8-1996

Applications of Artificial Neural Networks in Interactive Simulation

Payman Jula
Western Michigan University

APPLICATIONS OF ARTIFICIAL NEURAL NETWORKS IN INTERACTIVE SIMULATION

by

Payman Jula

A Thesis
Submitted to the
Faculty of The Graduate Collage
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Industrial and
Manufacturing Engineering

Western Michigan University
Kalamazoo, Michigan
August 1996

ACKNOWLEDGMENTS

I can not find the words to express my feeling at this moment. When I started this project, I didn't know what I would be put through. The guidance and support of many people helped me in this way. I would like to thank the Computer Aided Engineering Center's staff, especially Mr. Sridhar Erra, for their understanding and that they allowed me to extensively use the labs' facilities. I would also like to thank my advisor, Professor Azim Houshyar, for his guidance. I also want to thank Professor Frank Severance for his valuable suggestions. I offer my appreciation to Professor Richard E. Munsterman and Professor Anil Sawhney for their assistance and knowledge.

The last but not least, I would like to thank my family. Without their support I definitely wouldn't be able to do this project. I owe them for their sacrifice for my entire life. I hope they find their sacrifice worthwhile.

Payman Jula

APPLICATIONS OF ARTIFICIAL NEURAL NETWORKS IN INTERACTIVE SIMULATION

Payman Jula, M. S.

Western Michigan University, 1996

Although there have been many improvements in simulation technology over the past few years, it still suffers from many limitations. Simulation methods are usually time consuming and hence not suitable for the interactive decision making processes.

In this project, applications of Artificial Neural Networks (ANNs) to simulate manufacturing systems have been studied. The backpropagation Multiple Layer Perceptrons (MLPs) have been applied to simulate manufacturing systems. Some guidelines for developing appropriate ANNs have been presented. The results of ANN approach have been compared to those of conventional simulation methods.

TABLE OF CONTENTS

CHAPTER

Table of Contents--Continued

CHAPTER

Table of Contents--Continued

LIST OF TABLES

LIST OF FIGURES

List of Figures --Continued

CHAPTER I

INTRODUCTION

Although the progress made in computer technology in recent years has provided more abilities and power, computers still are not able to solve most industrial problems. On the other hand, humans are able to realize systems, classify and recognize texts, pictures and voices in a short period of time. Currently computers do not have these features. These abilities have motivated scientists to research the way humans think and the methods that the human brain uses to analyze problems. The success of researchers in introducing Artificial Intelligence motivated them to develop similar fields such as Artificial Neural Network, Fuzzy Logic and Genetic Algorithm. Each of these fields has shed light on one part of the human's capabilities. Throughout history, human beings have been interested in the sciences which help them solve their problems. One of these problems is faced by industrial engineers when they use simulation.

Simulation methods have the ability to manipulate large amounts of data, perform mathematical calculations and predict the performance of complex systems with some accuracy. But simulation methods are usually time consuming and hence not suitable for interactive decision making processes. In particular, if the decision makers are using the simulation on-line, they need to obtain the recommendation as

soon as possible. Furthermore, in many situations where the details of the system are not well known and only the input data and the output data are available, there is a need for a quick and rough estimation of the system's response to a new set of inputs. Unfortunately, traditional simulation methodology fails to respond to these particular situations.

On the other hand, experienced individuals can occasionally predict the output of systems much better and faster than computers. Knowledge, experience and intelligence are factors that help these experts to out-perform computers. By mimicking the human capabilities in the computer, researchers have strived to modify the simulation methodology to make it more intelligent. One approach to intelligent simulation is through Artificial Neural Networks (ANNs). The capabilities of ANNs in parallel processing, learning, generalization, classification, pattern recognition and memorizing make them good candidates to enhance the simulation methodology. Furthermore, ANNs' adaptability makes them suitable tools for dynamic systems. The potential applications of ANNs in simulation can range from having a small role in a simulator to being a stand-alone substitute to the existing simulators. For the simulation of manufacturing systems, two approaches might be considered: (1) to create a library of modules of the manufacturing models and assemble these modules to build more complex models, and (2) to consider the whole system as a black box and try to find an ANN estimator for the system.

Because the internal workings of ANNs are not clearly known, researchers have looked at the ANNs as black boxes that can be identified by their input/output

relationships. But several questions remain unanswered. For instance, what is the relation between the ANNs' inputs and outputs and the real world system? Are they the input and output of the real system, a part of the real system, or a combination of these? How can the ANNs be used instead of traditional simulation software applications and/or as a part of them? If the ANN concept can be applied to conventional simulation, what is the best architecture of the network? What is the best learning method? How many layers and nodes are needed?, etc.

In this thesis, the applications of ANNs to the simulation of manufacturing processes are studied and their advantages and disadvantages are discussed. In an attempt to present a systematic approach to the application of ANN, this thesis surveys the existing literature and examines the learning methods and structures of ANNs. Additionally, to answer some of the above questions and contemplate some concerns on the applicability of ANNs to interactive simulation models, some recommendations are presented. Based on the suggested guidelines, first an M/M/S queuing system is modeled by an ANN. This system shows the ability of ANNs in simulating static systems. The obstacles for the smooth operation are discussed to give the industrial engineers the feeling of a typical procedure of developing an appropriate ANN. Later, a simple manufacturing system is modeled using ANNs. This manufacturing system has stochastic behavior. Three approaches are suggested to capture its stochastic behavior. Finally, a modular approach is applied to this case and the results are critiqued. In brief, the manuscript is in this order:

In Chapter II, existing simulation methodologies are briefly discussed, along

with the classification of systems and the procedure used to make a suitable model for the systems.

In Chapter III, the basic concepts of ANNs, their learning rules and structures are reviewed. A useful structure and a learning method, which is used in later chapters, are explained.

In Chapter IV, the existing literature on the application of ANNs in industrial engineering, especially in simulation methodology and related fields are reviewed.

In Chapter V, some guidelines for implementing the appropriate ANN to simulating systems are presented.

In Chapter VI, simple queuing systems are modeled by using ANN, as a first attempt in creating a library of ANN modules which can be used to model complex systems. The results and methodologies are explained. A simple manufacturing system is also modeled by three different ANNs.

In Chapter VII, some suggestions for future studies in the field are offered. Computer source codes and bibliography are also attached.

CHAPTER II

SIMULATION METHODOLOGY

Introduction

Human societies are challenged by more complicated problems than ever before. The real world problems are growing in size and complexity. The need to develop tools and techniques for solving these problems has led to the use of computers; simulation has become one of the most powerful and widely used tools. Simulation is a popular tool in the analysis and design of complex systems, and is a decision support tool in monitoring and controlling these systems. Simulation modeling is a valuable tool for engineers, system analysts and researchers. It is a tool which aids managers in making decisions among different options.

Simulation can be used to evaluate the performance of existing or proposed systems. It can be used to evaluate the design of a new system, or evaluate changes to an existing system. It can be used to test operating policies and control algorithms, when testing and experimentation with the real system would be too expensive, too disruptive or too risky. It also helps engineers to do sensitivity analysis to answer what-if questions. In this chapter, computer simulation and its basic concepts are defined. The application and procedure for development of models are surveyed and the limitations and pitfalls of existing simulation methodologies are reviewed.

Definitions

A simulation model is a simplified representation of a system intended to enhance our ability to understand, predict, and possibly control the behavior of the system. In simulation terminology, "system" is "the collection of entities, e.g. people or machines, that act and interact together toward the accomplishment of some logical end" (Law, 1991). Pritsker (1986) defines computer simulation as "the process of designing a mathematical-logical model of a real system and experimenting with this model on a computer".

For simulation modeling, the system should be presented in the forms which are acceptable to a computer. If a system can be presented by a set of variables, then manipulation of the variable values simulates movement of the system from state to state. Thus, simulation involves observing the dynamic behavior of a model over time. The state of a system can change continuously over time or at discrete instants in time. The observed output of a process will be either deterministic or stochastic. The basic concept of simulating a system portraying the changes in the state of the system over time for both discrete and continuous systems are the same (Pritsker, 1986). Law (1991) categorizes the simulation models as follows:

1. Static vs. Dynamic Simulation Models: A static simulation model represents a system at a specific time, or a system in which time is not important. A dynamic simulation model is a representation of a system which changes over time.

2. Continuous vs. Discrete Simulation: A discrete system is one for which the

state variables change instantaneously at distinct points in time. In continuous systems, the state variables change continuously over time.

3. Deterministic vs. Stochastic Simulation Models: Deterministic model is the one which does not contain any probabilistic (i.e. random) components. In these models, when the input values and relationships in the model are identified, the output is determined. On the other hand, a stochastic simulation model has at least some random input components.

Applications of Simulation

The applications of computer simulation have grown rapidly in the past four decades. Advances in computer technology along with the continuing development of simulation languages have been important factors in this growth. Simulation is an iterative experimental problem-solving technique that can be used at different stages of the process design and process control. Simulation models can be used at four levels (Pritsker, 1986): (1) as explanatory devices to define a system, (2) as analysis vehicles to determine critical issues, (3) as design assessors to synthesize and evaluate proposed solutions, and (4) as predictors to forecast and aid in planning future developments.

Presently, simulation is widely used. A few examples are its application in manufacturing operations, project planning and control, health care systems, financial planning, environmental studies and transportation systems. Simulation makes it possible (Nuila, 1993):

1. To evaluate the performance before a newly designed system is operable. Before construction, new manufacturing facilities must be laid out, supplied with material handling equipment, documented with operating procedures and cost justified. In this case, the real system does not exist and it is expensive, hazardous, or time consuming to build and experiment with a prototype. Simulation methodology is recommended as a powerful tool for evaluating the performance of the potential systems in an operations planning and design phase.

2. To compare different operating strategies of a present system without changing the system's settings. Within an operating facility, management must react to a rapidly changing environment to meet production objectives. Decisions on work order release, scheduling and staffing must be made in light of new orders, equipment availability, absenteeism and other factors. In most cases, experimentation with the real system is usually expensive, dangerous, or likely to cause serious disruptions. Simulation methodology is recommended as a powerful tool for evaluating the various strategies in an operations control phase.

3. To expand or compress the system's operating time. Simulation is a useful tool to study the past, present, or future behavior of the system in real time, expanded time or compressed time.

4. To improve understanding of systems and enhanced communication between different parties.

Simulation Languages and Simulators

There are number of software packages specifically designed for simulation. The following are advantages of using such special-purpose packages when performing a simulation study (Shannon, 1970): (a) reduction of the programming task, (b) guidance in concept articulation and model formulation, (c) aid in communication and documentation of the study, (d) flexibility in embellishment or revision of the model, and (e) provision of the common support functions required in any simulation.

Simulation packages can be categorized into two major categories: simulation languages and simulators. Simulation languages help analysts develop models by writing programs using the language's modeling constructs. On the other hand, simulators can be used to develop a model with little or no programming. Examples of existing simulation languages are Automod II, GPSS/PC, PCModel, SIMAN/Cinema, SIMSCRIPT 11.5 and SLAMSYSTEM. Examples of simulators are FACTOR, SIMFACTORY and PROMODEL.

An appropriate package should be selected based on the requirements of the modeler and the features of the package. For a detailed analysis of an existing system, simulation languages are usually used. For an aggregate analysis of a proposed system, simulators are usually recommended (Nuila, 1993). The desirable features of the packages depend on the specifics of the problem, but can be categorized into (Law, 1992): (a) general features, (b) animation capability, (c) material handling capability,

(d) statistical capability, (e) report capability, and (f) customer services. Emshoff and Sisson (1970) listed the following support functions as required for any simulation language: (a) generation of random variates, (b) management of the simulated clock, (c) collection and recording of output data, (d) summarizing and statistical analysis of output data, (e) detection and reporting of error conditions, and (f) generation of standard output reports. These supporting functions are required for simulators as well.

Procedure for Developing a Model

The main purpose of modeling is to establish interrelationships between entities of a system. The process for the successful development of a simulation model consists of (Nuila, 1993):

1. Problem formulation: Stating the problem clearly, logically, and unambiguously is the first step in building a model. Simplicity is an essential criterion of a good model. Manpower, time and cost should be studied in this phase. Models are expressed in terms of (a) goals, (b) performance criteria and (c) constraints.

(a) Goals: The goals are the objectives the modeler is trying to achieve. For example: maximize throughput; reduce work-in-process; and reduce the work-force or maintain it at a fixed level.

(b) Performance criteria: The criteria are the specifications by which different alternatives are judged. For example: throughput which should be maximum; and work-in-process which should be minimum. The goals usually can be considered as

performance criteria. The more goals achieved, the better the system performs.

(c) Constraints: The constraints are the limitations that control the availability of different resources. Possible candidates are restrictions on availability and/or the use of men, machines, money, time, space and data.

2. Model building: In accordance with the problem formulation, the system should be expressed into mathematical-logical relationships.

3. Data acquisition: This step includes the identification, specification, and collection of data. The required data for specifying input parameters and probability distributions should be collected. It is necessary to characterize the random elements of a system by particular probability distributions. To select an appropriate distribution for an input process, the analyst must understand some of the basic properties of the common distributions and the circumstances in which those distributions arise. Another set of data should be gathered for validation. The performance criteria measures are usually selected for this purpose.

4. Model translation: The model should be presented in a way that is acceptable to the computer. The modeler should decide which computer software is suitable for the model. e.g., a general purpose language program; a simulation language; or a simulator.

5. Verification: In this step, the modeler verifies that the implemented computer program executes as intended. Most common techniques for verification include: (a) developing the program in a modular manner, (b) checking the output for any questionable results, (c) using interactive debuggers and traces to locate mistakes,

(d) systematically going through the program and check the codes in each step, and (e) using animation to observe the system's behavior.

6. Validation: In this step, the modeler checks if the desired accuracy between the simulation model and the real system exists. Most common techniques for validation include:

(a) The animation should be used to observe the system's behavior,

(b) For an existing system, steps should be taken to ensure that the model's performance measures closely follow those of the existing system. The model should then be modified to include the proposed changes,

(c) For a new system, the model performance measures should almost be the same as the proposed method,

(d) The techniques such as goodness-of-fit test should be applied to ensure that the computer program's output resembles output data taken from the actual system.

7. Strategic and tactical planning: The experimental conditions for using the model should be established in this step. The analyst must specify the appropriate choice of: (a) the length of each simulation run; (b) the number of independent simulation runs; (c) the initial conditions for each simulation run; and (d) the length of warm up period, if required.

8. Experimentation: The simulation model should be executed. The obtained output should be saved and shown in an appropriate way.

9. Analysis of results: The simulation outputs should be analyzed and

recommendation should be made for solving the problem.

10. Implementation and documentation: Finally, the decisions should be implemented and the model and the results should be documented for further study. No simulation process should be considered complete without its documentation for future implementation.

Simulation's Limits and Pitfalls

Along with widespread use of simulation has come a great deal of misuse. A few of the reasons for this widespread abuse are (Nuila, 1993):

1. The simulation always simulates something, but there is no reason it should simulate what the simulator had in mind.

2. Sometimes, computer outputs are taken as gospel truth.

3. Simulation languages have succeeded in making it easier to achieve impressive simulations, without making it easier to achieve valid simulations.

4. The promise of simulation is so great that it is easy to confuse hope with achievement.

Furthermore, even the suitable models suffer from some limitations. A well developed simulation model is expected to help researchers do tasks such as sensitivity analysis, optimization study and answer inverse questions. Unfortunately, the dynamic nature of simulation models makes them time consuming, especially when long runs and/or several replications are needed. Each run of a stochastic simulation model produces only estimates of the model's true characteristics for a particular set of input

parameters. Thus, several independent runs of the model will be required for each set of parameters to be studied. This makes simulation a slow iterative experimental problem-solving technique. So, simulation is not a fast technique for such tasks as optimization and sensitivity analysis.

Specifically, simulation techniques fail when the factor of time becomes important. In many cases, there is a need for the recommendations to be offered as soon as possible. For example consider a system which is supposed to reach a goal at a certain time. Unfortunately, because of unforeseen circumstances, the system is short of the goal. Thus, there is a need for recommendations to correct the system to catch up with desired schedule as soon as possible. The possible approaches to solving the problems caused by computational burdens of simulation are to obtain more powerful hardware, rewrite simulation to more computationally efficient way or develop fast approximations to simulation. In many cases, the first and second approaches have already been taken or were impractical due to lack of capital funds or the availability of simulation programmers. Thus, the third approach, approximating computer simulation, needs to be examined (Kilmer, 1996).

Simulation models are often expensive and time-consuming to develop. The relation between independent and dependent variables and the internal workings of the system should be very well known. The modeler should analyze the statistical behavior of the inputs and the system. Analyzing the internal part of a system is not always easy. There are some situations where the input and the output of the system are available in the form of databases. For example, the number of workers and the number of

machines in a shift can be considered as the input, and the throughput of the system can be considered as the output of the system. There is a need to find the output of the system from the set of new inputs which are not in the database.

Artificial Neural Networks are good candidates for solving the above mentioned problems of simulation. Within the next chapters, the capabilities of Artificial Neural Networks to solve the simulation's limitations will be discussed.

CHAPTER III

ARTIFICIAL NEURAL NETWORKS

Introduction

Artificial Neural Networks are one of the most important research subjects in recent years. Although there were a few practical applications of neural networks up to the late 1980's, the number of applications of ANN is amazing now. Currently, many engineering fields are trying to find innovative ways in which to use the Artificial Neural Networks in their real world applications. The capabilities of Artificial Neural Networks in parallel processing, learning, generalization, classification, pattern recognition and memorizing make them play important roles in industry, business and science.

The capabilities of ANNs are due to their simple-nonlinear computational elements which are parallel and densely interconnected. These computational elements are connected as networks through the use of weights. ANNs usually do their tasks by changing some of these weights. Instead of serial and sequential or systematic and algorithmic methods, which are common in the new computers, an ANN chooses parallel and non-systematic methods, as the human brain does. These characteristics make ANNs good candidates for solving experimental, multiple input-multiple output and non linear problems. Therefore, the "ANN" term is used to describe different

structures of processing elements, which introduce a new method of calculation. The main goal of ANN research is not to introduce machines which are able to do arithmetic calculations faster than the existing computers. Rather, the goal is to introduce machines which can be used in those fields that human beings perform better than computers do. So, the ANNs are complements of existing computers rather than their competitors.

In this chapter, natural neurons and their corresponding artificial neurons are discussed. Different network structures and learning rules are explained. Finally, Multilayer Perceptron (MLP) networks, which will be used in other parts of this work, are discussed.

Natural Neurons

In this section a simplified sketch of a natural neuron is described. There are four important parts in a biological neuron: (1) a neuron cell body called Soma, (2) branching extensions called dendrites, (3) an axon that carries the neuron's output to the dendrites of other neurons, and (4) synapses which connect different neurons together.

The soma contains the cell nucleus, various bio-chemical factories and some other components. A neuron operates by receiving signals from other neurons via dendrites. The combined stimuli from these input signals, in excess of a certain threshold level, activate a region called an axon hillock, where an outgoing tendril called an axon connects to the cell body. The axon then transmits the neuron's output

to other neurons through their dendrites. To transfer from destination neuron to target neuron, the signals pass a region called the "synapse region". In this region, these signals are controlled by biochemical agents. This process is usually modeled in electronic neurons by the changing of weights. The synapse represents the junction between an axon and a dendrite. The process of thinking is actually the collective effect of the presence or absence of firings in the pattern of synaptic connections between neurons. Figure 1 shows the simplified sketch of a natural neuron.

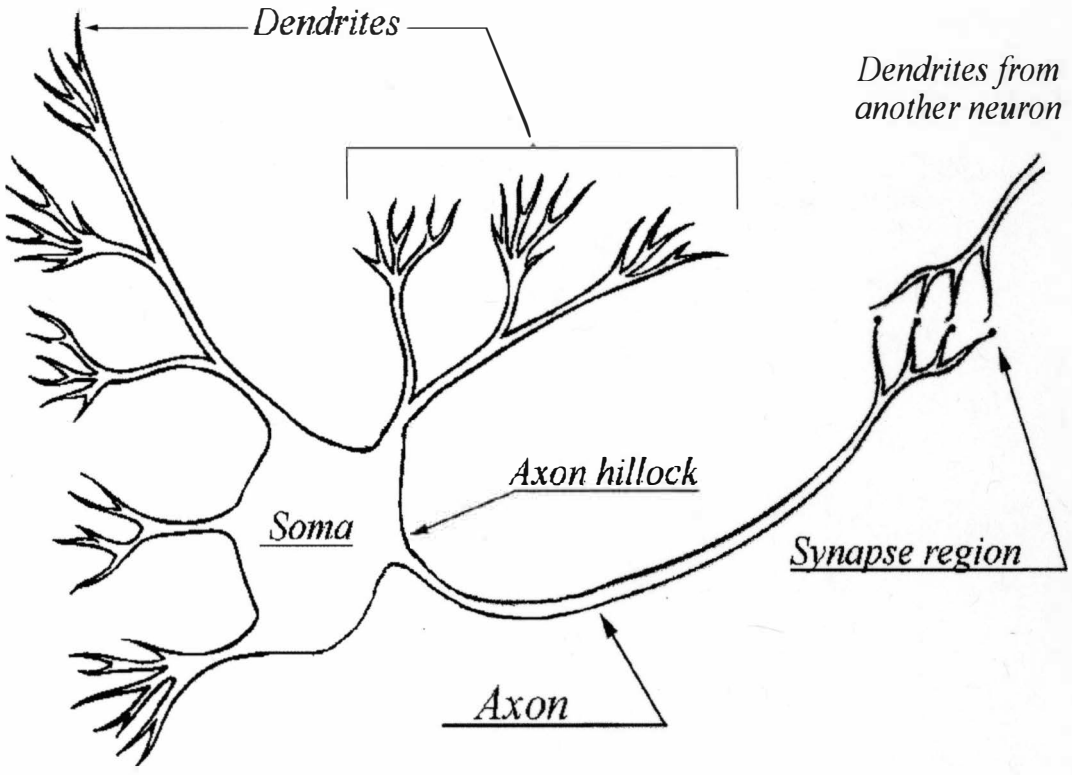


Figure 1. A Biological Neuron.

At rest, the neuron's electrical potential is around 40-60 millivolts. In the firing process, the potential will raise to 90-100 millivolts. This prompts a change in the potential, creates an electrical impulse which travels between 0.5 to 100 meters per second and lasts for about 1 millisecond. The neurons can not fire continuously. They need to take a rest - at least 10 milliseconds - before they can fire again.

If the signal speed or rate were the criteria for comparing the performance, the electronic computers would beat the human brain. With a speed of 200,000,000 meters per second and a switching rate of 100,000,000 per second, today's computers have a 2,000,000 fold advantage in signal transmission speed and a 1,000,000 fold advantage in signal repetition rate. But the factors that make the human brain think, are not solely the signal's speed or the rate of firing of neurons.

Although the neuron's switching time is about a million times slower than current computer elements, they have a thousand fold greater connectivity than today's super computers. It is estimated that the human nervous system contains over 100 billion (10^{11}) neurons and 10^{14} synapses. Studies of brain neuroanatomy indicate more than 1,000 synapses on the input and output of each neuron. Therefore, the human brain is not as quick as an electronic computer at arithmetic, but it is many times better and more capable at pattern recognition, learning and intelligence.

Artificial Neurons and Networks

An ANN is characterized by three characteristics: (1) basic processing elements, (2) topology or structure, and (3) learning rules. These are described next.

Basic Processing Elements

The Processing Elements, which are used in the network, usually are called "neurons", "nodes" or "units". In Figure 2, a simplified artificial neural model is shown. This model consists of multiple inputs and multiple outputs. Each input is multiplied by a weight. The neuron will combine these weighted inputs and, with reference to a threshold value and activation function and output function, use these to determine its output.

A new activation of the unit is computed from the output of preceding units with the current unit, the old activation of the unit and its bias. In many of the existing ANNs, the net function computes the net value simply by adding weighted activation. Then, the activation function converts the results with a function. A general neural model is shown in Figure 3.

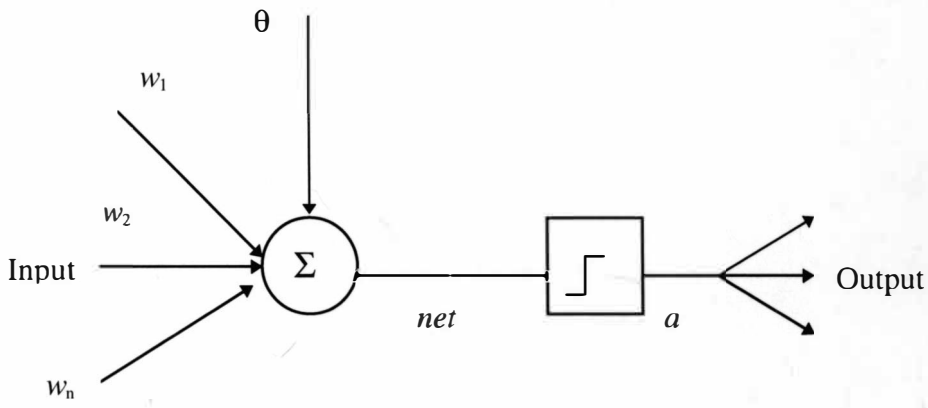
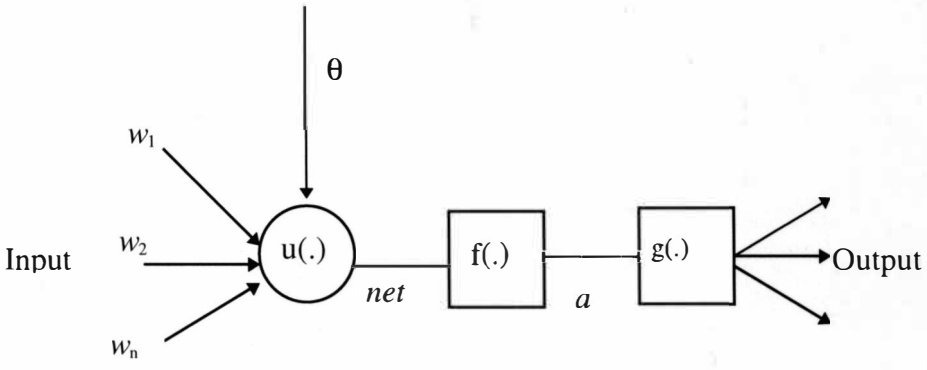


Figure 2. A Simplified Artificial Neural Model.



Legend. The net value is represented by a net function u(.) and activation function is shown by f(.) and output function by g(.).

Figure 3. A General Artificial Neural Model.

The general formula for activation function is:

$$a_j(t+1) = f_{act}(net_j(t), a_j(t)) \quad (1)$$

Where:

f_{act} activation function of unit j

$a_j(t)$ activation of unit j in step t

$net_j(t)$ net input in unit j in step t

With the additive net function, the net input $net_j(t)$ is computed with

$$net_j(t) = \sum_i w_{ij}\ o_i(t) - \theta_j \quad (2)$$

For example, $f_{act}(x)=1/(1+e^{-x})$ yields the well-known logistic (sigmoidial)

activation function. The activation function is shown as:

$$a(t+1) = \frac{1}{1 + e^{-(\sum_i w_{ij} o_i(t) - \theta_j)}} \quad (3)$$

$a_j(t)$	activation of unit j in step t
$net_j(t)$	net input in unit j in step t
θ_j	threshold (bias) of unit j
$o_i(t)$	output of unit i in step t
j	index for some units in the net
i	index of a predecessor of the unit j
w_{ij}	weight of the link from unit i to unit j

The output function computes the output of every unit from the current activation of the unit. The output function makes it possible to process the activation before an output occurs.

$$o_j(t) = g(a_j(t)) \quad (4)$$

$a_j(t)$	activation of unit j in step t
$o_j(t)$	output of unit j in step t
j	index for all unit in the net

Since the output function is usually set to identity function, many researchers combine the f and g function together. The output function has been addressed in this work due to consistency with the software used for modeling the neural networks.

Structures of ANNs

The neurons themselves do not have as much ability to perform as we expect them to. The connections among neurons, which are called weights, make them powerful to do their jobs. All ANNs perform essentially the same function: they map vectors. In this process, they accept a set of inputs (an input vector) and produce a corresponding set of outputs (an output vector). As shown in Figure 4, a vector mapper produces a set of outputs according to the input set and the mapping relationship encoded in its structure (Wasserman, 1993). Examples of input vector in manufacturing system are number of machines, number of workers and processing time. Throughput and Work In Process are examples of the output vector.

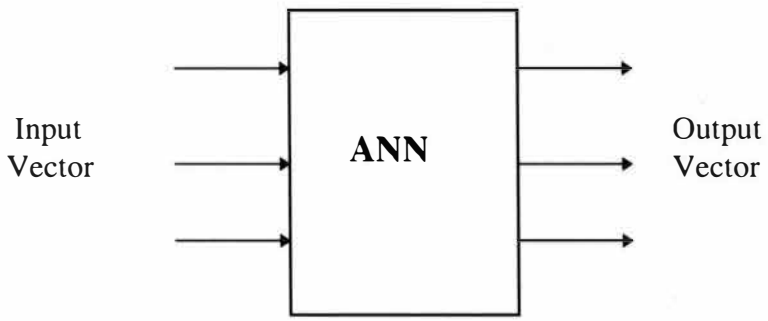


Figure 4. An Artificial Neural Network as a Vector Mapper.

The structure of the connections between the nodes are very important. Two main categories of network topologies are: (1) Feed-Forward Nets, and (2) Recurrent Nets.

Feed-Forward Nets

In these networks the signals flow only from input to output. Feed-forward networks have no memory, they are capable of implementing only static mappings. The mapping relationship between input and output vectors are static when each application of a given input vector always produces the same output vector (Wasserman, 1993). From a mathematical point of view, feed forward static networks are nonlinear functions in the form of $y = G(x)$, where $x \in R^n, y \in [0,1]^m$ or $y \in R^m$, where m and n are integers that represent the dimensions of x and y (Hush, 1993). One of the most important feed forward static networks is the Multiple Layer Perceptron (MLP) network. This network will be explained later in this chapter.

Recurrent Nets

In these networks the signals can flow forward and backward. Adding feedback to feed-forward networks makes them recurrent. Recurrent networks have memory and they are suitable for estimating dynamic systems. Dynamic systems are the systems where the output produced depends upon previous, as well as current, inputs and/or outputs. Dynamic networks' node equations are typically described by differential or difference equation. The Hopfield network is an example of recurrent nets.

Although the MLPs are used for processing static systems, they are also able to process time series data. This application will be elaborated later in this thesis.

Learning Rules of ANNs

Using an ANN has two phases: the learning phase and the recall phase. In the learning phase, the network learn the behavior of the system based on the training data. In the recall phase, the trained network tries to estimate the response of the system to a new set of data. These two phases can also occur simultaneously. In these cases, the network will learn the pattern on-line and meanwhile it will recall the patterns based on its previous experiences. By automatic adjustment of coefficients and parameters of the network, an ANN can be trained. This process is usually called Learning Algorithm which usually consists of the changes in the network's weights. By this definition, the Learning Algorithm does not change the structure of the network.

Extensive research has been done for developing the new learning procedures which train the ANNs through changing the number of layers or neurons. These techniques can be divided into three main categories (Bebis, 1995): pruning, constructive and weight sharing. Unfortunately, there is no mathematical or heuristic solution for optimization of the number of neurons, links and layers. Currently, Genetic Algorithm is a promising approach in this field. Throughout this thesis, only changing of weights will be considered as the learning method. In the next chapters, some recommendations for optimizing the performance of ANN through changing the number of layers and neurons will be offered.

There are two main approaches for learning in an ANN: (1) Supervised Learning, and (2) Unsupervised Learning.

Supervised Learning

In this method, the network is trained on training sets consisting of input-output vector pairs. One vector is applied to the input of the network and the desired results are considered as output of the ANN. These output signals are usually called "teacher" or "supervisor". The teacher is responsible for teaching the network until the desired output is obtained. The training is an iterative process. In each iteration, the network is trained by adjusting the weights so as to minimize the difference between the desired and actual output. Each iteration is called an epoch. After training, the performance of the network is criticized based on its power of generalization. The backpropagation is an example of supervised learning method. The concepts and terminology of supervised learning are explained later in this manuscript.

Unsupervised Learning

In this method, sometimes called self-organizing, there is no output reference for ANN and only input vectors are needed to train the network. The learning process is usually done based on local information and internal signals. During the training processes the network weights are adjusted so that similar inputs produce similar outputs. Kohenen and ART are among those networks which use this method. Recently, other methods, which are usually implemented when there is not much information available, have become popular. For example the "Reinforcement method" (Berenji, 1992) uses reward and punishment signals for getting the best results.

Since in the most simulation projects some inputs and outputs of systems are available, supervised ANNs, either feed-forward or recurrent, are usually recommended. For more information about the learning methods in linear networks please refer to (Baldi, 1995). Among the varieties of ANNs, only fully connected Multilayer Perceptron networks are used and explained in this thesis.

Multiple Layer Perceptrons

Multiple Layer Perceptrons (MLP), sometimes called multilayer perceptron, networks are feed-forward static ANNs. An example of MLP is shown in Figure 5.

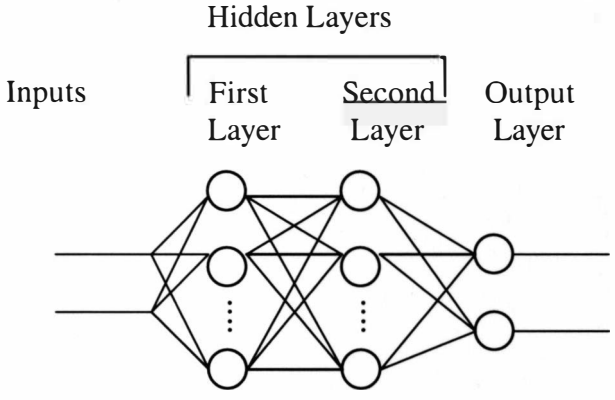


Figure 5. Three Layer Perceptron Neural Network.

In this kind of network, the input vector is applied to the first layer and the output of the first layer is connected to the second layer and so on and so forth. In the fully connected MLP, each neuron in layer l is connected to each neuron in the layer l+1. Figure 5 shows a three layer perceptron which has one input, one output and two

hidden layers. In some literature, the first two layers are called hidden because they are hidden from input and output. For consistency, it is suggested to name the ANNs based on their hidden layers rather than considering their inputs and output layers. For example, Figure 5 shows a two-hidden-layer network.

The Capabilities of Multiple Layer Perceptrons

Fully connected MLPs are able to perform these tasks:

1. Provide all Boolean logic functions. The two layer MLP is able to perform all logical functions (Hush, 1993).

2. Partition the sample space in classification problems. The MLP with one hidden layer is able to divide every convex region. For classification of both convex and concave regions, an MLP with two hidden layers is enough (Lippmann, 1987).

3. Perform all kinds of nonlinear mapping in functional approximation problems. The MLP is able to estimate and model every nonlinear mapping with the desired degree of precision in a R^n space (Hornik, 1989).

The Learning Algorithm for Multiple Layer Perceptron

For many years the most important problem of MLPs' learning was the adjustment of the weights of hidden layers in the network. Rumelhart (1986) introduced a method for solving this problem. This method, called backpropagation, is based on the steepest descent method. In the backpropagation, a sample of output error will be propagated in the entire network and it will be used as a reference for

adjusting the internal weights of network. Figure 6 shows an MLP network.

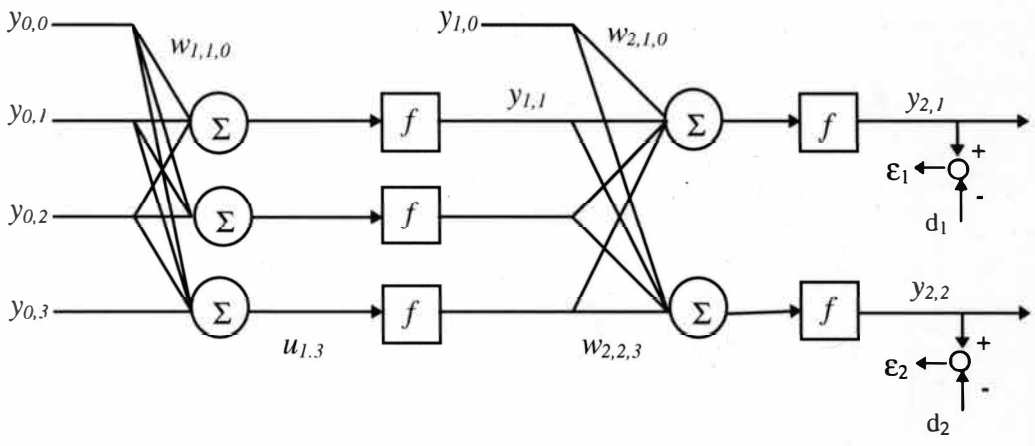


Figure 6. A Two-Layer MLP.

Where:

$u_{l,j}$ The weighted summation of jth neuron in the layer l.

$y_{l,j}$ The output of jth neuron in the layer l.

$w_{l,j,i.}$ The weight which connects the ith neuron of layer l-1 to jth neuron of layer l.

U_p The pth input pattern of the training set.

$d_j(U_p)$ The desired output of jth neuron for the pth input pattern.

$\varepsilon_j(U_p)$ The output error of jth neuron for the pth input pattern.

N_l The number of neurons in the layer l.

L The number of layers.

P The number of learning patterns.

For simplicity, this network is composed of only one hidden layer. The input layer is

called layer zero. For example, $y_{0,1}$ is a notation for the first input of the pth pattern.

Thresholds are considered as neurons number zero with input value of one. So, $w_{l,j,0}$ is the weight of lth layer, connected to jth unit of that layer, with the input of $y_{l,0}$ =1. In Figure 6, $f(x)$ is a continuous function and $f'(x)$ is its derivative with respect to x.

$$y_{l,j} = f(u_{l,j})$$

$$u_{l,j} = \sum_{i=0}^{N_{l-1}} w_{l,j,i}\, y_{l-1,i} \tag{5}$$

$$\varepsilon_q(U_p) = y_{L,q}(U_p) - d_q(U_p) \tag{6}$$

Equation (6) represents the error of qth output to the input pattern U_p. The total squared error to the input pattern U_p will be:

$$E_p = 1/2 \sum_{q=1}^{N_L} \varepsilon^2{}_q(U_p) \tag{7}$$

This error can be generalized over all input vectors in the sample space. This generates a global error function. This function, which is the total error of the network for all of the patterns, is called Sum-of-Squared-Error Criteria Function and is shown as:

$$J = \sum_{p=1}^{P} E_p \tag{8}$$

J is the goal function which is desired to be minimum. With the help of the gradient

method, weights of the network will be adjusted in such a way that fulfill this desire. To reach this goal, it is necessary to minimize the total error of the network to the input U_p in each iteration. This means that for a given input pattern to the network (U_p) the weights should be adjusted in a way that minimizes the error of the corresponding output.

From steepest descent method (Hush, 1993):

$$\Delta w_{l,j,i}(k) = w_{l,j,i}(k+1) - w_{l,j,i}(k) = -\eta \frac{\partial J}{\partial w} \Big|_{w=w(k)} \tag{9}$$

$$\Delta w_{l,j,i}(k) = -\eta \sum_{p=1}^{P} \frac{\partial E_p}{\partial w_{l,j,i}} \Big|_{w=w(k)} \tag{10}$$

In the above formulas, η is a positive constant which is called learning rate. Using the Chain Rule:

$$\begin{aligned} \frac{\partial E_p}{\partial w_{l,j,i}} &= \frac{\partial E_p}{\partial y_{l,j}} * \frac{\partial y_{l,j}}{\partial w_{l,j,i}} \\ \frac{\partial y_{l,j}}{\partial w_{l,j,i}} &= \frac{\partial y_{l,j}}{\partial u_{l,j}} * \frac{\partial u_{l,j}}{\partial w_{l,j,i}} \end{aligned} \tag{11}$$

$$\frac{\partial u_{l,j}}{\partial w_{l,j,i}} = \frac{\partial}{\partial w_{l,j,i}} \sum_{i=1}^{N_{l-1}} w_{l,j,i}\, y_{l-1,i} = y_{l-1,i} \tag{12}$$

From (5) and (11) it can be concluded that:

$$\frac{\partial y_{l,j}}{\partial w_{l,j,i}} = f'(u_{l,j})\, y_{l-1,i} \tag{13}$$

$$\frac{\partial E_p}{\partial w_{l,j,i}} = \frac{\partial E_p}{\partial y_{l,j}} . f'(u_{l,j}) y_{l-1,i} \tag{14}$$

The term $\frac{\partial E_p}{\partial y_{l,j}}$ expresses the sensitivity of E_p to the output of the node $y_{l,j}$ .This node affects the E_p through the nodes which belongs to higher layers. Therefore, it can be expressed as a function of the sensitivities of the higher layers' neurons.

$$\frac{\partial E_p}{\partial y_{l,j}} = \sum_{m=1}^{N_{l+1}} \frac{\partial E_p}{\partial y_{l+1,m}} . \frac{\partial y_{l+1,m}}{\partial y_{l,j}} \tag{15}$$

$$\frac{\partial y_{l+l,m}}{\partial y_{l,j}} = \frac{\partial y_{l+l,m}}{\partial u_{l+1,m}} . \frac{\partial u_{l+l,m}}{\partial y_{l,j}} \tag{16}$$

$$\frac{\partial u_{l+l,m}}{\partial y_{l,j}} = \frac{\partial}{\partial y_{l,j}} \sum_{q=0}^{N_l} w_{l+1,m,q} \, y_{l,q} = w_{l+1,m,j} \tag{17}$$

$$\frac{\partial y_{l+l,m}}{\partial y_{l,j}} = f'(u_{l+1,m}) . w_{l+1,m,j} \tag{18}$$

$$\frac{\partial E_p}{\partial y_{l,j}} = \sum_{m=1}^{N_{l+1}} \frac{\partial E_p}{\partial y_{l+1,m}} . f'(u_{l+1,m}) . w_{l+1,m,j} \tag{19}$$

So, the sensitivity $\frac{\partial E_p}{\partial y_{l,\mathrm{m}}}$ can be expressed based on the next layer's sensitivity, $\frac{\partial E_p}{\partial y_{l+1,\mathrm{m}}}$. This process can be continued up to the last layer which is the output layer.

In this layer the boundary conditions exist. From Equations (6) and (7):

$$\frac{\partial E_p}{\partial y_{L,j}} = \frac{\partial}{\partial y_{L,j}} \cdot \frac{1}{2} \sum_{q=1}^{N_L} \varepsilon^2{}_q(U_p) =$$

$$\sum_{q=1}^{N_L} \varepsilon_q(U_p) \cdot \frac{\partial \varepsilon_q(U_p)}{\partial y_{L,\mathrm{j}}} = \varepsilon_j(U_p) = y_{L,j}(U_p) - d_j(U_p) \qquad (20)$$

The expression in Equation (20) is called the output error, and the corresponding expression for hidden layer nodes in Equation (19) is referred to as the hidden layer error. Since the hidden layer error is calculated from the output error backwards, it is called "backpropagation error" and the algorithm is known as "Backpropagation Algorithm". By the help of this algorithm, all of the network's weights will be adjusted.

The learning parameter, η, is usually a constant for the whole network and defined in the open interval (0, 1). There are several other methods for setting η. Some of them suggest different values of η for each layer. The other methods prefer big values of η at the beginning of the learning process, continuing with small values as the learning process goes on. Choosing different values of η for each neuron is another option. Basically, selecting the right value for η is not easy and usually is done based on trial and error. Speed and performance of MLP radically depends on the values of η. In some circumstances, wrong values of η make the network's output unstable and divergent. One of the approaches for solving this problem is to make the η adaptive. It has even been suggested to make η adaptive with fuzzy logic sets. The simplest and most popular approach is to add a "Momentum" term to each weight update.

$$\alpha . \Delta w(k) = \alpha . (w(k) - w(k - 1)) \qquad 0 \prec \alpha \prec 1 \qquad (21)$$

In this formula, α is the coefficient of momentum. Adding (21) to (9) makes the variation of weights smoother.

Generalization

After the learning phase, the performance of the network is usually criticized based on its generalization power. Generalization is the network's ability to produce accurate results on new samples which do not belong to the training set. The Generalization depends on these factors: (a) the number of training points in the training phase, (b) the sequence and the nature of training data set, (c) the complexity of the system which is under consideration, and (d) the structure and size of ANN.

Limitations of MLPs

1. Currently, there exist no deterministic or heuristic method for choosing the best structure and optimum number of neurons and layers. If the network's size is small, the network will lose its ability to approximate a good model of the system. If the network's size is big, the number of local minima will increase and the speed of the network will rapidly decrease.

2. There is no evidence that the network will be able to learn the mapping function. Although the number of input sets might be very high, there is no guarantee that the weights will reach unique numbers in a reasonable amount of time.

3. The local minima problem in the perceptron networks has not been solved yet. This problem is due to the gradient method which intrinsically will stop and stay in the local minima. The method of choosing the nonlinear forms of elements, in the local minima problem, is very important. The more linear the activation function, the less the number of local minima. However, smooth nonlinearity is required by backpropagation technique. Figures 7, 8 and 9 show the Mean Square Error (MSE) surfaces of $E[(\varepsilon)^2]$ as function of two weight values in a one layer perceptron based on different activation functions (Widrow, 1990).

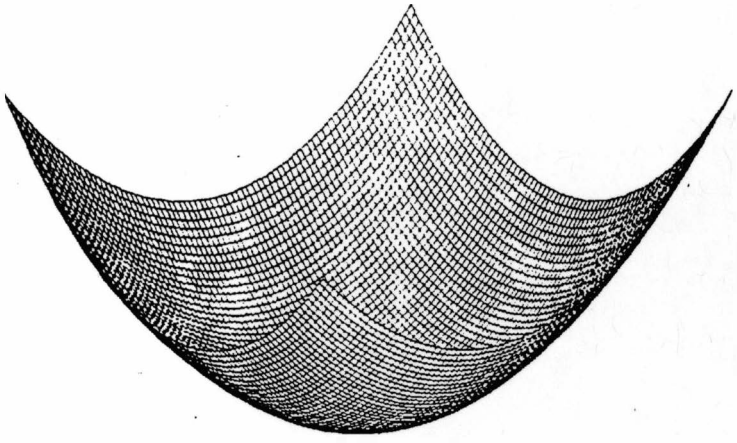


Figure 7. MSE Surface of Linear Error.

In Figure 7, the activation function is a linear function. In this case, the global minimum is accessible through the gradient method. Figure 8 shows the hyperbolic tangent activation function. This function is a nonlinear function but differentiable. In this function, selecting the right values for the gradient steps is very important and gaining the global minimum is possible, but of course not as easy as in the previous

one. In Figure 9, a threshold function has been considered.

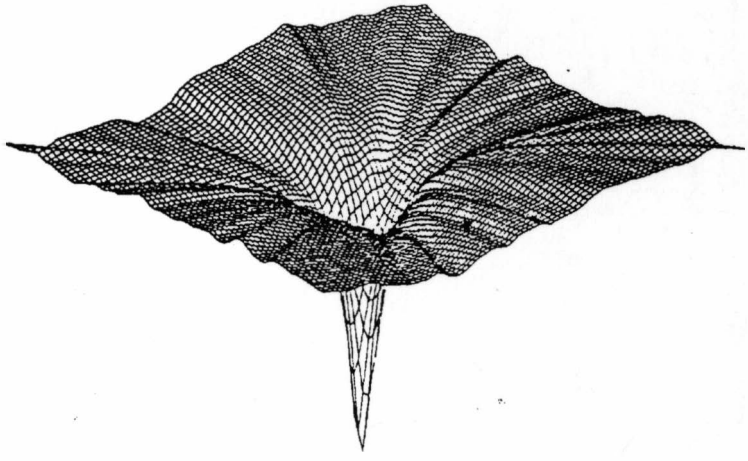


Figure 8. MSE Surface of Sigmoid Error.

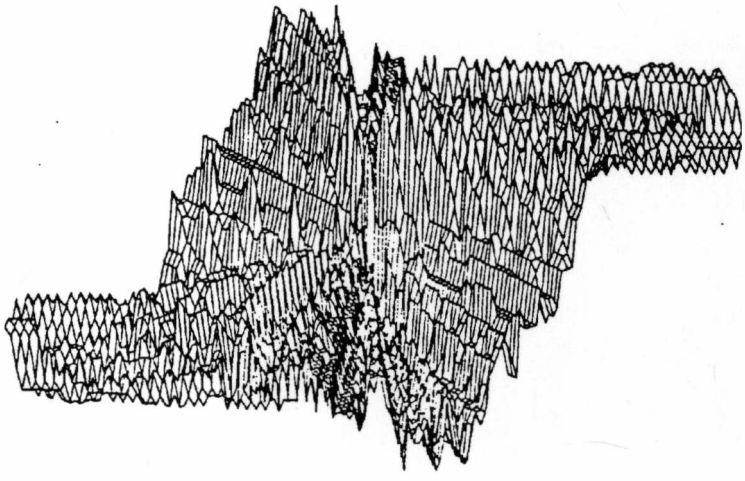


Figure 9. MSE Surface of Signum Error.

This function is not differentiable and nonlinear. As shown, there are many local minima and gaining the global minimum is almost impossible. Many methods,

such as Boltzman Machine and Simulating Annealing, have been suggested for solving this issue.

4. The backpropagation method is slow. this method usually takes time in the perceptron networks because of local minima, network size and the small initialization values. Increasing the values of η, is among the suggested methods for overcoming this limit. Although, it may make the network unstable.

Some guidelines on building good structures of MLPs will be suggested in Chapter V. Setting the parameters and variables will also be discussed.

CHAPTER IV

LITERATURE SURVEY

Introduction

The first simulation systems were mechanical and performed mathematical operations using combinations of gears and machines. In the late 1970's the microprocessor became a reality and greatly enhanced the role of simulation, permitting it to evolve from being a physical tool to acting as a mean for performing numerical analysis. Now, simulation methods have the ability to manipulate large amounts of data, perform mathematical calculations and predict the expected performance of a real system. Concurrent with the increased capability and flexibility of the simulation was the expansion to many continuous and stochastic processes including material handling systems, food-processing operations, health care systems, etc.

But simulation methods are time consuming and expensive in terms of computer time. Experiments must be repeated in full if new conditions require re-evaluation. When this is combined with the number of scenarios that the decision maker has in mind, make the total number of runs prohibitively high, rendering the simulation unattractive. Researchers have addressed this traditional problem as "computationally expensive" (Flood, 1995), "expensive in terms of processing time

and/or money requirements" (Kilmer, 1993), "resulting in computing costs" (Pierreval, 1992), etc.

In an attempt to solve this problem, a few researchers tried to change the simulation methodology to the method that human beings use for interpreting the systems in order to make the simulation "intelligent". Artificial Neural Networks (ANNs) is among the techniques that have been used to improve the performance of the simulation of manufacturing systems.

In this chapter, first the general applications of ANNs are briefly reviewed. Two areas of applications of ANNs in industrial engineering are explained. Then, some of the literature on application of ANNs in simulation are classified and discussed. Similar attempts in other fields and the articles which may be useful in some aspects of simulation are also reviewed.

Applications of ANNs

Although there is no reported practical application of ANNs in industries up to the late 1980's, the number of applications in industry, business and science in the mid 1990's is amazingly high (Widrow, 1994). ANNs are not able to compete with conventional techniques at performing numerical operations. However, they are useful for tasks involving ambiguity such as handwritten character recognition and speech recognition. The capabilities of ANN in pattern classification, prediction, control and optimization have been demonstrated by scientists. Most of these studies are based on articles published by Hopfield (1984), Rumelhart (1986) and Lippmann (1987).

Application of ANNs in Industrial Engineering

In the domain of industrial engineering, the application of ANNs in quality control, optimization and resource allocation have been reported by Widrow (1994).

One area of using ANN in industrial engineering is in Grouping Parts. Grouping Parts allows design and manufacturing to take advantage of geometric shape similarities between the parts. Chung and Kusiak (1994) use the recognition of objects for identification, classification, verification and inspection tasks in manufacturing. They have developed a feed forward - back propagation ANN to classify the parts based on their geometry.

Another area is job shop scheduling. Job shop scheduling is a resource allocation problem. The resources are called machines and basic tasks are called jobs. Foo, Takefuji and Szu (1995) investigate the applicability of ANN for solving job shop scheduling problems. They have used a hardware consisting of linear and nonlinear processors in their survey. Sim, Yeo and Lee (1994) try to apply an expert neural network to the dynamic job shop scheduling problem. The authors believe that the major disadvantage of the ANN, as compared to the knowledge based system, is its inability to explain the factors and decisions made in arriving at the solution. The expert neural networks system integrate the expert system and neural network to take advantages of both methods. They propose a network consisting of several sub parallel nets. The expert system activates sub networks according to the prevailing job shop environment. They have found that ANN can be used to meet the changeable demands

on production scheduling.

The job shop scheduling problem can be modeled by mixed integer-linear programming. Thus, the proposed approaches for solving this problem open up the application of ANNs in Operations Research and Linear programming problems.

Application of ANNs in Simulation

In the field of simulation, most researchers have applied ANN to simulate a process, a machine or a function without writing much about the procedure of building their network. In these cases, the readers find the ANN a useful tool in simulating a particular process but not applicable to the other problems, even to similar ones. On the other hand, there are a few articles which systematically approach how to build ANNs' models.

In this section, first the literature on the application of ANNs in simulation is surveyed. Some of this literature is about a particular application. The others talk about the general idea of application of ANNs in simulation. Then, the articles which have different approaches to the problem are reviewed. The similar attempts in using ANN in fields other than manufacturing engineering are the subject of the rest of this chapter.

Wildberger (1989) has made one of the first attempts in this field. He has studied the use of neural networks in enhancing the performance of a power plant. He has discussed the possibility of replacing an artificial intelligent system with neural network. The system assists operators and performance engineers in improving plant

efficiency in real-time (1992).

In a systematic attempt to apply ANN to simulation methodology, Fishwick (1989) compares the ANNs' performances with traditional methods such as linear regression and surface response. He has developed a neural network for simulating a ballistics model and, based on the results of his experiment, the neural net model appears to be inadequate in most respects. The result of ANN in comparison to the other two methods is so poor that he comes to the answer "NO" in reply to the question "Are neural network models useful as simulation models?". But, later literature shows more promising prospects for applications of ANNs in simulation.

Lampinen and Taipale (1994) present a neural network based system for estimating the final quality of paper from process measurements. They have realized that the final quality of paper depends on many process variables. Furthermore, it is very difficult to find theoretical rules of the behavior of paper properties when variables depend on each other. To solve these problems, they have suggested an MLP network to simulate and optimize the paper manufacturing process.

Sarne and Postorino (1994) use a supervised backpropagation ANN for simulating at each instant the values of traffic flows in a real transportation network. They have realized that the ANN can resolve both random aspects and the presence of a cyclic dependence among the variables of the problem.

Padgett and Roppel (1992) offer a systematic approach in prospective application of ANN to simulation. Based on their article, rapid computation, robustness and adaptability of ANNs are the main factors that make ANNs a good

candidate for simulation. They have pointed out that ANNs require fewer assumptions and less precise information about the system modeled than do some of the more traditional techniques. The authors suggest a design technique for a neural network simulation model and briefly explain how each step of this design may be implemented. Their suggested procedure has four steps: (1) definition of global system goals, (2) algorithm selection and design, (3) implementation and constraints, and (4) evaluation and performance measures. The necessity of examining the ways to integrate neural network with fuzzy logic, genetic algorithms, expert systems and other tools for knowledge based systems designers, has been addressed in their article.

Researchers have different approaches to the application of ANNs in simulation. Sometimes, they look at the same problem from different perspectives. Furthermore, ANNs have been used in a variety of fields in different applications. Some of these approaches can be applied to simulation of manufacturing systems as well. Thus, a classification of these approaches is presented here. Specifically, these categories are reviewed: (a) ANNs as map operators, (b) ANNs and metamodeling, (c) ANNs as a part of simulation software, (d) ANNs in statistics, and (e) ANNs in consultant projects.

ANNs as Map Operators

One approach to simulation methodology is to consider computer simulation as a map operator which maps a set of inputs to a set of outputs. In this approach, the input vector will be mapped to the output vector through a function which will be

provided by an ANN. ANNs have been known as a promising method of mapping vectors (Wasserman, 1993).

In an attempt to compare ANNs and multiple linear regression as two methods of mapping functions, Kilmer and Smith (1993) conduct an experiment of approximating a lot size-reorder point inventory system simulation for estimating mean total cost. The authors compare the ANN with the type of multiple linear regression typically used in the response surface method. In their experiment, the authors first produce the output with a traditional simulation language and assume that these outputs are perfect. Then, they try to compare the result of the ANN with the regression method. They conclude that regardless of training on mean data or on individual replication, the ANN outperform the corresponding regression models.

The authors conduct another experiment for calculating mean and variance of the total cost of a similar case (1994). They have labeled their approach as a metamodeling technique for discrete event stochastic simulation. This article will be reviewed next.

ANNs and Metamodeling

One of the approaches for applying ANNs to simulation is to simplify the real system and reduce the number of inputs and outputs of the model, without loss of generality. In this method, researchers try to simplify the real systems into smaller models in which only a selected subset of input variables will be considered. This method, called metamodeling, was first proposed by Blanning (1975) and later was

extensively used by many other researchers (Kleijnen, 1992), (Friedman, 1989). The application of this method as a post-simulation analysis tool has been discovered by researchers (Friedman, 1988). A review of published papers on the application of metamodels in manufacturing from 1975-1993 can be found in Yu (1994). Most of the researchers, especially those who want to predict the output of the real system based on the output of ANN, have used the metamodeling technique. These researchers have tried to replace the metamodels with an ANN.

One of the best attempts in this regard has been done by Pierreval and Huntsinger (1992). According to them, the advantages of using metamodels include: (a) reduction of computing costs (memory/time) in comparison to traditional software applications like SLAM II, SIMAN, GPSS; (b) performing sensitivity analysis; (c) model simplification; (d) enhanced exploration and interpretation of the model; (e) facilitating the transfer of models; (f) optimization; (g) answering inverse questions; and (h) reducing the number of inputs.

The authors have successfully implemented this method to a job shop metamodel as an example of discrete simulation. They compare a traditional simulation model with an ANN metamodel based on elapsed time and occupied memory of a computer. In another experiment the authors have used "The percolator coffeepot metamodel" as an example of the continuous system simulation. This system was previously modeled with partial differential equations and other mathematical techniques. Again, an ANN is used for simulating this system. The results in both cases show a significant reduction of use of computer time and memory.

Another attempt in this area has been made by Kilmer and Smith (1993). They apply neural networks as metamodels for discrete-event stochastic simulation. A classical (s,S) inventory simulation, taken from the experimental design and optimization chapter of Law and Kelton (1991), is translated to a metamodel through the development of parallel neural networks, one estimating total cost and one estimating variance of total cost. Kilmer and Smith show that the neural network metamodel is quite competitive in accuracy to the simulation itself, and is computationally more efficient.

ANN as a Part of Simulation Software Applications

Hurrion (1992) describes the use of neural networks to represent the results of simulation of a coal depot operations. The author has applied all results obtained from the simulation to a neural network. After a suitable period of training the quality of results obtained from the network is matched to the corresponding results of those obtained by running the original simulation model. The author concludes that the ANN, after a suitable period of training, would be able to predict the response of different model configurations without the need to re-run the simulation. Hurrion (1992) believes that "it is possible, with the help of ANN, to suggest the next best configuration to investigate. If this next suggestion is then simulated, then its results can be added back to the training set, improving the reliability of ANN model". Because the results obtained from ANN model are fast, it is possible to draw contour maps in real time. The author also brings up the idea of developing hybrid models

which are part simulation and part neural.

ANNs in Statistics

Today, simulation and statistics are convoluted. So any enhancement in statistics may be applied to simulation methodology. Most accredited simulator software applications have some parts which are directly concerned with statistics methods. Examples are random number generators on a given distribution function, fitting the best curve to a set of data, etc. There are many prospective applications of ANN in statistics. Realizing that the neural networks are not an amateurish tool in statistics, Hornik (1994) expresses the advantages of using ANN in this field. He points out that, in the long run, neural network modeling and special-purpose hardware realizations should become a standard tool in applied statistics. This section has been devoted to the application of ANN in statistics, especially the areas that are common with simulation.

It can be shown that MLPs can approximate any reasonable function arbitrarily well, provided that enough hidden units are available for internal computations and that their activation function be non polynomial (Hornik, 1989). Kopsco and Pipino (1993) investigate the applications of neural networks to the tasks of learning functional mapping and interpolation. They compare the performance of the neural networks in the interpolation to that of interpolating polynomials. The results show that neural network can be useful in learning functional mapping and interpolation. The authors suggest that neural network models can be added to conventional statistical

tools to aid in the recognition of underlying functional forms. The easiest way to interpolate the data is "by eye". But besides the lack of rigor, a main drawback of the "by eye" procedure is its inability to generalize beyond three dimensions. So, the authors recommend the ANN method when the data sets are higher dimensions than two. The authors have applied the ANN to Ancombe's (1973) data sets and compare the results with that of traditional regression method. They have found ANNs as tools which are able to model a phenomenon without explicit knowledge of its underlying function form. Finally, they recommend the ANN as the best method for finding the best fitted function to a set of data. Hashem and Schmeiser (1993) show that using MSE-optimal linear combinations of a set of trained feed-forward networks may significantly improve the accuracy of approximating a function and its first and second order derivatives.

The application of ANN in statistics is not only in the approximation of functions and their derivatives. Every user of simulation software applications, to some extent, is involved in statistical distribution of variables. One of the basic problems for model makers is to find the best fit of distribution for a set of data and to generate some random numbers which belong to that distribution. In most practical applications the distribution functions which exist in a software database are not suitable for representing the set of data. Inevitably, the model maker should choose the best functions which exist in the software database for representing the distribution of that data. Undoubtedly, this approach will increase the error.

Hurrion (1993) describes a method by which a neural network learns to fit a

distribution to a sample data. The author has found that the ANN can be an alternative approach to the problem of selecting suitable distributions and random variate generation techniques for use in simulation and mathematical models. The author has been able to fit a neural network to many continuous input distributions such as normal, uniform, negative exponential, gamma and beta. Several kinds of data such as one tail, two tails and range over fixed intervals are studied in the experiment. The statistical goodness-of-fitness shows no significant difference between the distributions learned by neural network and original theoretical distributions which were used to train them. They show that if it is not possible to fit a theoretical distribution to the sample data then this method is an alternative to sampling directly from the empirical histogram obtained from the sample data.

ANNs in Construction Projects

One of the applications of ANN is in simulation of construction engineering projects. Since there are many similarities between construction projects and manufacturing projects, the review of the literature in this field seems useful. Flood has been trying to find suitable ANNs to model construction processes. His works are mostly based on Radial-Gaussian neural networks (Flood, 1991). He has addressed the advantages of using Radial-Gaussian as: (a) they are fast, (b) guaranteed convergence on a solution during training, (c) an ability to model functions to a high precision, and (d) automatic determination of the number of neurons to include in a network.

Flood and Worley (1994 and 1995) use the rapid execution of simulation

through parallel processing in two experiments. The first experiment concerns a model of a simple chaotic function. The results prove that complicated behavior in recursive functions can be captured by using ANN. The second study involves modeling a non continuous scraper-based earth-moving system that, traditionally, had been modeled using discrete-event simulation algorithms. Both studies indicate the viability of the neural network approach in simulation.

Flood and Christophilos (1996) reevaluate a neural network approach to modeling the dynamics of construction processes that exhibit both discrete and stochastic behavior. The application of the technique to two classes of earth-moving systems is reassessed in the article. The results confirm the ability of the neural network to model the discrete and stochastic behavior of some classes of construction processes. They have realized the potential application of the method in two situations, one where there is limited theory describing the dependence between the variables and the second where there is a need for rapid execution.

CHAPTER V

DEVELOPING ANN MODELS

Introduction

Most of the available publications on ANNs emphasize the advantages of one method and ignore its limitations. Some of them are about the application of ANN to a particular domain. A few discuss applications of ANNs to manufacturing systems. Having read these publications, the modeler may have the impression that ANNs are useful tools for solving many real-world problems. However most modelers, especially those who are not familiar with ANNs' techniques, are not able to adapt this new technique to their own case. There are few guidelines to help them simulate their systems with ANNs.

Modelers face many questions. Some of them are as follows: "What kind of systems can be modeled by ANNs?", "How should the data be gathered and prepared?", "When should the learning process be stopped?", "What are the best values for network parameters?", and in brief , " How can a system be modeled by ANNs ?".

Most of the above questions are still open questions and there is no absolute answer for them. However, this chapter tries to give direction to answer the above questions. This information has been drawn from many sources and from the some

experimentation. Some of the proposed answers are based on proven theorems, the others are based on the rule of thumb. In the latter case, the modeler's judgment and ingenuity along with trial and error are very important. The goal of this chapter is to help modelers develop successful models with ANNs. Toward this goal, the simulation life cycle through ANN is reviewed. Each step is discussed and some suggestions are offered in each step.

Simulation Life Cycle Through ANNs

Developing a successful model through ANN contains these steps: (a) problem selection and formulation, (b) selection of appropriate ANN and simulation software, (c) data acquisition and preparation, (d) model translation (network building, network training), (e) testing the model, (f) experimentation, (g) analysis of results and denormalization, (h) implementation, and (I) documentation.

A schematic of the simulation life cycle through ANN is proposed in Figure 10. It is seen that the development of a successful model is more than just coding and training a network. Special attention should be paid to activities such as data gathering, normalization, building and training the network, testing, analyzing the results, etc.

Problem Selection and Formulation

The first step of the simulation life cycle through ANN involves selecting an appropriate problem and stating the problem clearly.

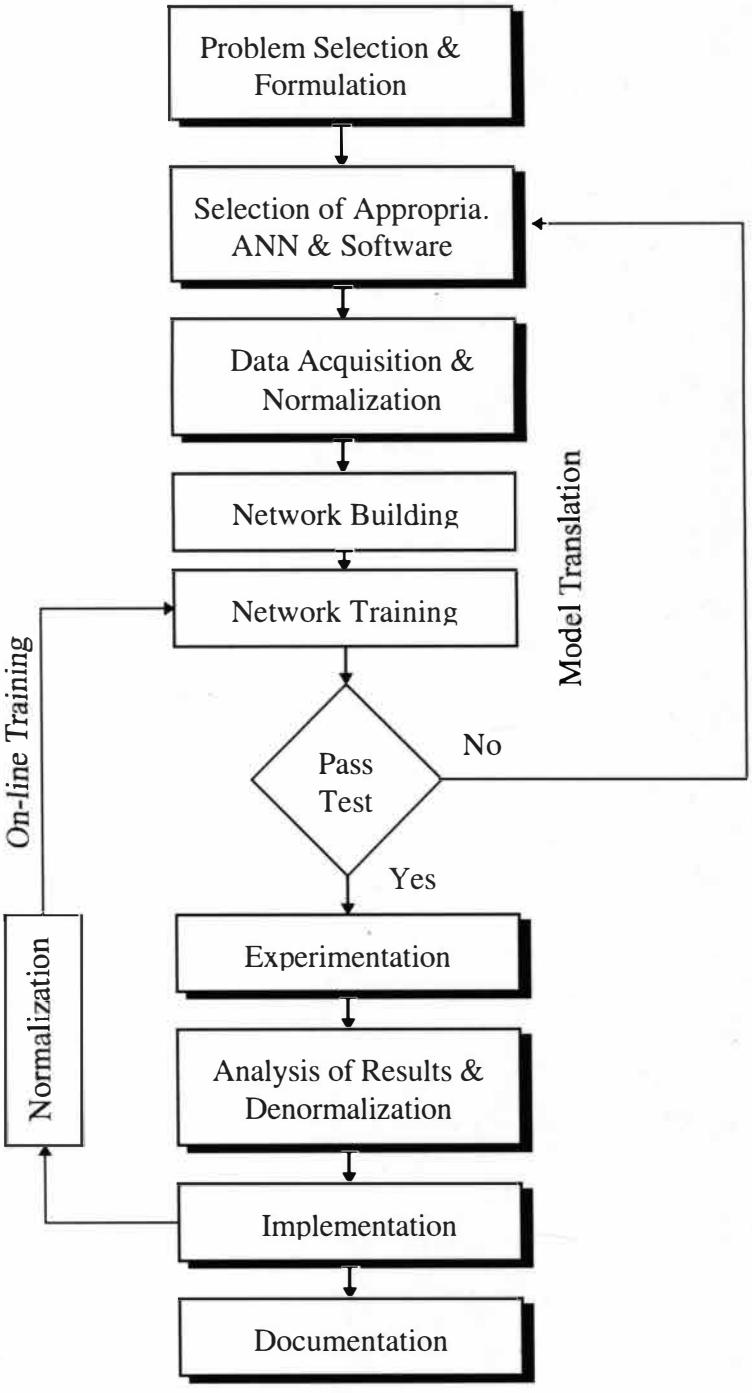


Figure 10. Simulation Life Cycle Through Artificial Neural Network.

Problem Selection

One of the first questions the modeler may ask is if the ANN is a good solution to a given simulation problem. ANNs' applications and capabilities are discussed in Chapters III and IV. However, there are still some points that the modeler should consider. For a given problem, if there is a mathematical solution, ANN is not recommended in most cases. ANNs usually require more computations and produce less accurate results than closed-form mathematical methods.

The power of ANNs are best shown when there is some ambiguity in the system. Ambiguity appears when the complexity or inaccessibility of the system makes the mapping -- relation between input vector and output vector -- unknown. ANNs have the best performance in the situations where a human's intelligence performs better than a computer. Examples in manufacturing systems include, but are not limited to the systems where the characteristics of machines are not well known or in the cases that the machines are not accessible.

Problem Formulation

Having selected the appropriate problem, the modeler should define the problem and objectives clearly. The modeler should then go through these steps: (a) define the data, and input and output vectors; (b) define the relation between the real systems input/outputs and ANN input/outputs; (c) specify the criteria for comparing alternative designs; and (d) study the project in terms of manpower, time and cost.

Selection of Appropriate ANN and Software

Having decided that ANN is a good approach for modeling a well-defined problem, the modeler should choose an appropriate ANN method and a good software for performing the experimentation. In this section, the criteria of choosing ANNs' methods and software environments are discussed.

ANN Methods

A question which usually arises is "What is the best ANN method for modeling the system?". The answer to this question depends on the data and the system to be modeled. According to Wasserman (1993), ANNs can be selected :

1. Based on static or dynamic behavior of the system. Static systems map a given input vector consistently to the same output vector. Non-recursive (feed-forward) networks are recommended for modeling static systems. On the other hand, dynamic systems' outputs depend on the current and previous input/output and on the state of the system. For dynamic systems, recursive ANNs are usually recommended. Most real-world problems have some dynamic characteristics. Thus, the recursive ANNs are good candidates for most practical problems. However, because of the complicated and lengthy training phase, recursive ANNs have found limited practical applications.

2. Based on the continuous or classified output of the system. ANNs can be categorized by those which map their input vectors to continuous valued outputs and

those which perform classification. The classifier ANNs can not perform continuous mapping, but the continuous ANNs can classify.

3. Based on the availability of input-output vector pairs. There are many ANNs that require supervised training and there are some that require unsupervised (self-organizing) training. For supervised learning the input-output vectors should be available. Unsupervised learning works based on the local information and internal signals. The input and output of the manufacturing systems are usually available. So, the supervised training ANNs are recommended in most of the cases. Furthermore, the theories of unsupervised training are still under development. Unsupervised ANNs have not proved their capabilities for solving practical problems yet. In this manuscript, first static systems are discussed and then suggestions for modifying the static model to simulate stochastic and dynamic systems are offered.

Not only in the field of simulation but also in many other fields, most of the researchers have used backpropagation for solving their practical problems. Backpropagation is usually used to perform supervised training on multilayer, non-recursive networks. With suitable training, backpropagation can be used for either continuous mapping or classification. Backpropagation's training algorithms for recursive networks have also been developed (Narendra, 1991). Furthermore, the backpropagation-through-time has been successfully applied to dynamic systems, e.g., chemical process industries (Werbos, 1992).

As discussed in Chapter III, backpropagation suffers from some limitations. To overcome the limitations of backpropagation, Flood and Christophilos (1996) use the

Radial-Gaussian method for simulation. Although there are many advantages of using Radial-Gaussian, after training it is generally slower to use, requiring more computation to perform a classification or functional approximation (Wasserman, 1993). In this method, the required number of hidden units increases geometrically with the number of inputs, so it becomes impractical for problems with many independent variables. Hence, more investigations are required to make the Radial-Guassian an appropriate method for interactive simulation.

From now on, only Multilayer Perceptron (MLP) networks with the backpropagation learning method, are considered. While MLP is not the only neural network that can be used in simulation, they are the most popular and clearly illustrate the major features of neural network approaches to simulation. This decision has been made based on the capabilities of MLPs (see Chapter III).

Selection of an ANN Software

The modeler can use either a general purpose computer language or a neural network simulation environment. The modeler can use high level languages such as C, PASCAL and BASIC. By using these languages, the modeler has control to get any information which may be needed. However, developing a well structured general purpose ANN simulation environment is time-consuming and requires thorough knowledge of computer systems.

On the other hand, the modeler can choose one of the prepared packages for ANNs simulation. As the field of neuroscience matures, new simulators are being

introduced. The available simulators can be classified as either supporting biological models of neurons or as being tailored toward the artificial neural network. According to Skrzypek (1994) all neural simulation tools can be classified into four categories:

1. The programs which are not documented and are dedicated to solve particular problems. These tools can not support a variety of applications.

2. Custom made software programs which are usually borrowed from other application domains and organized into libraries.

3. Sophisticated programs that integrate advanced graphical user interfaces (GUI) and analysis tools. These programs are usually dedicated to a particular class of architecture / algorithm.

4. Advanced simulation tools which are complete, system-level environments. These tools can support a wide range of neural networks and a variety of learning methods. These environments have graphical user interfaces and many tools for analysis, manipulation and visualization. Examples of this group are: SNNS (Stuttgart Neural Network Simulator) and SFINX (Structure and Function In Neural ConneXtions).

A good ANN modeling environment should help the modeler to build a model and to test the synthesized model computationally. The environment should also ease the problem of tracking all experimental data that has not yet been tested by the current model. A good GUI should be provided to make the simulator user friendly. GUI allows the modeler to interact with the simulator and to visualize data generated by the model.

In the domain of simulation of manufacturing systems, modelers are mainly interested in the neural network simulation environments rather than biological model environments. The former concerns network aspects, but the latter involves single neuron behavior. Interested readers can refer to Skrzypek (1994) for further information about the neural network simulation environments.

The Stuttgart Neural Network Simulator (SNNS) environment has been selected for implementing the experiments in this research. SNNS has been developed at the Institute for Parallel and Distributed High Performance Systems at the University of Stuttgart since 1989. SNNS is a tool for synthesizing, training, testing and visualizing artificial neural networks. Many different ANNs' methods and varieties of training algorithms are supported by this software. Its X graphical user interface (XGUI) gives a graphical representation of the neural networks and controls the kernel during the simulation run. XGUI is tailored for inexperienced users and helps them to create, manipulate and visualize neural nets. SNNS has been implemented in ANSI-C and the source codes of the programs are available to be modified. The availability of source codes makes it a unique tool for research purposes. It is a portable program which has been tested on numerous machines and operating systems. Complex networks can be created quickly and easily by this software. As is shown in Figure 11, the SNNS helps the users visualize, understand and possibly control the networks.

The reader should keep in mind that the ANN's hardware implementation radically increases the speed of the processes. Therefore, the ANN's hardware will become popular in the long run. These types of hardware are good candidates for

special purpose applications.

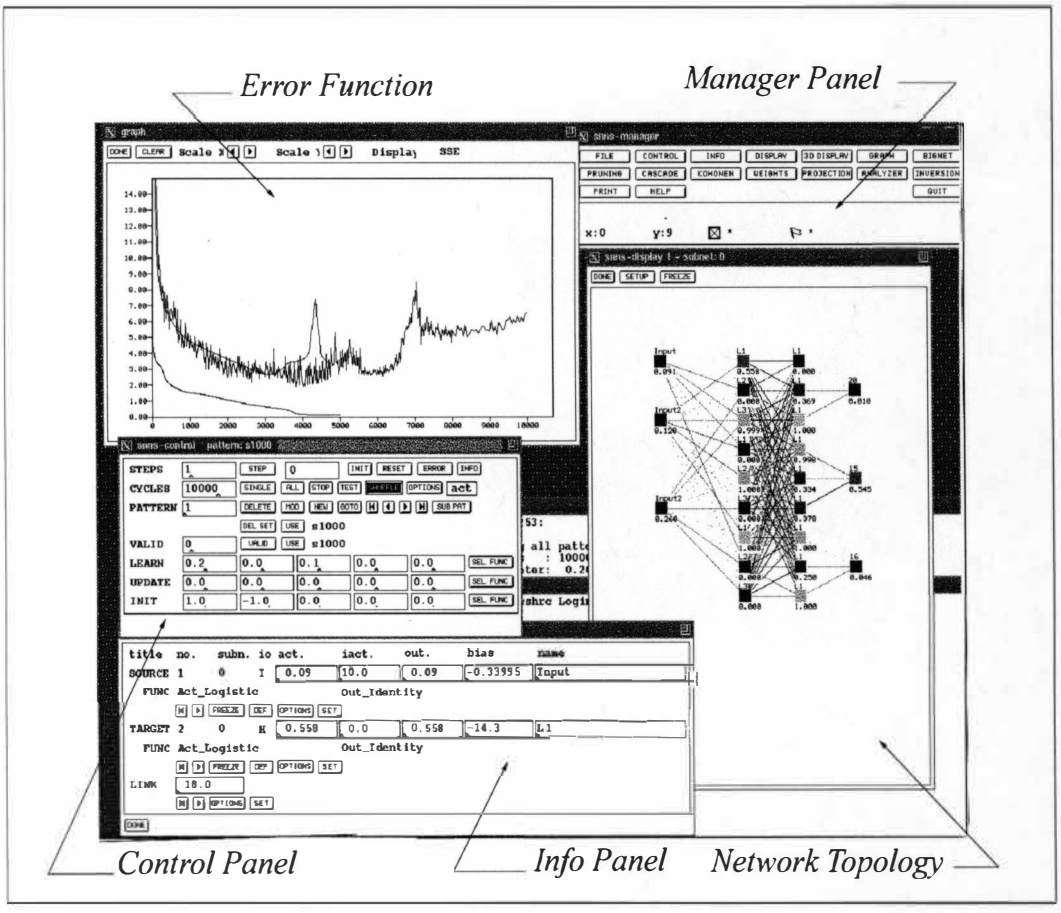


Figure 11. A Typical Screen of SNNS.

Data Acquisition and Preparation

The data should be collected on input vectors and corresponding output vectors. The modeler should split the data into at least two parts: one set on which training is performed, called the training data, and another part on which the performance of the network is measured, called the test set. The idea is that the

performance of a network on the test set estimates its performance in real use. In other words, absolutely no information about the test set should be available during the training process. In most cases the training data is also subdivided into a training set and a validation set. The validation set is used as a pseudo-test in order to evaluate the quality of a network during training. This method, called cross-validation, will be explained in this chapter.

The training set is used to train the network and the validation set helps the modeler to choose the best size of the network or to stop the training. The test set is used to determine the accuracy of the performance of the network. There is no formula for the ratio of training, verification and test set. However, as a rule of thumb, 40%, 30% and 30% are fair estimates of the ratio of training set, verification set, and test set, respectively (Smith, 1996).

The training set must provide an accurate representation of the problem domain; otherwise the performance of network in the real data might not be satisfactory. Special attention should be paid to the boundaries and the domain in which the ANN is supposed to work. The output of an ANN which is trained based on ill-defined training sets might be misleading. Experience and creativity are important to select a good training set.

Having gathered the data, the modeler should normalize them. The neural network outputs can only range from 0 to 1 (in some cases from -1 to 1). Although there is no limitation for input values, it is suggested to normalize inputs to the linear interval of activity functions of neurons. It is suggested to reserve some margins in the

normalized domain (e.g. normalize the values within .05 to .95). In this case, if new input or output values are above or below the values used to train or test the neural network model, the ANN can continue its performance.

The modeler can use linear or nonlinear functions for normalization. The modeler should have a detailed knowledge of the nature of the data to select an appropriate normalizing function (Wasserman, 1993). Using an appropriate function has a great affect on network's performance. In all of the experiments in this research, linear transformation has been used. The linear transformation is presented by Equation (22).

$$D_{new} = L_{min} + \frac{D_{old} - D_{min}}{D_{max} - D_{min}} .(L_{max} - L_{min}) \qquad (22)$$

D_{new} The normalized data

D_{old} The non-normalized data

D_{max} The maximum of the non-normalized data set

D_{min} The minimum of the non-normalized data set

L_{max} The maximum limit of normalized data

L_{min} The minimum limit of normalized data

The modeler should note that the sequence in which the data is fed to network may be important. It is recommended to feed the data samples randomly. In this research, all of the data has been "shuffled" before feeding to networks.

The generalization power of ANNs is the critique for measuring the accuracy of the network. A larger number of data samples usually do a better job for modeling the system. However, the number of data samples can not be increased infinitely, even if the data is available. There is a relation between the size of network and the number of training data points. Gathering the data is usually expensive and time consuming. Therefore, modelers are usually looking for the best size of the network to give the best generalization of a given set of data.

It is suggested that the number of training samples should be approximately ten times the number of weights (Hush, 1993). Thus, increasing the network's size not only requires more time for learning but also might give a poor result of generalization. So, it is desirable to find methods for reducing the network size, while at the same time retaining the capability of solving the problem.

Model Translation

Model translation is the process of preparing the model for computer processing. Two major features of model translation are building the network and training the network. Building the network is the process of synthesizing the appropriate network. Choosing the right network size and setting the network's parameters appropriately, play important roles in building a network. Training the network involves choosing the best values for weights in such a way that maximizes the network's generalization power. In the training phase of a network, the initialization of the weights and stopping criteria of training are very important.

Building a Network

Building a network is the process of defining layers and neurons and the connection between them. The input layer, output layer and hidden layers should be defined and the neurons in each layer should be recognized. The connection among neurons should be established and weights should be assigned to these connections. The network's parameters such as momentum and learning rate should be set. There is a close relation between building a model and its training. In many cases, building and training are done together and setting the parameters of one of them affects the other. In this section selecting the network size and setting the network's parameter are discussed.

Network Size. A question that the modeler may face is " What is the optimum size of network for building the model?" The size of the network plays an important role in the performance of the network. If the network is too small, it will not be capable of fulfilling the modeler expectations. On the other hand, more complex networks are capable of learning more patterns. However, one that is too large will be slow and computationally expensive, and may require a large training set to generalize well. The network size depends on the complexity of the system and the diversity of training set.

The number of inputs and outputs are automatically defined by the structure of the problem. Obtaining the optimum number of hidden layers and neurons mostly depends on the experience of the model-builder along with some experimentation.

Although it is shown that a one-hidden-layer MLP can perform any real-world mathematical function, these results do not necessarily imply that there is no benefit in having more than one hidden layer. It is shown that for some problems, a small two-hidden layer network can be used where a one-hidden-layer would require an infinite number of nodes (Chester, 1990). Multiple hidden layers can significantly reduce the need for large numbers of neurons and make the network flexible for generalization during the learning process.

Two approaches might be considered. The first one is to start with a small network and increase the size. Cascade Correlation, Project Pursuit are examples of this approach (Hush, 1993). In these methods additional nodes are created during the learning processes. Another possibility is to start with a large network and then remove weights and/or nodes which do not play important roles in the network; this method is called Pruning (Reed 1993). It has been shown that the maximum number of hidden layer nodes needed for the MLP to implement the training data is on the order of the number of training samples (Huang, 1991). So, one should never use more hidden layer nodes than training samples. Setting the network size is a process of adjustment and iteration.

Practically, it is a good method to start with one or two hidden layers. Add additional layers only when the training is impossible or difficult or ability of the neural network is unsatisfactory. It might be a good idea to increase the number of layers and neurons in large jumps (e.g. make it double in each step) and look for successful training. Afterwards, decrease the network size in small steps and find a satisfactory

size. In this research, and in most of the fully connected MLPs, no more than two-hidden layers are typically used. In recent years, genetic algorithm has been shown to have a promising application in estimating the optimum number of neurons and layers (Bebis 1995).

Setting the Parameters of the Network

There are many adjustable parameters in a network. Learning rate and momentum are among those. The adjustment of learning rate and momentum control the way in which the error is used to correct the weights in the network.

Setting the Learning Rate. When the learning rate is set to high values (close to 1), the network may become unstable. But lower values (close to 0) of the learning rate may result in longer training times. The modeler can set the learning rate to high values and then decrease that if any unstable behavior is seen. On the other hand, the modeler may start with a low learning rate and try to increase that if training is taking too long. Another approach is to make the learning rate proportional to RMS (Root Mean Square) error. In this approach, when a neural network is far from being correctly trained, the learning rate will be maximum because the RMS error is high. When the RMS error is reduced, the learning rate will be set to low values.

Setting the Momentum. The momentum makes the current search direction for new weights to be an exponentially weighted average of past directions. It keeps the weights moving across flat portions of the performance surface after they have

descended from steep portions (Hush, 1993).

The momentum should be set in the interval of (0,1). The higher the values of momentum, the greater the percentage of previous errors applied to weight adjustment in each training case. The higher value of momentum smoothes out the training process. The lower values of momentum are suggested for data which is more regular and smoother with relatively simple relation. Again the modeler should try different values of momentum and find an appropriate value for the network.

Training the Network

Training the network is the process of applying input-output vectors (in the supervised method) and choosing an algorithm that set the weights in the way that the network can generalize best. Since backpropagation has already been selected for training the network, only the number of training time and weight initialization are discussed here.

Stopping the Learning Procedure. The learning process might stop if :

1. The magnitude of gradient of weights becomes small enough.

2. The Sum-of-Squared-Error Criterion function (goal function) becomes less than a specific value.

3. The number of iterations exceeds a specific value. In this method, there is no guarantee that after this number of iterations, the algorithm reaches the minimum or near the minimum.

4. There are no more changes in the goal function if the learning process continues.

5. The cross-validation method (explained next) reaches an optimum point.

Since the generalization performance is the criterion for termination in the cross-validation, this method is recommended in most of the cases. Furthermore, on contrary to most of the other methods, cross-validation does not suffer from premature termination. While cross-validation is a widely accepted method, it can be computationally intensive and if the number of data samples is limited, this method reduces the size of the training data even further (Hush, 1993).

Cross - validation

According to Smith (1996) it is not recommended to stop training only by looking at the error on the training sample. The error on the training sample always goes down. At some point, hidden nodes find features that are present in the training sample but not in the population in general. At this point overfitting begins.

For solving this problem, in the cross-validation method, the performance of the network on the training set and validation set are considered during the learning process. The performance of the network on the training set will usually continue to improve, but its performance on the validation set will only improve to a certain point; after that the performance starts to degrade.

At this point the learning algorithm should be terminated. After this point the network starts to overfit the training set data. The error is a good indication of the

system's performance. The error is the sum of the quadratic differences between the teaching input and the real output over all output units summed over the number of patterns presented.

The weights in the cross-validation method should be saved after each iteration. Each iteration in training phase is called epoch. In each epoch the network adjusts the weights in the direction that reduces the error. Many epochs are usually required before training is completed. To make its weight adjustments, the network can be trained with a single pattern for the number of training cycle or it can be trained with all patterns for the number of training cycles specified. In this research the networks are always trained with all patterns. After finding the optimum point, the weights corresponding to that point should be applied to the network.

Figure 12 shows the Sum Square Error vs. number of epochs in a system. As is seen, the training error is always decreasing. On the other hand, validation error decreases to a point and after that it suddenly increases. This point is the optimal point and it happens after 71,000 epochs.

When the neural network is forced to learn the target values more exactly, overtraining may happen. In this case, the network tries to memorize the training set rather than learn the pattern. An overtrained network has an acceptable error of the trained data but it suffers from generalization ability. To avoid overtraining several methods are suggested:

1. One method is to put some noise in the training data. The amount of noise should not be so high that the nature and the relationships be overwhelmed, but it

should be sufficient to make the generalization better.

2. Another method is to use more training data. The complete training data may be applied to either trained network or blank net. In the former case, the training will resume with the existing weight set but using the new data set. In the later case, the training restarts from a blank neural network.

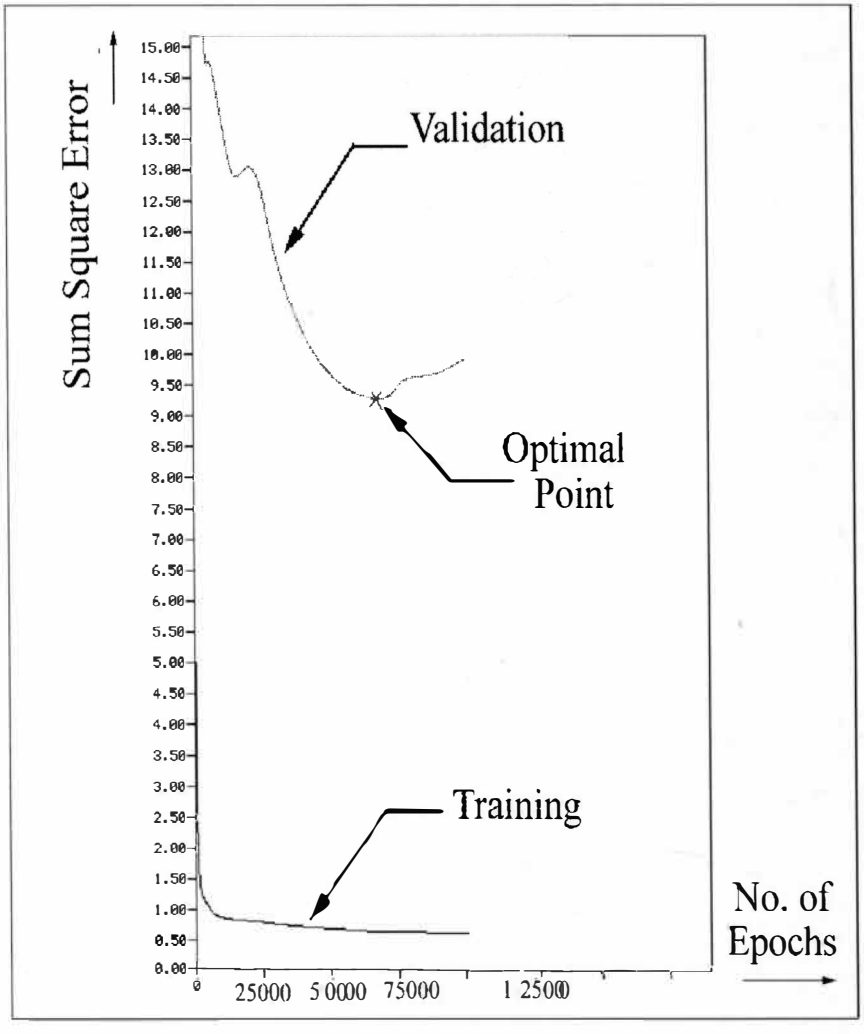


Figure 12. Error of Training and Validation Samples in the Cross-Validation Method.

Initializing the Weights. According to Chapter III, the current values of weights depend on their prior values, so the initial values of weights are very important. If the initial weights for all neurons were the same, then all neurons with the same input and output would be adjusted by the same number. In this case, the learning processing may fail. So, it is suggested to initialize the weights to random values. According to Hush (1992) initializing the weights to small values starts the search in a "relatively safe position". However, it is possible for the random initial values to change the solution that the neural net finds each time. In this case, more than one weight set satisfy the training constraints. Furthermore, the small initial values make the learning processes slower, because it takes more time for weights to reach their final values. By the way, in this research the weights are always initialized to small random numbers within the interval (-1,1).

Testing the Model

In this step, the model should be tested to prove that it is a correct representative of the real system. Testing the model is usually done after the network is built and trained. The power of the system in generalization is a good critique for testing the model. The generalization ability of the network is usually measured by using the test set. The test may be simply comparing the result from the network and the results from the real system. It may also be in the form of contour plots or charts. The less the difference, the better the model performs.

In some applications, the modeler can simply ignore this stage. In these cases,

only the training set and validation set exist. The modeler assumes that the model's performance on the validation set is a good indication of general performance of the model. In other words, the validation set and the test set are the same. However, for more precise processes it is recommended to use test samples which have not been applied to the network during the learning phase. Again, the model should be tested on the domain in which the system is supposed to work (it is crucial for the system to work within the domain that the network has been trained and tested).

Experimentation

The most attractive part of the simulation life cycle through ANN is experimentation. Rapid and parallel processing make the ANN capable of estimating the results almost instantly in the experimentation phase. Regardless of the elapsed time in the training phase, the mapping of ANNs in the experimentation phase is immediate. This unique characteristic makes ANNs suitable for interactive simulation of manufacturing systems. Having built the model, the modeler may ask what-if questions. For example, questions such as "What will be the throughput of the system if another labor is assigned to the job? or the number of machines is increased? or decreased?", etc. The modeler simply needs to apply the new input and get the output vector rapidly. The input may be a new set of data or it may be the data which has been used to train, validate or test the model. In the simple static mapping, one run of the simulation is enough but in the dynamic systems, several runs may be needed. The methodology of implementing the dynamic systems with MLPs will be discussed later.

Analysis of Results and Denormalization

Since training a neural network involves some random initialization, the results of several training runs of the same algorithm on the same data set may differ. It is suggested to make several runs and report statistics on the distribution of results obtained. Furthermore, the results should be double-checked before implementation. The modeler should check the results if they are in an acceptable range or not. The result may become misleading and implementation of these results may become dangerous if: (a) the network has not been trained properly, or (b) the data which is used to train the system has not been collected properly, or (c) if the execution of the model is based on the data set which is not in the expectation range. So, the analysis of the results is very crucial.

The ANNs usually perform very well on the data with which they are trained but not on the data which is not in their training set. If the network is trained on-line with a manufacturing system, it usually learns the routine tasks in the system. Since the results from the network are in the interval of (0,1) and sometimes (-1,1), they should be denormalized. The process of denormalization depends on the function with is used for normalization.

Implementation

This step involves implementation of decisions concluded from the simulation experimentation. The results of implemented strategy should feed back to the network.

This method is on-line training. The network tries to find new patterns and tries to learn the new behavior of the system. This helps the network to update itself with new situations. It is very useful for systems which change through time. Aging of machine tools is a good example of these phenomena. The characteristics of machines usually change through the years because of aging.

Documentation

The documentation of the model and its use is very important and essential for further study. These documents will be useful for troubleshooting and maintaining the system. They can also be used as references for implementing similar models. The following items should be considered in a document: (a) problem definition including the name, address, version, objectives, comparing criteria; (b) the data: training set, validation set and test set; (c) network topology including nodes, connections, activation functions; (d) initialization; (e) algorithm parameters (momentum and learning rate); (f) termination criteria; (g) error function and its value in the reported result; (h) number of runs; and (i) the hardware, operating system and software name and version which is used in the experiments.

Most of the accredited software environments provide users with facilities for saving the networks and their parameters. It is recommended that modeler have several backups of the model and the results of its implementation.

Although many suggestion and guidelines were provided in this chapter, still the design and implementation of an appropriate ANN depends mainly on the

experience, innovation and knowledge of modeler. In brief, design and implementing a good ANN is as much an art as a science.

CHAPTER VI

MANUFACTURING SYSTEMS

Introduction

Most manufacturing systems can be modeled based on a combination of queues and machines. Figure 13 shows a simple manufacturing system.

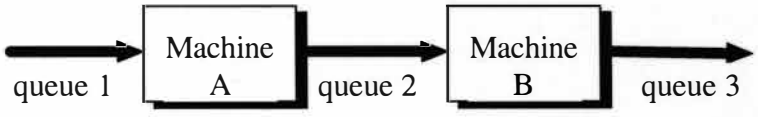


Figure 13. A Simple Manufacturing System.

To use ANN in modeling manufacturing systems such as Figure 13, two approaches may be considered:

1. To consider the whole system as a black box and try to find an appropriate ANN structure which is able to estimate the output of the black box based on its input;

2. To consider the system as consisting of components (e.g. queues and machines) and try to find appropriate ANN topologies which can simulate these components' behavior. These ANN modules can then be assembled together to simulate the whole system.

In both approaches, the modeler should go through the procedure offered in

Chapter V: gathering data, building and training the network, etc. However, the complexity of systems and implementation of results may differ. Both methods are time consuming and involve many trials and errors. In both cases, the trained networks respond rapidly to the new set of data.

A basic queuing system and a simple manufacturing system are modeled in this chapter. Modeling the queuing system is an example of modular approach to simulate basic components in complex manufacturing systems. It also gives the direction of how ANNs can be used for modeling static systems. The static modeling will be the base of modeling the stochastic and dynamic systems offered later in this chapter. The manufacturing system modeled in this chapter shows the ANNs' capability to capture the behavior of stochastic processes. Three approaches are examined for capturing the stochastic behavior of the a manufacturing system. This system is also modeled through modular approach. The results in each case are explained and compared to those of conventional simulation methodology.

Simulation of Queuing Systems Using ANNs

A basic queuing system has been modeled in this survey. There are three reasons for choosing this system:

1. The behavior of queuing systems is well known and it is possible to compare the ANN's accuracy with a real system.

2. The procedures for developing ANN to model a queuing system are typical procedures for many manufacturing systems. So, it is useful in solving problems which

may arise when a modeler is trying to use ANN to model a manufacturing system.

3. It is a systematic attempt to create a library of ANN modules for manufacturing simulation.

Definitions

Waiting lines - called queuing systems - usually occur when the demand for current service exceeds the current capacity of servers. Providing the right amount of service in queue is important in manufacturing systems. Not enough capacity causes long waiting lines and too much capacity involves excessive costs.

The most common type of queue is one in which a single waiting line forms in the front of a facility which has one or more servers. The entities of a queue are usually generated by an input source (based on a statistical distribution). Each entity then waits in the queue (waiting line). After spending some time (waiting time), each entity is served by one of the servers. Examples of entities are unfinished parts, pieces of equipment and finished products. For studying the queuing models, it is assumed that all interarrival times and service times are independent and identically distributed. A queue is recognized by its input distribution, service distribution, number of servers and probably the maximum capacity of the queue. The queues are usually labeled based on their characteristics. Figure 14 shows the labeling method of queues.

Each distribution has its own label. For instance M stands for exponential distribution (Markovian), or D stands for degenerate distribution (constant times). By this definition, M/M/4 stands for a queue system with exponential interarrival time,

exponential service time and 4 servers.

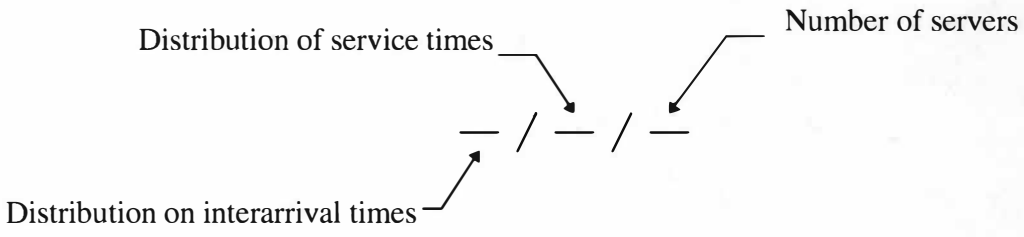


Figure 14. The Method of Labeling a Queue.

Industrial engineers are usually interested in studying the probability of no entity in the queue (P_0), mean waiting time in the queue (W_q) and mean length of the queue (L_q). They usually address the queues based on the mean of interarrival rate (λ), mean of service rate (μ) and the number of servers. The relation between these variables is shown in (Hillier, 1995). In this section, first two simple system of M/M/1 and M/M/2 are considered, followed by a general model of M/M/S which is modeled through ANN.

M/M/1 and M/M/2 Systems

The first set of experiments was concerned with the application of ANN in modeling M/M/1 and M/M/2 queuing systems. In these experiments ANN was considered as a map function which generated P_0, W_q and L_q based on λ and μ. The schematic of this approach is shown in Figure 15.

In the experiment, λ and μ were allowed to vary between 1 and 50 and the

number of servers was fixed to 1 and 2 in M/M/1 and M/M/2, respectively. Only two sets of data, training and test set, were generated by a program written in C language. The generated data was then converted to the format which was compatible with SNNS software. These data then were applied to an SNNS software version 3.1 which had been installed in IPC Sun workstations with Sun OS 4.1.4 operating system.

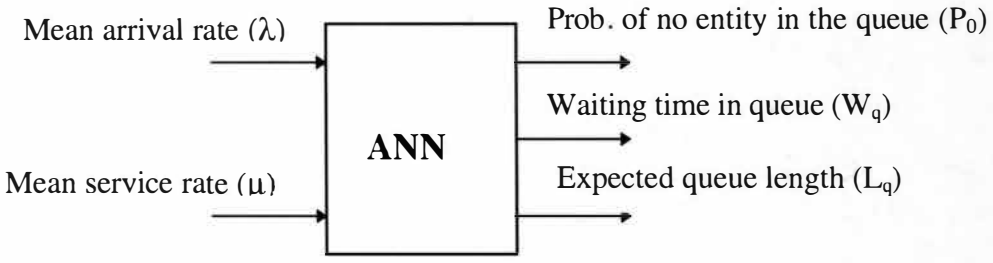


Figure 15. Application of ANN in Modeling M/M/1 and M/M/2 Systems.

Proposed methodology is based on the efforts that have been done for solving the problems which occurred in this experimentation. these problems. For example, through trial and error and survey of literature, it was found that:

1. The network performs better if the input vector is normalized to the interval which has some upper and lower margins;

2. Generating the training set randomly or applying it after shuffling has a great effect on the learning ability of the network. In those experiments in which the training data was followed a pattern (say, λ and μ increase by 3 in each step), the network was not able to generalize appropriately.

In the M/M/1 case, at first, only the probability of no entity in the queue (P_0)

was considered as the output of ANN. This decision was due to the simple relationship between P_0, λ and μ. The MLP network with the backpropagation momentum learning method was considered. Using Hush's (1992) recommendations, the initial weights were set to be random numbers in interval of (-1, 1). The input and output vectors were normalized. The learning rate of 0.2 and momentum of 0.1 were selected.

The MLP network with 6 nodes in one hidden layer was able to approximate the system effectively. Adding waiting time (W_q) and expected queue length (L_q) to the output of the ANN makes the system so complicated that ANN was not able to understand the system. After testing some networks, a network consisting of two hidden layers with 9 neurons in each layer was found to be able to estimate the queue's behavior.

The network was trained by 176 training points which had been generated randomly. Only the training set was considered for stopping the training procedure. The training was stopped after the Sum of Squared Error (SSE) of the training set became almost smooth with a value of less than 0.3. The system was tested by 1,176 data points. The results were promising (the similar results of an M/M/2 are shown in this chapter). The same approach was chosen for the M/M/2 queuing system. This time the network was tested by 1801 data points. Figures 16, 17 and 18 compare the contour plot results from ANN and the corresponding real M/M/2 system. According to the queuing theory, the modeled systems are valid only when the interarrival rate is less than the service rate. Therefore, the readers should notice that the proposed figures are valid where ($\lambda < \mu$).

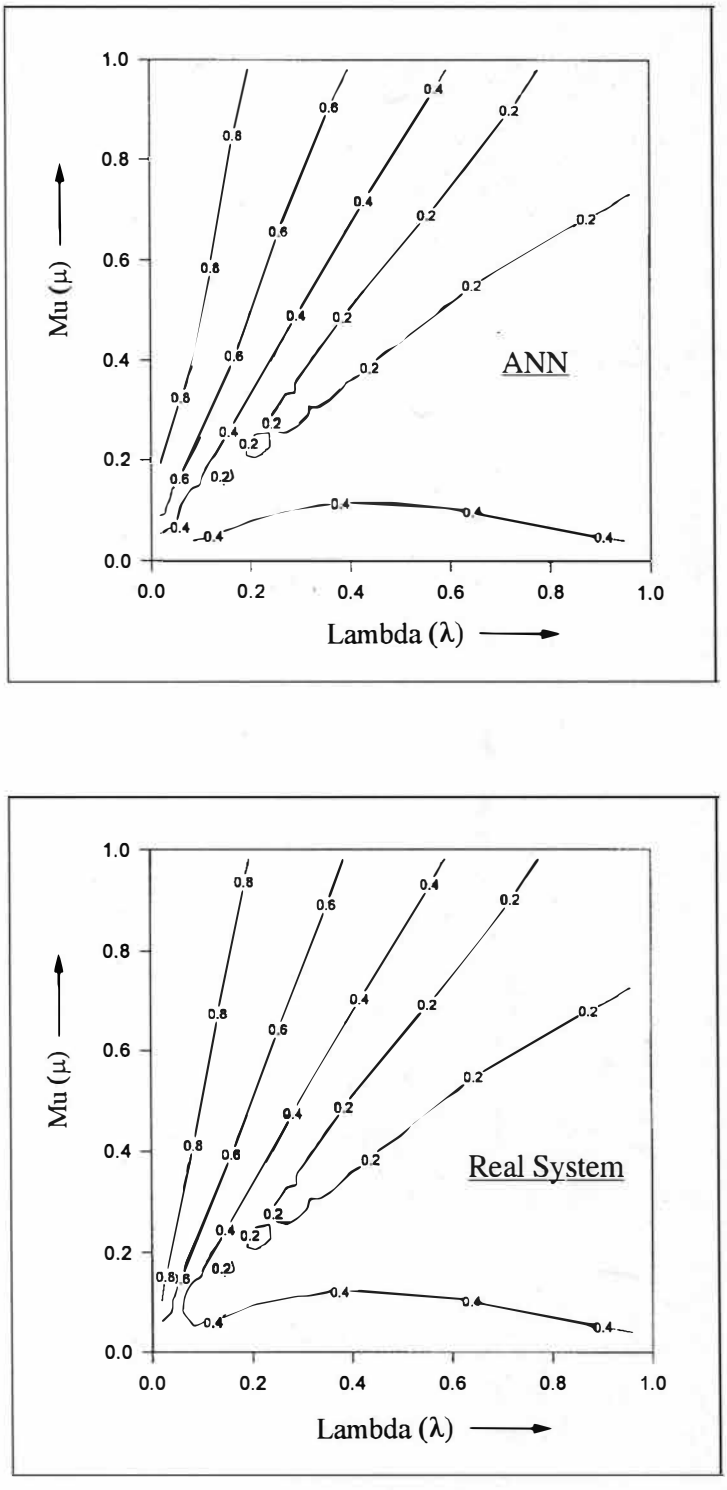


Figure 16. Comparison Between Probability of No Entity (P_0) in the M/M/2 Queuing System Generated by ANN and the Real System.

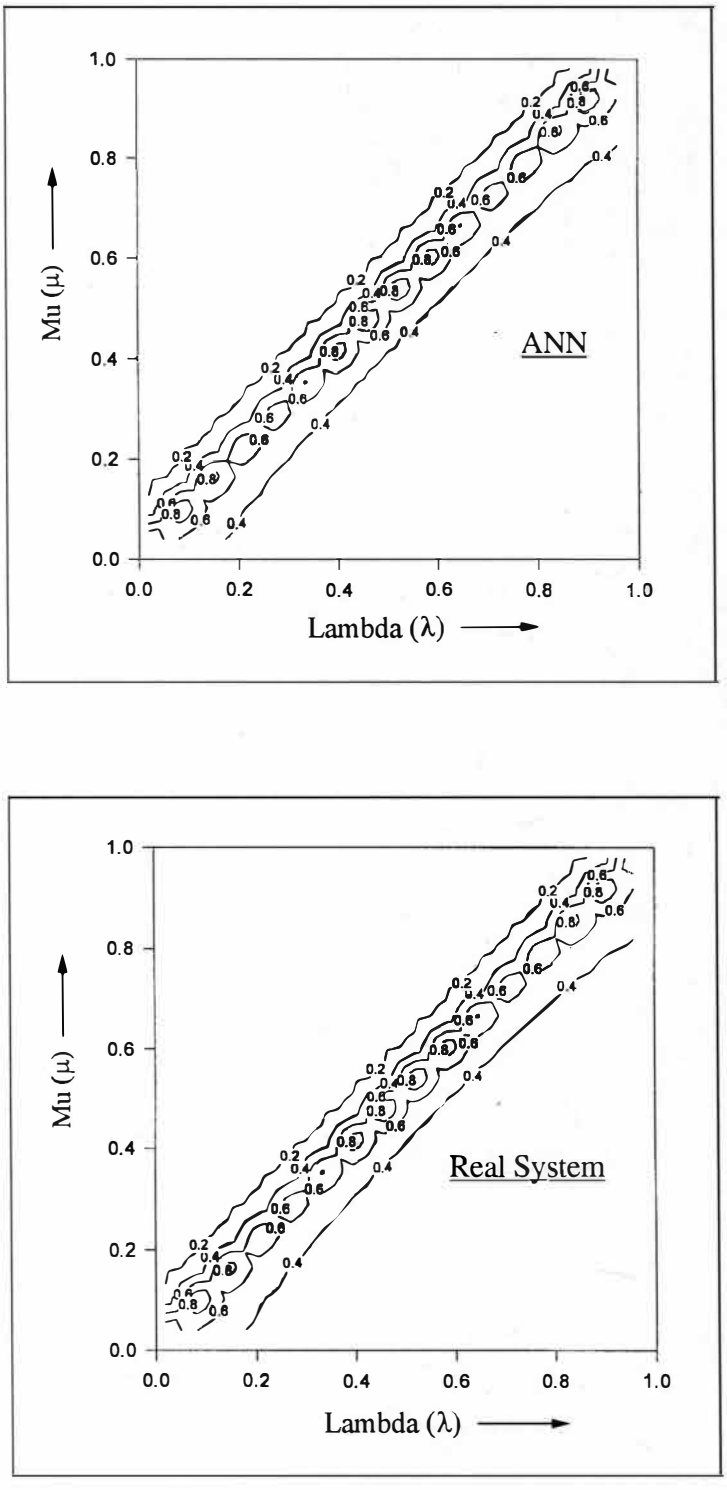


Figure 17. Comparison Between Waiting Time (W_q) in the M/M/2 Queuing System Generated by ANN and the Real System.

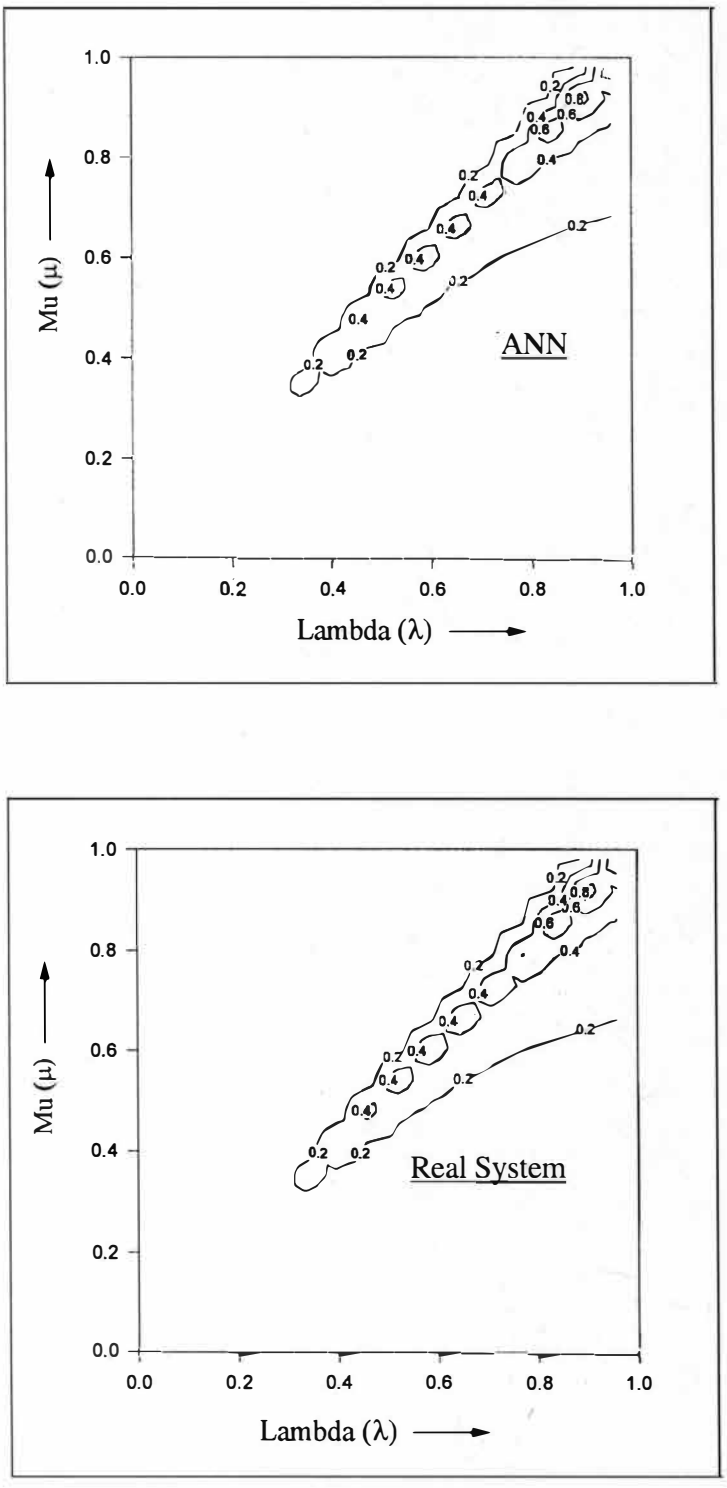


Figure 18. Comparison Between Expected Queue Length (L_q) in the M/M/2 Queuing System Generated by ANN and the Real System.

As shown, the ANN is able to realize the pattern of the queuing systems. The results were so promising that encouraged us to study the behavior of the network in more general situations. Therefore, a general M/M/S queuing system was considered. However, many questions still remained unanswered in that stage; such as "What is the best topology of ANN for a given problem?".

M/M/S Queuing System

This system consists of queues with exponentially distributed interarrival time, exponentially distributed service time and several servers. The theory and discussion of this system are presented in (Hillier, 1995). Once more, the ANN strategy shown in Figure 15 was used. But in these experiments, another input (number of servers, S) was introduced as well. The outputs of the ANN system remained unchanged (L_q, W_q and P_0). In these experiments, λ and μ were allowed to be any integer number in the interval (1, 50). The number of servers can be between 1 and 10. Training, validation and test data sets were generated by a program written in C language. The generated data were normalized and prepared in a format compatible with the SNNS software. The programs developed for generating the training and validation data set are provided in Appendix A. These normalized sets were then applied to different configurations of ANN to find the best architecture.

Several Sun Sparc-5 workstations with Solaris operating system (version 5.5) were used. Using SNNS software version 4.0, the MLP with backpropagation momentum learning method was studied. Again, the initial weights were set to random

numbers in the (-1,1) interval. The learning rate of .2 and momentum of .1 were chosen. The cross validation method was used to obtain the optimum number of epochs.

For finding the best network size, several experiments were conducted. The networks with small sizes and large sizes were considered. In each case, Sum of Squared Error (SSE) of training set and validation set were drawn together. The effect of the network's size on the network's performance was studied.

In the first experiment, an MLP network with 9 neurons in one hidden layer was considered. The network was trained with a training set consisting of 300 points. As shown in Figure 19, the network performs very well on the training data

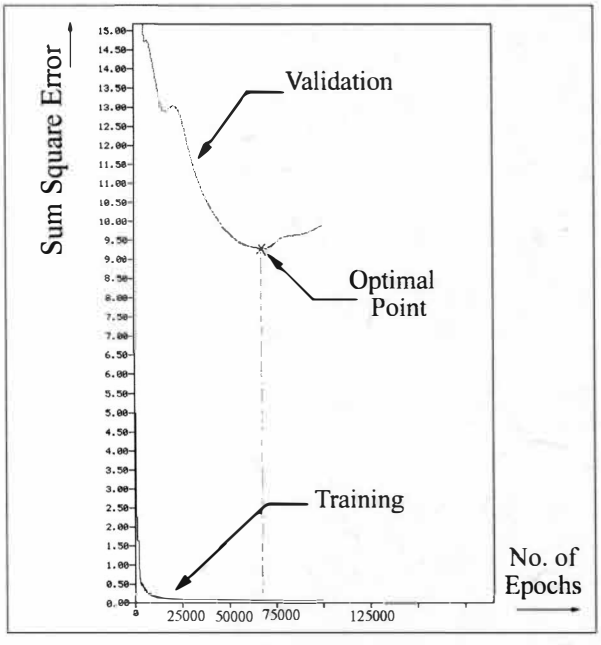


Figure 19. The SSE of an MLP Consists of 9 Neurons in One Hidden Layer Trained by 300 Points.

After 71,000 epochs the network finds a pattern which exists in the training set but it is not valid for the validation set. At this point, the overfitting occurs. Even in this optimum point the SSE of validation set is far from the ideal (the minimum value of SSE of validation is more than 9). After studying the trained network with a test set, it was found that in some areas the network has not been trained properly. This was because of a lack of training points in these areas. So, the training set was not a good representative of all of the data points. Another experiment with 18 neurons in one hidden layer MLP confirmed that the poor generalization is not due to the network size. The results of this experimentation are shown in Figure 20.

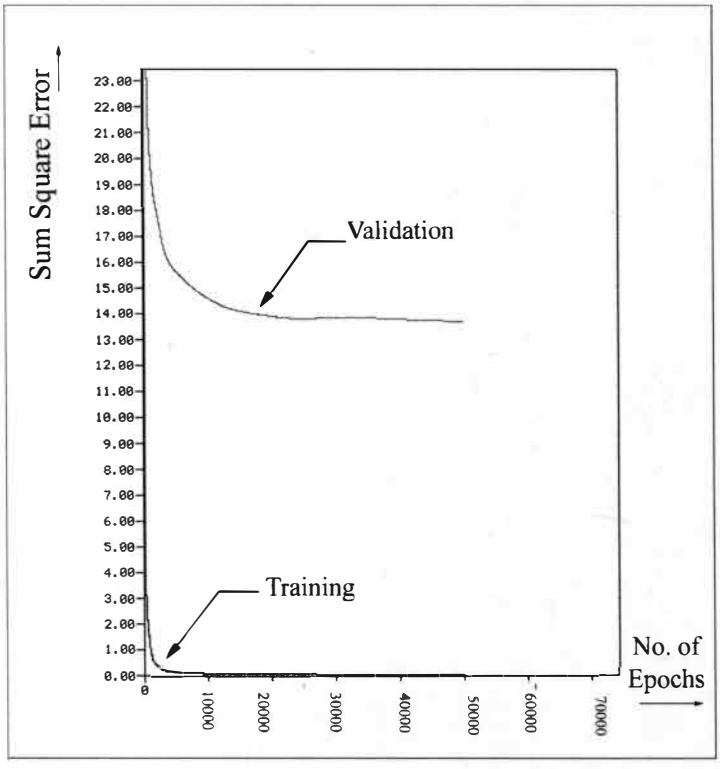


Figure 20. The SSE of an MLP Consists of 18 Neurons in One Hidden Layer Trained by 300 Points.

As shown in Figure 20, the minimum of SSE in the validation set is still around 14, but the SSE of training set is almost zero. Therefore, the performance of the network in training points has been improved, but the network still suffers from good generalization ability.

Based on these two experiments it was decided to generate the training set which was a better representation of the data points. Therefore, a training set consisting of 1000 points was generated.

Once more, a network with 9 neurons in one hidden layer was studied. Figure 21 shows the performance of this network on the training and validation sets.

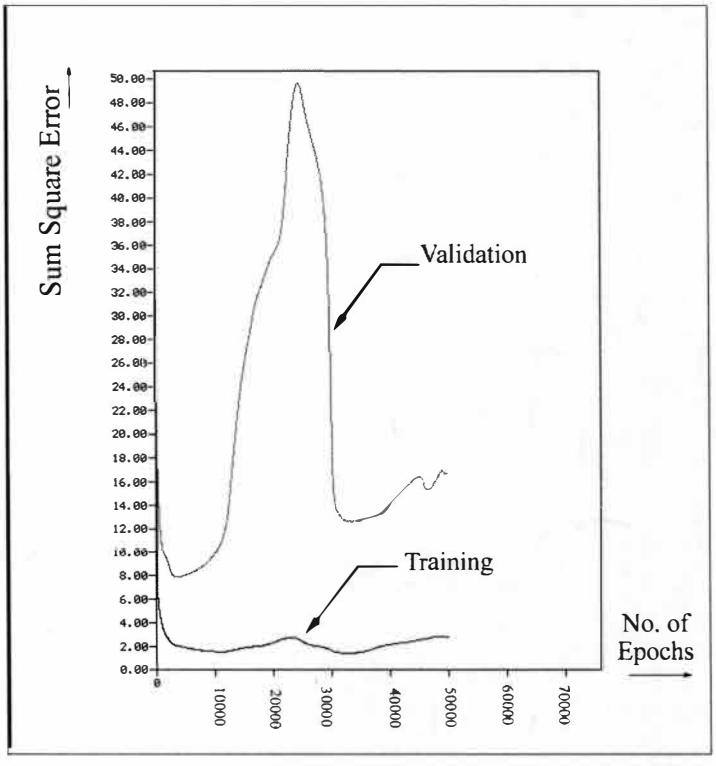


Figure 21. The SSE of One Hidden Layer MLP Consists of 9 Neurons Trained by 1000 Points.

As seen in Figure 21, the network has difficulties understanding the training set. If the network can not be trained properly, it will not have satisfactory performance on the validation and test sets.

Thus, another experiment was conducted to understand if this phenomena is due to network size. This time a network with 18 neurons in one hidden layer was considered. As shown in Figure 22, the network was able to successfully understand the training data. The SSE of validation set showed rapid decreasing and optimum point had an error which was less than 5.

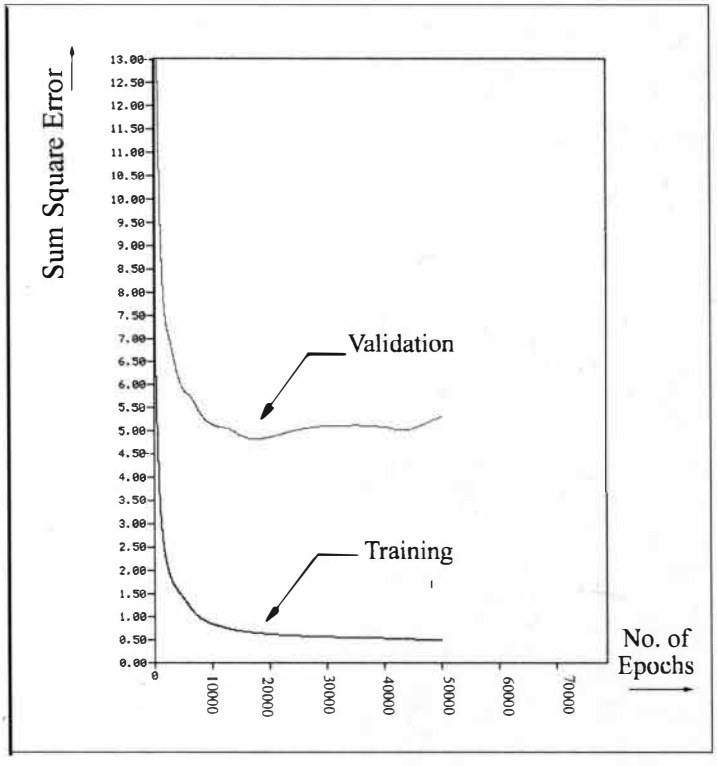


Figure 22. The SSE of an MLP Trained With 1000 Points for One Hidden Layer Consists of 18 Neurons.

According to Smith (1996), improving the network size increases its power to learn more complex patterns. But the question was "How big may the size of network be?".

To answer this question in another experiment an MLP with 72 nodes in one hidden layer was chosen. This network was trained by 1,000 points. Figure 23 shows the performance of this network. The network can learn from training sets. The minimum of SSE of the validation set is almost the same as the previous experience. However, the network suffers from some instability. It seems that the system has some noisy behavior.

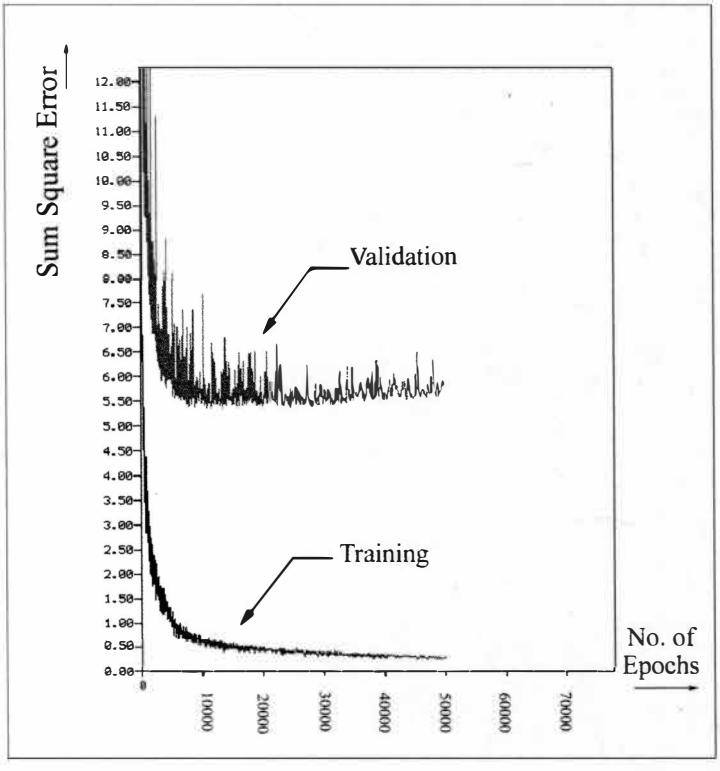


Figure 23. The SSE of an MLP With 72 Neurons in One Hidden Layer Trained by 1,000 Points.

For studying the effect of larger networks, another experiment was conducted. In the experiment, a one hidden layer MLP with 144 neurons was considered and trained with 1,000 data points. Figure 24 shows the results of this experiment.

As shown in Figure 24, although the envelope of SSE of validation and data sets are almost the same as the network with 9 nodes (Figure 22), the network suffers from some random behavior.

Based on the last two experiments, it was concluded that increasing the network's size not only requires more learning time, but also might give a poor result of generalization.

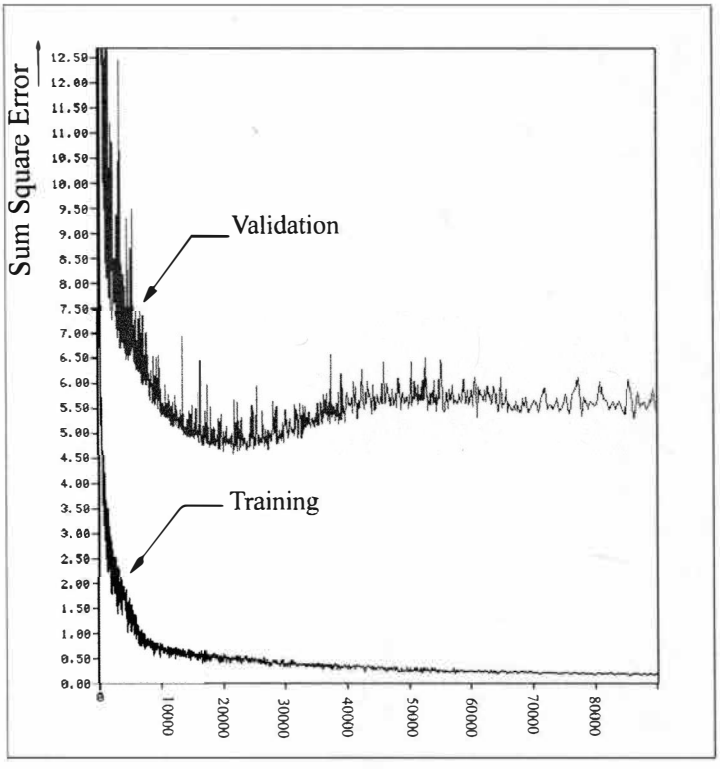


Figure 24. The SSE of an MLP With 144 Neurons in One Hidden Layer Trained by 1,000 Points.

Another question was "What would happen if the number of layers are increased?". To answer this question, two experiments were conducted. In the first experiment an MLP with two hidden layers was considered. The first layer had 36 neurons and the second one had 9. Again, the network was trained with 1,000 points. The SSE of validation and training are shown in Figure 25.

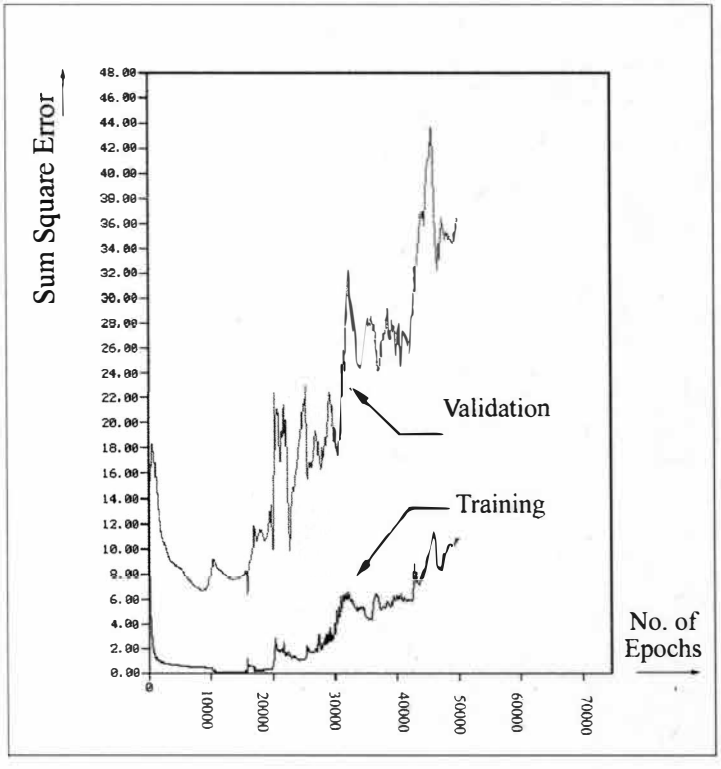


Figure 25. The SSE of an MLP With 36-9 Neurons in Two Hidden Layers Trained by 1,000 Points.

In a similar experiment, an MLP with two hidden layers --36 nodes in the first layer and 27 nodes in the second layer-- was considered. As shown in Figure 26, the network's performance is better than the previous case, however, the performance of

the system is not as good as Figure 22. In one point, the SSE of validation decreases to 3.5. This point can not be considered as an optimal point because it is not stable and the SSE of the training set shows an increase in that area. Based on these experiments it can be concluded that increasing the number of neurons and layers is not necessarily helpful to the network's performance. The modelers should search for the optimum number of neurons and layers.

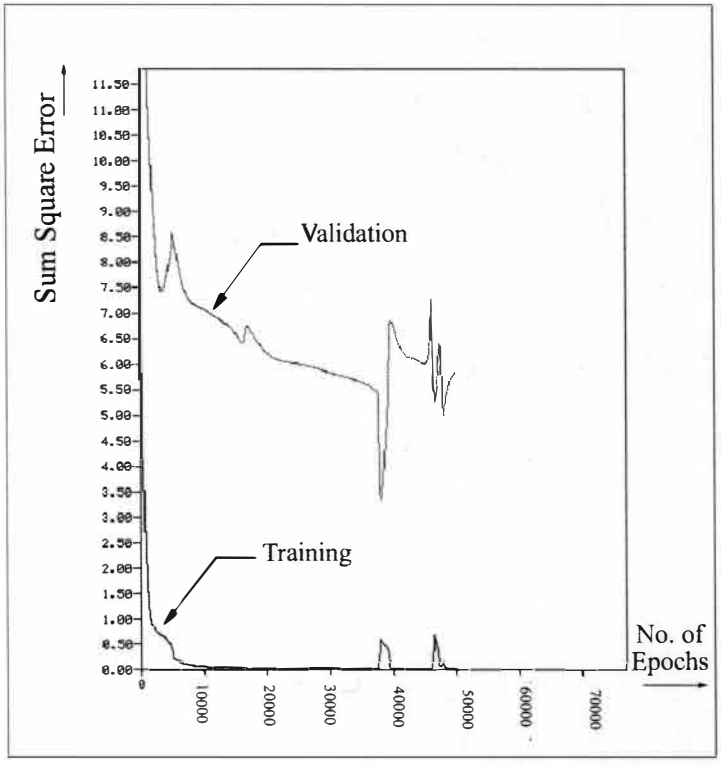


Figure 26. The SSE of an MLP With 36-27 Neurons in Two Hidden Layers Trained by 1,000 Points.

Testing the Network

The network with 18 neurons in one hidden layer showed the best performance

among the experimented networks (consider Figure 22). To criticize the performance of this network, an experiment was conducted. In the experiment, the performance of the network was compared to that of the real world. First, a test set consisting of 1,000 randomly-generated data points was applied to the network. This set had not been applied to the network before and it was the first time that they were applied to the network. After applying the input data (λ, μ and S), the network almost instantly came up with output data (L_p, W_q and P_0). The network output vector was then compared to that of the real system. Since the input data include three dimensions (λ, μ and S), it was difficult to compare the results through contour plots. Thus, other statistical tests were chosen for comparing the vector pairs. Each vector pair consisted of the vector generated by network and the corresponding vector for the real system. For example, P_0 generated by the network and P_0 from the real system. Using Minitab software version 9.1 for VAX/VMS, for each vector pair the following two statistical tests were conducted:

1. Two sample t-test was run to see if the mean of first vector (M_1) was equal to the mean of the second vector (M_2). A 95% confidence interval for $M_1 - M_2$ was also constructed.

2. By subtracting corresponding vectors ($d = d_1 - d_2$), a new vector was generated. The mean and variance of this vector were studied and the histogram of points were drawn. In the ideal case, the mean and variance should be close to zero. In this case, the results of ANN are exactly the same as the results of the real system.

Table 1

Comparison Between the Probability of No Entity in the Queue (P_0) Generated by ANN and the Real System

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
Real System	1000	0.294	0.198	0.00626
Neural Network	1000	0.294	0.198	0.00625

95% confidence interval for $M_1 - M_2$ = (-0.01726, 0.01743)

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	1000	-0.00009	-0.00037	0.00874	-0.06916	0.05352

Histogram of (d1-d2); each * represents 20 obs.

Midpoint	Count	
-0.07	1	*
-0.06	0	
-0.05	1	*
-0.04	1	*
-0.03	3	*
-0.02	24	**
-0.01	150	**********
0.00	689	**********
0.01	84	******
0.02	25	**
0.03	14	*
0.04	4	*
0.05	4	*

Table 2

Comparison Between the Waiting Time in the Queue (W_q)
Generated by ANN and the Real System

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
Real System	1000	0.0241	0.0942	0.00298
Neural Network	1000	0.0240	0.0901	0.00285

95% confidence interval for $M_1 - M_2$ = (-0.008063, 0.008103)

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	1000	-0.00002	0.00000	0.04805	-0.76511	0.42500

Histogram of (d1-d2); each * represents 20 obs.

Midpoint	Count	
-0.8	1	*
-0.7	0	
-0.6	0	
-0.5	2	*
-0.4	0	
-0.3	2	*
-0.2	6	*
-0.1	1	*
0.0	963	**********
0.1	14	*
0.2	8	*
0.3	1	*
0.4	2	*

Table 3

Comparison Between the Length of the Queue (L_q)
Generated by ANN and the Real System

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
Real System	1000	0.01354	0.0616	0.00195
Neural Network	1000	0.0138	0.0588	0.00186

95% confidence interval for $M_1 - M_2 = (-0.005546, 0.005019)$

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	1000	0.00026	0.00000	0.03603	-0.74954	0.34821

Histogram of (d1-d2); each * represents 20 obs.

Midpoint	Count	
-0.7	1	*
-0.6	0	
-0.5	0	
-0.4	1	*
-0.3	1	*
-0.2	1	*
-0.1	4	*
0.0	980	**********
0.1	6	*
0.2	4	*
0.3	2	*

Based on statistical experiments, the ANN is able to successfully model the system. The performance of ANN in simple functions (such as P_0) is better than complex functions. There are still a few points which are out of the acceptance range. These points mostly belong to the areas that neural network has not been trained properly. More data makes the ANN more accurate.

Simulation of a Manufacturing System Using ANNs

The results of the previous section's experiments showed the capability of ANNs in simulating a queuing system. It showed the power of ANNs in modeling of static systems. Modeling the stochastic systems is not as easy as that of the static systems. Because, in stochastic systems each input set may generate different output sets. The output values depend on random distribution of processes.

In this section, a simple manufacturing system is modeled by several ANNs. Three methods are suggested to capture the stochastic behavior of the system. The offered methodologies are explained in each case and the results are discussed.

Illustrative Example

This example (Figure 27) has been taken from (Nuila and Houshyar, 1993). Consider a simple manufacturing system with a machining center, an inspection station, and a rework station. Products of the machining center are inspected at the inspection center. Ninety percent of the inspected parts are acceptable and are send to shipping, whereas the remaining 10 percent are unacceptable and are sent to the

rework station for rework. Upon completion of rework they are also subject to inspection. Raw material randomly arrives at the plant at a rate of 1 per minute (i.e., interarrival time between parts is exponentially distributed with a mean of 1 minute). Processing time at the machining center, inspecting time at the inspection center, and reprocessing time at the rework station are random. They follow uniform, uniform, and exponential distribution, respectively. Their corresponding parameters are:

1. Machining center processing time is uniformly distributed between 2.5 and 3.2 minutes. The number of machines can vary between 1 and 10.

2. Inspection time is uniformly distributed between 2 and 3 minutes. The number of inspection stations can also vary between 1 and 10.

3. Rework processing time is exponentially distributed with mean of 10 minutes. The system starts out with no parts present, the machines and the inspector are idle and ready for the operations. In addition, there are no set-ups, interruptions, and or breakdowns.

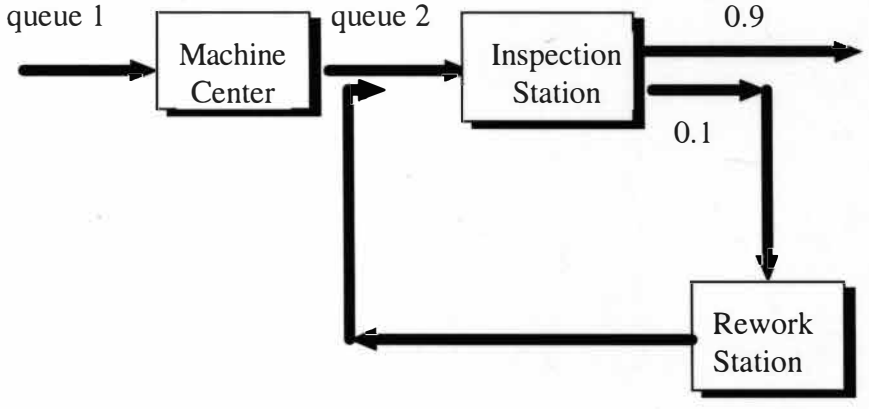


Figure 27. Graphical Representation of the Illustrative Example.

Many variables affect on the behavior of the system, e.g.: processing time of machine centers and inspection stations, failure rate, queues' capacities, the number of machine centers and the number of inspection stations. For simplicity, we focus on the number of machine centers and inspection stations. Specifically, we are interested in the system's performance for an eight hour shift based on the number of machine centers and inspection stations. The readers should notice that other variables can be used in combination or as substitutes of these two variables. Regardless of selected variables, the proposed methodologies can be used. Figure 27 is a graphical representation of the system with one machine center and one inspection station.

This simple manufacturing system was modeled using SLAMSYSTEM software (student version 4.5), and the statistics on the throughput of the system was gathered. The SLAM's model is presented in Appendix B. It is assumed that the results from SLAMSYSTEM are the same as the results from the real system.

To select the number of test points, Montgomery's (1991) recommendations was used. The null hypothesis checked if the mean of data generated by SLAMSYSTEM (μ_1) was equal to the mean of results generated by ANNs (μ_2). It was assumed that the two population variances were unknown but almost equal. Furthermore, the sample sizes from the two populations were assumed to be equal. We wanted to reject the null hypothesis 95% of time if the difference between the normalized means ($|\mu_1 - \mu_2|/2$) was equal or more than 0.15. Therefore the probability of type II error (β) was 0.05. Assuming that the standard deviations would not exceed

0.4, yielded $d = \frac{|\mu_1 - \mu_2|}{2\sigma} = 0.9375$. The operating characteristic curves for the two sided t-test with $\alpha = 0.05$ (Montgomery, 1991, page 32) implied n=9. To be on the safe side, the number of test points was set to be 10.

The number of iterations was calculated based on $n = (\frac{S * t_{\alpha/2,n-1}}{\mu - \mu_0})^2$. T-distribution itself is a function of n. However, n was calculated by using trial and error. With 95% confidence interval and assuming that error of μ would be less 15 (readers should notice that the μ can vary between 170 and 530), the number of runs calculated to be n= 17.52. To be in the safe side, 20 iterations considered for each experiment. The number of machine centers and inspection stations were generated randomly. To capture the stochastic behavior of the system three methods were examined.

In each of these methods, an MLP network with 18 nodes in one hidden layer was used and the backpropagation momentum learning method was applied. Using the guidelines offered in the Chapter V, the initial weights were set to be random numbers in the interval of (-1, 1). A training set consisting of 31 points was generated. The learning rate of 0.2 and momentum of 0.1 were selected. The gathered data were normalized to the interval of (0.1, 0.9). The training was stopped only after 5,000 epochs. After training, the results from ANN were compared to those of SLAMSYSTEM. Readers should keep in mind that these comparisons are done based on raw output data. It means the normalized outputs were used for these comparisons. These methods are discussed next.

Method One (Mean and Standard Deviation)

In this method, the number of machine centers and the number of inspection stations are considered as the input of ANN. The mean of throughputs and standard deviation of throughputs are considered as output. Figure 28 shows the schematic of this method. After training the network, the performance of the network was tested based on test data. The results are shown in the Tables 4 and 5.

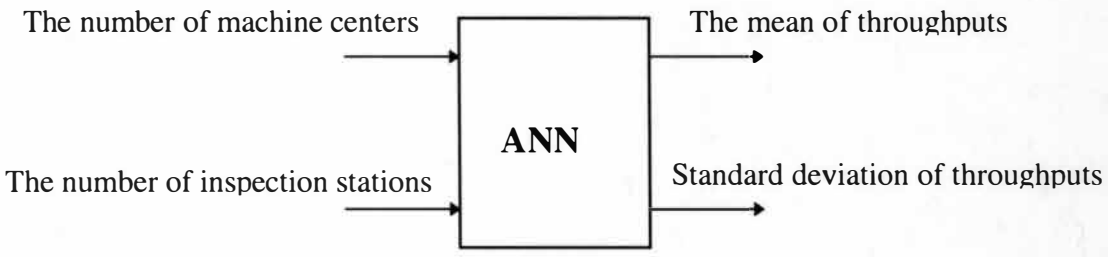


Figure 28. Using the Mean and Standard Deviation to Capture Stochastic Behavior of a Manufacturing System.

According to the results, ANN is capable of understanding the mean and the standard deviation of throughputs in the illustrative example. Based on two sample t-test, there is no significant difference between the results generated by ANN and the results from ANN. The histogram of difference between two methods also shows that the ANN can effectively capture the system's behavior. Using the mean and standard deviation is one way to capture the stochastic behavior of the system. There are other techniques in this field which are discussed next.

Table 4

Comparison Between the Mean of Throughputs Generated by ANN and SLAMSYSTEM in Method One

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
SLAMSYS	10	0.579	0.352	0.111
Neural Network	10	0.605	0.329	0.104

95% confidence interval for $M_1 - M_2$ = (-0.3474, 0.2956)

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	10	0.0259	0.0327	0.0425	-0.0454	0.0957

Histogram of (d1-d2)

Midpoint	Count	
-0.04	1	*
-0.02	2	**
0.00	0	
0.02	1	*
0.04	4	****
0.06	1	*
0.08	0	
0.10	1	*

Table 5

Comparison Between the Variance of Throughputs Generated by ANN and SLAMSYSTEM in Method One

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
SLAMSYS	10	0.454	0.307	0.0972
Neural Network	10	0.462	0.298	0.0941

95% confidence interval for $M_1 - M_2 = (-0.2937, 0.2772)$

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	10	0.0082	0.0016	0.1019	-0.1809	0.1708

Histogram of (d1-d2)

Midpoint	Count	
-0.20	1	*
-0.15	0	
-0.10	1	*
-0.05	2	**
0.00	2	**
0.05	2	**
0.10	0	
0.15	2	**

Method Two (Mean and Confidence Interval)

Modelers can also use the confidence intervals of the output for modeling their stochastic processes. According to Hurrion (1992), the MLPs are able to capture the randomness of the systems if the upper bounds and lower bounds of the confidence interval are also included in the output. This method is similar to the method which was used in the previous section; however, in this method, upper and lower bounds of confidence interval are considered instead of standard deviation.

In this method, several replicates of the desired output should be gathered for each set of input. The mean and upper and lower confidence intervals of the output should be calculated for each set of input. Then the network should be trained based on the input vector and corresponding desired output and output's upper and lower bounds. Figure 29 shows this method. The results of this approach are shown in Tables 6-8.

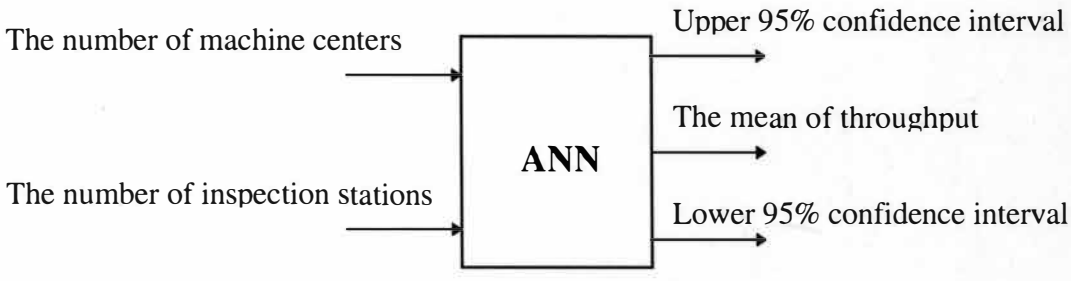


Figure 29. Using the Upper and Lower Confidence Interval to Capture Stochastic Behavior of a Manufacturing System.

Table 6

Comparison Between the Mean of Throughputs Generated by ANN and SLAMSYSTEM in Method Two

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
SLAMSYSTEM	10	0.579	0.352	0.111
Neural Network	10	0.600	0.337	0.107

95% confidence interval for $M_1 - M_2$ = (-0.3461, 0.3039)

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	10	0.0211	0.0249	0.0347	-0.0376	0.0796

Histogram of (d1-d2)

Midpoint	Count	
-0.04	1	*
-0.02	1	*
0.00	2	**
0.02	2	**
0.04	2	**
0.06	1	*
0.08	1	*

Table 7

Comparison Between the 95% Upper Bound Confidence Interval of the Mean of Throughputs Generated by ANN and SLAMSYSTEM

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
SLAMSYSTEM	10	0.579	0.352	0.111
Neural Network	10	0.598	0.339	0.107

95% confidence interval for $M_1 - M_2 = (-0.3460, 0.3062)$

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	10	0.0199	0.0244	0.0362	-0.0347	0.0855

Histogram of (d1-d2)

Midpoint	Count	
-0.04	1	*
-0.02	1	*
0.00	2	**
0.02	2	**
0.04	3	***
0.06	0	
0.08	1	*

Table 8

Comparison Between the 95% Lower Bound Confidence Interval of the Mean of Throughputs Generated by ANN and SLAMSYSTEM

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
SLAMSYSTEM	10	0.580	0.351	0.111
Neural Network	10	0.603	0.335	0.106

95% confidence interval for $M_1 - M_2$ = (-0.3475, 0.3010)

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	10	0.0232	0.0263	0.0340	-0.0379	0.0767

Histogram of (d1-d2)

Midpoint	Count	
-0.04	1	*
-0.02	1	*
0.00	2	**
0.02	3	***
0.04	1	*
0.06	1	*
0.08	1	*

Method Three (Performance Exceedance Probability)

Another approach is through performance exceedance probability (Flood, 1996). Performance exceedance probability is an indication of the performance which is more than a certain limit at a specific percentage of time. For example, 0.1 represents the performance that is exceeded 10% of the time or 0.9 refers to the throughput which is exceeded 90% of the time (Figure 30).

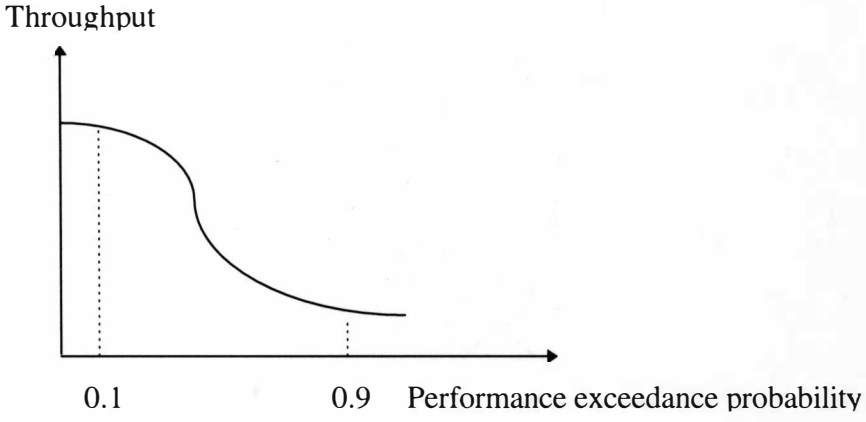


Figure 30. Throughput of the System vs. Performance Exceedance Probability.

According to this method, the performance exceedance probability should be added as an input to the ANN (Figure 31). The throughput can be considered as the output of ANN. The network should be trained on these input/output sets. After training, the output of the network estimates the throughput that corresponds to the probability exceeding value presented at the input. The main advantage of this method is its ability to give better information about the output. Industrial Engineers are

usually interested in more than mean and variance or confidence intervals; they are sometimes looking for the distribution of the outputs. This method gives better understanding of the distribution of outputs.

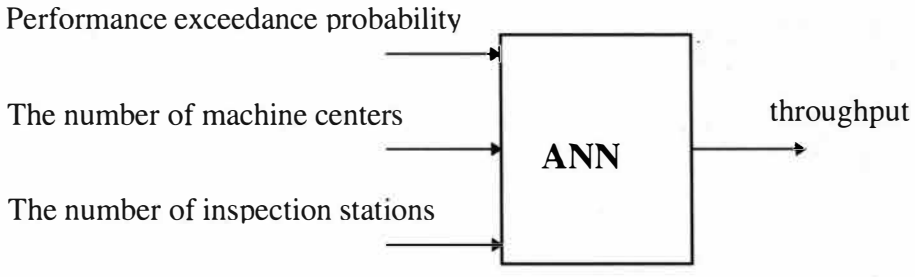


Figure 31. Performance Exceedance Approach for Capturing the Stochastic Behavior of the Manufacturing System.

After training, a test set including 10 samples was used to test the network's performance. Each sample included the number of machine centers and the number of inspection stations which were generated randomly. Figure 32. shows the performance of the network for 4 machines and one inspection center. This test sample was also existed in the training set. As shown, the ANN could learn the pattern very efficiently. The ability of network in learning the distribution of the output was not limited to the training points. ANN were also able to generalize the distribution.

Figures 33-40 show the performance of the network in the new set of data. This set of inputs had not been applied to the network before. The results show that ANN can also be used in estimating the distribution of desired outputs in a manufacturing system.

P.E.P.	SLAMSYS	ANN
0	0.30360	0.29445
0.1	0.30051	0.29422
0.2	0.30017	0.29403
0.3	0.29674	0.29392
0.4	0.29468	0.29390
0.5	0.29331	0.29401
0.6	0.29262	0.29426
0.7	0.29108	0.29468
0.8	0.28782	0.29529
0.9	0.28628	0.29609
1	0.27787	0.29709

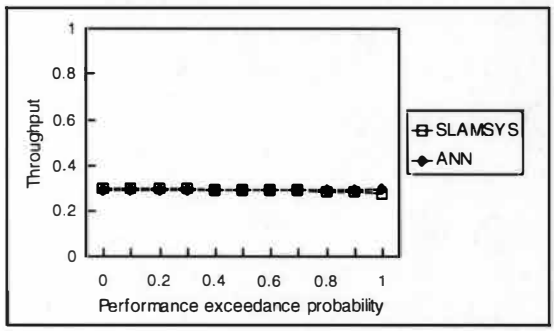


Figure 32. The Performance of ANN for 4 Machines and 1 Inspection Center.

P.E.P.	SLAMSYS	ANN
0	0.30360	0.34717
0.1	0.30206	0.33993
0.2	0.30051	0.33271
0.3	0.29897	0.32556
0.4	0.29503	0.31852
0.5	0.29331	0.31163
0.6	0.29160	0.30493
0.7	0.29160	0.29844
0.8	0.28782	0.29221
0.9	0.28285	0.28624
1	0.27959	0.28056

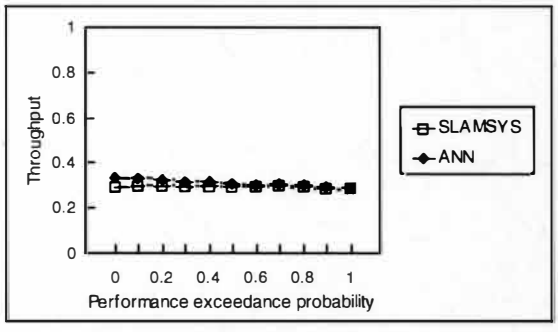


Figure 33. The Performance of ANN for 8 Machines and 1 Inspection Center.

P.E.P.	SLAMSYS	ANN
0	0.59863	0.65433
0.1	0.59691	0.65028
0.2	0.59348	0.64568
0.3	0.59177	0.64057
0.4	0.59074	0.63502
0.5	0.58319	0.62907
0.6	0.57907	0.62278
0.7	0.57633	0.61619
0.8	0.57427	0.60935
0.9	0.57273	0.60231
1	0.56261	0.59511

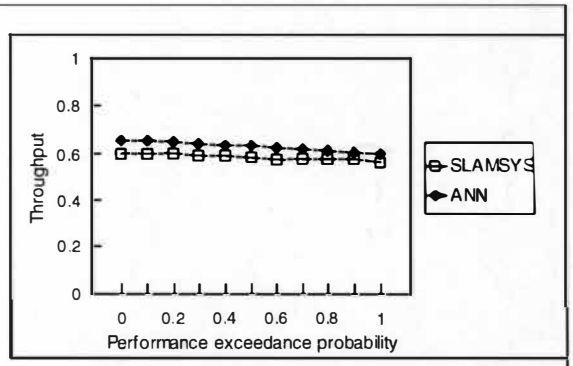


Figure 34. The Performance of ANN for 4 Machines and 2 Inspection Centers.

P.E.P.	SLAMSYS	ANN
0	0.57461	0.57421
0.1	0.57118	0.56873
0.2	0.56947	0.56294
0.3	0.56775	0.55692
0.4	0.56672	0.55074
0.5	0.56604	0.54449
0.6	0.56604	0.53823
0.7	0.56552	0.53203
0.8	0.56398	0.52594
0.9	0.55901	0.52002
1	0.55746	0.51431

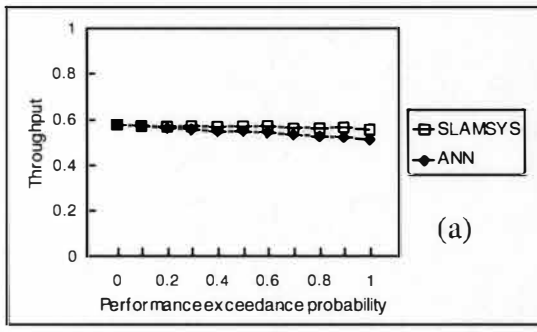


Figure 35. The Performance of ANN for 2 Machines and 3 Inspection Centers.

P.E.P.	SLAMSYS	ANN
0	0.85420	0.84011
0.1	0.83928	0.83228
0.2	0.81578	0.82372
0.3	0.81183	0.81443
0.4	0.80549	0.80441
0.5	0.79931	0.79368
0.6	0.79177	0.78224
0.7	0.77993	0.77011
0.8	0.77256	0.75732
0.9	0.76398	0.74391
1	0.71870	0.72992

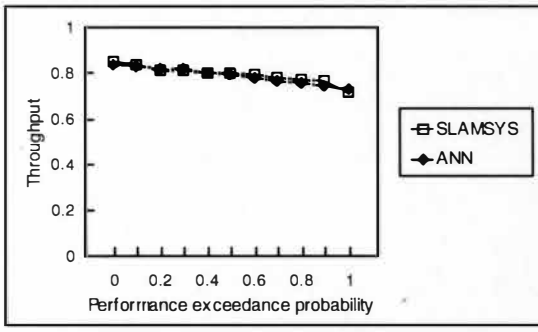


Figure 36. The Performance of ANN for 6 Machines and 3 Inspection Centers.

P.E.P.	SLAMSYS	ANN
0	0.28988	0.26142
0.1	0.28662	0.25785
0.2	0.28645	0.25466
0.3	0.28645	0.25189
0.4	0.28645	0.24957
0.5	0.28473	0.24774
0.6	0.28473	0.24642
0.7	0.28473	0.24562
0.8	0.28302	0.24537
0.9	0.28130	0.24565
1	0.28130	0.24649

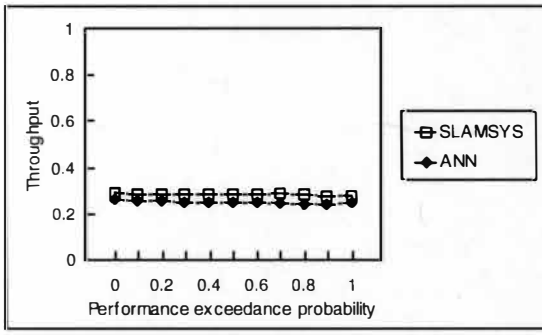


Figure 37. The Performance of ANN for 1 Machine and 4 Inspection Centers.

P.E.P.	SLAMSYS	ANN
0	0.85592	0.86530
0.1	0.84580	0.85605
0.2	0.84391	0.84604
0.3	0.81750	0.83525
0.4	0.80961	0.82367
0.5	0.79846	0.81129
0.6	0.78834	0.79810
0.7	0.78010	0.78412
0.8	0.74957	0.76935
0.9	0.74563	0.75383
1	0.72213	0.73759

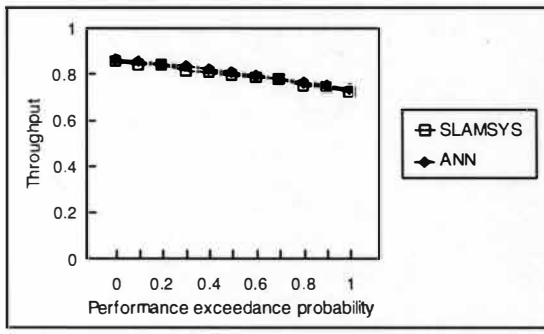


Figure 38. The Performance of ANN for 8 Machines and 5 Inspection Centers.

P.E.P.	SLAMSYS	ANN
0	0.87307	0.85783
0.1	0.85986	0.84741
0.2	0.84940	0.83612
0.3	0.83928	0.82394
0.4	0.82813	0.81085
0.5	0.81732	0.79686
0.6	0.79863	0.78197
0.7	0.79451	0.76622
0.8	0.78113	0.74963
0.9	0.76295	0.73227
1	0.75472	0.71421

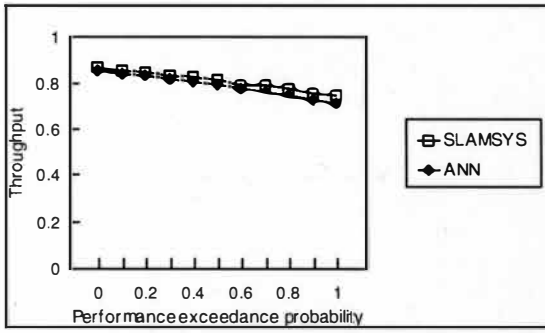


Figure 39. The Performance of ANN for 6 Machines and 8 Inspection Centers.

P.E.P.	SLAMSYS	ANN
0	0.87479	0.86325
0.1	0.85146	0.85588
0.2	0.83808	0.84789
0.3	0.81853	0.83927
0.4	0.80686	0.83000
0.5	0.80274	0.82005
0.6	0.79177	0.80942
0.7	0.78696	0.79808
0.8	0.77187	0.78605
0.9	0.75643	0.77330
1	0.75129	0.75987

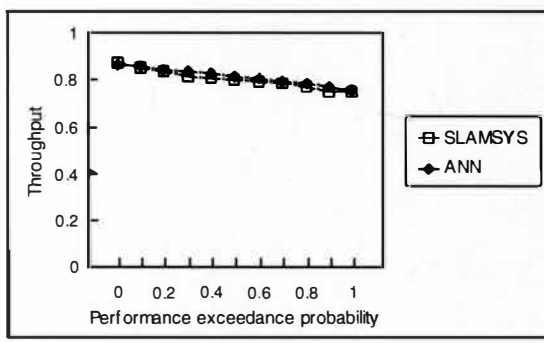


Figure 40. The Performance of ANN for 9 Machines and 1 Inspection Center.

Modular Approach

As mentioned before, one approach to simulate a manufacturing system is to consider that the system consists of several simple components and to try to find appropriate ANNs for these components. These networks can later be assembled together to estimate the behavior of the complex system.

In one experiment, the modular approach was applied to the illustrative manufacturing system (Figure 27). In this approach, the system was considered as two subdivisions which were connected together. As shown in Figure 41, two ANNs were trained to capture the behavior of each division.

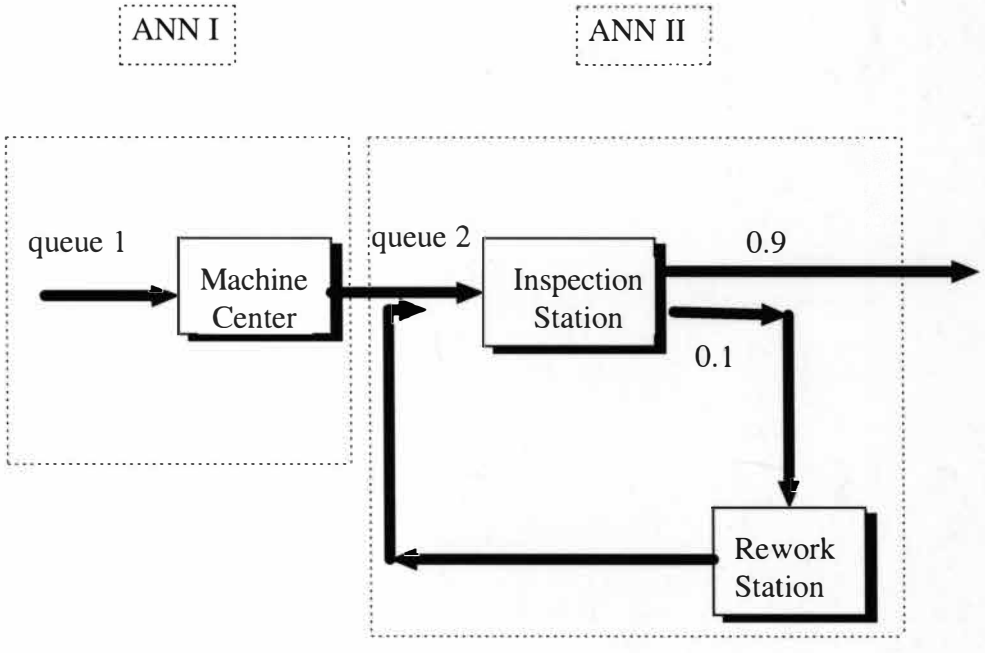


Figure 41. Modular Approach for Simulating the Manufacturing System.

The number of machine centers was considered as the input of the first network (ANN I). The output of this network was the mean of interdeparture time. This output was applied as an input (interarrival time) to another network (ANN II). The number of inspection stations was another input to this network. Finally, the output of ANN II was the throughput of the whole system. This structure is shown in Figure 42.

First, ANN I and ANN II were trained to learn the relationship between their input and output values. One more time, MLP network with 18 nodes in one hidden layer was used and the backpropagation momentum learning method was applied. The initial weights were again set to be random numbers in the interval of (-1, 1). The learning rate of 0.2 and momentum of 0.1 were selected. The gathered data were normalized to the interval of (0.1 and 0.9). The training was stopped only after 10,000 epochs.

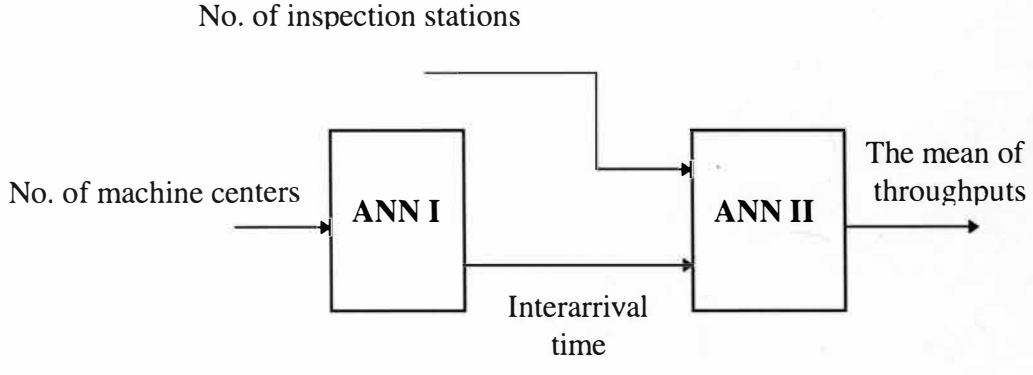


Figure 42. The Outline of Modular Approach for Simulating the Manufacturing System.

After training, the results from ANNs were compared to those of SLAMSYS. 39 points were compared and the results are shown in Table 9.

Use of the modules are usually involved in some assumptions which are not always true. For example, in this experiment, ANN II was trained based on exponential interarrival time inputs. The readers should notice that the output of ANN I is not necessarily exponentially distributed. More unreal assumptions will result in more inaccurate results.

Another assumption, which is very crucial in some systems, is that the networks are isolated. In most of manufacturing systems, the components have mutual effects on each other. This interaction is not usually considered when the networks are trained separately. As an example, in Figure 36, the queue 2 capacity is given to be infinity which makes the ANN I and ANN II work isolated from each other. If there was a queue with limited capacity, it might block the machine center. In this case, the machine center can not processes any further entities until a free space be available in the queue. Therefore, the ANN II can affect ANN I.

To investigate this phenomena, the capacity of queue 2 was limited to one entity at each instant of time. The number of inspection centers was set to be 3 and the number of machine centers was changed from 1 to 10. The results from modular approach and global approach are compared in Table 10.

In conclusion, there is a need for more investigation to make modular approach an appropriate method in simulating complex manufacturing systems.

Table 9

Comparison Between the Results Generated by
ANNs' Modules and SLAMSYSTEM

Two Sample t-Test

System	# No.	MEAN	STDEV	SE MEAN
SLAMSYSTEM	39	0.649	0.301	0.0482
Neural Network	39	0.657	0.311	0.0498

95% confidence interval for $M_1 - M_2 = (-0.1460, 0.1302)$

The Pair-wise Comparison (d1-d2)

	N	MEAN	MEDIAN	STDEV	MIN	MAX
d1-d2	39	0.0079	-0.0042	0.0732	-0.1758	0.1422

Histogram of (d1-d2)

Midpoint	Count	
-0.16	1	*
-0.12	1	*
-0.08	1	*
-0.04	12	**********
0.00	12	**********
0.04	2	**
0.08	3	***
0.12	6	******
0.16	1	*

Table 10

Comparison Between the Results Generated by Modular and Global Approach

No of Machines	Type	MEAN	STD	SE MEAN	95% Confidence Interval for M_1-M_2
1	Modular	171.7	11.6	3.68	(-3.287, 13.49)
	Global	166.6	1.43	0.452	
2	Modular	303.6	11.1	3.5	(-35.96, -19.84)
	Global	331.5	2.12	0.671	
3	Modular	379.3	10.1	3.2	(-93.40, -67.60)
	Global	459.8	16.2	5.13	
4	Modular	384	13.3	4.21	(-103.9, -70.34)
	Global	471.1	21	6.64	
5	Modular	384	13.3	4.21	(-99.61, -65.59)
	Global	466.6	21.4	6.77	
6	Modular	378.5	11.4	3.59	(-94.72, -71.68)
	Global	461.7	13	4.11	
7	Modular	386.1	9.04	2.86	(-76.29, -55.71)
	Global	452.1	12.4	3.92	
8	Modular	384.3	16.1	5.09	(-103.6, -69.41)
	Global	470.8	19.9	6.3	
9	Modular	387.6	7.56	2.39	(-98.69, -66.51)
	Global	470.2	21.8	6.91	
10	Modular	384	13.3	4.21	(-85.63, -59.17)
	Global	456.4	14.7	4.64	

Dynamic Systems

As mentioned in Chapter III, a dynamic system can be considered as static at each instant of time. Flood (1996) has used this property to develop a static network which can model dynamic systems. According to the author's, the network can produce a series of output values, each corresponding to a successive point in time. The network would process the information of the system at time "t" to generate output defining the state of the system at a slightly later point in time, "t+1". A loop would feed this information back to the input and the entire process would be repeated. Figure 38 shows this approach. This procedure will continue until the final point is reached. Since each point depends on previous points, and the procedure involves using random values, it is recommended to run this system several times to get more accurate results. Uncertainty in the system can be captured by including a random value as an input in each iteration.

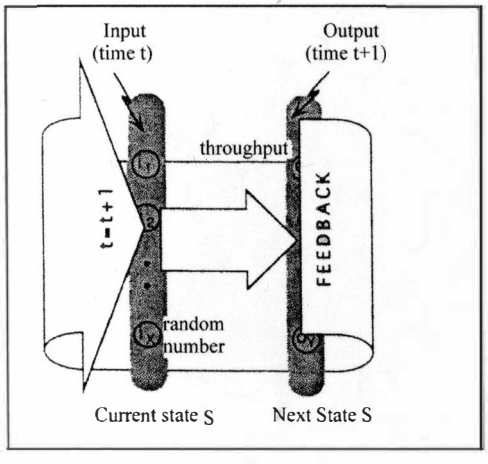


Figure 43. Capturing Dynamic Behavior of a System Through Static ANNs.

CHAPTER VII

CONCLUSIONS AND FURTHER STUDY

This document surveyed the prospective applications of Artificial Neural Networks (ANNs) in interactive simulation. ANNs have proven to be a promising technique in this field. They can be used in two main categories of applications: (1) situations where there is a need for quick response to a new set of data, and (2) situations where the effects of factors involved in the system are poorly understood. Literature survey and experimentation show the main advantages of ANNs over conventional simulation as follows:

1. They can learn from example (experience).
2. They do not need any particular assumption about the data (e.g. normality).
3. Fewer assumption and less precise information about the system is necessary.
4. They do rapid and parallel processing.
5. They can first be developed "off-line", to be used "on line".
6. They can re-tune themselves within changing environments.
7. They are robust to noise and missing data.

However, the current lack of knowledge and guidelines for implementing ANNs in practical problems create a gap between the capabilities of this technology

and its application to modeling manufacturing systems. To bridge this gap, publications have been reviewed and some practical guidelines have been offered in this survey. Due to ambiguity associated with ANNs, it is difficult to have clear guidelines. Many decisions should be made based on previous experiences and some trial and error experimentation. Throughout this research many limitations and pitfalls of ANNs have been realized. Among these:

1. They do not always learn a satisfactory solution to a problem.

2. It is not always easy to find a good architecture for an ANN.

3. Due to ambiguities associated with the weights and their meaning corresponding to the real world system, they may not be helpful for understanding and interpreting the components of the real system.

With the help of recommended guidelines, M/M/S queuing system was modeled by an ANN. The procedure of modeling M/M/S is the typical procedure of modeling a static manufacturing system. Based on static modeling, some methods have been offered to capture the stochastic and dynamic behavior of manufacturing systems. A simple manufacturing system was modeled through three different ways. The manufacturing system was also modeled through modular approach. In each case, the results were criticized and compared to conventional simulation methodology.

Although some research has been done in the field of ANNs and their applications in manufacturing systems, there are still many areas which are unclear. Based on our attempt and other studies in this field, these areas should be studied further:

1. Research should be conducted in the field of ANNs. Although MLPs have many capabilities, they suffer from many limitations (see Chapter III). For example, some networks used in modeling the M/M/S queuing system took days to be trained. Therefore, alternative types of ANNs and faster training procedures are among the areas that should be investigated.

2. The applications of new technologies such as Artificial Intelligence, Fuzzy Logic and Genetic Algorithms as complementary tools of ANNs, should be investigated. These sciences have been proved to have potential usage in the ANNs. For example, Genetic Algorithm can be used to optimize the number of layers and neurons. The modelers should use these sciences to simplify the recommended guidelines.

3. The performance of other ANN types such as recursive networks should be investigated. Most literature and research in the field deal with static modeling. It is suggested to investigate the recursive networks' capabilities especially in modeling the dynamic and stochastic manufacturing systems.

4. The library of manufacturing modules should be enriched. As a first step of creating a library of manufacturing systems, M/M/S queuing system was modeled in this research. General manufacturing components should be modeled and assembled together to estimate complex manufacturing systems. According to this project, the modular approach suffers from lack of precision because of the interaction between modules. More research is needed in this field.

5. Several different real manufacturing systems should be modeled through ANNs. Most studies in the field have been done with computer generated numbers rather than real data. One of the capabilities of ANNs is to capture unknown factors in the system. Many of these factors are simply neglected when the system is modeled by a conventional simulation software. The performance of ANNs should be compared to that of conventional simulation software on data collected from real manufacturing sites.

6. The application of ANNs in optimization should be further studied. Through this research (see Chapter VI), it has been realized that the SSE of ANNs are usually dropped after a few epochs. ANNs can quickly realize the direction of minimum error. This phenomena persuades the author to apply ANNs in the optimization. Investigation is needed to find appropriate procedures toward this goal.

7. The offered guidelines should be enriched and updated. This document has tried to provide industrial engineers with some recommendations and guidelines to help them simulate their systems through ANNs. However, implementing many steps of these guidelines depend on the previous experiences of the modeler and trial and error experimentation. These recommendations may not be attractive for those people who are looking for explicit formulas and/or clear cut guidelines. Neuroscience is not mature enough yet to support these guidelines with closed-form formulas. Questions such as "What is the best topology for the network?", "How many layers and nodes are needed?" are still open. Thus, the modelers who are interested in applying ANNs to manufacturing systems should get involved in neuroscience and update the

suggested guidelines based on forthcoming innovations in that field.

8. New methods should be developed to use static ANNs in simulating stochastic processes. The ANNs mostly transform stochastic processes into deterministic models. Industrial engineers are usually interested in distribution of data rather than mean and variance or upper/lower confidence intervals of data. Thus, more approaches similar to performance exceeding probability should be developed.

Finally, scientists with extensive background in ANNs who think that the approximation of computer simulation is trivial, should pay attention to Kilmer's (1996) comments in this regard.

> The idea of using an ANN to approximate a computer simulation may initially seem routine to researchers with an extensive background in neural networks. The reason for such an assessment is that there are many examples of researchers using computer simulations in order to obtain data to train their networks. The majority of these cases involved research in modifying or developing new ANN methodologies, techniques, or procedures. Thus, instead of expending valuable time and effort to obtain data from a real system, these researchers obtained their data from computer simulations that were built with the sole purpose of "feeding" an ANN. However, while it might be fairly trivial to build a computer simulation to provide training data to an existing ANN, this does not mean that it will be easy to build an ANN that will be able to receive and learn the relationships of an existing, complex stochastic computer simulation.

Appendix A

Programs' Source Codes

/* Program for generating M/M/s inputs for an ANN*/

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
long int facto(int i)
 {
 if( i > 1)
 return double(i) * facto(i-1);
 else
        return 1;
 }

double summ(double rou, int s)
 {
 double sum=0.;
 int n;
         for (n=0 ; n <= s-1; n++)
          sum = sum + pow(rou,(double)n)/(double)facto(n);
 return sum;
 }

main(void)
{
        FILE *fq;
        int lamda,mue,i,Lamda[2000],Mue[2000],c=0,j,temp,S[2000],s;
        char ch;
        double P_nut[2000],L_q[2000],W[2000],rou,rou1,temp1,temp2,p;
        double L_q_max=0.,W_max=0.,L_q_temp=0.,W_temp=0.;
        fq = fopen("c:\sstest.pat","w+");
        randomize();

        for(j=1;j<1001;j++){
         /*  lamda = 1 + random(49);
                s = random(10) + 1;
                temp = (int) ( 50. - (float)lamda/(float)s);
                mue =lamda + random(temp)+1; */
                lamda = random(49)+1;
                mue = random (49)+1;
                s = random(10)+1;
                while( (p = (float)lamda/((float)mue*(float)s)) < .1 || p> .99){
                lamda = random(49)+1;
```

```
                mue = random(49)+1;
                s= random(10)+1;                        }
                rou = (float)lamda/(float)mue;
                rou1= rou/(float)s;
                Mue[c] = mue;
                Lamda[c] = lamda;
                S[c] = s;
                temp1=summ(rou,s)+pow(rou,(double)s)/((double)facto(s)*(1.-rou1));
                P_nut[c] = 1./temp1;
                temp2=(1.-rou1)*(1.-rou1)*(double)facto(s);
                L_q[c]= P_nut[c]*pow(rou,(double)s)*rou1/temp2 ;
                W[c] = L_q[c]/(float)lamda;
                c=c+1;
                                                }
                for (i=0;i<c;i++)
                {
                L_q_temp = L_q[i];
                W_temp = W[i];
                if ( L_q_max < L_q_temp)
                        L_q_max = L_q_temp ;
                if ( W_max < W_temp)
                        W_max = W_temp;
                }
                fprintf(fq,"SNNS pattern definition file V3.2\n");
                fprintf(fq,"generated at Sat Aug 19 13:35:27 1995\n\n\n");
                fprintf(fq,"No. of patterns : %d\n",c);
                fprintf(fq,"No. of input units : 2\n");
                fprintf(fq,"No. of output units : 1\n\n");
                fprintf(fq,"1.1 * W_max= %f 1.1 * L_q_max=%f
\n",W_max,L_q_max);

                for (i=0;i<c;i++)
                {
                fprintf(fq,"# Input pattern %d:\n",i+1);
                fprintf(fq,"%6.5f %6.5f \n
%6.5f",(float)S[i]/11.,(float)Lamda[i]/50.,(float)Mue[i]/50.);
                fprintf(fq,"# Output pattern %d:\n",i+1);
                fprintf(fq,"%6.5f %6.5f %6.5f
\n",P_nut[i],W[i]/(1.076596),L_q[i]/(51.222081));
                }
  fclose(fq);
   exit(0);
  }
```

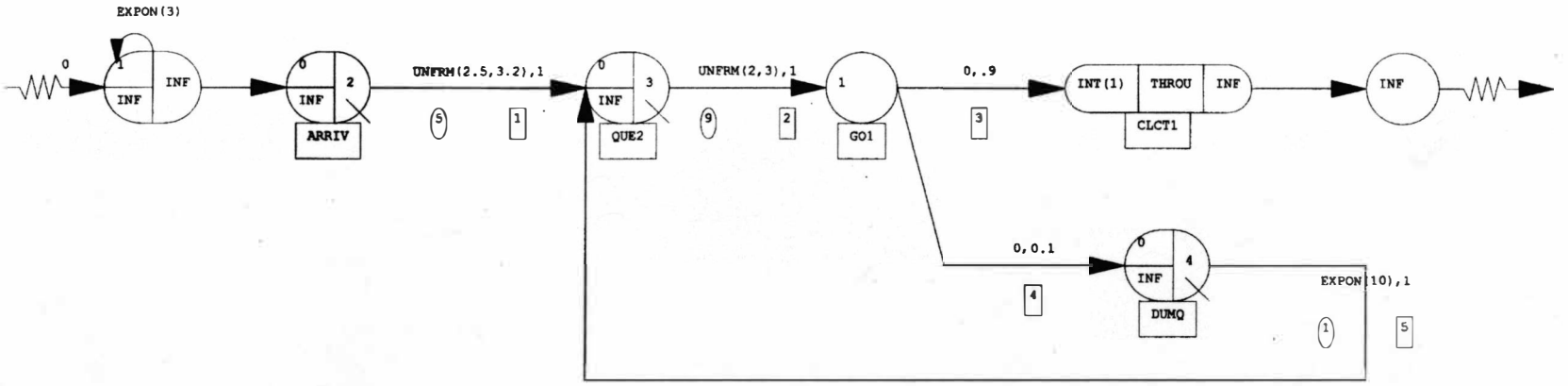
```
/* Program  for validation of M/M/S Queing System, By Payman Jula*/
main(void)
 {
    FILE  *fq;
    int lamda,mue,i,Lamda[2000],Mue[2000],c=0,j,temp,S[2000],s;
    char ch;
    double P_nut[2000],L_q[2000],W[2000],rou,rou1,temp1,temp2,p;
    double L_q_max=0.,W_max=0.,L_q_temp=0.,W_temp=0.;
          fq = fopen("c:\sstest.pat","w+");
          randomize();
         for (s=1; s<11; s=s+2) {
          for (lamda=1;lamda<50;lamda=lamda+3)     {
            for (mue=lamda/s + 1; mue<50;mue=mue+2){
                  rou = (float)lamda/(float)mue;
                  rou1= rou/(float)s;
                  Mue[c] = mue;
                  Lamda[c] = lamda;
                  S[c] = s;
                  temp1=summ(rou,s)+pow(rou,(double)s)/((double)facto(s)*(1.-rou1));
                  P_nut[c] = 1./temp1;
                  temp2=(1.-rou1)*(1.-rou1)*(double)facto(s);
                  L_q[c]= P_nut[c]*pow(rou,(double)s)*rou1/temp2 ;
                  W[c] = L_q[c]/(float)lamda;
                  c=c+1;                    } }       }
                  for (i=0;i<c;i++)
                  {       L_q_temp = L_q[i];
                          W_temp = W[i];
                          if ( L_q_max < L_q_temp)
                                  L_q_max = L_q_temp ;
                          if ( W_max < W_temp)
                                  W_max = W_temp; \
                  }
                  fprintf(fq,"SNNS pattern definition file V3.2\n");
                  fprintf(fq,"generated at Sat Aug 19 13:35:27 1995\n\n\n");
                  fprintf(fq,"No. of patterns : %d\n",c);
                  fprintf(fq,"No. of input units : 2\n");
                  fprintf(fq,"No. of output units : 1\n\n");
                  fprintf(fq,"1.1*W_max= %f 1.1*L_q_max=%f \n",1.1 * W_max,1.1 *
L_q_max);
                  for (i=0;i<c;i++)
                  {
                  fprintf(fq,"# Input pattern %d:\n",i+1);
```

```
                fprintf(fq,"%6.5f %6.5f %6.5f
\n",(float)S[i]/11.,(float)Lamda[i]/50.,(float)Mue[i]/50.);
                fprintf(fq,"# Output pattern %d:\n",i+1);
                fprintf(fq,"%6.5f %6.5f %6.5f \n",P_nut[i],W[i]/(1.1 *
W_max),L_q[i]/(1.1 * L_q_max));
                }

 fclose(fq);
  exit(0);
 }
```

Appendix B

SLAMSYSTEM's Network



A Simple Manufacturing System Modeled By SLAMSYSTEM.

BIBLIOGRAPHY

Anscombe, F.J. (1973). Graphs in statistical analysis. American Statistician, 27, 17-21.

Baldi, P. F., & Hornik, K. (1995). Learning in linear neural networks: a survey. IEEE Transactions on Neural Networks, 6(4), 837-858.

Bebis, G., & Georgiopoulos, M. (1995). Improving generalization by using genetic algorithms to determine the neural network size. Southcon/95 conference record, 392-397.

Berenji, H. R., & Khedkar, P. (Sept., 1992). Learning and tuning fuzzy logic controllers through reinforcements. IEEE Transactions on Neural Networks, 3(5), 724-740.

Blanning, W. R. (1975). Response to Michael, Kleijnen and Permut, Interface, 5, 24-5.

Chester, D. L. (1990). Why two hidden layers are better than one. Proceedings of the International Joint Conference on Neural Networks, 1, 265-268.

Chung, Y. & Kusiak, A. (1994). Grouping parts with a neural network. Journal of Manufacturing Systems, 13(4), 262-275.

Emshoff, J. P., & Sisson, R. L. (1970). Design and Use of Computer Simulation Models. Macmillan, London.

Fishwick, P. A. (1989). Neural network models in simulation: a comparison with traditional modeling approaches. Proceedings of the 1989 Winter Simulation Conference, 702-710.

Flood, I. (1991). A Gaussian-based feed forward network architecture and complementary training algorithm. Proceedings of 1991 IEEE International Joint Conference on Neural Networks, 1, 171-176.

Flood, I., & Christophilos, P. (1996). Modeling construction processes using artificial neural networks. Automation in Construction, 4(4), 307-320.

Flood, I., & Worley, K. (1994). Simulation using artificial neural networks. Proceedings of the 1994 Summer Computer Simulation Conference (26th, San Diego), 217-222.

Flood, I., & Worley, K. (1995). An artificial neural network approach to discrete-event simulation. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 9(1), 37-49.

Foo, S. Y., & Takefuji, Y., & Szu, H. (May, 1995). Scaling properties of neural networks for job-shop scheduling. Neurocomputing, 8(1), 79-91.

Friedman, L. W. (1989). The multivariate metamodel in queuing system simulation. Computers and Industrial Engineering, 16(2), 329-337.

Friedman, L. W., & Pressman. I. (1988). The metamodel in simulation analysis: can it be trusted? Journal of Operational Research Society, 39(10), 939-948.

Hashem, S., & Schmeiser, B. (1994). Improving model accuracy using optimal linear combinations of trained neural networks. Proceedings of World Congress on Neural Networks, 3, 4.

Hillier, F. S., & Lieberman, G. J. (1995). Introduction to Operations Research, Sixth edition, McGraw Hill.

Hopfield, J. J. (May, 1984). Neurons with a graded response have collective computational properties like those of two-state neurons. Proceedings of the National Academy of Science, 81.

Hornik, K. (1989). Multilayer feedforward Networks are universal approximators. Neural Networks, 2.

Hornik, K. (1994). Neural networks: more than statistics for amateurs? Proceedings of 11th Symposium on Computational Statistics. 223-35.

Huang, S. C., & Huang, Y. F. (1991). Bounds on the number of hidden neurons in the multilayer perceptrons. IEEE Transactions on Neural Networks, 2(1), 47-55.

Hurrion, R. D. (1992). Using a neural network to enhance the decision making quality of a visual interactive simulation model. Journal of the Operational Research Society, 43(4), 333-341.

Hurrion, R. D. (1993). Representing and learning distribution with the aid of a neural network. Journal of Operational Research Society, 44(10), 1013-1023.

Hush, D. R., & Horne, B. G. (Jan., 1993). Progress in supervised neural networks. IEEE Signal Processing Magazine, 10(1), 8-39.

Hush, D. R., & Salas, J. M., & Horne, B. (1992). Error surfaces for multi-layer perceptrons. IEEE Transactions on Systems, Man and Cybernetics, 22(5), 1152-1161.

Kilmer, R. A. (1996). Applications of artificial neural networks to combat simulations. Mathematical and computer modeling, 23, No 1-2, PP 91-9.

Kilmer, R. A., & Smith, A. E. (1993).Using artificial neural networks to approximate a discrete event stochastic simulation model. Intelligent Engineering Systems Through Artificial Neural Networks, 3, ASME Press, 631-636.

Kilmer, R. A., & Smith A. E. (1994). Neural Networks as a metamodeling for discrete event stochastic simulation. Intelligent Engineering Systems Through Artificial Neural Networks, 4, ASME Press, 1141-1146.

Kleijnen, J. P. C. (1992). Regression metamodels for simulation with common random numbers: comparison of validation tests and confidence intervals. Management Science, 38(8), 1164-1185.

Kopsco, D., & Pipino, L., & Rybolt, W. (Nov., 1993). Neural networks as adjuncts to statistics software. Collegiate Microcomputer, 11(4), 229-239.

Lampinen, J., & Taipale, O. (1994). Optimization and simulation of quality properties in paper machine with neural networks. IEEE International Conference on Neural Networks, June 27-29, 1994, Orlando, Florida, Vol. 6, 3812-3815.

Law, A. M., & Kelton, W. D. (1991). Simulation Modeling and analysis. 2nd edition, McGraw-Hill, New York.

Law, A. M., & McComas, M. G. (July, 1992). How to select simulation software for manufacturing applications. Industrial Engineering, 24(7), 29-35.

Lippmann, R. P. (April, 1987). An introduction to computation with neural nets. IEEE ASSP Magazine.

Montgomery D. C. (1991). Design and Analysis of Experiments. Third edition. John Wiley & Sons.

Narendra, K. S., & Parthasarathy, K. (1991).Gradient methods for the optimization of dynamical systems containing neural networks. IEEE Transactions on Neural Networks, 2(2), 252-262.

Nuila, V. H., & Houshyar, A. (1993). Manufacturing Systems Simulation Manual. Whirlpool Corp., Benton harbor, Michigan.

Padgett M. L., & Roppel T. A. (1992). Neural network and simulation: modeling for applications. Simulation , 58(5), 295-305.

Pierreval H. & Huntsinger R. C. (1992). An investigation on neural capabilities as simulation metamodels. Proceedings of the 1992 Summer Computer Simulation Conference (July 27-30), Reno, Nevada, 413-417.

Pritsker, A. A. (1986). Introduction to Simulation and Slam II. 3rd edition, John Wiley & Sons Press, New York.

Reed, R. (Sept., 1993). Pruning algorithms - a survey. IEEE Transactions on Neural Networks, 4(5), 740-747.

Rumelhart, D. E. (1986). Learning internal representations by error propagation. Explorations in the Microstructure of Cognition, Vol. 1: Foundations. MIT Press.

Sarne, G. M. L., & Postorino, M. N. (May, 1994). Application of neural networks for the simulation of traffic flows in a real transportation network. Proceedings of the International Conference on Artificial Neural Networks, Sorrento, Italy, May 26-29, Vol. 2, 831-834.

SFINX, Structure and Function In Neural ConneXtions, University of California at Los Angeles

Shannon, R. E., & Biles, W. E. (1970). The utility of certain curriculum topics to operations research practitioners. Operations Research, 18, 1011-1025.

Sim, S. K., & Yeo, K. T., & Lee, W. H. (Aug., 1994). An expert neural network system for dynamic job shop scheduling. International Journal of Production Research, 32(8), 1759-1773.

Skrzypek, J. (1994). Neural Networks Simulation Environments. Kluwer Academic Publishers.

Smith, M. (1996). Neural Network for Statistical Modeling. International Thomson Computer Press.

SNNS, Stuttgart Neural Network Simulator, User Manual, Version 4.0, University of Stuttgart.

Wasserman, P. D. (1993). Advanced methods in neural computing. Van Nostrand Reinhold Press, New York.

Werbos, P. J. (1992). Neural networks, system and control in the chemical process industries. Hand book of intelligent control. Van Nostrand Reinhold, New York.

Widrow, B., & Lehr, M. A. (Sept., 1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. Proceedings of the IEEE, 78(9), 1415-42.

Widrow, B., & Rumelhart, D. E., & Lehr, M. A. (March, 1994). Neural networks: applications in Industry, Business and Science. Communications of the ACM, 37(3), 93-105.

Wildberger, A. M. (Aug., 1989). Application of expert systems, simulation and neural networks combined to enhance power plant performance. Proceedings of the AI and Simulation Workshop, AAAI.

Wildberger, A. M., & Hickok, K. A.(1992). Power plant modeling and simulation using artificial intelligence and neural networks. Progress in Simulation, 1, 100-125.

Yu, B. & Popplewell, K. (1994). Metamodels in manufacturing: a review, International Journal of Production Research, 32(4), 787-796.