10
I29A
599

UILU-ENG-95-2002

CIVIL ENGINEERING STUDIES

STRUCTURAL RESEARCH SERIES NO. 599

ISSN: 0069-4274

NEURAL NETWORK-BASED MATERIAL MODELING

By
XIPING WU
and
JAMSHID GHABOUSSI

DEPARTMENT OF CIVIL ENGINEERING
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN
URBANA, ILLINOIS
April 1995

NEURAL NETWORK-BASED MATERIAL MODELING

BY

XIPING WU
JAMSHID GHABOUSSI

Department of Civil Engineering
University of Illinois at Urbana–Champaign

Urbana, Illinois
April 1995

ABSTRACT

A Neural network–based material modeling methodology for engineering materials is developed in this study. With this approach, the complex stress–strain behavior of an engineering material can be captured within the weight structure of a multilayer feedforward neural network trained directly on the stress–strain data obtained from experiments. The feasibility of this approach is verified through constructing neural network–based constitutive models of plain concrete in biaxial stress states and in uniaxial cyclic compression. A composite material model simulating the stress–strain behavior of reinforced concrete as a generic composite material in a biaxial stress state is built with experimental data from Vecchio and Collins' tests on reinforced concrete panels in both pure shear and combined shear with normal stresses.

An adaptive neural network simulator is developed by implementing a dynamic node creation scheme and a higher order learning algorithm. Representation schemes, network architectures, training and testing methods, stress– and strain–based approaches for material modeling are investigated. An elastic unloading mechanism is studied with a concrete material model in biaxial compression. Main issues concerning the implementation of neural network material models in finite element solution procedures are discussed. The results on the stress–strain relations of a material predicted by a neural network–based model are compared with experimental data. All neural network material models developed in this study match well with experimental results and the network testing results are reasonable. The developed approach shows promise in the constitutive modeling of composite materials.

ACKNOWLEDGEMENTS

This report is based on the dissertation of Dr. Xiping Wu, submitted to the Graduate College of the University of Illinois at Urbana–Champaign in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Civil Engineering, in July 1991. The research was conducted under the guidance and supervision of Professor Jamshid Ghaboussi.

Financial support was provided by the Department of Civil Engineering and the Graduate College of the University of Illinois at Urbana–Champaign, as well as the US Army Construction Engineering Research Laboratory. Research assistantships received from the Department of Civil Engineering are gratefully acknowledged.

Numerical simulations and program development were conducted on the HP/Apollo Workstation Laboratory of the Department of Civil Engineering, and on a Convex computer operated by the Computing Services Office of the University of Illinois. The use of these facilities is greatly appreciated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Modern research in material modeling, which addresses complex behaviors such as ductile yielding, micro–cracking, brittle fracture, localization and softening, aims to construct *mathematical models* to describe the relationship between stresses and strains and possibly include some non-local effects. These models consist of mathematical rules and expressions that capture these varied and complex behaviors. From the time of Hooke to the present, these material models for various behaviors have been developed more or less in the same way: 1) a material is tested and its behavior observed; 2) a mathematical model is postulated to explain the observed behavior and material parameters are determined; 3) the mathematical model is used to predict yet untested stress paths and checked against results from additional existing or new experiments; and 4) the mathematical model is then modified to account for behaviors observed but unexplained by the model.

The idealization or the mathematical formulation of conceptual material models plays an essential role in the material modeling process. Depending on the nature of the problem and the availability of computing power, mathematical models of materials with various levels of sophistication can be contemplated. As the requirement for accuracy in predicting the behaviors of materials becomes more stringent, the material model needs to be more sophisticated. This need for more accuracy almost always causes increased complexity in material parameter determination and computations.

This phenomenon is well demonstrated in the material modeling process of composites, such as reinforced concrete. In the current plasticity-based or plasticity–fracture mechanics-based reinforced concrete models, more complex behaviors such as working hardening, strain softening, etc., are being accounted for. However, more material parameters need to be determined and the computation process becomes very expensive. In addition to increased expense of computation, the complexity in some material models not only makes them difficult to implement but also vulnerable to the violation of fundamental mechanics principles. For example, a valid material model should satisfy the principles of thermodynamics, the requirements for symmetry and frame indifference.

The modeling of material behavior is of vital importance in structural analysis and design. With advancing development in computing technology and in computer-based numerical methods such as the finite element, finite difference, and boundary element methods, it is possible to carry out detailed stress analysis of very large and complex structures, such as offshore platforms and cooling towers. Nevertheless, it has been realized that the validity and reliability of structural behaviors predicted from computer-based numerical analyses are mainly determined by the appropriateness of material models utilized within those numerical procedures. As a consequence, structural behavior predicted using an inappropriate material model can be of limited validity or usage.

In this century, reinforced concrete, as a composite material, has become one of the most important building materials in civil and structural engineering, and extensive research has been conducted on the modeling of its behaviors accordingly. However, even at present, it is highly difficult to develop analytical models that capture the full spectrum of its complex behavior, due to the highly nonlinear behavior of the constituent materials, concrete and reinforcing steel, and their interaction.

Research interest in neural networks, as a paradigm of computational knowledge representation, has experienced considerable increase in recent years. This new interest is supported by the realization that neural computing is inherently parallel and functionally more close to the operation of the brain; that is, it has the capability of self–organization or learning. With the advance and sophistication in some branches of neural networks, the technology has been successfully tailored for a wide range of applications, such as the modeling of some cognitive processes, vision, image processing, pattern recognition, and some engineering fields, as illustrated by the large range of subjects covered in papers appearing in conferences on neural networks (IEEE: Proceedings 1987 and 1988; IJCNN: Proceedings 1989, 1990a and 1990b; NIPS: Proceedings 1988 and 1989). It is obvious that with the continuous development on the computational theory and hardware implementation of neural networks, this technology will potentially provide an efficient and viable tool for solving certain engineering problems that are difficult for mere conventional approaches.

Research in the application of neural networks to problems in computational mechanics is very recent (Ghaboussi, et al., 1990, 1991). For material modeling, which is essentially a mapping problem with nonlinear functions, the computational characteristics of neural networks has facilitated the development of a decidedly different approach to the derivation and representation of material behavior. With their intrinsic property of self-organization and learning, and their utilization of numerical solution methods, neural networks, as a computational tool, are of potential for building constitutive models of engineering materials in computational mechanics.

1.2 Objectives and Scope

The aim of this study was to develop a neural network-based material modeling methodology for engineering materials, and to verify the approach through modeling constitutive stress-strain relations of plain concrete and reinforced concrete in different stress states. With available experimental data, neural network-based material models were developed specifically for modeling the stress-strain behavior of concrete in biaxial stress states and in uniaxial cyclic compressive stress state, and that of reinforced concrete in biaxial stress states.

The detailed objectives and scope of this research were as follows:

- Develop an efficient and flexible modeling environment for multilayer feed-forward neural networks by incorporating a variant of the adaptive hidden node generation scheme (Ash, 1989) and higher order learning algorithms, such as Quickprop (Falman, 1989), within the framework of an error backpropagating neural network.

- Introduce a neural network-based material modeling methodology in computational mechanics for engineering materials such as concrete and reinforced concrete. Investigate technical aspects involved in the development of neural network-based material models, specifically representation schemes, network architectures, training and testing, using stress- vs. strain-based modeling approaches, training data selection, and error determination.

- Develop neural network-based material models representing the stress-strain behaviors of concrete in biaxial stress states and in uniaxial cyclic compressive stress state to show that the proposed concepts and approach are feasible and efficient for material modeling, and investigate the incorporation of unloading mechanisms with a neural network-based concrete model in biaxial compression.

- Develop neural network-based material models for reinforced concrete by modeling it as a generic composite material in biaxial stress states, using testing data on reinforced concrete panels (Vecchio and Collins, 1982). The objective is to investigate the applicability of the proposed approach for the modeling of composite materials. The emphasis of this work is placed on the determination of a quasi-minimal training data set, generalization assessment, and performance study of different representation schemes.
- Discuss the framework and issues associated with the integration of neural network-based material models within Finite Element Methods.

1.3 Organization

In chapter 2, neural networks in general, multilayer feedforward neural networks in particular, and their salient computational properties are briefly discussed. The architecture adaptation process and some higher order learning algorithms are described in detail. Then, an adaptive modeling environment for multilayer feedforward neural networks is presented.

Chapter 3 gives a brief overview of the mathematical approach to material modeling, illustrated for the modeling of concrete, and introduces the concept of a neural network-based material modeling methodology for engineering materials.

Based on the material modeling methodology described in Chapter 3, neural network-based stress- and strain-controlled models of plain concrete in biaxial stress states and in uniaxial cyclic compressive stress state are constructed in Chapter 4. The incorporation of an unloading mechanism for a stress-controlled concrete model in biaxial compression is also investigated. The validity of a neural network-based model is verified through studying the generalization capability of a trained network by comparing its predictions on the stress-strain relation of concrete under certain stress combinations with results from experiments conducted by other researchers.

In chapter 5, the proposed approach is explored for modeling the stress-strain behavior of composite materials, by considering reinforced concrete as a generic composite material, and then modeling the behavior of reinforced concrete panels in biaxial stress states. Experimental data are drawn from Vecchio and Collins, of reinforced concrete panels subjected to both in–plane shear and combined shear with normal stresses (1982). Both stress-controlled and strain-controlled models are developed in this study.

Neural network-based material models are of a network form, differing from analytical models explicitly expressed as mathematical formulae for use within a finite element procedure. Consequently, issues specially associated with the implementation of neural network-based material models in the finite element analysis are briefly addressed in Chapter 6.

Finally, a summary and major conclusions from this investigation, as well as recommendations for future research following this study are presented in Chapter 7.

1.4 Summary of Notations

a_i =	activation value of a node or a neuron;
[B], or **B** =	the strain−displacement transformation matrix;
$[C_t]$ =	material flexibility matrix in $\{d\varepsilon\} = [C_t]\{d\sigma\}$;
$[D_t]$ =	the constitutive or stress−strain matrix of the material;
$\{d\varepsilon\}$ =	vector of incremental strain;
$\{d\sigma\}$ =	vector of incremental stress;
f =	a binary threshold function for a McCulloch-Pitts neuron;
f'_c =	cylinder uniaxial compressive strength of concrete;
f'_{cr} =	tensile cracking stress of concrete;
f_l =	normal stress in the longitudinal direction;
f_{max} =	maximum stress ratio in a given stress cycle;
f_{min} =	minimum stress ratio in a given stress cycle;
f_n =	biaxial normal stresses;
f_t =	normal stress in the transverse direction;
f_{yl} =	yield stress of longitudinal steel;
f_{yt} =	yield stress of transverse steel;
F_i, F_i' =	activation function and its derivative, respectively;
$[k_t]$ =	tangential element stiffness matrix;
$\mathbf{K}_t$ =	tangential stiffness matrix of the structure;
M =	mass matrix of the structure;
N =	the total number of training cases;
N_j =	net input to a unit in a neural network;
o_j =	output from a unit in a neural network;
$\mathbf{P}_k$ =	vector of externally applied nodal loads;

S_j =	outgoing signal from a McCulloch-Pitts neuron;
t_j =	expected output in the output layer;
$\mathbf{U}, \Delta\mathbf{U}$ =	vectors of nodal displacements and their increments, respectively;
$\ddot{\mathbf{U}}_k$ =	vector of nodal accelerations;
V =	volume of the element;
$v, \Delta v$ =	shear stress and increment, respectively;
$w_{ij}, \Delta w_{ij}$ =	strength of connection between unit i and j and its increment;
x_j =	activity of a McCulloch-Pitts neuron;
δ_j =	gradient of the total error with respect to the net input at unit j;
$\varepsilon, \Delta\varepsilon$ =	strain and strain increment, respectively;
ε_0 =	concrete cylinder strain corresponding to f_c';
$\varepsilon_1, \varepsilon_2$ =	major principal strain and minor principal strain, respectively;
$\gamma, \Delta\gamma$ =	shear strain and shear strain increment, respectively;
η =	a learning constant called the "learning rate";
θ_j =	a threshold value in the McCulloch-Pitts neuron;
ϱ_l =	reinforcement ratio for longitudinal steel;
ϱ_t =	reinforcement ratio for transverse steel;
$\sigma, \Delta\sigma$ =	stress and stress increment, respectively;
σ_1, σ_2 =	major principal stress and minor principal stress, respectively.

CHAPTER 2

NEURAL NETWORKS

2.1 Introduction

2.1.1 Historical Perspective

Neural networks, also referred to as Connectionist models, and Parallel Distributed Processing (PDP), are computational models inspired by our understanding on the biological structure of neurons and the internal operation of the human brain. Research in neural networks was started in the 1940's when an endeavor in the search for means of constructing a brain–like computing machine was undertaken, and the mathematical foundation for this learning paradigm was essentially laid during that period. Since then, the advancement of this field has been dramatized by the landmark conceptualization of computational models of neurons, the maturation of concepts of associative memory and connectionism, and the breakthrough in the development of learning algorithms.

The first computational model of a neuron or a processing unit in a neural network, which is capable of threshold logical operation, was proposed by McCulloch and Pitts in the early 1940's (McCulloch and Pitts, 1943). In spite of the simplicity of this idealized learning model and the ignorance about the behavior of real neurons, these models were proved to be able to construct a general computing machine (Minsky, 1967). Though studies in psychology and cognitive science fostered the concept of associative memory, it was Hebb who conceptualized the first learning rule, called the

'Hebb synapse' (Hebb, 1949), in which synaptic connections are modified according to the correlation between pre- and post-synaptic neuronal activities. The first computer simulation with Hebbian learning was performed by Farley and Clark (1954), without much success. Later, it was realized that the mechanism of inhibition should be an integral part in Hebbian learning (Rochester, et al., 1956). Using the McColluch-Pitts model of neurons, Rosenblatt built his two-layer learning system – Perceptrons, and developed the perceptron convergence procedure (1962) which was similar to the so called delta-rule or LMS (lest mean squared) procedure proposed by Windrow and Hoff for their Adaline model (1960) to systematically impart knowledge to the network. Research on neural networks attracted tremendous interests and showed promise in solving certain simple functional mapping problems during that era. However, due to architectural limitation and the lack of powerful learning scheme for general feedforward multi-layer system, Minsky and Papert (1969) pointed out, after rigorous mathematical analysis, that the perceptron was not able to solve some nonlinear separable problems such as the computation of parity function typified by the XOR function. This critical analysis alone almost halted research in neural networks in the 1970's. It was Hopfield's energy approach with associative memory (1982) that partially revived research activity in this field. Afterwards, the modern era of multilayer neural networks was ushered in by the introduction of backpropagation neural networks with the use of a sigmoidal activation function and the development of the *generalized delta rule* (Rumelhart, et al., 1986). With this new learning algorithm, the multilayer feedforward neural network has emerged as a powerful tool for solving a large range of problems in knowledge representation and functional modeling (Rumelhart, et al., 1986).

Presently, the computational capability of neural networks and their intrinsically parallel structure and operation have been well recognized. With new development in learning algorithms, the perspective on this computational model which is capable of learning to discover hidden relationships in data appears bright and encouraging. In addition to the recognition of the capability of neural networks and the development of computing technology, other factors have contributed to the recent explosion of interest in this area: 1) it is a universal approximator; an appropriate neural network with an appropriate training rule has the capability of solving virtually any computational task; 2) it takes a middle ground between traditional mathematical approach and symbolic artificial intelligence (AI) approach by using numerical methods for learning and expansive representation schemes, as well as adopting a functional use of experimental knowledge; 3) it provides an alternative with efficient performance in solving currently difficult problems with conventional approach such as speech and natural language processing, vision and image analysis, and pattern recognition with the recent insights into algorithms that improve the learning ability of a neural network; 4) it may provide some insight into the understanding of the computational characteristics of the brain; and 5) neural networks are compatible with massively parallel hardwares (Aleksander, 1989; Barto, 1989; Cybenko, 1989; and Hornik, et al., 1989).

2.1.2 General

A neural network is a nonlinear dynamic system consisting of a large number of highly inter-connected processing units, or processors. Each processing unit in the network maintains only one piece of dynamic information (its current level of activation) and is capable of only a few simple computations (adding inputs, computing a new activation level, or performing threshold logical calculation). A neural network performs "computations" by propagating changes in activation between the processors; it stores the knowledge it has "learned" as strengths of the connections between its processors. The large number of these processing units, and even larger number of inter—connections, similar to the neuronal structure of human brain, give the neural networks their capability of knowledge representation. In addition, it is through self-organization or "learning" that a neural network approaches some representation of a particular knowledge or discovers the hidden relationships in data.

Self-organization or "learning" is a key characteristic of neural networks. Unlike traditional sequential programming techniques, neural networks are trained with examples of the concepts to capture. The network then internally organizes itself to be able to reconstruct the presented examples. Several other interesting and valuable characteristics are: 1) their ability to produce correct, or nearly correct, responses when presented with partially incorrect or incomplete stimuli; and 2) their ability to generalize rules from the cases on which they are trained and apply these rules to new stimuli. Both of these latter characteristics stem from the fact that a neural network, through self–organization or learning, develops an internal set of features that it uses to classify the stimuli presented to it and returns the expected response.

The operation of a processor in a neural network computation is very simple. The output of a processor, which is computed from its activation level and many times is the same as the activation level, is sent to other "receiving" processors via the processor's outgoing connections. Each connection from one processor to another processor possesses a numeric weight representing the strength or weight of the connection. The strength of connection is a filter (in the form of a multiplicative coefficient) of the output sent from one processor to another processor, and may serve to increase, or decrease, the activation of the receiving processor. Each processor computes its activation level based on the sum of the products of connection strengths and outputs coming into the processor over its incoming connections, computes its output based on this net input, and then sends its output to other processors to which it has outgoing connections.

The propagation of activation in a neural network can be feedforward, feedback, or both. In a feedforward network, a type of signal can be propagated only in a designated direction, whereas in a network with feedback mechanism this type of signal can flow in either direction or recursively. For example, in a strictly feedforward multilayer network, only inter-layer connections between adjacent layers are allowed, and the intra-layer connections or lateral connections among nodes in the same layer are suppressed.

In a multilayer feedforward networks with certain type of learning rules, the amount of error defined as a measure of the difference between the computed output pattern and the expected output pattern is very much dependent on the weights of the connections between the processors. Hence, the "program", or definition of the computation, is embodied within the connection strengths of a neural network. However, the programming of a neural network does not involve manually setting the numeric values of the connection strengths, but rather, involves training the network with many examples of corresponding patterns of input and output and having it automatically modify the connections through the utilization of learning rules.

It is this ability to modify its own weights, i.e., to self–organize, that makes neural computing feasible, for it would be impossible to set the connection strengths manually for all but the simplest of neural networks with the simplest problems. In addition, self–organization leads to the observed neural network characteristics of robustness and the ability to generalize. By modifying the connection strengths between processors, neural networks can create internal features that: 1) might not be apparent from the data and thus would have defied the manual setting of connection strengths; and 2) can be used to produce correct, or nearly correct, patterns of output for patterns of input not encountered before, but having similar internal features to those input patterns previously encountered.

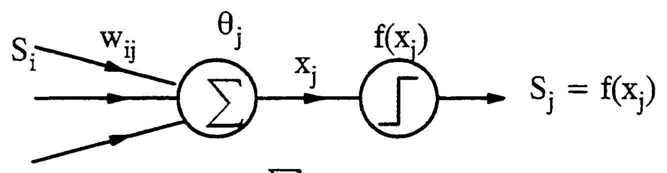
Rumelhart, et al. (1986), provided a description of the basic anatomy of neural networks, consisting of seven basic aspects: 1) a set of processing units, 2) the state of activation of a processing unit, 3) the function used to compute output of a processing unit, 4) the pattern of connectivity among the processing units, 5) the rule of activation propagation, 6) the activation function, and 7) the rule of learning employed. The network topology, and the form of the rules and functions are all learning variables in a neural network learning system and lead to a wide variety of network types. Some of the well known types of neural networks are: the Competitive Learning (Grossberg, 1976; Rumelhart and Zipser, 1985), the Boltzmann Machine (Hinton, et al., 1984), the Hopfield Network (Hopfield, 1982), the Kohonen network (Kohonen, 1984), the Adaptive Resonance Theory (ART) (Carpenter and Grossberg, 1987), and the backpropagation neural networks (Rumelhart, et al., 1986). The backpropagation neural network is given its name due to the way that it learns – by backpropagating the errors seen at the output nodes. Backpropagation networks and their variants, as a subset of multilayer feedforward networks, are currently the most widely used networks in applications. The following paragraphs describe the salient features of multilayer feedforward neural networks, with emphasis on backpropagation learning.

2.2 Multilayer Feedforward Neural Networks

2.2.1 The McCulloch-Pitts Model of Neuron

Though it was developed in 1943, the McCulloch-Pitts neuron still remains at the heart of most present day neural networks (Anderson and Rosenfeld, 1989). With simplification and idealization on the physiological structure and the computational characteristics of a biological neuron, McCulloch and Pitts put forth a simple computational model of a neuron as a binary threshold unit, as shown in Fig. 2.1.

In the McCulloch–Pitts model, the effect of incoming synapses that transmit incoming signal S_i to the neuron, is computed as a weighted sum of input activity from these neurons, where the weight w_{ij} represents the strength of the synapse connecting two neurons; and the outgoing signal or activation of the neuron, S_j, is calculated from a unit step function, $f(x_j)$, according to the value of the difference between the sum and a threshold value θ_j. If the difference, x_j, is larger than zero, then the activation value is 1; otherwise, the activation value is 0. The formulas for calculating the weighted sum and the threshold step function are shown in Fig. 2.1.



$$x_j = \sum_i S_i \, w_{ij} - \theta_j$$

$$f(x_j) = \begin{cases} 1, & \text{if } x_j \geq 0; \\ 0, & \text{otherwise.} \end{cases}$$

Fig. 2.1 – The McCulloch–Pitts Model of a Neuron

2.2.2 Backpropagation Neural Networks

Multilayer feedforward neural networks, developed from Perceptron which is composed of only two layers (Rosenblatt, 1958), are also referred to as multilayer perceptrons. The major distinction among feedforward neural networks is manifested by the learning rule utilized. The backpropagation network is a multilayer feedforward neural network with the generalized delta rule as its learning rule.

The processing units in a backpropagation neural network, which are similar to McCulloch-Pitts neurons with the exception that the activation function is a continuous sigmoidal instead of a threshold step function, are arranged in layers. Each neural network has an input layer, an output layer, and a number of hidden layers. Propagation takes place in a feed forward manner, from input layer to the output layer. The pattern of connectivity and the number of processing units in each layer may vary with some constraints. No communication is permitted between the processing units within a layer. The processing units in each layer may send their output to the processing units in higher layers.

Associated with each connection is a numerical value which is the strength or the weight of that connection: w_{ij} = strength of connection between units i and j. The connection strengths are modified during the training of the neural network. At the beginning of a training process, the connection strengths are assigned random values. As examples are presented during the training, through application of the "rule of learning", the connection strengths are modified in an iterative process. At the successful completion of the training, when the iterative process has converged, the collection of connection strengths of the whole network has captured and stored the knowledge and the information presented in the examples used in its training. Such a trained neural network is ready to be used. When presented with an input pattern, a feed forward network computation results in an output pattern which is the result of the generalization and synthesis of what it has learned and stored in its connection strengths.

Therefore, in a backpropagation network, two computational procedures are performed in a learning cycle: the feedforward computation of activations and the backward propagation of error signals for the modification of connection weights via the generalized delta rule. A feedforward computation proceeds as follows:

1) The units in the input layer receive their activations in the form of an input pattern and this initiates the feed forward process;

2) The processing units in each layer receive outputs from other units and perform the following computations:

a) Compute their net input N_j,

$$N_j = \sum_{k=1}^{M} w_{jk} \, o_k \qquad (2.1)$$

where o_k = output from units impinging on unit j, and M = number of units impinging on unit j.

b) Compute their activation values from their net input values,

$$a_j = F_j(N_j) \qquad (2.2)$$

where F_j is usually a sigmoid function and its exact form is determined by the specified range of activation values. For example, if the activation values are taken in the range of $(-1.0, 1.0)$, then $F(N) = 2.0\,(1/(1 + e^{-(N-\Theta)}) - 0.5)$, where Θ is the bias value at that processing unit.

c) Compute their outputs from their activation values. Usually, the output is taken the same as the activation value.

$$o_j = a_j \qquad (2.3)$$

3) The output values are sent to other processing units along the outgoing connections.

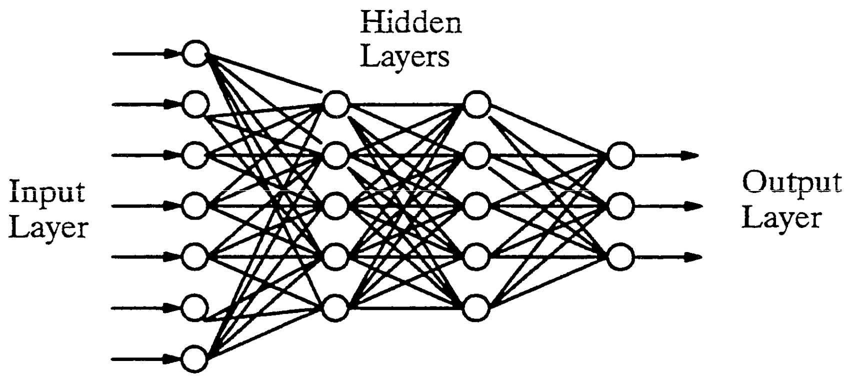


Fig. 2.2 — A Sample Backpropagation Neural Network

4) This process continues until the processing units in the output layer compute their activation values. These activation values are the output of the neural computations.

Several mechanisms for imparting self–organization or learning to these multilayer feedforward networks have been developed (Rumelhart, et al., 1986). In general, there are three classes of neural network learning procedures: supervised learning, reinforcement learning, and unsupervised learning (Hinton, 1989). Supervised learning means that the expected output is included in what the network is to learn; reinforcement learning only requires a single scalar evaluation of the output vector, giving information on whether each output is correct or incorrect; and the unsupervised learning means that the network is not told what it is to learn about the input with which it is presented and must, on its own, discover regularities and similarities among the input patterns. One form of supervised learning, developed by Rumelhart, et al. (1986), is called the *generalized delta rule* and is the learning mechanism used in backpropagation neural networks. Similar rules were also developed by some other researchers but they were less known to the public (Warbos, 1974; and Parker, 1982). All backpropagation neural networks, which use the generalized delta rule for self–organization and derive their name from the need to backpropagate error, have the same general architecture shown in Fig. 2.2.

The modification of the strengths of the connections in the generalized delta rule, described in (Rumelhart, et al., 1986), is accomplished through performing the gradient descent on the total error space in a given training case.

$$\Delta w_{ij} = \eta \, \nabla E(w_{ij}) = \eta \, \delta_j \, o_i \tag{2.4}$$

In this equation, η = a learning constant called the "learning rate", $\nabla E(w_{ij})$ = gradient of the total error with respect to the weight between units i and j, and δ_j = gradient of the total error with respect to the net input at unit j. At the output units δ_j is determined from the difference between the expected activations t_j and the computed activations a_j:

$$\delta_j = (t_j - a_j) \; F'(N_j) \tag{2.5}$$

where F' is the derivative of the activation function.

At the hidden units the expected activations are not known a priori. The following equation calculates δ_j for the hidden units:

$$\delta_j = \left(\sum_{k=1}^{M} \delta_k \, w_{jk}\right) F'(N_j) \tag{2.6}$$

In this equation, the error attributed to a hidden unit depends on the error of the units it influences. The amount of error from these units attributed to the hidden unit depends on the strength of connection from the hidden unit to those units; a hidden unit with a strong excitatory connection to a unit exhibiting error will be "blamed" for this error, causing this connection strength to be reduced.

2.2.3 Higher Order Learning and Adaptive Architecture Determination

As has been stated in the previous section, the generalized delta rule (Rumelhart, et al., 1986) is basically a steepest descent scheme with constant step length in a network setting, performing a gradient descent on the error function with respect to the weight space. For multilayer feedforward neural networks, the error function is usually a highly nonlinear function defined as:

$$E(\mathbf{w}) \equiv \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} E_k \tag{2.7}$$

where $E_k = | t(\mathbf{x}_k) - o(\mathbf{x}_k, \mathbf{w}) |^2$; $t(\mathbf{x}_k)$ is the expected output; $o(\mathbf{x}_k, \mathbf{w})$ is the network prediction which is a function of the input vector **x** and network weight vector **w**; and N is the number of training cases. This error surface is dominated with flat areas and troughs, which render the learning with the generalized delta rule in a backpropagation network very slow (Hecht-Nielsen, 1989). Another drawback of a standard back-propagation network is the need for pre–determination of network architecture and the inability to incorporate a priori possessed knowledge.

The modeling capability and performance of a backpropagation network is mainly determined by the network architecture and its rule of learning. Recently, several approaches have been proposed to improve the performance of backpropagation neural networks. In general, there are five ways to approach a solution to this problem: 1) using a better data representation scheme for input and output, 2) employing higher order learning algorithms or heuristic algorithms that more quickly find the minimum of the error surface (Becker and le Cun, 1988; Fahlman, 1988; Jacobs, 1987; and Moody, 1989), 3) preprocessing the input pattern, introducing independence into the input vector space (Orfanidis, 1990), thus facilitating the determination of the decision space, 4) designing innovative training schemes so that certain knowledge is pre-oriented in the network before the final training session (Fahlman and Lebiere, 1990; Tenorio and Lee, 1989), and 5) incorporating network geometry adaptation with efficient learning algorithms.

From the derivation of the generalized delta rule, it is tempting to postulate that all the minimization schemes are applicable as learning rules for multilayer feedforward networks. Furthermore, numerical analysis tells us that higher order schemes such as Newton's method, quasi-Newton methods, and Conjugate Gradient methods have better numerical properties than the steepest descent method with respect to the rate of convergence and numerical stability (Hageman and Young, 1981; Golub and VanLoan, 1983). Nevertheless, for neural network learning algorithms which are eventually to be employed in massively parallel hardware implementation of these networks, it is desirable that they not only be computationally efficient, but also suitable for implementation via local update only, thus conserving the parallelism of network operations. With the generalized delta rule, the formula for weight update with a momentum term is:

$$\Delta w(t) = -\eta \ \partial E/\partial w(t) + \alpha \ \Delta w(t-1) \tag{2.8}$$

where η is the learning rate and α the momentum factor, and both of them are assumed constants. The update of weights can be proceeded either in batch mode or in on–line mode. The former refers to updating weights after all the training sets have been presented, and the later after each training set. For second and higher order algorithms with adaptive determination of η or α, the update of weights is usually implemented in batch mode. To date, numerous new learning schemes have been proposed, such as the Quickprop algorithm (Falhman, 1988), the Delta-Bar-Delta algorithm (Jacobs, 1988), the Pseudo-Newton algorithm (Becker and le Cun, 1988), and quasi-Newton style methods (Watrous, 1987), etc., using either heuristic rules or higher order information to compute the learning parameters. Experience shows that heuristic rules are simple, robust, and computationally efficient, while the acquisition of higher order information is usually computationally expensive.

Except for some trivial problems, the network architecture on the hidden layers cannot be determined in advance. The common approach to architecture determination uses trial and error, for simple problems. For real world engineering problems such as material modeling, it is imperative to have adaptive or dynamic mechanisms to determine the network architecture. Since the input and output of a network are determined by the nature of the problem and the representation scheme selected, adaptive schemes for architecture determination have adopted mechanisms of either "growing" or "pruning" the number of processing units in hidden layers. A "growing" process starts with a basic or small network (usually one or a small number of hidden units), and then adds or grows additional processing units or a set of units including layer(s) to the network as the training process progresses until the convergence of training is reached. A "pruning" process usually starts with a larger network than needed, and then deletes redundant processing units or links during or after a training session with the hope that the generalization capability of the trained network would be improved. Sometimes, "pruning" is also performed on nodes in the input and output layers in order to determine the most important set of variables in the representation scheme. The former approach is represented in the Dynamic Node Creation scheme (Ash, 1989), the Cascade Correlation Learning Architecture (Fahlman and Lebiere, 1990), and the Self-Organizing Neural Network (Tenorio and Lee, 1989), and the latter in Skeletonization (Mozer and Smolensky, 1989), and Karnin's pruning scheme (1990).

In general, the "growing" approach is more efficient and robust than the "pruning" scheme for the determination of network architecture. For certain classification problems, pruning can be incorporated to improve network generalization. However, for real value functional mapping problems in which accuracy on predictions becomes more demanding, pruning might have an adverse effect.

On functional mapping, theoretical studies have proven that a multilayer feedforward network with one hidden layer and enough hidden nodes is a universal approximator, i.e., any function can be embedded in a three layer network (Cybenko, 1989; and Hornik, et al., 1989). This conclusion is valid in the limit sense of statistical measurement. However, for efficiency in learning, two or more hidden layers are usually used in applications (Lippmann, 1987).

In the following paragraphs, the Quickprop learning algorithm (Fahlman, 1988), the Delta-Bar-Delta algorithm (Jacobs, 1988), and the Dynamic Node Creation Scheme (1989) are described.

2.2.3.1 The Quickprop Learning Algorithm

The Quickprop algorithm was proposed by Falhman (1988), to improve the rate of convergence of learning in the backpropagation neural network through adaptive calculation of the momentum factor α in Eq. (2.8). It is a second order method in a sense, based loosely on Newton's method, but it is more heuristic than formal. The information required is the gradient of the error surface at the previous training epoch and that at the current epoch, along with the gradient difference between that of the previous and current epoches. The algorithm is derived based on the following assumptions: 1) the error vs. weight curve for each weight can be approximated by a parabola; 2) the change in the slope of the error curve, as seen by each weight, is independent of the other weights that are changing at the same time; and 3) the momentum factor plays a more important role than the learning rate. Therefore, the formula for the update of weights is:

$$\Delta w(t) = -\eta \; \partial E/\partial w(t) + \frac{\partial E/\partial w(t)}{\partial E/\partial w(t-1) - \partial E/\partial w(t)} \Delta w(t-1) \qquad (2.9)$$

Numerical experiments have shown that the algorithm has good learning convergence property and seems to scale up well for large training problems.

2.2.3.2 The Delta-Bar-Delta Algorithm

The Delta-Bar-Delta algorithm proposed by Jocobs (1987) for the adaptation of learning rate η in Eq. (2.8), is inspired by: 1) Kesten's observation (1958) on the performance of the steepest descent method that a weight value is oscillating if consecutive changes of the weight possess opposite signs, and 2) by Saridis' (1970) approach for learning rate adaptation – the learning rate is increased if consecutive derivatives of a weight possess the same sign, and decreased otherwise. Hence, the Delta-Bar-Delta algorithm consists of the following rules (Jacobs, 1987): 1) each weight has its own learning rate; 2) every learning rate can vary over time; 3) the learning rate is increased if the corresponding gradient has the same sign for several consecutive time steps; and 4) the learning rate is decreased if the corresponding gradient flips signs for several consecutive time steps. The algorithm can be expressed mathematically as follows:

$$\Delta\eta_{ij}(t) = \begin{cases} \eta & \text{if} \quad \bar{g}_{ij}(t-1)\; g_{ij}(t) > 0 \\ -\phi\;\eta_{ij}(t) & \text{if} \quad \bar{g}_{ij}(t-1)\; g_{ij}(t) < 0 \\ 0 & \text{otherwise} \end{cases} \qquad (2.10)$$

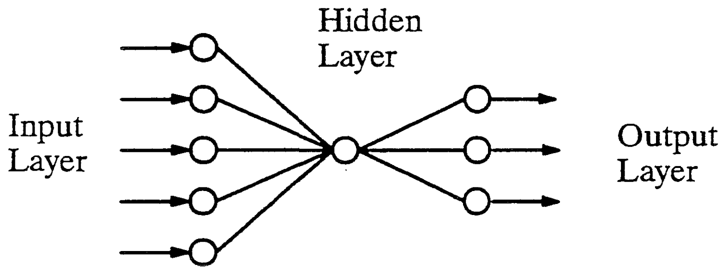
where $g_{ij}(t) = \nabla E(w_{ij}(t))$ and $\bar{g}(t) = (1-\theta)\, g(t) + \theta\, \bar{g}(t-1)$. The parameters of η, ϕ and θ are specified before use. Subsequently, the formula for weight update is:

$$\begin{cases} \Delta w_{ij}(t) = -\eta_{ij}(t)\ \partial E/\partial w_{ij}(t) + \alpha\ \Delta w_{ij}(t-1) \\ \eta_{ij}(t+1) = \eta_{ij}(t) + \Delta\eta_{ij}(t) \end{cases} \tag{2.11}$$

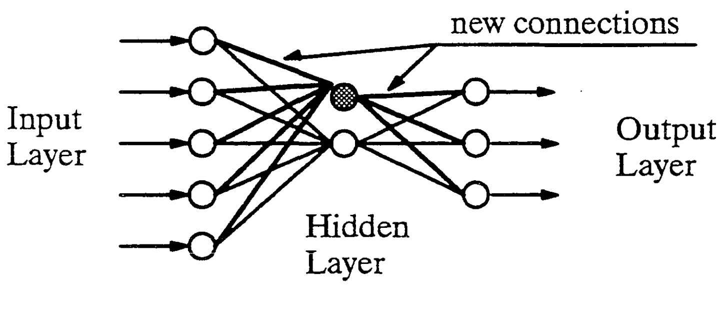
With the additional three learning parameters in the algorithm, numerical experiments have shown that performance of this algorithm depends heavily on the fine tuning of those parameters. On the other hand, the momentum factor α is not adaptively determined. Recently, a modification was proposed by Minai and Williams (1990) by introducing a mechanism for momentum factor adaptation and setting constraints on the learning parameters. However, the limited improvement on the robustness of the learning algorithm is overshadowed by the added complexity of additional five learning parameters.

2.2.3.3 The Dynamic Node Creation Scheme

Based on the theoretical conclusion that a three layer feedforward neural network is a universal approximator (Hornik, et al., 1989), Ash developed the Dynamic Node Creation scheme (1989) within backpropagation networks. This algorithm starts by fixing the network architecture to three layers with one hidden layer, starting training with one hidden node, and continually adding one hidden node at a time during a training period until the convergence of learning is realized. When a hidden node is added to the hidden layer, connections of this node to all the other input and output nodes are created, and the connection weights initialized. The decision to add a new hidden node is governed by whether the currently estimated average error slope over a



(a) The initial Architecture of a Neural Network



(b) The Current Architecture after Adding One Hidden Node

Fig. 2.3 – The Dynamic Node Creation Process

certain number of epoches is less than a pre-defined gradient tolerance, called "the trigger slope." Though this scheme appears to be robust and slightly faster than standard backpropagation algorithm on training some simple benchmark problems, the proper selection of the trigger slope tends to be problem dependent. However, this scheme is simple to use and usually results in an quasi-optimal architecture. The word "quasi" is used here because the convergence of learning depends on many factors such as the condition of the randomly generated initial weight matrix, and thus an optimal architecture may not be obtained through only a few learning sessions. However, because the network architecture grows from a small one, it would be reasonable to consider the resulted final network to fall into the neighborhood of the optimal architecture. The dynamic hidden node creation process is schematically shown in Fig. 2.3.

In the following section, the learning algorithm, architecture determination scheme, data structure, the framework of implementation, and a performance evaluation study of the simulator developed for use in studying the neural network–based material models are described.

2.3 Development of A Dynamic Neural Network Simulator

2.3.1 Learning Procedures of the Model

Analysis of the two most important aspects in neural network modeling, namely, architecture and learning process determination, indicates that an efficient modeling environment would be one consisting of both a dynamic architecture generation scheme and a higher order, or heuristic-based, fast learning algorithm that adapts its learning parameters. In the material modeling domain, the amount of information or the number of stress–strain relations corresponding to a certain number of stress paths that need to be trained is not known a priori, rather that it has to be determined during the training and testing session. This factor alone makes the ability to dynamically adjust the network architecture extremely important, for the previously trained network may have to be augmented in order to accommodate for additional training data. On the other hand, the highly nonlinear behavior of materials represented in the stress-strain relations and the large amount of experimental data used for training require a robust and efficient learning algorithm.

The simulator developed for building neural network-based material models utilizes a modified Dynamic Node Creation scheme (Ash, 1989) for architecture adaptation and a variation of the Quickprop algorithm (Fahlman, 1988) with heuristic rules from the Delta-Bar-Delta (DBD) algorithm (Jacobs, 1987) as the learning rule. The learning procedures can be briefly described as follows:

- Select an initial learning rate and momentum factor; do a standard back-propagation step; and store the gradient information and weight increments at the current step;
- If the sign of the weight increment is the same as the sign of consecutive gradient values, then perform weight update with the generalized delta rule using the maximum learning rate prescribed; otherwise, update weight using Quickprop learning rule to calculate the momentum factor;
- The learning rate adaptation is optional. In implementation, the learning rate can be either fixed as a small constant value or updated with the DBD scheme by using an exponential function for the estimation of its increment. A limit on the maximum learning rate allowed is also defined. The learning rate adaptation is usually activated when the network has already settled down in a neigh-

borhood of the solution set or when about 90% of the patterns have been learned, and when additional accuracy is demanded.

The mathematical formulae used in this algorithm for the adaptation of learning parameters are as follows:

$$\begin{cases} \Delta w_{ij}(t) = -\eta_{ij}(t)\ \partial E/\partial w_{ij}(t) + \dfrac{\partial E/\partial w_{ij}(t)}{\partial E/\partial w_{ij}(t-1) - \partial E/\partial w_{ij}(t)}\ \Delta w_{ij}(t-1) \\ \eta_{ij}(t+1) = \eta_{ij}(t) + \Delta\eta_{ij}(t) \\ \Delta\eta_{ij} = \eta\ e^{(-|\partial E/\partial w_{ij}(t)|)}, \text{ if } \nabla E(w_{ij}(t-1))\nabla E(w_{ij}(t)) > 0 \end{cases} \tag{2.12}$$

The architecture adaptation scheme is essentially based on the Dynamic Node Creation Scheme (Ash, 1989) with minor modifications. By using a flexible data structure, hidden nodes can be added in increments of any size and any reasonable number of hidden layers can be used. It was found through training networks on the experimental data of plain concrete that the use of a "trigger slope" as a generally applicable rule for guiding the creation of new hidden nodes was not reliable because the "trigger slope" tended to be case dependent and varied with different problems. In addition, once a new training case is added to the training data, the previously prescribed trigger slope has to be re-adjusted, and there is no reliable rule for this process. Therefore, from a pragmatic point of view, a fully automatic network architecture adaptation scheme is not practically realizable if only the "trigger slope" is used as the control signal for hidden nodes generation. Based on this observation, a heuristic for adding nodes to hidden layers is implemented in the simulator for the preliminary stage of training, in which a trigger parameter defined as the percentage of total correct predictions over the training sets is used. When the network has roughly settled in the solution space, say 80% of the training cases have been successfully learned, the architecture adaptation is then manually controlled. The training process converges when both the maximum absolute error and the total error are below their tolerances, respectively.

2.3.2 Data Structure and Implementation

To construct a general multilayer feedforward neural network, there needs to be defined at least three basic structures, namely, a network structure, a layer structure, and a weight structure for any two connected layers. On the network level, the number of linked lists needed to define a connected network depends on the allowable flexibility of connection schemes. If the network only allows connections between adjacent layers, then one linked list – a doubly linked layer list, is required for linking all the layers together to form a network. If connections from one layer to more than one layer are allowed, another singly linked weight list is needed to facilitate the computations involved.

The network structure serves the purpose of overall control of the program. Its elements include pointers to layer structures of the input and output layers, and head and tail pointers to the doubly linked layer list to facilitate access to any layer in the network. In the layer structure, the geometrical composition of each layer, the information that needs to be stored at each node such as the activation value and the error value, and pointers to the weight structure existed in front and at back of the layer, are specified. The weight structure is defined between two connecting layers, in which pointers to the front and back layer structures, data arrays storing values of the current weight matrix and the increments of weights at previous epoch, as well as the gradients of the error function at previous and current epoches are specified. Full connection is enforced to simplify the data structure of weights.

To facilitate a flexible construction of connection schemes, two linked lists are implemented in the simulator: 1) a doubly linked layer list, in which its elements consist of a pointer to current layer structure, a forward pointer to next layer, and a backward pointer to previous layer structure in the list; and 2) a singly linked weight list, in which a pointer to current weight structure and a forward pointer to next weight structure in the list are defined. This weight list is activated when the connection scheme from one layer to many layers is prescribed in the architectural design of the network.

The framework of implementation of the simulator can be schematically described in the following pseudocode:

```
○ If (in the training mode) then
   While (in the adding node mode) do
     If (at the first training epoch)
       setup data structures for the network; initialize learning parameters
       and weight matrix; read in the training data;
     Else
       modify the network architecture; reset data structures; and initialize
     weights on the new connections;
     End If
     Teach the network with Quickprop learning algorithm;
     If (the training is convergent)
       save the current weights;
     Else
       switch the training mode to the adding node mode;
     End If
   End While
  End If
```

```
○ If (in the testing mode) then
     setup the network architecture and data structure; reload in the previously
  trained weights;
     test the network with testing data.
  End If
```

The C programing language was used to implement the proposed simulator because of the inherent advantage provided by the language itself in data structure manipulation and dynamic memory allocation. This simulator was written in ANSI–C and ran on an Apollo DN3500 workstation under either the Domain SR 10.2 or the Unix operating system.

2.3.3 Performance Evaluation

The performance of the simulator implementing a variation of the Quickprop learning algorithm and a dynamic node generation mechanism (DQP), was studied by training some well defined benchmark problems, such as the encoding problem, parity, symmetry, and some nonlinear time series functions such as the Mackey–Glass function (Rumelhart, et al., 1986; Lapedes and Farber, 1987). It turned out that the learning rate of the simulator in terms of training epoches was in general agreement with that reported by other researchers (Fahlman, 1988; Jacobs, 1987). With an identical network architecture, the current scheme takes about an order less training epoches to reach convergence within a designated error tolerance than the standard backpropagation with the generalized delta rule. Moreover, with the use of the dynamic node generation scheme, the size of hidden layers is determined in the run time, and the final size is in the neighborhood of the minimal value determined by others (Rumelhart, et al., 1986). Tables 2.1 – 2.2 show the computational expenses in terms of training

Table 2.1 – The Computational Expense (Training Epoches) and Network Architecture Evolution for Training the N–bit Encoding Problem

N	Nodes	DQP (epoches)				DBP (epoches)				QP	BP
	2	50	100	80	70	100	200	500	50	N.C.	N.C.
16	3	90	164	84	77	1000	1000	1000	1415	131	N.C.
	4					196	234	186		66	487
	Mean	178				1471					
	2	50	100	30	50	50	100	100	500	N.C.	N.C.
	3	50	50	50	100	50	100	500	500	N.C.	N.C.
32	4	99	96	99	76	1100	1906	1088	1246	124	1590
	5					327				77	655
	Mean	213				1892					

N: size of the input pattern; Nodes: size of the hidden layer;
DQP: dynamic node generation with quickprop;
DBP: dynamic node generation with backpropagation;
QP: quickprop learning algorithm; BP: backpropagation learning algorithm;
N.C.: not converged after training for 5000 epoches.

epoches and the evolution of network architecture on the hidden layer for training the encoding and parity problems with different learning schemes including the standard backpropagation (BP), the dynamic node generation with backpropagation (DBP), and the standard quickprop algorithm (QP).

There are many factors that affect the convergence rate of a learning algorithm, such as the condition of the initial weight matrix, types of the activation function,

Table 2.2 – The Computational Expense (Training Epoches) and Network Architecture Evolution for Training the N–bit Parity Problem

N	Nodes	DQP (epoches)				DBP (epoches)				QP	BP
4	2	50	50	200	50	100	100	200	200	N.C.	N.C.
	3	50	300	300	400	500	1000	500	500	N.C.	N.C.
	4	423	340	170	172	1000	1000	2000	893	N.C.	N.C.
	5					3205	2847	144		235	N.C.
	7									183	738
	Mean	626				3547					
5	4	100	150	300	400	500	1000	2000	2000	N.C.	N.C.
	5	300	197	248	427	1000	1000	2500	2500	510	N.C.
	6	300				1000	1000	1500	3000	599	N.C.
	7	281				N.C.	N.C.	N.C.	N.C.	309	N.C.
	Mean	676									

N: size of the input pattern; Nodes: size of the hidden layer;
DQP: dynamic node generation with quickprop;
DBP: dynamic node generation with backpropagation;
QP: quickprop learning algorithm; BP: backpropagation learning algorithm;
N.C.: not converged after training for 5000 epoches.

connection schemes, and the values of learning parameters. In order to compare the performance of different learning algorithms, it is important to use a fixed set of values for learning parameters, such as the learning rate and the momentum factor. On the other hand, however, the optimal value of a learning parameter varies from one scheme to another. Therefore, it is extremely difficult to run a bias free comparative study on different learning algorithms. Nevertheless, with the use of a fixed set of learning parameters and some well defined benchmark problems, the study can always shed light upon the general performance of each individual learning scheme. For training those benchmark problems, it is generally agreed on that a small learning rate and an intermediate momentum factor should be used (Rumelhart, et al., 1986). Hence, a learning rate of 0.25 and a momentum factor of 0.80 were used, and all weights were initialized to random values in the range of (0.0, 1.0) in this study. Of course, the momentum factor was adaptively determined in schemes that use the Quickprop learning algorithm.

From the results shown in Tables 2.1 – 2.2, it can be seen that the DQP scheme is very effective and efficient in solving all the testing problems, especially when the size of training data set becomes large. Moreover, the final hidden size determined is always within the neighborhood of the minimal value. For example, the DQP solves the 16-bit Encoding with 3 hidden nodes, 32-bit Encoding with 4, 4–bit parity with 4, and 5–bit parity with 5 hidden nodes, which are actually the minimal values obtained from theoretical analysis (Rumelhart, et al., 1986). For larger problems such as the 7–bit parity, even though the testing results are not shown here, the DQP scheme always converges, whereas the BP and the DBP schemes have difficult to reach convergence even after training for over 5000 epoches. Though the QP scheme can also solve these benchmark problems very efficiently, it requires the network architecture be determined in advance.

Several interesting observations were made on the learning performance of the quickprop learning algorithm and the dynamic node generation scheme. The quickprop algorithm is very effective and fast in bring the internal representation of the problem within a network to a neighborhood of the ultimate solution set. This is expectable as the magnitude of the gradient value is not of a higher order than the momentum value in the early stage of training. For large problems, once the network gets into the vicinity of the solution set, or when 90% of the training patterns have been learned, the adaptive adjusting of learning rate becomes more effective. In the training of these benchmark problems with binary values in the input and output, especially parity, sometimes it is easy for the network to get trapped at a local minimum before the hidden size reaches the minimal theoretical value. The dynamic node generation mechanism usually provides a very effective relaxation agent for the network to escape from the local minimum, and the final network always converges to a quasi-optimal architecture.

CHAPTER 3

NEURAL NETWORK-BASED MATERIAL MODELING METHODOLOGY

3.1 Introduction

A constitutive law or a material model is conventionally described as a *mathematical* model represented as stress–strain relations that convey human perception of the behavior of a material. In engineering mechanics, material modeling constitutes an integral part in the study of the structural behavior under external excitation. With the availability of powerful computing machines and sophisticated computational methods, such as the finite element method, and the advances in experimental instrumentation and testing methods, the importance of the role that material modeling plays in computational mechanics is greatly enhanced. On the other hand, with the introduction of modern composite materials, the constitutive modeling of their complex behaviors becomes increasingly more involved.

Recent advances in neural networks, especially the new insights in developed learning algorithms, has facilitated the development of a fundamentally different approach to material modeling using neural networks. Within the framework of information science, the constitutive modeling of a material is a knowledge acquisition and representation process, in which the knowledge to be acquired and represented is the complex behavior of a material. Clearly, the learning or self–organizing capability of neural networks can thus be utilized to build a model of the behavior of a material, given an appropriate amount of data on that material's response to external stimuli. This realization, along with developments in hardware-based programmable neural networks and neural computing theory, have drawn a potentially new horizon for research in material modeling.

This chapter presents: 1) a brief review of the analytical approach to material modeling, specifically of concrete, 2) a summary on the neural network modeling procedures, 3) a description of the concept and principles of the neural network-based material modeling methodology, and 4) a comparison between the neural network-based and analytical approaches to material modeling.

3.2 Analytical Approaches to the Material Modeling of Concrete

The analytical approach to material modeling mainly consists of two aspects: the mathematical formulation of constitutive equations and the determination of material parameters (Desai and Siriwardane, 1984). The former involves the use of principles of mathematics and continuum mechanics, and the latter usually relies heavily on the rational identification and determination of material parameters through analysis of experiments performed on the material. To illustrate the major aspects involved in the analytical approach to material modeling, the constitutive modeling of concrete is briefly described in the following paragraphs.

To describe the behavior of concrete, the constitutive, or stress–strain, relation and the failure criteria need to be defined. There are several approaches to model the stress-strain behavior of concrete under various stress states. As summarized in the ASCE report (ASCE, 1982), analytical material models for concrete are generally developed based on the theory of elasticity and its generalization, the perfect and work-hardening plasticity theory, the plastic-fracture theory, and the endochronic theory of plasticity. For biaxial behavior of concrete, an equivalent uniaxial stress-strain relation is usually used, and stress-strain relations are established by curve fitting the test data. In the case of multi-dimensional analysis, the behavior of concrete is assumed to be incrementally elastic with variable moduli (Chen, 1982).

Failure theory of a material represents stress and strain states at which the material can no longer maintain its load carrying capacity. Some well known early failure theories for engineering materials include Rankine's maximum stress theory, the maximum strain theory, Coulomb's internal friction theory, Mohr's theory, and von Mises' octahedral shear stress theory. For concrete, because of its anisotropic behavior, those theories cannot adequately characterize its failure behavior. Recently, based on extensive testing on the stress-strain behavior of concrete, many new theories have been introduced specifically for concrete material, such as the biaxial maximum stress envelope developed by Kupfer and Gerstle (1973), triaxial maximum stress surface (William and Warnke, 1974), crack friction theory (Bazant and Tsubaki, 1980), and criteria based on the fracture mechanics theory.

Because of the limited availability of computing power, early constitutive models of concrete are primarily based on the theory of elasticity. The majority of these models are of the nonlinear elastic type and are used primarily to represent concrete behavior under monotonic or proportional loading (Kupfer, et al., 1969; and Liu, et al., 1972). In the uniaxial cyclic stress state, various models are developed to match behavior of structural elements (Karsan and Jirsa, 1969; and Sinha, et al., 1964). For concrete in biaxial stress states, incremental isotropic models (Gerstle, 1981), incremental anisotropic models (Darwin and Pecknold, 1977; and Liu, et al., 1972), and isotropic total stress-strain models (Kupfer and Gerstle, 1973) have been developed. The advantage of elasticity-based models is their conceptual simplicity and ease of use, as well as a reasonable representation of the overall behavior of concrete. However, the behavior of concrete at ultimate stress states is usually not well represented in these models.

Plasticity-based material models for concrete are developed based on the realization that the behavior of concrete is intrinsically inelastic, which is well manifested in the cyclic testing of concrete (Sinha, et al., 1964), and with the recent advances in computing technology so that complex numerical computations can be efficiently carried out. In this case, the stress-strain behavior is divided into the recoverable portion before yielding and irrecoverable portion after yielding. As such, elasticity theory is used to treat the recoverable, or elastic, behavior, while plasticity theory is applied to handle the irrecoverable, or plastic, behavior. In general, models based on the theory of plasticity describe concrete as an elastic-perfectly plastic material (Mikkola and Schnobrich, 1970; Nayak and Zienkiewicz, 1972; Salem and Mohraz, 1974) or as an elastic strain hardening plastic material (Chen and Chen, 1975; Buyukozturk, 1977).

For an elastic–perfectly plastic model of concrete, stress-strain relations have to be defined for concrete to characterize its behavior in pre–yielding, during plastic flow, and post failure stages. Before yielding, concrete is assumed to be elastic; during plastic flow, it is described by the plastic stress-strain relation; and the post failure behavior is governed by the constitutive relation of fractured concrete. In the determination of the plastic stress-strain relation during plastic flow, the yield criterion or yield function and failure criterion need to be defined. After defining the yield function, the incremental plastic stress-strain relation can be derived from the flow rule. Models based on this approach are mainly used for limit analysis.

In order to simulate the softening behavior, the elastic strain hardening plastic material models of concrete have been developed. Because of the introduction of a hardening mechanism, the material model becomes more complex, and consequently more rules need to be defined. In these models, there are hardening rules describing the motion of the subsequent yielding surface, the yield function that defines the initial and subsequent yield surfaces, the flow rule for determining the incremental plastic stress-strain relation, the criterion for loading and unloading during plastic deformation, and very often, the equivalent uniaxial stress-strain curve to calculate the plastic hardening modulus (Chen, 1982). The predictions based on this kind of model have traditionally exhibited good agreement with experimental observations, but these models are more complex and computationally more expensive than the elasticity-based models.

Of course, there are many other models such as the plastic fracturing model (Bazant and Kim, 1979), endochronic models (Bazant, 1976), etc. The common trend seen in the construction of mathematical material models is that as more complex material behaviors are captured in a model, more rules, parameters, and criteria need to be introduced and determined. Consequently, analytical material models built along this direction become more and more complex.

The advances in information science and computing technology have facilitated the introduction of new approaches to material modeling. In the following section, a fundamentally different material modeling approach based on the neural network modeling procedure is introduced.

3.3 Neural Network Modeling Procedure

In applying neural networks as a computational and knowledge representation tool to solve any non – trivial problem, the modeling process usually involves the following aspects: 1) problem representation, 2) architecture determination, 3) learning process determination, 4) training of the neural network with training data, and 5) testing of the trained network with testing data for generalization evaluation. These five aspects also constitute the framework of the neural network-based material modeling process to be described later.

In general, the problem representation process consists of evaluating the applicability of the neural network paradigm, the selection of the type of neural networks, data acquisition, data processing, and the design of representation schemes for input to and output from the network. The representation schemes are determined not only by the nature of the problem, but also by the way that models are to be used. There are basically two kinds of representation schemes: distributed representations and local representations. For function mapping problems such as material modeling, local representation scheme is usually adopted.

Architecture determination usually involves the selection of the number of layers and nodes in each layer, as well as the inter–connection scheme. Obviously the size of the input and output layer is solely determined by the representation scheme devised. However, the size of each hidden layer and the number of hidden layers are strongly influenced by the complexity of the problem, features or regularities embedded in the training data, and the efficiency of learning algorithms. In another aspect, the way that nodes in different layers are connected is also very important because it controls the pathway for information flow or propagation in a network. Though the connection between layers can be forward, backward, and recurrent; or be established between subsets of processing units in different layers, for simplicity, complete connection between adjacent layers is usually enforced in multilayer feedforward neural networks, especially when dealing with function mapping problems.

After the data representation scheme and initial network architecture are defined, the determination of a generic learning process involves making decision on the type of processing units such as the Σ unit and the $\Sigma\Pi$ unit, the selection of activation function, and the design of learning algorithms. Once the whole learning system is constructed, the training and testing process can be performed.

Training means that the defined network is presented with processed training data and learns or discovers the relationships embedded in the data using learning algorithms. Convergence of learning is reached if the error associated with the network prediction falls within a specified error tolerance. If a presentation of the whole training data to the network is defined as a learning cycle or an epoch, the iterative training process usually requires many hundreds or thousands epoches to reach convergence. After the network is properly trained, its generalization capability is evaluated in the testing phase. If the trained network generalizes reasonably well on novel but similar cases, the resulting neural network can then be qualified as a legitimate model for use in the problem domain.

For real world engineering problems, this whole modeling process is likely to be an iterative process, and the generalization evaluation on the trained network from the testing phase functions more like a feedback signal. Since a neural network learning system is an integration of different mutually interacting learning components, one or sometimes even all of the previous processes may need to be examined and adjusted if the generalization capability of the trained network is unsatisfactory. The discrepancy between the expected output and network prediction may be result from any of the following sources: 1) an inappropriate representation scheme of the problem; the training data is not comprehensive enough to represent the essence of the problem; or the domain is not suitable to neural networks; 2) the current architecture of the network is insufficient to accommodate the knowledge to be captured; 3) the learning algorithm is not efficient and robust enough to handle the complexity of the problem; and 4) the training is pre–maturely terminated. Some aspects of these arguments will be illustrated in the determination of minimal training data sets for the construction of material models of concrete and reinforced concrete.

3.4 Neural Network-Based Material Modeling Methodology

The basic strategy for developing a neural network-based model of material behavior is to train a multilayer feedforward neural network on the stress-strain results (data) from a series of experiments on a material. If the experimental data about the material behavior are fairly comprehensive, the trained neural network would contain sufficient information about the material behavior to qualify as a material model. Such a trained neural network not only would be able to reproduce the experimental results it was trained on, but through its generalization capability it should be able to approximate the results of other experiments on the same material. For example, as will be illustrated in Chapters 4 and 5, neural networks are trained on the results of several proportional stress paths of concrete and reinforced concrete. These trained networks can simulate the test results for other proportional and even non-proportional stress paths that fall within those on which it was trained. The degree of accuracy in this generalization depends on both how comprehensive and representative the training set is and how well the network is trained.

Clearly, the procedures used in the construction of a neural network-based constitutive model of a material would fall into the general framework of the neural network modeling process described in the previous section. Because of the nature of a material model and its intended use within the finite element method, the modeling procedure has its own characteristics and requires special considerations.

As has been mentioned before, the first step in constructing a neural network-based material model is the determination of representation scheme for material behavior in the input and output. The composition of the input and output layers depends primarily on the intended use of the neural networks. Although neural networks offer considerable flexibility in this regard, it is natural that the first attempt in the development of neural network-based material models should follow the traditional mathematical models for use with finite element methods. As such, the processing units in the input and output layers all represent stresses, strains, their increments, and in some cases a portion of the stress-strain history. Since the material behavior is highly path dependent, the input must have sufficient information for the neural network to characterize the stress-strain state of the material and contain certain information on the previous history. Therefore, two representation schemes – the so called one–point and three-point schemes, are introduced to characterize the behavior of a material in different stress states. These representation schemes can be either stress–controlled which means that the network is to predict strain increments corresponding stress increments, or strain-controlled on the contrary.

For instance, in a stress-controlled one–point representation scheme, the stress–strain state of a material at one point in the stress space and strain space and the next stress increments at that point are included in the input, and the corresponding strain increments are in the output. For a strain-controlled one-point representation scheme, however, the strain increments are in the input and stress increments are in the output. The three–point representation scheme is an expansion of the one–point scheme, with an expanded input including two additional stress–strain states in the stress–strain history.

Decisions regarding the neural network architecture are of primary importance in the successful construction of neural network-based material models. The capacity of a neural network is a function of the number of hidden layers and the number of processing units in each layer (Hornik, et al., 1989). The pattern of connectivity between the layers is also part of this equation. However, in this study a simple pattern of connectivity is used: each processing unit has outgoing connections to all the processing units in the next layer. The capacity of the neural network is also somehow related to the amount of the information in the training data and the complexity of the knowledge contained in that data. Currently there are no quantitative theories or good qualitative rules for determining the capacity of a multilayer feedforward neural network, as this aspect is not yet well understood. Though theoretical studies have concluded that one hidden layer with enough hidden nodes can accomplish the modeling of any functions (Hornik, et al., 1989; Cybenko, 1989), in practice, especially with modeling of continuous functions, it has been observed that the use of two hidden layers would yield a more efficient training. Therefore, two hidden layers are used with all the networks in this study.

For material modeling problems, with the use of two hidden layers, the size of each hidden layer is determined by the modified dynamic node creation scheme described in the previous chapter. Consequently, the final size of each hidden layer thus determined corresponds to the network architecture when a minimal or optimal training data set is successfully trained. This minimal or optimal training data set is defined as a set of data that contains sufficient information to characterize the behavior of a material.

To facilitate the representation of tensile and compressive data, a sigmoidal activation function defined in the range of $(-1, 1)$ is used. A learning algorithm based on Quickprop (Fahlman, 1988) has been developed for handling the large amount of training data with diverse characteristics resulting from the anisotropic behavior of materials. Before the training process starts, one immediate question is how large a training data set should be such that the material behavior is essentially characterized.

As there is no theoretical guidance on this question, an engineering approach is developed in this study to estimate the comprehensiveness of the training data set iteratively. This approach is designated as "incremental training with generalization control."

Whether or not a neural network has been trained with the minimal training data set is indicated by how well the trained network generalizes on the testing cases. Ideally, if the network is trained with a quasi-optimal or quasi-minimal training set, reasonable generalization should be observed on the testing results. Otherwise, if the training set is too small, poor testing performance would be expected, as the trained network has not been presented with all examples of the relevant information so as to generalize properly. On the other hand, if the training data set is too large, no substantial improvements would result from further training, after the network has been trained with the minimal training data set.

In the incremental training scheme proposed, training and testing proceed in the following way: 1) start with a small network and a small training set, and train the network until convergence; 2) add additional data to the training set, and restart training on the augmented data with the previously converged network; add nodes to hidden layers as needed; 3) when a training set containing a reasonable number of stress – strain data has been successfully trained, perform the generalization tests on untrained stress – strain cases; and 4) if all the testing results appear in good agreement with expected behavior, stop training; otherwise repeat the data set addition and generalization testing processes.

There are some benefits to using incremental training with tests for generalization evaluation. First, with the use of the dynamic node generation scheme and incremental presentation of the entire training data set, the network is not overwhelmed by the large amount of information at the initial stage of training so that the learning process converges faster than when guessing a network architecture and presenting the network with the whole training set at once. Secondly, starting with a small amount of data and monitoring the generalization performance of the neural network at certain stages of training, a quasi–minimal training set can usually be obtained. However, the true minimal training set is not theoretically defined at this time, but it is known to depend on both the comprehensiveness of the available experimental data on a material and the characteristics of the problem.

3.5 Comparison of Neural Network-Based Approach with Analytical Approach to Material Modeling

The neural network-based material models differ in some fundamental ways from the traditional mathematical models of material behavior such as the plasticity models. In order to compare these two radically different methods of material modeling, we reiterate the steps involved in the development of traditional material models. Three basic steps can be identified in the development of mathematical models: 1) the identification of the main features of material behavior from the experimental data, which includes features such as: elastic behavior, yielding, strain hardening, strain softening, brittle failure, micro-cracking, and localization; 2) development of a set of mathematical rules and expressions such as: yielding function, loading criteria, flow rule, hardening rule, and failure criteria; and 3) determination of the material parameters from the experimental results.

These mathematical rules and expressions are constrained to satisfy the fundamental laws of mechanics such as the energy conservation laws, namely the first and second laws of thermodynamics, conservation of mass, balance of linear momentum, balance of angular momentum, hence symmetry requirements, principle of invariance, and frame indifference. That is, the constitutive equation must be consistent with the physical laws.

By contrast, in the proposed method of material modeling, neural networks are used as computational tools capable of both capturing the "knowledge of the material behavior" directly from the experimental data and storing this knowledge within the connection strength of the network.

Unlike traditional mathematical models, neural network-based material models do not explicitly represent the features of material behavior; thus the trained neural network is used like a "black box". Nor can the neural network-based material models be rigorously proven to obey the energy conservation laws and symmetry and invariance requirements. However, if the neural network is trained on a comprehensive set of experimental data, it is reasonable to assume that the resulting material model will approximate all the laws of mechanics which the material actual obeys. This belief is the consequence of the recent proof (Hornik et al. 1989; Cybenko, 1989) that the feedforward multilayer neural networks are universal approximators of any function if the networks have sufficient processing units in their hidden layers, and have been presented with a sufficient number of examples of this function. Note, no definition of sufficiency is given in this theorem. Of course, such a comprehensive set of material data may not be available for most materials at the present.

In the next chapter, the neural network-based material modeling methodology introduced in this chapter is applied to represent the behaviors of plain concrete under biaxial stress states and under uniaxial cyclic compression so that the applicability of this approach is verified. The extension of this approach to constitutive modeling of composites, specifically reinforced concrete material, is presented in chapter 5.

CHAPTER 4

NEURAL NETWORK MATERIAL MODELS OF PLAIN CONCRETE

4.1 Introduction

In this chapter, the neural network-based material modeling methodology is applied to the constitutive modeling of plain concrete under short-term monotonic biaxial loading and under uniaxial cyclic compressive loading. Both stress–controlled and strain-controlled models for concrete in biaxial stress states are studied. A simple elastic unloading mechanism is investigated in the stress state of biaxial compression.

The behavior of concrete has been well understood through numerous experiments and extensive theoretical research in the past several decades. It is generally agreed upon that the mechanical behavior of concrete is decisively determined by the formation, distribution, and propagation of microcracks in concrete before and during loading. As an inhomogeneous material that is composed of coarse aggregates and mortar, concrete experiences a highly nonlinear behavior in its stress-strain relation even in the simple uniaxial stress state. It is thus clear that a valid constitutive model of concrete should be able to represent various facets of its complex nonlinear behavior. To facilitate the discussion of neural network constitutive modeling, a brief summary of some of those experimental observations on the behavior of concrete is presented in the following section.

4.2 Material Characteristics of Plain Concrete

In general, experimental observations indicate that concrete behaves as a linear elastic material when it is subjected to low level of stresses; it exhibit highly nonlinear behavior under higher levels of stresses. The strength and stiffness of concrete under uniaxial compression are different from those under multiaxial loading. In addition, the duration of loading has a significant effect on the stress–strain response of concrete, as illustrated in its behavior under long-term sustained loading and under dynamic loading such as impact.

When subjected to monotonic uniaxial compression, according to Winter and Nilson (1979), concrete behaves elastically for stress level up to 30 percent of its maximum compressive strength f'_c as the microcracks are stable at low stresses. For stresses above this early threshold, concrete starts to soften until it reaches its maximum strength at a strain level between 0.002–0.003. During this period, microcracks start forming at the mortar-aggregate interfaces, and propagates to form more complex cracking systems upon further loading. After reaching maximum strength, if subjected to increasing compressive strain, the concrete exhibits the descending portion of its stress-strain relation.

Under uniaxial tension, the shape of the stress-strain curve of concrete is similar to that under uniaxial compression, but its tensile strength is only about 5–10 percent of its compressive strength (Hughes and Chapman, 1966). This behavior is mainly caused by the rather low tensile strength at the mortar-aggregate interface of this composite material. Due to difficulties associated with the control of a tension test, the softening behavior of concrete after reaching its tensile strength is not easy to obtain, and concrete behaves more like a linear elastic brittle material in tension.

The behavior of concrete observed from uniaxial cyclic compression tests (Karsan and Jirsa, 1969; and Sinha, et al., 1964) shows clear degradation in both stiffness and strength of concrete with increasing number of applied cycles for a stress level above

$0.60f'_c$. The inelastic behavior is illustrated in the hysteresis loop formed during each cycle of loading and unloading. The stress-strain curve under monotonic compression can serve as a reasonable envelop curve for the peak values of stress under cyclic loading (Sinha, et al., 1964).

The behavior of concrete under biaxial states of stress is somewhat different from that under a uniaxial state of stress. Because of the presence of the second stress component, the behavior of concrete is represented specifically in the three regions: compression-compression, compression-tension, and tension-tension. Experimental results (Kupfer, Hilsdorf and Rusch, 1969; Nelissen, 1972) show that the loading stress ratio has a significant effect on the strength of concrete in both biaxial compression, and compression-tension. The compressive strength of concrete increases by almost 25 percent with a stress ratio of $\sigma_1/\sigma_2 = 0.5$, and is reduced by about 16 percent at $\sigma_1/\sigma_2 = 1.0$. In the region of compression-tension, the compressive strength of concrete is drastically reduced with the increase of tension stress. However, in biaxial tension, the behavior of concrete is similar to that in uniaxial tension with an unchanged or slightly increased tensile strength.

4.3 Neural Network-Based Material Models of Concrete in Biaxial Stress States

In structural analysis, many important classes of structures, such as beams, panels, and thin shells can be simplified by considering them as being in a biaxial stress state. As a first application of the neural network-based material modeling methodology, the objective is thus to construct a neural network constitutive model to represent the biaxial behavior of plain concrete. The experimental results reported in (Kupfer, et al., 1969) cover a wide range of loading stress paths in the state of biaxial stresses and are judged to be comprehensive enough for the purpose of training a multilayer feedforward neural network. The experimental results in terms of stress–strain relations are stress controlled tests of plain concrete undergoing monotonic proportional loading. Consequently, the trained network can be expected to have sufficient information on the proportional or nearly proportional loading stress paths. Obviously, on the basis of this information alone, the neural network can not gain information on stress reversals and highly non-proportional stress paths. In the second application, we perform some experiments on training a different neural network on uniaxial compressive cyclic tests on plain concrete.

4.3.1 The Representation Schemes and Architecture Determination

The requirements on the design of representation schemes for this problem are that both the characteristics of behaviors of the material should be represented, and it should support the possible use of these material models within a numerical solution scheme. For this material modeling problem, the obvious inputs and outputs are stresses and strains. However, material behavior is path dependent. In order to capture the path-dependence of material behavior, the network was structured so as to predict strain increments given the current state of stress and strain and a stress increment. Because the strain increments predicted by the network are path dependent, one must use the network iteratively to predict a strain state for a given stress state. This is done by starting at a known stress-strain state (usually a stress-strain free state), incrementing the stresses by small amounts, and using the neural network to predict the strain increments. These strain increments can then be added to determine the new state of strain which can be used to predict the strain increments for another stress increments. The model would thus be stress–controlled, i.e., increments of stress as input and increments of strain as output. However, a strain–controlled model (i.e., increments of strain as input and increments of stress as output) would be more directly usable within the Finite Element Method. Since the experimental data are obtained from stress-controlled tests, it is more natural to develop stress-controlled material models to simulate the behavior of the material. For completeness of this investigation, both stress–controlled and strain–controlled models were developed, and the strain-controlled training data are directly converted from stress-controlled training data by switching the positions of stress components with strain components.

As has been described before, this representation scheme is designated as a one-point scheme, as it only uses information at one point on the stress-strain curve to predict the strain increments. For monotonic proportional loading cases, the one-point scheme can adequately capture the characteristics of the material behavior. To accommodate the conditions of cyclic loading or with reloading-unloading, a three-point scheme is needed in which stress-strain relations at previous two points and current on the stress-strain history along with stress increments at the current point are used as input to predict the new strain increments as output. The three-point scheme is required for a unique distinction between the case of reloading and unloading.

After the representation scheme is defined, the preprocessing (or most often the scaling) on the original experimental data is usually performed because the use of sigmoidal activation functions restricts the range of data value accepted by the network. In this study, linear scaling is used throughout all the data types including stresses, strains, and their increments, and all the data are linearly transformed to the interval of $(-1.0, 1.0)$. The data transformation process proceeds as follows: 1) to find the maximum and minimum values of each stress and strain data (max_σ_1, max_σ_2, max_ε_1, max_ε_2, min_σ_1, min_σ_2, min_ε_1, min_ε_2), and linearly transform each set of data from the range of either $(-\max, -\min)$ to $(-1.0, 0.0)$ or from $(\min, \max)$ to $(0.0, 1.0)$; and 2) to calculate the stress and strain increments, and linearly transform the increments in the same way as with stress and strain data. After all the stress-strain data corresponding to different stress paths have been pre-processed, a part of this data is designated as the training data set and the remaining as the testing data set. The optimal size of the training data set is usually not readily determinable at the beginning of the training, but an estimation based on the modeler's understanding of the problem would be a good starting point. In this study, the problem is solved via "incremental training with generalization control."

The first neural network model developed for concrete in biaxial stress states uses one-point representation scheme and has six units in the input layer and two units in the output layer. The size of the two hidden layers is dynamically determined during training. For the stress controlled model, the six input units are two stresses, two strains and two stress increments (σ_1, σ_2, ε_1, ε_2, $\Delta\sigma_1$, $\Delta\sigma_2$), while the output units are two strain increments ($\Delta\varepsilon_1$, $\Delta\varepsilon_2$), as shown in Fig. 4.1. In the strain controlled model the last two units of the input layer represent the strain increments and the output units represent the stress increments.

4.3.2 Training and Testing of Neural Network-Based Concrete Models

Training a neural network material model involves presenting the network with the experimental data and have it self–organize, or modify its weights, such that it correctly reproduces the strain increments when presented with the current states of stress and strain and the stress increments (for a stress–controlled model). The training process is iterative and its convergence is reached only when certain specified convergence criteria, such as some error tolerances, are satisfied. For a strain–controlled model, the same data were used, but the strain increments were presented to the network as input and the stress increments as output. It was discovered that an effective way to train the neural network was to present the data to the network incrementally.

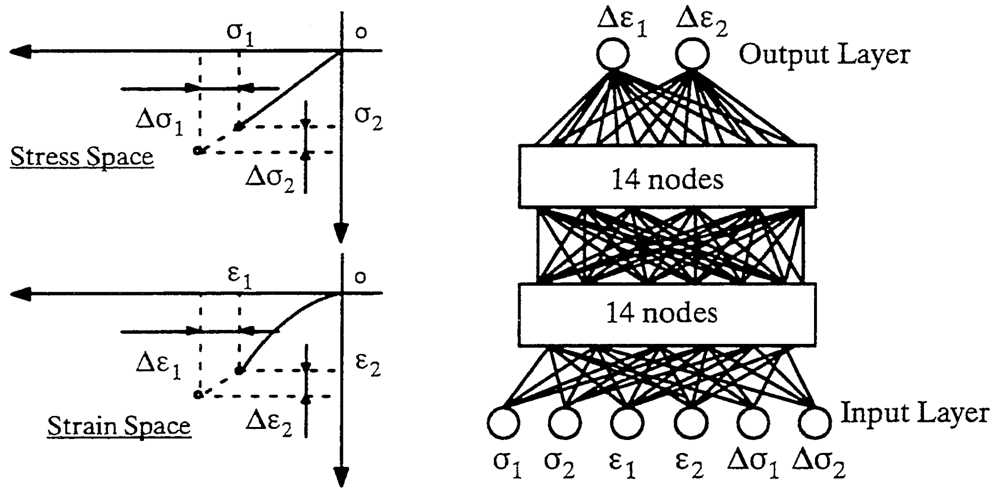


Fig. 4.1 – Architecture of the Stress-controlled Neural Network Material Model of Concrete in Biaxial Stress State

For this problem, there were totally 18 stress/strain relations available from experiments (Kupfer, et al., 1969; and Nellissen, 1971). With one-point stress-controlled representation, initially 9 stress paths with 3 in each of the stress regions were selected as training cases, and the remaining 9 stress paths as testing cases. With incremental training, the stress-strain data were then presented to the network with an increment of 3 stress paths at a time. Those data were trained until the maximum error was reduced to less than 0.09

After the network is successfully trained with the training data set, testing is performed to determine the generalization capability of the network on testing cases. With a stress-controlled neural network material model, the untrained stress paths are presented to the network to predict their corresponding strain paths. This testing aspect is vital to the determination of the validity of a neural network material model. If the predictions on stress–strain relations of some testing cases from the network trained so far does not match well with the experiment data, then those testing stress/

strain data need to be included in the training data set so that the associated features that are currently lacking will be captured through further training.

With an architecture adaptation capability of the simulator, the initial architecture of the neural network consists of a small number of nodes in each of the two hidden layers. During training, as the information – the content of the data sets presented to the network increases, more nodes are added to each hidden layer. This process continues until the learning process converges for the training data. For instance, with the stress-controlled one-point scheme, the network starts with 10 nodes in each of the two hidden layers, and 4 more nodes are added in increments of 2 during training so that the final architecture has 14 nodes in each hidden layer, as shown in Fig. 4.1.

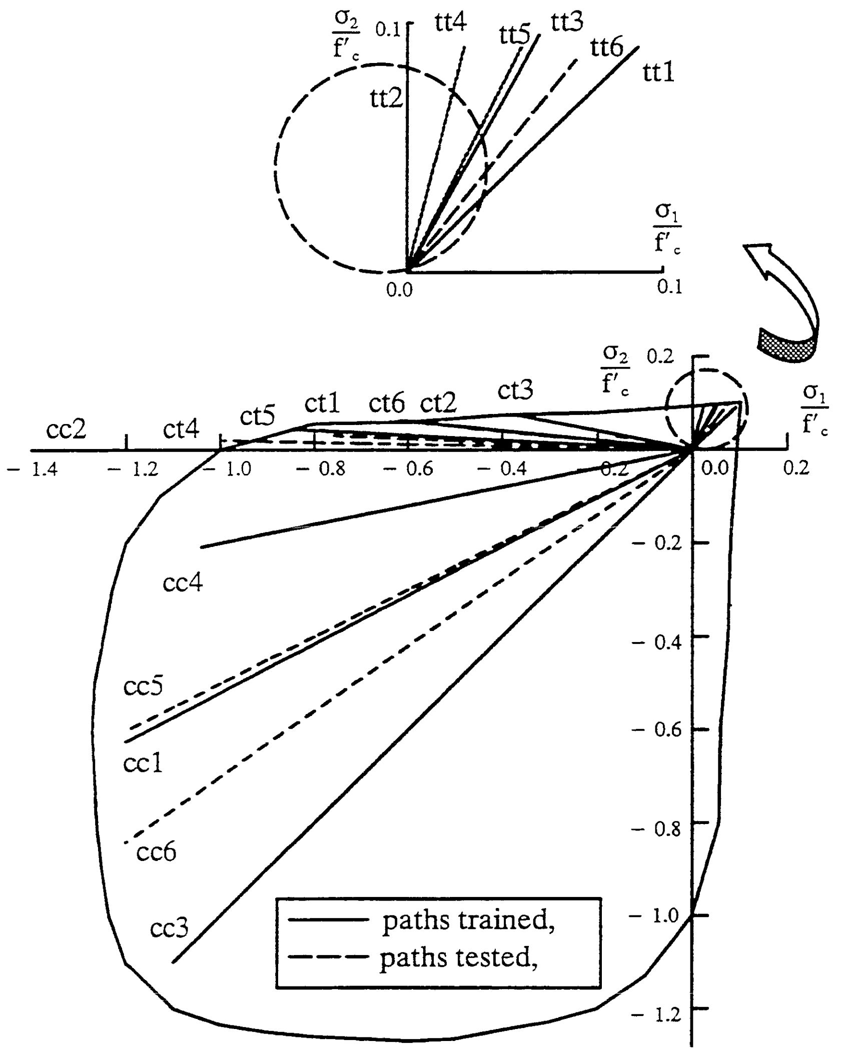
The determination of composition of training data is also an iterative process. For example, after training the stress–controlled material model with one-point scheme on stress–strain relations corresponding to 9 stress paths, a testing was performed on the 9 training and the remaining 9 untrained cases. Though these training and testing results were not presented here, it was observed that the learning error was very small (< 0.09) on all the training cases, and testing results on strain paths from 8 out of 9 testing stress paths were reasonable with good accuracy (maximum testing error < 0.10) except that of one stress path ($\sigma_1/\sigma_2 = -1.0/-0.20$). On testing that stress path, large discrepancies were seen on the stress–strain relation predicted by the network in the minor stress direction ($\sigma_2 - \varepsilon_2$), even though the network prediction in the major stress direction ($\sigma_1 - \varepsilon_1$) was reasonable. Therefore, the stress-strain relation corresponding to that stress path was added to the training data, and the previously converged network was given additional training cycles with these 10 stress/strain paths until the same error tolerance was satisfied. Afterwards, when other stress paths from the testing data set were added to the training data set, there were no substantial improvements on the testing results. This indicates that for this problem, 10 stress/strain relations constitute a quasi-minimal training data set. With those architecture adaptation and modified new learning algorithm described in Chapter 2, the whole training process took about several thousand of epoches of training to reach convergence and the subsequent retraining was very fast.

In the following paragraphs, the training and testing results of both stress-controlled and strain-controlled models with one-point scheme are described in detail.

4.3.2.1 A Stress-controlled Material Model with One–point Scheme

For the stress-controlled model with one-point scheme, the results after training the network on the 10 stress/strain paths shown in Fig. 4.2, are shown in Fig. 4.3. The graph in Fig. 4.3 shows the strain paths (i.e., strain increments) predicted by the neural network (shown as dashed lines) against those expected and used as training cases (shown in solid line). The labelling of paths is defined in Fig. 4.2. Another way to look at these results is to plot stress–strain relations corresponding to each training case. The graphs in Figs. 4.4 – 4.6 show the expected stress/strain curves (experimental results) against those predicted by the neural network. The results shown in Figs. 4.3 and 4.4 - 4.6 illustrate that neural networks can be used to capture the material behavior information on which they are trained with reasonably good accuracy (maximum error < 0.10).

The next question is how well does this trained network generalize what it has “learned” when presented with stress paths for which it was not trained. In theory, the training and testing processes are two separate processes and one is performed after the other. Nevertheless, as have been described before, in the neural network material modeling, the training and testing processes become two mutually interactive and



Values of σ_1 / σ_2:
cc1: −1.0/−0.52, cc2: −1.0/0.0, cc3: −1.0/−1.0, cc4: −1.0/−0.20, cc5: −1.0/−0.5, cc6: −1.0/−0.7
ct1: −1.0/0.052, ct2: −1.0/0.103, ct3: −1.0/0.204, ct4: −1.0/0.02, ct5: −1.0/0.04, ct6: −1.0/0.10
tt1: 1.0/1.0, tt2: 0.0/1.0, tt3: 0.55/1.0, tt4: 0.25/1.0, tt5: 0.50/1.0, tt6: 0.75/1.0

Fig. 4.2 − Behavior of Plain Concrete Under Biaxial Loading − Stress Paths and Strength Envelope (Kupfer, et al., 1969)

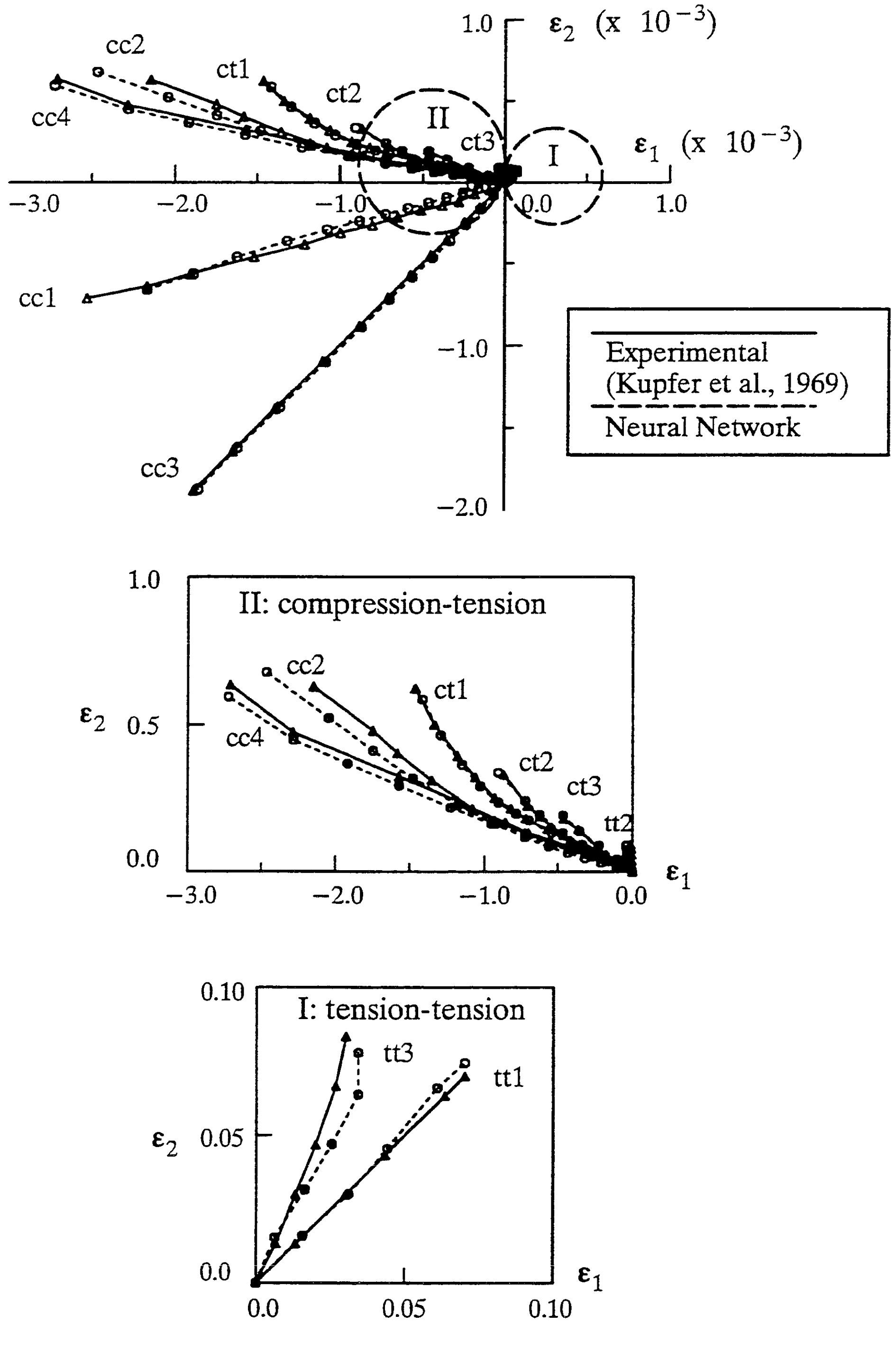
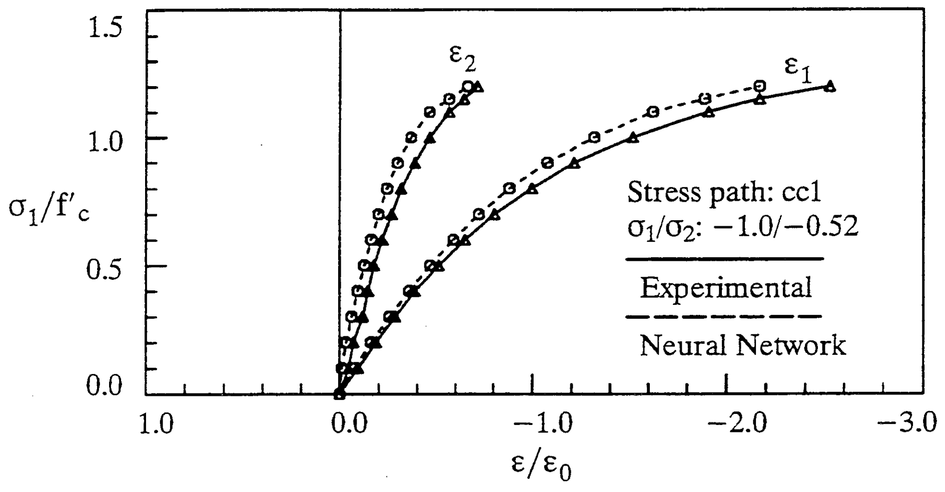


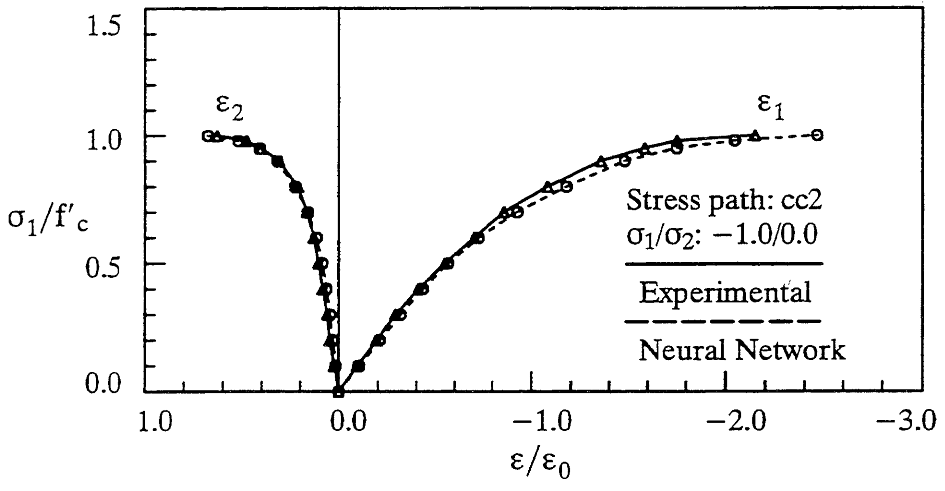
Fig. 4.3 – Prediction of the Neural Network on Strain Paths – Training Results (with Stress-controlled One-point Scheme)

integrated parts of the whole process, for the quasi–minimal training data set is determined through testing for generalization using the testing data. Therefore, once a material model is properly trained via the approach described in Section 3.4, its generalization capability has also been nominally checked.

Testing of the neural network material model involves presenting the network with stress paths that are not part of the training set and seeing what strain paths are predicted. With the incremental testing scheme, the resulting strain paths corresponding to the test stress paths are generated as follows: 1) the stress path is started at the origin free of any stress and strain, where the initial stress-strain state with the first set of stress increments is presented to the network to generate the strain increments; and 2) the strain increments that produced by the network are added to the previous state of strain to get the new current state of strain; and with a new stress increments, the process continues recursively until the final stress state is reached.

The stress paths on which the trained neural network was tested are shown in dashed lines in Fig. 4.2 and the corresponding strain paths are shown in Fig. 4.7. The testing results in terms of stress–strain relations on the testing stress paths (total 8) are shown in Figs. 4.8 – 4.10. In Figs. 4.8 – 4.10, the stress-strain relationships predicted by the trained neural network are compared with the experimental results reported in another experiment (Nelissen, 1972). Although the neural network was not trained for these stress paths, it is obvious from these figures that the network predictions are quite reasonable with acceptable accuracy (maximum error < 0.09 for testing cases in biaxial tension, < 0.10 in biaxial compression, and < 0.15 in tension-compression). The larger error seen at the higher stress region in the major stress-strain direction ($\sigma_1 - \varepsilon_1$) for testing cases in tension-compression was caused by the error accumulated during the initial stage of the incremental testing. In spite of this error accumulation phenome-





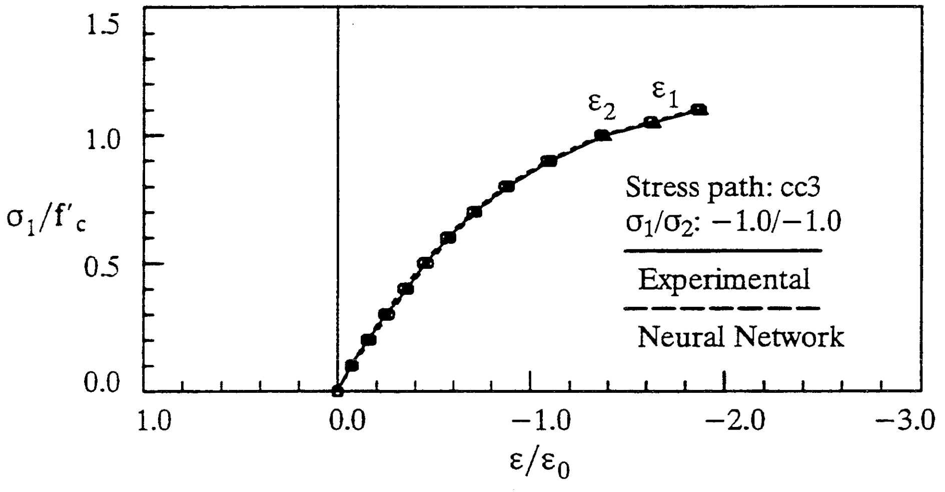
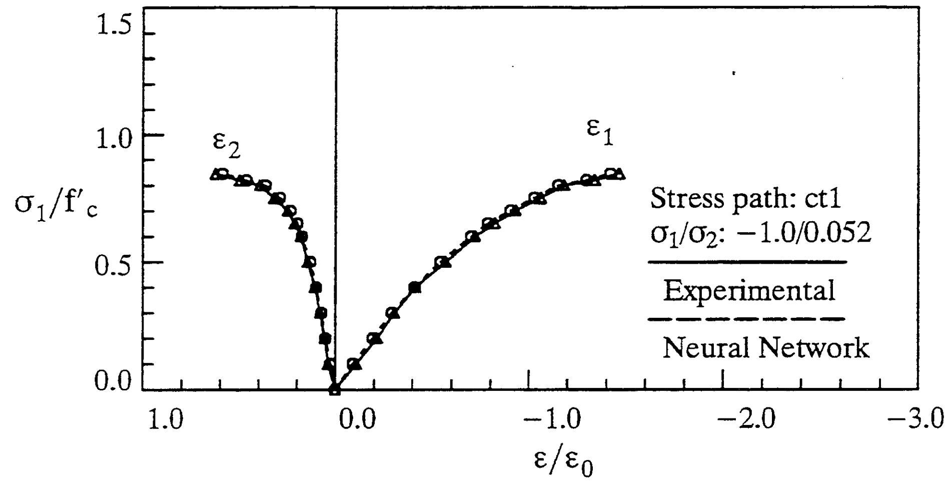
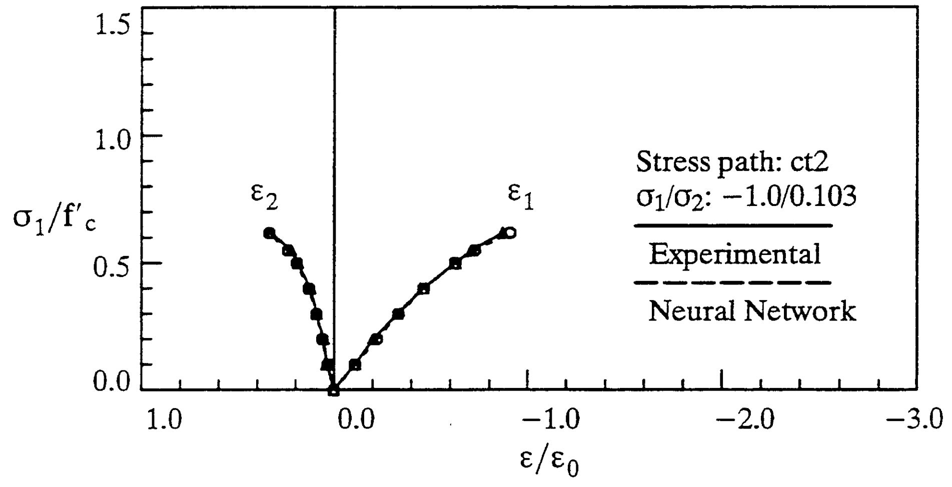


Fig. 4.4 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Stress-controlled One-point Scheme)





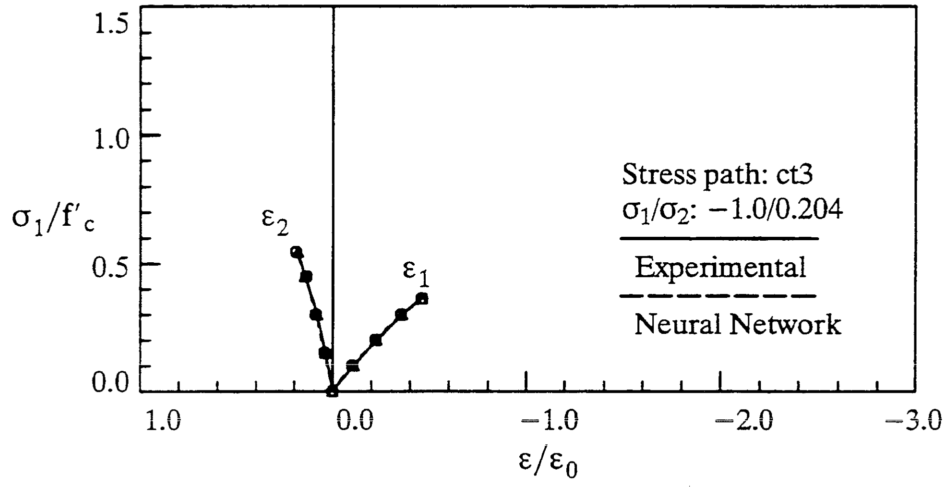
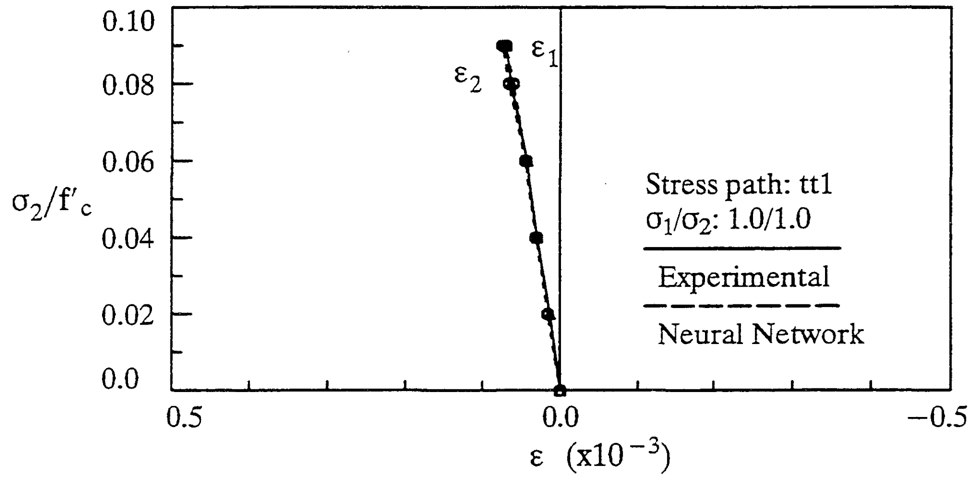
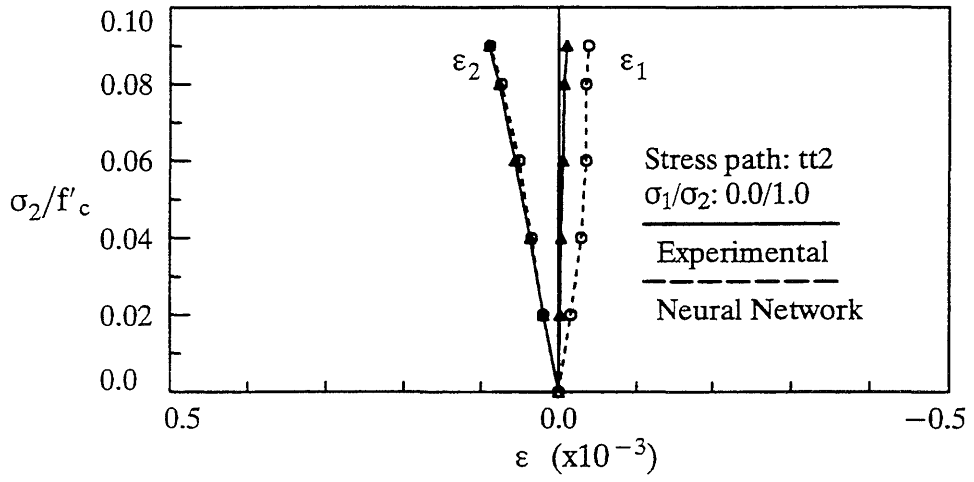


Fig. 4.5 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Stress-controlled One-point Scheme)





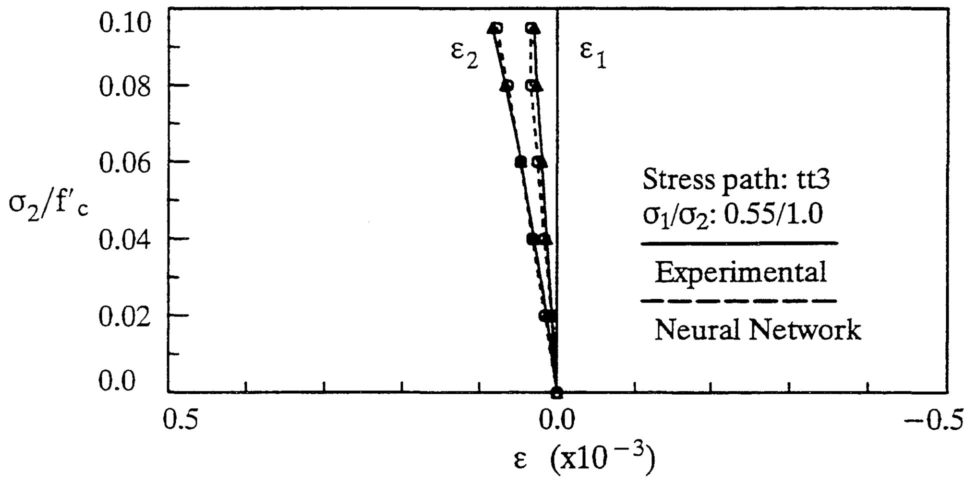


Fig. 4.6 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Stress-controlled One-point Scheme)

non, the predictions of the network on testing stress paths matched reasonably with experimental results, which indicates that the trained neural network is capable of generalization and synthesis from the stress paths on which it was trained.

One of the attractions of using a neural network to model material behavior is that a strain-controlled model can be developed with no more effort than is required for the development of a stress-controlled model. Such a strain-controlled model is, of course, more suitable for usage in Finite Element analysis. Since the experimental data (Kupfer, et al., 1969) were obtained with only stress-controlled tests, no strain-controlled data are directly available. Therefore, it is reasonable to directly transform the stress-controlled data in a strain-controlled format, which is accomplished by simply exchanging the positions of stresses and stress increments with their strain counterparts. In this way, the strain-controlled training data set is derived from the stress-controlled training data set. In the next section, training and testing results of a strain-controlled material model of concrete with one-point scheme are described.

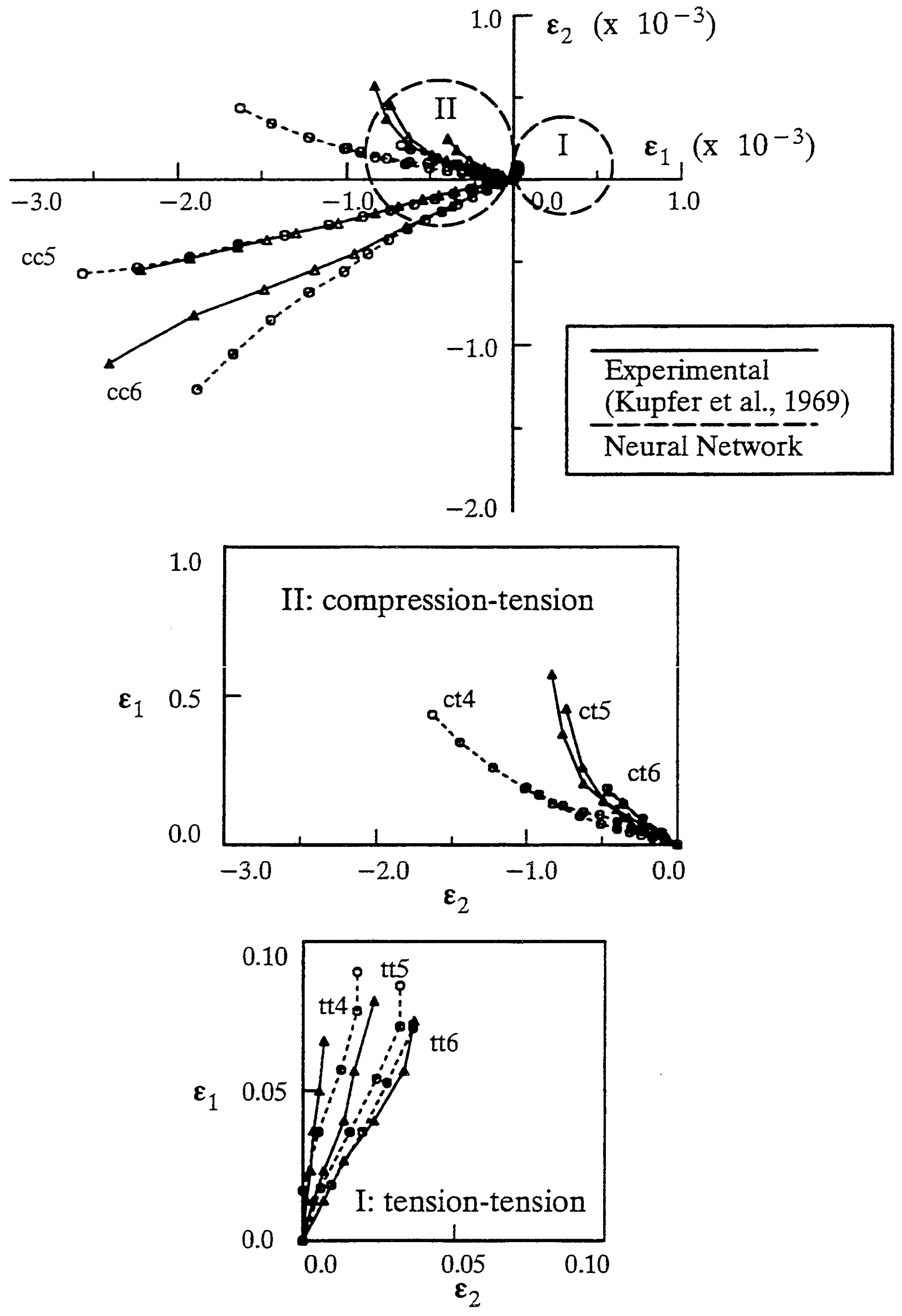


Fig. 4.7 – Predictions of the Neural Network on Strain Paths – Testing Results (with Stress-controlled One-point Scheme)

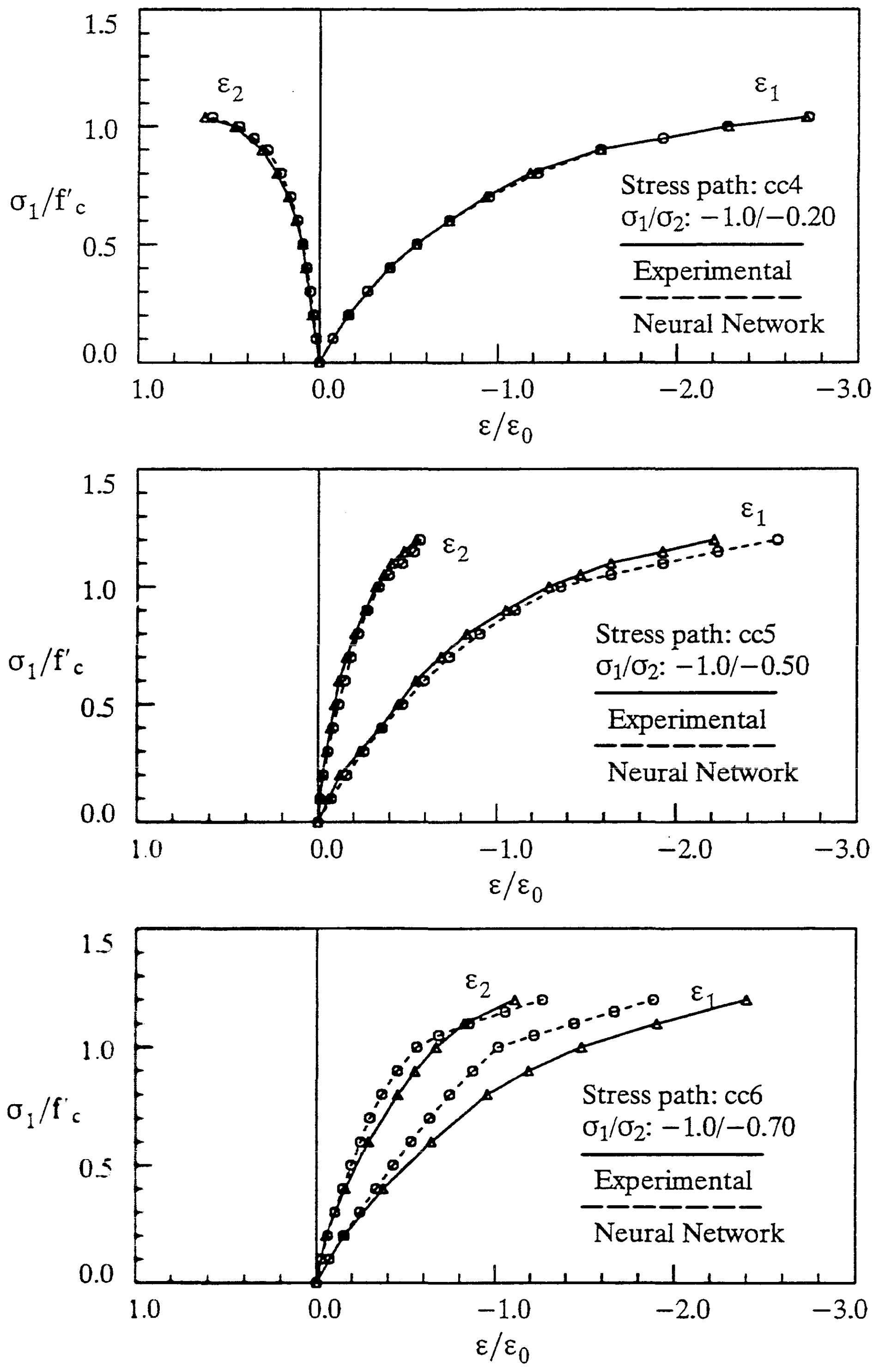
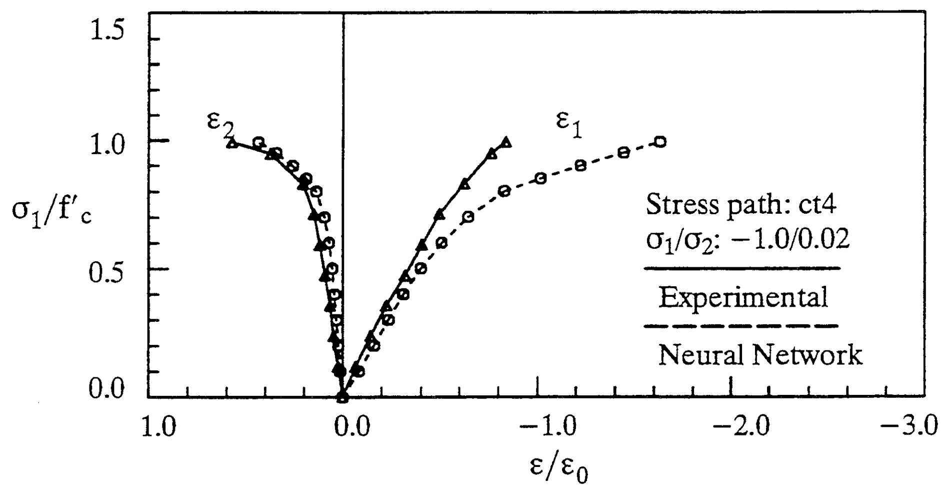
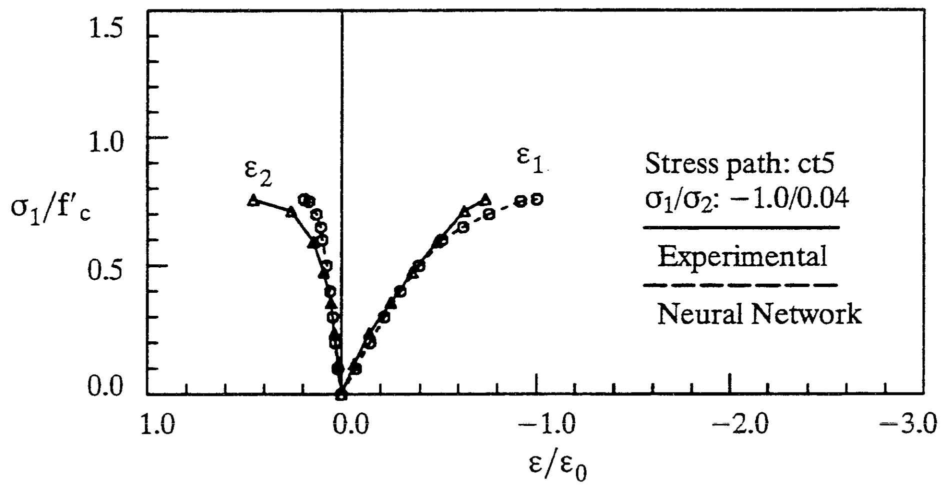


Fig. 4.8 – The Stress-Strain Relation Predicted by the Neural Network (Training Result for cc4 and Testing Results for cc5 and cc6)





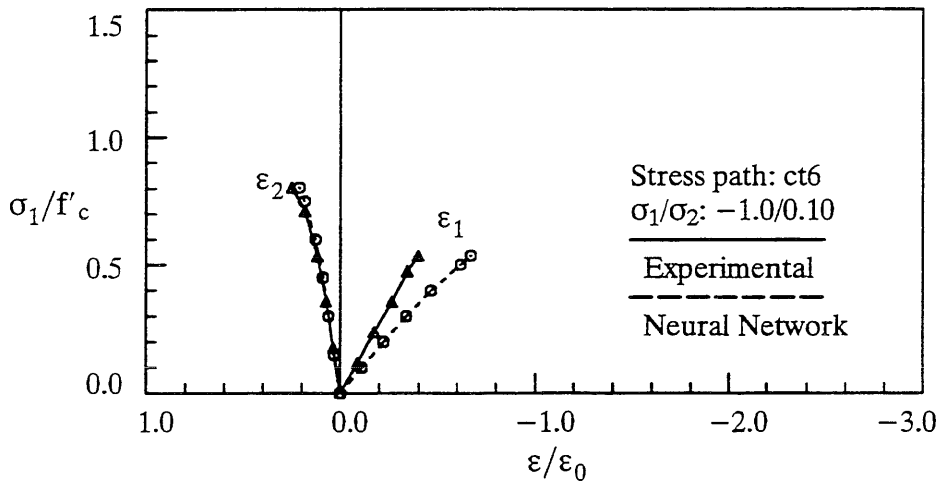
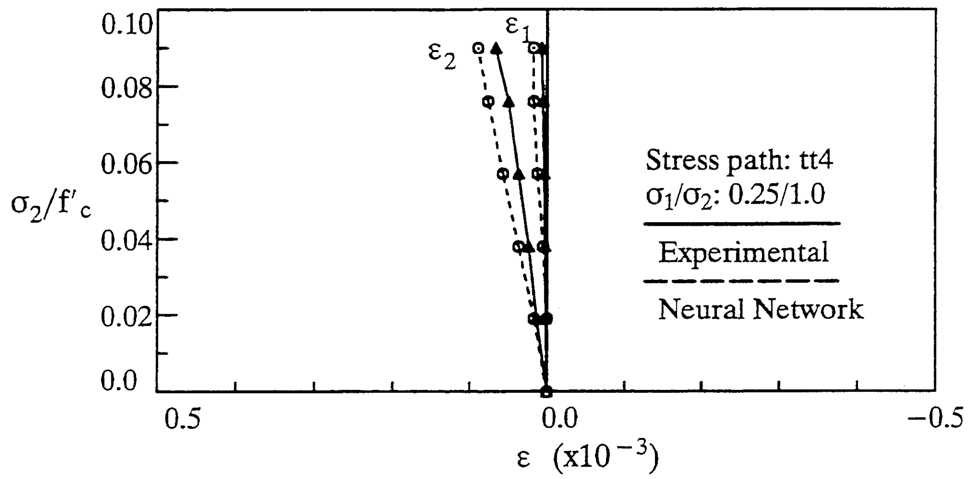
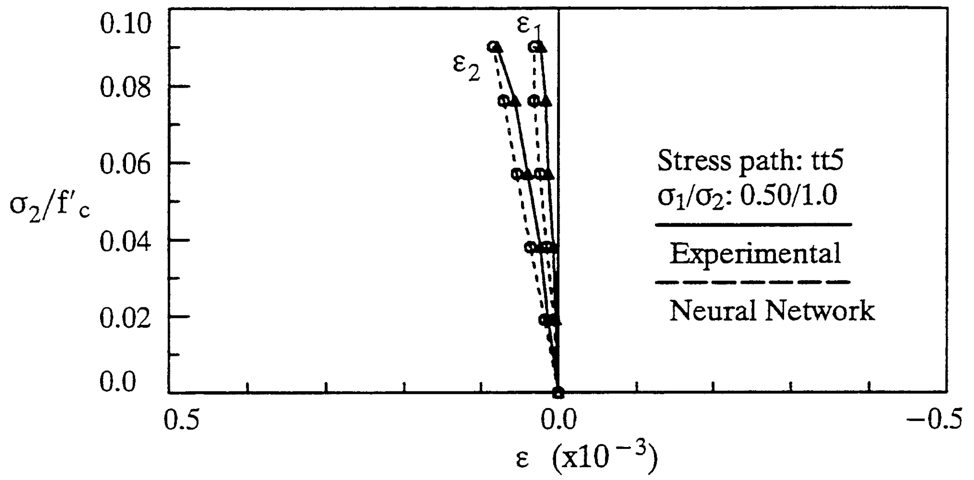


Fig. 4.9 – The Stress-Strain Relation Predicted by the Neural Network (Testing Results with Stress-controlled One-point Scheme)





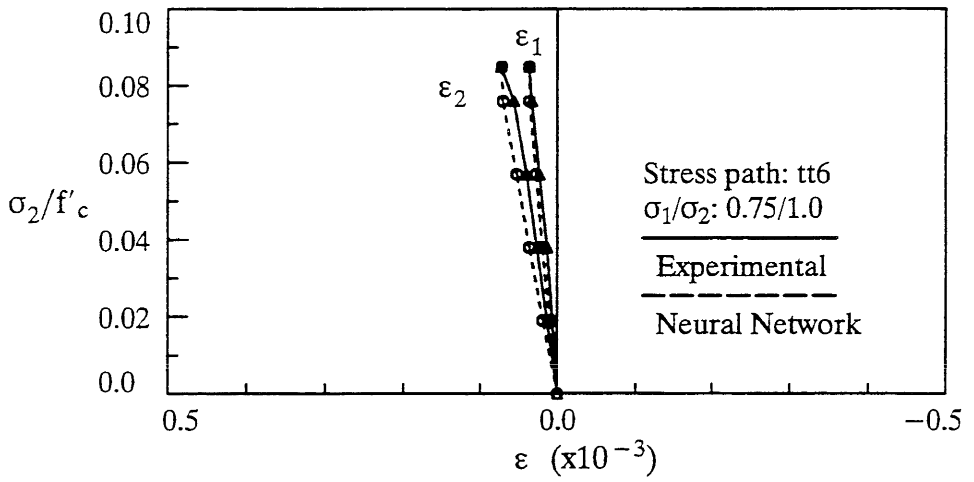


Fig. 4.10 – The Stress-Strain Relation Predicted by the Neural Network (Testing Results with Stress-controlled One-point Scheme)

4.3.2.2 A Strain-controlled Material Model with One-point Scheme

For the strain controlled model with one-point representation scheme, the architecture of the network consists of six input units representing two stresses, two strains and two strain increments (σ_1, σ_2, ε_1, ε_2, $\Delta\varepsilon_1$, $\Delta\varepsilon_2$), while the output units are two stress increments ($\Delta\sigma_1$, $\Delta\sigma_2$), as shown in Fig. 4.11. The hidden nodes are determined through adaptive node generation to be 30 nodes in each hidden layer when the maximum training error on the stress-strain relations corresponding to the 10 strain paths is reduced below 0.09.

It is interesting to note that the architecture for the strain-controlled model takes more hidden nodes than those needed in the stress-controlled model in order to capture the biaxial behavior of concrete to a comparable accuracy. This is basically caused by the simple way the training data are prepared, as the proportionality of loading is not apparent in the strain-controlled data, as it was for the stress-controlled data sets. The

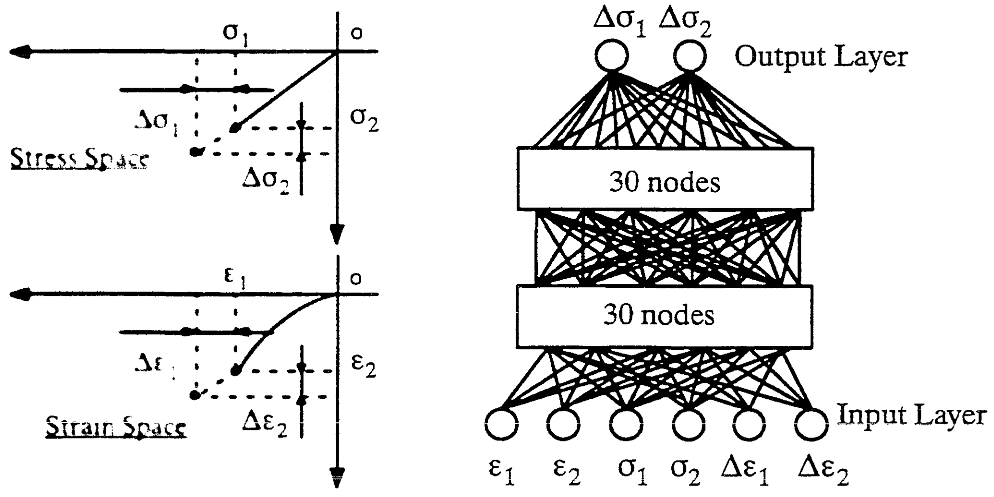


Fig. 4.11 – Architecture of the Strain-controlled Neural Network Material Model of Concrete in Biaxial Stress State

network has to use additional resources and more efforts to discover the proportionality by itself. This phenomenon suggests that a more elaborate way of preparing the training data would make the network smaller and the subsequent training more efficient.

The results after training the 10 stress-strain relations are illustrated in Fig. 4.12, in which the stress paths predicted by the neural network are shown as dashed lines, and those expected and used as training cases are shown in solid lines. The labelling of paths is the same as defined in Fig. 4.2. The graphs in Figs. 4.13 – 4.15 show the expected stress/strain curves (experimental results) against those predicted by the neural network. Note that with the neural network architecture described for the strain-controlled model, the network has been able to "learn" the biaxial behavior of concrete equally as well as for the stress-controlled model. With a strain-controlled model, because the error in the network prediction is shown in the stress part and the scaling schemes for stresses and strains are different, the error manifested in the overall stress–strain relation appears to be smaller than with a stress-controlled model.

The testing of the network proceeds in the same way as with a stress-controlled model, except the outputs predicted are stress increments instead of strain increments. The testing results in terms of stress-strain relations are shown in Figs. 4.17 – 4.19. From these three figures, it clearly shows that the stress-strain relations predicted by the network on the testing cases are reasonable and have good accuracy (maximum error < 0.10). In addition, the stress paths predicted on the testing cases, are shown in Fig. 4.16. Though the testing result on stress paths is not as accurate as that on strain paths with a stress-controlled model, the close agreement shown in the stress-strain relations predicted on all the testing cases indicates that the construction of a strain-controlled model is feasible by even using training data that were crudely prepared. Of course, if experimental data from strain-controlled tests are available, it can be expected that a neural network model trained with those data would be more natural to simulate the experimental results.

From the training and testing results of both the stress-controlled and the strain-controlled concrete model with one-point representation scheme, it is interesting to note that the representation scheme and the way the training data are prepared play important roles in determining the amount of work and efforts that are needed for the successful construction of the material model. In modeling the behavior of concrete in biaxial stress states, theoretically speaking, the information content embedded in the 10 stress/strain relations should be perceived as the same for both stress- and strain-controlled models. However, the larger number of nodes in hidden layers required for the strain-controlled model clearly indicate that a neural network "sees" the information in data differently with different representation schemes. Therefore, a good representation scheme with an appropriately processed training data set would considerably speed up the material model construction process.

In the next section, the training and testing results of neural network-based models of plain concrete in biaxial stress states, using three-point representation schemes, are presented.

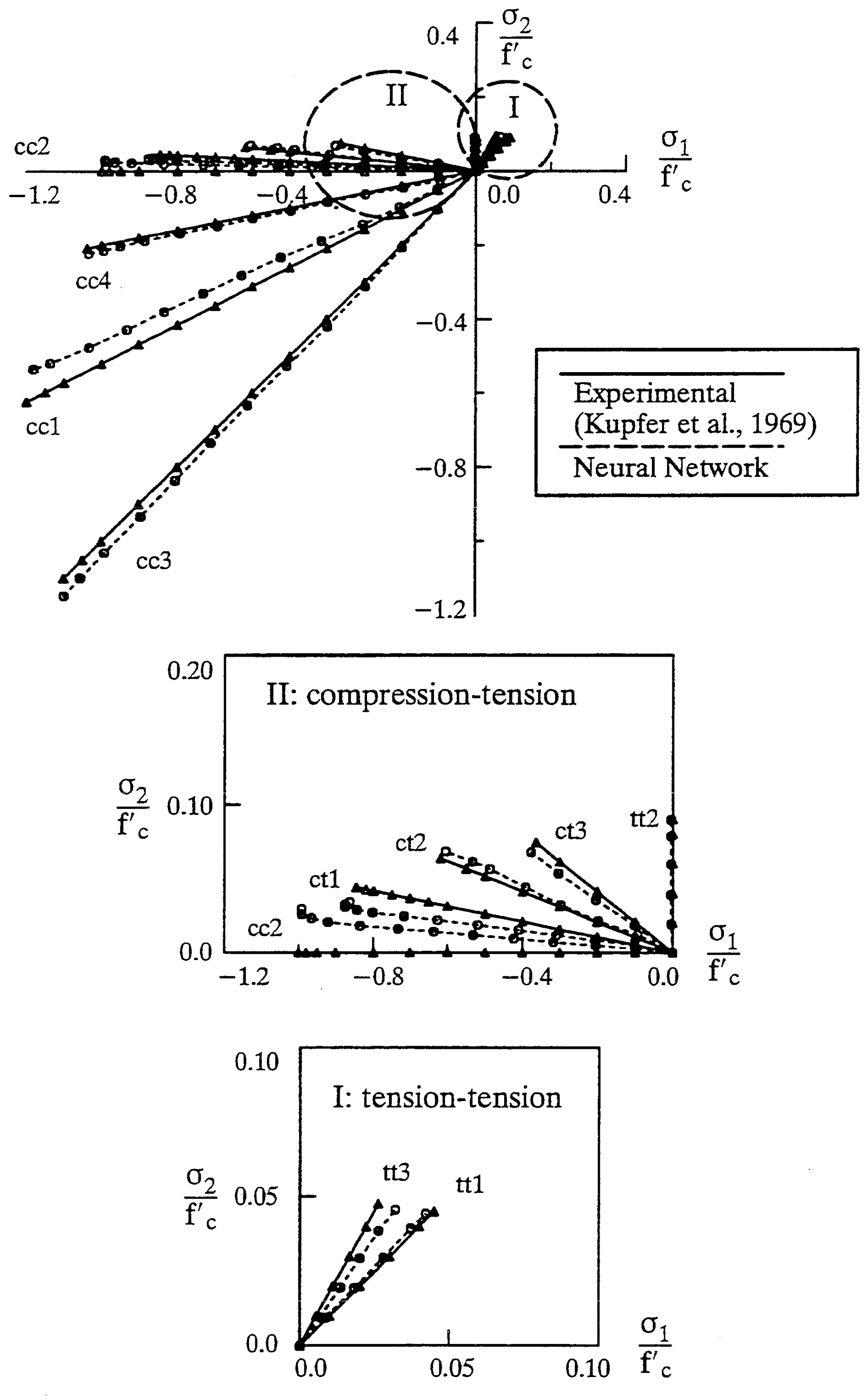


Fig. 4.12 – Predictions of the Neural Network on Stress Paths – Training Results (with Strain-controlled One-point Scheme)

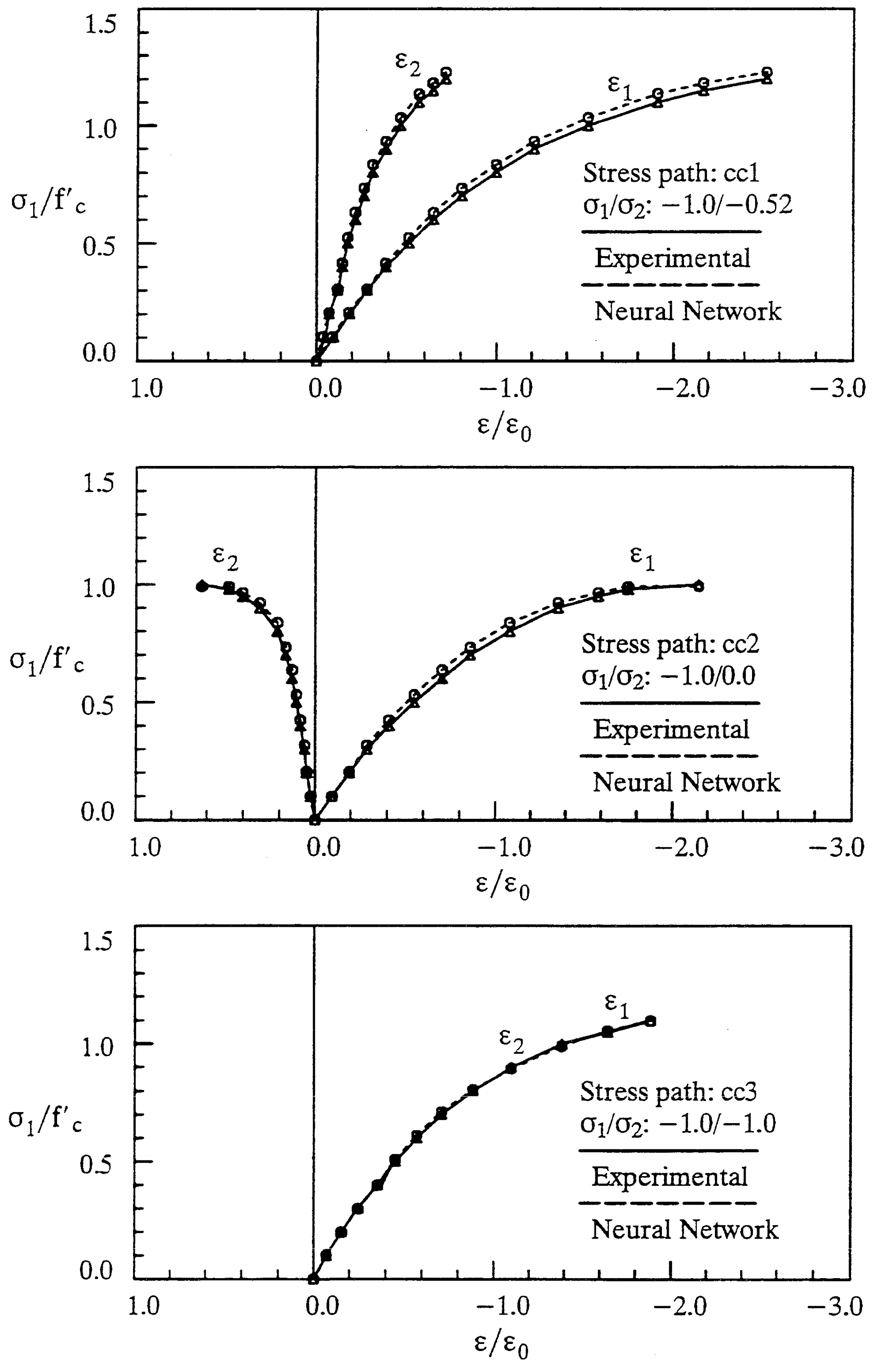
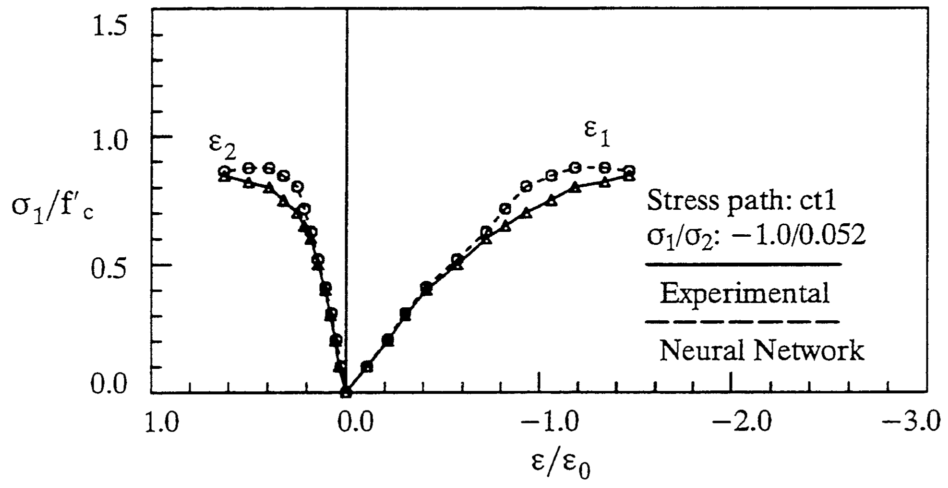
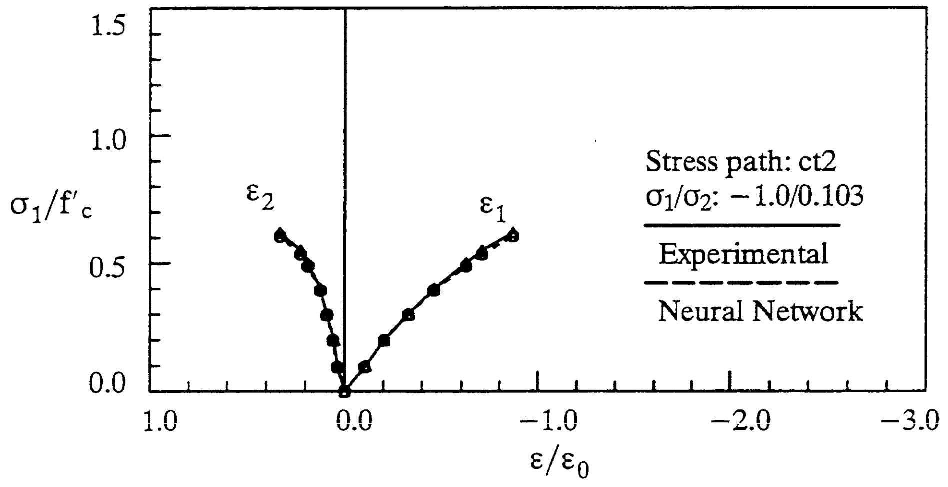


Fig. 4.13 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Strain-controlled One-point Scheme)





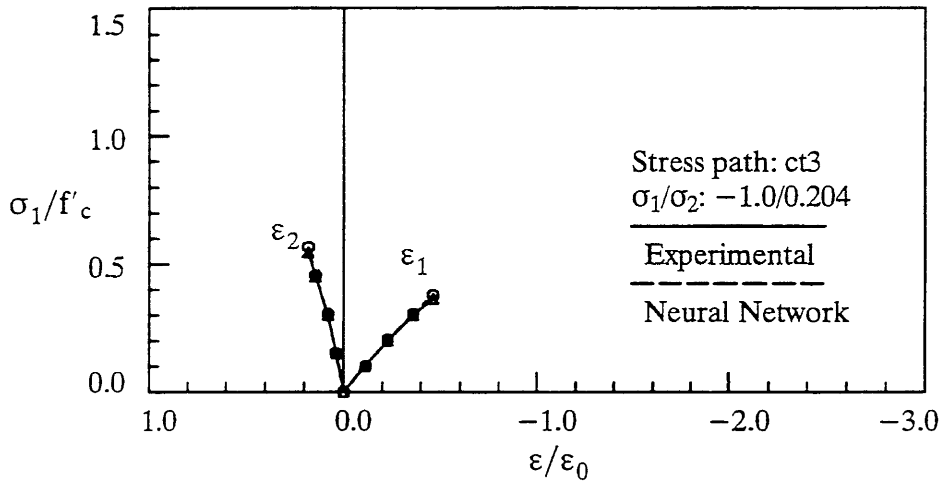


Fig. 4.14 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Strain-controlled One-point Scheme)

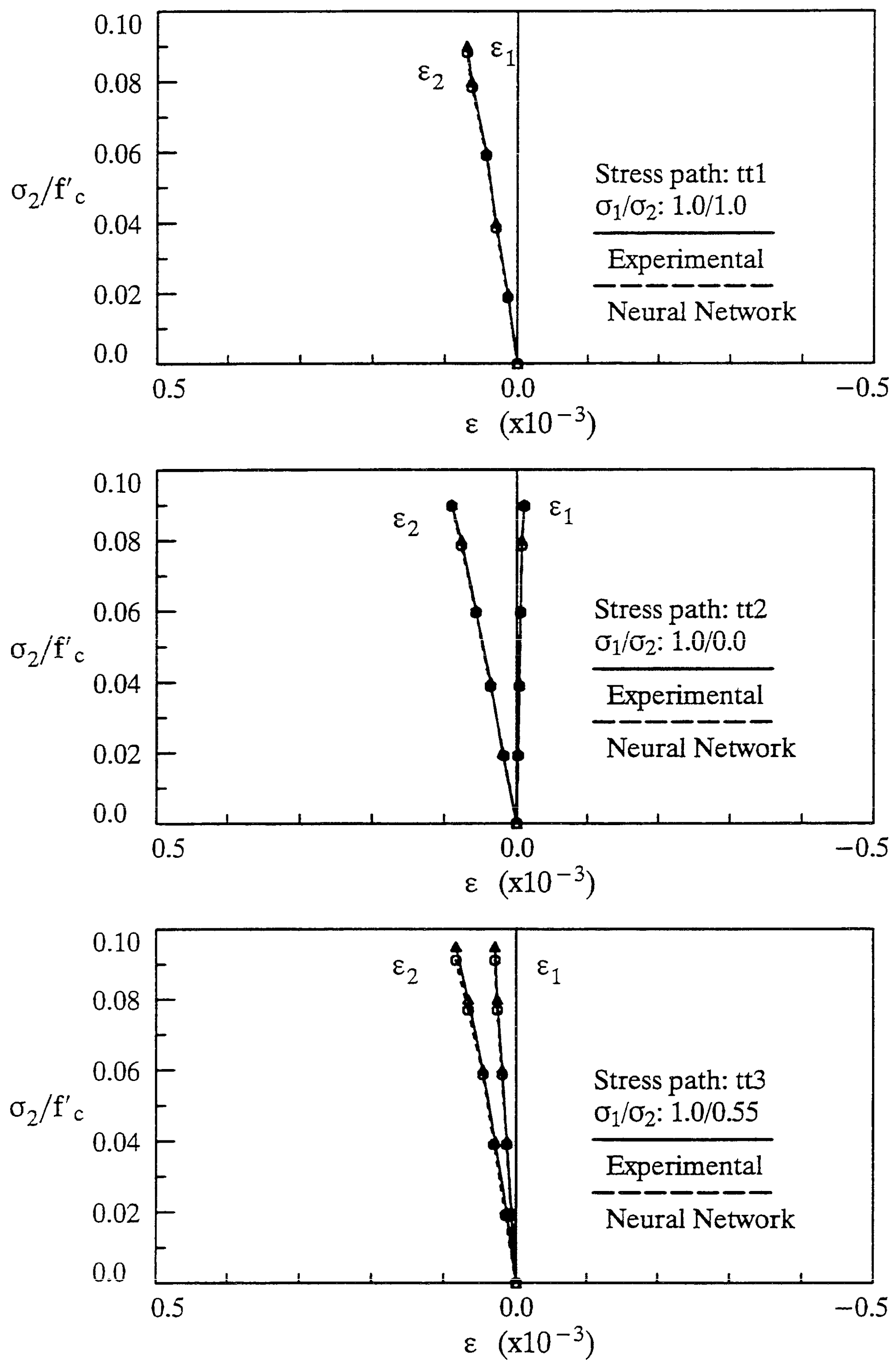


Fig. 4.15 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Strain-controlled One-point Scheme)

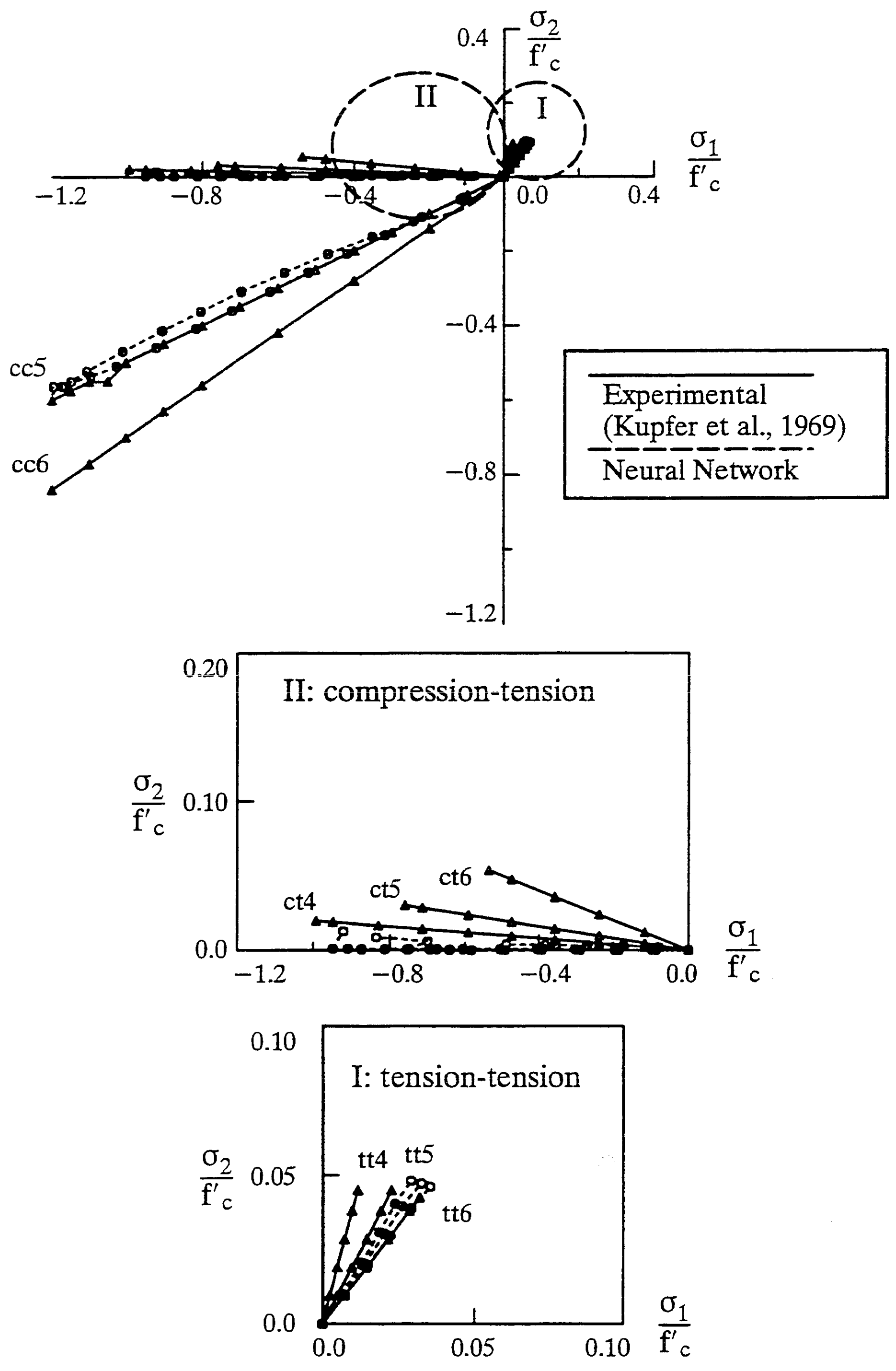
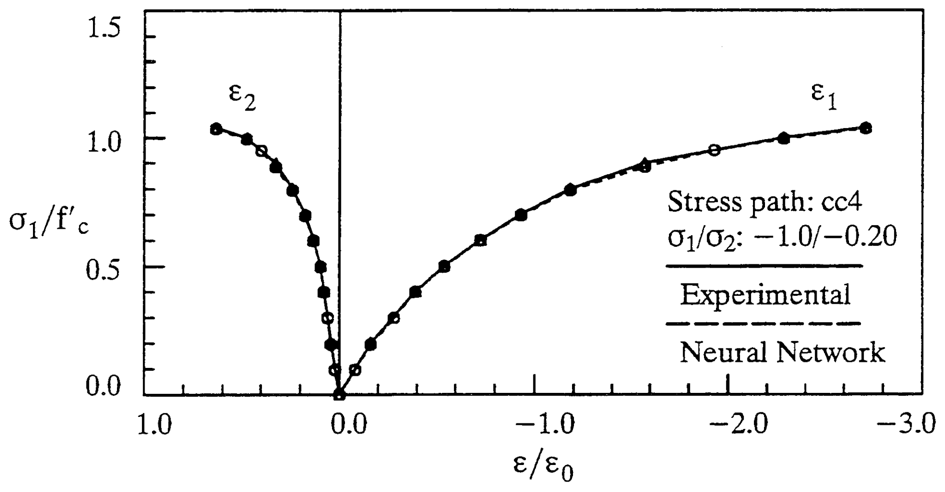
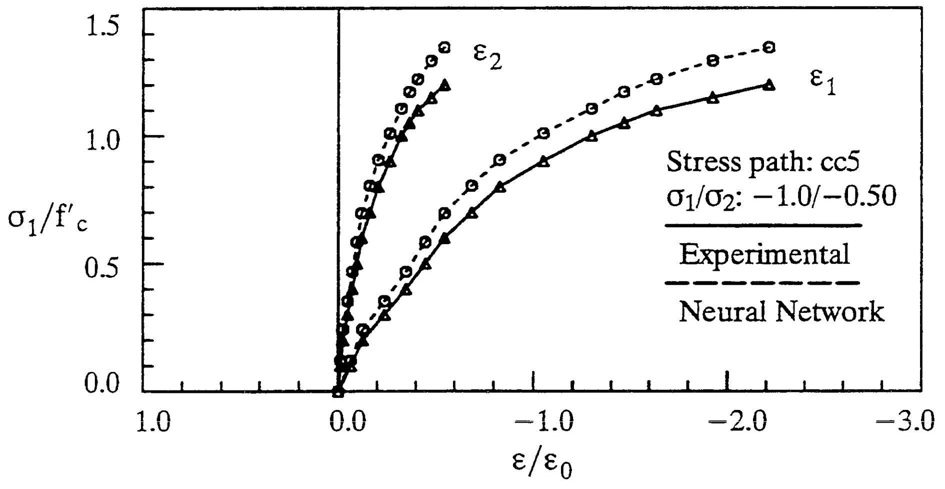


Fig. 4.16 – Predictions of the Neural Network on Stress Paths – Testing Results (with Strain-controlled One-point Scheme)





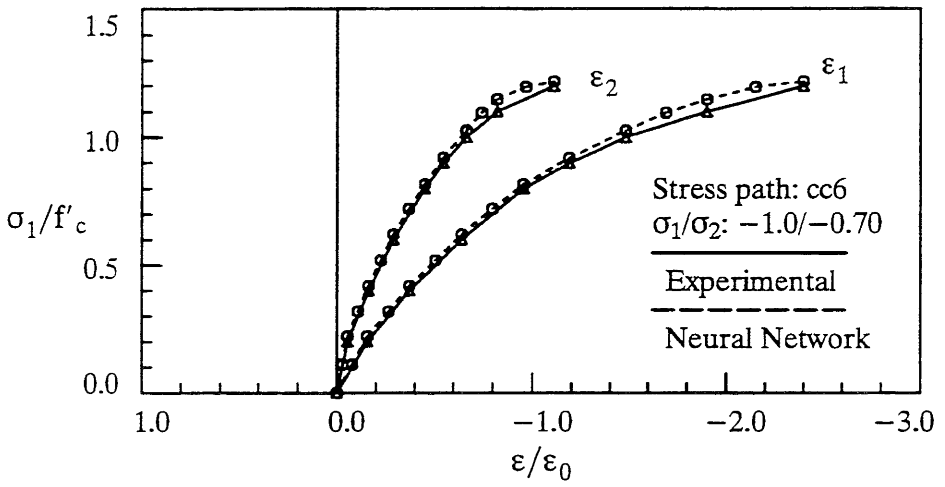
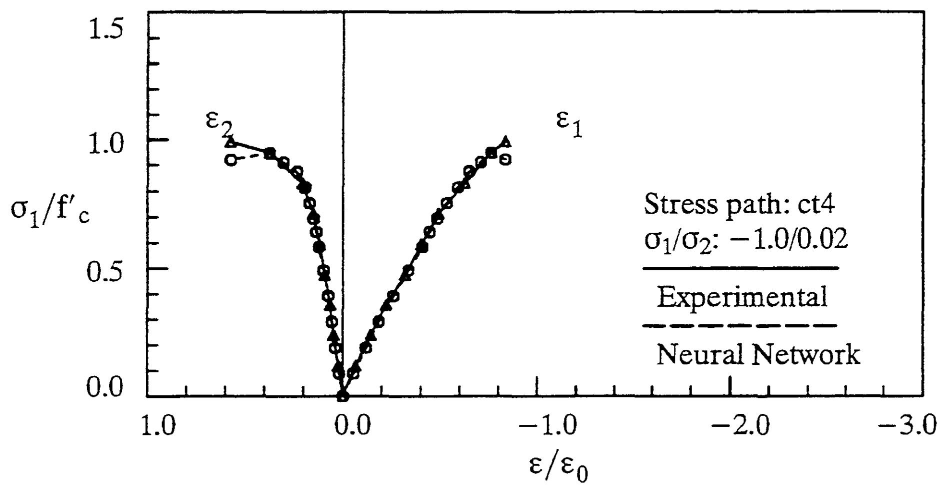
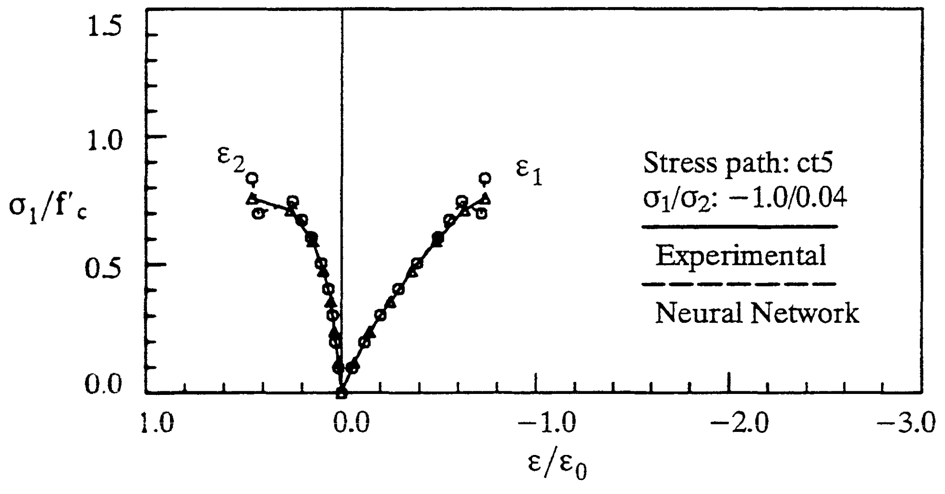


Fig. 4.17 – The Stress-Strain Relation Predicted by the Neural Network (Training Result for cc4 and Testing Results for cc5 and cc6)





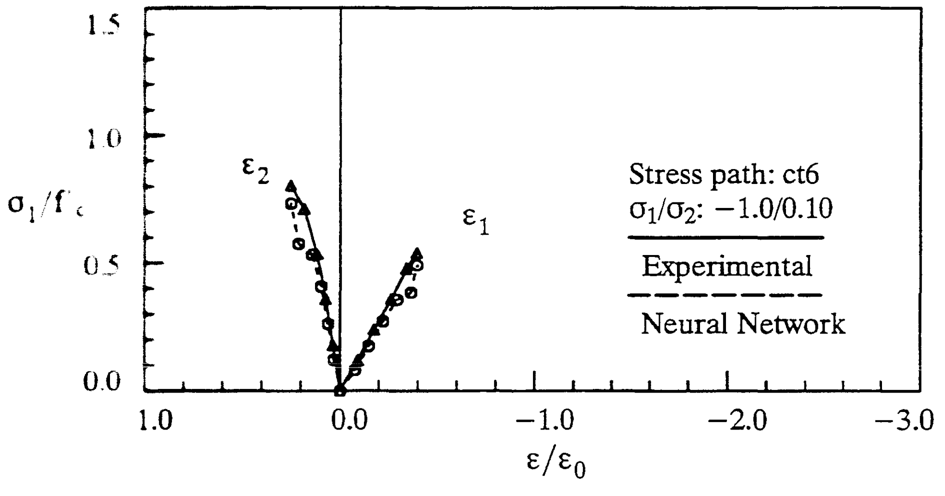
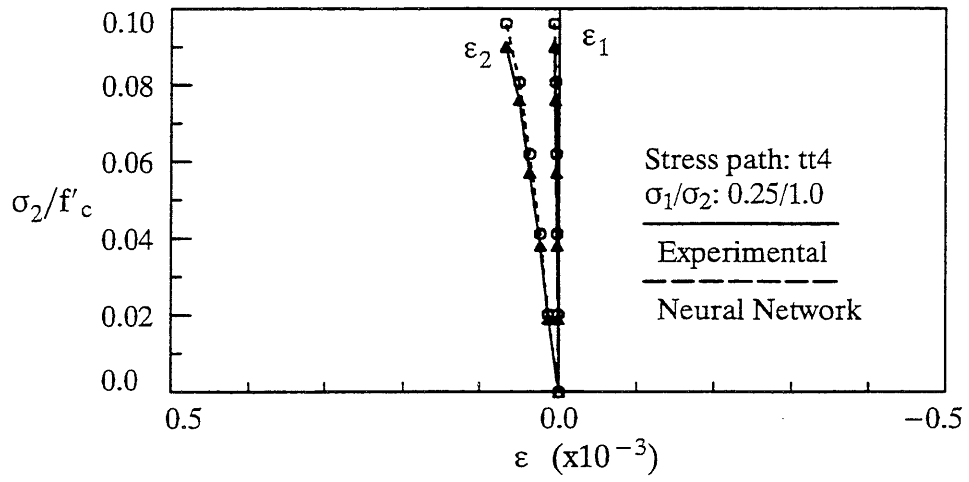
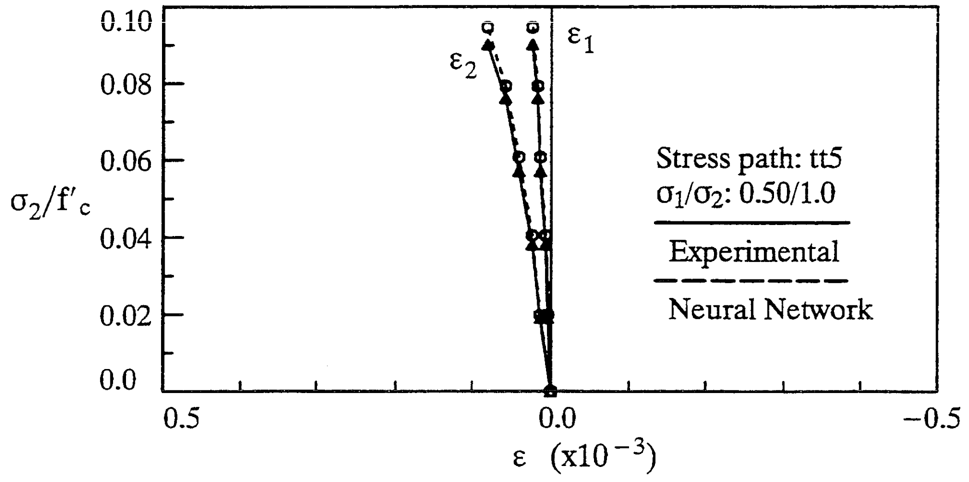


Fig. 4.18 – The Stress-Strain Relation Predicted by the Neural Network (Testing Results with Strain-controlled One-point Scheme)





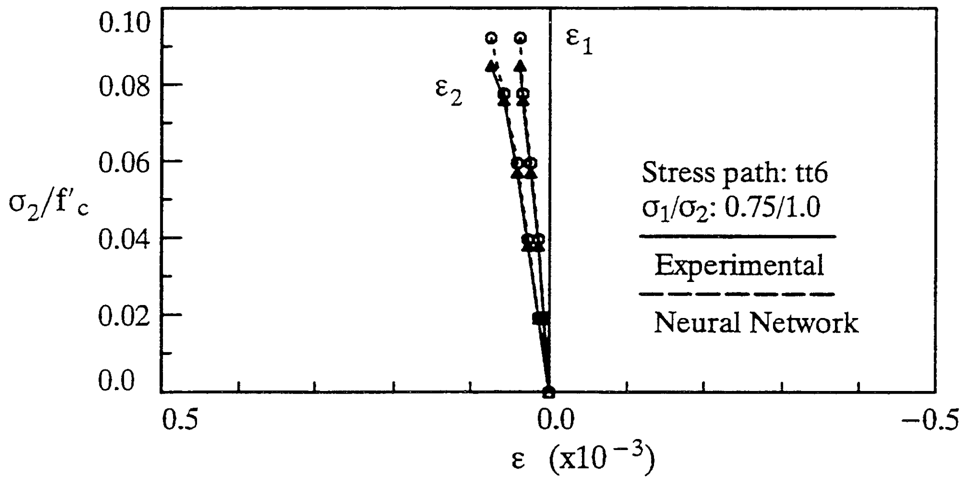


Fig. 4.19 – The Stress-Strain Relation Predicted by the Neural Network (Testing Results with Strain-controlled One-point Scheme)

4.3.2.3 A Strain-controlled Material Model with Three-point Scheme

To determine the effect of different representation schemes on the behavior of neural network material models, both a stress-controlled and a strain-controlled three-point scheme are also used to model the biaxial behavior of plain concrete with the expectation that the network may learn the stress-strain behavior of concrete better if more information is provided. However, after the training and testing were completed, the performance of these models with three-point scheme was almost the same as that with one-point scheme and no substantial improvement was observed. This phenomenon indicates that one-point representation scheme has enough information capacity within the network to characterize the biaxial behavior of concrete under monotonic proportional loading, and the information presented on the two previous stress-strain states with the three-point scheme appears to be redundant. To illustrate the behavior of neural network constitutive models of concrete built with three-point scheme, the training and testing results of a strain-controlled model is described in the following paragraphs.

For a strain-controlled model with three-point representation scheme, the input of the network is increased to 14 units ($\sigma_1(i-2)$, $\sigma_2(i-2)$, $\varepsilon_1(i-2)$, $\varepsilon_2(i-2)$, $\sigma_1(i-1)$, $\sigma_2(i-1)$, $\varepsilon_1(i-1)$, $\varepsilon_2(i-1)$, $\sigma_1(i)$, $\sigma_2(i)$, $\varepsilon_1(i)$, $\varepsilon_2(i)$, $\Delta\varepsilon_1$, $\Delta\varepsilon_2$), and the output has 2 units ($\Delta\sigma_1$, $\Delta\sigma_2$). The training data were then constructed according to this representation scheme. After training the stress–strain data corresponding to the 10 stress–strain paths used for one-point scheme, with the maximum training error being reduced to below 0.09, the hidden units in each of the two hidden layers were dynamically determined to be 24 nodes. The training results in terms of stress paths are shown in Fig. 4.20, in which the network prediction (in dashed lines) is plotted against the expected value (in solid lines). The training results are also presented in the stress-strain rela-

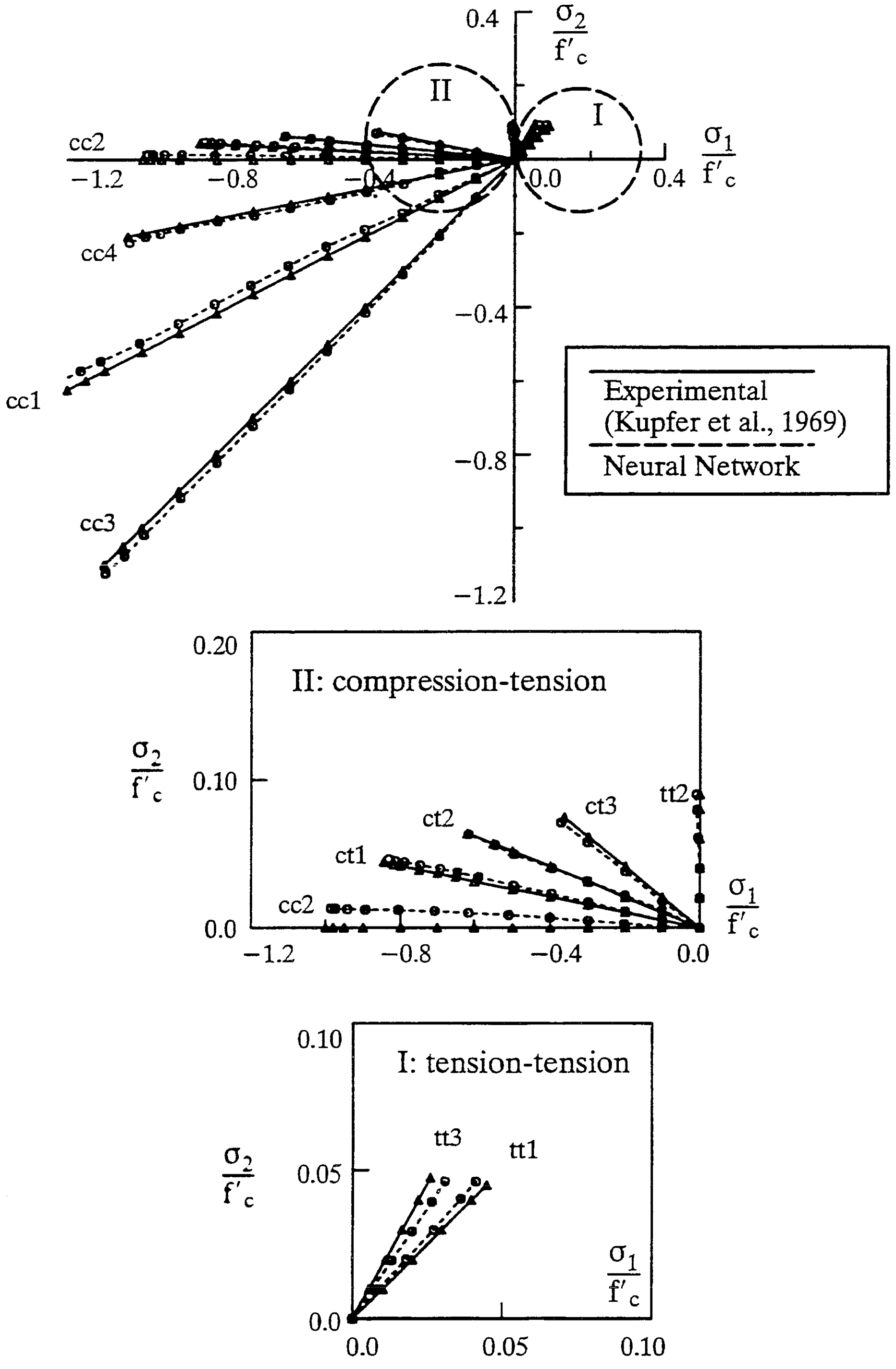


Fig. 4.20 – Prediction of the Neural Network on Stress Paths – Training Results (with Strain-controlled Three-point Scheme)

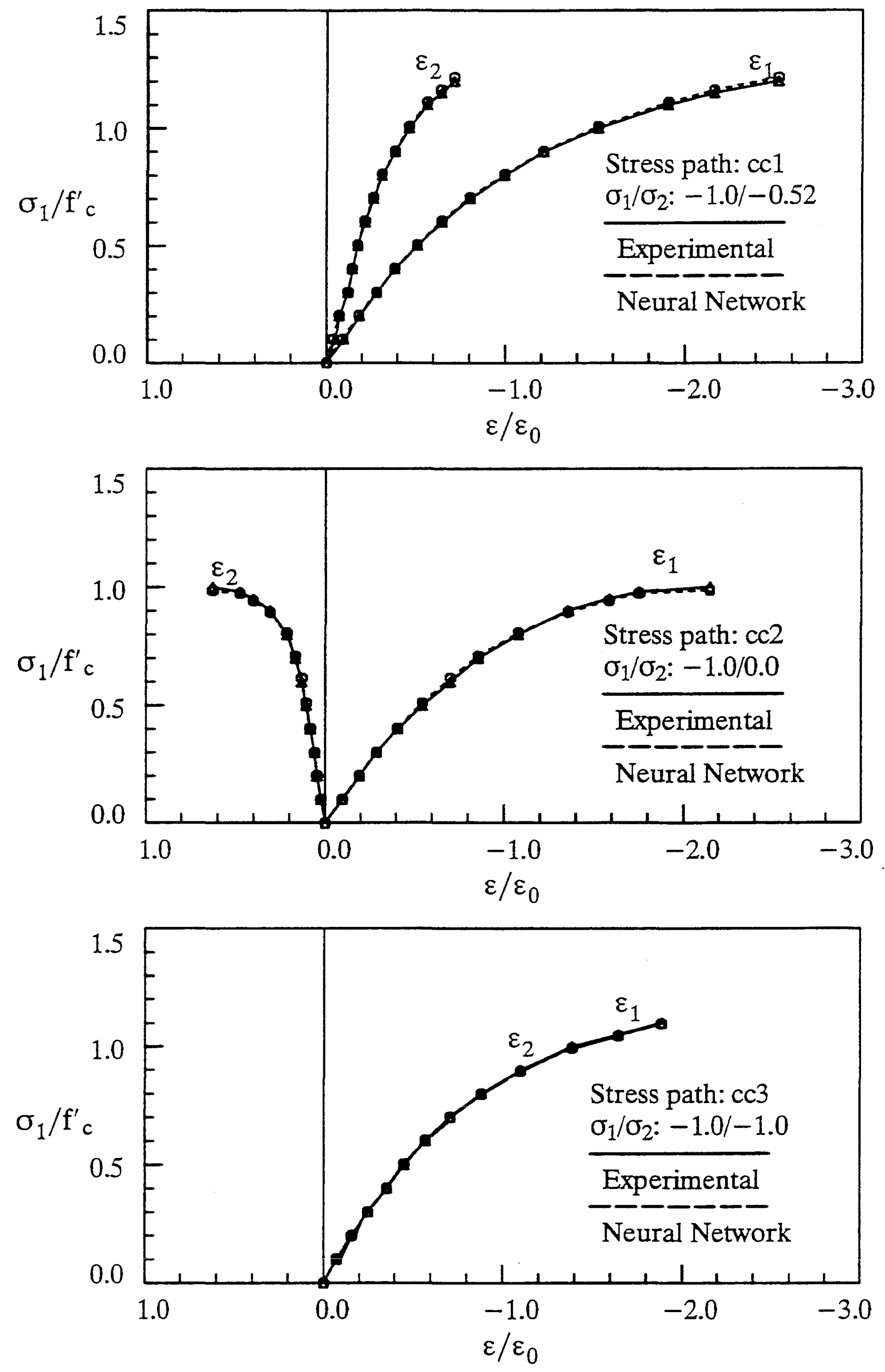
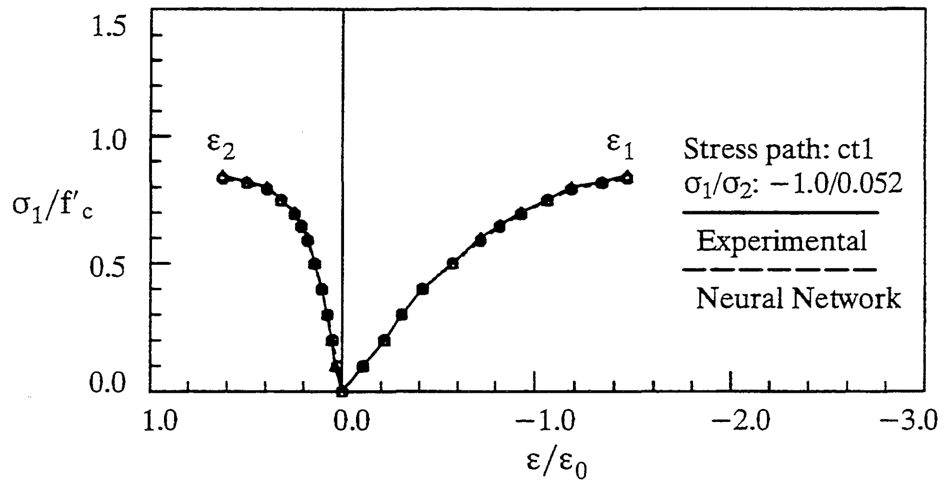
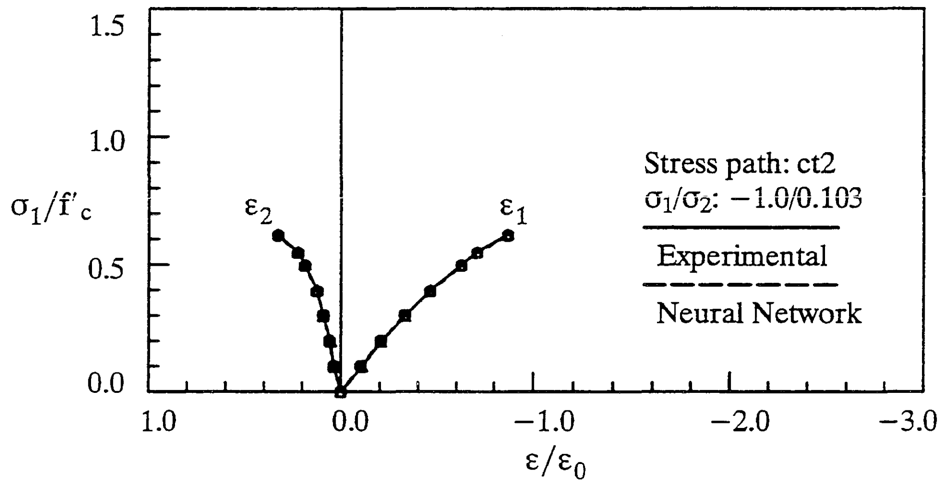


Fig. 4.21 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Strain-controlled Three-point Scheme)





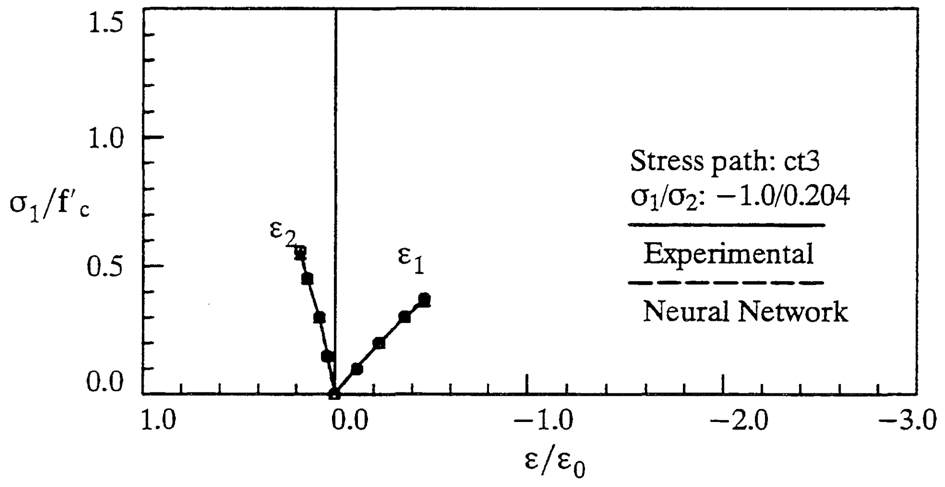
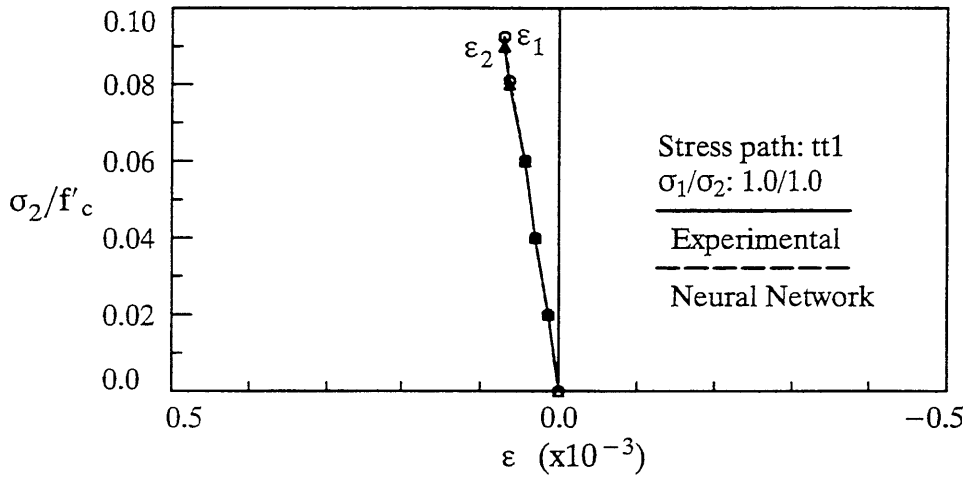
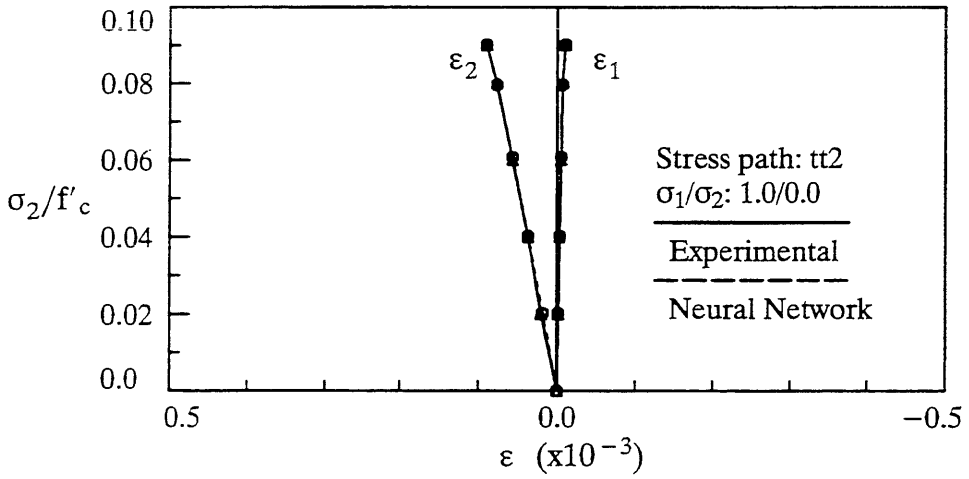


Fig. 4.22 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Strain-controlled Three-point Scheme)





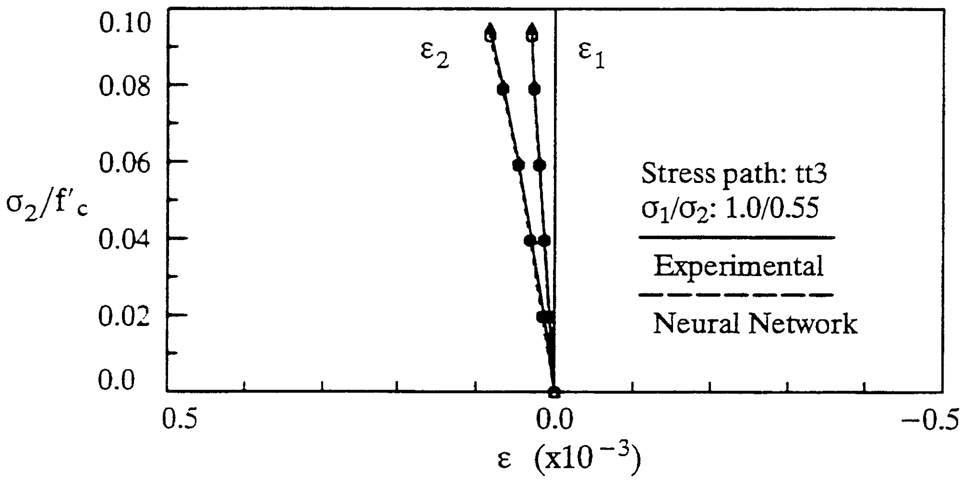


Fig. 4.23 – The Stress-Strain Relation Predicted by the Neural Network (Training Results with Strain-controlled Three-point Scheme)

tions, as shown in Figs. 4.21 – 4.23. From these figures, it is clear that the the network has learned the biaxial behavior of concrete presented in the training data with a slightly improved accuracy over that with one-point scheme.

Testing of the network with a three-point scheme proceeds in the same way as with one-point scheme. The testing results on the untrained cases were illustrated in Fig. 4.24, in which the predicted and expected stress paths were plotted. Besides, the stress-strain relations on the testing stress paths were shown in Figs. 4.25 – 4.27. The predictions of the current network on testing cases improved to certain degree over those of the strain-controlled model with one-point scheme, but the improvement was not substantial. For a stress-controlled model with three-point representation scheme, the training and testing results are essentially the same as those with strain-controlled model. That is, the stress–strain behavior captured in the neural network material model with both three-point schemes is essentially the same as that with one-point scheme. To avoid redundant presentation, the training and testing results of the stress-controlled model is not included here.

4.3.3 Plain Concrete in Biaxial Compression with Unloading

In the implementation of a material model within a finite element procedure, it is important for the material model to be able to handle all phases of stress states including loading and unloading. For an analytical material model, the material behavior during unloading can be assumed to be elastic with certain material parameters fixed at some values. For instance, in Ottosen's isotropic nonlinear elastic model, the unloading and reloading path takes secant values of Young's modulus and Poisson's ratio at the starting unloading point (Ottosen, 1979). However, in a neural network material model, the unloading mechanism can only be included in the model through an appropriate representation scheme and training on certain experimental data that have

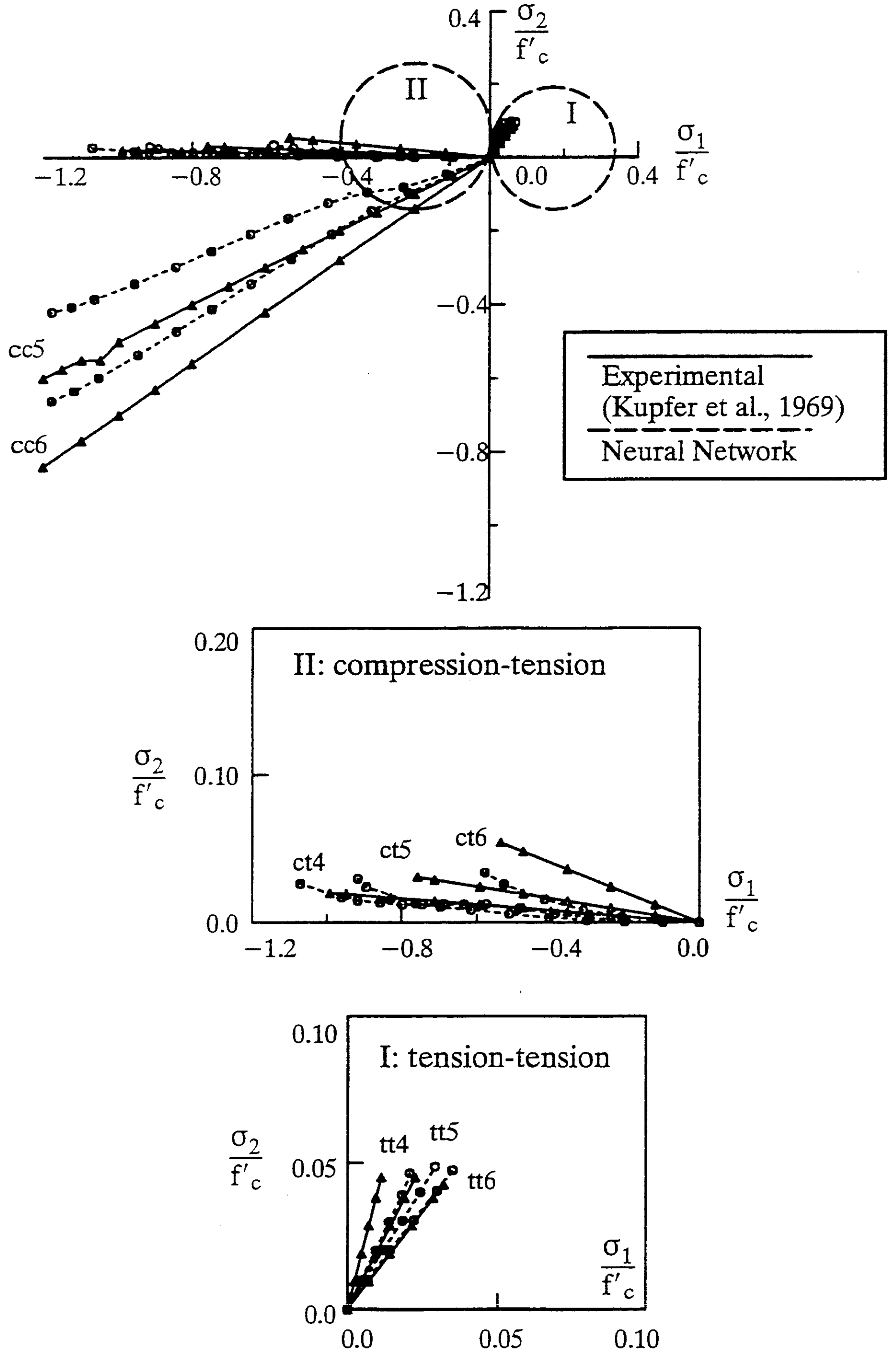


Fig. 4.24 – Prediction of the Neural Network on Stress Paths – Testing Results (with Strain-controlled Three-point Scheme)

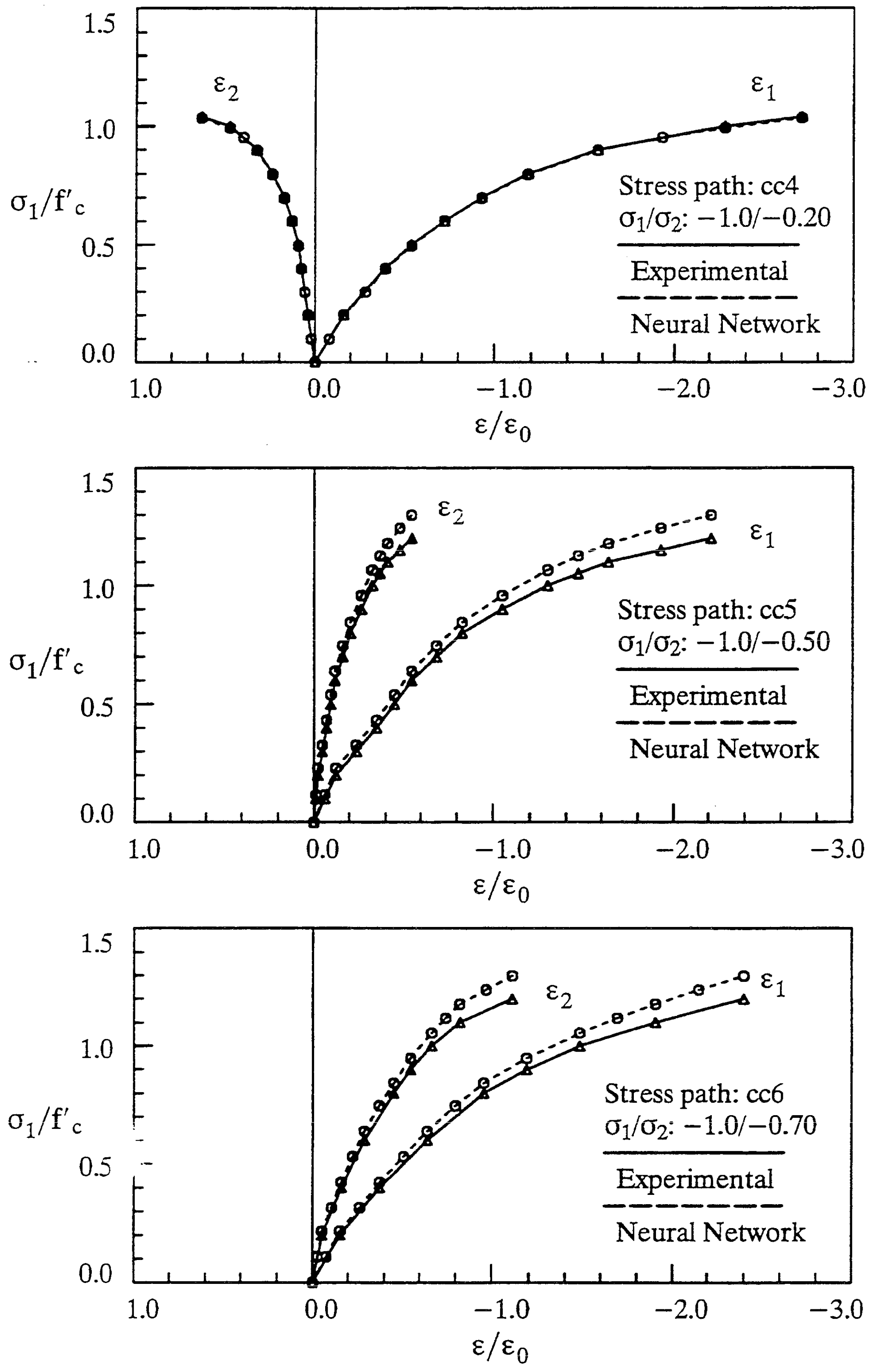
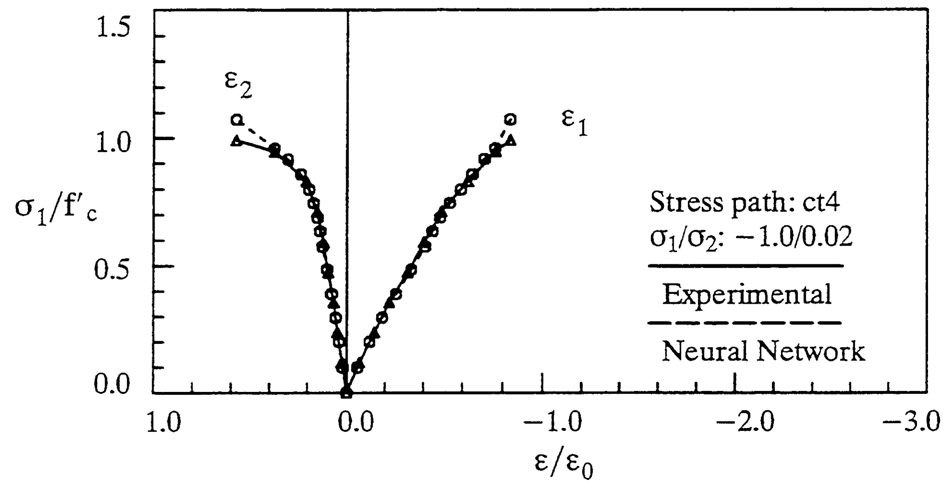
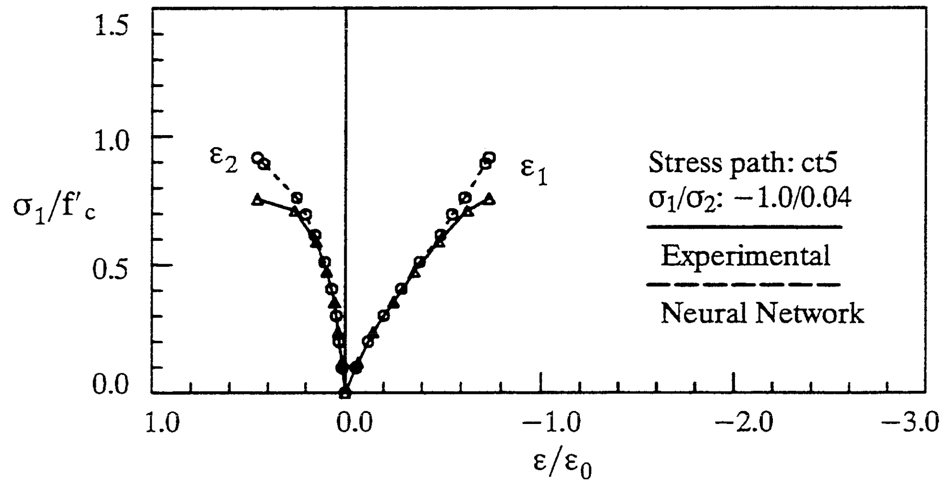


Fig. 4.25 – The Stress-Strain Relation Predicted by the Neural Network (Training Result for cc4 and Testing Results for cc5 and cc6)





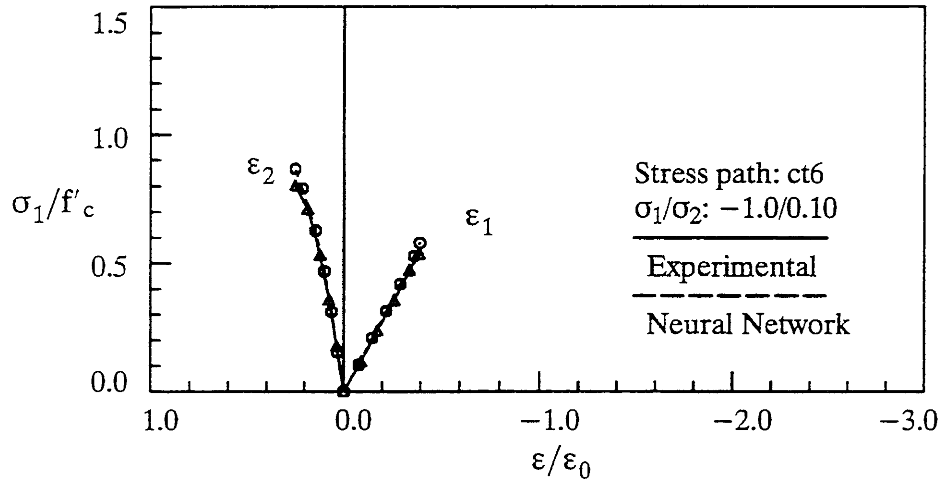
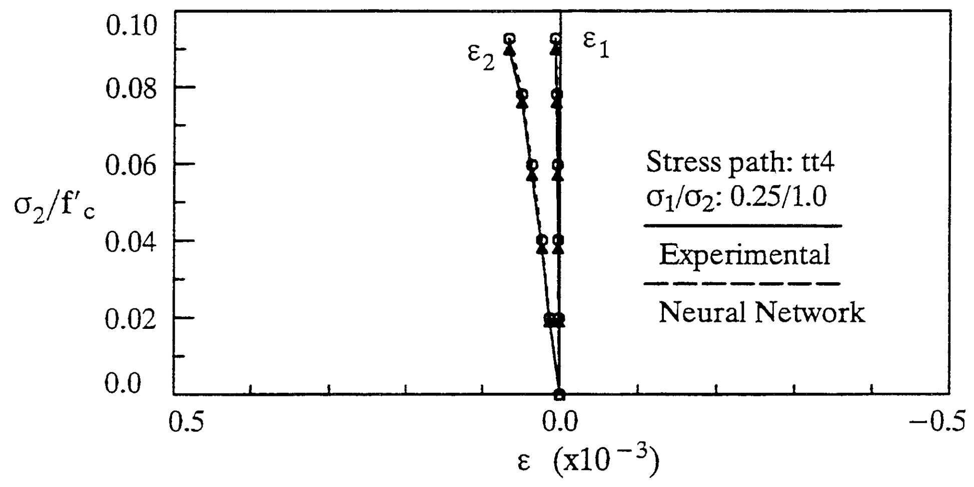
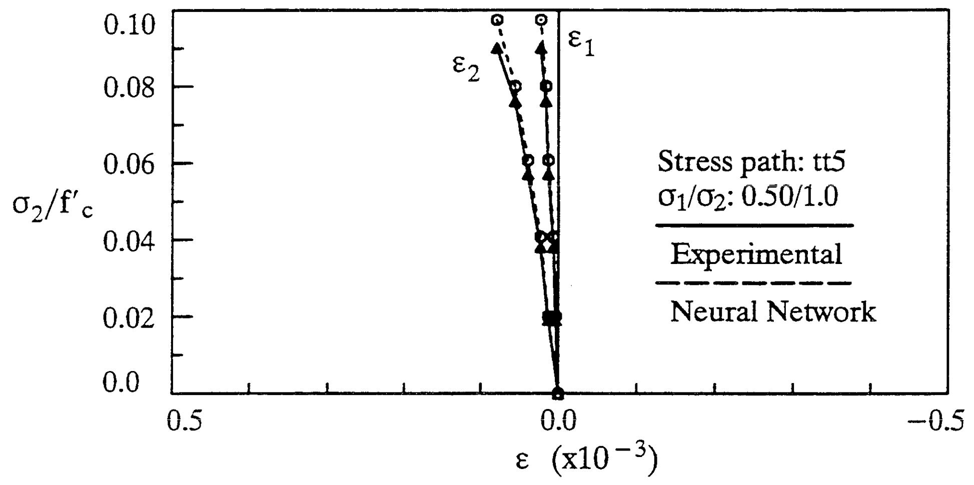


Fig. 4.26 – The Stress-Strain Relation Predicted by the Neural Network (Testing Results with Strain-controlled Three-point Scheme)





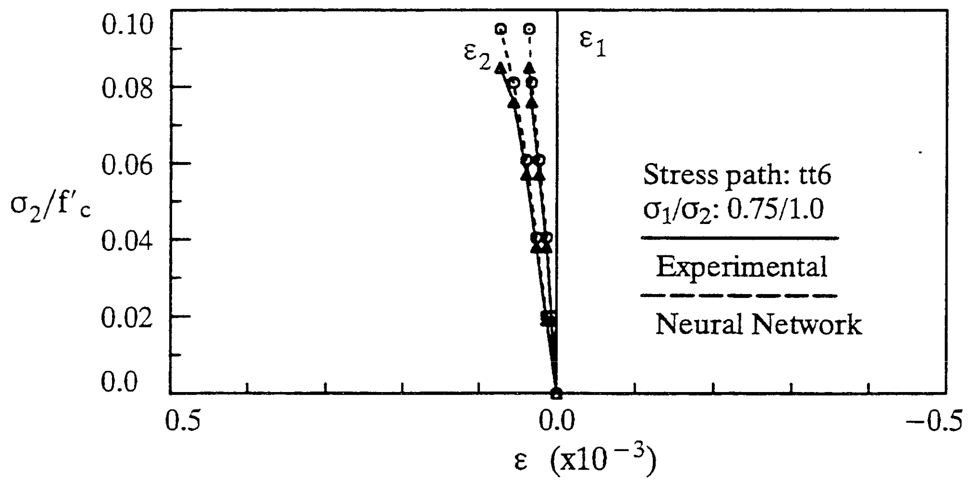


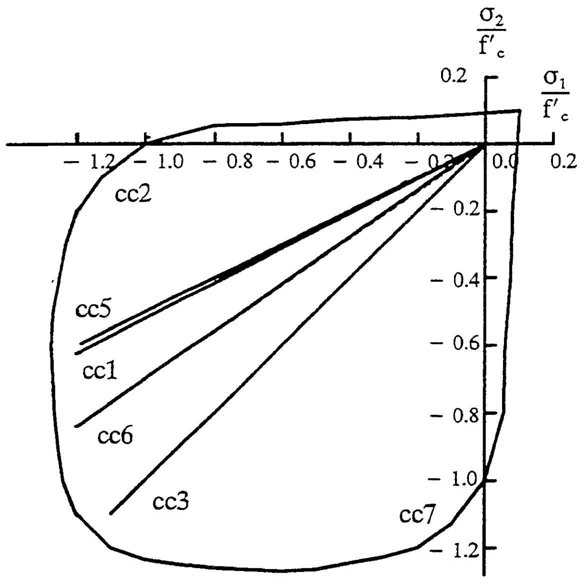
Fig. 4.27 – The Stress-Strain Relation Predicted by the Neural Network (Testing Results with Strain-controlled Three-point Scheme)

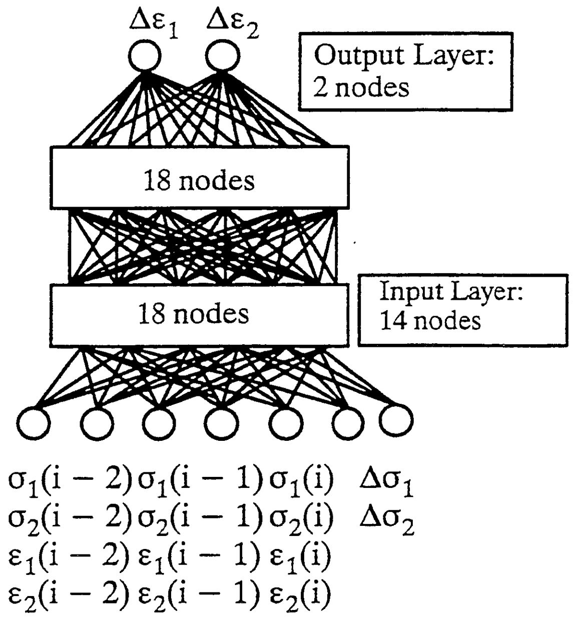
experienced unloading during testing. For concrete in a biaxial state of stresses, unfortunately, experimental data with unloading were not readily available for use in this study.

To illustrate the approach for the incorporation of unloading mechanism in the neural network-based material model, a set of pseudo-unloading data was generated by assuming elastic unloading with initial modulus of the loading path. Those data were generated only for biaxial compression cases. In the following sections, the representation scheme and the training and testing of the material model with elastic unloading in biaxial compression are described.

4.3.3.1 Representation Scheme and Architecture Determination

To capture the onset and continuation of unloading, it is important to include a portion of the loading history into the representation scheme. Therefore, a stress-controlled three-point scheme was selected for this problem. In the whole biaxial compression region, totally six stress–strain paths were chosen as training and testing cases, which included two uniaxial compression cases (σ_1/σ_2: $-1.0/0.0$ and $0.0/-1.0$) and 4 biaxial compression cases (σ_1/σ_2: $-1.0/-0.52$, $-1.0/-1.0$, $-1.0/-0.50$, $-1.0/-0.70$). For each stress path, it was determined that the training data be prepared with stress-strain data corresponding to two unloading cases where one starts unloading at a lower stress level ($0.60\ \sigma_1/f'_c$) and another at the highest stress level, and the testing case would be the one that starts unloading at an intermediate stress level ($0.80\ \sigma_1/f'_c$) on the stress path. With this design on the training and testing of unloading, the neural network approach thus becomes: if, after proper training, the network can capture the characteristics associated with the simple unloading mechanism, it should be able to give a reasonable prediction on the intermediate unloading cases. Subsequently, the final training data compiled consisted of 12 training cases with 2 stress-strain relations





$\Delta\varepsilon_1$ $\Delta\varepsilon_2$
Output Layer: 2 nodes
18 nodes
18 nodes
Input Layer: 14 nodes
$\sigma_1(i-2)\,\sigma_1(i-1)\,\sigma_1(i)$ $\Delta\sigma_1$
$\sigma_2(i-2)\,\sigma_2(i-1)\,\sigma_2(i)$ $\Delta\sigma_2$
$\varepsilon_1(i-2)\,\varepsilon_1(i-1)\,\varepsilon_1(i)$
$\varepsilon_2(i-2)\,\varepsilon_2(i-1)\,\varepsilon_2(i)$

Fig. 4.28 – Stress Paths and Architecture of the Stress-controlled Network Model of Concrete with Unloading in Biaxial Compression

for each stress path and 6 testing cases.

Similar to the strain-controlled model described in the previous section, in this stress-controlled model with unloading mechanism, the architecture of the network has 14 input units ($\sigma_1(i-2)$, $\sigma_2(i-2)$, $\varepsilon_1(i-2)$, $\varepsilon_2(i-2)$, $\sigma_1(i-1)$, $\sigma_2(i-1)$, $\varepsilon_1(i-1)$, $\varepsilon_2(i-1)$, $\sigma_1(i)$, $\sigma_2(i)$, $\varepsilon_1(i)$, $\varepsilon_2(i)$, $\Delta\sigma_1$, $\Delta\sigma_2$), and the 2 output units ($\Delta\varepsilon_1$, $\Delta\varepsilon_2$) representing the strain increments corresponding to current stress increments. Two hidden layers are also used, and the final hidden units were determined to be 18 nodes, when the training on the 12 training cases converged and the maximum error reduced to below 0.085. The stress paths used for training and testing as well as the final network architecture determined are shown in Fig. 4.28.

4.3.3.2 Training and Testing of the Network with Simple Unloading

Training on the 12 sets of stress-strain data with unloading proceeds incrementally as with all the other models built so far, and the convergence of learning is reached when the maximum training error is below 0.085. At the onset of training, the initial network starts with 10 hidden nodes in each of the two hidden layers. During the whole training process, the node generation scheme was activated for 4 times with an increment of 2 hidden nodes at each time. Thus the final architecture consists of 18 hidden nodes in each hidden layer. The training and testing results represented as stress-strain relations for each stress path are shown in Figs. 4.29 – 4.34, in which the network prediction is plotted as dashed lines and the training data as solid lines. Note that in each figure, the training results of the two unloading cases are shown in the top and bottom graphs (graphs (a) and (c)), and the testing result of the intermediate unloading case is shown in the middle graph (graph (b)). For example, Figs. 4.29 (a) and (c) represent the training results for the two training cases on the stress path cc1 ($\sigma_1/\sigma_2 = -1.0/0.0$), and Fig. 4.29 (b) shows the testing result.

It is obvious from these training results that the network has captured the unloading mechanism presented in the training data, that is, the unloading stress-strain paths predicted by the network are almost identical to or at least parallel with the training unloading paths. This is very encouraging because it indicates that a neural network is able to learn certain mechanical mechanisms so long as they can be appropriately represented in the training data. The testing results on the untrained intermediate unloading case for all the six stress paths are shown in Figs. 4.29 (b) – 4.34 (b). The stress-strain relations predicted by the network on the testing unloading cases show remarkably good agreement with those expected from the designated unloading mechanism.

During the course of this study, numerical experiments were also conducted with another scheme in preparing the training data with unloading, using only the stress-strain data corresponding to the unloading case where one started unloading at the highest stress level. Thus the training data set consisted of 6 sets of stress-strain data, and the size of the testing data set became 12. After the network was successfully trained, tests on the remaining 12 unloading cases were performed. Even though those training and testing results were not shown here, it was observed that the testing results on the 6 untrained intermediate unloading cases were reasonable, but the results with those cases where unloading started at a lower stress level showed larger discrepancies. This experiment also indicated the importance of incremental training with testing on generalization in the determination of a quasi–minimal training data set.

From the training and testing of this unloading mechanism, it can be expected that if a set of experiment data on the unloading behavior of concrete in biaxial stress states is available, the neural network should be able to learn the behavior through training on this set of data. Similarly, the same elastic unloading mechanism can also be trained for concrete in the stress states of biaxial tension and tension-compression.

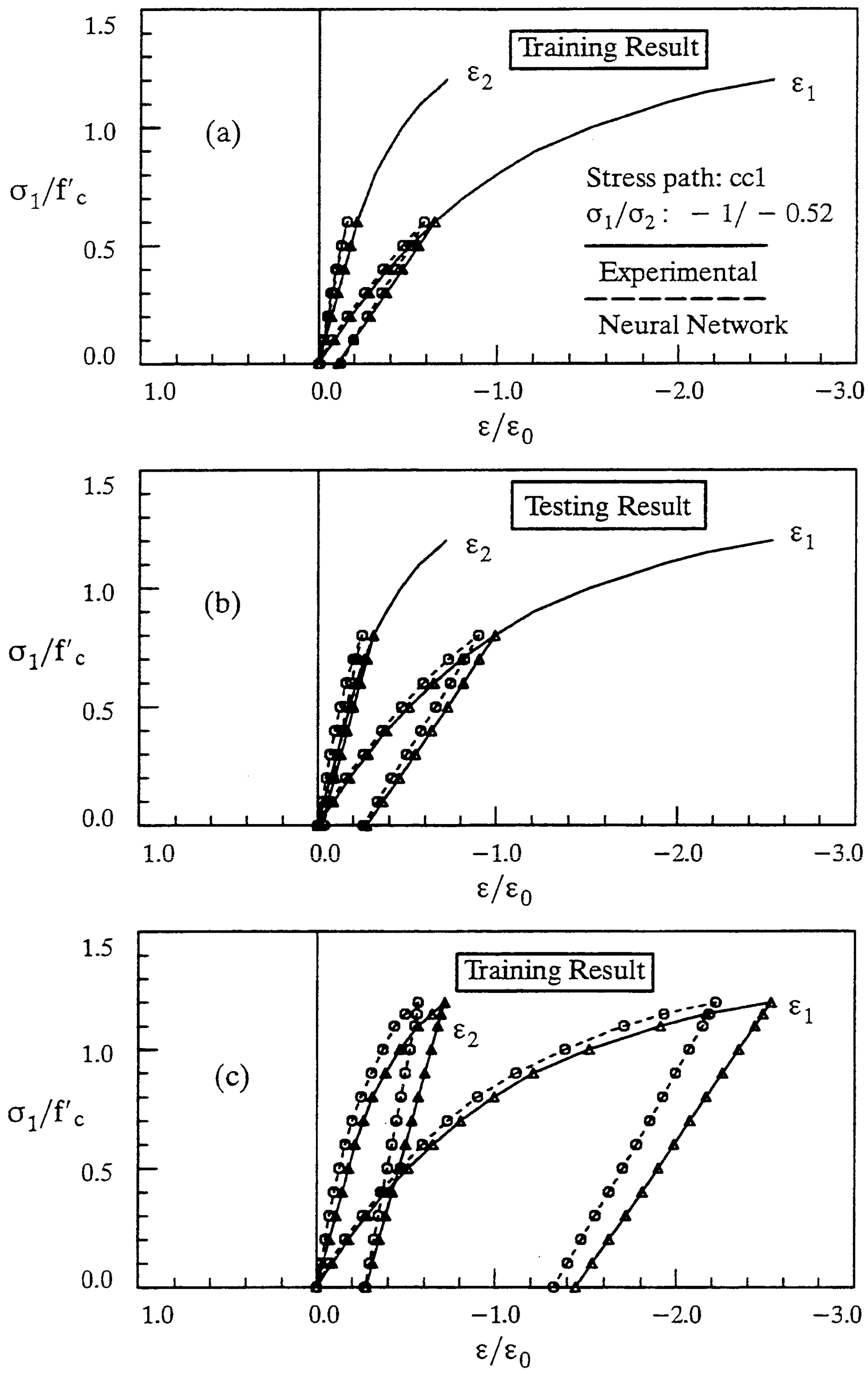
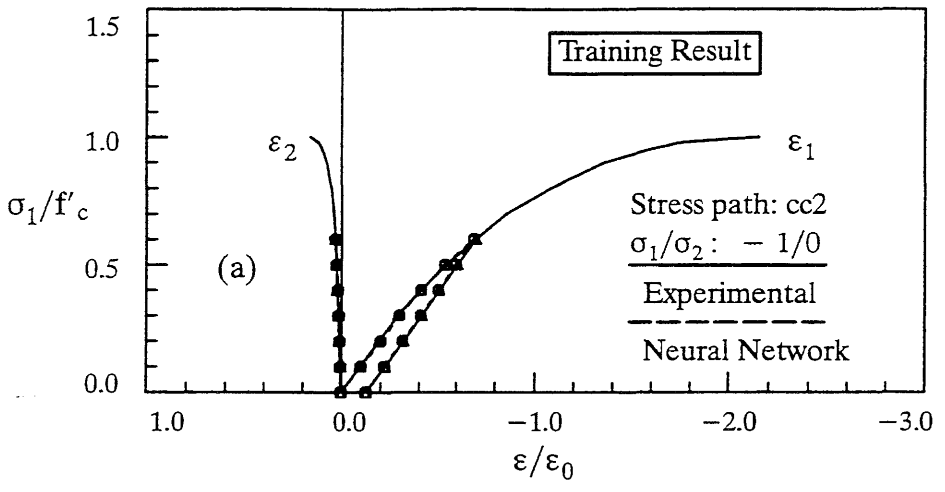
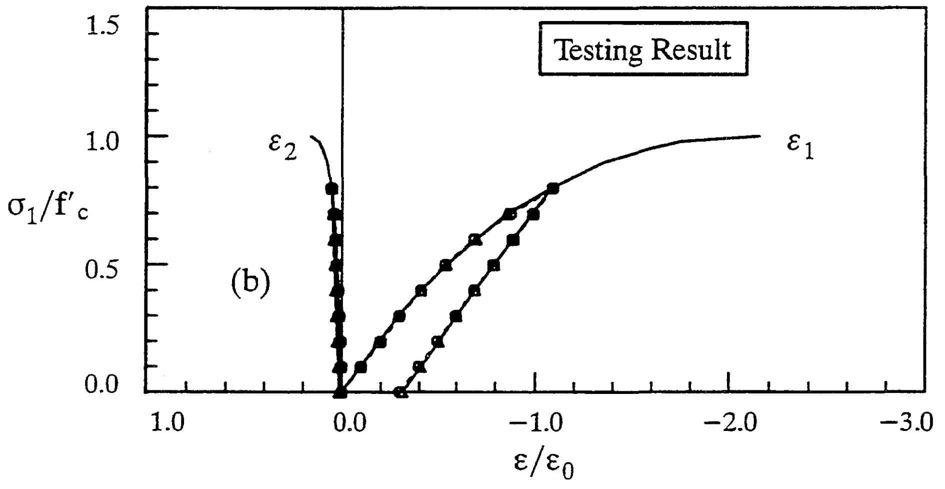


Fig. 4.29 – The Stress-Strain Relation Predicted by the Neural Network on Stress Path cc1 (with Elastic Unloading)





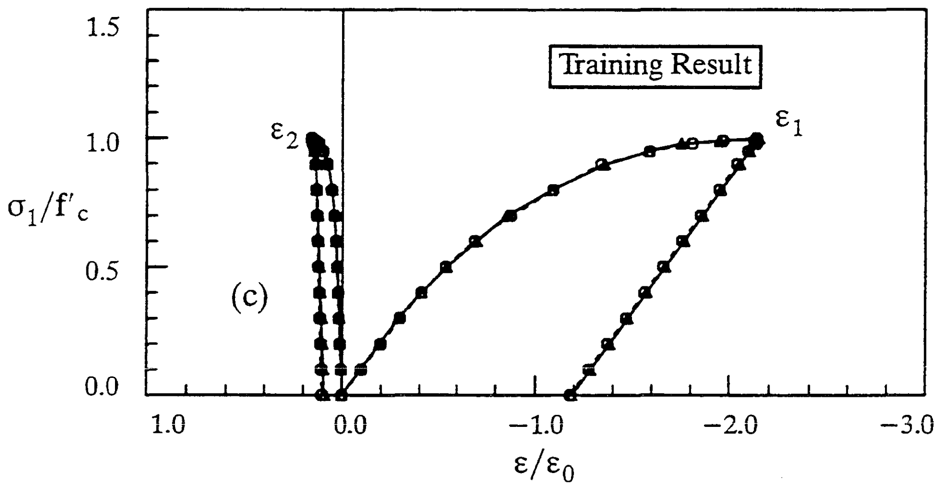
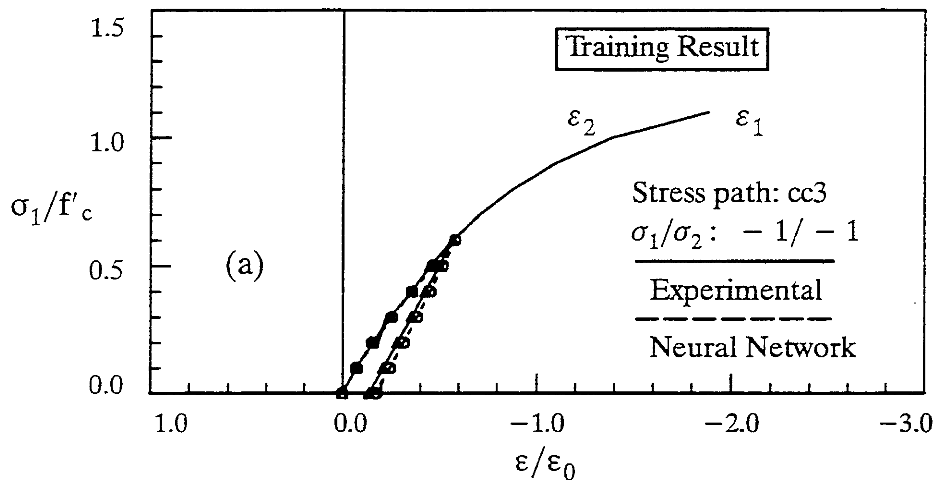
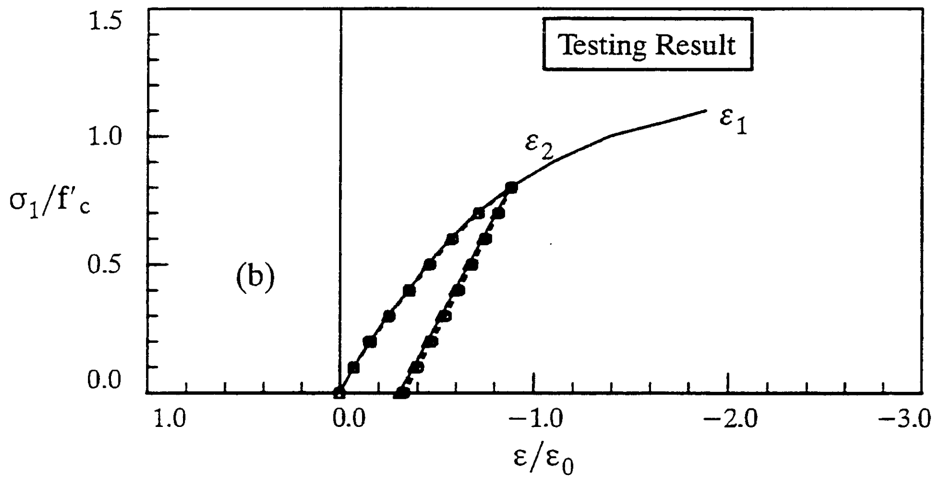


Fig. 4.30 – The Stress-Strain Relation Predicted by the Neural Network on Stress Path cc2 (with Elastic Unloading)





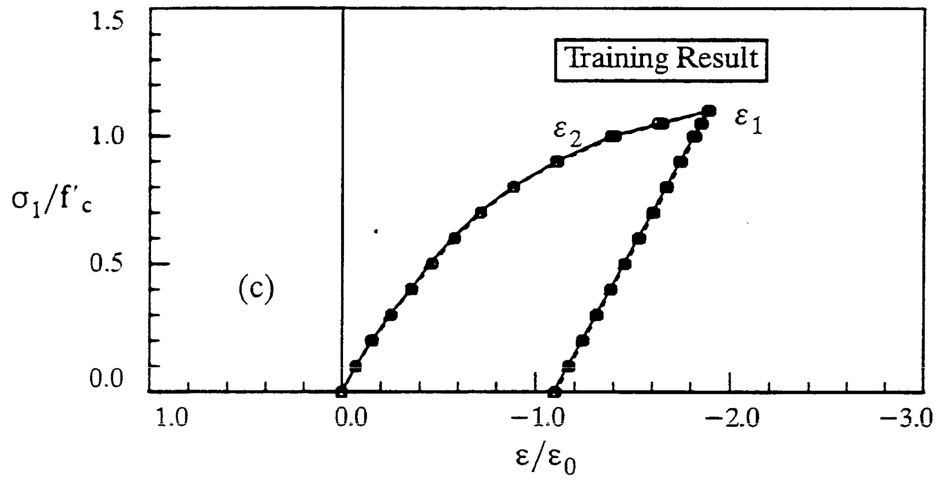


Fig. 4.31 – The Stress-Strain Relation Predicted by the Neural Network on Stress Path cc3 (with Elastic Unloading)

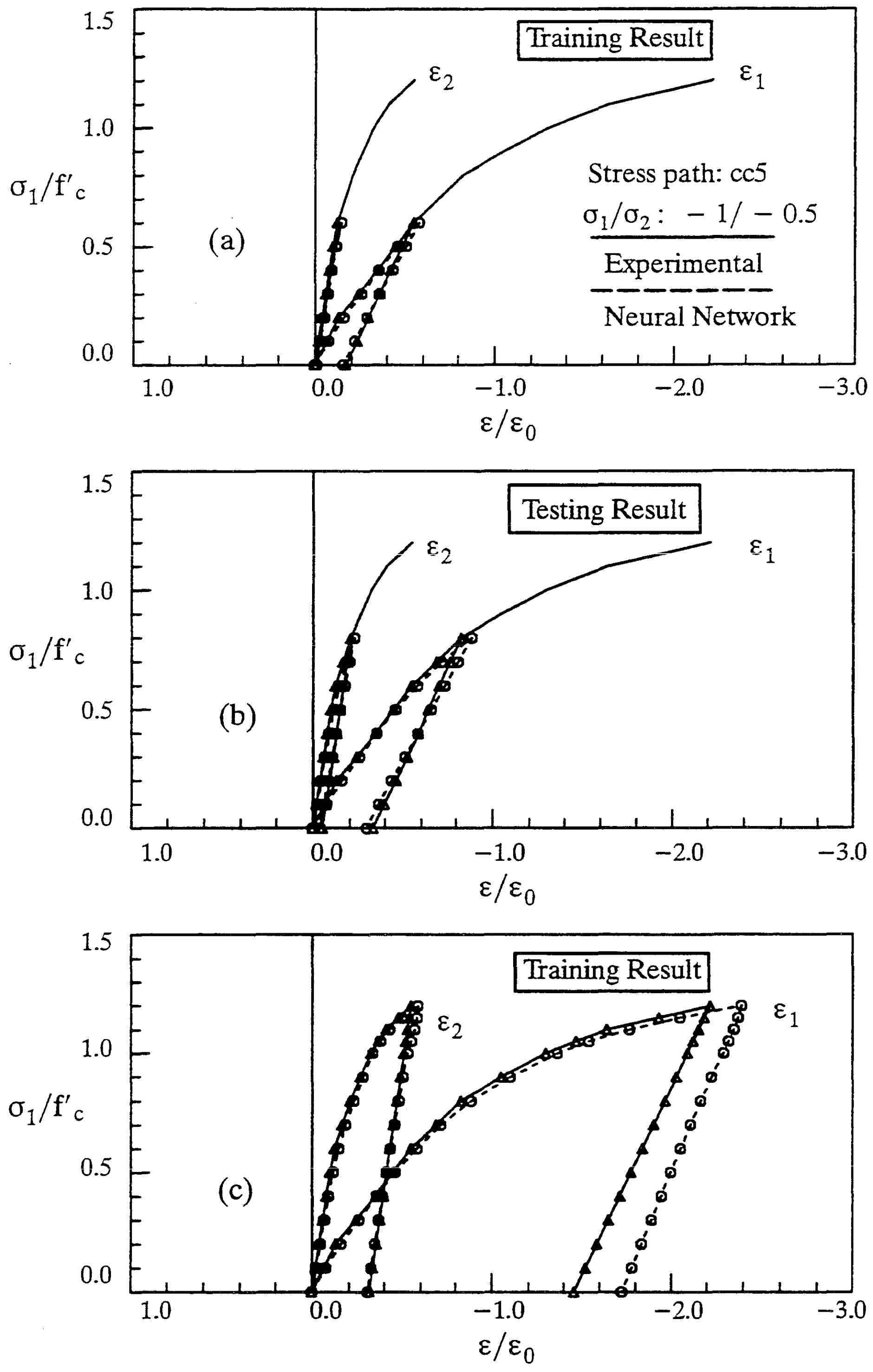
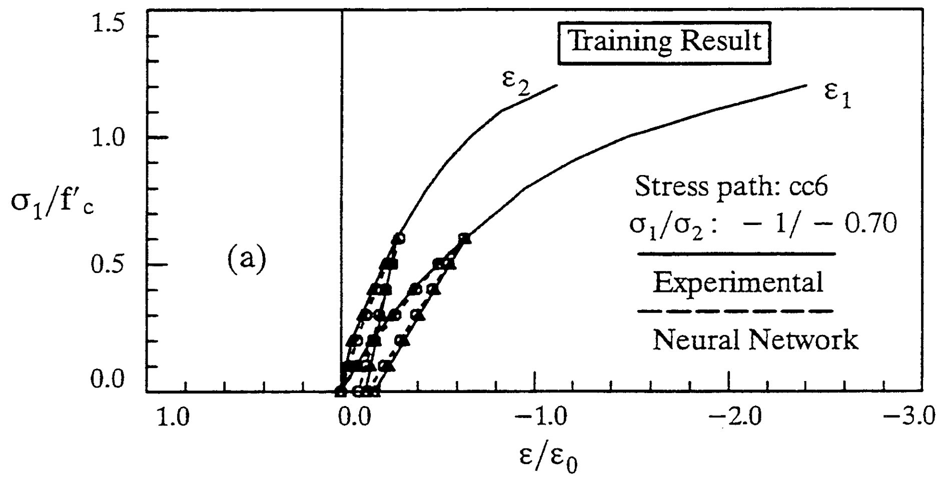
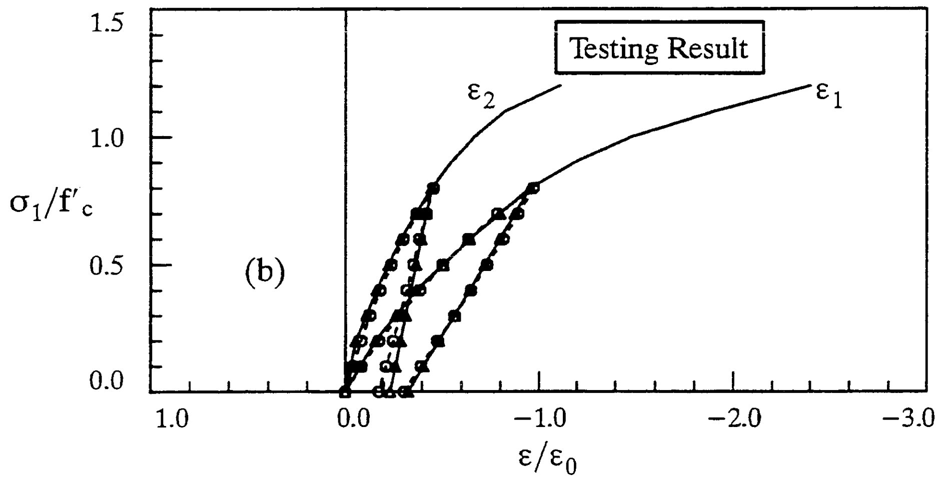


Fig. 4.32 – The Stress-Strain Relation Predicted by the Neural Network on Stress Path cc5 (with Elastic Unloading)





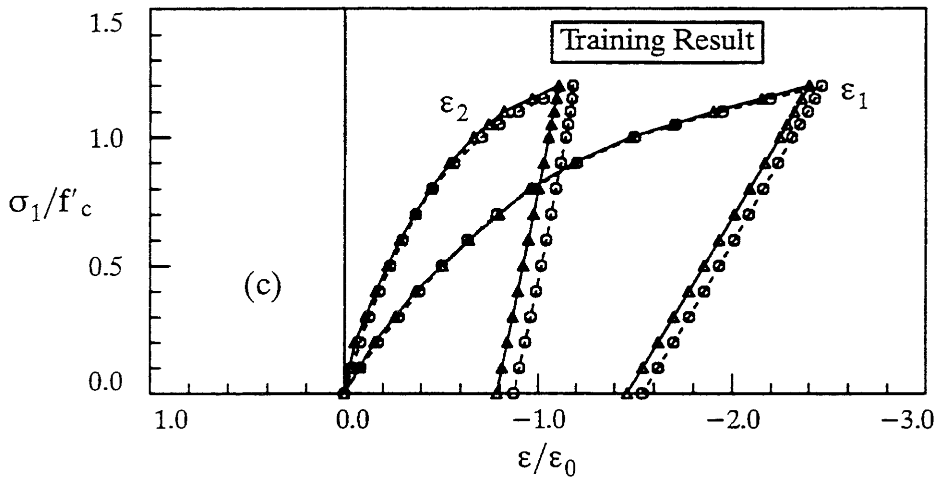


Fig. 4.33 – The Stress-Strain Relation Predicted by the Neural Network on Stress Path cc6 (with Elastic Unloading)

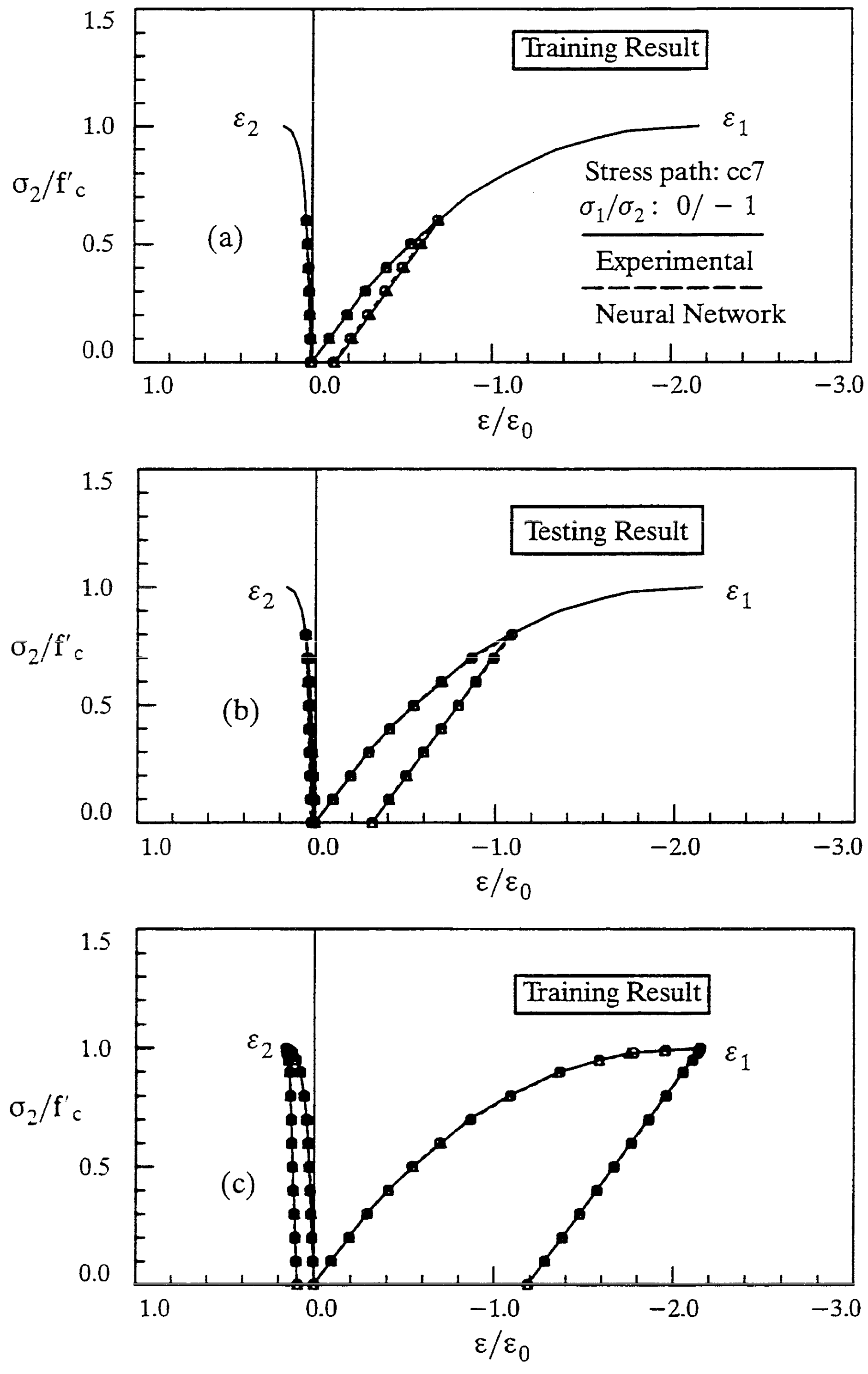


Fig. 4.34 – The Stress-Strain Relation Predicted by the Neural Network on Stress Path cc7 (with Elastic Unloading)

4.4 Neural Network-Based Material Model of Concrete in Uniaxial Cyclic Compression

4.4.1 Introduction

The constitutive models of concrete in biaxial stress states constructed so far all simulate the behavior of concrete under monotonic proportional loadings. As most loadings to which reinforced concrete structures may be subjected during their service life are of dynamic nature, it is important to model the behavior of concrete under cyclic or repeated loading. Since 1960's, many experimental tests have been conducted to study this aspect of the behavior of plain concrete, and most of those tests were performed on concrete under cyclic compressive loading (Sinha, et al., 1964; Karsan and Jirsa, 1969). The salient feature observed is the degradation in both stiffness and strength of concrete with increasing number of cycles at a higher enough stress level. That is, loading history plays a more decisive role in determining the behavior of concrete under cyclic loading. With the difficulty involved in the identification and determination of material parameters, though some empirical models have been proposed (Darwin and Pecknold, 1977; Fafitis and Shah, 1986; Fardis, et al., 1981; Yankelevsky and Reinhardt, 1987), the analytical constitutive modeling of concrete under cyclic loading is still very difficult (ASCE, 1982).

In the following paragraphs, the neural network-based material modeling methodology is applied to model the behavior of concrete in uniaxial compressive cyclic loading.

4.4.2 Problem Representation and Architecture Determination

To model the stress-strain behavior of plain concrete under cyclic loading, not only does it need to model the unloading and reloading curves, but it also ought to define the uniqueness of stress-strain relation by distinguishing different loading histories. Obviously, a one-point scheme is inadequate to handle the uniqueness of stress-strain relation associated with reloading and unloading. Therefore, a three-point scheme with some information on the previous loading history is designed to represent the cyclic behavior of concrete under uniaxial compression, in which the path dependency can be adequately captured by inputting the current point and two previous points on the stress-strain curve. Consequently, in a stress-controlled model, the seven processing units in the input layer represent the stresses and strains of three points on the curve, plus a stress increment. The corresponding strain increment is represented by the only processing unit in the output layer. The network architecture and the representation scheme are shown in Fig. 4.35. As usual, two hidden layers are used in the network.

In this problem, the number of hidden nodes in the two hidden layers is deter-

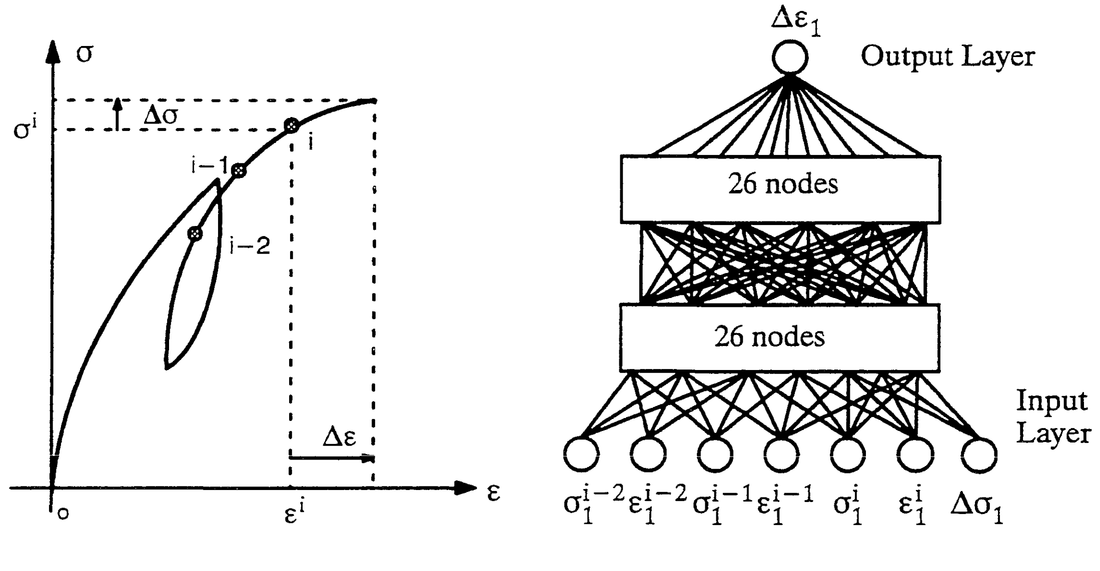


Figure 4.35 – Architecture of the Neural Network for Stress-Controlled Material Model Under Uniaxial Compressive Cyclic Loading

mined by the size of the training data or the number of reloading-unloading cycles included in the training data. For training 4 cycles of stress-strain relation from Karsan and Jirsa's experiment (1969), with the maximum learning error reduced to below 0.10, totally 26 hidden nodes are determined by the dynamic nodes creation scheme, as shown in Fig. 4.35. However, it needs 32 hidden units in each hidden layer to successfully train 6 cycles of stress-strain data.

4.4.3 Training and Testing of the Neural Network-Based Concrete Model

The experimental data used in training the neural network are the normalized stress-strain curves in series AC2−9, reported in the reference (Karsan and Jirsa, 1969). Those data are then re-scaled to stress-strain values in the range of (−1.0, 1.0) to generate the training data set. The results of training the neural network on the first four cycles are shown in Fig. 4.36, in which the neural network prediction is plotted as a

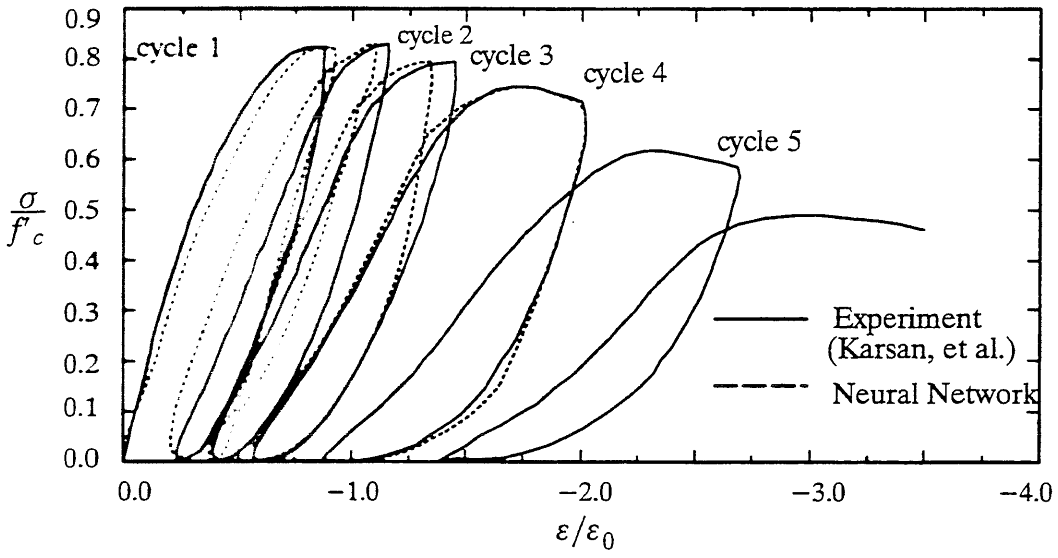


Fig. 4.36 − The Stress-strain Relation Predicted by the Neural Network − Training Results (after Training the First 4 Cycles)

dotted line and the experimental data as a solid line. The neural network seems to be capable of learning the presented data reasonably well. Next, we present the results of some tests to explore the generalization capability of the trained network.

First, the trained network was tested with the first four cycles of a completely different cyclic test, reported in the reference (Sinha, et al., 1964). The neural network results are compared with the experiments in Figure 4.37, and appear reasonable. Secondly, a number of experiments were performed to test how well a trained network applies the knowledge learned about the material behavior in few cycles to predict additional cycles. The neural network was first trained on the first stress-strain cycle and tested on the second cycle, the results of which are shown in Figure 4.38. Next, the neural network was trained on the first three cycles and tested on the fourth cycle, the results of which are shown in Figure 4.39. Finally, the network was trained on the first five cycles of stress-strain relation and tested on the sixth cycle, and the results are

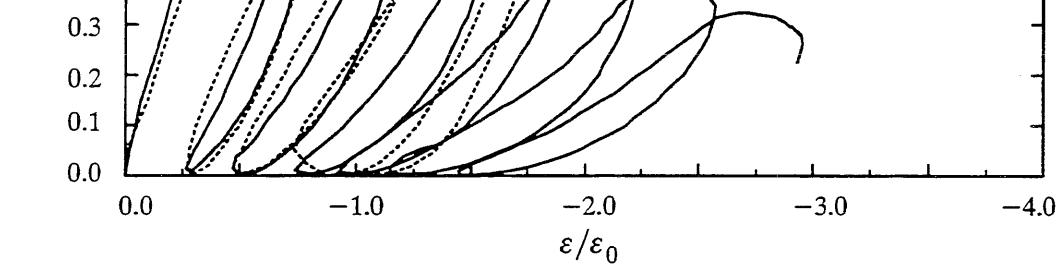


Fig. 4.37 – The Stress-strain Relation Predicted by the Neural Network – Testing Results on Different Data (Sinha, et al., 1969)

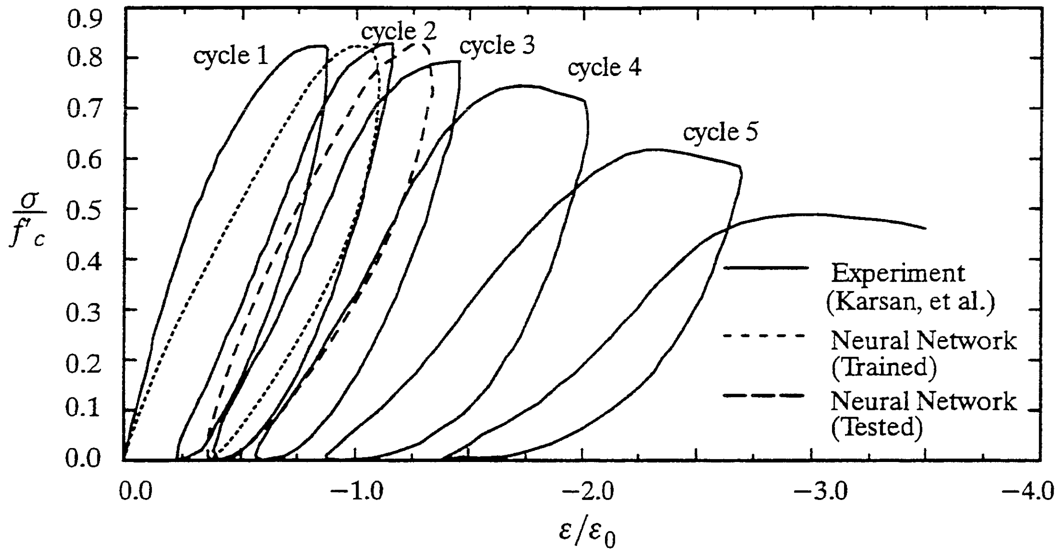


Fig. 4.38 – The Stress-strain Relation Predicted by the Neural Network on the Second Cycle after Training the First Cycle

shown in Fig. 4.40. The performance of the neural network on the testing cycle appears to be incrementally improving as more cycles of stress-strain data are added to the training data set. This is interesting as it indicates that it needs to train more than four cycles of stress-strain data for the network to learn the degradation in stiffness and strength. Though these testing results appear to be satisfactory and much better than initial expectations, there is room for improvement.

The neural network originally trained on the first four cycles (Fig. 4.36) cannot be reasonably expected to give rational results when subjected to smaller stress cycles inside the Smith and Young (1955) envelope, since it was not provided with any information on these cycles. The results of such tests are shown in Figs. 4.41a – 4.41.b. The fact that these results look somewhat reasonable attests to the generalization capabilities of these neural networks.

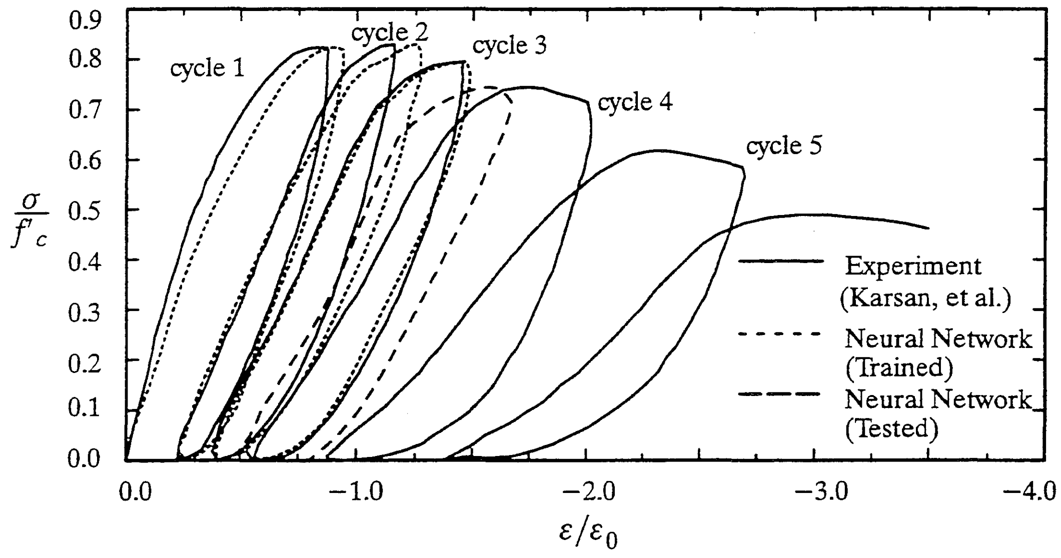


Fig. 4.39 – The Stress-strain Relation Predicted by the Neural Network on the Fourth Cycle after Training the First Three Cycles

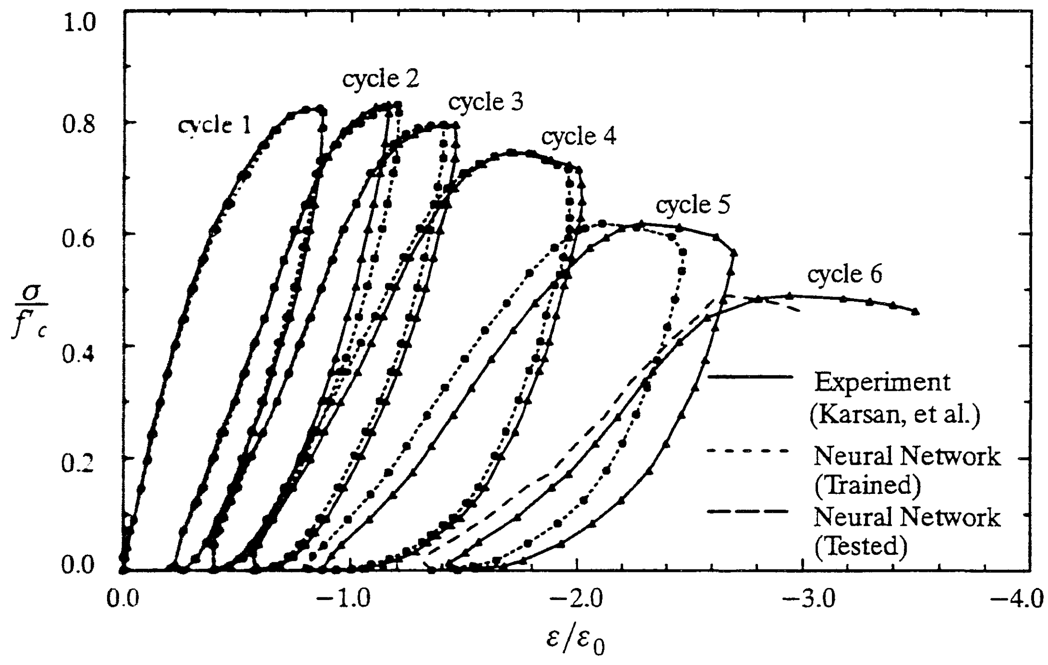


Fig. 4.40 – The Stress-strain Relation Predicted by the Neural Network on the Sixth Cycle after Training the First Five Cycles

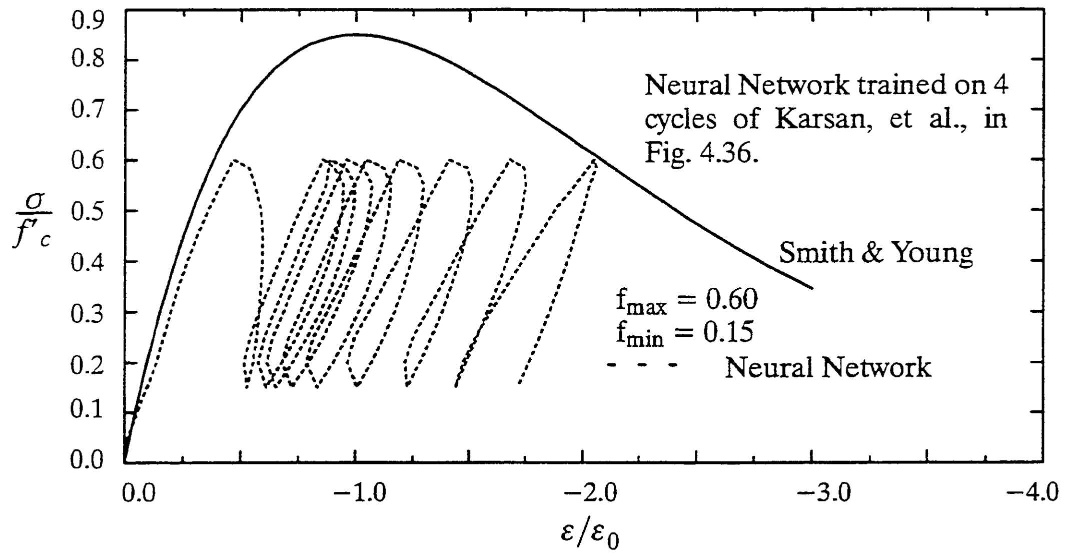


Fig. 4.41.a – The The Stress-strain Relation Predicted by the Neural Network on Low Stress Cycles

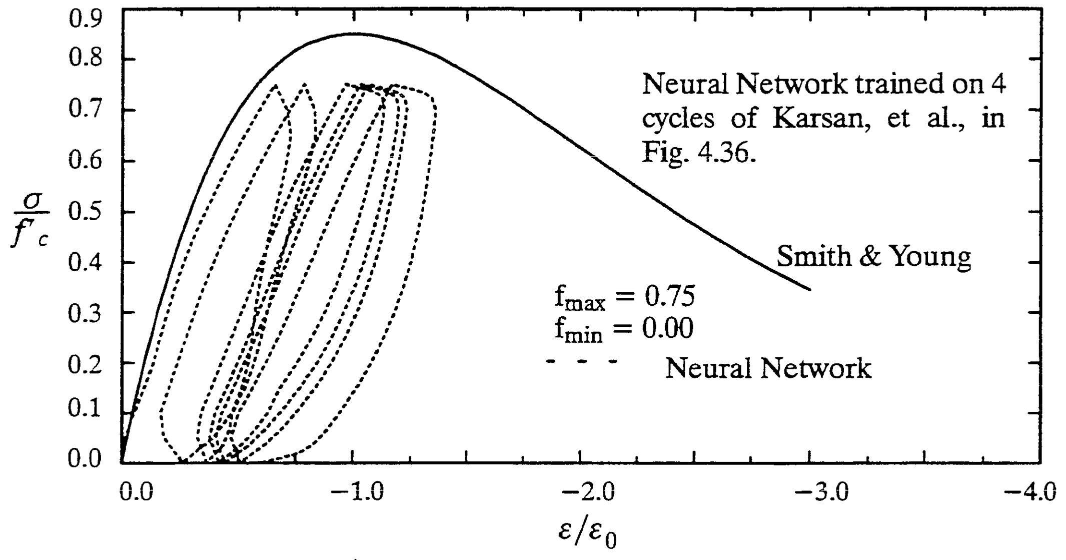


Fig. 4.41.b – The Stress-strain Relation Predicted by the Neural Network on Low Stress Cycles

CHAPTER 5

NEURAL NETWORK MATERIAL MODELS OF REINFORCED CONCRETE

5.1 Introduction

The neural network-based material modeling methodology developed in this study is particularly applicable to composite materials, such as reinforced concrete and modern fiber-epoxy composites. The current analytical approach to the constitutive modeling of these materials simulates the behavior of constituent materials and then separately accounts for the interaction between them using numerical procedures. The main difficulty associated with conventional approaches to modeling of composites may have resulted from the complexity in material parameter identification and the difficulty of including non-mechanics parameters in the analytical constitutive model.

With neural networks, it is possible to treat the whole composite as a material because, as Rumelhart stated, these networks are ideally suited to treat a multitude of different kinds of constraints simultaneously (Rumelhart, 1986). The basic approach would be to include the essential features of the composite as input material parameters in training a neural network. For example, for reinforced concrete (RC), part of the input to the neural network would be information about the concrete and rebars, such as the reinforcement ratio, placement or direction of rebars, etc., in addition to information on stresses and strains. In fiber-epoxy composites, the input to the neural network would include the number of layers and their thickness, type and directions of the fibers in various layers, and any other information needed to characterize the composite. With the use of neural networks, the material parameters can be of various types, each having different characteristics.

In general, the training of such a neural network would require the results of a comprehensive set of experiments. These experiments should not only measure the response of the material to various stress paths, but also test the material for all the possible data ranges of the input parameters characterizing the material. Though such experimental results are not readily available at the moment, this research explores the neural network approach for simulating the behavior of a composite material based on some fairly well conducted experiments.

In this chapter, neural network material models for reinforced concrete in biaxial states of stress are constructed using the experimental results reported by Vecchio and Collins (1982) on the behavior of reinforced concrete panels subjected to in-plane shear and combination of in-plane shear with normal stresses.

5.2 Review of Vecchio and Collins' Experiments

The response of reinforced concrete panels subjected to in-plane shear and normal stresses under various stress paths has been extensively investigated by Vecchio and Collins (1982) through a series of tests on 30 specimens. These panels were loaded by forces applied to the "shear keys", which were anchored into the perimeter edges of the specimens as shown in Fig. 7.1. Each shear key was attached by two links oriented at 45 degrees to the side of the specimen. The links were connected to a series of hydraulic jacks. Three of the links were rigid so as to stabilize the panel within the test rig. Different combinations of shear force and normal tension and compression were generated by changing the magnitude and directions of forces in the links. The details involved in the design and construction of the test rig are described in the report by Vecchio and Collins (1982).

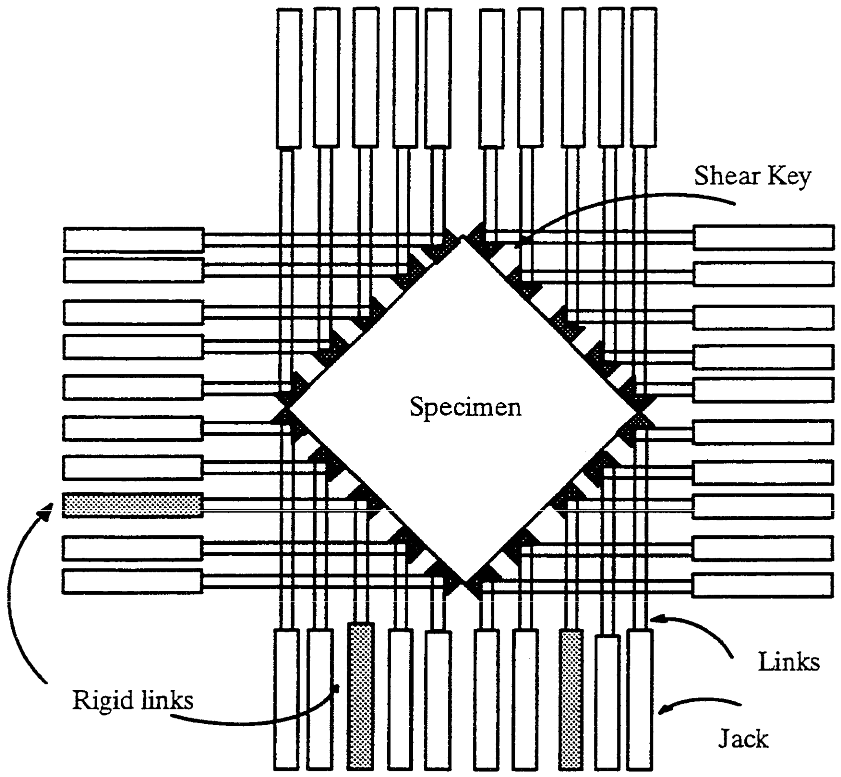


Fig. 7.1 – Test Set-Up for Vecchio-Collins Panels

The testing specimens were constructed as concrete panels with a size of 890 mm square by 70 mm thick, reinforced with two layers of welded wire mesh. The wires of the mesh were placed parallel with the sides of the panel, with the two directions of the wires being identified as “longitudinal” and “transverse.” The reinforcing mesh was constructed of smooth wires welded into an orthogonal grid, typically at 50 mm centers. The percentages of transverse and longitudinal reinforcements are varied, covering both isotropic and anisotropic reinforcements in both directions. The plan view of the reinforcement and loading directions are shown in Fig. 7.2.

For each panel, the material properties such as the concrete cylinder strength, f'_c, the concrete cylinder strain at peak compressive stress, ε_0, the cracking strength of

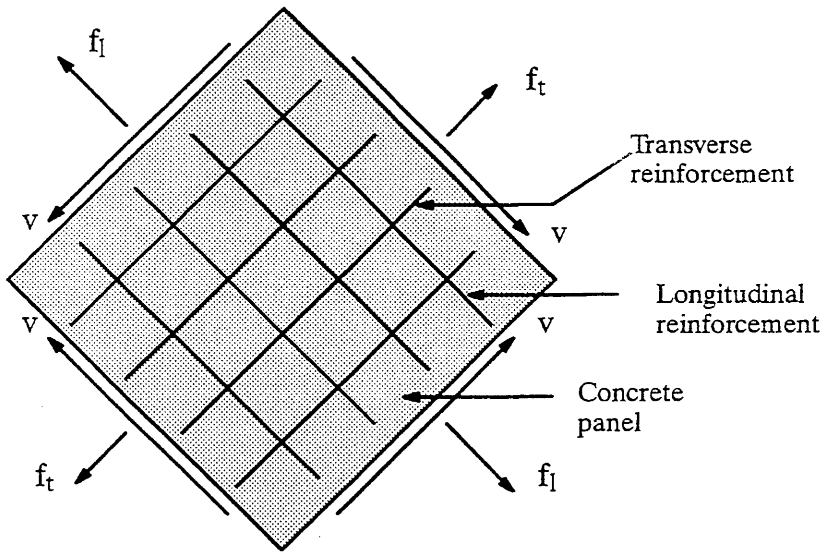


Fig. 7.2 – Plan View of Reinforcement and Loading Directions for Vecchio-Collins Specimens

concrete, f'_{cr}, and the yielding stress in the longitudinal steel, f_{yl}, and in the transverse steel, f_{yt}, were determined at the time when the panel was tested. During the test, load was monotonically increased from zero to a level until failure occurred. A test typically consisted of 7 to 12 load stages until the specimen reached its failure state. Totally six loading patterns were employed for the testing of the 30 panels, in which 22 panels were tested in pure shear, 3 panels in combined shear with compressive normal stresses, 1 panel in combined shear with normal tension stresses, 1 panel in pure shear with changing loading condition, 2 panels in uniaxial compression, and 1 panel in reversed cyclic shear. The material properties, reinforcements, and loading condition of each panel are summarized in Table 5.1.

For the 22 panels tested in pure shear, 10 panels (PV2–PV9, PV14 and PV16) were reinforced isotropically, but the percentage of reinforcement was varied from 0.183% to 2.616%; another 12 panels (PV1, PV10–PV13, PV18–PV22, PV26, and

Table 5.1 – Material Properties and Loading Conditions for Vecchio-Collins Specimen

Speci-men	Load-ing	Longitudinal Steel		Transverse Steel		Concrete		
		ϱ_l (%)	f_{yl} (MPa)	ϱ_l (%)	f_{yl} (MPa)	ε_0 (10^{-3})	f'_c (MPa)	f'_{cr} (MPa)
PV 1	PS	1.785	483	1.680	483	2.00	34.5	2.21
PV 2	PS	0.183	428	0.183	428	2.25	23.5	1.10
PV 3	PS	0.483	662	0.483	662	2.30	26.6	1.66
PV 4	PS	1.056	242	1.056	242	2.50	26.6	1.79
PV 5	PS	0.742	621	0.742	621	2.50	28.3	1.73
PV 6	PS	1.785	266	1.785	266	2.50	29.8	2.00
PV 7	PS	1.785	453	1.785	453	2.50	31.0	1.93
PV 8	PS	2.616	462	2.616	462	2.50	29.8	1.73
PV 9	PS	1.785	455	1.785	455	2.80	11.6	1.38
PV 10	PS	1.785	276	0.999	276	2.70	14.5	1.86
PV 11	PS	1.785	235	1.306	235	2.60	15.6	1.66
PV 12	PS	1.785	469	0.446	269	2.50	16.0	1.73
PV 13	PS	1.785	248	–	–	2.70	18.2	1.73
PV 14	PS	1.785	455	1.785	455	2.23	20.4	1.93
PV 15	UC	0.740	255	0.740	255	2.00	21.7	–
PV 16	PS	0.740	255	0.740	255	2.00	21.7	2.07
PV 17	UC	0.740	255	0.740	255	2.00	18.6	–
PV 18	PS	1.785	431	0.315	412	2.20	19.5	2.00
PV 19	PS	1.785	458	0.713	299	2.15	19.0	2.07
PV 20	PS	1.785	460	0.885	297	1.80	19.6	2.21
PV 21	PS	1.785	458	1.296	302	1.80	19.5	2.35
PV 22	PS	1.785	458	1.524	420	2.00	19.6	2.42
PV 23	SBC	1.785	518	1.785	518	2.00	20.5	2.28
PV 24	SBC	1.785	492	1.785	492	1.90	23.8	0.83
PV 25	SBC	1.785	466	1.785	466	1.80	19.2	1.31
PV 26	PS	1.785	456	1.009	463	1.85	21.3	2.00
PV 27	PS	1.785	442	1.785	442	1.90	20.5	2.04
PV 28	SBT	1.785	483	1.785	483	1.85	19.0	2.21
PV 29	CLR	1.785	441	0.885	324	1.80	21.7	2.21
PV 30	RCS	1.785	437	1.009	472	1.90	19.1	1.55

PS: Pure Shear; UC: Uniaxial Compression; SBT: Combined Shear and Biaxial Tension; RCS: Reversed Cyclic Shear; CLR: Changing Load Ratio; SBC: Combined Shear and Biaxial Compression.

Table 5.2 – Loading Ratio of Biaxial Stress (f_n) to Shear Stress (v)

Specimen	PV23	PV24	PV25	PV28
f_n / v	−0.39	−0.83	−0.69	0.32

PV27) were reinforced anisotropically with the amount of longitudinal reinforcement being held constant while the transverse reinforcement varied. A wide range of strength values of concrete and the yield stresses of reinforcement in both directions were obtained during the course of testing.

Four panels (PV23–PV25, PV28) were tested under combined in-plane shear and biaxial normal stresses. Specimens PV23–PV25 were subjected to varying degree of biaxial compression and shear, while specimen PV28 in biaxial tension and shear. The ratio of biaxial normal stress (f_n) to shear stress (v) as given in the report is shown in Table 5.2, in which tensile stresses are shown as positive and compressive stresses as negative. All the four panels were isotropically reinforced, with a constant percentage of reinforcement of 1.785%.

Specimen PV29, which was anisotropically reinforced, was tested under changing loading conditions. It was loaded in pure shear up to 80% of the predicted ultimate shear strength; and then, the loading was changed to combined shear and biaxial compression. In the remaining three panels, PV30 was tested in reversed cyclic shear; PV15 and PV17 were tested in uniaxial compression.

The results of these tests on the stress-strain behavior, specifically, relationships between the shear stress and shear strain, the normalized principal stresses and corresponding principal strains, of the concrete of each panel were computed and described in the report (Vecchio and Collins, 1982). Those stress–strain data along with information on the composition of each specimen and properties of constituents materials are to be used as material parameters in the construction of a neural network-based material model of reinforced concrete.

5.3 Neural Network-Based Modeling Techniques for Reinforced Concrete

The concepts and procedures involved in the neural network-based material modeling of reinforced concrete are similar to those with the modeling of plain concrete. That is, a neural network-based material model of reinforced concrete material can be constructed through training a multilayer feedforward neural network on a comprehensive set of experimental results characterizing the behavior of the material. Though the experimental data from Vecchio and Collins' tests may not be qualified as "comprehensive", it is of importance to verify the proposed approach for simulating the nonlinear stress–strain behavior of reinforced concrete panels under diverse loading conditions.

After studying the experimental results reported by Vecchio and Collins (1982) on 30 reinforced concrete panels under different loading combinations, it was determined that the stress-strain relations corresponding to part of the 22 panels tested under pure shear would be used as training data, and the trained network would then be tested for its generalization on remaining panels loaded with pure shear or with combined shear and normal stresses.

Before the representation scheme is determined, the material parameters or variables that characterize the behavior of these panels need to be identified. Obviously, there are three kinds of material variables for the neural network model: 1) the stress–strain variables including shear stress and shear strain, the compressive and tensile principal stresses and the corresponding principal strains; 2) material properties of the constituent materials including yield stresses of reinforcement in both longitudinal and transverse directions, the cylinder strength of concrete and the corresponding strain, and the cracking strength of concrete; and 3) the composition of the specimen: the reinforcement ratios and directions of reinforcement. The material properties of the concrete in each specimen are implicitly incorporated in the stress–strain material variables through normalization on the principal compressive and tensile stresses, and principal compressive strains, when preparing the training data. Specifically, the principal compressive and tensile stresses are normalized with respect to the cylinder strength (f'_c) and cracking strength of concrete (f'_{cr}), respectively; whereas the principal compressive strains are normalized with respect to the concrete cylinder strain at peak compressive stress, ε_0.

To characterize the strain softening behavior of concrete in tension, it is important to include a portion of the loading history in the representation scheme. Therefore, a three-point representation scheme is devised for both stress-controlled and strain-controlled schemes. Before training the network with stress-strain data prepared according to the representation scheme, the experimental results including all the material variables are linearly transformed into the range of $(-1.0, 1.0)$. The schemes derived for the transformation of stresses, strains, and their increments were determined in the same way as that with plain concrete in biaxial stress states described in Chapter 4.

5.4 Representation Schemes and Architecture Determination

For a stress-controlled model, the network will need 25 input nodes representing information on the two previous stress/strain states and that of the current state

($\sigma_1(i-2)$, $\sigma_2(i-2)$, $v(i-2)$, $\varepsilon_1(i-2)$, $\varepsilon_2(i-2)$, $\gamma(i-2)$, $\sigma_1(i-1)$, $\sigma_2(i-1)$, $v(i-1)$, $\varepsilon_1(i-1)$, $\varepsilon_2(i-1)$, $\gamma(i-1)$, $\sigma_1(i)$, $\sigma_2(i)$, $v(i)$, $\varepsilon_1(i)$, $\varepsilon_2(i)$, $\gamma(i)$), plus the reinforcement ratios (ϱ_l, ϱ_t) and yield stresses of reinforcement in both longituainal and transverse directions (f_{yl}, f_{yt}), as well as the stress increments ($\Delta\sigma_1$, $\Delta\sigma_2$, Δv). The three output units represent the strain increments ($\Delta\varepsilon_1$, $\Delta\varepsilon_2$, $\Delta\gamma$) corresponding to the current stress increments. For a strain-controlled model, only the positions of stress components are switched with those of strain components in a stress-controlled model. As usual, two hidden layers are used in the network, and full connections between layers are enforced.

Before the stress-controlled neural network model was trained with any stress–strain data, it was decided that an optimal training data set would be determined in this investigation. The approach still follows the scheme of incremental training with tests for generalization evaluation. It starts with an initial training set consisting of stress–strain data of 10 panels tested in pure shear, which include panels PV1, PV4, PV5, PV6, PV9, PV10, PV11, PV13, PV18, and PV19. After reducing the training error to below 0.06, the network was tested with stress–strain data of the untrained panels. At that stage, each hidden layer registered 20 hidden units, which is shown in Fig. 7.3. By analyzing the training and testing results at that stage, it was observed that the network predictions on some testing panels had substantial discrepancy with the experiment results. Then, the stress–strain data corresponding to 4 more panels (PV22, PV14, PV16, and PV2), on which the largest discrepancy occurred, were added to the original training set, making the size of the training data to 14 panels. After reducing the training error on this augmented training set to below 0.060, it was discovered that further training was not needed even with some new panels added to the training set. That is, the presentation of a new training set does not increase the training error at all. This latter discovery clearly indicates that the network can essentially characterize the stress – strain behavior of these reinforced panels under pure shear after being trained on the behavior of 14 panels. The final architecture consists of 22 hidden nodes in each hidden layer. To fully illustrate this approach, the training and testing results of the stress-controlled model at the intermediate stage after training experimental results of 10 panels and at the final stage when 14 panels were trained, are described in the following paragraphs.

5.5 Training and Testing of Neural Network-Based RC Models

5.5.1 A Stress-controlled Model

With incremental training on the stress–controlled model, the initial network

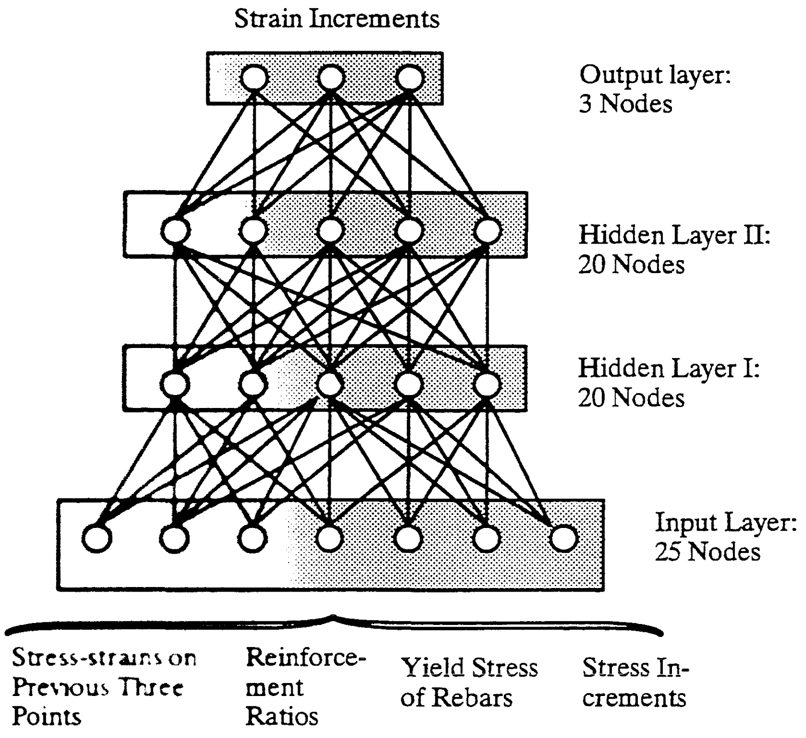


Fig. 7.3 – The Architecture of the Stress-controlled Neural Network Material Model of Reinforced Concrete in Biaxial Stress State

starts with 10 hidden nodes in each hidden layer. After the stress–strain data of 10 panels were successfully trained, the hidden nodes had been increased to 20, as shown in Fig. 7.3. The training results on the prediction of stress–strain relations were shown in Figs. 7.4 – 7.13, in which the network predictions shown as thick dotted lines were plotted against the experimental values represented as solid lines. In each figure, the top graph represents the normalized principal compressive stress-strain relation, the middle one the normalized principal tensile stress–strain relation, and the bottom one the shear stress–strain relation. From analyzing those figures, it is obvious that the neural network has generally learned the behavior of reinforced concrete panels in biaxial stresses with reasonable accuracy. Testing was also performed incrementally. The testing results on untrained panels including those tested under pure shear, under combined shear and normal stresses, and under pure shear with changing loading conditions, were shown in Fig. 7.14 – 7.30. It was observed that the majority of the testing results were reasonable, even on those panels tested in combined shear and normal stresses, but the degree of accuracy varied with each testing panel and there was room for improvement. For those panels that were tested in pure shear, it appeared that the network predictions on 4 panels (PV2, PV14, PV16, and PV22) displayed the largest discrepancy in their stress-strain relations. To incorporate material behavior exhibited by these 4 panels into the network (which so far was not adequately learned), the stress-strain relation corresponding to the 4 panels were added to the training data set. Subsequently, further training was performed on the network with this augmented training data set until the same convergence criterion was satisfied. As has been stated before, those 14 sets of data constitute a quasi–optimal training set for this problem.

The training and testing results after training stress–strain data from 14 panels, represented as thin dotted lines, are plotted against those after training data from 10 panels, and also shown in Figs. 7.4 – 7.30. The improvement on the prediction of stress–strain relations with this additional training can be easily seen in all the testing results shown in Figs. 7.14 – 7.30. However, both networks have shown essentially the same performance on all the training cases because the learning convergence criteria were specified as the same. From those figures, one can easily observe that not only the the latest testing results on the panels tested in pure shear are reasonable, but the network predictions on other loading cases are equally well. The remarkably reasonable predictions on all the panels tested under combined shear and normal stresses, and even the panel tested under pure shear with changing loading conditions clearly demonstrate the robustness of a neural network-based material model with novel but similar cases in the same problem domain, after the network has been appropriately trained on a quasi–optimal set of data.

After analyzing the training and testing results of the neural network, it is clear that the learning was not performed uniformly on the three stress–strain relations of each panel. In most of the testing cases, the network predictions match the experimental results better with the principal compressive stress–strain and the shear stress–strain than with the principal tensile stress–strain relation. This learning behavior may be caused by the apparently more noises in the experimental data of the principal tensile stress–strain relation. On the other hand, the highly nonlinear strain softening behavior with the principal tensile stress-strain relation may have introduced extra difficulty in learning.

To determine whether different training approaches may have some effects on the generalization performance of the trained neural network, the initial training data set (10 panels) as a whole was also trained by a network using the architecture predetermined from the incremental training. After the network was successfully trained with the same convergence criteria, the training and testing results from the latest model were compared with those from incremental training. As expected, the performance of both models was essentially the same. However, the incremental training took less time to train the whole data set.

One interesting observation made during the initial preparation of testing data for the material model was that the original experimental data were rather noisy and the stress increments were mostly irregular. Initially, it was anticipated that the learning would be more difficult than with smooth data. However, on the contrary, the actual learning was far more easier than with training the plain concrete model described in the previous chapter, and the model was also remarkably robust which was manifested by the reasonable predictions on those testing panels subjected to combined shear and normal stresses. This learning behavior indicates that a neural network model can be readily built with raw or preliminary processed data because of the statistical learning and fuzzy information processing capability of the neural network.

The construction of a strain-controlled neural network model and a comparison of its predictions on training and testing results with the stress-controlled model are presented in the next section.

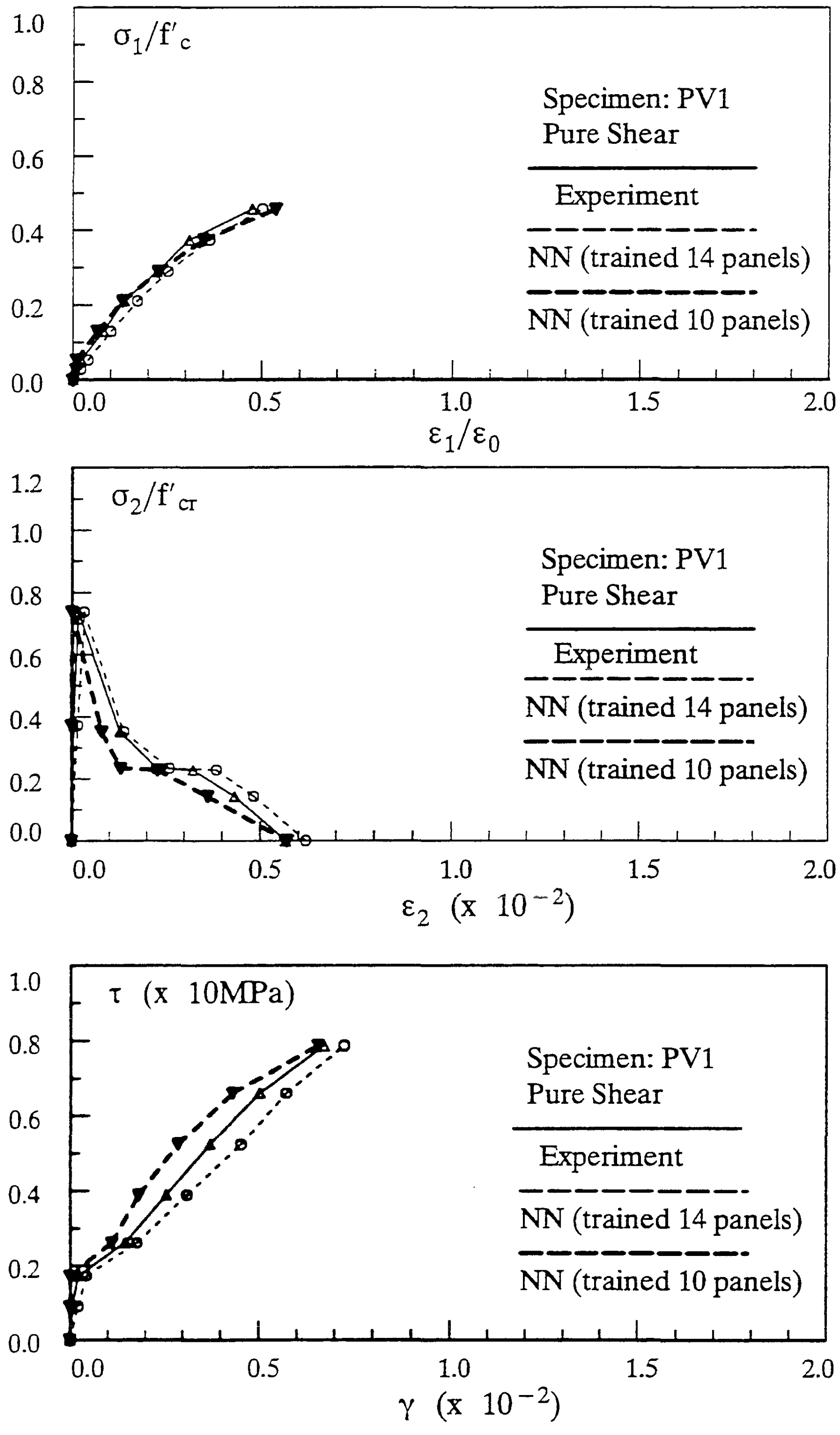
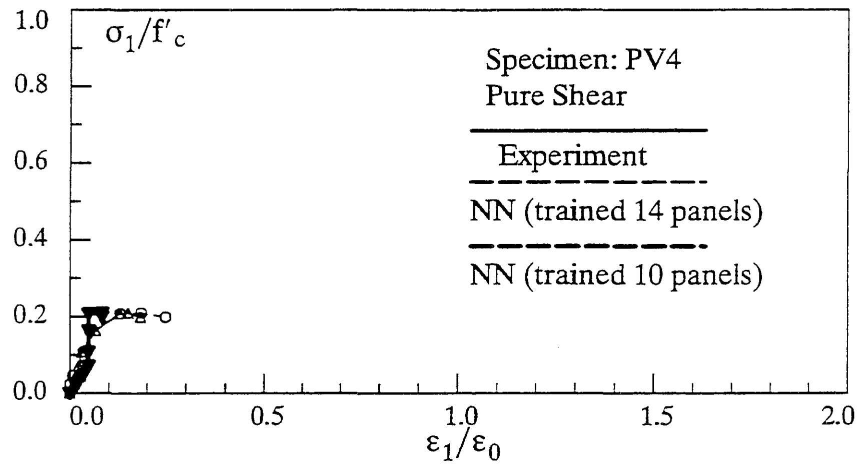
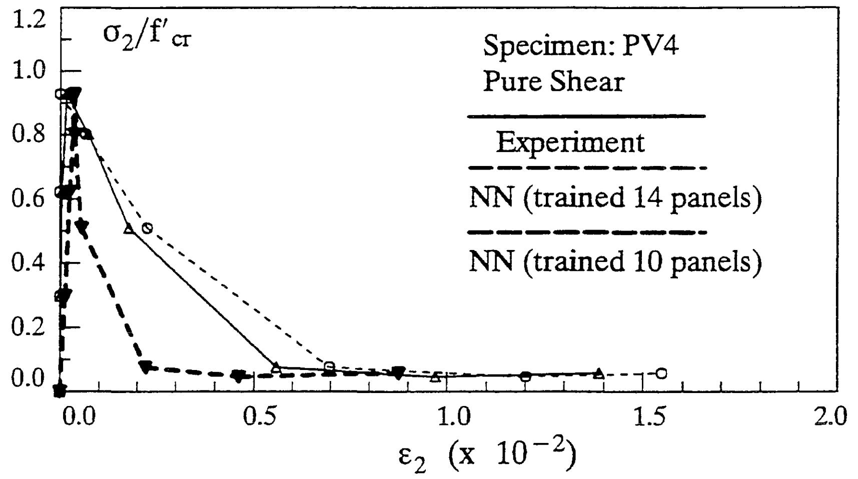


Fig. 7.4 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models





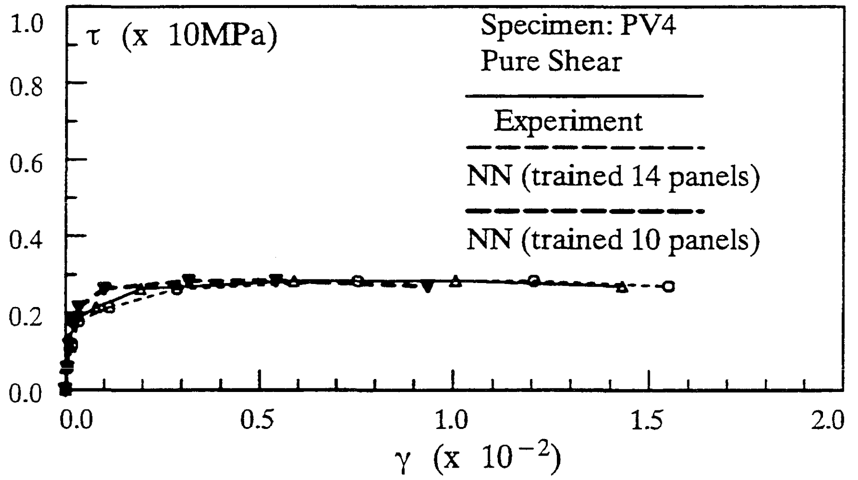
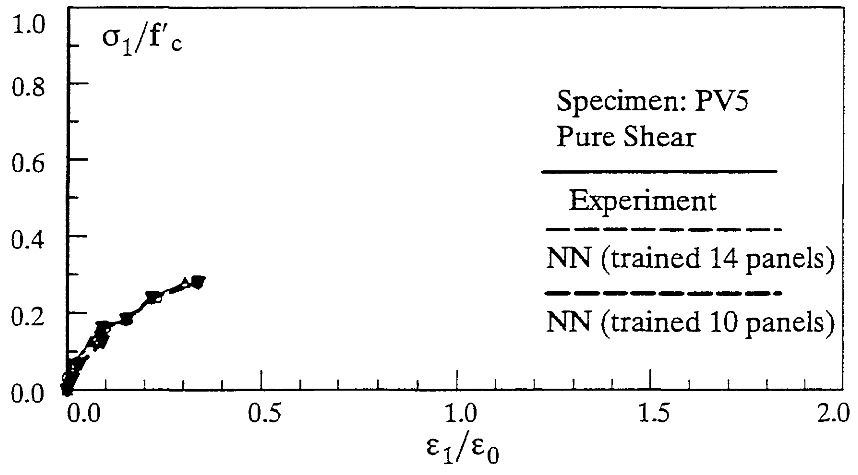
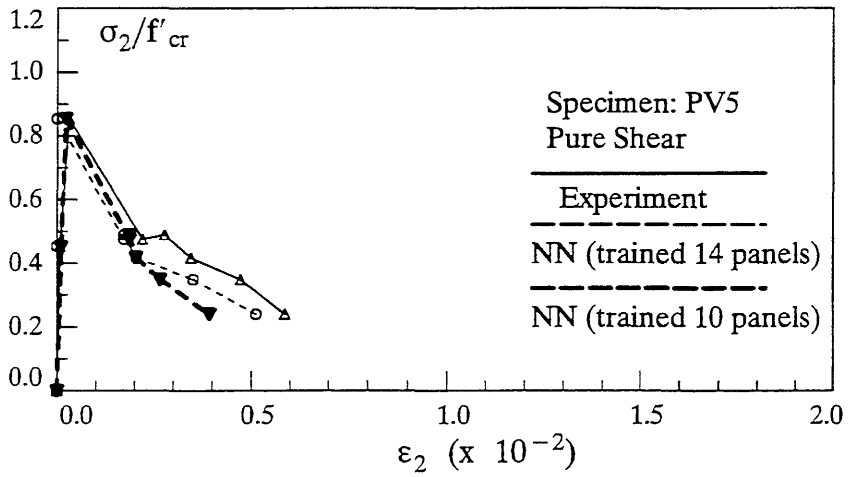


Fig. 7.5 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models





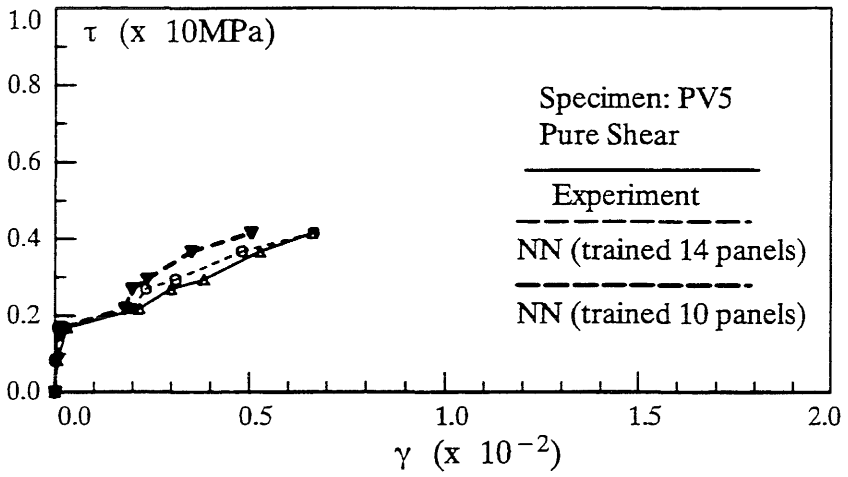


Fig. 7.6 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models

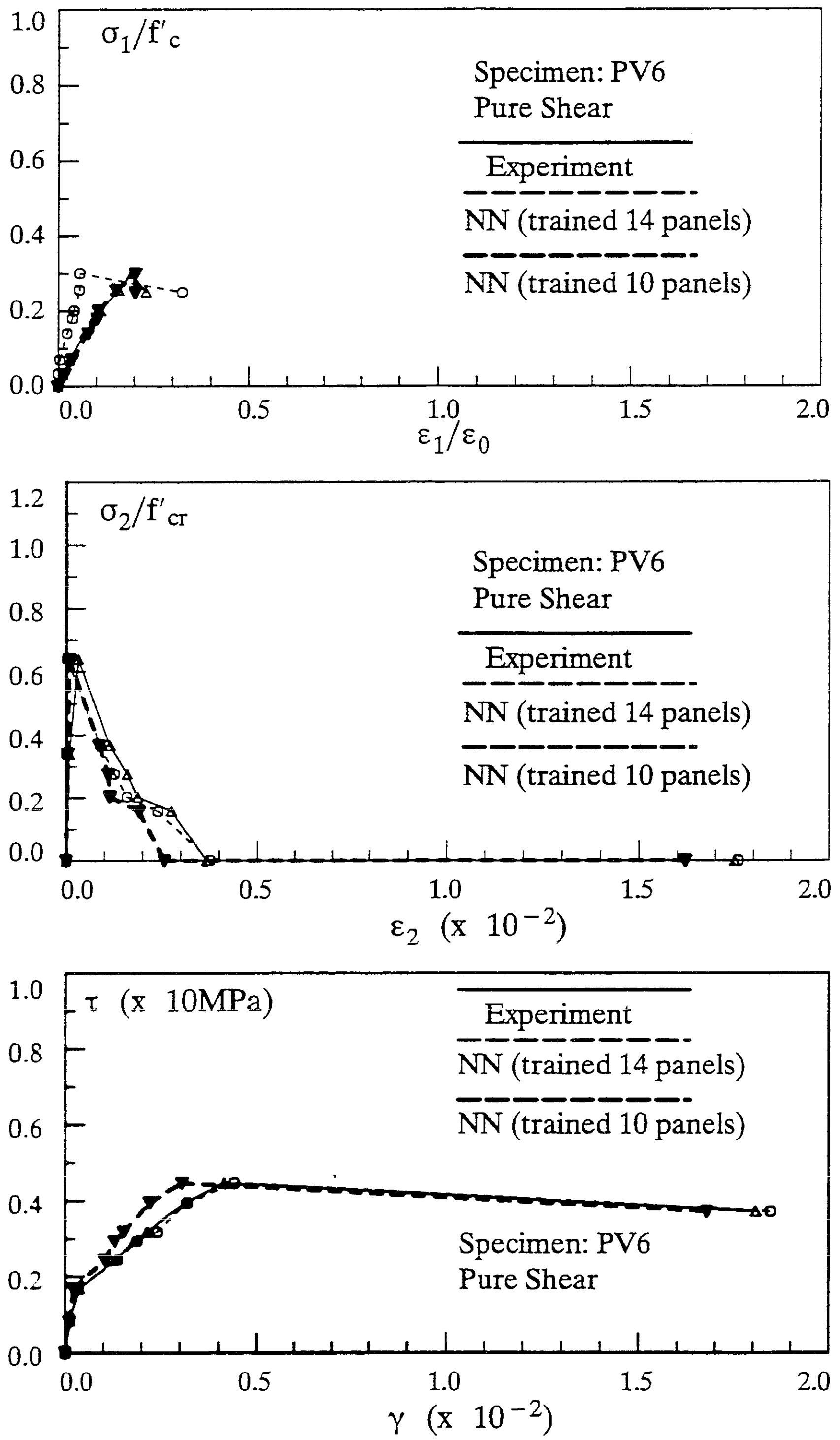
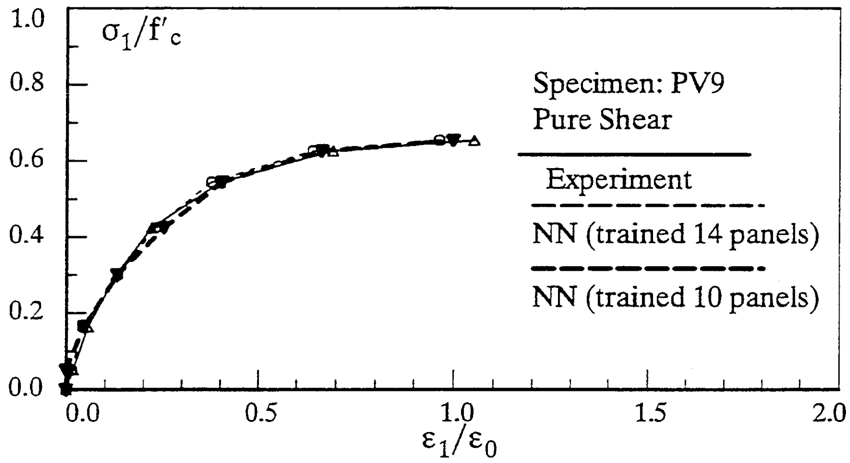
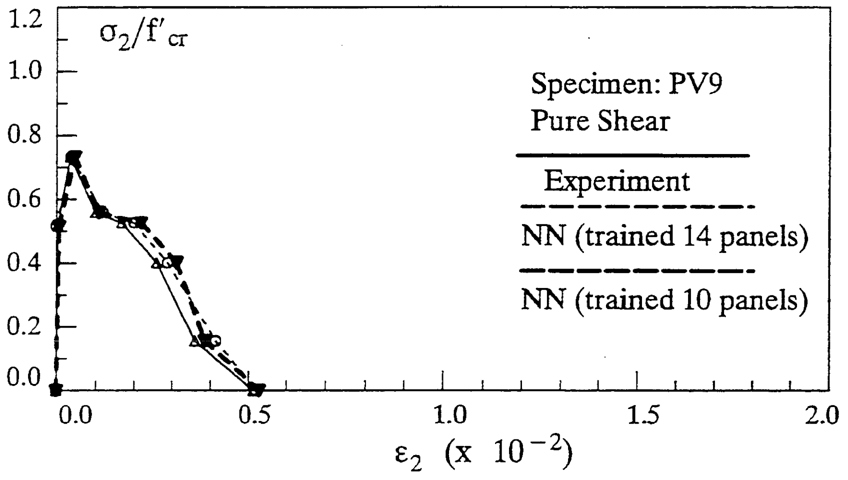


Fig. 7.7 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models





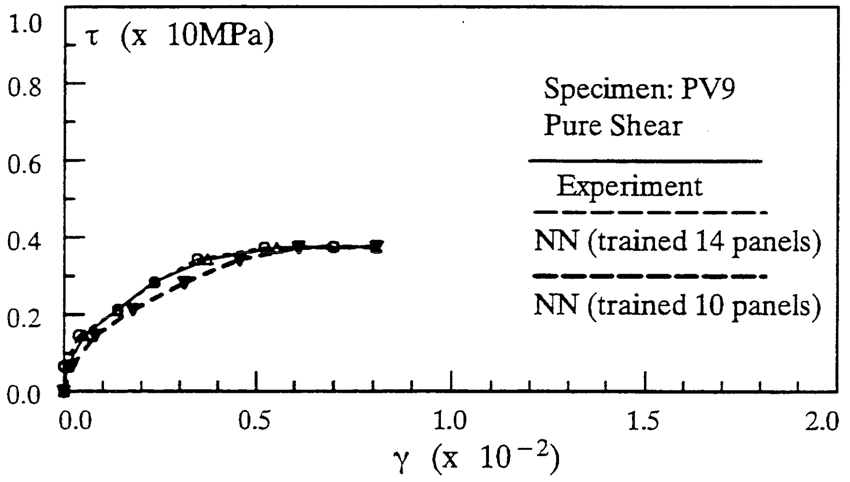
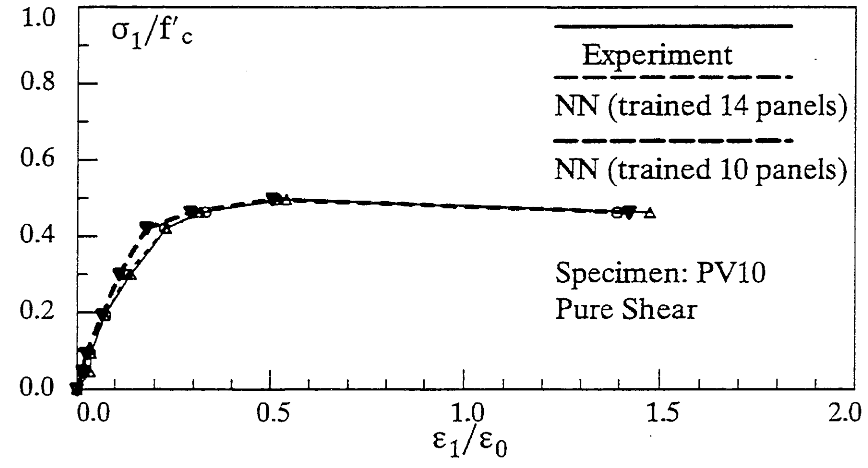
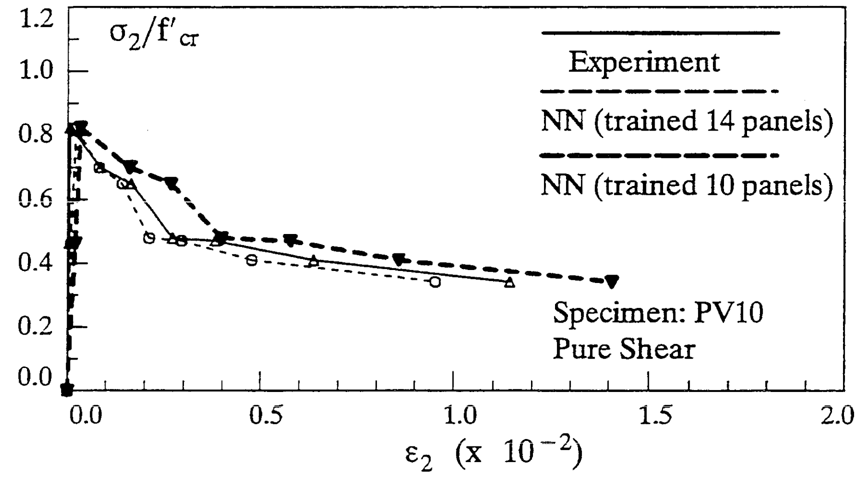


Fig. 7.8 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models





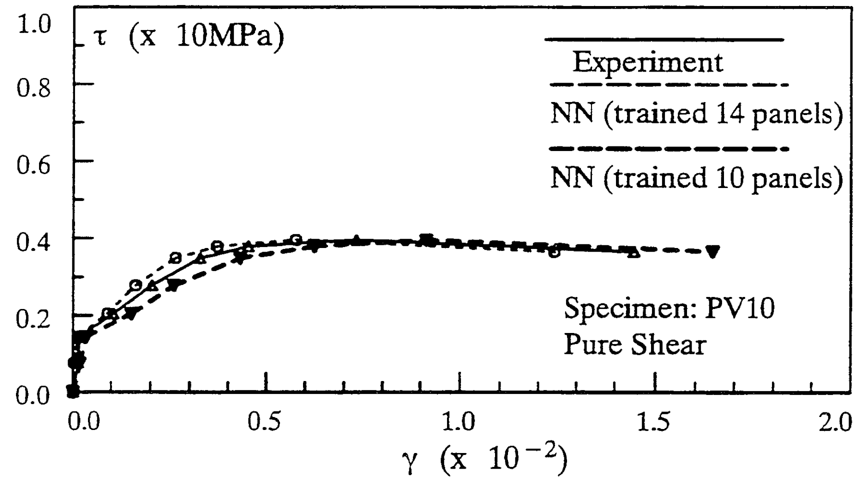
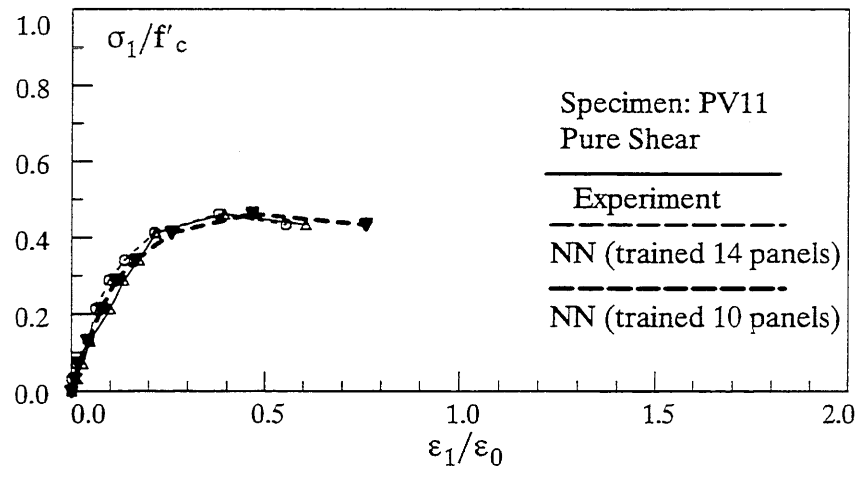
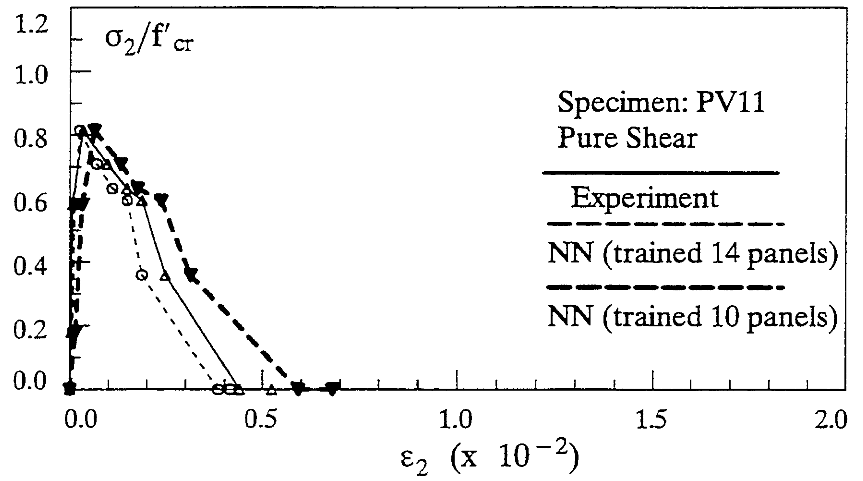


Fig. 7.9 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models





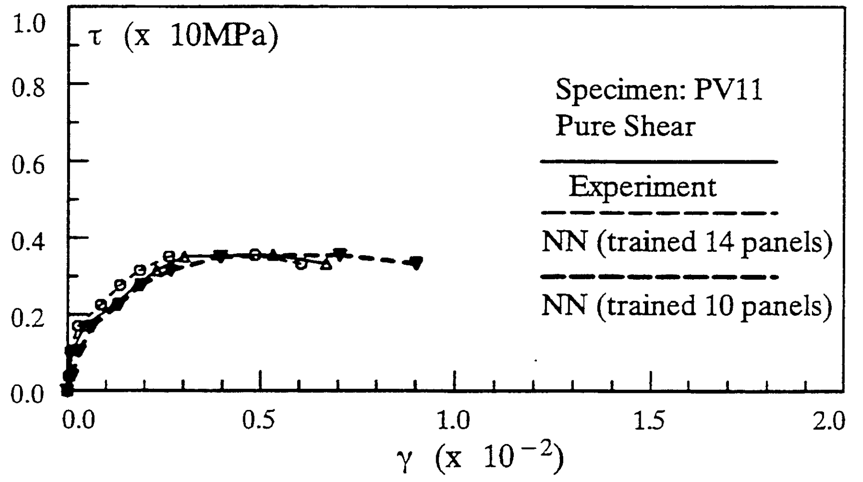
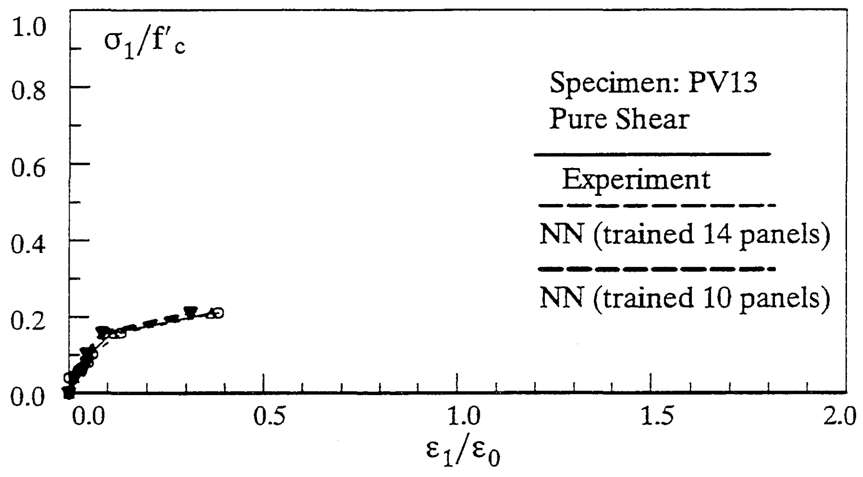
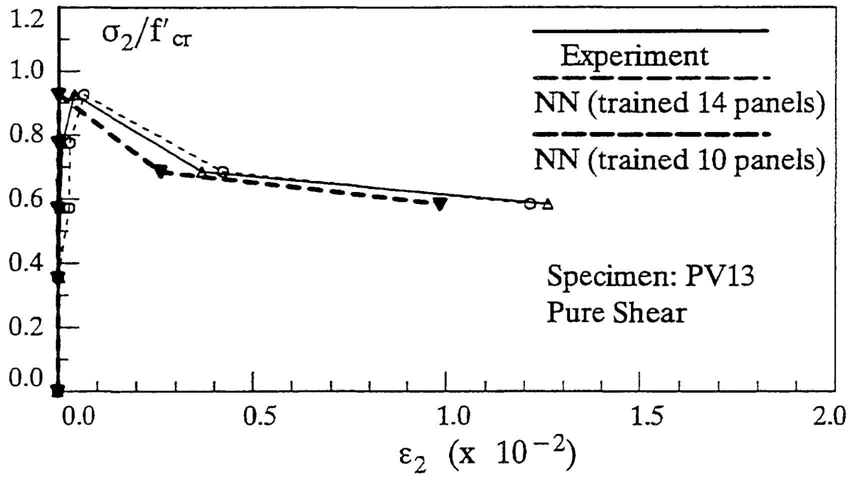


Fig. 7.10 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models





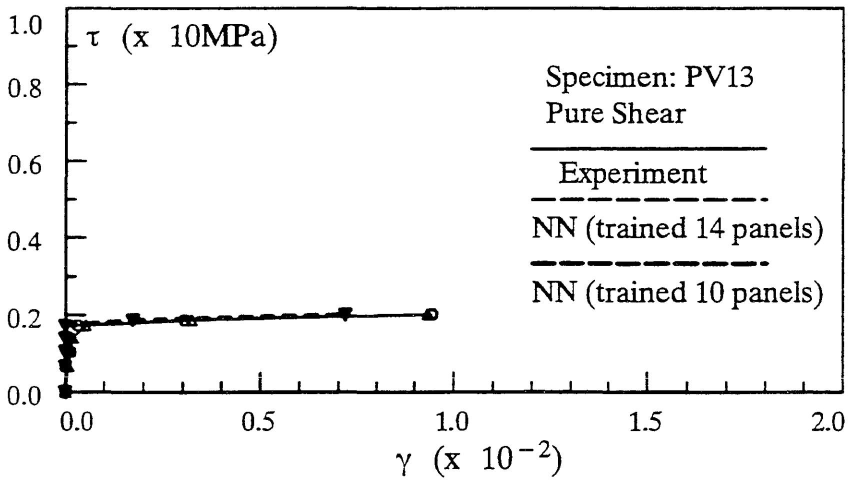
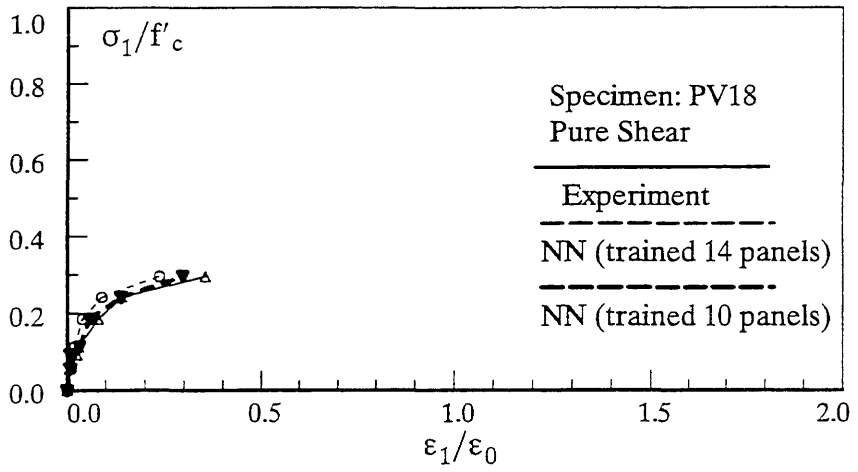
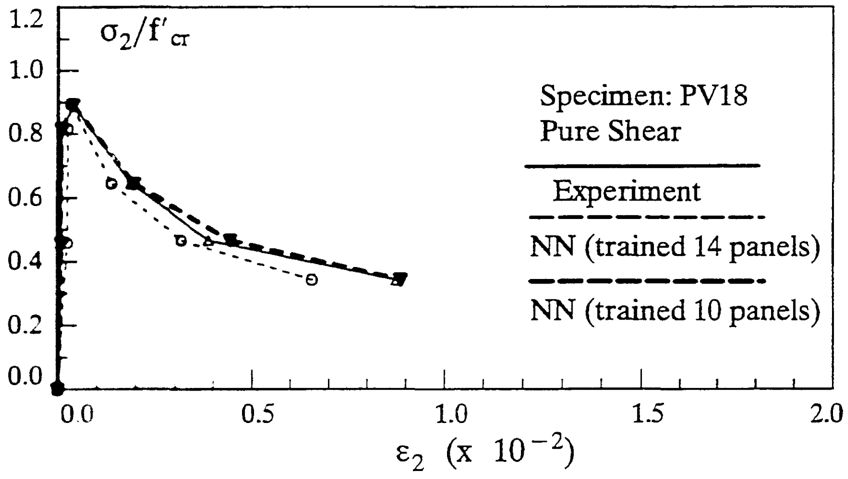


Fig. 7.11 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models





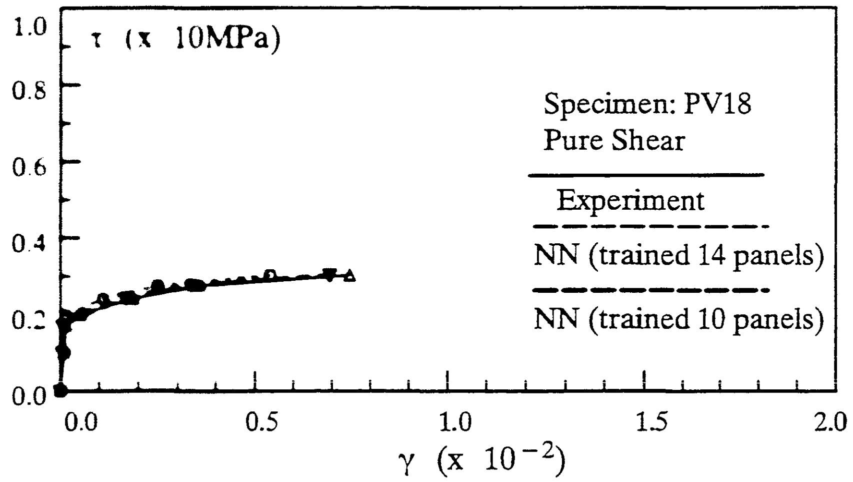


Fig. 7.12 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models

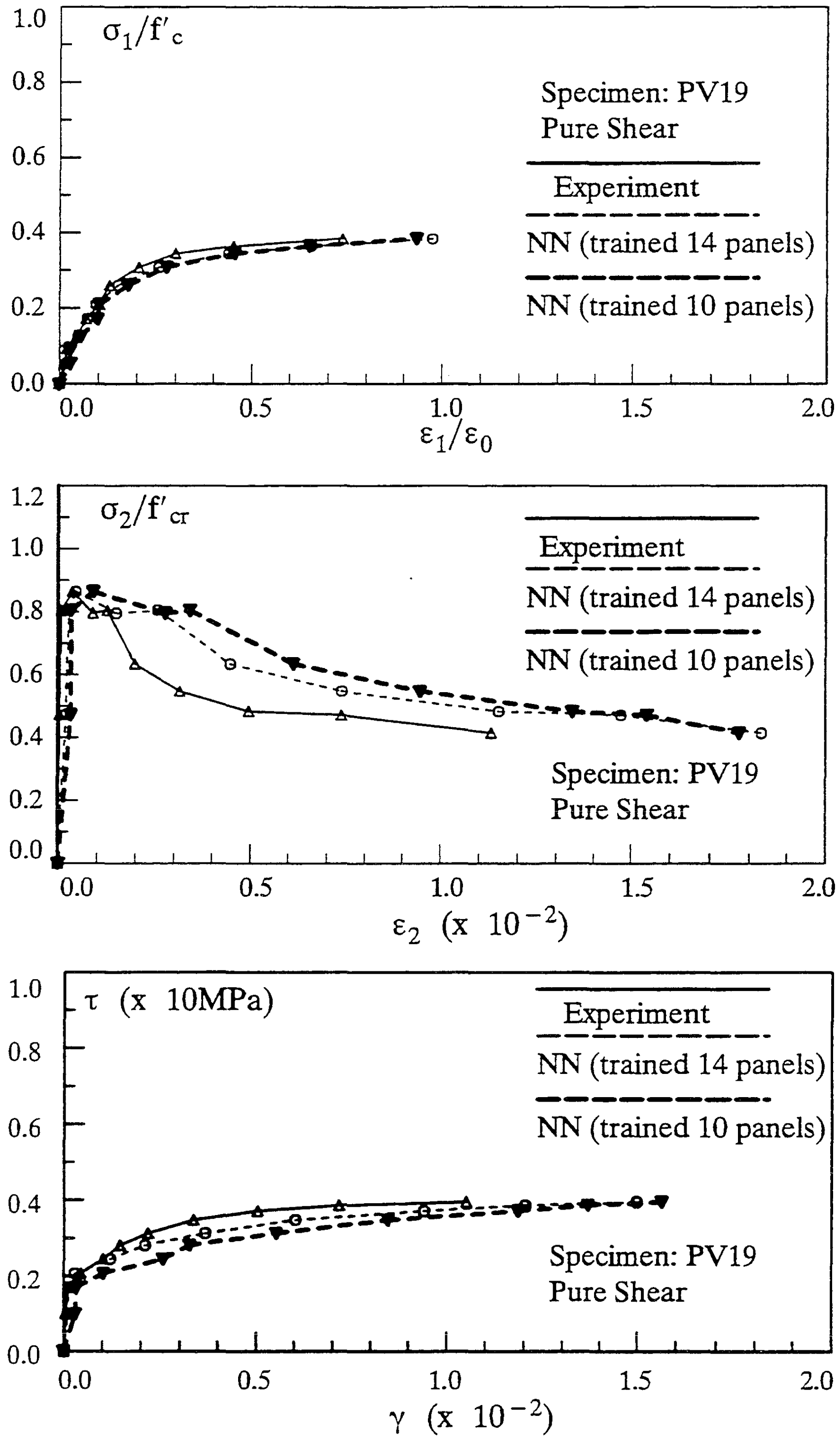


Fig. 7.13 – The Stress-Strain Relations Predicted by the Network – Training Results of Stress-controlled Models

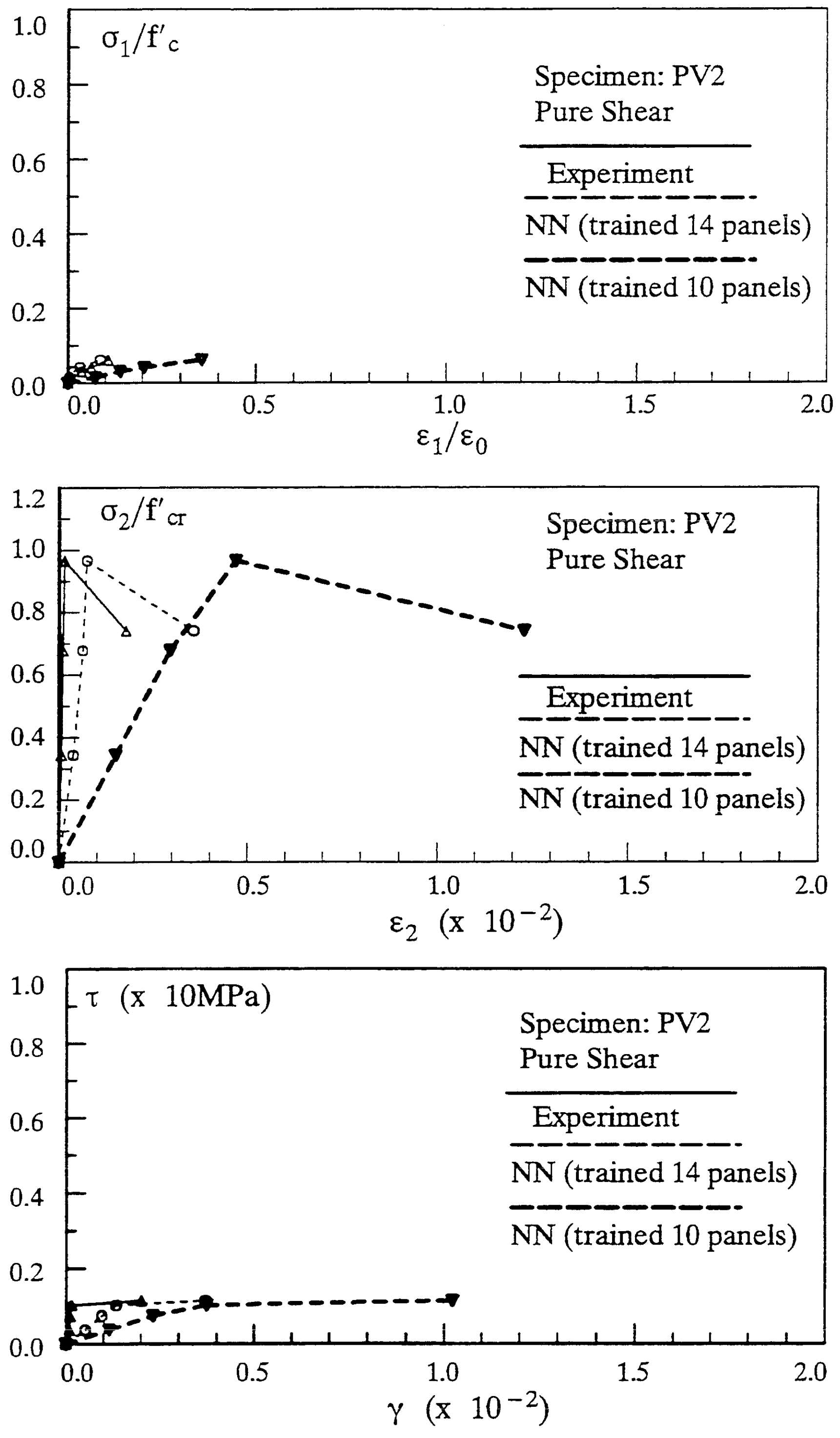


Fig. 7.14 – The Stress-Strain Relations Predicted by the Network – Testing Results with 10 Panels and Training Results with 14 Panels

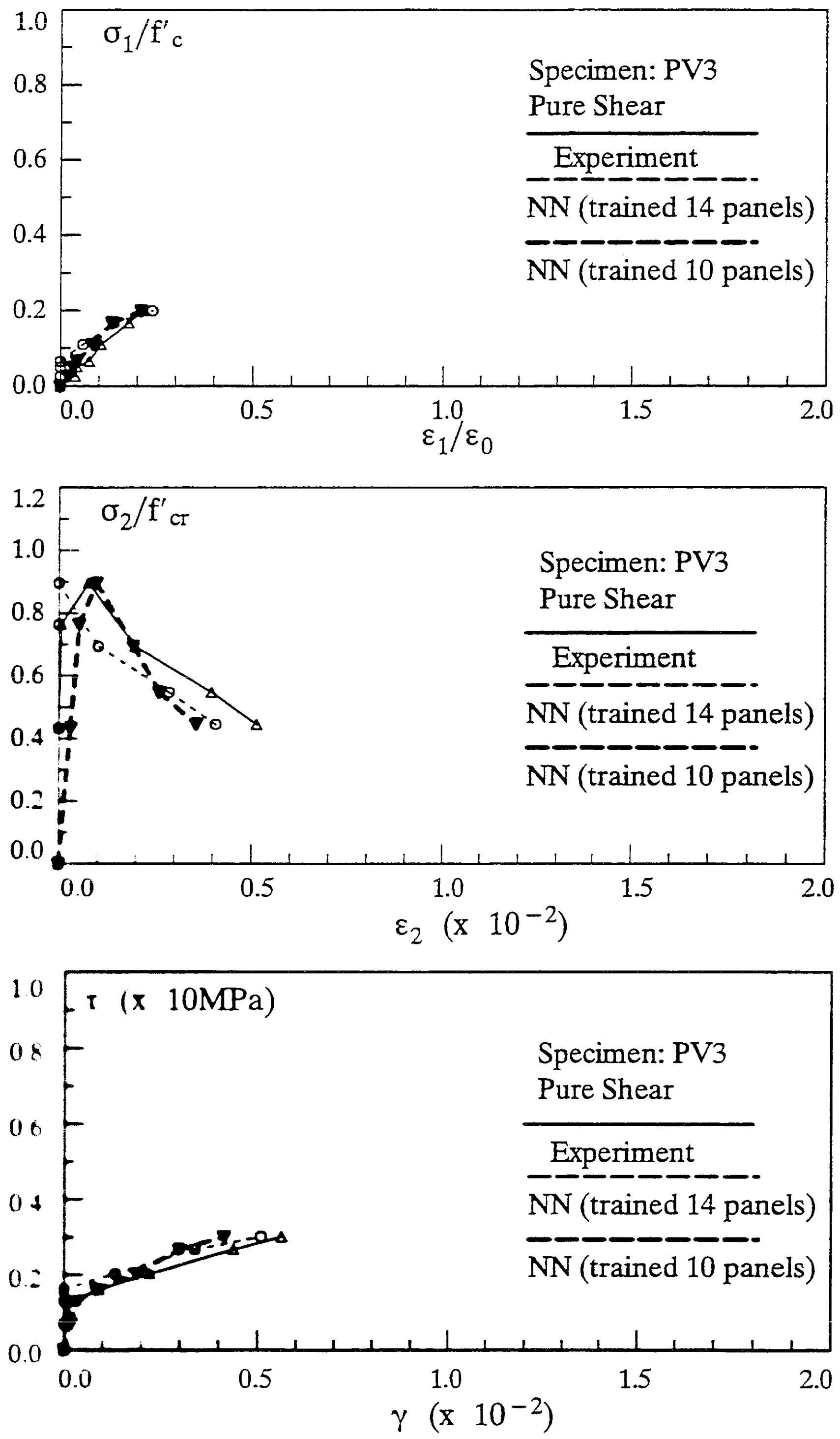
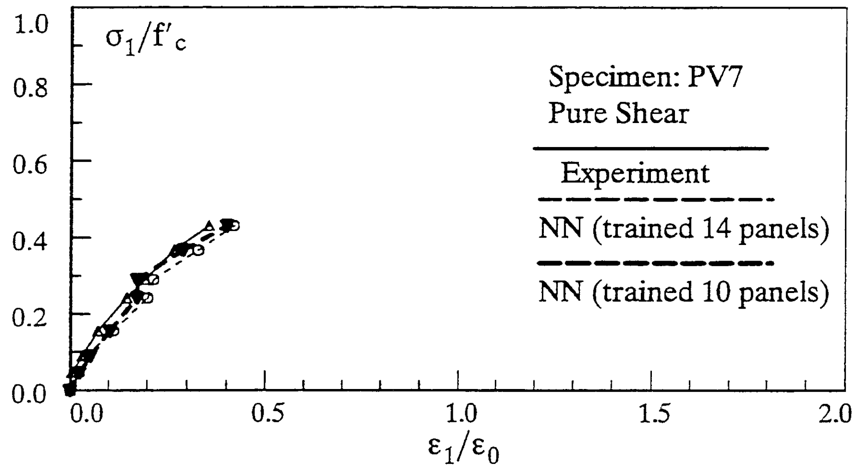
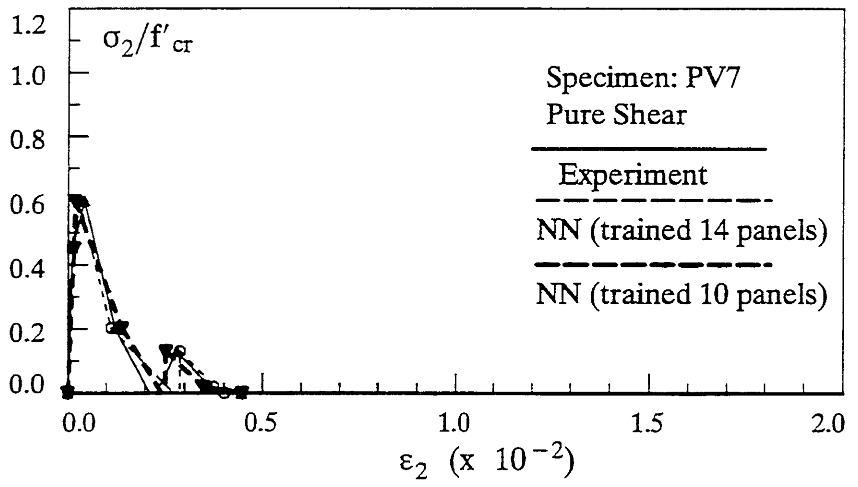


Fig. 7.15 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models





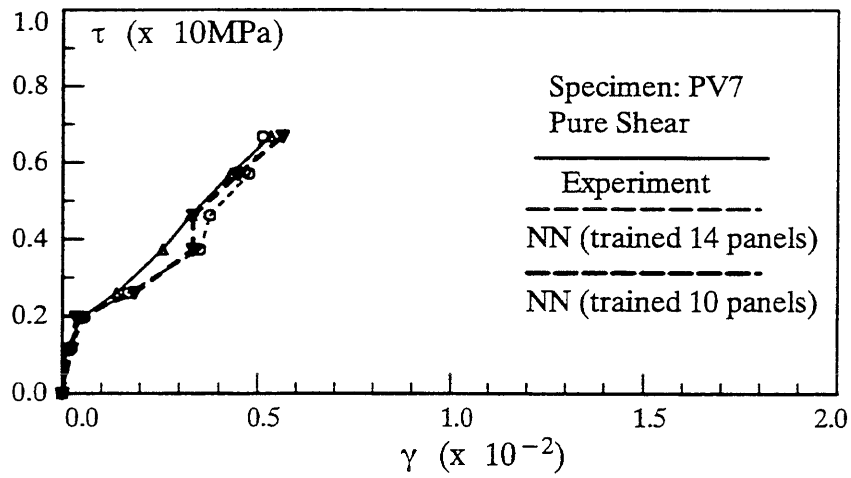


Fig. 7.16 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models

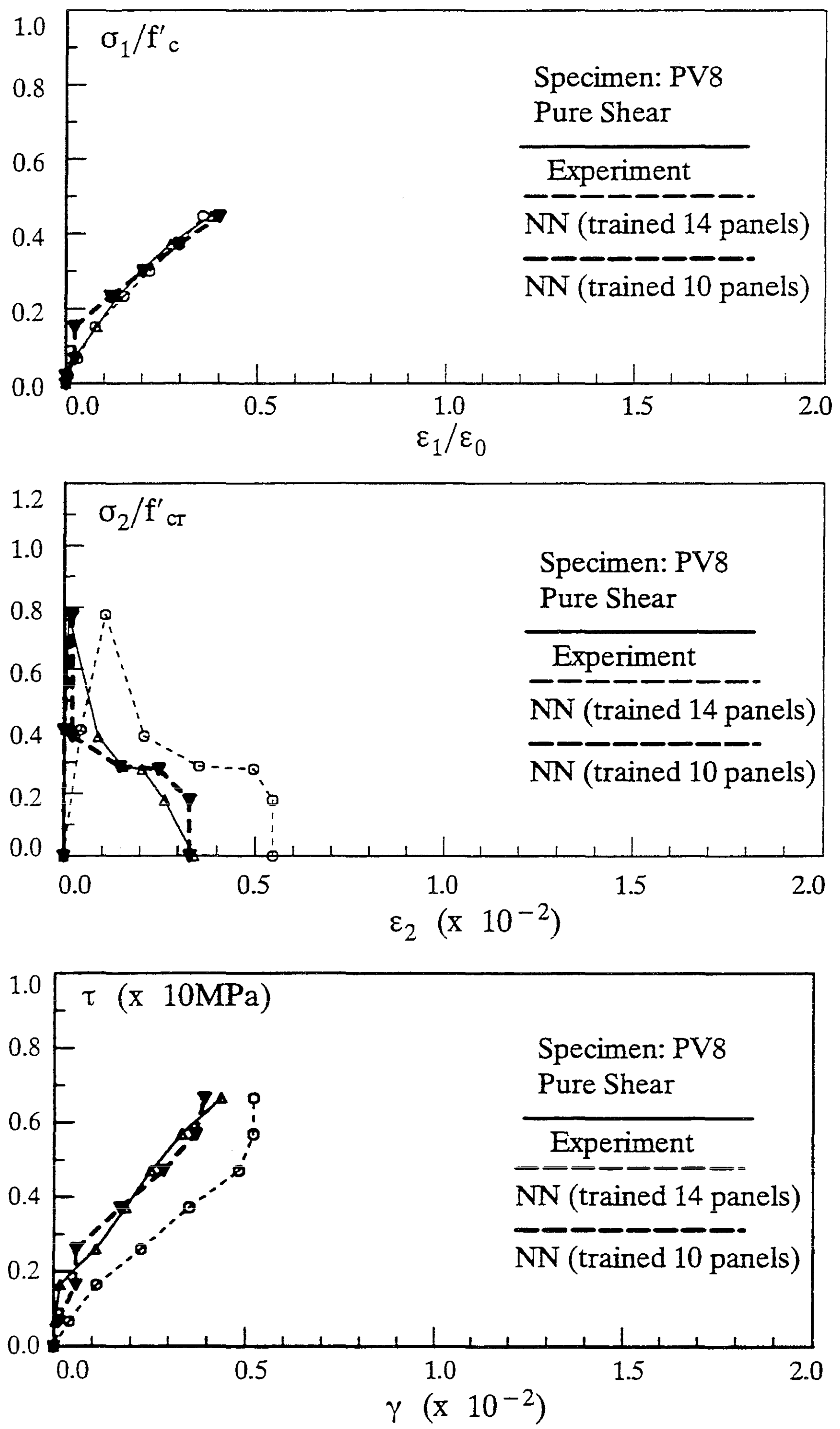


Fig. 7.17 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models

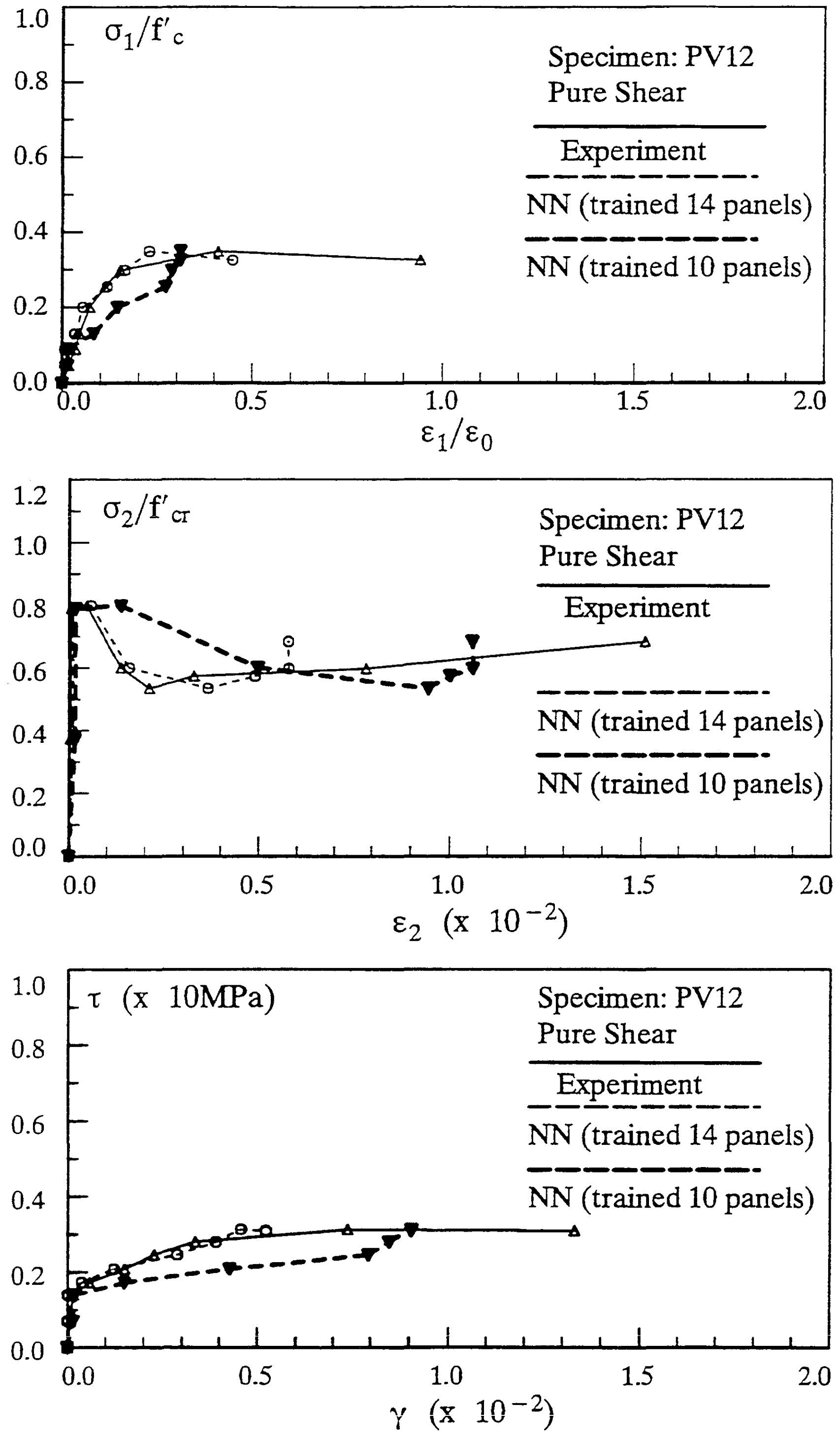


Fig. 7.18 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models

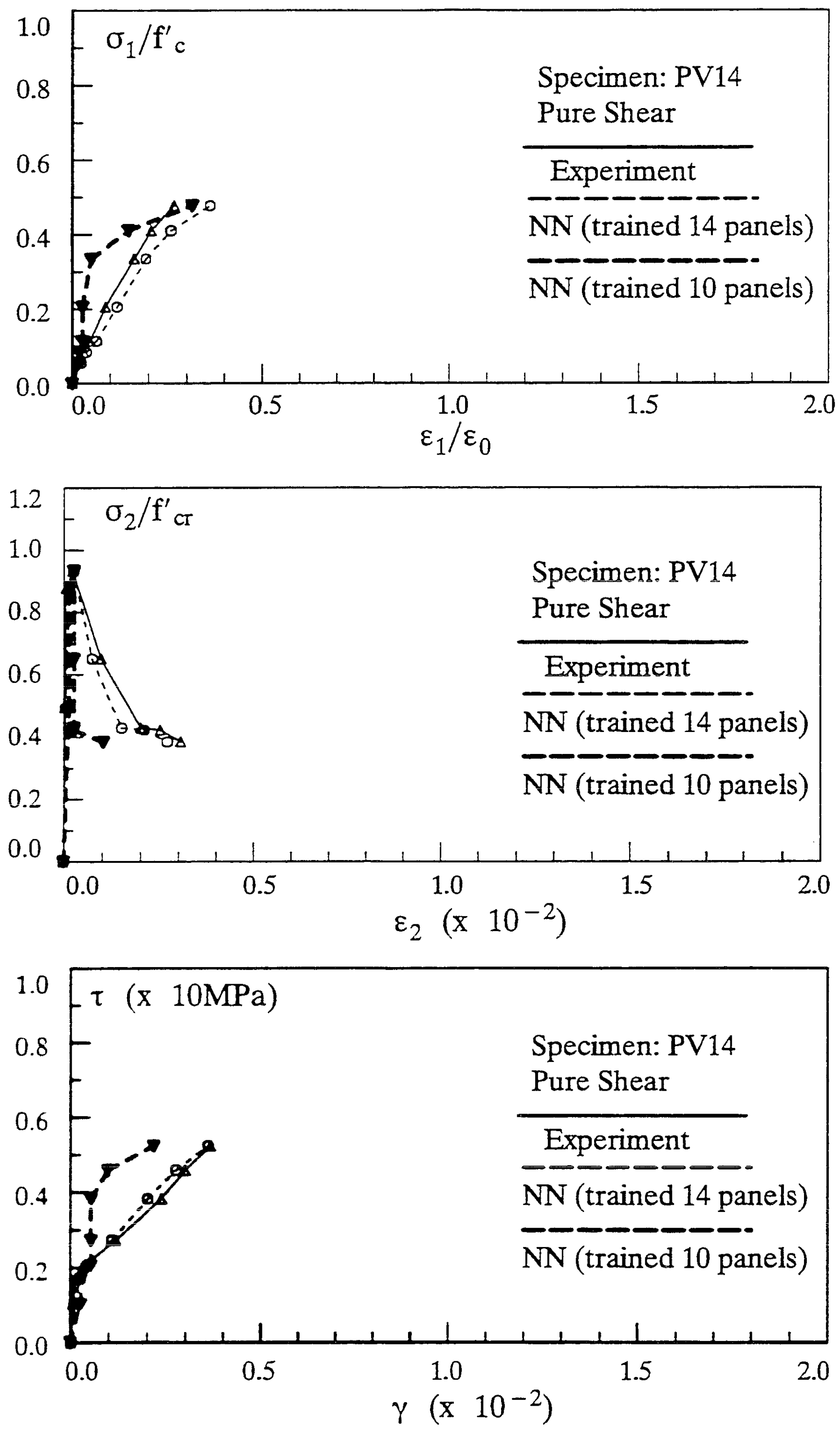
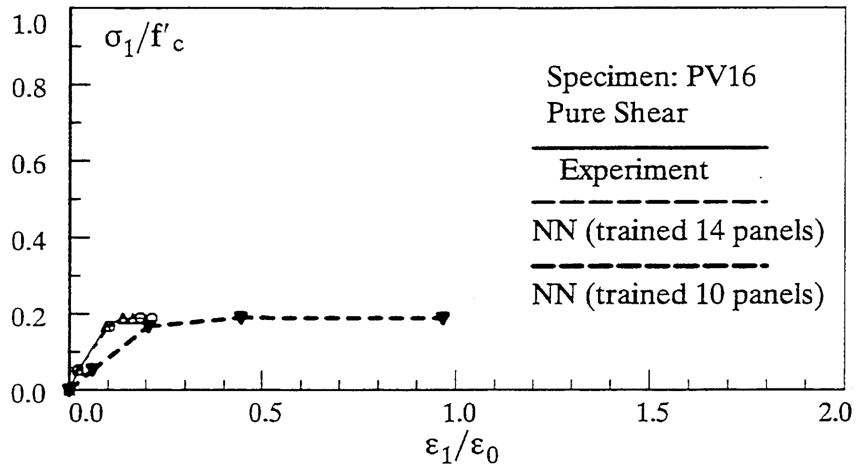
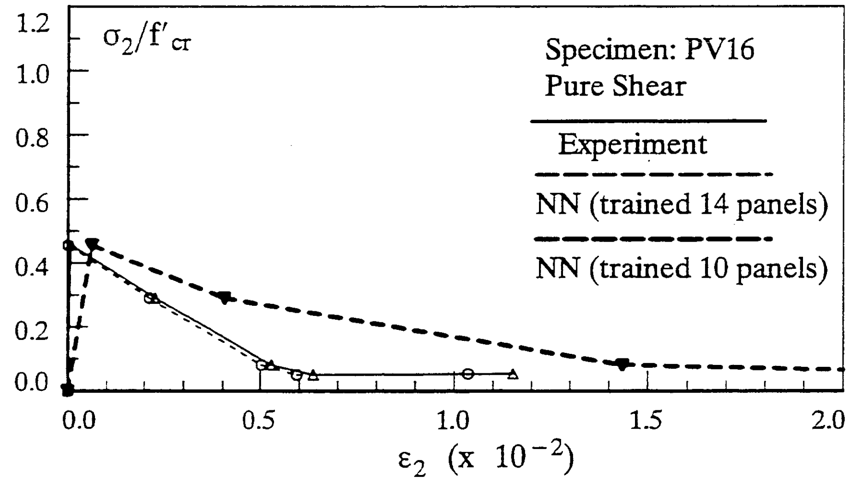


Fig. 7.19 – The Stress-Strain Relations Predicted by the Network – Testing Results with 10 Panels and Training Results with 14 Panels





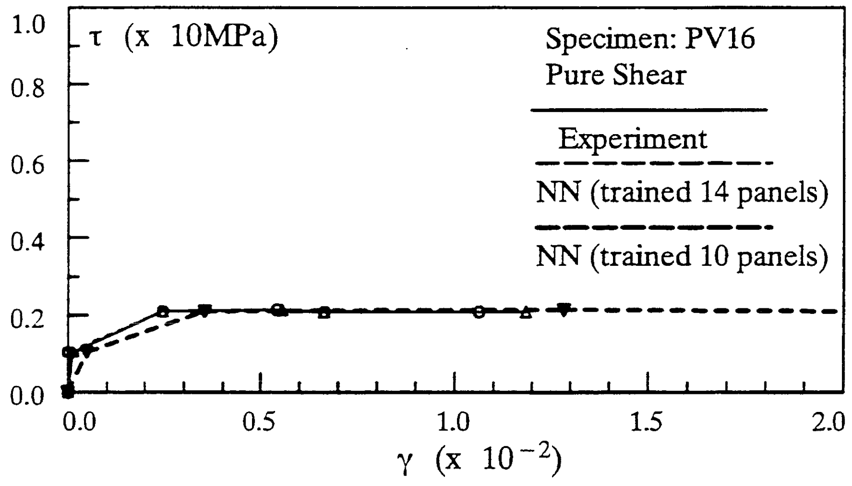
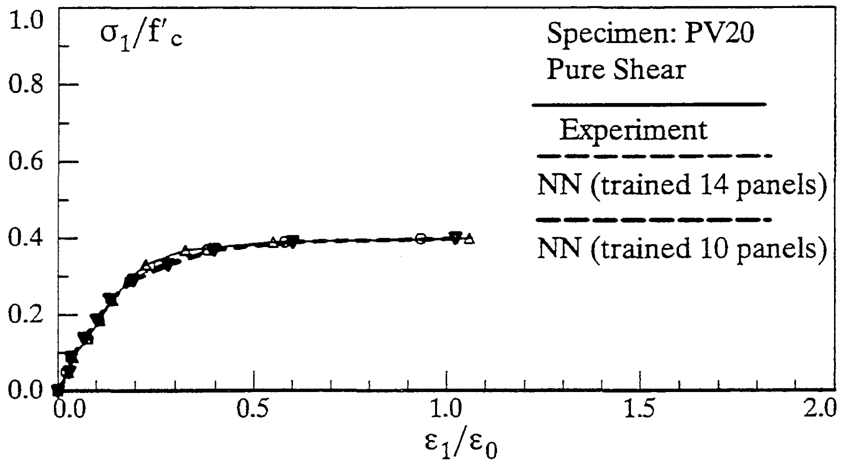
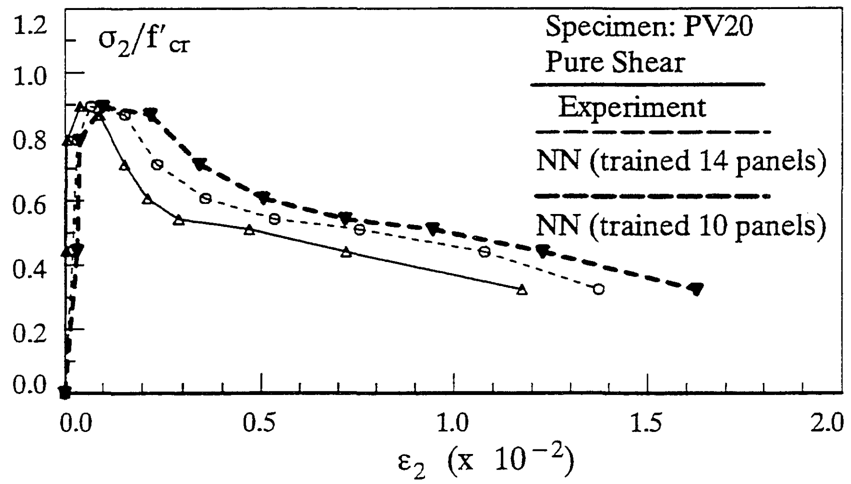


Fig. 7.20 – The Stress-Strain Relations Predicted by the Network – Testing Results with 10 Panels and Training Results with 14 Panels





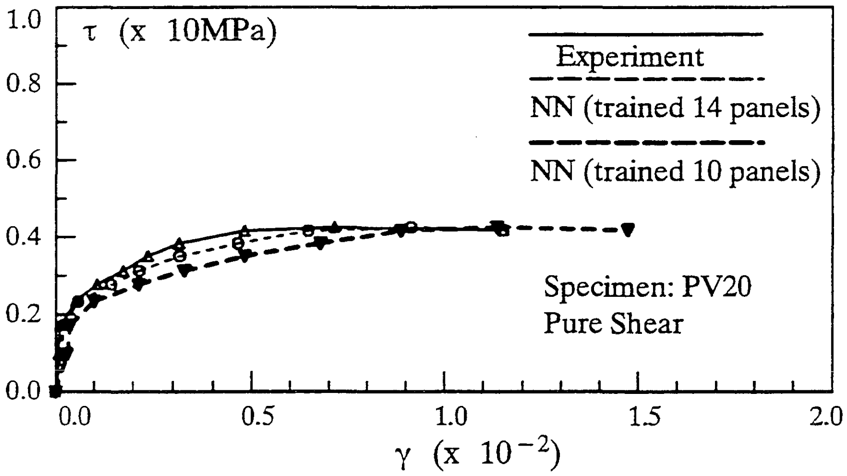


Fig. 7.21 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models

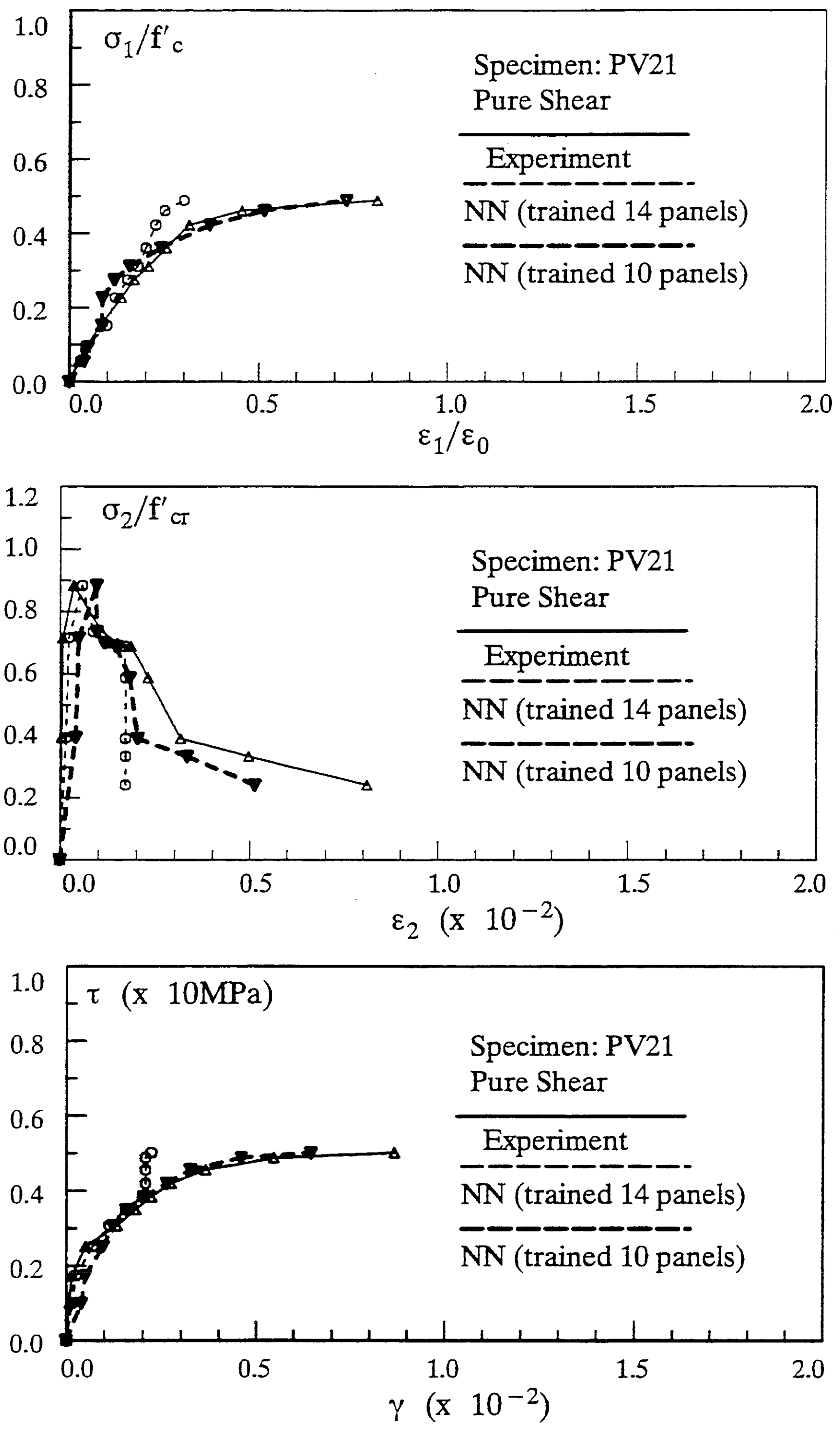
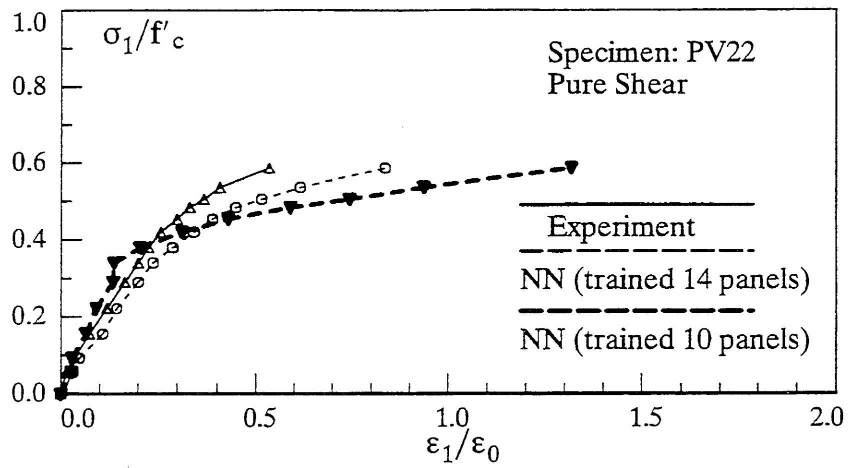
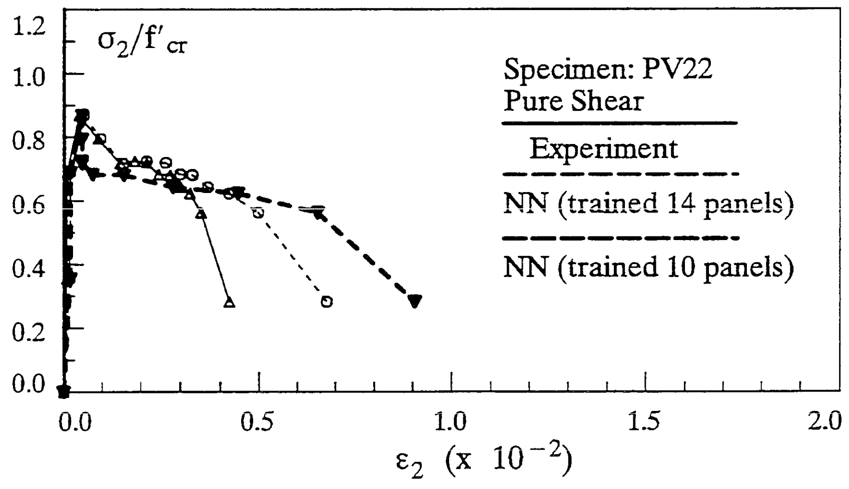


Fig. 7.22 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models





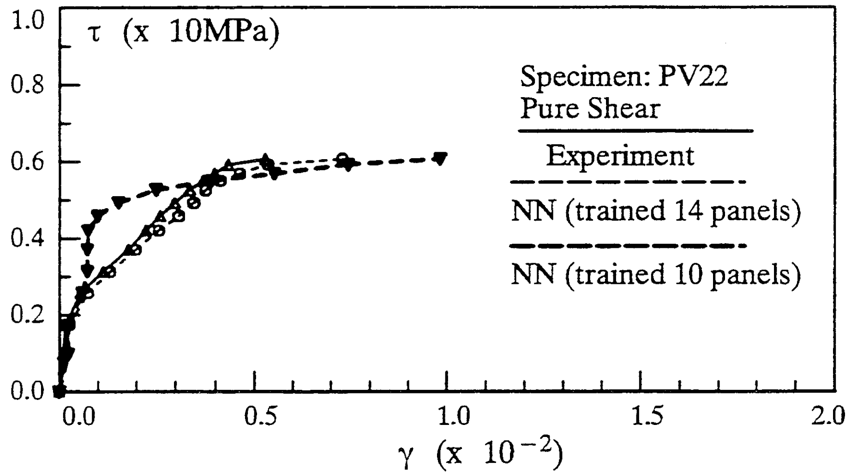
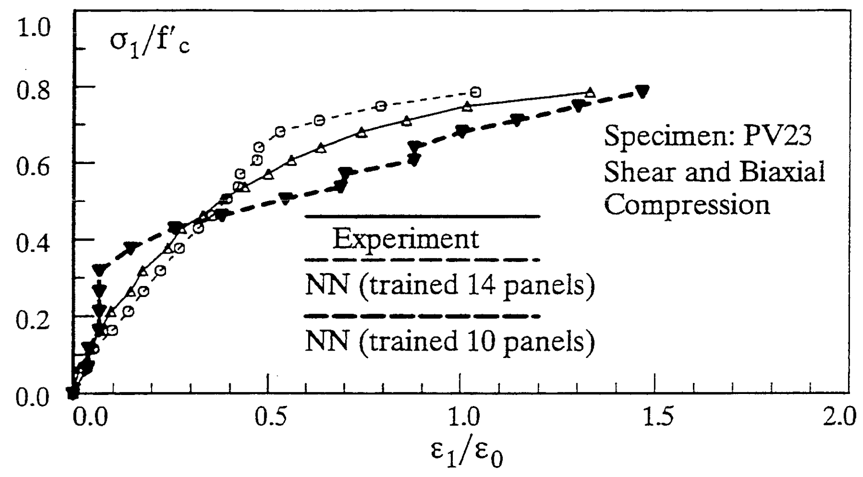
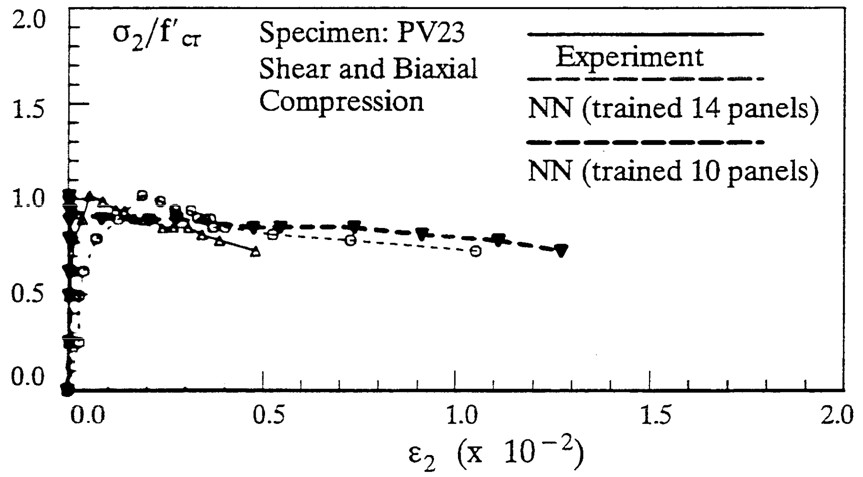


Fig. 7.23 – The Stress-Strain Relations Predicted by the Network – Testing Results with 10 Panels and Training Results with 14 Panels





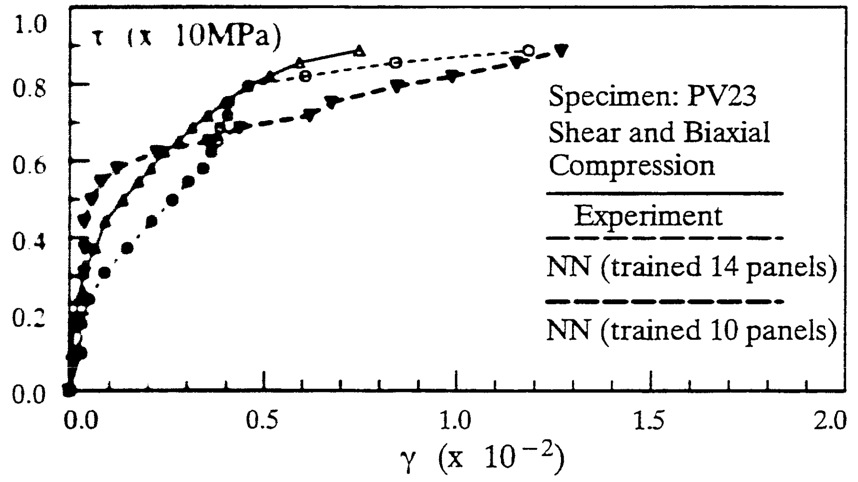


Fig. 7.24 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models

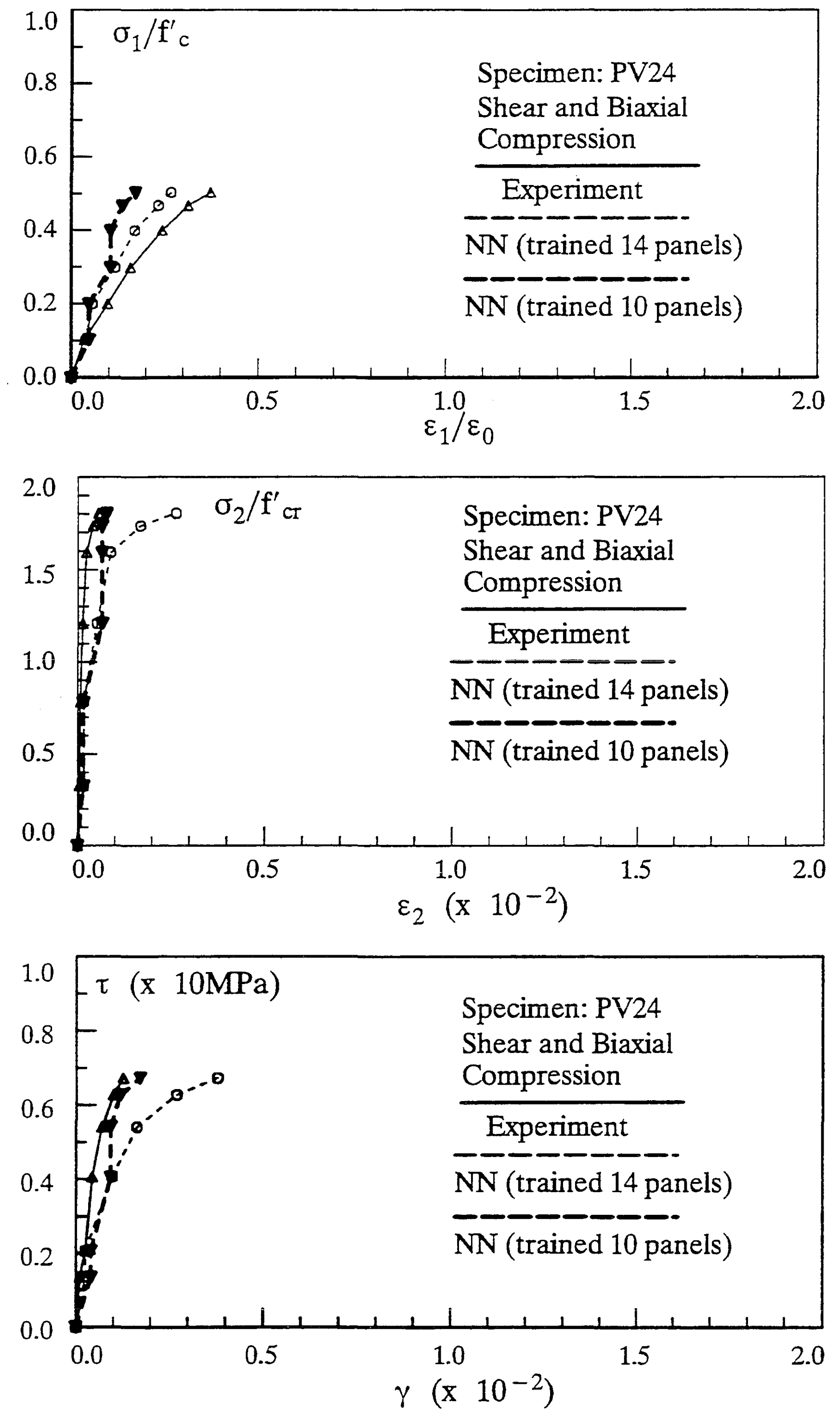
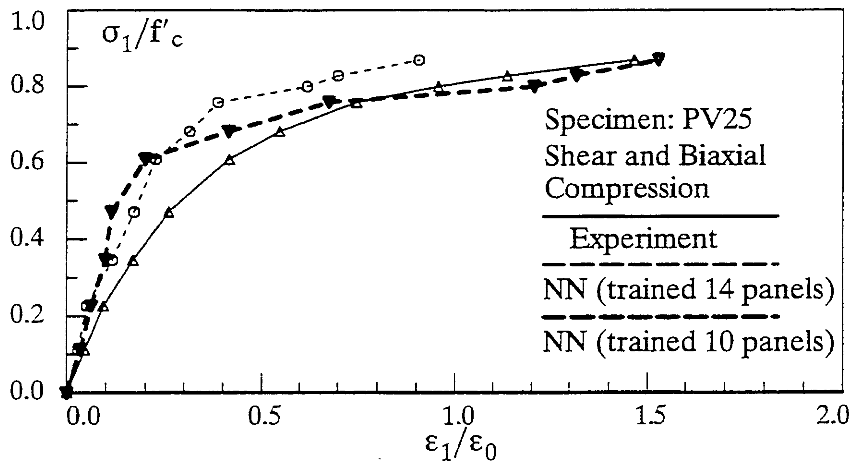
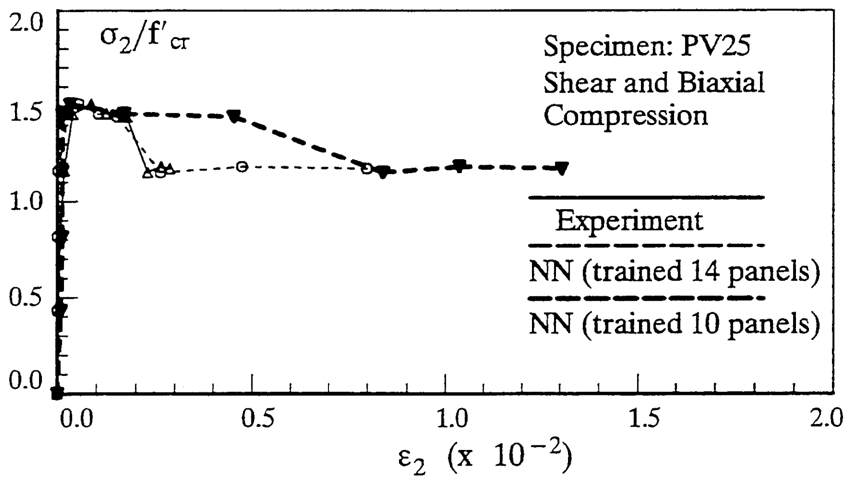


Fig. 7.25 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models





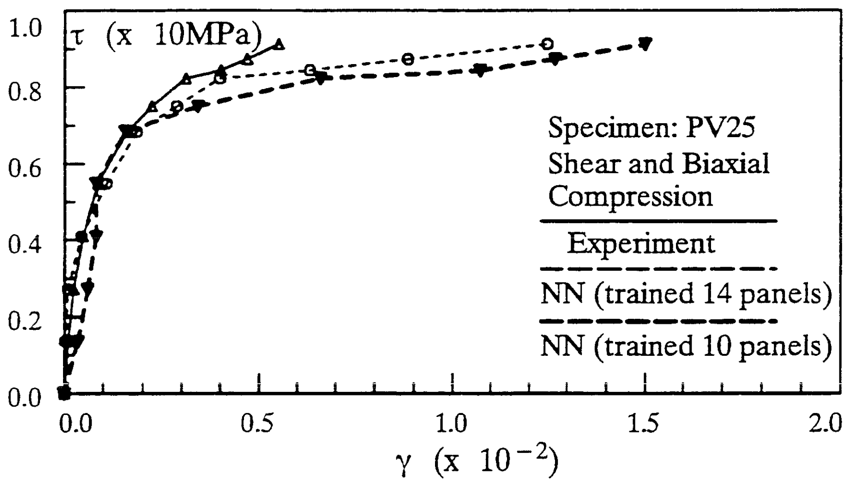


Fig. 7.26 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models

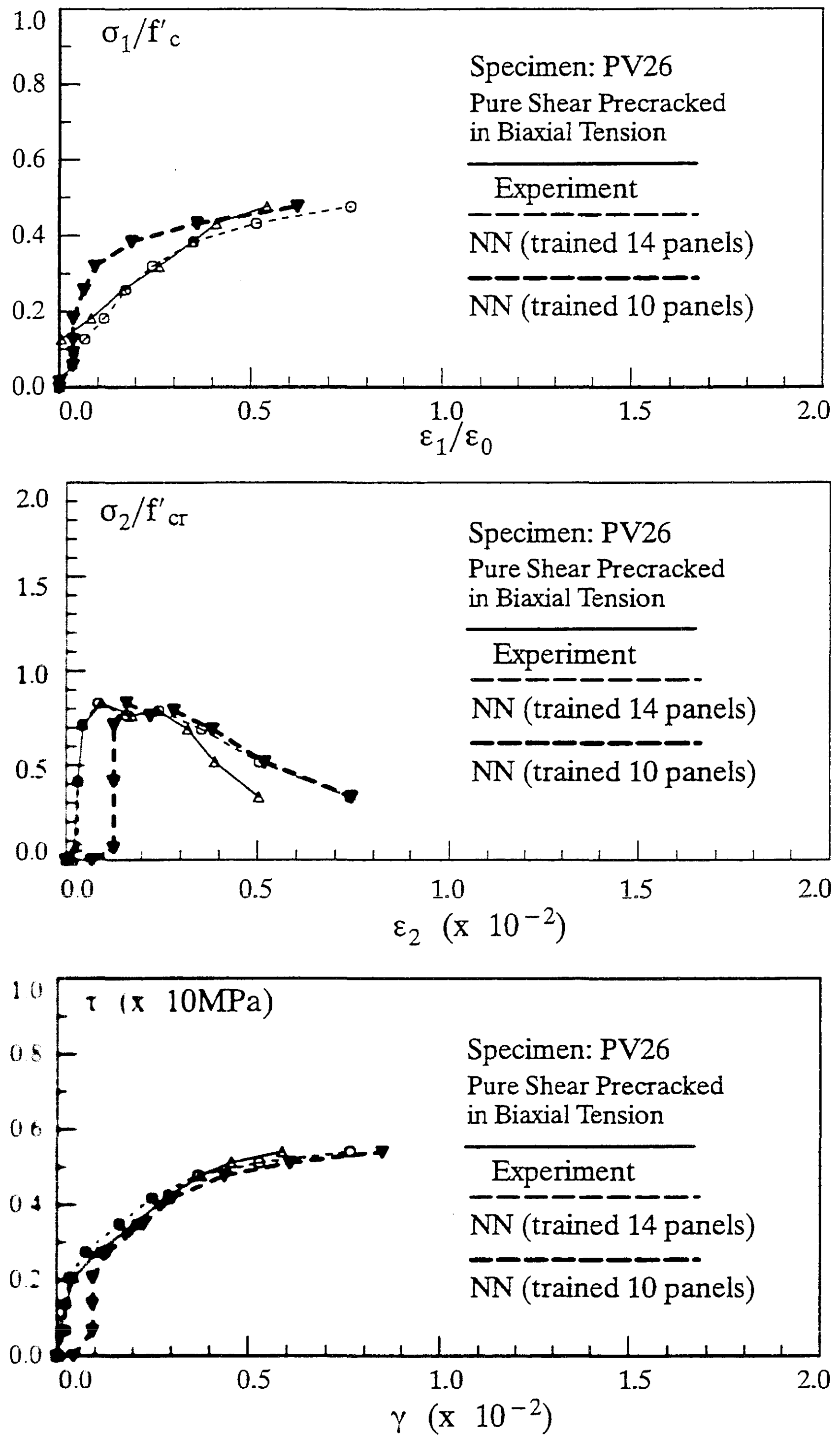
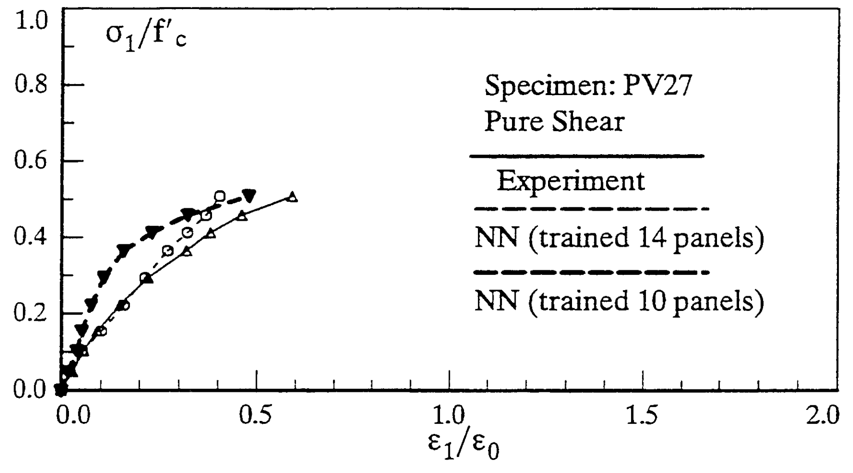
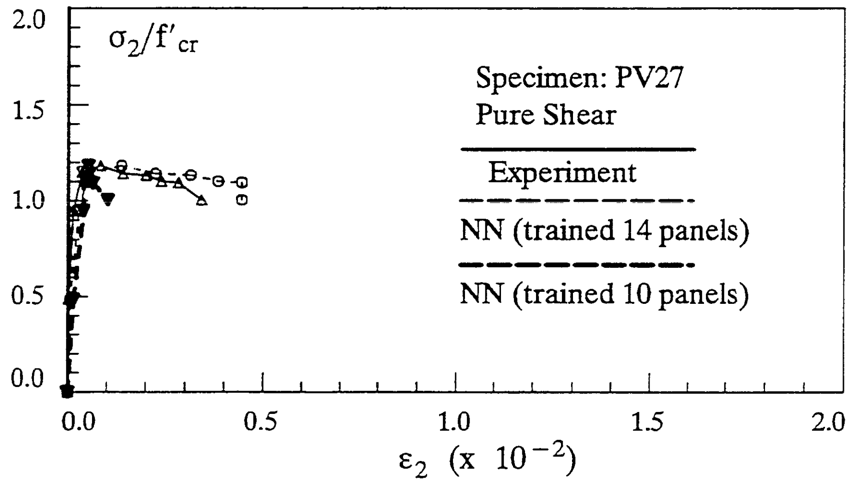


Fig. 7.27 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models





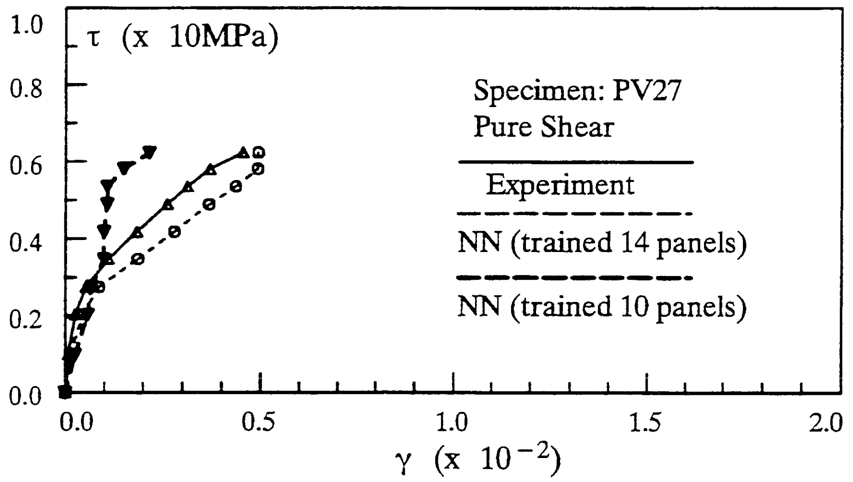
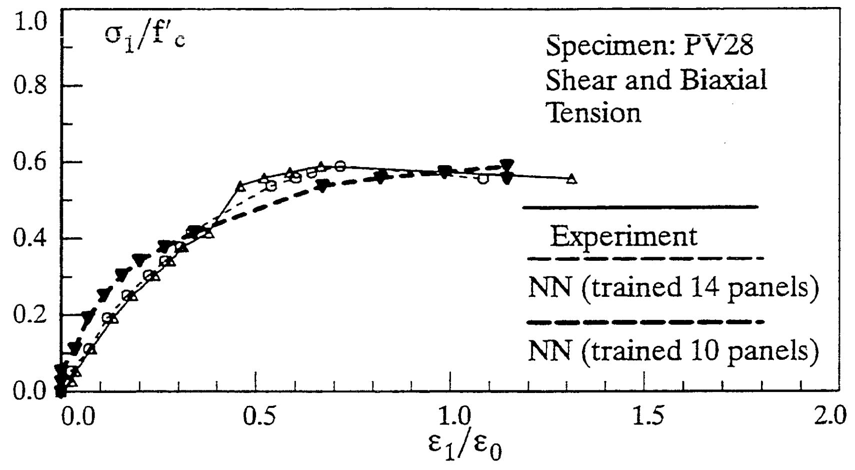
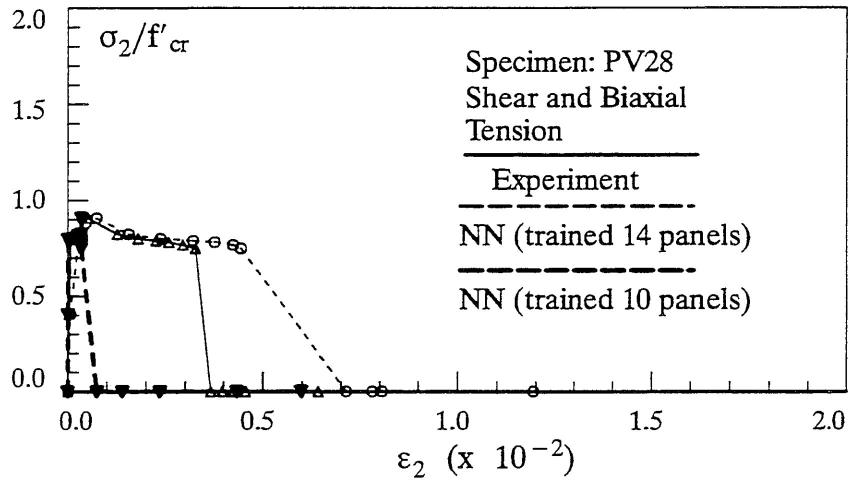


Fig. 7.28 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models





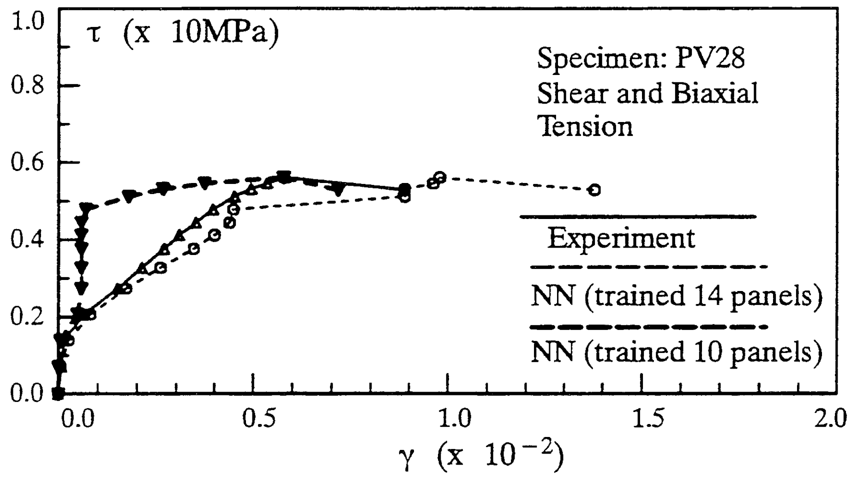
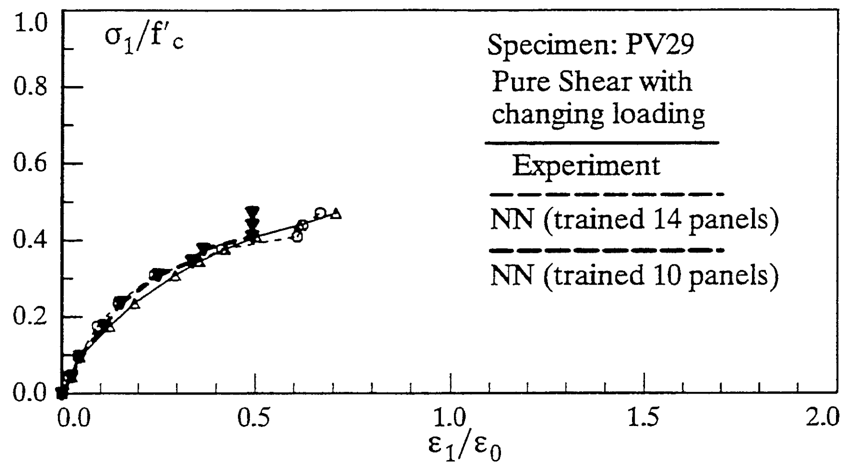
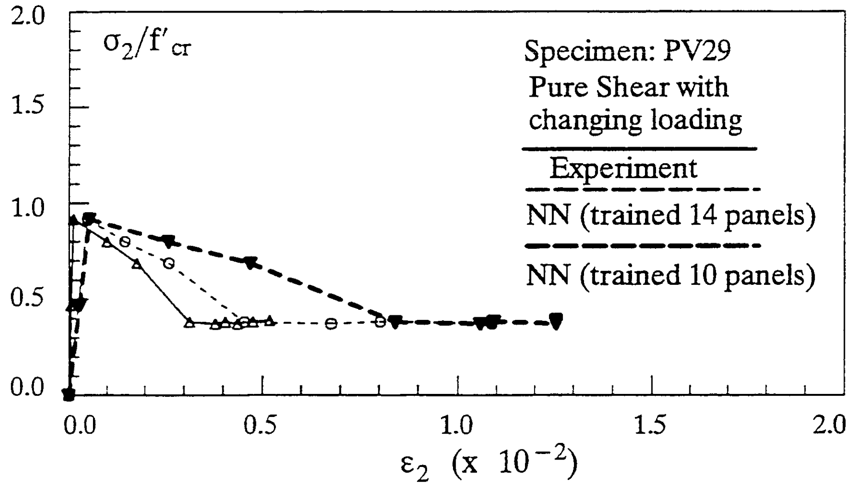


Fig. 7.29 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models





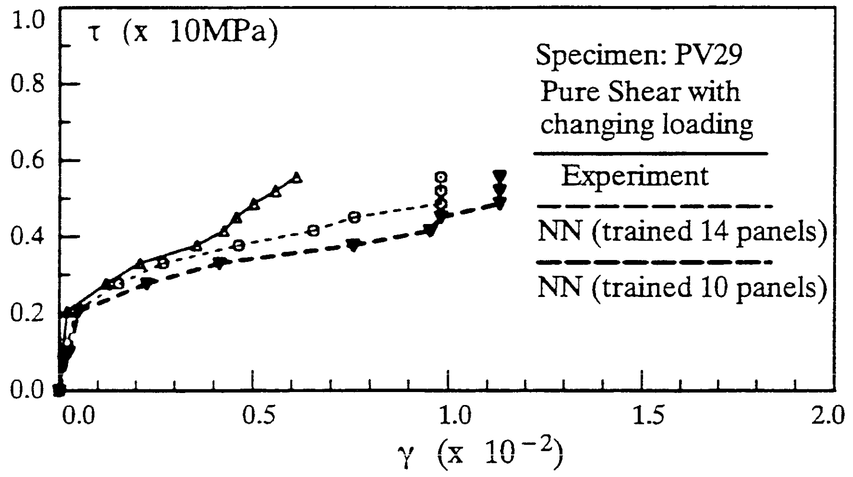


Fig. 7.30 – The Stress-Strain Relations Predicted by the Network – Testing Results of Stress-controlled Models

5.4.3 A Strain-controlled Model

A strain-controlled constitutive model of reinforced concrete in biaxial stress states was also developed in this study. In a strain-controlled model, the neural network is to predict stress increments when presented with strain increments. The three-point representation scheme of the strain-controlled model was obtained directly from that of the stress-controlled model by switching the stress increments to the output and the strain increments to the input. The training data set of the strain-controlled model was also constructed accordingly.

Following the same approach with the stress-controlled model, a neural network architecture with two hidden layers was used for the strain-controlled model. At the onset of training, there were 10 hidden nodes in each hidden layer. After the stress-strain data of 14 panels tested in pure shear were successfully trained, the final number of hidden nodes in each of the hidden layer was determined to be 25. As with strain-controlled model of plain concrete, training of the strain-controlled reinforced concrete model required more efforts than with stress-controlled model. Furthermore, it was harder to push the accuracy of training or to reduce the maximum training error. Therefore, the convergence criterion of training was relaxed in order to facilitate the learning convergence. The learning process was stopped when the maximum training error had been reduced to below 0.10. Testing of the network also proceeded incrementally as before.

After training, the network was tested on the trained and untrained panels to evaluate its performance on learning and generalization. The training and testing results represented as stress−strain relations are presented in Figs. 4.33 − 4.59, in which the predictions from the stress-controlled model are also included for comparison. In those figures, the stress-strain relations predicted by the strain-controlled model are plotted as thick dotted lines, the experimental data as solid lines, and those of the stress-controlled model as thin dotted lines. From studying training results shown in Figs. 4.33 – 4.46, it is apparent that both stress-controlled and strain-controlled models learned the training data equally well, though the later had a slightly larger learning error. For testing results shown in Figs 4.47 – 4.59, the strain-controlled model performs reasonable on almost all the testing cases. However, the overall predictions on all the testing cases from the stress-controlled model are more accurate than those from the strain-controlled model. This difference in performance was primarily caused by the slack learning accuracy with training of the strain-controlled model.

As has been discussed before, the accuracy of network predictions for each panel is not evenly distributed among the three stress-strain relations, rather that it varies with the individual stress-strain case. The predictions on the principal compressive stress-strain relation and that of shear stress-strain relation for both models are better than those on the principal tensile stress-strain relation for all the cases that were tested both in pure shear and in combined shear and normal stresses. However, the stress-controlled model has better predictions on the principal tensile stress-strain relation than the strain-controlled model. Moreover, for panels PV8, PV23, and PV29, the tensile stress–strain behaviors during the later stage of loading, were not fully captured in the strain-controlled model. This indicates that further training with additional training data from these three panels needs to be conducted in order to capture the material behavior exhibited in these test results within the strain-controlled model.

When the stress–strain predictions from both stress-controlled and strain–controlled models are plotted in the same figure, it should be noted that the error of prediction is accumulated and displayed in the strain components with a stress–controlled model, and in the stress components with a strain-controlled model, because of the way that incremental testing is performed. Obviously, the performance of the strain-controlled model would be further improved with additional training on an updated training data set.

In spite of the difference between the testing results from strain-controlled and stress-controlled models, the predictions from the strain-controlled model are generally reasonable and of acceptable accuracy in almost all of the cases tested in pure shear and in most of the cases tested in combined shear and normal stresses. Therefore, it can be concluded that the strain-controlled model is able to capture the material behavior from testing results obtained from stress-controlled experiments. With a dedicated scaling scheme in the preparation of training data for a strain-controlled model, further improvements on the accuracy of training and testing can be expected.

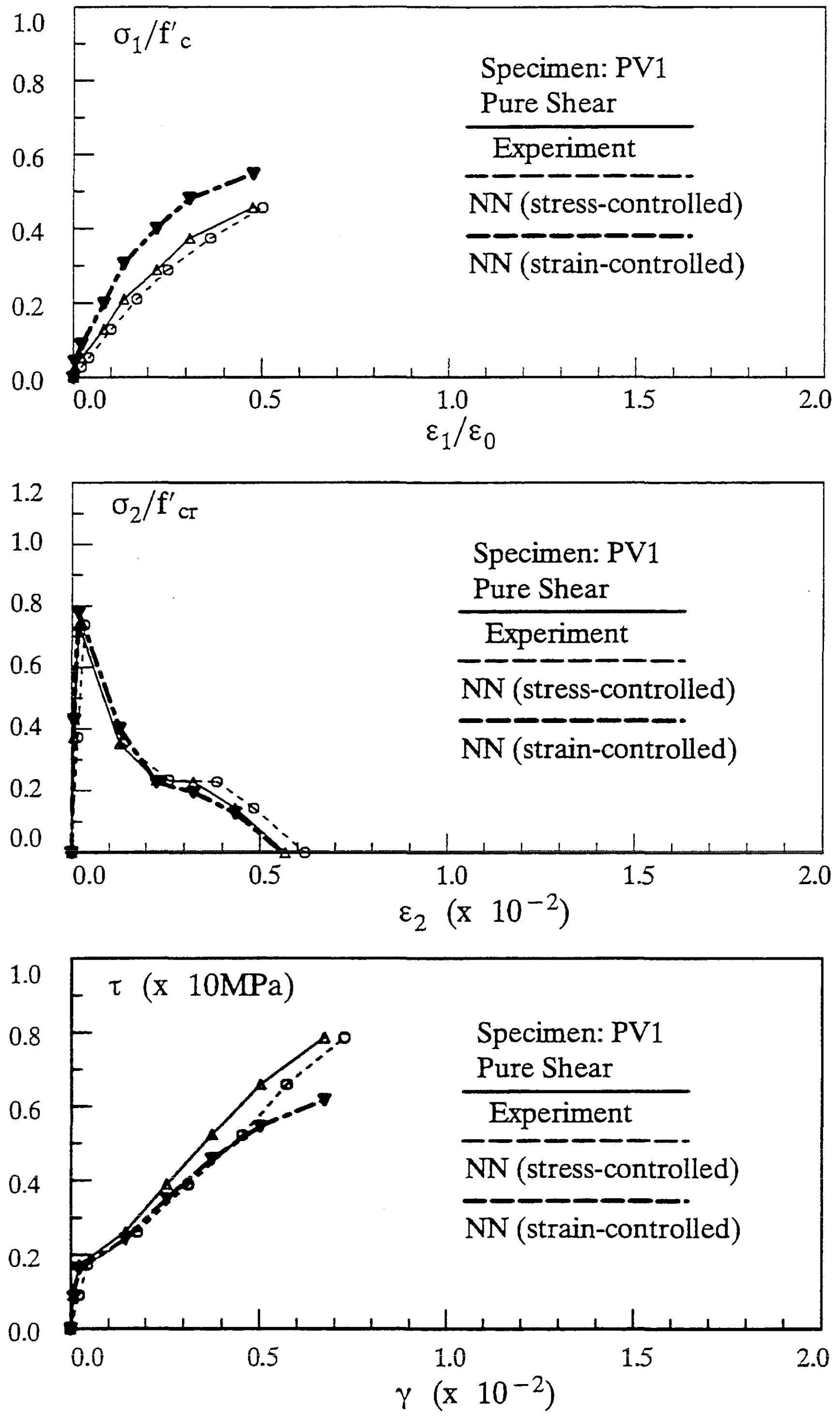
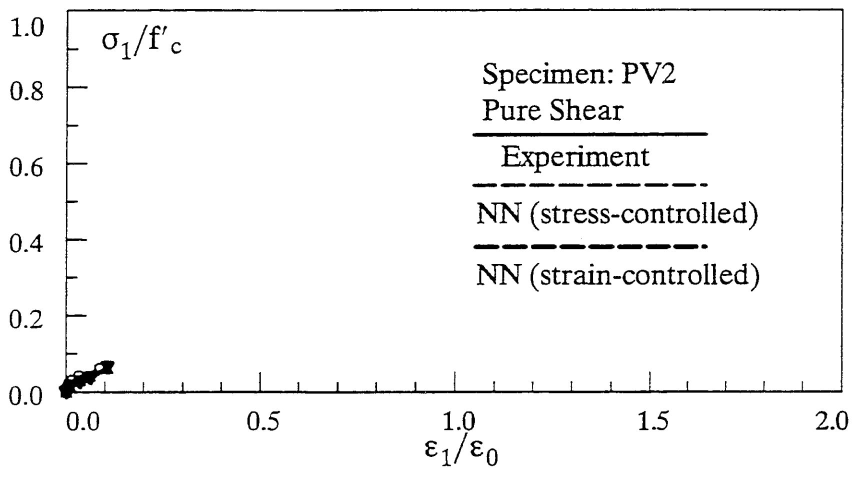
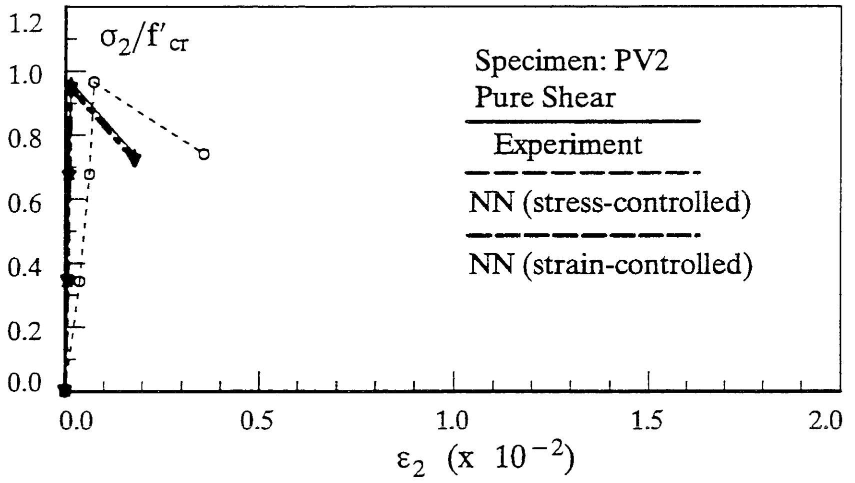


Fig. 4.33 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)





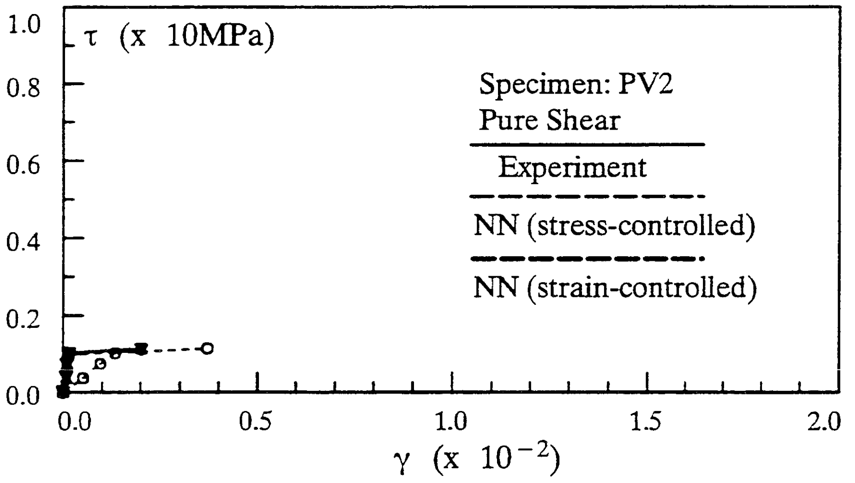


Fig. 4.34 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

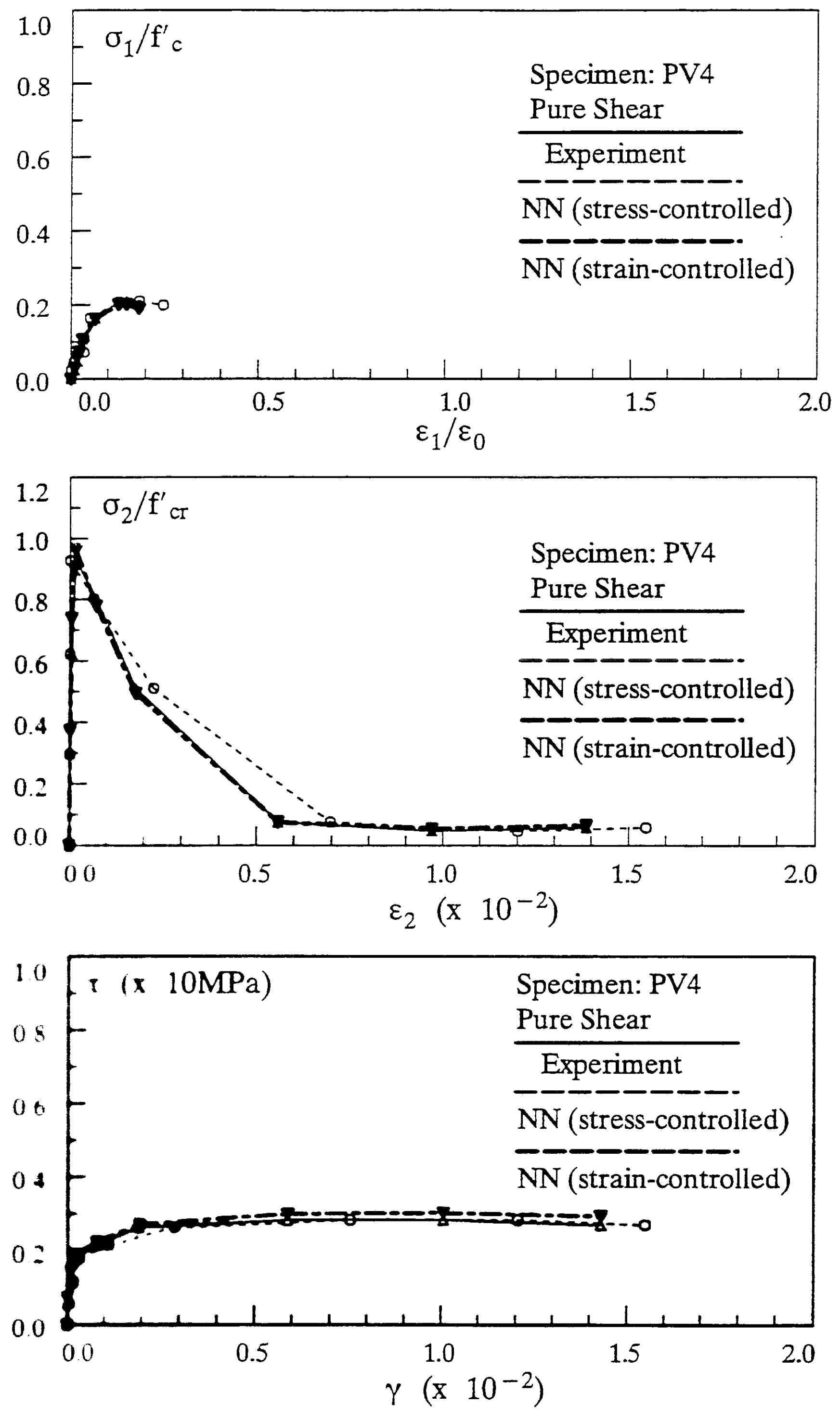


Fig. 4.35 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

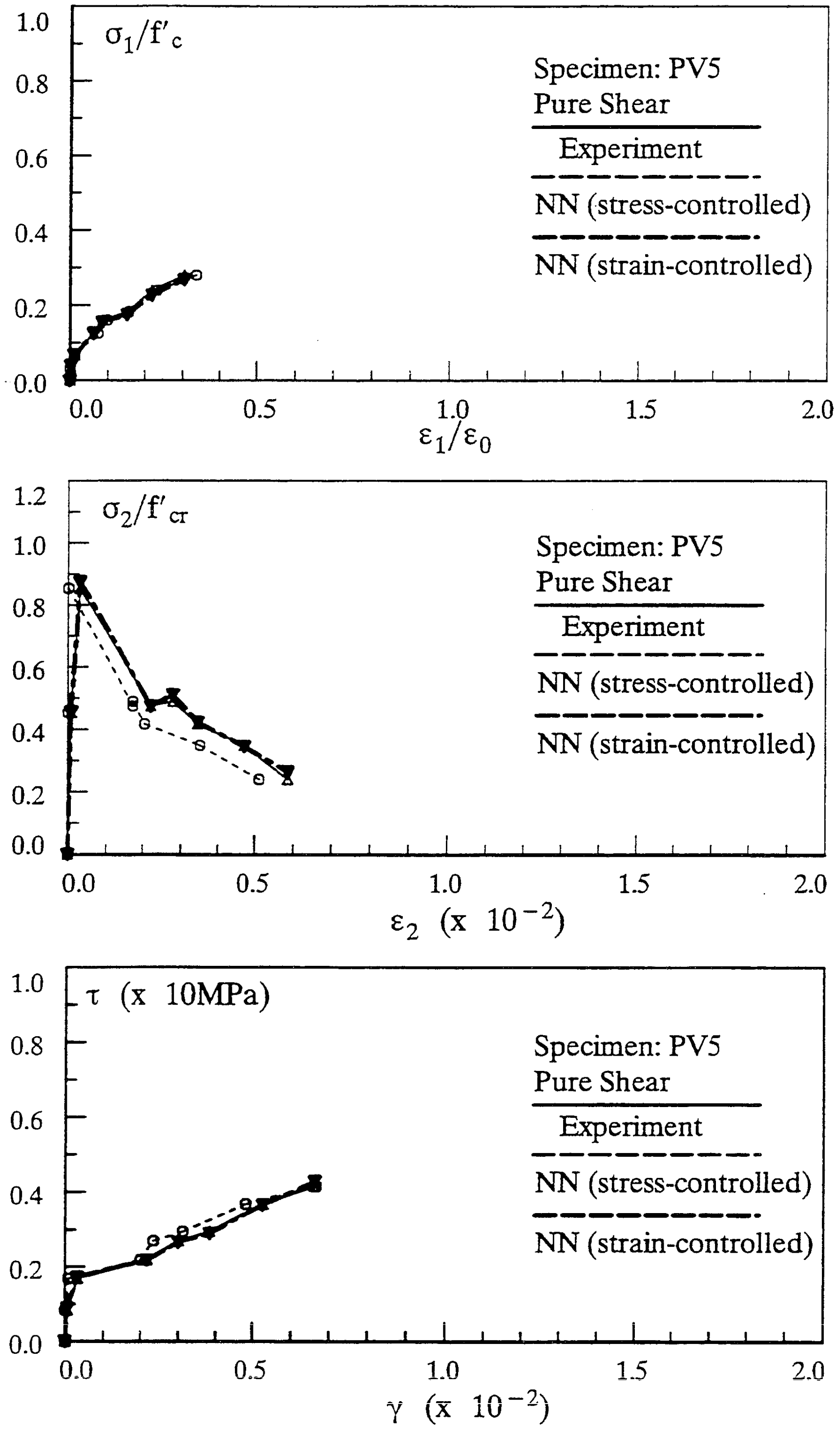


Fig. 4.36 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

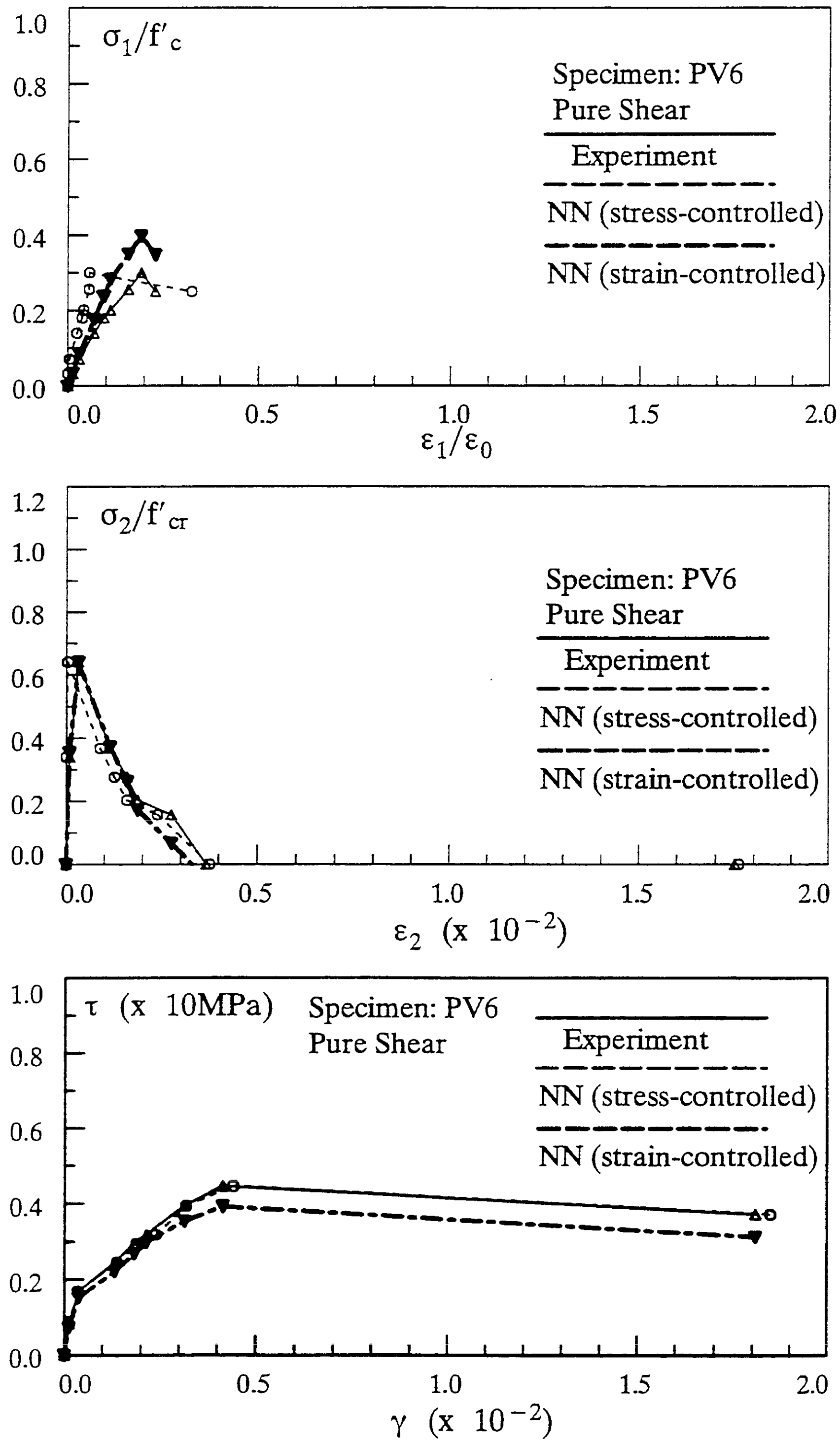


Fig. 4.37 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

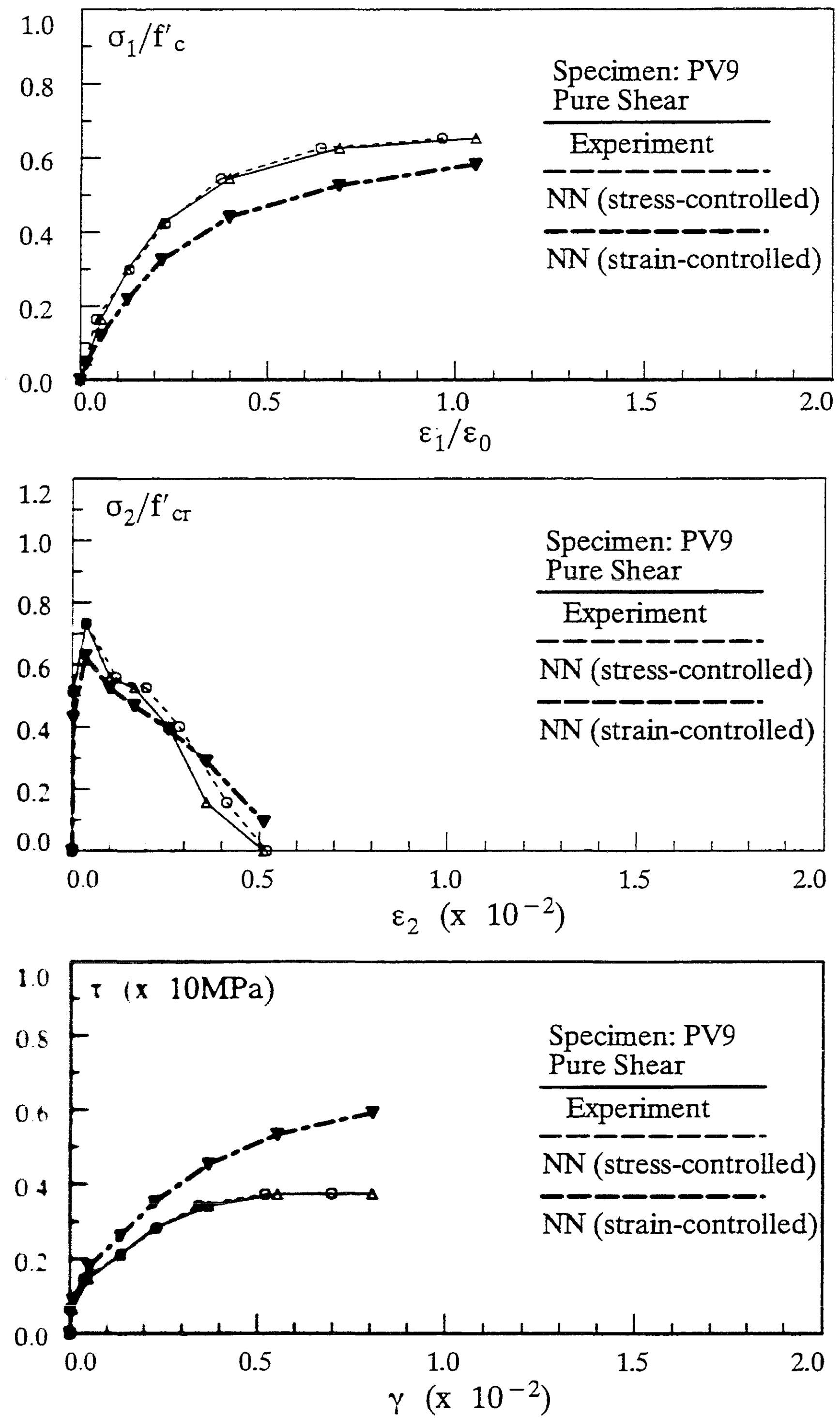


Fig. 4.38 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

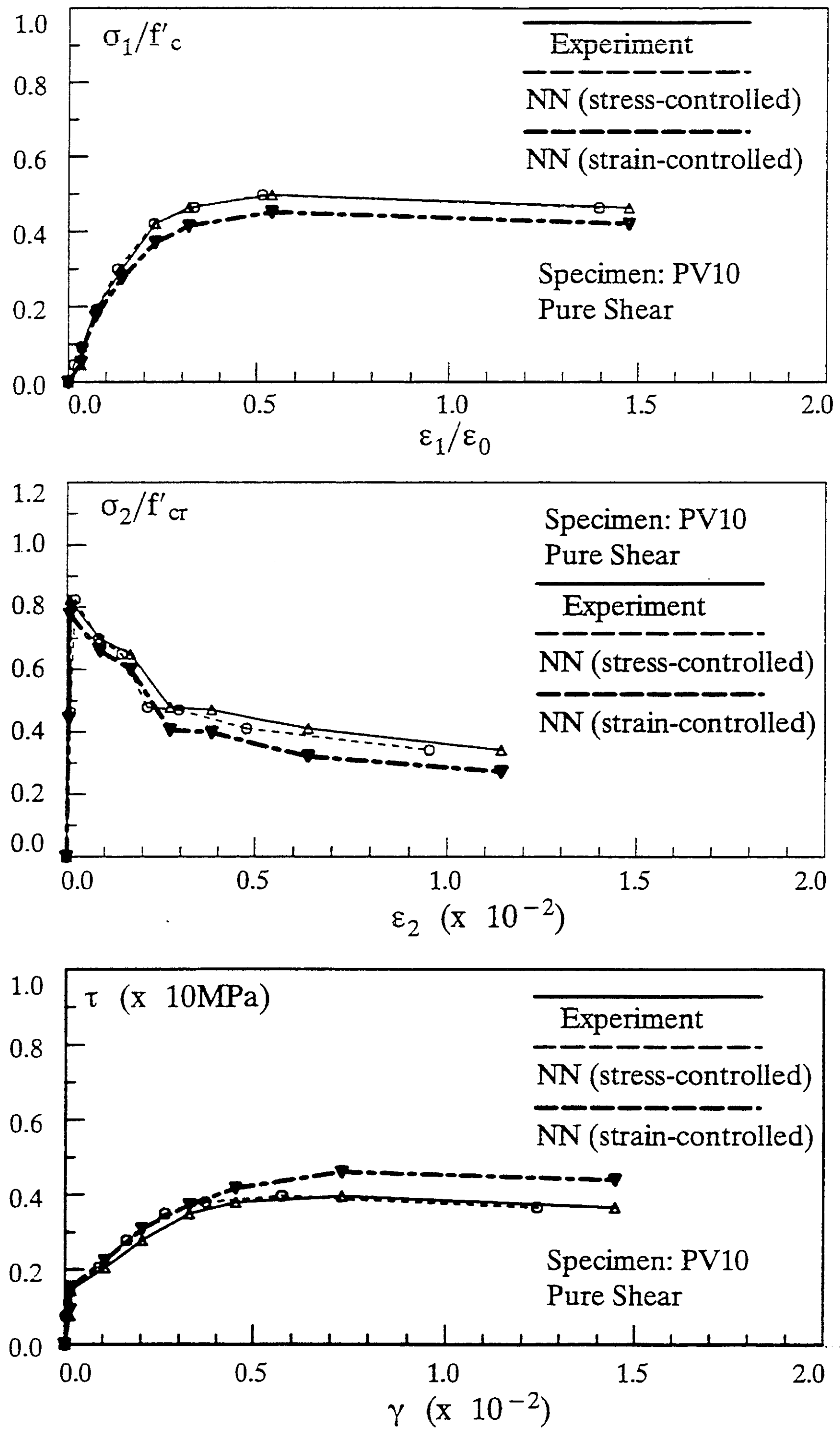


Fig. 4.39 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

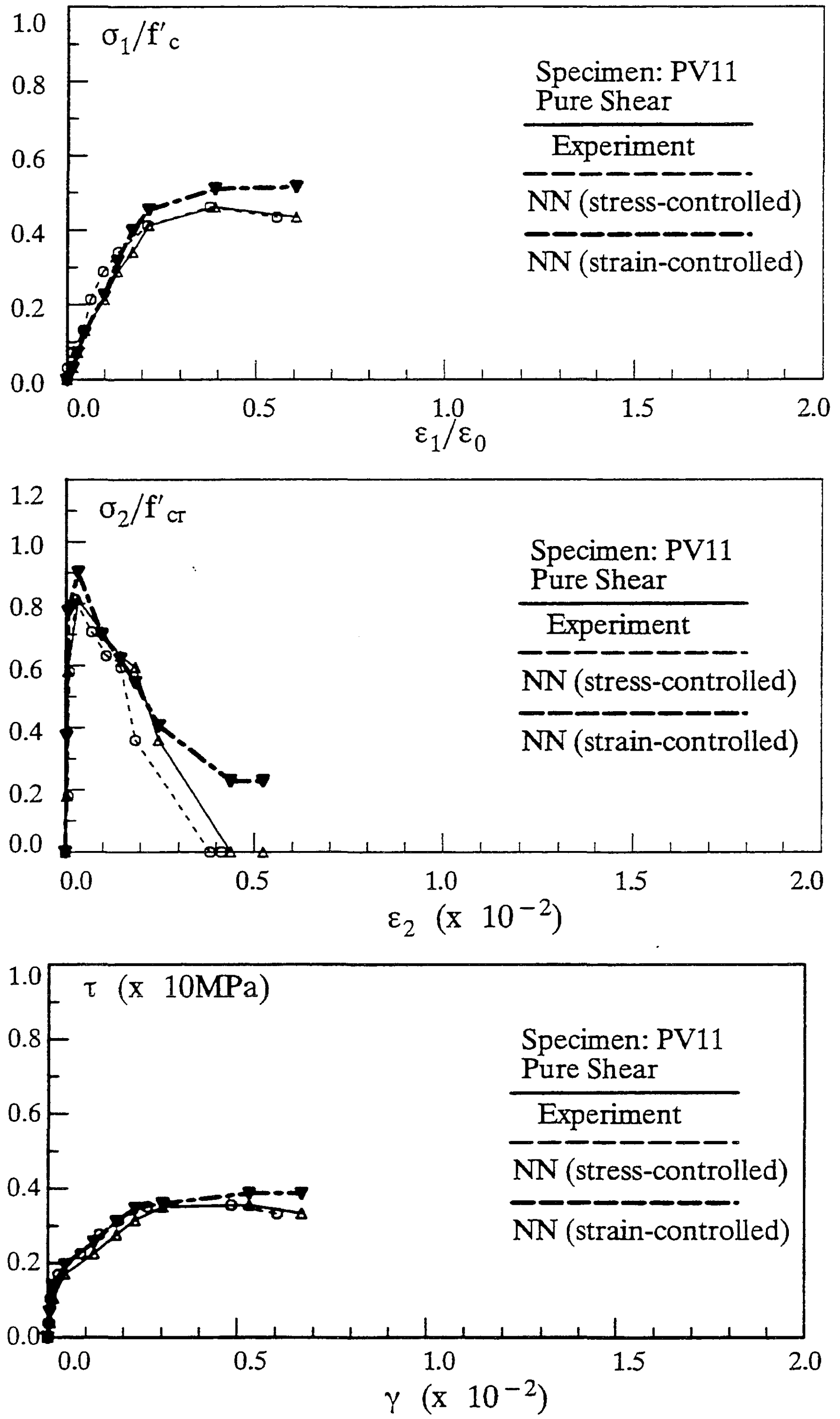


Fig. 4.40 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

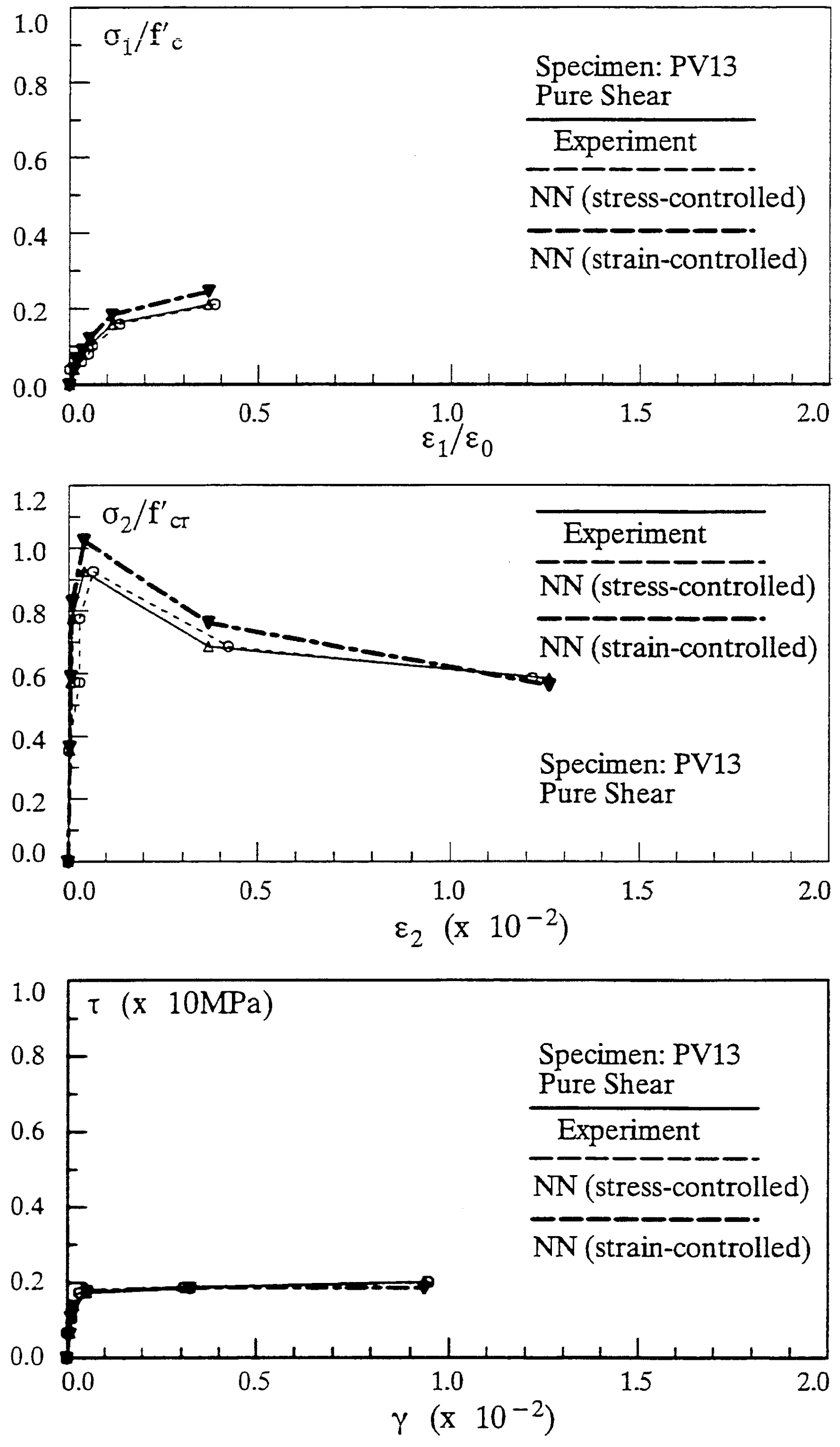
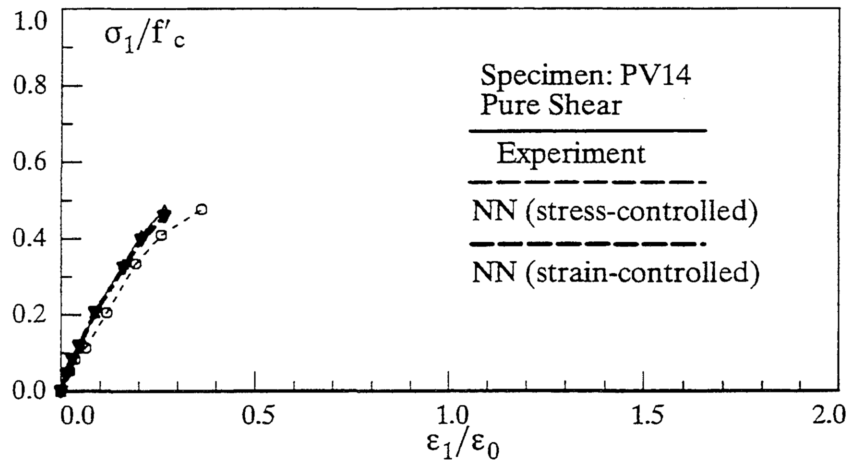
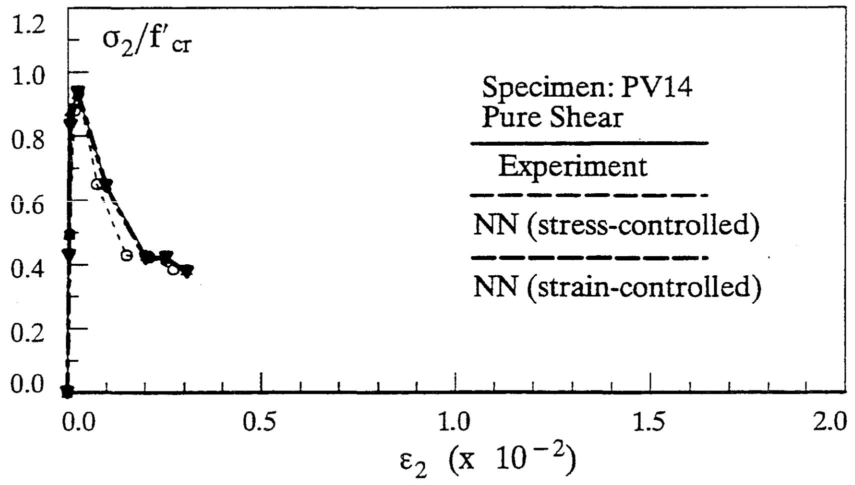


Fig. 4.41 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)





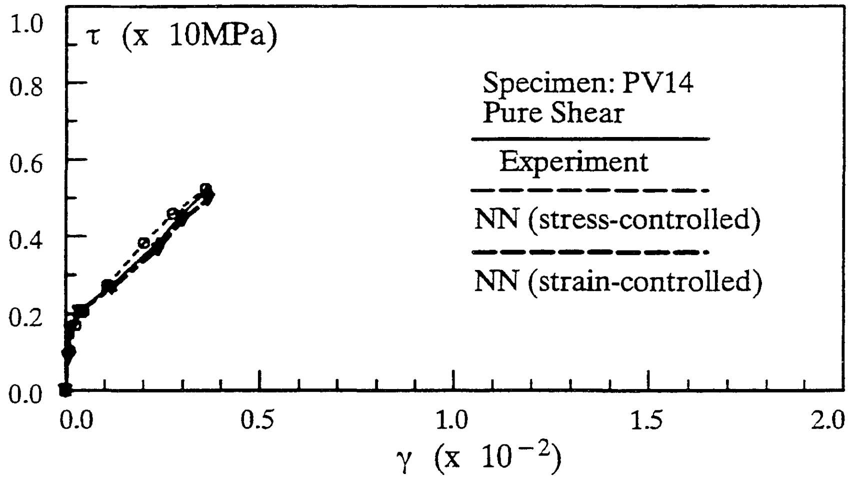


Fig. 4.42 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

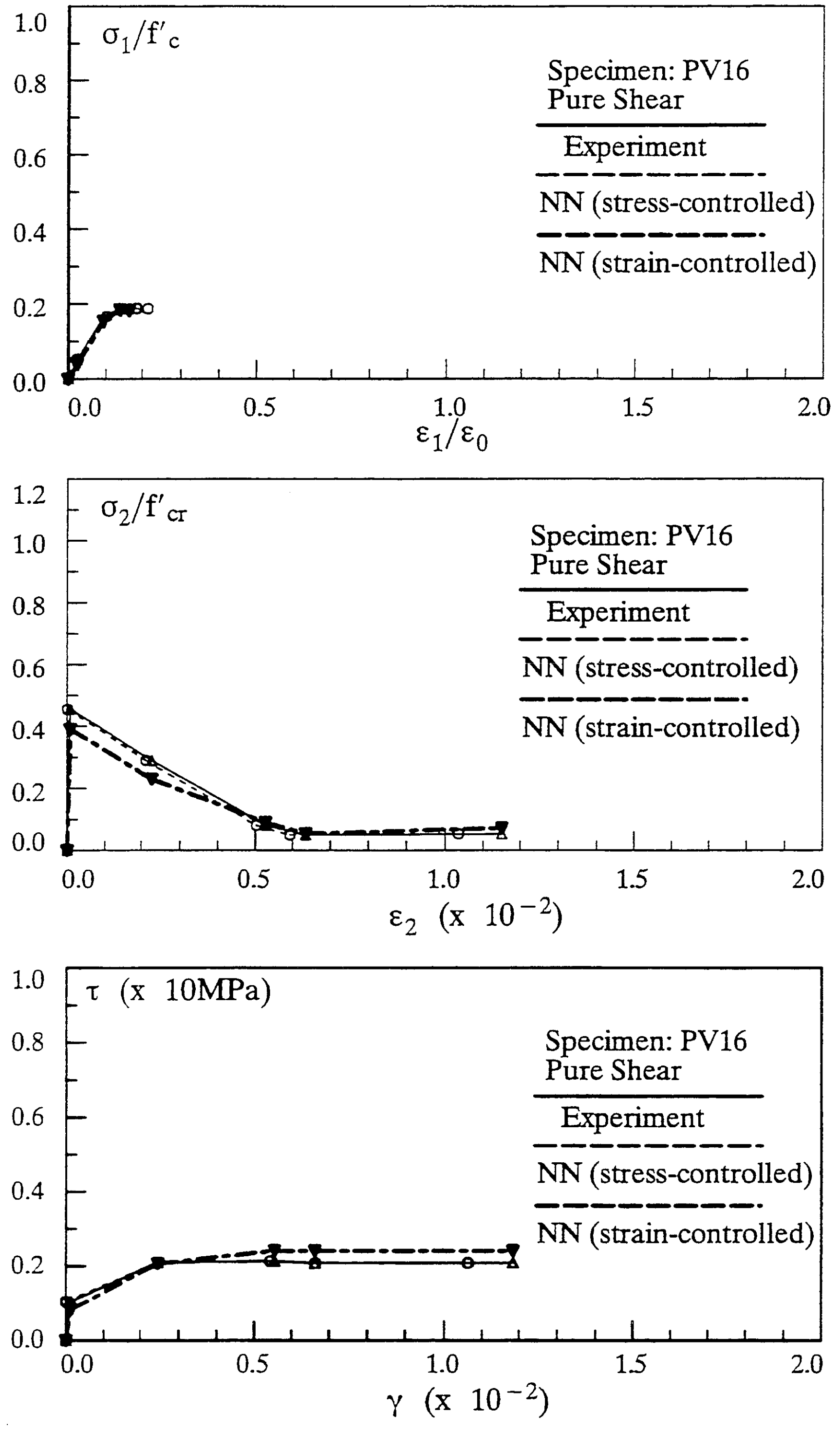
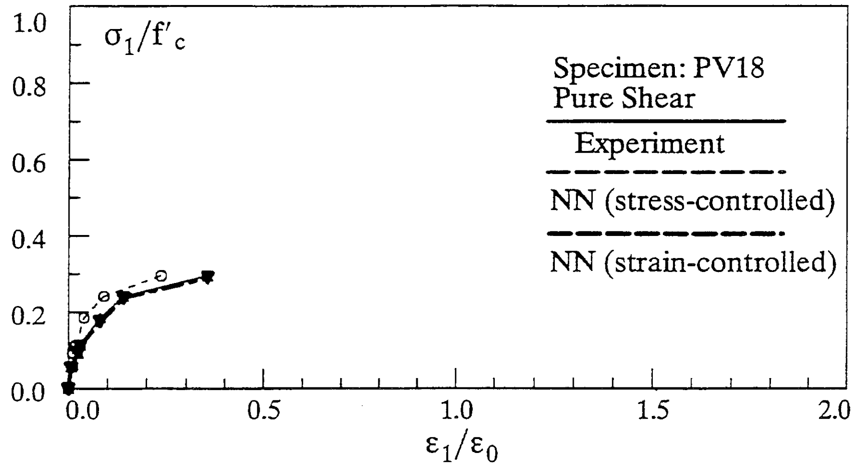
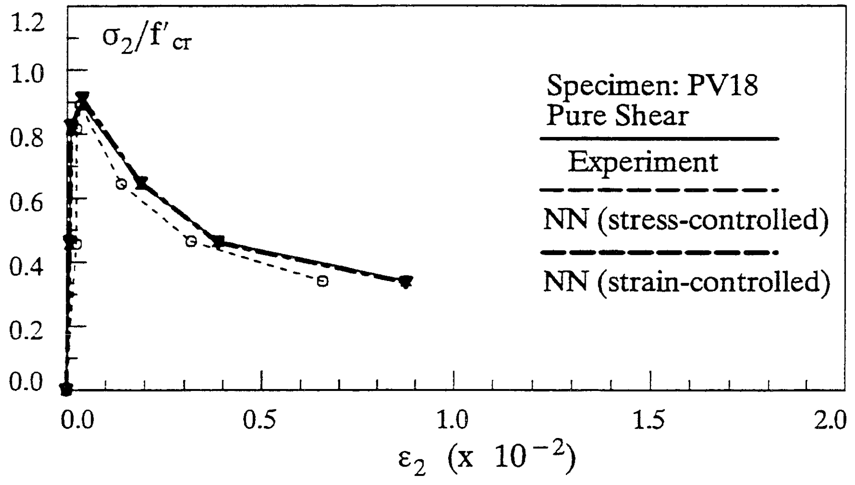


Fig. 4.43 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)





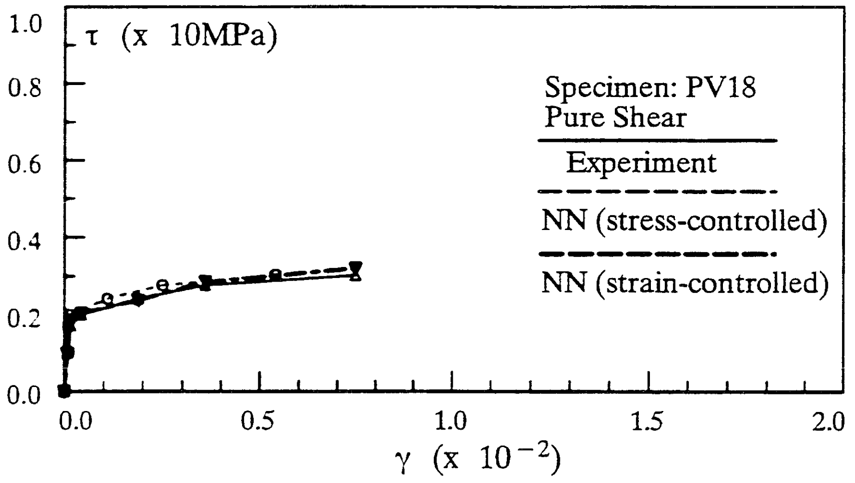


Fig. 4.44 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

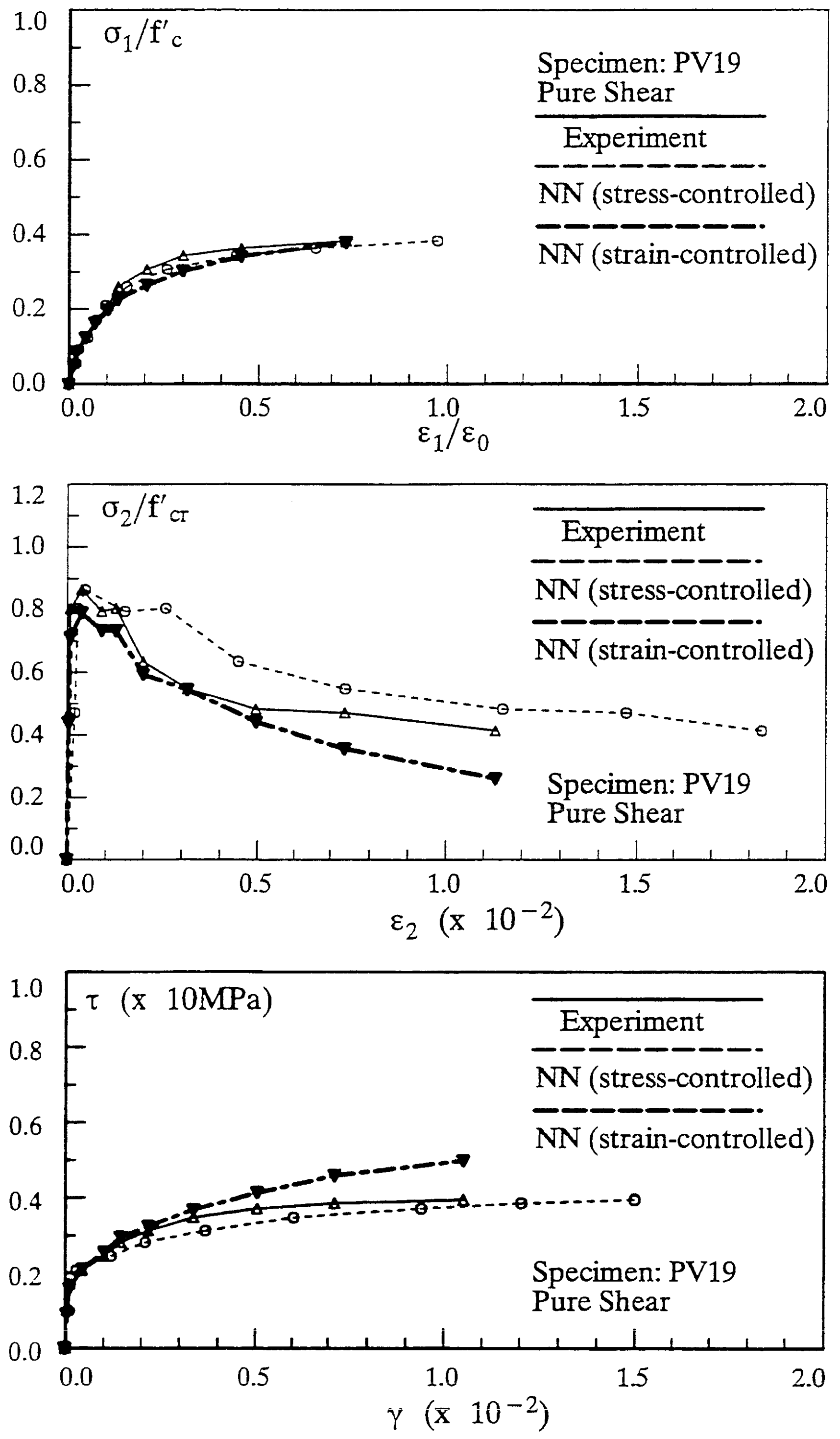
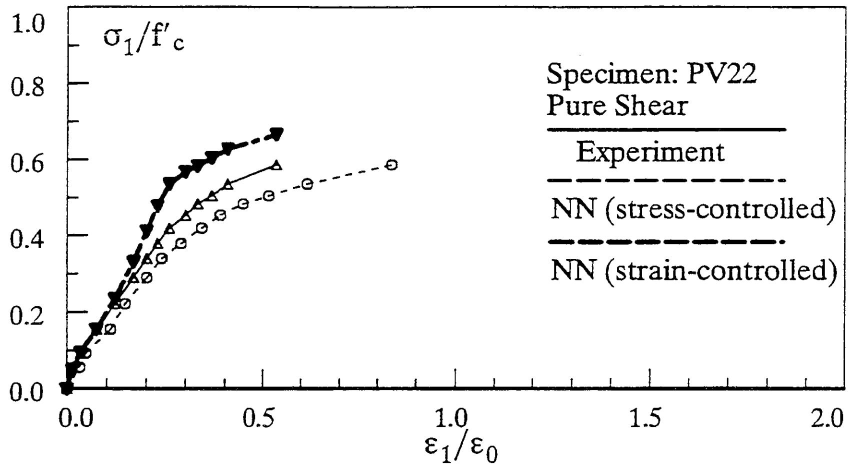
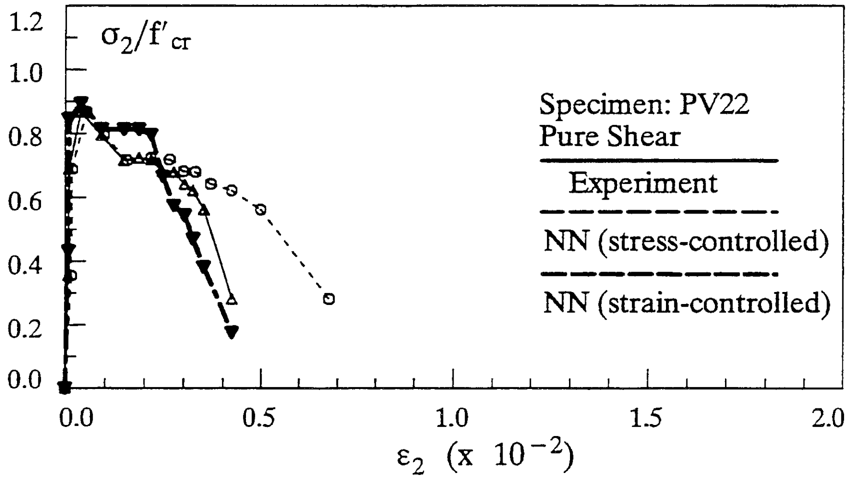


Fig. 4.45 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)





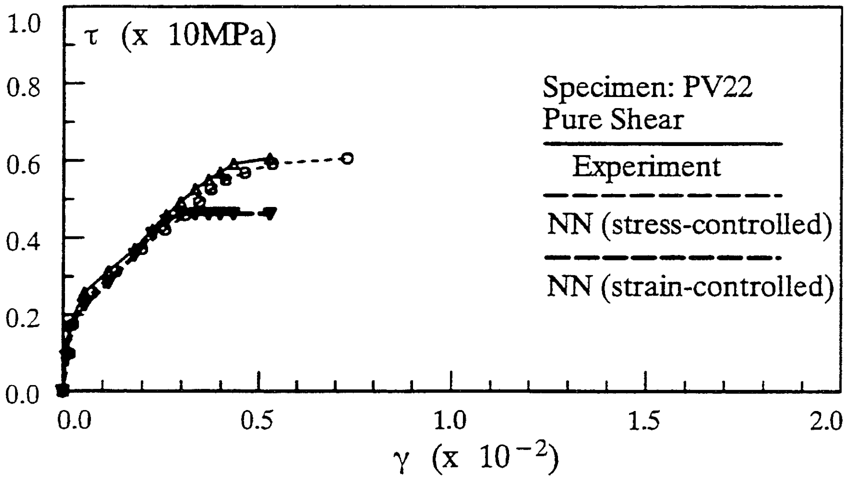


Fig. 4.46 – The Stress-Strain Relations Predicted by the Network – Training Results (after Training Data on 14 Panels)

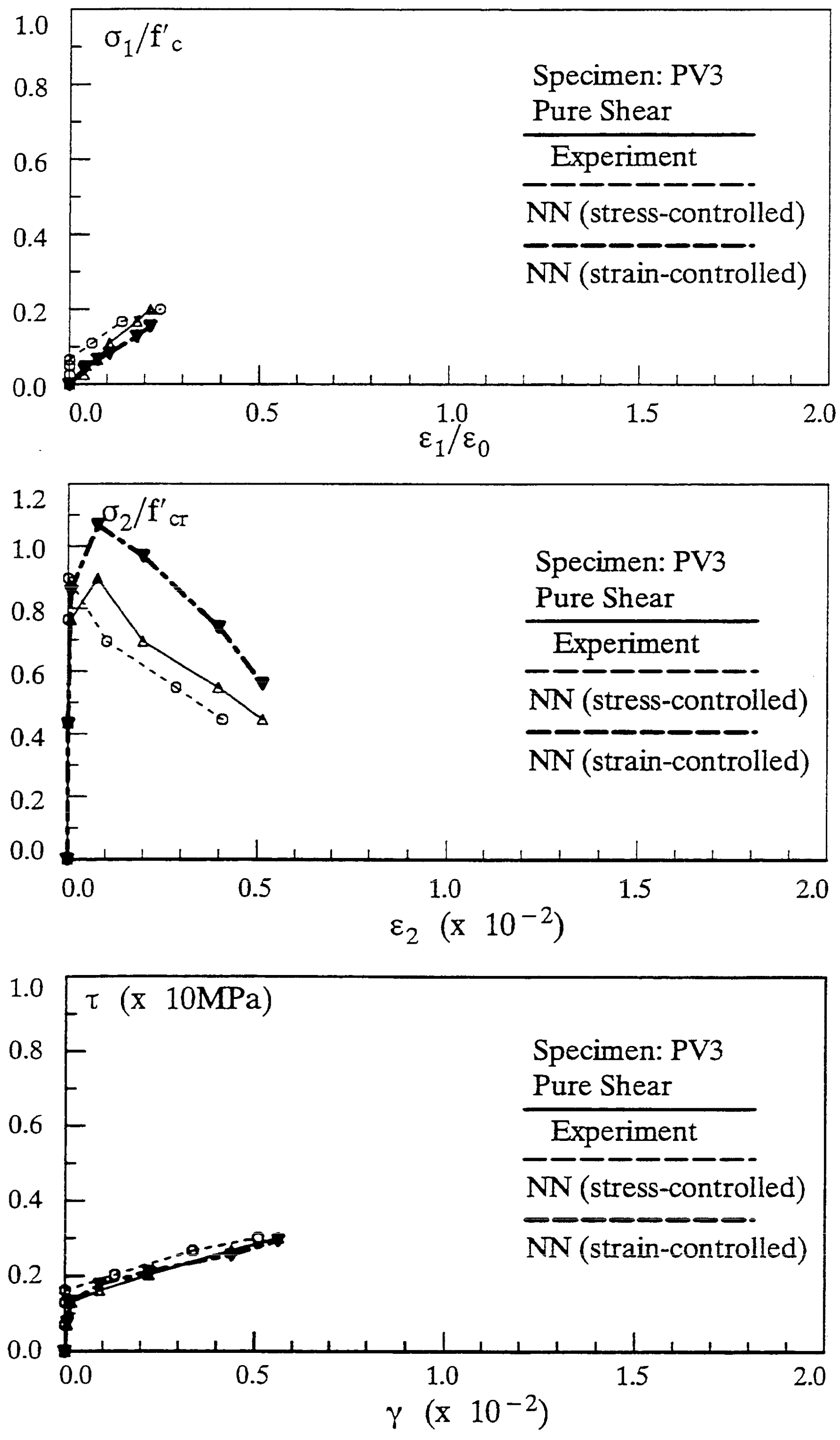


Fig. 4.47 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

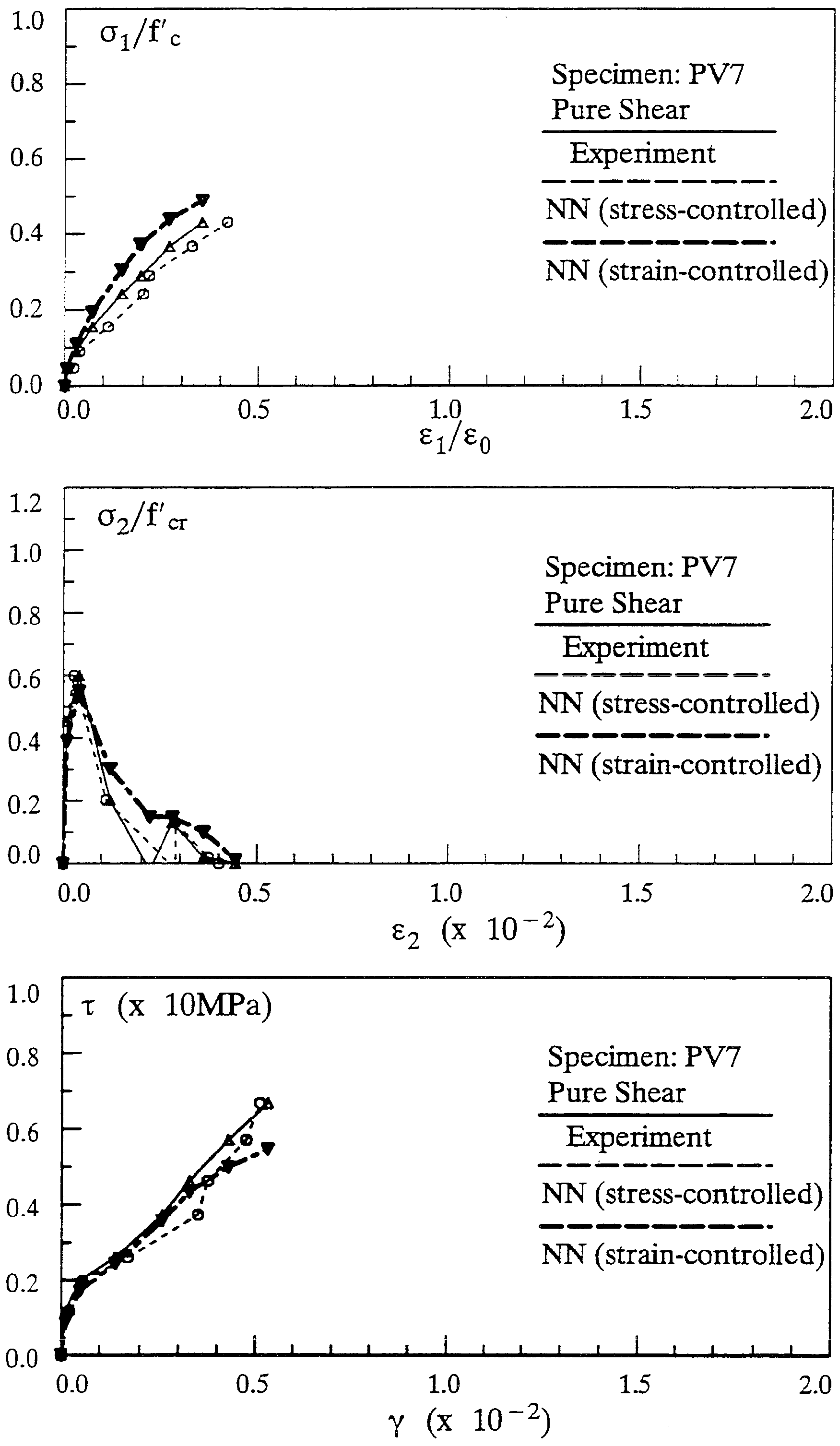


Fig. 4.48 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

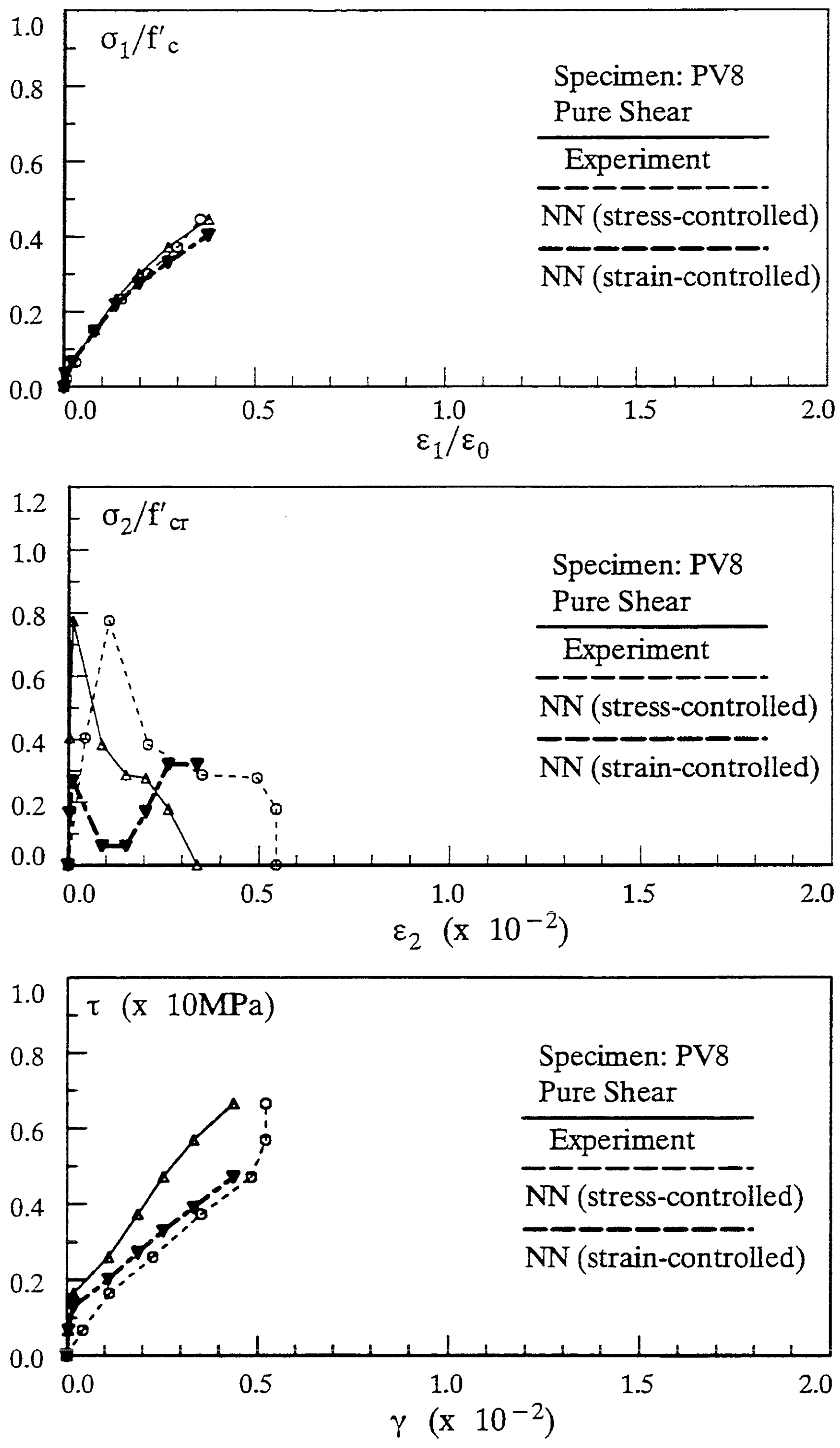


Fig. 4.49 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

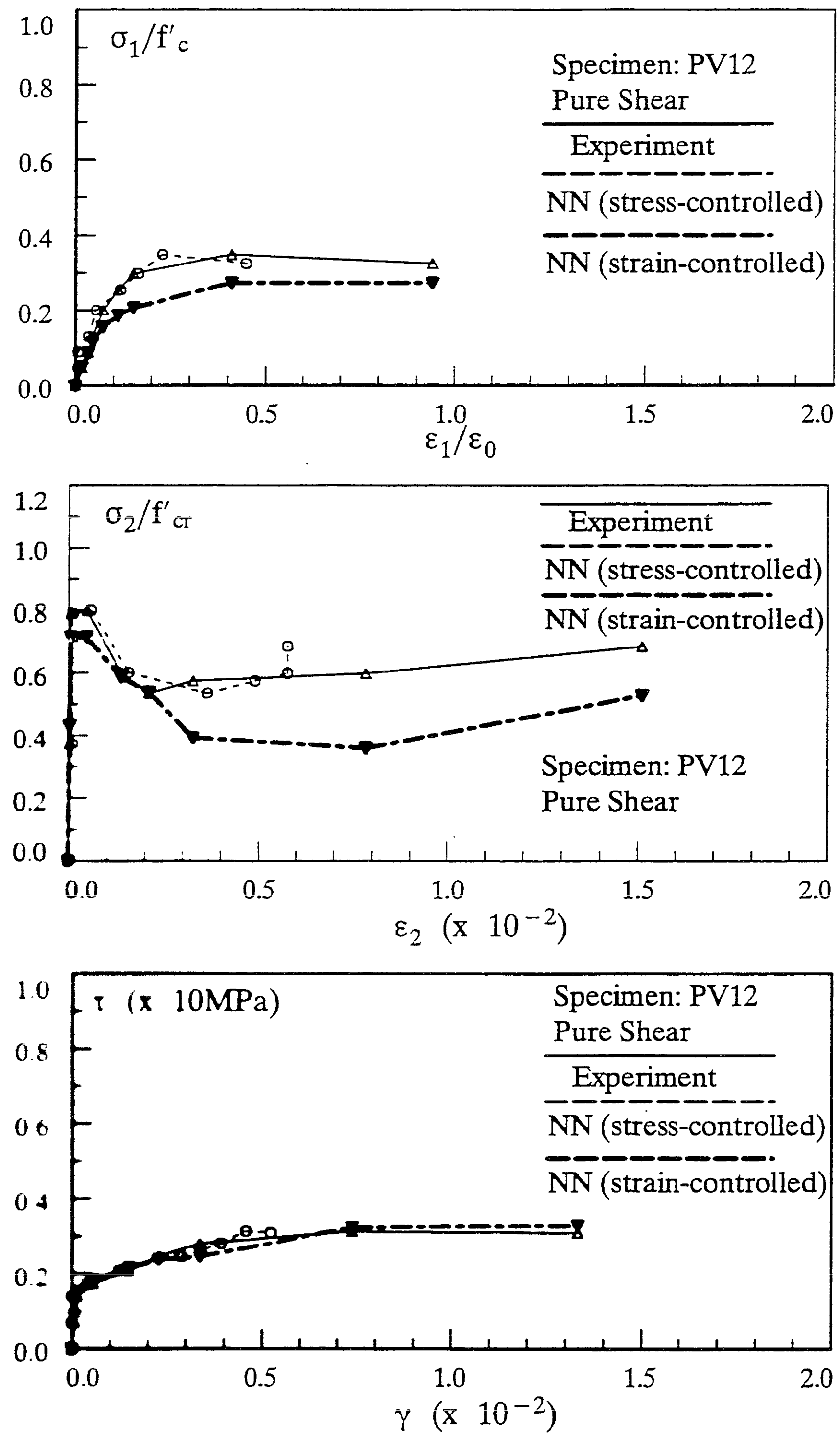
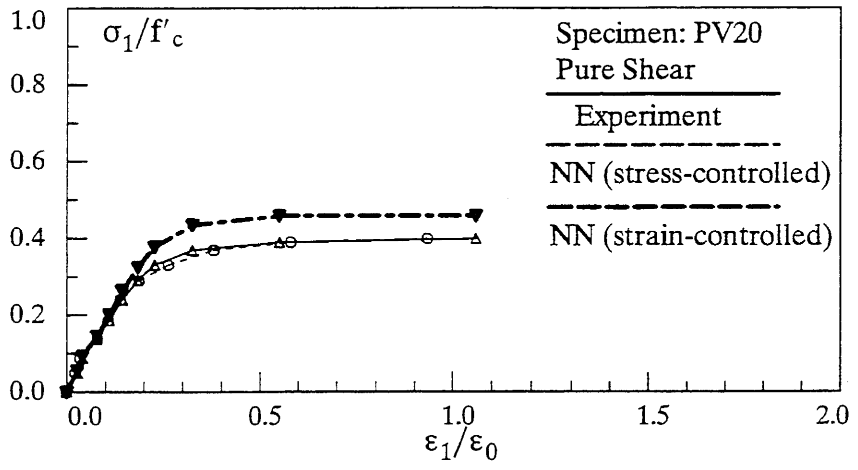
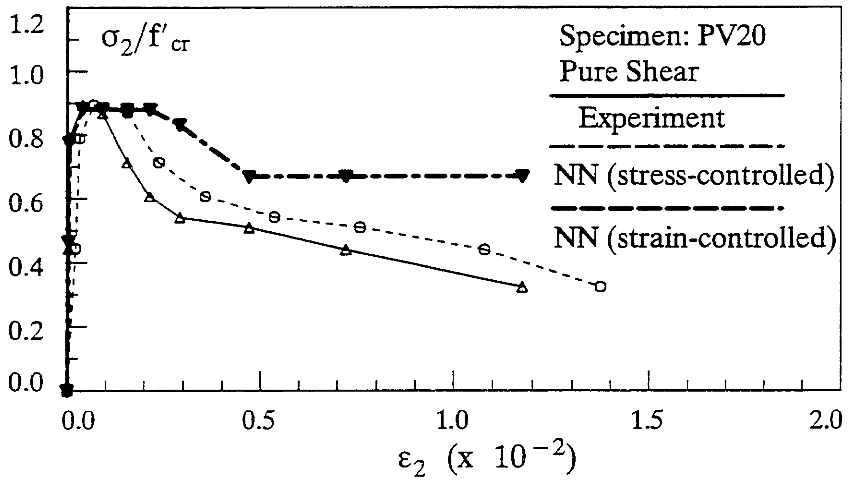


Fig. 4.50 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)





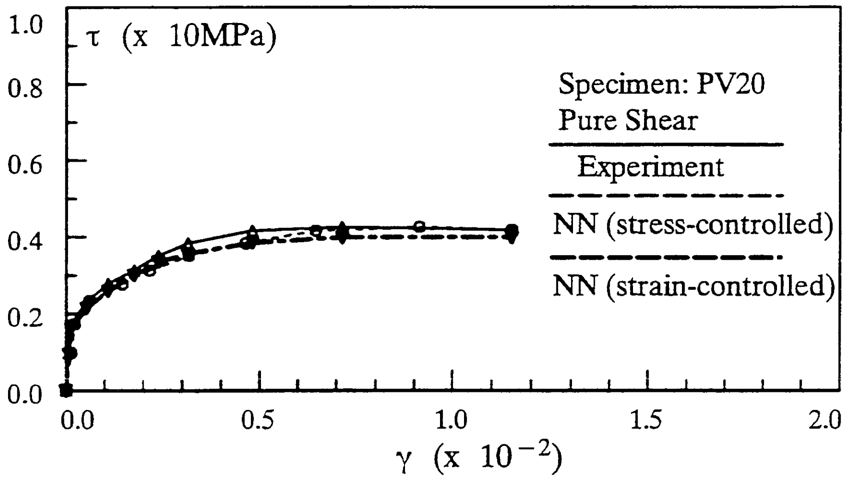


Fig. 4.51 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

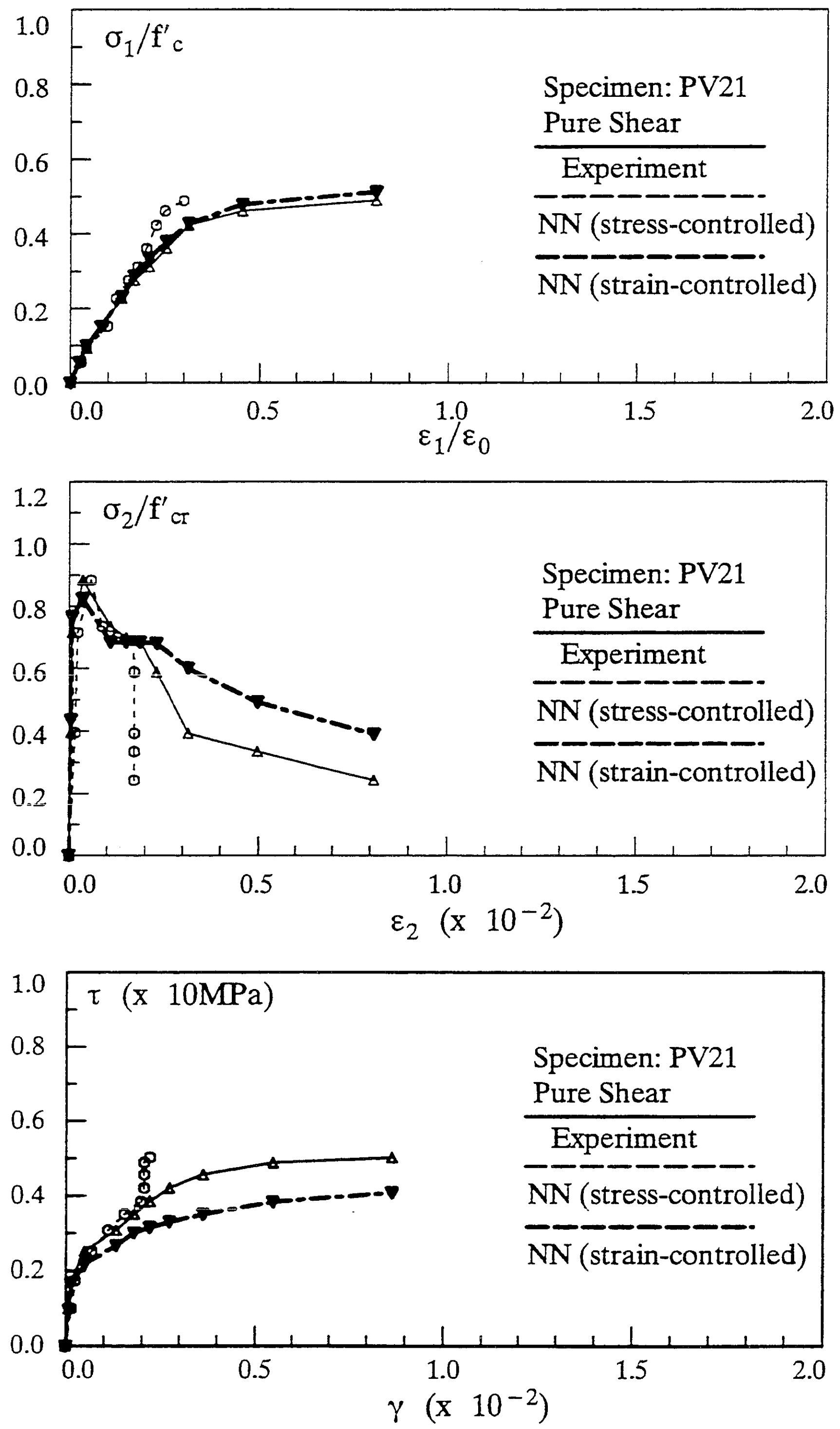


Fig. 4.52 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

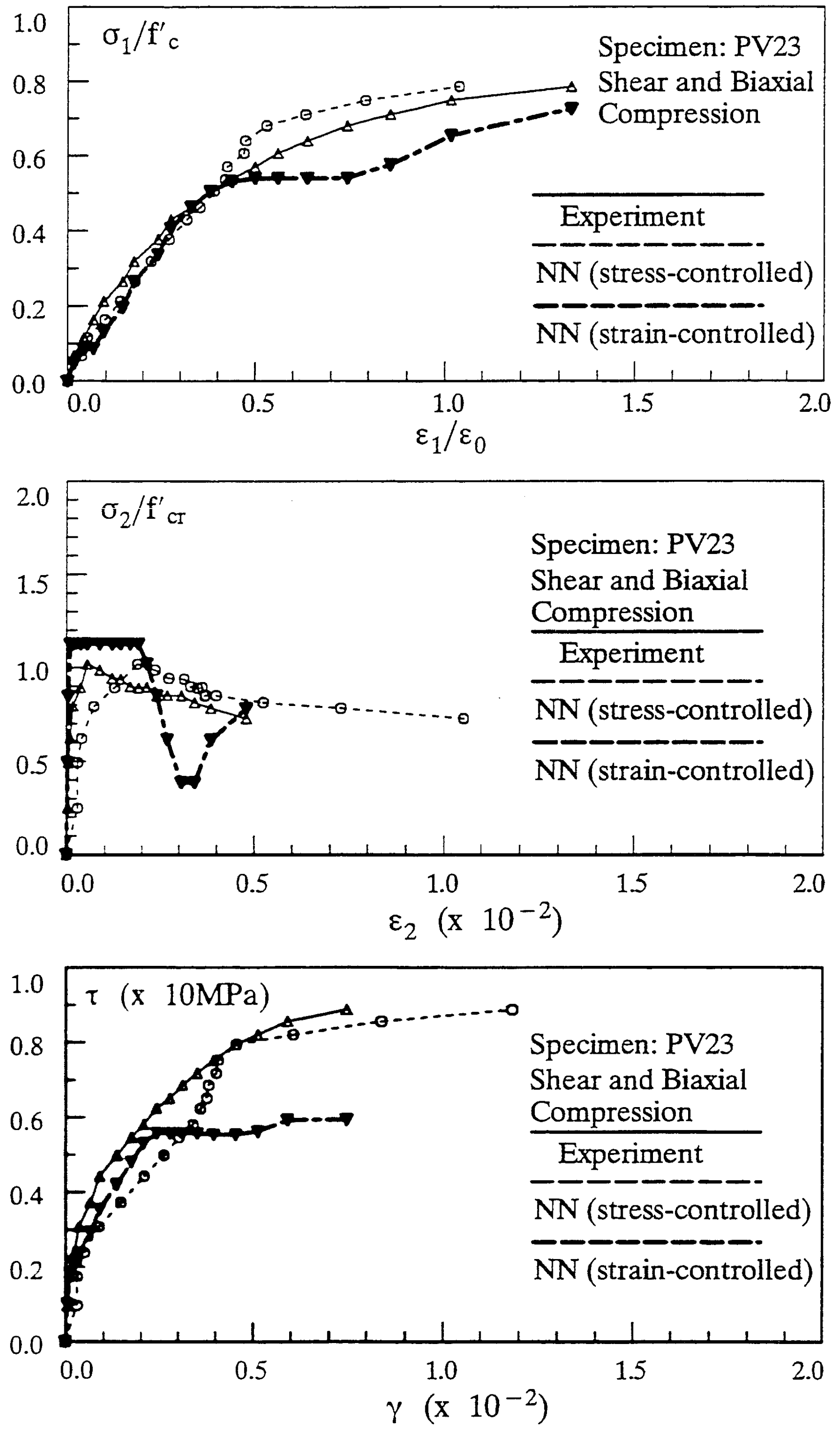
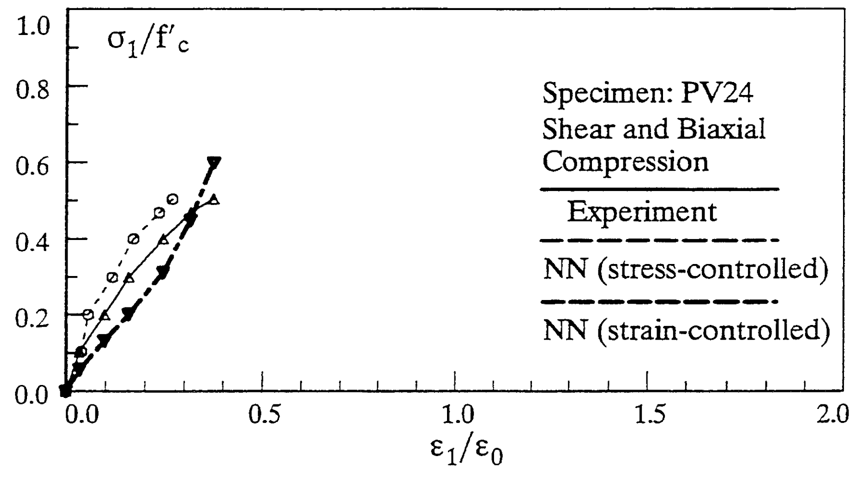
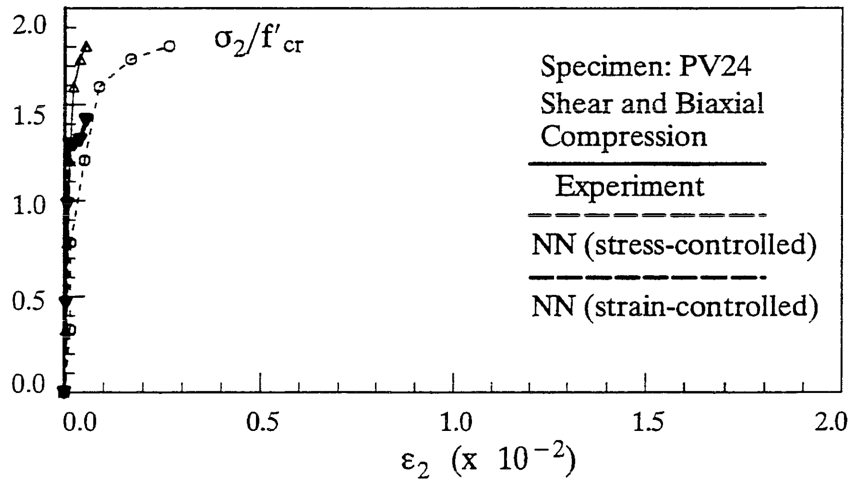


Fig. 4.53 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)





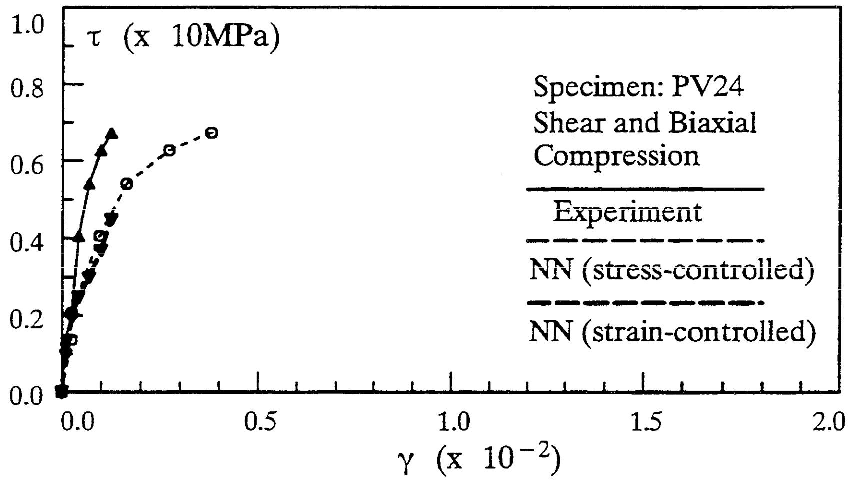


Fig. 4.54 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

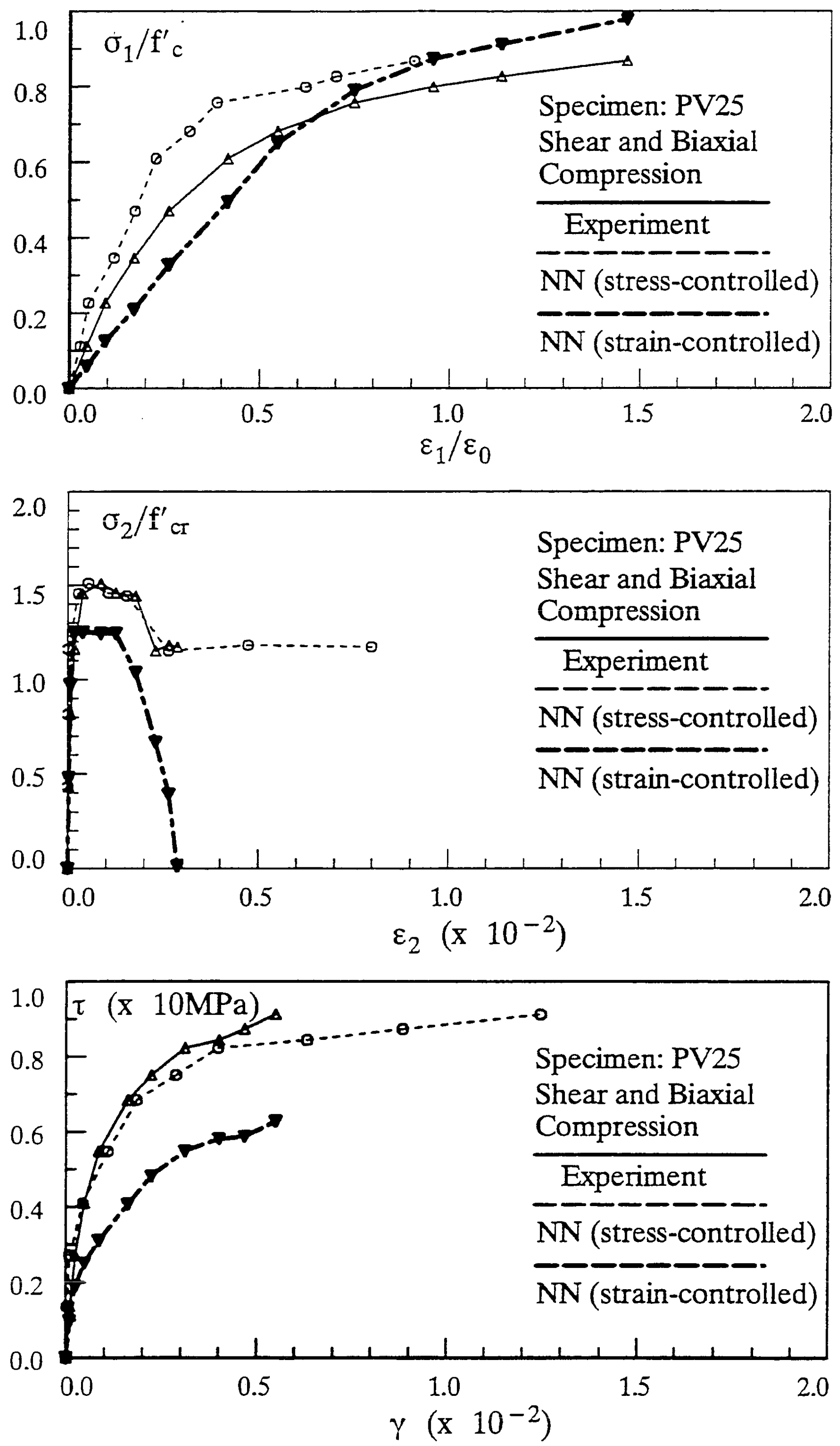
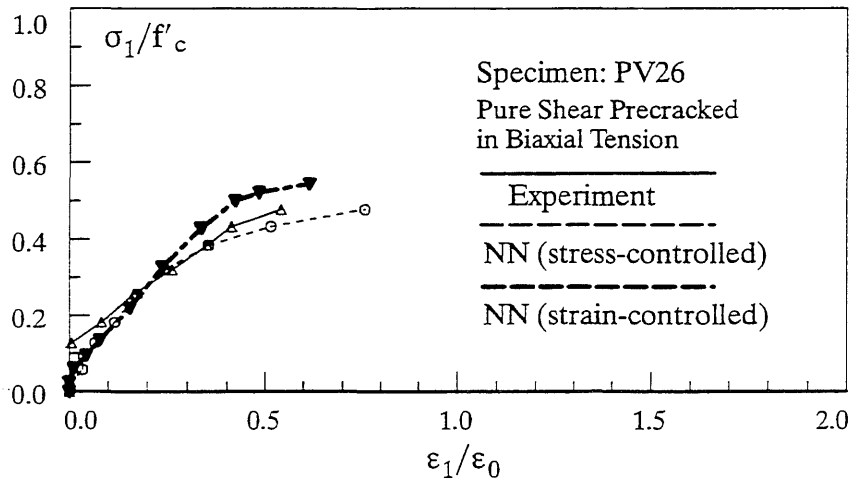
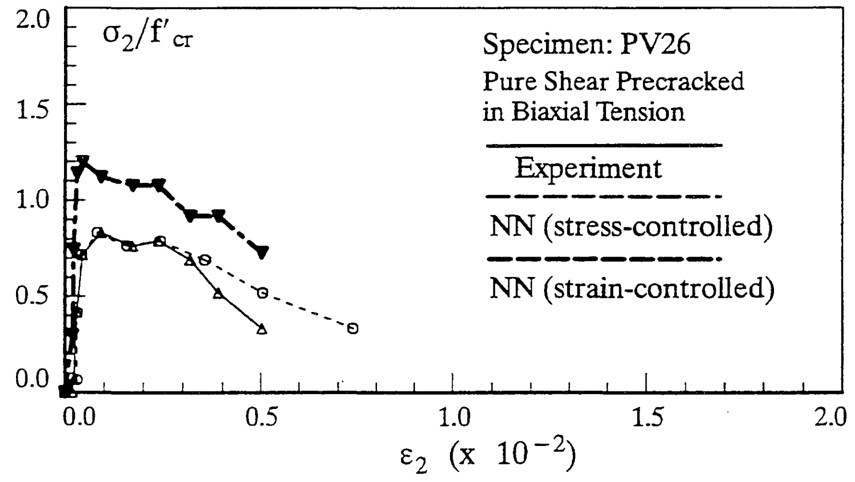


Fig. 4.55 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)





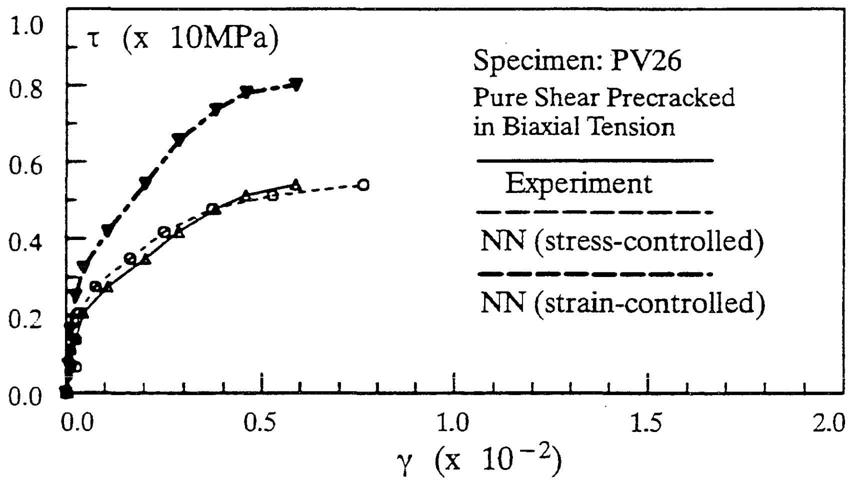


Fig. 4.56 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

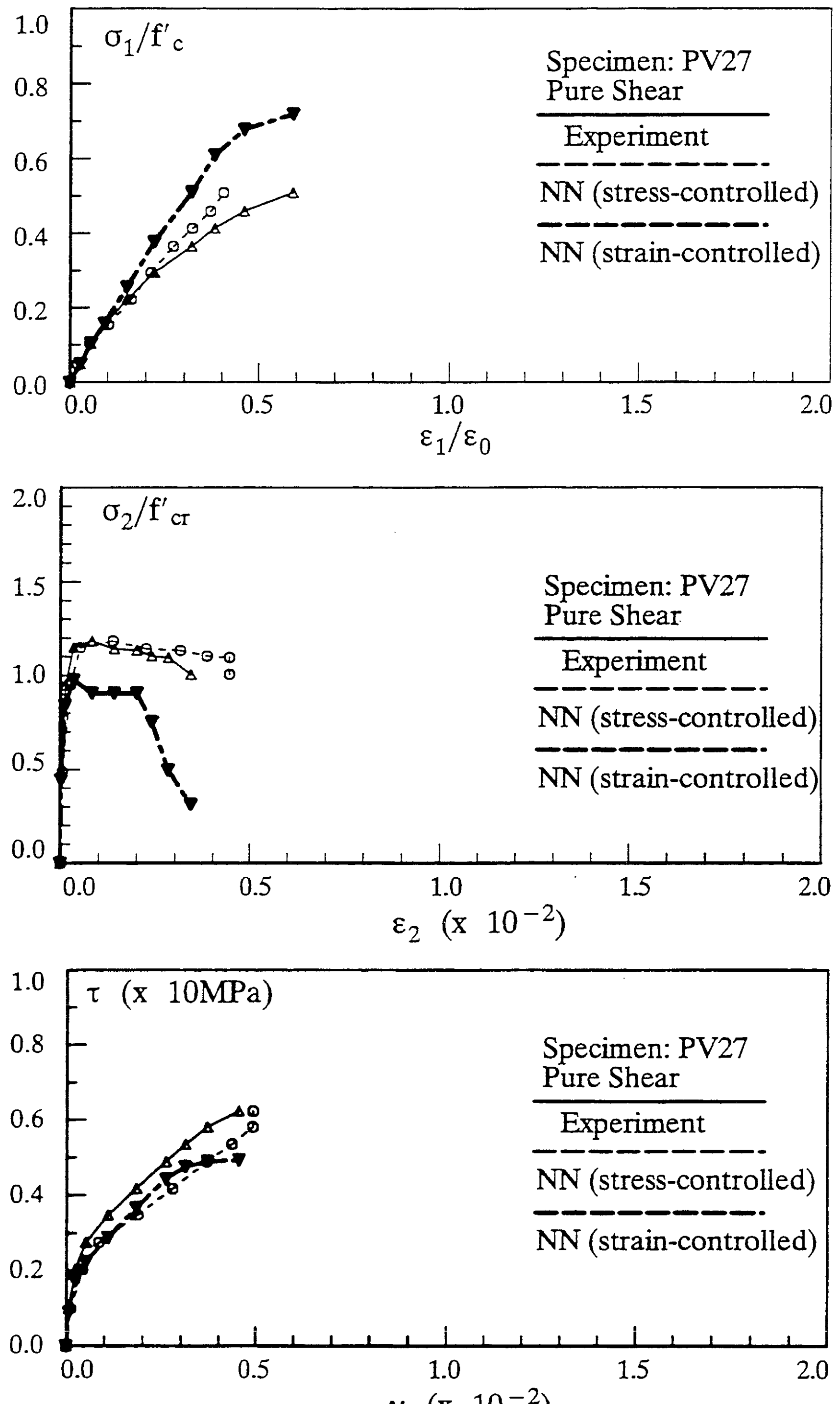
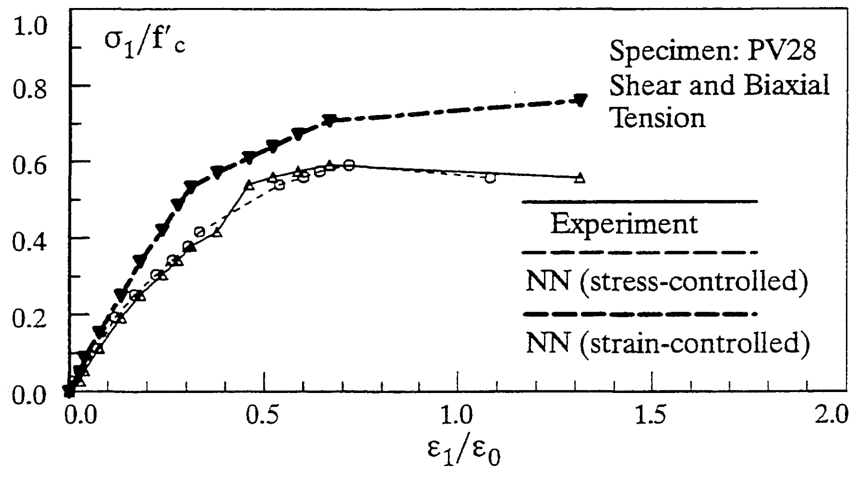
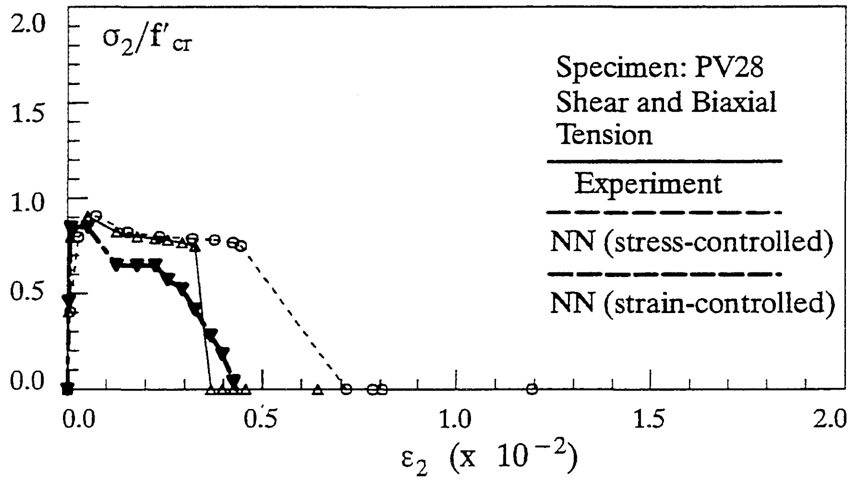


Fig. 4.57 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)





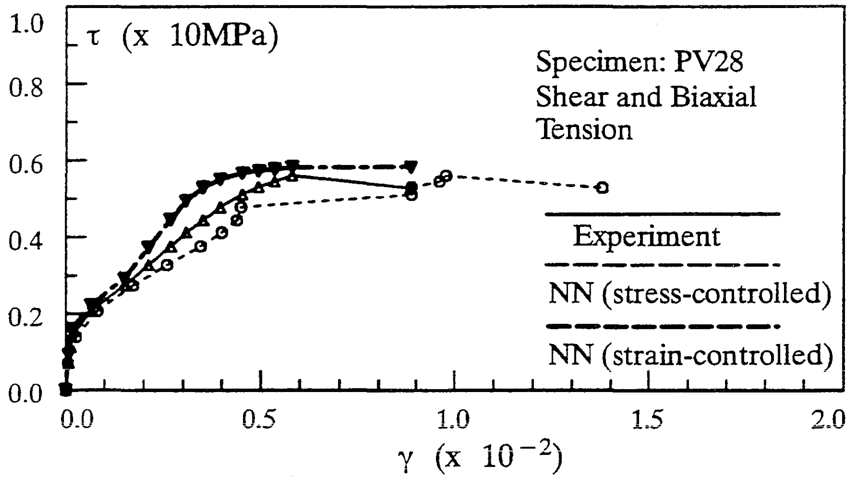


Fig. 4.58 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

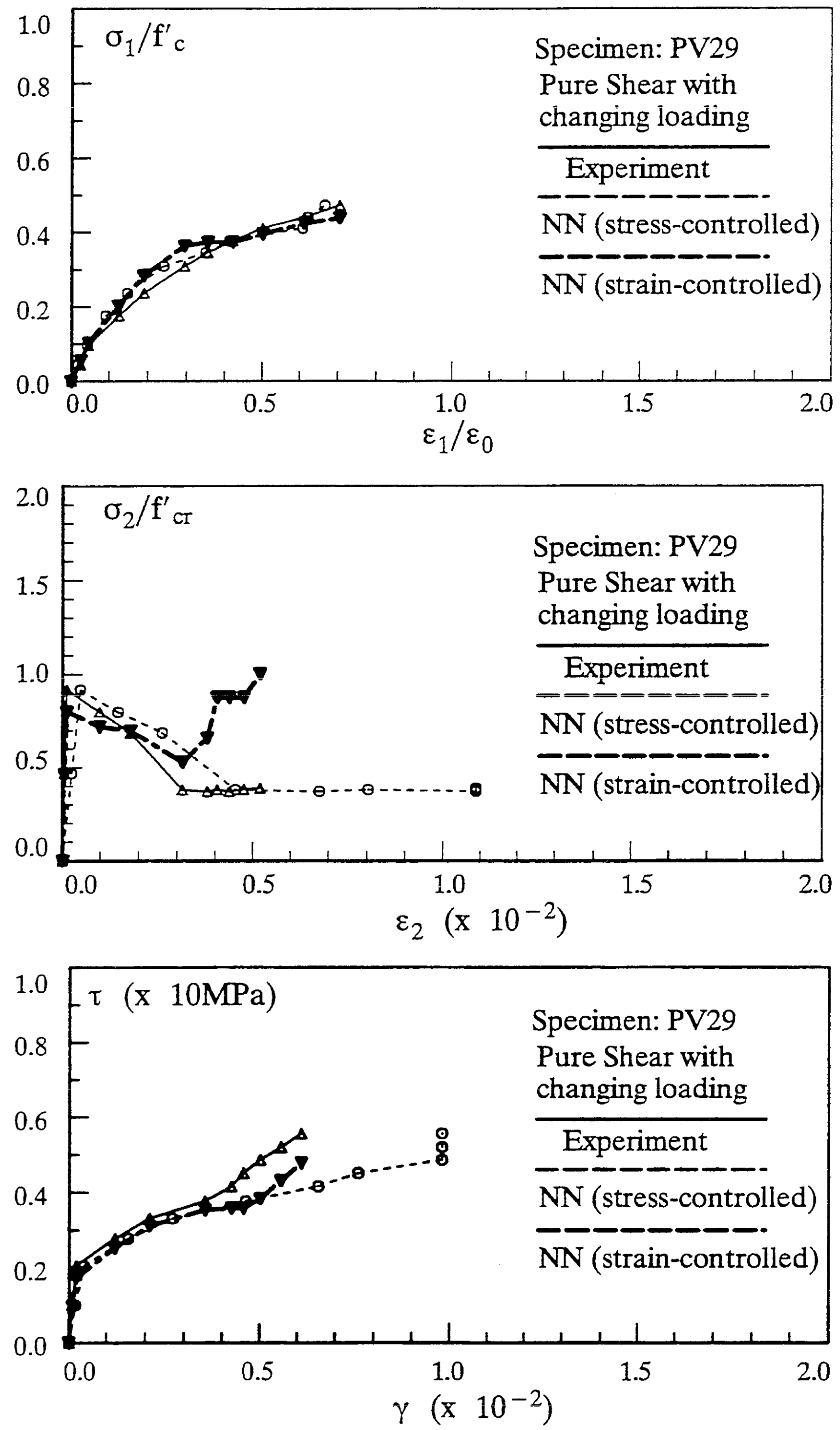


Fig. 4.59 – The Stress-Strain Relations Predicted by the Network – Testing Results (after Training Data on 14 Panels)

CHAPTER 6

USE OF NEURAL NETWORK-BASED MATERIAL MODELS IN FINITE ELEMENT ANALYSIS

6.1 General

The objective of constructing rational constitutive models of engineering materials is to use the material model in a numerical solution procedure such as the finite element method so that the corresponding structural behavior can be reliably predicted. In general, the material model serves two purposes in the finite element modeling of structures. The first deals with the evaluation of the current material constitutive matrices, and hence the formulation of element stiffness matrices; and the second is the determination of the current state of stress corresponding to a given increment of strain from an initial stress–strain state.

In finite element procedures, the tangential element stiffness matrix $[k_t]$ is usually expressed as (Zienkiewicz, 1977)

$$[k_t] = \int_V [B]^T[D_t][B] \; dV \tag{6.1}$$

where [B] is the strain–displacement transformation matrix, V the volume of the element, and $[D_t]$ the constitutive or stress-strain matrix of the material. Since most of the finite element procedures for nonlinear analysis of reinforced concrete structures utilize an incremental loading procedure to solve the nonlinear equation, naturally, the constitutive relation is expressed in an incremental form. In the incremental form, the stress–strain matrix is determined from

$$\{d\sigma\} = [D_t]\{d\varepsilon\} \tag{6.2}$$

where $\{d\sigma\}$ and $\{d\varepsilon\}$ are vectors of incremental stresses and strains.

For an analytical material model represented in the form of Eq. (6.2), its incorporation within a finite element procedure can be readily accomplished because the elements of the constitutive matrix $[D_t]$ are expressed as mathematical expressions and can be directly determined. On the other hand, for a neural network–based material model represented as the distributed weight structure embedded in a trained multilayer feedforward neural network, the constitutive matrix $[D_t]$ is somewhat different because of the representation scheme used. Moreover, the elements in the neural network constitutive matrix $[D_t]$ is not explicitly determined and no mathematical formulae are observable, so that its use within a finite element procedure needs some special considerations. Nevertheless, even with this difference in the way the material behavior is represented, as a computational entity, the neural network-based material model functions exactly the same as an analytical model does, and there is no technical barrier for its incorporation within some solution procedures. In the following paragraphs, considerations associated with the implementation of neural network–based material models are generally discussed.

6.2 Considerations on Implementing Neural Network-Based Material Models

6.2.1 Direct Use of Neural Network Models

The implementation of constitutive models is greatly influenced by the solution procedure utilized for nonlinear finite element analysis of structures. For reinforced concrete structures, an incremental loading procedure together with equilibrium iteration is usually adopted in practice. As such, the governing equilibrium equations or equations of motion of a structure in the incremental form for use in a modified Newton iteration are derived as follows: (Bathe, 1982)

1) for static analysis:

$$\mathbf{K}_t \Delta \mathbf{U}_k = \mathbf{P}_k - \sum \int_V \mathbf{B}^T \sigma_{k-1} \, dV \tag{6.3}$$

2) for dynamic analysis with implicit time integration scheme:

$$\mathbf{M}\ddot{\mathbf{U}}_k + \mathbf{K}_t \Delta \mathbf{U}_k = \mathbf{P}_k - \sum \int_V \mathbf{B}^T \sigma_{k-1} \, dV \tag{6.4}$$

3) for dynamic analysis with explicit time integration scheme:

$$\mathbf{M}\ddot{\mathbf{U}}_k = \mathbf{P}_k - \sum \int_V \mathbf{B}^T \sigma_k \, dV \tag{6.5}$$

where $\mathbf{M}$ is the mass matrix of the structure, $\mathbf{K}_t$ the tangential incremental stiffness matrix, $\mathbf{P}_k$ the vector of externally applied nodal loads, $\Delta\mathbf{U}_k$ the vector of nodal displacement increments, and $\ddot{\mathbf{U}}_k$ denotes the vector of nodal accelerations.

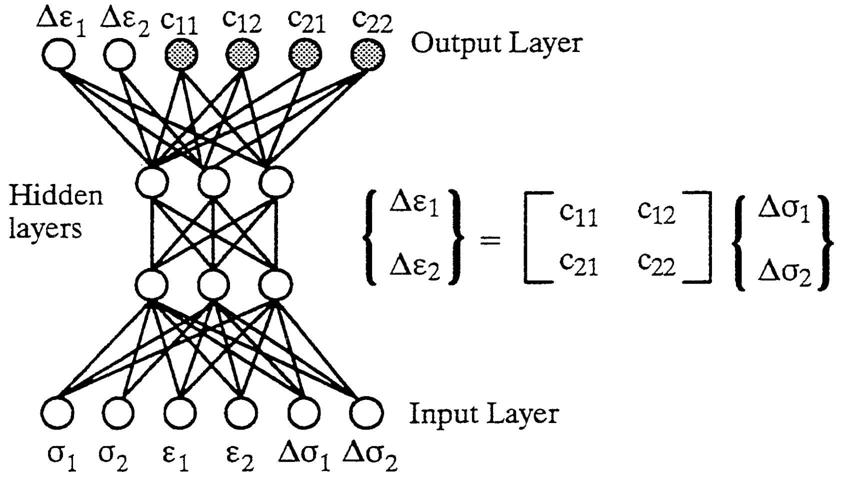
The constitutive model in a finite element program is generally utilized in three computation stages: the formation of element stiffness matrix, the computation of internal resistant forces in equilibrium iteration, and the calculation of element stresses (ASCE, 1982). Apparently, if a Newton–Raphson scheme is used to solve the static and dynamic problems represented in Eqs. (6.3) – (6.5), an explicit form of the stress–stain matrix $[D_t]$ is needed for the first two kinds of problems except for the third one. Thus, without explicit formation of the material stiffness matrix, the neural network material model can only be directly used for dynamic analysis of structures with explicit time integration schemes such as the central difference method.

However, for static analysis, it has been well known that the explicit formulation of element stiffness matrices is unnecessary if some iterative schemes such as the conjugate gradient method (Hestenes and Stiefel, 1952) or precondition conjugate gradient methods with element-by-element implementation (Hughes, et al. 1986) are used as solution methods. In these cases, the explicit form of the stress–strain material matrix $[D_t]$ is not required, rather that a mechanism to evaluate the stress increments given a strain increment and the current stress–strain state, which corresponds to a feedforward presentation to a strain-controlled neural network-based material model, is needed. Clearly, a strain-controlled neural network–based material model can be directly used in finite element procedures for both static analysis with conjugate gradient solution schemes and dynamic analysis with explicit integration methods, whereas it cannot be directly employed for solving nonlinear dynamic problems with implicit integration schemes.

6.2.2 Indirect Use of Neural Network Models

From the forgoing discussion, it has realized that the current form of neural network material model should be implemented with some special solution schemes for nonlinear finite element analysis, if the model is directly used. Nevertheless, with the flexibility in the design of representation schemes and the preparation of training data for a neural network material model, the evaluation of the stress-strain matrix $[D_t]$, or $[C_t]$ in a stress-controlled model in which $\{d\varepsilon\} = [C_t]\{d\sigma\}$, is realizable either through proper training on an augmented training data set with additional information but still using the representation schemes developed in this study or through the introduction of a new representation scheme with a correspondingly generated new training data set.

The first training approach is based on the observation that the information

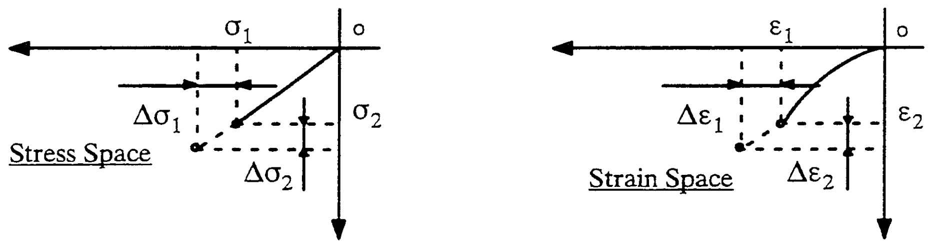

$\Delta\varepsilon_1$ $\Delta\varepsilon_2$ c_{11} c_{12} c_{21} c_{22}
Output Layer
Hidden layers
Input Layer
σ_1 σ_2 ε_1 ε_2 $\Delta\sigma_1$ $\Delta\sigma_2$

$$\begin{Bmatrix} \Delta\varepsilon_1 \\ \Delta\varepsilon_2 \end{Bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{Bmatrix} \Delta\sigma_1 \\ \Delta\sigma_2 \end{Bmatrix}$$

Fig. 6.1 – The Representation Approach for Determining the Stress–Strain Matrix in a Stress-controlled Neural Network Material Model

captured in a neural network material model is determined solely by the information embedded in the training data. If the information of the stress-strain matrix $[D_t]$ or $[C_t]$ is properly represented in the training data, a material model trained on this data set should contain the relevant information. As a consequence, elements of the material matrix can then be computed from a trained network. On the other hand, with the second representation approach, elements of the stress–strain matrix needs to be explicitly included into the current representation scheme for the output. Of course, experimental data on the elements of the constitutive matrix are not readily available, and have to be estimated from the test results. If a reliable set of data containing explicit information of the constitutive matrix is obtained, this additional information can be readily incorporated in the neural network-based material model through training on the new data prepared with the updated representation scheme. This representation approach is illustrated in Fig. 6.1.

To illustrate the first training approach, the evaluation of $[C_t]$ in a stress–controlled plain concrete model under biaxial loading is analyzed as an example. With a



$$\begin{Bmatrix} \Delta\varepsilon_1 \\ \Delta\varepsilon_2 \end{Bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{Bmatrix} \Delta\sigma_1 \\ \Delta\sigma_2 \end{Bmatrix}$$

Fig. 6.2 – The Training Approach for Determining the Stress–Strain Matrix in a Stress-controlled Neural Network Material Model of Concrete

one–point representation scheme for the behavior of concrete in biaxial stress states, the network input consists of current stress-strain state and the stress increments (σ_1, σ_2, ε_1, ε_2, $\Delta\sigma_1$, $\Delta\sigma_2$), and the output includes the corresponding strain increments ($\Delta\varepsilon_1$, $\Delta\varepsilon_2$). In order to evaluate elements of the stress–strain matrix $[C_t]$ that correlates the expected strain increments with the given stress increments, as shown in Fig. 6.2, apparently, a set of data containing strain increments corresponding to stress states with a non–zero stress increment in one direction and a zero stress increment in another direction, should be added to the original training data set. Therefore, totally two sets of data are needed during training: 1) the strain increments when the stress increment in one direction is assumed to be non–zero, and zero in the remaining directions; and 2) the strain increments corresponding to nonzero stress increments.

Similarly, for other representation schemes, these two approaches can be extended to derive either the augmented representation scheme or the additional training data set for the explicit determination of the constitutive matrix.

CHAPTER 7

SUMMARY AND CONCLUSIONS

7.1 Summary

A neural network-based material modeling methodology for engineering materials is developed in this study. With this material modeling methodology, the stress-strain behavior of a material is captured within the distributed weight structure of a multilayer feedforward neural network trained directly on the stress-strain data obtained from experiments. The feasibility of this approach is verified through constitutive modeling of the behavior of plain concrete under monotonic proportional biaxial loadings and that under uniaxial cyclic compression. The general applicability of the approach is illustrated in the construction of a composite material model for reinforced concrete in a biaxial stress state by training stress–strain results on behaviors of reinforced concrete panels tested in both pure shear and combined shear with normal stresses (Vecchio and Collins, 1982).

A flexible and efficient simulator for multilayer feedforward neural networks is developed in this study, by implementing a dynamic node creation mechanism for architecture generation and a fast learning algorithm based on the Quickprop (Fahlman, 1988) and the Delta–Bar–Delta (Jacobs, 1987) algorithms for weight update. The simulator is successfully used in training different material models with varying sizes and degrees of complexity in the training data.

Representation schemes for modeling the stress–strain behavior of concrete and reinforced concrete under various stress states are developed for both stress-controlled and strain-controlled models. A one-point representation scheme is devised for capturing the behavior of concrete under biaxial monotonic loading, and a three-point representation scheme is introduced to account for the history dependency of the material behavior especially in cyclic loading. The three-point representation schemes are used for concrete models in biaxial compression to account for the unloading mechanism, and for reinforced concrete models to characterize the tensile strain softening behavior.

A scheme combining incremental training with a generalization evaluation is investigated to estimate the size of a quasi–optimal or quasi–minimal training data set, and the approach is illustrated in the constitutive modeling of both concrete and reinforced concrete materials.

With the use of neural networks, both stress–controlled and strain–controlled models can be readily built from experimental data. Because the experiment was usually conducted as stress–controlled, the simulation of material behavior with a stress–controlled model would be more straight forward, but a strain–controlled model is more readily usable when implemented within a finite element procedure. Both stress–controlled and strain–controlled models for concrete in biaxial stress states and for reinforced concrete are constructed, and their performance evaluated in the study. The learning and generalization capability of these neural network–based material models are verified through comparison of stress–strain behaviors predicted by a neural network with those obtained from experiments.

The neural network-based models of concrete in biaxial stress states are constructed from experimental results obtained by Kupfer, et al. (1969) and Nelissen (1971), with a one-point representation scheme. A three-point representation scheme is also utilized to study the effect of redundant representation on the learning and testing performance of material models. The presence of redundant information does not substantially improve the performance of these material models.

To add an unloading mechanism into the neural network-based material model, a simple elastic unloading scheme is studied in the training of a concrete model under biaxial compression with the use of a three-point representation scheme. The training data set is artificially generated with stress–strain data corresponding to two unloading cases where one starts unloading at a lower stress level and another at the highest stress level, and the testing case is the one that starts unloading at an intermediate stress level. The initial stiffness is used for the unloading stress–strain paths.

The neural network–based model for concrete under uniaxial cyclic compression is constructed with a three-point representation scheme, and trained on a set of experimental data from Karsan and Jirsa (1969). The predictability of the network that has been trained on a certain number of cycles of stress–strain data is investigated by testing the net on the stress–strain behavior of the next cycle. The testing is also conducted with a similar experiment reported by Sinha, et al. (1964).

The feasibility of applying the neural network–based approach to composite material modeling is studied by simulating the stress–strain behavior of reinforced concrete as a generic composite material; the experimental data used are from Vecchio and Collins' tests of reinforced concrete panels in both pure shear and combined shear and normal stresses (1982). Training data are extracted from experimental results on all the panels tested in pure shear. A general three–point representation scheme which includes stress–strain data, constituents material properties, and information of the reinforcements, is designed to capture the diverse material parameters of the composite. A quasi–minimal training data set is determined for this model, and the testing on generalization is conducted with panels tested in pure shear, combined shear with normal stresses, and pure shear with changing loading conditions.

Though the implementation of neural network–based material models in a finite element procedure is not conducted in the study, major issues with respect to the use of these models in finite element procedures for nonlinear structural analysis are generally discussed.

7.2 Conclusions

From the analysis and discussion on the performance of neural network–based material models of concrete and reinforced concrete under varying loading conditions, it can be concluded that the use of neural networks for the modeling of material behavior is viable and promising. Such an approach does not make a priori assumptions about the behavior of a material, but rather bases its prediction of stress–strain behavior on the experimental data with which it has been trained. Furthermore, the ability to employ neural networks for developing material models makes it possible to represent the state of material behavior knowledge as it evolves. Based on the results in this study, the following conclusions can be drawn:

- The neural network-based material modeling methodology is effective, flexible, and generally applicable for the modeling of complex behavior of engineering materials. With a proper design on the representation schemes, behaviors of concrete and reinforced concrete under different stress states can be readily captured in neural networks by training directly on the experimental results. The flexibility of representation scheme for the determination of material parameters presented in the input and output makes it possible to accommodate non–conventional material parameters.

- The simulator developed in this study provides a flexible and efficient modeling tool in solving the architecture determination problem and the estimation of a quasi–minimal training data set associated with the modeling of stress–strain behaviors of concrete and reinforced concrete. However, the use of a constant "trigger slope" for controlling the hidden node adaptation process is found to be not reliable, especially in the late training stage when a stringent learning accuracy is demanded. It is appears that a heuristic rule or manual control on the hidden node generation is more reliable.

- The scheme of incremental training with testing for generalization evaluation appears to be a viable way to estimate a minimal training data set for the construction of material models.

- A three-point representation scheme is able to include enough information to characterize the stress–strain history of a material. By including additional information about the material behavior, such problems as the representation of path dependency and the unique determination of stress–strain states can be resolved. A one–point representation scheme is sufficient to represent the stress–strain behavior of concrete in biaxial stresses under monotonic and proportional loadings.

- The predictions of the stress–strain behavior by neural network–based models of concrete and reinforced concrete match closely with the experimental results. These models can also generalize well on the untrained stress–strain paths by giving very reasonable predictions. For the reinforced concrete model, after training on results from pure shear tests, the generalization capability of the network is remarkably good even on results from tests in combined shear and normal stresses.

- The implementation of neural network–based material models within a finite element procedure can be readily accomplished for static analysis with a conjugate gradient solution methods and for dynamic analysis with explicit integration schemes.

7.3 Suggestions for Further Study

The neural network–based approach to material modeling in computational mechanics provides an alternative way to modeling the complex behavior of engineering materials including concrete and reinforced concrete. As has been discussed in this study, there are many benefits to using a neural network–based approach to material modeling; and there are also some drawbacks which should not be overlooked. In a word, this research is a first step in developing intelligent material models and much work needs to be done in the near future. A list of further extensions and some research needs arising out of this study are listed in the following paragraphs.

- The neural network material models of concrete and reinforced concrete need to be implemented within a finite element package to analyze reinforced concrete structures so that the validity of these material models on the structural level can be further studied, and associated issues with the implementation process should be investigated.

- The material model for concrete under uniaxial cyclic compression needs to be trained on stress–strain data with reloading and unloading inside the envelope stress–strain curve. Experimental data on those testings need to be compiled and processed.

- The approach developed in this study can be extended to model the stress–strain behavior of engineering materials in general stress states so long as the experimental results exist and an appropriate representation scheme can be designed. More experiments on the behavior of concrete under biaxial cyclic loading are required in order to build a comprehensive material model for concrete in biaxial stress states.

- It is felt that a series of comprehensive testing on different kinds of reinforced concrete structural members including panels need to be conducted in order to have enough experimental results for the construction of reasonable composite material models in different stress states.

- The approach can be extended to study the behavior of granular materials and soils as experimental results for those materials are abundant and readily available.

LIST OF REFERENCES

1. *AIP Conference Proceedings 151: Neural Networks for Computing*, J. S. Denker (Ed.), American Institute of Physics (AIP), Snowbird, UT, 1986.

2. Aleksander, I. (Ed.), *Neural Computing Architecture*, The MIT Press, Cambridge, MA, 1989.

3. Anderson, J. A., and Rosenfeld, E. (Eds.), *Neurocomputing: Foundations of Research*, The MIT Press, Cambridge, Massachusetts, 1988.

4. ASCE Task committee on Concrete and Masonry Structure, *State of the Art Report on Finite Element Analysis of Reinforced Concrete*, ASCE, New York, 1982.

5. Ash, T., "Dynamic Node Creation in Backpropagation Networks," *ICS Report 8901*, Institute for Cognitive Science, University of California, San Diego, La Jolla, Feb. 1989.

6. Barto, A. G., "Connectionist Learning for Control: An Overview," *COINS Technical Report 89–89*, Dept. of Computer and Information Science, University of Massachusetts. Amherst, MA, 1989.

7. Bathe, K.–J., *Finite Element Procedures in Engineering Analysis,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

8. Bazant, Z. P., "Endochronic Inelasticity and Incremental Plasticity," *International J. of Solids and Structures*, Vol. 14, pp. 691–714, 1978.

9. Bazant, Z. P., and Kim, S. S., "Plastic–Fracturing Theory for Concrete," *J. of Engineering Mechanics*, ASCE, Vol. 105, pp. 407–428, June 1979.

10. Bazant, Z. P., and Tsubaki, T., "Total Strain Theory and Path Dependence of Concrete," *J. of Engineering Mechanics*, ASCE, Vol. 106, pp. 1151–1173, Dec. 1978.

11. Becker, S., and Le Cun, Y., "Improving the Convergence of Backpropagation Learning with Second Order Methods," *Proceedings of the 1988 Connectionist Models Summer School*, Carnegie Mellon University, Pittsburgh, 1988.

12. Buyukozturk, O., "Nonlinear Analysis of Reinforced Concrete Structures," *Computers and Structures*, Vol. 7, pp. 149–156, 1977.

13. Carpenter, G. A., and Grossberg, S., "A Massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics, and Image Processing* 37, pp. 54–115, 1987.

14. Chen, A. C. T., and Chen, W. F., "Constitutive Relations for Concrete," *J. of Engineering Mechanics*, ASCE, Vol. 101, No. EM4, pp. 465–481, 1975.

15. Chen, W. F., *Plasticity in Reinforced Concrete*, McGraw–Hill, New York, 1982.

16. Cybenko, G., "Approximations by Superpositions of a Sigmoidal Function," *CSRD Report No. 856*, Center for Supercomputing Research and Development, University of Illinois at Urbana–Champaign, Feb. 1989.

17. Darwin, D., and Pecknold, D. A. W., "Inelastic Model for Cyclic Biaxial Loading of Reinforced Concrete," *Civil Engineering Studies*, Structural Research Series No. 409, University of Illinois at Urbana–Champaign, July 1974.

18. Desai, C. S., and Siriwardane, H. J., *Constitutive Laws for Engineering Materials*, Prentice–Hall, Inc., Engleweed Cliffs, NJ, 1984.

19. Fafitis, A., and Shah, S. P., "Constitutive Model for Biaxial Cyclic Loading of Concrete," *J. of Engineering Mechanics*, ASCE, Vol. 112, No. 8, pp. 760–775, August 1986.

20. Fahlman, S. E., "Faster-Learning Variations on Backpropagation: An Empirical Study," in *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann Publishers, Inc., 1988.

21. Fahlman, S. E., and Lebiere, C., "The Cascade-Correlation Learning Architecture," *Technical Report CMU-CS-90-100*, School of Computer Science, Carnegie Mellon University, Pittsburgh, Feb. 1990.

22. Fardis, M. N., Alibe, B., and Tassoulas, J. L., "Monotonic and Cyclic Constitutive Law for Concrete," *J. of Structural Engineering*, ASCE, Vol. 109, No. 2, pp. 516–536, April 1983.

23. Farley, B., and Clark, W. A., "Simulation of Self-organizing Systems by Digital Computer," *IRE Transactions on Information Theory* 4, pp. 76–84, 1954.

24. Gerstle, K. H., "Simple Formulation of Biaxial Concrete Behavior," *J. of American Concrete Institute*, Vol. 78, No. 1, pp. 62–68, 1981.

25. Ghaboussi, J., Garrett, Jr., J. H., and Wu, X., "Material Modeling with Neural Networks," in *Proceedings of the International Conference on Numerical Methods in Engineering: Theory and Applications*, Swansea, U. K., pp. 701–717, 1990.

26. Ghaboussi, J., Garrett, Jr., J. H., and Wu, X., "Knowledge-based Modeling of Material Behavior with Neural Networks," *J. of Engineering Mechanics*, ASCE, Vol. 117, No. 1, pp. 132–153, Jan. 1991.

27. Golub, G. H., and VanLoan, C. F., *Matrix Computations*, The Johns Hopkins University Press, 1983.

28. Grossberg, S., "Adaptive Pattern Classification and Universal Recoding: Part 1. Parallel Development and Coding of Natural Features Detectors," *Biological Cybernetics* 23, pp. 121–134, 1976.

29. Hageman, L. A., and Young, D. M., *Applied Iterative Methods,* The Academic Press, New York, 1981.

30. Hebb, D. O., *The Organization of Behavior*, New York, Wiley, 1949.

31. Hecht–Nielsen, R.,"Kolmogorov's Mapping Neural Network Existence Theorem," *Proceedings of the IEEE International Conference on Nerual Networks*, III–11, 1987.

32. Hecht-Nielsen, R., "Theory of the Backpropagation Neural Networks," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. I, pp. 593-605, 1988.

33. Hestenes, M. R., and Stiefel, E., "Methods of Conjugate Gradients for Solving Linear Systems," *J. Res. National Bureau of Standards* 49, pp. 409–436, 1952.

34. Hinton, G. E., Sejnowski, T. J., and Ackley, D. H., "Boltzman Machines: Constraint Satisfaction Networks That Learn," *Technical Report No. CMU–CS–84–110,* Department of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1984.

35. Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceeding of the National Academy of Sciences* 79, pp. 2554-2558, 1982.

36. Hornik, K., Stinchcomebe, M., and White, H., "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, Vol. 2, pp. 359–366, 1989.

37. Hughes, B. P., and Chapman, G. P., "The Complete Stress-strain Curve for Concrete in Direct Tension," *RILEM Bulletin*, No. 30, pp. 95–97, March 1966.

38. Hughes, T. J. R., Ferencz, R. M., and Hallquist, J. O., "Large-Scale Vectorized Implicit Calculations in Solid Mechanics on a CRAY X–MP/48 Utilizing EBE Preconditioned Conjugate Gradients," *Computer Methods in Applied Mechanics and Engineering* 61, pp. 215–248, 1987.

39. Jacobs, R. A., "Increased Rate of Convergence through Learning Rate Adaptation," *Technical Report* (COINS TR 87-117), Dept. of Computer and Information Science, University of Massachusetts at Amherst, MA, 1987.

40. Judd, S., *Neural Network Design and the Complexity of Learning*, The MIT Press, Cambridge, MA, 1990.

41. Karnin, E. D., "A Simple Procedure for Pruning Backpropagation Trained Neural Networks," *IEEE Transactions on Neural Networks*, Vol. 1, No. 2, pp. 239–242, June 1990.

42. Karsan, I. D., and Jirsa, J. O., "Behavior of Concrete under Compressive Loading," *J. of Engineering Mechanics*, ASCE, Vol. 99, No. EM4, pp. 2543–2563, August 1973.

43. Kesten, H., "Accelerated Stochastic Approximation," *Annals of Mathematical Statistics* 29, pp. 41–59, 1958.

44. Kohonen, T., *Self-organization and Associative Memory*, Springer-Verlag, New York, Second Edition, 1988.

45. Kupfer, H., Hilsdorf, H. K., and Rusch, H., "Behavior of Concrete Under Biaxial Stresses," *J. of American Concrete Institute*, Vol. 66, No. 8, pp. 656–666, 1969.

46. Kupfer, H., and Gerstle, K. H., "Behavior of Concrete under Biaxial Stresses," *J. of Engineering Mechanics*, ASCE, Vol. 99, No. EM4, pp. 852–866, August 1973.

47. Lapedes, A., and Farber, R., "Nonlinear Signal Processing Using Neural Networks: Prediction and System Modeling," *LA–UR–87–2662*, Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM, July 1987.

48. Lippmann, R. P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, pp. 4–22, April 1987.

49. Liu, T. C. Y., Nilson, A. H., and Slate, F. O., "Biaxial Stress–strain Relations for Concrete," *J. of Structural Engineering*, ASCE, Vol. 98, No. ST5, pp. 1025–1034, May 1972.

50. Mikkola, M. J., and Schnobrich, W. C., "Material Behavior Characteristics for Reinforced Concrete Shells Stressed beyond the Elastic Range," *Civil Engineering Studies*, Structural Research Series No. 367, University of Illinois at Urbana–Champaign, 1970.

51. McCulloch, W. S., and Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics* 5, pp. 115-133, 1943.

52. Minai, A. A., and Williams, R. D., "Acceleration of Backpropagation through Learning Rate and Momentum Adaptation," *Proceedings of the International Joint Conference on Neural Networks*, Washington, D. C., I–676, Jan. 1990.

53. Minsky, M. L., *Computation: Finite and Infinite Machines*, Prentice-Hall, 1967.

54. Minsky, M., and Papert, S., *Perceptrons*, The MIT Press, Cambridge, MA, 1969; the Expanded Edition, 1988.

55. Moody, J., "Fast Learning in Multi-resolution Hierarchies," in D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 1*, Morgan Kaufmann, 1989.

56. Mozer, M. C., and Smolensky, P., "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment," *CU–CS–421–89*, Dept. of Computer Science, University of Colorado at Boulder, Jan. 1989.

57. Nayak, G. C., and Zienkiewicz, O. C., "Elasto–Plastic Stress Analysis: A Generalization for Various Constitutive Relations Including Strain Softening," *International J. for Numerical Methods in Engineering*, Vol. 5, No. 1, pp. 113–135, 1972.

58. Nelissen, L. J. M., "Biaxial Testing of Normal Concrete," *Heron*, Delft, The Netherlands, Vol. 18, No. 1, pp. 1–90, 1972.

59. Orfanidis, S. J., "Gram-Schmidt Neural Nets," *Neural Computation* 2, pp. 116–126, 1990.

60. Ottosen, N. S., "Constitutive Model for Short-time Loading of Concrete," *J. of Engineering Mechanics*, ASCE, Vol. 105, No. EM1, pp. 127–141, 1979.

61. Parker, D. B., "Learning Logic," *Invention report, S81–64, File 1*, Office of Technology Licensing, Stanford University, October 1982.

62. *Proceedings of the IEEE First International Conference on Neural Networks*, IEEE, New York, June 1987.

63. *Proceedings of the IEEE International Conference on Neural Networks*, IEEE, New York, June 1988.

64. *Proceedings of the 1988 Connectionist Models Summer School*, D. Touretzsky, G. Hinton, and T. Sejnowski (Eds.), Carnegie Mellon University, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

65. *Proceedings of the 1988 IEEE Conference on Neural Information Processing Systems (NIPS) – Natural and Synthetic: Advances in Neural Information Processing Systems 1*, D. S. Touretzky (Ed.), Morgan Kaufmann Publishers, San Mateo, CA, 1989.

66. *Proceedings of the 1989 IEEE Conference on Neural Information Processing Systems (NIPS) – Natural and Synthetic: Advances in Neural Information Processing Systems 2*, D. S. Touretzky (Ed.), Morgan Kaufmann Publishers, San Mateo, CA, 1990.

67. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Co–Sponsored by IEEE and the International Neural Network Society (INNS), Washington, D. C., 1989.

68. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Co–Sponsored by IEEE and the International Neural Network Society (INNS), Washington, D. C., 1990.

69. *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, Co–Sponsored by IEEE and the International Neural Network Society, San Diego, 1990.

70. *Proceedings of the Third International Conference on Genetic Algorithms*, J. D. Schaffer (Ed.), Morgan Kaufmann Publishers, San Mateo, CA, 1990.

71. Rochester, N., Holland, J. H., Haibt, L. H., and Duda, W. L., "Tests on a Cell Assembly of the Action of the Brain, Using a Large Digital Computer," *IRE Transactions on Information Theory* 2, pp. 80–93, 1956.

72. Rosenblatt, F., "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," *Psychological Review* 65, pp. 368-408, 1958.

73. Rosenblatt, F., *Principles of Neurodynamics*, Spartan Books, New York, 1962.

74. Rumelhart, D., and Zipser, D., "Feature Discovery by Competitive Learning," *Cognitive Science* 9, pp. 75–112, 1985.

75. Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing, Vol. 1: Foundations,* D. E. Rumelhart, and J. L., McClelland (Eds.), The MIT Press, MA, 1986.

76. Rumelhart, D., and McClelland, J. (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition; Vol. 1: Foundations*, The MIT Press, Cambridge, MA, 1986.

77. Saridis, G. N., "Learning Applied to Successive Approximation Algorithms," *IEEE Transactions on Systems Science and Cybernetics,* SSC–6, pp. 97–103, 1970.

78. Sinha, B. P., Gerstle, K. H., and Tulin, L. G., "Stress-strain Relations for Concrete under Cyclic Loading," *J. of American Concrete Institute*, Vol. 61, No. 2, pp. 195–211, Feb. 1964.

79. Smith, G. M., and Young, L. E., "Ultimate Theory in Flexure by Exponential Function," *ACI Journal*, Vol. 52, No. 3, pp. 349–359, Nov. 1955.

80. Tenorio, M. F. M., and Lee, W.–T., "Self–Organizing Neural Network for Optimum Supervised Learning," *TR–EE–89–30,* School of Electrical Engineering, Purdue University, June 1989.

81. Vecchio, F., and Collins, M. P., "The Response of Reinforced Concrete to Inplane Shear and Normal Stresses," *Publication No. 82–03,* Dept. of Civil Engineering, University of Toronto, 1982.

82. Warbos, P., *Beyond Regression: New Tools for Prediction and Analysis in Behavioral Sciences*, Ph.D. Thesis, Harvard University, 1974.

83. Watrous, R. L., "Learning Algorithm for Connectionist Networks: Applied Gradient Methods of Nonlinear Optimization," *Proceedings of the IEEE International Conference on Neural Networks*, Vol. II, pp. 619-627, 1987.

84. Widrow, B., and Hoff, M. E., "Adaptive Switching Circuits," *1960 IRE WESCON Convention Record*, New York, pp. 96-104, 1960.

85. William, K. J., and Warnke, E. P., "Constitutive Model for the Triaxial Behavior of Concrete," International Association of Bridge and Structural Engineers, *Seminar on Concrete Structure Subjected to Triaxial Stresses*, Paper III−1, Bergamo, Italy, May 17−19, 1974.

86. Winter, G., and Nilson, A. H., *Design of Concrete Structures*, 9th Edition, McGraw-Hill, New York, 1979.

87. Wolpert, D. H., "A Mathematical Theory of Generalization," *Complex Systems* 4, pp. 151−249, 1990.

88. Yankelevsky, D. Z., and Reinhardt, H. W., "Model for Cyclic Compressive Behavior of Concrete," *J. of Structural Engineering,* ASCE, Vol. 113, No. 2, pp. 228−240, Feb. 1987.

89. Zienkiewicz, O. C., *The Finite Element Method*, The Third Edition, McGraw-Hill, New York, 1977.