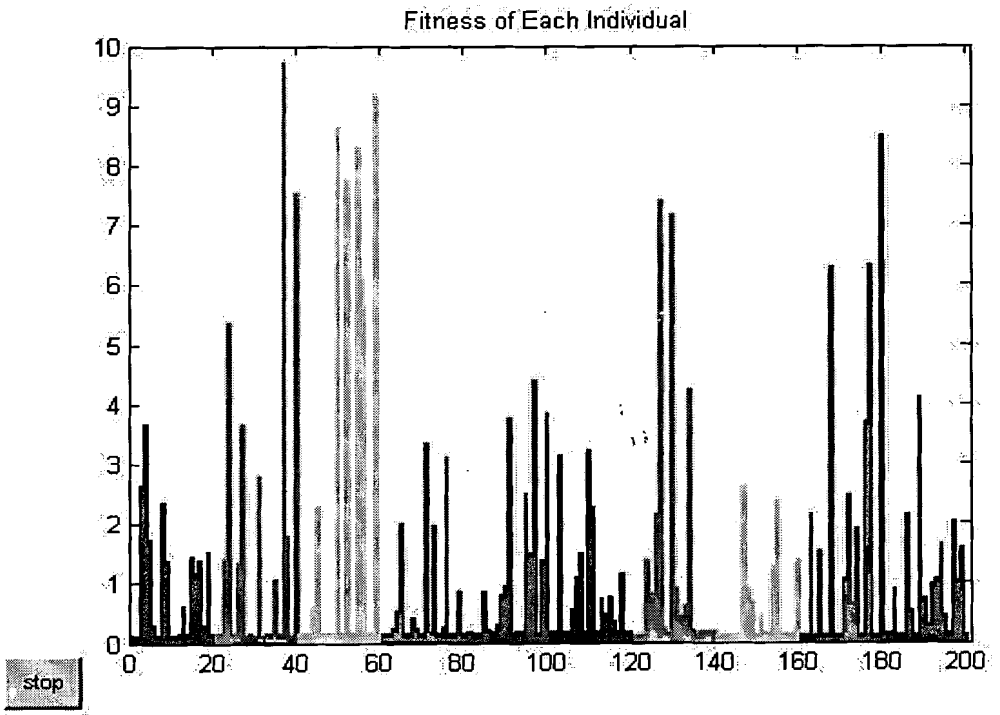


Figure A15: Fitness behavior after 400 gener

Weissinger

Wessinger_main

```
clc;
close all;
clear all;
a=-4; b=4;
W1=a+(b-a).*rand(1,15);
subpop=[20,20,20,20,20,20,20,20];
options = gaoptimset('FitnessLimit',0.000001,'Generations',800,'PopulationSize',subpop);
[W1, fval]=ga(@fit_fun_w1,15,options);

x=0:0.1:3.9;
plot(x,sqrt(x.^2+.5),'color','black');
xlabel('t');ylabel('y(t)');
hold;
y1=output(W1);
title('Analytical & DE NN Solution for weissingers ODE')
plot(x,y1,'*','color','black');
legend('Exact','DE NN',2);
```

Wessinger_fit_function

```
function e=fit_fun_w1(W)
i=0;
for t=0:0.2:4.0

[output,derivative]=eq_parametersw1(t,W);

error(i+1)=(t*(output^2)*derivative^3-(output^3)*derivative^2+t*(t^2+1)*derivative-
(t^2)*output)^2;
i=i+1;
end
e1= sum(error)/i;
t=0;
[in_value]=eq_parametersw2(t,W);
e2=(in_value-sqrt(.5))^2;
t=4;
[in_value]=eq_parametersw2(t,W);
e3=(in_value-sqrt(16.5))^2;
   e=(e1+e2+e3)/3;
```

Wessinger_eq_parameter

```
function [output,derivative]=eq_parameterssw1(t,W)
[fm,n]=size(W);
w=W(1:(n/3));
Alpha=W((n/3)+1:(2*n/3));
beta=W((2*n/3)+1:n);
Mul=Alpha.*w;
Mul2=Alpha.*w.^2;
x=w*t+beta;
for k=1:n/3
   log_s(k)    =1/(1+exp(-x(k)));
   dlog_s(k)   =1/(2+exp(x(k))+exp(-x(k)));
end
output=sum(Alpha.*log_s);
derivative=sum(Mul.*dlog_s);
```

Wessinger_output

```
function our_output=output(W)
[m,n]=size(W);
for j=1:m
   i=0;
    for t=0:0.1:3.9
      w=W(j,1:(n/3));
      Alpha=W(j,(n/3)+1:(2*n/3));
      beta=W(j,(2*n/3)+1:n);
      x=w*t+beta;
      for k=1:n/3
         log_s(k)    =1/(1+exp(-x(k)));
      end
      output1(j,i+1)=sum(Alpha.*log_s);
      i=i+1;
   end
end
our_output=output1;
```

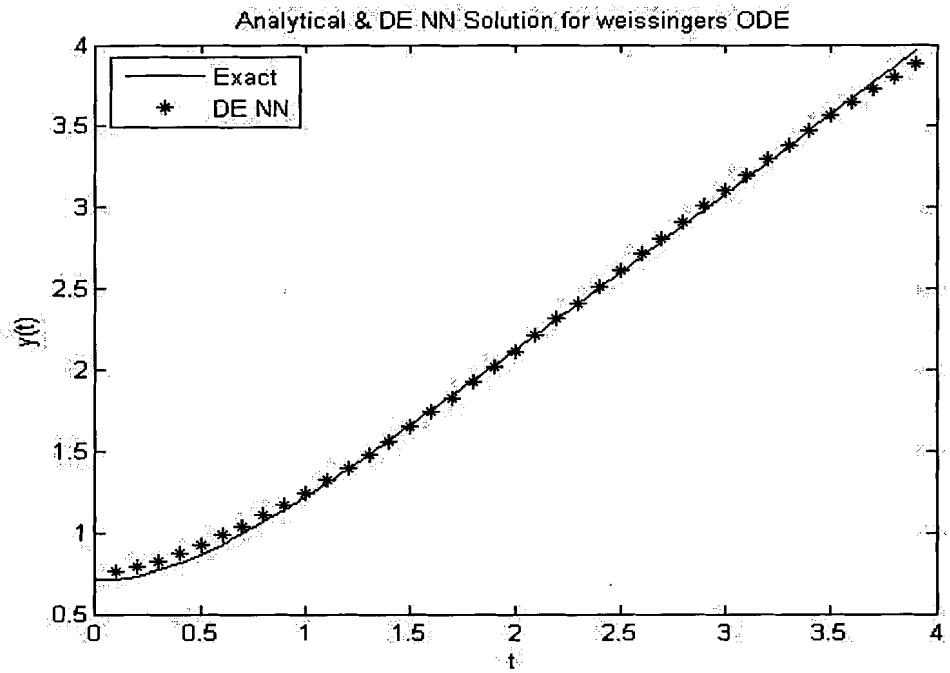


Figure A16: Analytical & DENN solution for Weissinger's ODE

Weissinger's Chromosome Table

0.65	0.61	0.41	-0.31	-0.71	0.63	3.61	1.301	-0.71	-0.62	-0.42	-2.21	0.2	0.801	1.

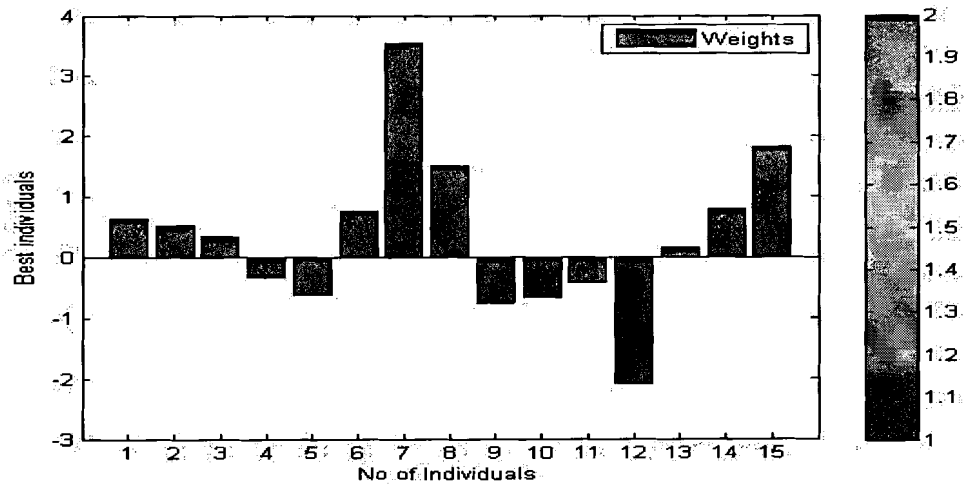


Figure A17: Beast individuals in Weissinger's equation

MSE Result:

$MSE = 7.72e^{-5}$

Generations:

400

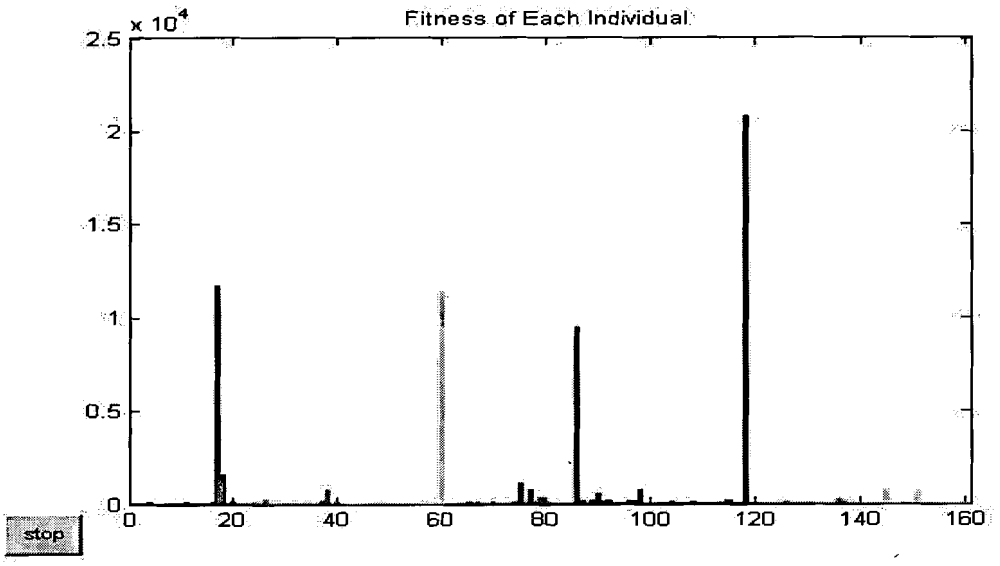


Figure A18: Fitness behavior after 100 generations

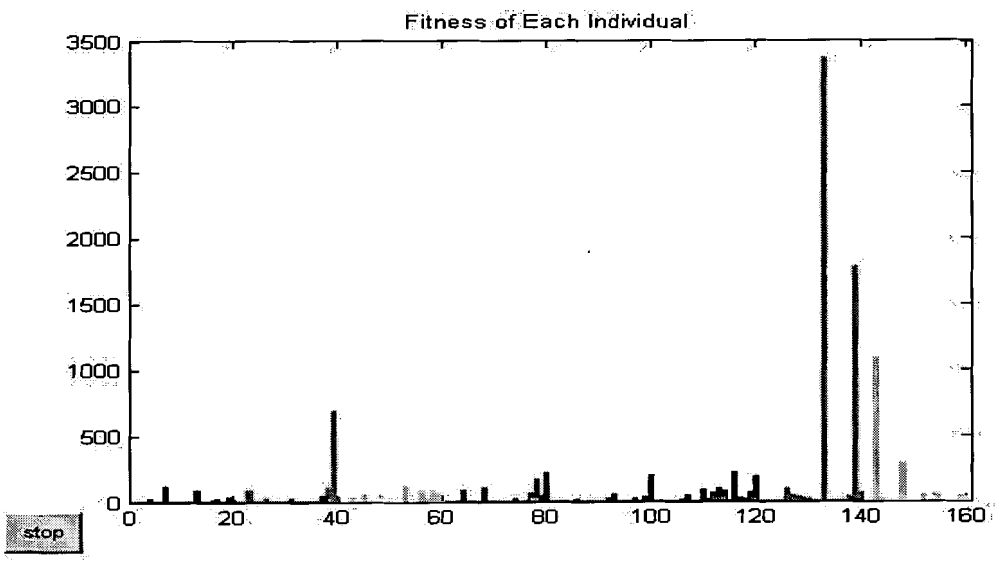


Figure A19: Fitness behavior after 200 generations

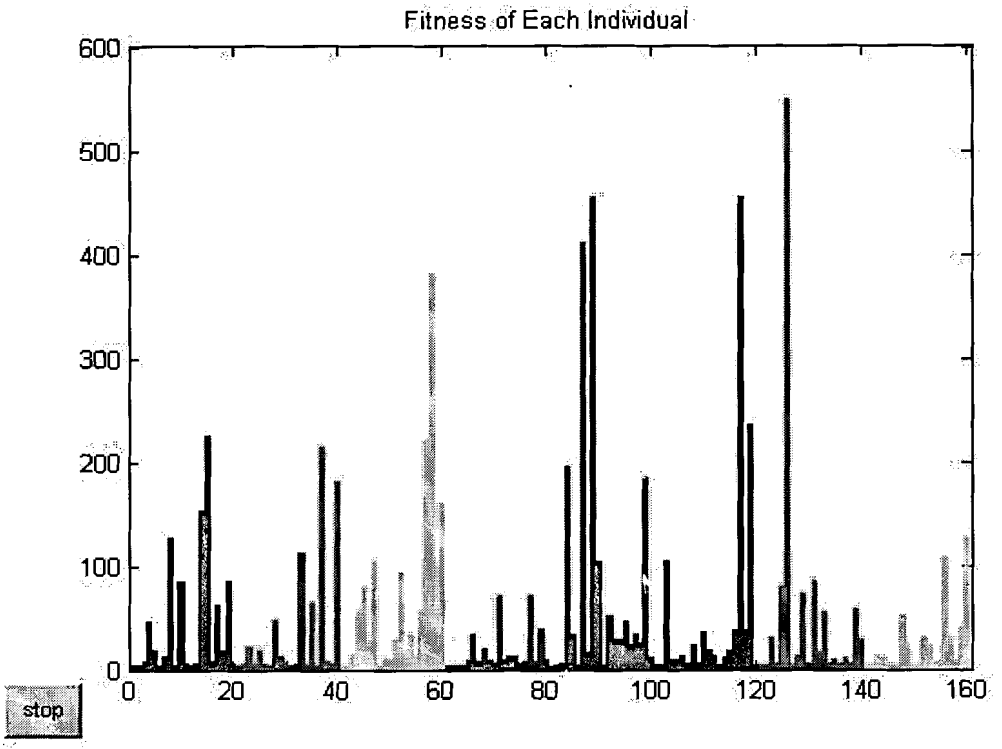


Figure A20: Fitness behavior after 300 generations

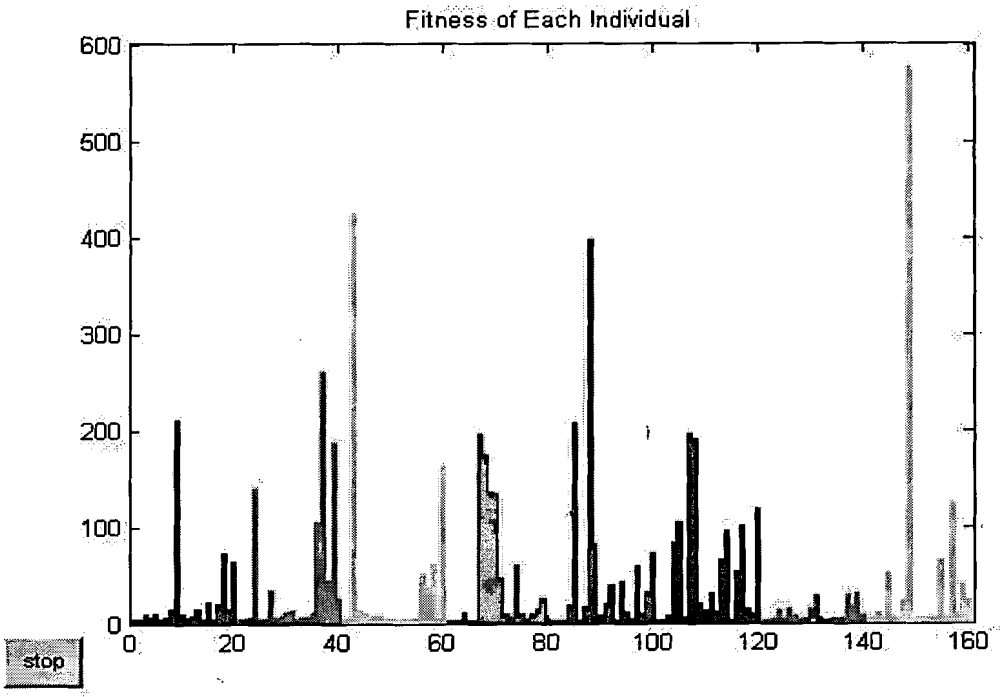


Figure A21: Fitness behavior after 400 generation

Nomenclature

NN	Neural Network
FFDENN	Feed Forward Differential Equation Neural Network
MSE	Mean Square Error
LMS	Least Mean Square
ANN	Artificial Neural Network
IVP	Initial Value Problem
GA	Genetic Algorithm
SOM	Self Organization Maps
ML	Machine Learning
DE	Differential Equation
RK	Range Kutta
PAC Learning	Probably Approximating Correct Learning
ODE	Ordinary differential equation
Logsig	Log sigmoid
BVP	Boundary value problem
AI	Artificial intelligence
GA Fit_fun	Genetic algorithm fitness function
Eq_parameter	Equation parameter

Glossary

Activation:

A node's level of activity; the result of applying the activation function to the net input to the node.

Activation Function:

A function that transforms the net input to a neuron into its activation, it is also called as transfer function.

Architecture:

Arrangement of the nodes and the pattern in the connection links in a neural network.

Axon:

Long fiber over, which a biological neuron transmits its output signal to other neuron.

Bias:

Bias is a weight on the connection between node j and a mythical unit.

Binary Sigmoid:

It is Continuous differentiable S-shaped activation function whose range is between 0 and 1.

Bipolar Sigmoid:

Its range is from -1 to 1 and is also continuous differentiable S-shaped activation function.

Competitive Learning:

It's an unsupervised learning in which a competitive neural network adjusts its weights after the winning node has been chosen

Competitive Neural Network:

A neural network in which, a group of neurons compete for right to become active.

Convergence:

A system is said to be converged if the neuron don't update in next few iterations.

Delta Rule:

Learning rule based on minimization of squared error for each training pattern. Used for single layer perceptron. It is also called LMS and Widrow-Hoff learning.

Epoch:

It is one presentation of each training pattern.

Euclidean Distance:

This is the distance defined between two vectors $x1, x2, x3, ...xn$ and $y1, y2, y3, ...yn$ as following

$$D^2 = \sum_{i=1}^{n} (xi - yi)^2.$$

Excitatory Connection:

These are connection link between two neurons with a positive weight; its serves to increase the response of the unit that receives the signal in contrast, inhibitory connection.

Exemplar:

A vector that represents the patterns placed on a cluster; this may be formed by the neural net during training as in SOM or specified in advance as in the hamming nets.

Extended Delta Rule:

Learning rule based on minimizing the error of the single layered net in which the output inputs may have any differentiable function for their activation function.

Fast Learning:

This is the learning mode for ART in which it is assumed that all weights updates reach equilibrium on each learning trial.

Fault Tolerance:

A neural net is fault tolerant if removing some nodes from the net makes little difference in the computed output , also neural nets are in journal tolerant of noise in the input patterns.

Feed Forward:

A neural net in which, the signals pass from the input \units to the output units (possible through intermediate layers of hidden units) without any connection back to previous layers. In contrast recurrent nets have feed back connections.

Fixed Weight Nets:

Neural nets in which he weight don't change for example Hopfield nets.

Hidden Units:

Units that are neither input neither unit nor output units.

Inhibitory Connections:

Connection links between two neuron such that a signal sent over this link will reduce the activation of the neuron that receive the signal. This may result from the connection having the negative weight or from the signal received being used to reduce the activation of neuron by scaling the net input the neuron receive from other neurons.

Input Unit:

The units that; receive signals from out side the neural net. Typically they transmit the input signal to all neurons to which they arte connection with performing any change. They have identity function as the activation function.

Kohonen Learning Rule:

Weight update rule in which the new weight is convex combination of the old weight and the current input pattern, the coefficient that multiplies the input pattern the learning rate is gradually reduced during learning process.

Kohonen Self Organizing Maps:

A clustering neural net with topological structure among cluster units .

Layer:

Pattern of weighted connection between two slabs of neurons; in neural net literature the term layer is also used frequently for a group of neuron that function in the same way, a slab.

Learning Rate:

A parameter that controls the amount by which; weights are changed during training. In some nets the learning rate may be constant.

References

1. A. Barto, A. Sutton, &R. Anderson,"Neuron like adaptive elements that can solve difficult leming control problems," *IEEE transaction on systems, Man, and Cybernetics.*, 13(5), 834-846. 1983
2. R. Crites, & A. Barto "Improving elevator performance using reinforcement learning," *In D.S.Touretzky, M.C.Hasselmo(eds), advances in neural information processing systems*, 8. 1996
3. L.J. Lin., "Self improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, 8,293-321, 1992.
4. W.F Allman, "Apprentices of wonder: Inside the neural network revolution," *New York: Bantam Books*, 1989.
5. W.E Boyce, and R.C. DiPrima, "Elementary differential equations and boundary value problems," *Fifth edition, john Willey &sons, Inc.,New York*, 1992.
6. H.D. Block,"The Perceptron: A Modal for Brain Functionaing,1,"."Review of ModernPhysics", 34:123-135, 1962. Reprinted in *Anderson&Rosenfild*,pp.138-150,1988.
7. J. Von Neumann, "The Computer and the Brain," *New Heaven:Yale University Press.* Pages 66-82, 1958, are reprinted in *Anderson &Rosenfield*, pp.83-89, 1988.
8. B. Ph van Milligen, V.Tribaldos, J.A.Jimenez, "Neural network differential equation and plasma equilibrium solver," *Physical Review Letters* 75(1995) 3594-3597.
9. H. Kesten, "Accelerated Stochastic Approximation," *Annals of mathematical statistics*, 29:41-59, 1958.
10. T. Kohonen, "Self Organization and associative memory," *(3rd ed.), Berlin:Springer-Verlag*, 1989a.
11. S. Lee "Supervised learning with Gaussian potentials," *In B.Kosko (ed.), neural networks for signal processing. Englewood Cliffs, NJ:prentice-Hall*, pp.189-228, 1992.
12. R.J. MacGregor, "Neural network and brain modeling," *San Diego Academic press*, 1987.
13. R.P. Lippmann, "An Introduction to computing with neural nets," *IEEE ASSP magazine*, 4:4-22, 1987.
14. Tom M.Mitchell, "Machine Learning," *wcb/McGraw-Hill.*
15. S. Haykin, Neural Networks, "A comprehensive foundation," *Prentice Hall, New York*, 1999 2nd Ed.

16. M. Quito, Jr.,C. Monterola, C. Saloma, " Solving N-Body problem with neural network Physical," *Review 86* pp.4741-4744, 2001.
17. H.D. Block.H.D, "The Perceptron: A Modal for Brain Functionaing,1." *Review of ModernPhysics*, 34:123-135, 1962. Reprinted in *Anderson&Rosenfild*, pp.138-150, 1988.
18. J.Von Neumann, "The Computer and the Brain," *New Heaven:Yale University Press.* Pages 66-82, 1958, are reprinted in *Anderson &Rosenfield* ,pp.83-89, 1988.
19. Hetch-Nielsen R. Kolmogorov's, "Mapping neural network existence theorem," *1st IEEE inter. Conf. on neural networks, San Diego CA*, 3(1987)11.
20. K.I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks* 2 (1989) 183.
21. R. Hetch-Nielsen, "Theory of backpropagation neural network," *proc. Int. Joint Conf. on neural networks, Washington D.C New York, IEEE press,* pp. I.593-I.605, 1989.
22. K. Hornik, M. Stinchcombr and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feed forward networks," *Neural Networks 3*, 551-560, 1990.
23. C.M. Bishop, "Neural networks for pattern recognition," *Clarendon Press, Oxford*, 1995.
24. D. Ridder, "Shared weights neural networks in image analysis," *Master Thesis, Delft University of Technology, Delft*, 1996.
25. X.Yao, "Evolving artificial neural networks," *processing of IEEE 87(9)*, 1423-1447, 1999.
26. C.R. Houck, J.A Joines and M.G. Kay, "A genetic algorithm for function optimization," *A matlab implementation, Technical Report NCSU-IE TR 95-09, North Carolina State University, Raleigh NC*,1995.
27. H. Pohlheim, "Documentation for genetic and Evolutionary algorithm toolbox for use with matlab (GEATbx)," *version 1.92, http://www.geatbx.com*, 1999.
28. D. Whitley, "Applying genetic algorithms to neural network problems," *International neural network society* p. 230, 1988.
29. Zbigniew Michalewicz, "Genetic algorithms + data structure= Evolution programs," *2nd edition, Springer-verlag, Berlin; New York*, 1994.
30. W.E. Boyce and R.C. DiPrima, "Elementary differential equations and boundary value problems," *Fifth edition, john Willey &sons, Inc.,New York*, 1992.
31. C. Monterola, C. Saloma, "Characterizing the dynamics of constraints physical systems with unsupervised neural network," *Physical Review E 57*, 1247R-1250R, 1998.

32. C. Monterola, C. Saloma, "Solving the non linear Schrodinger equation with an unsupervised neural network," *Optics Express 9*, 72-84, 2001.

33. I.Rivals and L.Personnaz, "Mono-Layer polynomials and multi-layer Perceptrons for nonlinear modeling," *journal of machine learning research 3*, 1383-1398, 2003.

34. P. Koduru, H.W. Hsu, S. Das, S. Welch, J.L Roe, "Dynamic system prediction using temporal artificial neural networks and multi-objective genetic algorithms," *Proceeding of the IASTED International Conference on Computational Intelligence*, pp. 214-219, 2005.

35. J. Gao, B. Liu, "Fuzzy multilevel programming with a hybrid intelligent algorithm," *computers and mathematics with applications*, 49(9-10), pp.1539-1548, 2005. *Cited 12 times. Uncertainty Theory Laboratory, Department of Mathematical Sciences, Tsinghua University, Beijing 100080, China*

36. Adamiec-Wojcik,I.,Warnas, K.,Wojciech,S, "Braking torque optimization in time domain" *proceeding of the 2004 International Conference on Noise and Vibration Engineering, ISMA*, pp.1981-1995,2004.

37. H.C. Fayad, R.C. Peralta, "Multi-Objective conjunctive use optimization," *Processing of 2004 World Water and Environmental resources Congress: Critical Transition in water and Environmental resources Management*,pp.1800-1809, 2004.

38. T. Kanamaru, M. sekin, "Detecting chaotic structures noisy pulse trains based on interspike interval reconstruction," *Biological Cybernetics*, 92 (5), pp. 333-338, 2005.