CONSTRAINED NEURAL NETWORKS

by

Ümit Özer

B.S., Chemical Engineering, Boğaziçi University, 2010

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Chemical Engineering

Boğaziçi University

2012

ACKNOWLEDGMENTS

I would like to express my grateful thanks to my thesis advisor Prof. Uğur Akman for his everlasting support and encouragement during my graduate study.

I would also like to thank to the members of my thesis committee for their time devoted to read and comment on my thesis as well as to my professors at the Department of Chemical Engineering who provided me a high quality education.

I would also like to thank my family for their support.

I am also indebted to all my friends for their friendship and support throughout the thesis.

ABSTRACT

CONSTRAINED NEURAL NETWORKS

Artificial Neural Networks (NNs) are very functional tools employed in regression, forecasting, classification, pattern recognition, data-based process modeling, and nonlinear process control. However, the data-based process modeling via NNs may be problematic since the field of chemical engineering has many strict rules in the form of implicit or explicit inequality and/or equality constraints that need to be satisfied, such as the mass and energy balances, bounds on mole fractions, conversion limits, and safety limits. The purpose of the thesis is to develop novel methods to implement equality and/or inequality constraints on/between the outputs of a feedforward NN during its training phase so that constraint violations of the model constructed by the NN are within the tolerable limits both in the training and testing phases. For this purpose, two alternative methods are developed. In the first method, the functions of MATLAB's NN Toolbox are utilized however an unconstrained or a constrained generic optimizer from MATLAB's Optimization Toolbox is employed for the training of the NN. In the second method, the algorithm is the same as in the first method however the NN building and NN topology functions of NN Toolbox are not used, instead these are coded explicitly. Direct or indirect (penalty function) constraint handling approaches are used for the implementation of inequality and/or equality constraints. The two methods of Constrained Neural Networks (CNNs), tested on six examples, accomplished the purpose of the thesis; they successfully learn the equality and/or inequality constraints imposed on/between the outputs of the NN together with the relations between the input and output data. The performances of the CNNs are more pronounced especially in realistic cases where the data, corrupted by measurement errors, do not satisfy the constraints. The CNNs act as model-/constraint-based filters and can learn and forecast process behavior even under noise-corrupted data via satisfying the imposed inequality and/or equality constraints.

ÖZET

KISITLI YAPAY SİNİR AĞLARI

Yapay sinir ağları regresyon, tahmin, sınıflandırma, örüntü tanıma, veri tabanlı proses modelleme ve lineer olmayan proses denetimi gibi alanlarda kullanılan işlevsel bir araçtır. Fakat, kimya mühendisliği alanında, kütle ve enerji denklikleri, mol kesri ve dönüşüm yüzdesi sınırları ve güvenlik limitleri gibi eşitlik yada eşitsizlik kısıtları göz önünde bulundurulmalıdır. Bu gibi kısıtlardan dolayı yapay sinir ağlarının kimya mühendisliği gibi katı kurallar içeren alanlarda kullanımı sorun yaratabilmektedir. Bu tez çalışmasında, ileri beslemeli sinir ağları için, sistemin içerdiği ya da dışarıdan çıktılar üzerinde tanımlanan eşitlik ya da eşitsizlik kısıtlarının girdiler ve çıktılar arasındaki ilişki ile birlikte yapay sinir ağına öğretilmesi için yeni yöntemler geliştirilmiştir. Bu sayede, oluşturulan sinir ağının öğrenme ve sınama aşamasındaki kısıt ihlaleri istenilen seviyede kalmıştır. İki alternatif yöntem geliştirilmiştir. Birincisinde MATLAB'ın yapay sinir ağları araç kutusu fonksiyonları kullanılmış, fakat öğrenme aşamasında MATLAB'ın optimizasyon araç kutusunun kısıtlı ya da kısıtsız optimizasyon fonksiyonları kullanılmıştır. İkinci yöntemde ise, ilk yöntemde sinir ağlarını oluşturmak için kullanılan yapay sinir ağları araç kutusu fonksiyonları elle kodlanmıştır. Eşitlik yada eşitsizlik kısıtlarının uygulanmasında doğrudan ve dolaylı (ceza fonksiyonu) olmak üzere iki yöntem uygulanmıştır. Geliştirilen iki adet kısıtlı sinir ağı yöntemi altı örnek üzerinde uygulanmış ve tezde istenilen amaca ulaşılmıştır. Sinir ağları, girdiler ve çıktılar arasındaki bağıntı ile birlikte, çıktılar üzerindeki eşitlik yada eşitsizlik kısıtlarını da başarıyla öğrenmişlerdir. Bu yöntemlerin başarısı özellikle ölçüm hatası içeren verilerin olduğu gerçek durumlarda daha çok ortaya çıkmaktadır, çünkü bu gibi durumlarda öğrenme verisi de kısıt ihlali içermektedir. Kısıtlı sinir ağları, model-/kısıt tabanlı filtreler gibi davranıp, ölçüm hatası içeren verilerde dahi eşitlik ya da eşitsizlik kısıtlarını sağlayarak prosesi öğrenebilmekte ve tahminde bulunabilmektedirler.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv
ABSTRACT v
ÖZET vi
LIST OF FIGURES viii
LIST OF TABLES xiii
LIST OF SYMBOLS xiv
LIST OF ACRONYMS/ABBREVIATIONS xvi
1. INTRODUCTION 1
2. INTRODUCTION TO NEURAL NETWORKS 3
2.1. Multilayer Neural Networks 4
2.2. Activation Functions 6
2.2.1. Logistic Function 6
2.2.2. Hyperbolic Tangent Function 7
2.2.3. Hyperbolic Tangent Sigmoid Function 7
2.2.4. Linear Function 8
2.2.5. Gaussian Function 9
2.3. Designing the Neural Network 9
2.4. Initializing the Neural Network 10
2.5. Training the Neural Network 11
2.5.1. Levenberg – Marquardt Method 13
2.5.2. Regularization 14
3. CONSTRAINED NEURAL NETWORK (CNN) ALGORITHMS 16
3.1. First Method for Constrained Neural Networks - CNN-I 18
3.2. Second Method for Constrained Neural Networks - CNN-II 20
3.3. Third Method for Constrained Neural Networks - CNN-III 23
3.4. Some Application Details of CNN-I 24
3.5. Some Application Details of CNN-II 28
3.6. Some Details on Optimization and Constraint Handling 30
3.6.1. Penalty Function Technique 31

3.6.2. Constraint Aggregation 32
3.7. Notation 34
4. APPLICATIONS OF CONSTRAINED NEURAL NETWORKS 36
4.1. Synthetic Time Series 36
4.1.1. Synthetic Linear Time Series 36
4.1.2. Synthetic Nonlinear Time Series 54
4.2. Ternary Vapor-Liquid Equilibrium 64
4.2.1. Reproducibility Analysis 75
4.3. Ternary Vapor-Liquid Equilibrium with Measurement Errors 78
4.4. Propylene Glycol Production in CSTR 88
4.5. Runaway in a Packed Bed Reactor 101
4.6. Williams-Otto Plant 112
5. CONCLUSIONS AND RECOMMENDATIONS 126
5.1 Recommendations 131
APPENDIX: SOME SELECTED MATLAB CODES 133
A.1. Source Code of Direct Constraint Implementation of CNN-I for Section 4.1.1 of Synthetic Linear Time Series 133
A.2. Source Code of Indirect Constraint Implementation of CNN-I for Section 4.1.1 of Synthetic Linear Time Series 138
A.3. Source Code of Direct Constraint Implementation of CNN-II for Section 4.1.1 of Synthetic Linear Time Series 143
A.4. Source Code of Indirect Constraint Implementation of CNN-II for Section 4.1.1 of Synthetic Linear Time Series 148
A.5. Source Codes of Common Functions Employed in CNN-II 153
REFERENCES 157

LIST OF FIGURES

Figure 2.1.	A biological neuron (Samarasinghe, 2007).	3
Figure 2.2.	Neuron representation (Samarasinghe, 2007).	3
Figure 2.3.	Biological (left) and Artificial (right) Neural Networks (Samarasinghe, 2007).	4
Figure 2.4.	Multilayer Neural Network.	5
Figure 2.5.	A neuron and its activation function (Samarasinghe, 2007).	6
Figure 2.6.	Logistic function (Samarasinghe, 2007).	7
Figure 2.7.	Hyperbolic tangent function (Samarasinghe, 2007).	7
Figure 2.8.	Hyperbolic tangent sigmoid function.	8
Figure 2.9.	Linear function.	8
Figure 2.10.	Gaussian function (Samarasinghe, 2007).	9
Figure 2.11.	One input, one hidden, and one output multilayer NN.	12
Figure 2.12.	Principle of training an Artificial NN (Beale et al., 2010).	13
Figure 3.1.	Illustration of CNN-I design.	17
Figure 3.2.	Illustration of CNN-II design.	17
Figure 3.3.	Illustration of CNN-III design.	18
Figure 3.4.	Multilayer feedforward Neural Network.	21
Figure 3.5.	New notation of NN structure.	22
Figure 3.6.	Output order of the `getx` function.	25
Figure 3.7.	Input order of the `setx` function.	25
Figure 3.8.	Data preprocessing.	26
Figure 3.9.	Optimizer interface of the CNN-I scheme.	27
Figure 3.10.	CNN-II training phase.	29

Figure 3.11. Implicit representation of the block of NN functions. 34

Figure 4.1. NN model of Synthetic Linear Time Series. 37

Figure 4.2. Output **X** of CNN-I-D for the Synthetic Linear Time Series. 39

Figure 4.3. Outputs **Y** and **Z** of CNN-I-D for the Synthetic Linear Time Series. .. 40

Figure 4.4. Output **X** of CNN-II-D for the Synthetic Linear Time Series. 41

Figure 4.5. Outputs **Y** and **Z** of CNN-II-D for the Synthetic Linear Time Series. 41

Figure 4.6. Regression plot of the complete data set for CNN-II-D case. 42

Figure 4.7. Regression plot of output X in CNN-II-D case for training data (effect of the inequality constraint $\mathbf{X} \geq -\mathbf{0.1}$ on the NN outputs is clearly visible). 43

Figure 4.8. Regression plot of output Y in CNN-II-D case for training data. 43

Figure 4.9. Regression plot of output Z in CNN-II-D case for training data. 44

Figure 4.10. Regression plot of output X in CNN-II-D case for test data (effect of theinequality constraint $\mathbf{X} \geq -\mathbf{0.1}$ on the NN outputs is vaguely visible). 44

Figure 4.11. Regression plot of output Y in CNN-II-D case for test data. 45

Figure 4.12. Regression plot of output Z in CNN-II-D case for test data. 45

Figure 4.13. Output X of CNN-I-P with penalty function for Synthetic Linear Time Series. 47

Figure 4.14. Outputs Y and Z of CNN-I-P with penalty function for the Synthetic Linear Time Series. 47

Figure 4.15. Output X of CNN-II-P with penalty function for Synthetic Linear Time Series. 48

Figure 4.16. Outputs Y and Z of CNN-II-P with penalty function for the Synthetic Linear Time Series. 49

Figure 4.17. Weights and biases of the NN Toolbox and CNN methods for Synthetic Linear Time Series. 50

Figure 4.18. Progress of SSE in training by NN Toolbox, CNN-I, and CNN-II for the Synthetic Linear Time Series. 51

Figure 4.19. Training CPU times for the Synthetic Linear Time Series. 52

Figure 4.20. NN model of Synthetic Nonlinear Time Series. 55

Figure 4.21. Outputs of CNN-II-D for Synthetic Nonlinear Time Series. 57

Figure 4.22. Outputs of CNN-II-P for Synthetic Nonlinear Time Series. 59

Figure 4.23. Weights and biases of the NN Toolbox and CNN-II for Synthetic Nonlinear Time Series. 60

Figure 4.24. Progress of SSE in training by NN Toolbox and CNN-II for Synthetic Nonlinear Time Series. 61

Figure 4.25. Training CPU times for Synthetic Nonlinear Time Series. 62

Figure 4.26. NN model of ternary VLE. 64

Figure 4.27. Outputs of CNN-II-D for ternary VLE. 67

Figure 4.28. Violations of equality constraints for CNN-II-D for ternary VLE data. 68

Figure 4.29. Outputs of CNN-II-P for ternary VLE. 70

Figure 4.30. Violations of equality constraints for CNN-II-P for ternary VLE data. 71

Figure 4.31. Weights and biases of the NN Toolbox and CNN-II for ternary VLE. 72

Figure 4.32. Progress of SSE in training by NN Toolbox and CNN-II for ternary VLE. 73

Figure 4.33. Training CPU times for ternary VLE. 73

Figure 4.34. Sensitivity of NN Toolbox to random initializations. 76

Figure 4.35. Sensitivity of CNN-II-D to random initializations. 76

Figure 4.36. Sensitivity of CNN-II-P to random initializations. 77

Figure 4.37. Outputs of CNN-II-D for ternary VLE with measurement errors. 79

Figure 4.38. Violations of equality constraints for CNN-II-D for ternary VLE with measurement errors. 80

Figure 4.39. Violations of equality constraints for CNN-II-D for ternary VLE with measurement errors in the training phase. 81

Figure 4.40. Outputs of CNN-II-P for ternary VLE with measurement errors. 82

Figure 4.41. Violations of equality constraints for CNN-II-P for ternary VLE with measurement errors. 83

Figure 4.42. Violations of equality constraints for CNN-II-P for ternary VLE with measurement errors in the training phase. 83

Figure 4.43. Weights and biases of the NN Toolbox and CNN-II for ternary VLE with measurement errors. 85

Figure 4.44. Progress of SSE in training by NN Toolbox and CNN-II for ternary VLE with measurement errors. 86

Figure 4.45. Training CPU times for ternary VLE with measurement errors. 86

Figure 4.46. Propylene Glycol production in CSTR. 89

Figure 4.47. NN model of Propylene Glycol production. 91

Figure 4.48. Outputs of CNN-II-D for Propylene Glycol production. 93

Figure 4.49. Violations of equality constraints for CNN-II-D for Propylene Glycol production. 94

Figure 4.50. Outputs of CNN-II-P for Propylene Glycol production. 96

Figure 4.51. Violations of equality constraints for CNN-II-P for Propylene Glycol production. 97

Figure 4.52. Weights and biases of the NN Toolbox and CNN-II for Propylene Glycol production. 98

Figure 4.53. Progress of SSE in training by NN Toolbox and CNN-II for Propylene Glycol production. 99

Figure 4.54. Training CPU times for Propylene Glycol production. 100

Figure 4.55. PBR. 102

Figure 4.56. Input and output data for PBR. 103

Figure 4.57. NN model of PBR. 104

Figure 4.58. Outputs of CNN-II-D for PBR. 106

Figure 4.59. Outputs of CNN-II-P for PBR. 108

Figure 4.60. Weights and biases of the NN Toolbox and CNN-II for PBR. 109

Figure 4.61. Progress of SSE in training by NN Toolbox and CNN-II for PBR. 110

Figure 4.62. Training CPU times for PBR. 111

Figure 4.63. Williams-Otto Plant floowsheet. 113

Figure 4.64. NN model of WOP. 116

Figure 4.65. Outputs of CNN-II-D for WOP. 118

Figure 4.66. Violations of equality constraints for CNN-II-D for WOP. 119

Figure 4.67. Outputs of CNN-II-P for WOP. 120

Figure 4.68. Violations of equality constraints for CNN-II-P for WOP. 121

Figure 4.69. Weights and biases of the NN Toolbox and CNN-II for WOP. 122

Figure 4.70. Progress of SSE in training by NN Toolbox and CNN-II for WOP. ... 123

Figure 4.71. Training CPU times for WOP. 123

LIST OF TABLES

Table 3.1.	NN Toolbox functions employed in CNN-I.	24
Table 4.1.	Comparison of NN Toolbox and CNN methods for the Synthetic Linear Time Series.	53
Table 4.2.	Comparison of NN Toolbox and CNN-II method for Synthetic Nonlinear Time Series.	63
Table 4.3.	Comparison of NN Toolbox and CNN-II method for ternary VLE data.	74
Table 4.4.	Comparison of NN Toolbox and CNN-II Method for ternary VLE with measurement errors.	87
Table 4.5.	Comparison of NN Toolbox and CNN-II method for Propylene Glycol production.	100
Table 4.6.	Comparison of NN Toolbox and CNN-II method for PBR.	111
Table 4.7.	WOP rate equation constants.	114
Table 4.8.	Parameter specifications of WOP.	115
Table 4.9.	Comparison of NN Toolbox and CNN-II method for WO Plant.	124

LIST OF SYMBOLS

A	Constant multiplier for the reaction rate equation
b	Biases of the Neural Network
C	Augmented vector of inequality constraints
C_A	Concentration of reactant
C_{A0}	Reactant inlet concentration
C_P	Heat capacity of the materials
d_m	Sum of error gradients
F	Mass flow of materials
F_A	Fresh feed of component A
F_{A0}	Mass flow rate of entering reactant
F_B	Fresh feed of component B
F_m	Total stream flow rates (m: R, S, D, P, T, Y)
$F_{m,i}$	Component flow rates (i: A, B, C, E, G, P)
$\mathbf{g}$	Vector of inequality constraints
$\mathbf{h}$	Vector of equality constraints
$\mathbf{H}$	Hessian matrix
$\mathbf{I}$	Identity matrix
k	Rate constant
$KS(x, \rho)$	Kreisselmeier-Steinhauser function
$L(u)$	Activation function
m_c	Cooling fluid mass flow
Q_g	Amount of heat generated in the reactor
Q_r	Amount of heat removed from the reactor
R	Reaction rate equations
R^2	Regression coefficients
r_a	Reaction rate
s	Neural Network topology
t	Target data
T	Temperature
T_a	Cooling fluid temperature

T_{out}	Outlet temperature
T_0	Inlet temperature
u	Input of the hidden layer neurons
v	Weighted sum of inputs to output layer
V	Reactor volume
w	Weights of the Neural Network
X	Conversion
x	Inputs of the Neural Network
x^L	Lower bound of x decision variables
x^U	Upper bound of x decision variables
x^t	Inputs training data
x^s	Inputs testing data
y	Output of hidden layer neurons
y^t	Targets training data
z	Outputs of the Neural Network
z^t	Outputs training data
z^s	Outputs testing data
α	Split fraction
δ	Regularization parameter
ϵ	Gaussian white noise
η	Gaussian white noise
$\mathcal{N}(s, w, x)$	Implicit notation of Neural Network block functions
μ	Penalty parameter
ξ	Gaussian white noise
ρ	Approximation parameter
$\sigma(u)$	Activation function
ϑ_0	Entering volumetric flow rate

LIST OF ACRONYMS/ABBREVIATIONS

CNN	Constrained Neural Network
CNN-I	First method of Constrained Neural Networks
CNN-I-D	Direct constraint implementation case of first method of Constrained Neural Networks
CNN-I-P	Indirect constraint implementation case of first method of Constrained Neural Networks
CNN-II	Second method of Constrained Neural Networks
CNN-II-D	Direct constraint implementation case of second method of Constrained Neural Networks
CNN-II-P	Indirect constraint implementation case of second method of Constrained Neural Networks
CNN-III	Third method of Constrained Neural Networks
MSE	Mean square error
NN	Neural Network
NN Toolbox	Neural Network Toolbox of MATLAB
ODE	Ordinary differential equation
SSE	Sum of square error
VLE	Vapor liquid equilibrium
WOP	Williams Otto plant

1. INTRODUCTION

The term Neural Network (NN) is actually used for the network of biological neurons. However, in computational science and engineering it is used in place of the term Artificial Neural Network (ANN), which consists of artificial neurons and refers to computational mimicking of real biological neurons. From now on, the term NN will be used throughout the thesis. NNs are formed by interconnecting artificial neurons that mimic the functions of biological neuron. Thus, problem solving method of the brain, and memory can be used (simulated) by NNs. The entire mechanism of a NN is highly complex. NNs use the basic steps of problem solving methods of biological NNs. A biological body mainly makes three steps in life experiences. First, it observes a situation, and makes mistakes (errors). Then, it learns from these mistakes. Finally, it uses this knowledge for oncoming similar situations. NNs do not need to form a complete model of a system. The network can be enlarged or reduced with respect to complexity of a system. Hence, they can be used in wide range of areas.

The NNs are mainly used to accomplish the following tasks:

- Function fitting.
- Pattern recognition.
- Data clustering.
- Time series learning and forecasting.

Multilayer NNs are the most basic type NNs and they are usually used in function fitting. In this thesis, equality and/or inequality constraints are applied to multilayer NNs. The proposed novel approaches can be easily integrated to other NN types since the training algorithms and mathematical models of NNs are generally similar to each other.

A NN algorithm consists of four basic sections. They are the preprocessing of data, creation and configuration of the NN, training of the NN, and simulation of the NN. This thesis concerns mainly with the training of the NN. During the learning step (training of the network), existing NN algorithms use the unconstrained optimization algorithms. The

Levenberg-Marquardt algorithm is the most widely used algorithm, and it is very effective. A general explanation of NNs is given in Chapter 2.

In many systems that are of interest for, for instance, chemical engineers, the problems have constraints originating from mass and energy balances. The existing NN tools cannot be used in such systems because the NNs trained with unconstrained optimization algorithms may violate the system balances. For example, the total mole fraction may not be equal to one during the training of a NN. Thus, the results that violate system balances in the training step cannot be reliable in the simulation (testing, forecasting) step. In the literature, there is not any NN algorithm which considers the constrained systems. The main purpose of this thesis is to generate NN algorithms and tools that take into account the system constraints by using constrained and unconstrained optimization methods. The unconstrained optimization methods are used with the penalty function method. The proposed approaches will enable the use of NN tools in constrained and unconstrained learning environments. In Chapter 3, these novel approaches will be given in detail.

In Chapter 4, several example problems pertinent to chemical engineering and other areas are presented and the results of the classical NN training and constrained NN training will be compared. Conclusions and recommendations are made in Chapter 5.

Two different approaches are considered in this thesis in developing and implementing of the constrained NNs. One approach directly utilizes the NN Toolbox of MATLAB as building blocks (object oriented). The other approach does not rely on MATLAB's NN Toolbox, but involves explicit implementation of the NN equations in constrained or unconstrained optimization environments. The sample codes are given at appendix section.

2. INTRODUCTION TO NEURAL NETWORKS

NNs copy the working mechanism of neural systems, and they use this mechanism in problem solving. A biological neuron consists of a cell body, dendrites, and an axon, which are shown at the Figure 2.1.

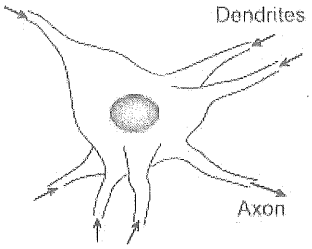


Figure 2.1. A biological neuron (Samarasinghe, 2007).

A biological neuron takes input signals from dendrites. The signals weighted with their signal strengths while they are going through a cell body. All weighted signals are summed up, and then the sum is processed by the cell body. The output is transferred through axons to other neurons.

An artificial neuron mimics the system described above. In Figure 2.2, the working scheme of an artificial neuron is shown. First, the inputs are weighted, and added up. Then, the sum is processed by the activation function. Finally, the output is send to the other artificial neurons.

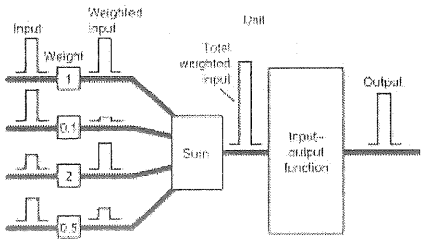


Figure 2.2. Neuron representation (Samarasinghe, 2007).

Biological neurons are connected each other (Figure 2.3), but without physical connection. They transmit the signals with neurotransmitters from one cell to another.

When a signal comes to the neuron, it is multiplied by some weight, processed, and transmitted to another cell. The weights are changed with respect to the passing signal. If same signal is passing many times, the weights are adapted for these signals. Then, the network can produce an output with a partial input or without an actual input. Within these networks, the brain is the largest neural network. It uses this mechanism to produce signals such as remembering a memory with a word or taking a breath unconsciously. The memory is the most amazing property of the brain.

A NN uses this logic to solve problems. First, the network is trained with input – output data which contain the information of a system. The weights are adjusted during the training, thus, NNs learn the correlation between the input and the output of a system. Then, the trained network is used to estimate outputs for new inputs. Therefore, a NN provides a correlation between an input and an output of a system by its trained weights and activation functions (Samarasinghe, 2007).

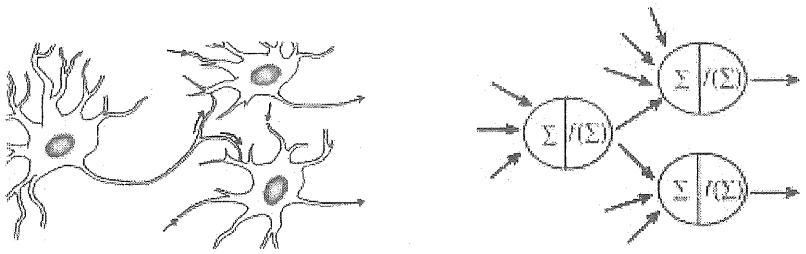


Figure 2.3. Biological (left) and Artificial (right) Neural Networks (Samarasinghe, 2007).

2.1. Multilayer Neural Networks

In order to handle the complexity of a system, NNs can be adjusted by increasing or reducing the number of neurons in the network. Most of the time, the relation between inputs and outputs are nonlinear. Layers of neurons in series can process the input signals to the desired output signals. Multilayer NNs are networks which consist of many layers of neurons as the name implies. These are suitable networks to obtain the nonlinear input-output relationships. The network constructs these relationships by activation functions inside hidden layer neurons. The output is formed from these nonlinear functions by modifying their weights. First layer of neurons is called an input layer, last layer of neurons is called an output layer, and layers between those are called hidden layers. The

number of neurons in the hidden layer and the number of hidden layers are important for handling the nonlinearity. If there is not enough number of neurons, misfitting occurs. If the neuron number is more than enough, then overfitting occurs. A general multilayer NN outlook is depicted by Figure 2.4.

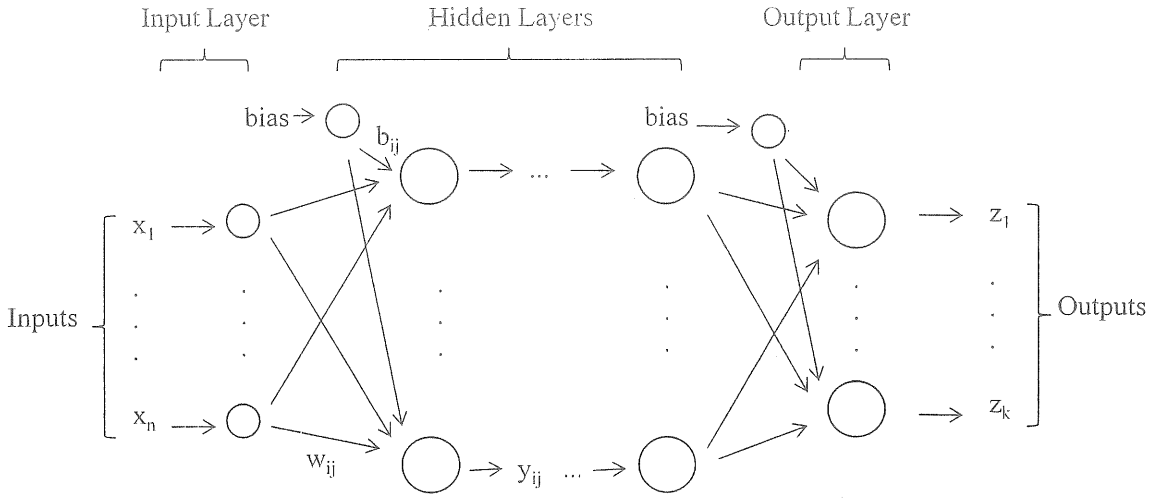


Figure 2.4. Multilayer Neural Network.

The input of the hidden layer neurons can be described with:

$$u_j = b_{0j} + \sum_{i=1}^{n} w_{ij} x_i \tag{2.1}$$

The output from hidden layer neurons can be described with:

$$y_j = f(u_j) \tag{2.2}$$

The weighted sum of inputs to output layer can be described with:

$$v_k = b_{0k} + \sum_{j=1}^{m} w_{jk} y_j \tag{2.3}$$

The outputs of the network can be described with:

$$z_k = f(v_k) \tag{2.4}$$

2.2. Activation Functions

Activation functions process the input coming into the neuron. The input to the activation function is the sum of weighted inputs coming to the neuron. A neuron with its activation function σ is depicted in Figure 2.5. If a nonlinear function is used in the neuron, the nonlinear relationship between input and output data can be obtained by adjusting the input weights. The activation functions are also called transfer functions.

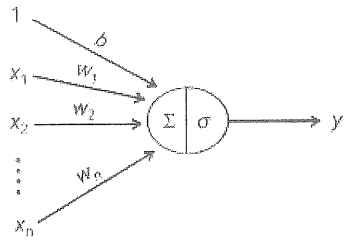


Figure 2.5. A neuron and its activation function (Samarasinghe, 2007).

The sum of inputs is:

$$u = b + w_1 x_1 + \cdots + w_n x_n \tag{2.5}$$

The output of the neuron is:

$$y = \sigma(\sum_1^n w_n x_n + b) \tag{2.6}$$

Sigmoid functions are S-shaped functions, such as logistic and hyperbolic tangent functions. Gaussian, sine, and arc tangent are commonly used as the activation function.

2.2.1. Logistic Function

Logistic function (Figure 2.6) is the widely used sigmoid function. The logistic function maps the input between zero and one. The nonlinearity can be adjusted by parameters w and b that enter the logistic function equation via u. They adjust the slope and position, respectively.

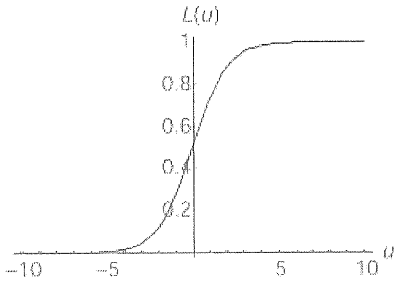


Figure 2.6. Logistic function (Samarasinghe, 2007).

The logistic equation is:

$$y = L(u) = \frac{1}{1+e^{-u}} \tag{2.7}$$

2.2.2. Hyperbolic Tangent Function

Another widely used sigmoid function is the hyperbolic tangent function (Figure 2.7).

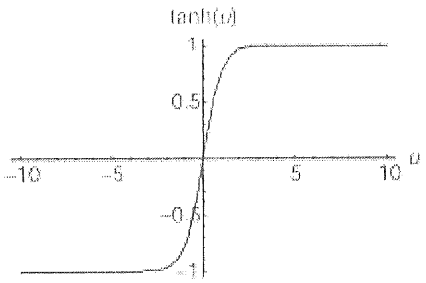


Figure 2.7. Hyperbolic tangent function (Samarasinghe, 2007).

The hyperbolic tangent equation is:

$$y = L(u) = \frac{1+e^{-u}}{1-e^{-u}} = \frac{\sinh(u)}{\cosh(u)} \tag{2.8}$$

This function maps the input between minus one and plus one.

2.2.3. Hyperbolic Tangent Sigmoid Function

Another widely used sigmoid function is the hyperbolic tangent sigmoid function (Figure 2.8).

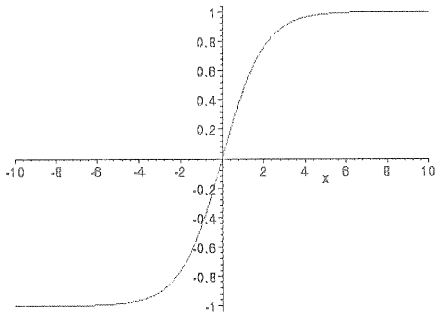


Figure 2.8. Hyperbolic tangent sigmoid function.

The hyperbolic tangent sigmoid equation is:

$$y = L(u) = \frac{2}{1+e^{-2u}} - 1 \tag{2.9}$$

This function maps the input between minus one and plus one.

2.2.4. Linear Function

Linear function is widely used in output neuron (Figure 2.9).

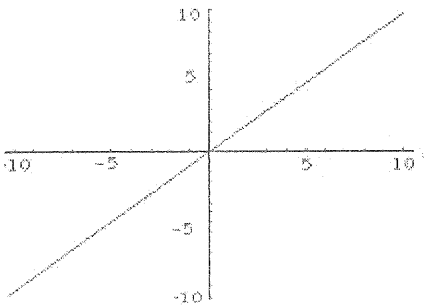


Figure 2.9. Linear function.

The linear function equation is:

$$y = L(u) = u \tag{2.10}$$

2.2.5. Gaussian Function

The standard normal curve has a symmetric bell shape and represents the standard normal distribution. Its mean is zero and standard deviation is one. The range of the function is zero to one. The function has its maximum when weighted sum of inputs is zero (u=0). Hence, it is highly sensitive to u values around zero. The Gaussian function is depicted in Figure 2.10.

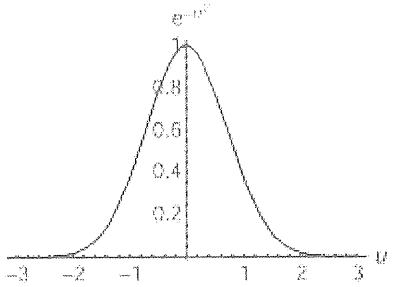


Figure 2.10. Gaussian function (Samarasinghe, 2007).

The Gaussian equation is:

$$y = e^{-u^2} \tag{2.11}$$

The Gaussian-type activation functions are used mainly in the radial basis function NNs.

2.3. Designing the Neural Network

As mentioned before, there are three main steps in biological neural network. These are observing, learning, and implementation. These steps of designing a NN are elaborated as follows (Beale, Hagan, & Demuth, 2010):

- Collect and preprocess the data
- Create the network
- Configure the network

- Initialize the weights
- Train the network
- Validate the network
- Implement the network

First, the network is created. The type of the network (feed forward or recursive) is determined with respect to the type of the system. Then, the necessary number of layers is determined. Neuron number of each layer and corresponding activation functions are configured for the level of nonlinearity. The weights are initialized. The randomness of initialization is important. The correlation between weights can affect the results. After the initialization, the training of the network starts. The data is divided into three parts: training, validation, and test data. At the training process, the error in the validation data is also considered, and the process is stopped when the error of validation starts to increase. The overfitting for the training data is thus prevented. The test data are treated as unseen input data and are used for performance measurement. After the training step, the network finds its optimum weights. The network can then estimate outputs for the new unseen inputs.

2.4. Initializing the Neural Network

Initialization of a network (selection of initial weights and biases) is an important and complicated topic. Always, there is a chance to get trapped in a local optimum rather than the global one. A good selection of initial points can bring the output closer to the global optimum.

NN Toolbox has its special random number generators such as RANDS, RANDC, and RANDR. The RANDS generates random numbers uniformly distributed and symmetric around zero. It is equivalently to (2*RAND – 1), where the range of the uniform random number generator RAND function is zero to one. By this change, the mean of the random numbers becomes zero and the range of the random number becomes minus one to plus one. The RANDC is the normalized by columns form of the RANDS function. Similarly, RANDR function is the normalized by rows form of the RANDS function.

The NN Toolbox of MATLAB uses the INITNW function for the initialization of the NN. This function uses the Nguyen–Widrow method, which generates random numbers that have active regions evenly distributed over the input space (Pavelka and Procházka, 2004).

Among various random number generators, the Nguyen–Widrow method is the best one to obtain optimum weights and biases suitable for feedforward NNs. Pavelka and Procházka (2004) makes a comparison between ten different algorithms of initialization weights and biases, and the best results are achieved by Nguyen–Widrow method. Another very efficient method is the RANDS function of NN Toolbox and its results resembles the Nguyen–Widrow method.

The range of the data can be very large and the ranges of the activation functions are given in Section 2.2 above. The data are mapped to activation functions range (minus one to plus one) for better results before training. The ranges of random functions of the NN toolbox (RANDS, RANDC, and RANR) are therefore adjusted to the range minus one to plus one.

2.5. Training the Neural Network

The network is ready to give an output after the initialization. Training needs initial weights in order to start the optimization algorithm. The training algorithm starts with the presentation of inputs to the network. Then, the output is compared with the target data. At this step, mean square error (MSE) is widely used in error calculation.

$$MSE = \frac{1}{2N}\sum_i^N (t_i - z_i(x_i, w_i))^2 \tag{2.12}$$

where, t is the target value of the training data, and z is the NN output. Alternatively, sum of squared error can be used.

$$SSE = \sum_i^N (t_i - z_i(x_i, w_i))^2 \tag{2.13}$$

The MSE equation can be written in explicit form as dependent on inputs and weights. If the NN structure given in Figure 2.11, where the network has two layers including one neuron, and logistic activation function, and one input, and one output, is considered, the equations Equation 2.14 to 2.21 can be written.

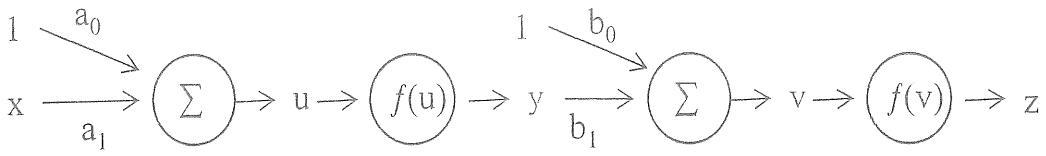


Figure 2.11. One input, one hidden, and one output multilayer NN.

$$u = a_0 + a_1 x \tag{2.14}$$

$$y = f(u) = \frac{1}{1+e^{-u}} \tag{2.15}$$

$$y = \frac{1}{1+e^{-(a_0+a_1 x)}} \tag{2.16}$$

$$v = b_0 + b_1 y \tag{2.17}$$

$$v = b_0 + b_1 \frac{1}{1+e^{-(a_0+a_1 x)}} \tag{2.18}$$

$$z = f(v) = \frac{1}{1+e^{-v}} \tag{2.19}$$

$$z = f(v) = \frac{1}{1+e^{-(b_0+b_1\frac{1}{1+e^{-(a_0+a_1 x)}})}} \tag{2.20}$$

$$MSE = \frac{1}{2N}\sum_i^N (t_i - \frac{1}{1+e^{-(b_0+b_1\frac{1}{1+e^{-(a_0+a_1 x)}})}})^2 \tag{2.21}$$

The objective of the NN training algorithm is to find the optimum NN which shows the system (data) characteristics. The optimum is the minimum value of the above MSE equation at which the network provides a good output for validation set. If the NN output of the validation set gets worse while the MSE decreases, the training stops at this point in order to provide good generalization. The network may further learn the data with lower MSE values, however further training is not desired since the NN memorizes the (overfits) rather than learning it. The aim is to learn the correlation between inputs and outputs from training data and to apply this correlation for estimating outputs. The training algorithm is given by Figure 2.12.

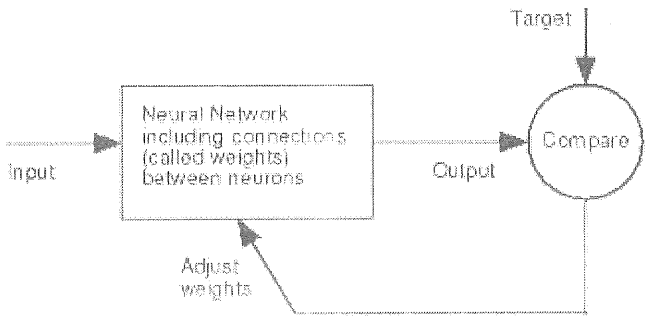


Figure 2.12. Principle of training an Artificial NN (Beale *et al.*, 2010).

The minimum of the MSE is achieved by adjusting the NN weights (including biases) after each representation of training data to the network. The optimum weights can be found by various algorithms, most of which use first order derivatives (such as the back propagation method), or sometimes both first and second order derivatives (such as the Gauss-Newton method). The general weight adjustment expression is:

$$w_{i+1} = w_i + \Delta w \tag{2.22}$$

2.5.1. Levenberg – Marquardt Method

The unconstrained Levenberg-Marquardt (LM) method is the most widely used NN training algorithm. It is a second-order method. Hence, it uses gradient of the error, and also the information about the curvature of the error surface as represented by the Hessian of the error function (MSE). This method uses the speed and initial condition independence of the steepest descent method at the beginning, and the speedy convergence of the Newton method later on towards the end.

Batch learning algorithm is usually preferred in training. In this method, the weight changes are made after the entire data are presented to the network. The error gradient is calculated for the entire data with respect to each weight, and then they are summed. Thus, the gradient is smoothed and averaged for the entire data. (Samarasinghe, 2007)

The sum of error gradients with respect to NN weights for the entire data is given as:

$$d_m = \sum_{n=1}^{N} \left[\frac{\partial E}{\partial w_m}\right]_n \tag{2.23}$$

The weight change expression of LM method is given by:

$$\Delta w_m = -\frac{d_m}{H + e^{-\lambda_I}} \tag{2.24}$$

where, the term **H** in the denominator denotes the second derivative (Hessian) matrix.

The Hessian matrix is defined as:

$$H = \frac{\partial^2 E}{\partial w_i w_j} = \frac{2}{N} \sum_{i=1}^{N} \left\{ \left(-\frac{\partial z_i}{\partial w_i}\right) \left(-\frac{\partial z_i}{\partial w_j}\right) \right\} \tag{2.25}$$

As it can be seen from Equation 2.24, LM method uses both the first and second derivatives in the weight change expression (Δw_m).

2.5.2. Regularization

One of the most important issues in the training is overfitting. Overfitting is the learning of the training data with errors (including noise and outliers), which adds the network more (false) information than the system has. In such overfitted cases, the output estimation of the network is good (or even perfect) for the training data, but not for the test data even though the training and test data all come from the same system. The generalization ability of the network gets worse with overfitting.

One method to increase generalization is early stopping. A validation data set is taken apart from the entire data set. During the training, early stopping method controls the MSE of the validation data together with the MSE of the training data. When the error starts to increase in the validation set, the method stops the training. Therefore, the consistency between two data sets is provided.

Some of the neurons are more active than others, and this can appear in the magnitude of the weights and biases of these neurons. Unused neurons have weights and biases that are close to zero. During the training, weights of the active neurons grow more than others, and this helps the memorization of the training data after some point. The weight and bias parameters of apparently very active neurons can be kept from over-growing by a technique called regularization. Regularization is a method to keep weights small by reducing the flexibility of the network. Sum of squares of weights are added to the error as a penalty term, and thus, the training method decreases the magnitude of the weights together with the training error term. With the regularization term, the objective function becomes:

$$Objective\ Function = MSE + \delta \sum_i^N {w_i}^2 \tag{2.26}$$

The effect of the weights on the objective function is adjusted by the regularization parameter (δ) (Samarasinghe, 2007). The objective function can also be further modified to the following equation where $0 \leq \delta \leq 1$.

$$Objective\ Function = \delta \times MSE + (1 - \delta) \times \frac{1}{N} \sum_i^N {w_i}^2 \tag{2.27}$$

It is hard to find the optimum value of the regularization parameter δ. If this parameter is too large (or δ = 1 in Equation 2.27) the overfitting occurs at training data. If it is too small (or δ = 0 in Equation 2.27), network does not fit to the data, and misfitting occurs. A method that adjusts the regularization parameter automatically is called Bayesian regularization, where the weights and biases of the network are assumed to be random variables with specified distributions, and the regularization parameters are related to the unknown variances associated with these distributions (Beale *et al.*, 2010).

Bayesian regularization can be used by `trainbr` function of the NN Toolbox of MATLAB. This function works best when the data are in the range of minus one to plus one. If the data are not in this range, it can be mapped to the minus one to plus one range by using MATLAB functions `mapminmax` or `mapstd` (Beale *et al.*, 2010).

3. CONSTRAINED NEURAL NETWORK (CNN) ALGORITHMS

The purpose of the thesis is to develop novel methods to implement equality and/or inequality constraints on/between the outputs of a feedforward NN during its training step (during minimization of MSE between the NN outputs and target data).

Most famous toolbox for various types of NNs is the NN Toolbox of MATLAB. It employs several learning algorithms during training step, e.g., the Levenberg-Marquardt (LM) backpropagation, one-step secant backpropagation, and Bayesian regularization backpropagation. The LM method is the preferred one among the others. However, all the algorithms in the NN Toolbox are basically unconstrained optimization (MSE minimization) methods.

In the conventional NN tools, algorithm starts with the creation and configuration of the NN topology. Then, the network is initialized, basically with random numbers (initial weights and biases). The next step is the training of the NN. The objective function of the training is the minimization of MSE or SSE equation between the NN output and target data. The training algorithm, e.g., the LM algorithm (which is also a well known unconstrained optimization algorithm), minimizes this error by changing the weights and biases. The implementation of constraints should be made at this step while the optimization (training) is performed.

The objective of the optimization (training of the NN) is given as:

$$min\ (MSE) = min\ (\frac{1}{2N}\sum_i^N \left(t_i - z_i(x_i, w_i)\right)^2) \tag{3.1}$$

where, t is the target, z is the NN output, and x is the input. Alternatively, regularization can be used in the objective function in order to limit the magnitudes of weights and biases. The regularized objective of the training of the NN is given as:

$$min\ (\delta \times MSE + (1-\delta) \times \frac{1}{N}\sum_i^N {w_i}^2) \tag{3.2}$$

where δ is the regularization parameter as discussed in Section 2.5.2.

There are three novel Constrained Neural Network (CNN) training schemes proposed in this thesis. These CNN schemes minimize the MSE/SSE with/without regularization under equality and/or inequality constraints by implementing the following three different designs that are called CNN-I, CNN-II, and CNN-III, which are illustrated in Figures 3.1 to 3.3.

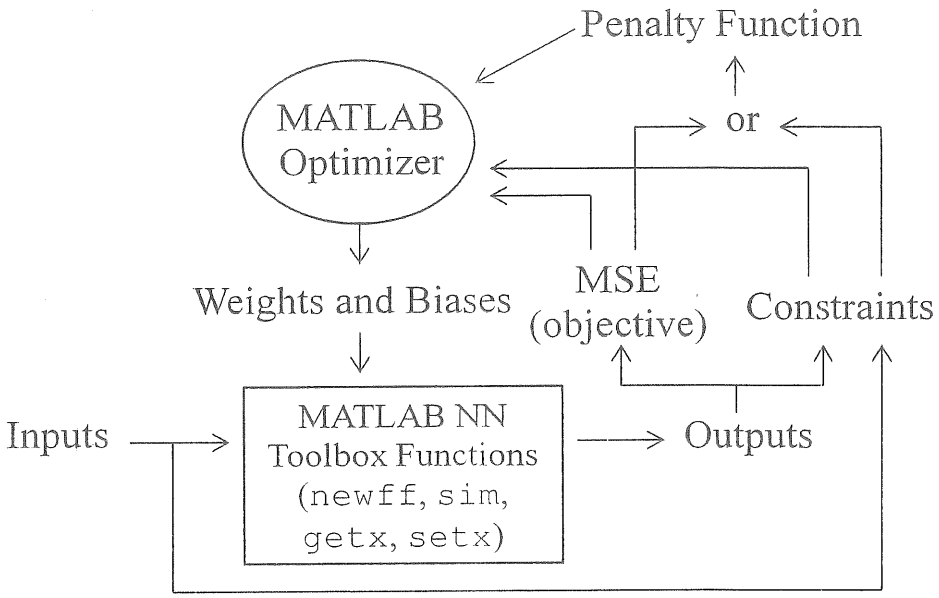


Figure 3.1. Illustration of CNN-I design.

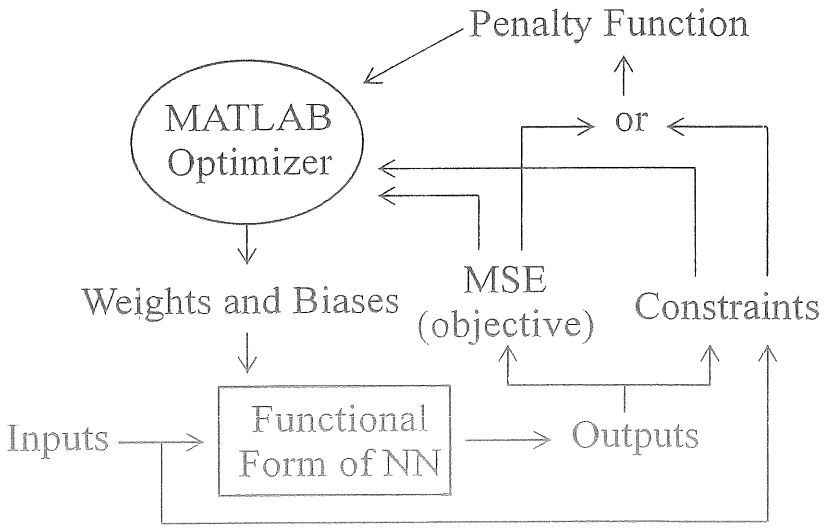


Figure 3.2. Illustration of CNN-II design.

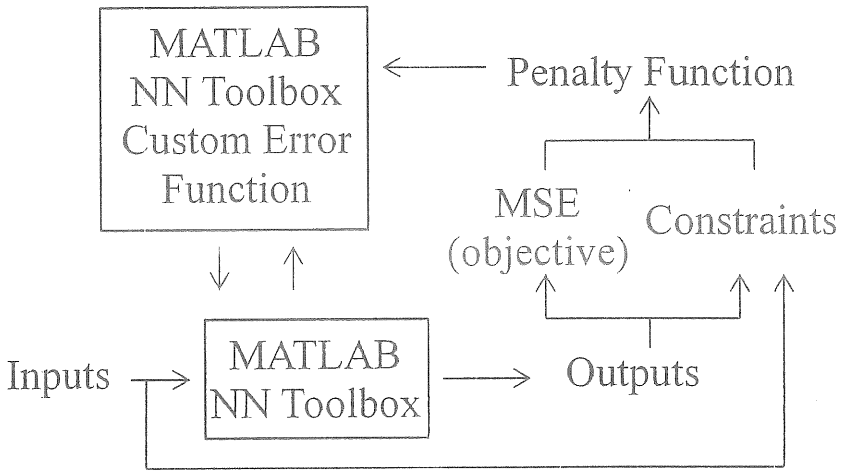


Figure 3.3. Illustration of CNN-III design.

3.1. First Method for Constrained Neural Networks - CNN-I

The purpose of the first constrained learning method is to employ the MATLAB's NN Toolbox functions, particularly the NN building and topology setting functions, as building blocks (as objects) directly without any modification. In this CNN scheme depicted in Figure 3.1, the NN training functions (e.g., the `trainlm` function that implements the LM training method) are not used. Instead, an unconstrained or constrained optimization routine (a generic optimizer, not tailored for NNs), selected from the MATLAB's Optimization Toolbox, is employed for the training of the NN (i.e., for the minimization of the MSE objective function under inequality and/or equality constraints). In other words, the NN building and topology setting functions of the NN Toolbox are employed only to define the NN structure and the MSE computation is done in the objective function routine of the generic optimizer. In this way, alternative constrained optimization functions (e.g., `fmincon`) available in the MATLAB's Optimization Toolbox provide a platform for direct implementation of the equality and/or inequality constraints in the NN training task. On the other hand, for indirect implementation of the equality and/or inequality constraints in the NN training task, the penalty function technique (where constraint violations are included as a penalty term in the augmented objective function) can be employed together with unconstrained optimization functions (e.g., `fminunc`) available in the Optimization Toolbox.

The penalty function provides an ability to use unconstrained optimization tools in the constrained optimization problems, and its generic form can be given as follows:

$$P = MSE(x, w) + \mu(measure\ of constraint\ violations) \tag{3.3}$$

where, μ is a penalty parameter. The measurement of constraint violations may take several different forms and these will be explained in later sections of this chapter.

The regularization term (Equation 2.26) can be added to the objective function (whether pure objective or penalty function form) to limit the growth of weights and biases.

The novel algorithm CNN-I begins with the preprocessing of the data. Preprocessing is necessary and recommended before the upcoming steps, since the performance of NNs gets dramatically better when the data are scaled to the activation function's range. In the preprocessing, first the data are divided into two parts: training and testing (simulation). Then, it is scaled into activation function's range, which is usually between zero and one. In the next step, the NN is created and configured. The number of layers and the number of neurons, the types of activation functions in each layer, and the type of objective function (e.g., MSE or SSE) are all determined. The training of the NN starts in the next step. In CNN-I scheme, the training is made by an unconstrained or constrained optimization method. Optimization method in the training step utilizes the minimization of MSE or SSE between the NN output and target data (with/without a regularization term) as the objective function. In case the equality and/or inequality constraints are handled indirectly via the penalty function technique, both the constraints and the objective function are given within the objective function routine and the augmented objective function involves the constraint violations as well. In case the equality and/or inequality constraints are handled directly, the constraint functions are given to the optimizer from a separate constraint-function routine, which is called by the optimizer along with the objective function routine. The optimizer accordingly (based on objective function improvement and constraints' feasibility gap) modifies the weights and biases (decision variables) during iterations. The iterations of the optimizer continue until a local minimum of the objective function and acceptable constraint violations are reached. Therefore, in the CNN-I scheme, the training

(MSE minimization) and constraint satisfaction are achieved simultaneously by the optimizer.

After the unconstrained or constrained optimizer locates the optimum values of weights and biases (optimization decision variables), they are assigned to the NN. At this point, the NN is ready for testing (simulation) tasks. If the outputs (data) are scaled before the training, the NN outputs are to be unscaled back to the original interval of the raw data.

3.2. Second Method for Constrained Neural Networks - CNN-II

The purpose of the second constrained learning method, depicted in Figure 3.2, is to form the objective function of the NN training and constraint handling without using the MATLAB's NN Toolbox. In order to construct the objective function and to measure constraint violations, the required NN building and NN topology setting tasks are coded explicitly based on these tasks' mathematical representations without employing any of the MATLAB's NN Toolbox functions. Other than the creation and configuration of the NN, the remaining parts of the CNN-II scheme (i.e., interactions of the NN and optimizer) are similar to CNN-I. The constrained or unconstrained optimization routine from the Optimization Toolbox of MATLAB is used for the training of the NN. In this scheme as well, the equality and/or inequality constraints are implemented during NN training either directly (by the constrained optimization functions of the Optimization Toolbox, e.g., `fmincon`) or indirectly (by the unconstrained optimization functions of the Optimization Toolbox, e.g., `fminunc`, using the penalty function technique). Similarly, the regularization term (Equation 2.26) can be added to the objective function with/without penalty term.

The CNN-II algorithm begins with the preprocessing of the data. Then, the NN is created and configured. The training of the NN starts in the next step. The mathematical expressions used are for a generic multilayer feedforward NN and are coded in the custom made functions called by the optimizer during NN training. The feedforward multilayer NN considered in this thesis is depicted in Figure 3.4.

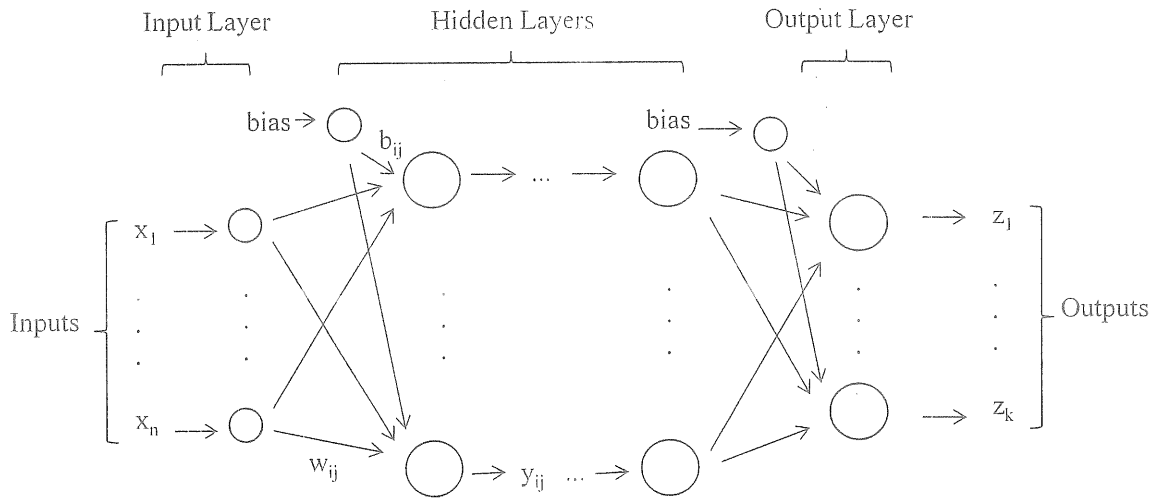


Figure 3.4. Multilayer feedforward Neural Network.

When a generic case of multilayer feedforward NN is considered, there are n inputs, m hidden neurons in several hidden layers, and k output neurons for k number of outputs. The information flow from inputs to outputs starts from the input layer and passes to output layer through several hidden layers. First, the inputs are processed in the first hidden layer. The input (u_j) of the *j*th hidden neuron in the first hidden layer is:

$$u_j = b_{0j} + \sum_{i=1}^{n} w_{ij} x_i \tag{3.4}$$

where x_i is the *i*th input, w_{ij} is the weight assigned to the input i and neuron j, and b_{0j} is the associated bias of hidden neuron j. Generally, hyperbolic tangent sigmoid activation function is preferred in the hidden neurons. Using this activation function the output (y_j) of the *j*th hidden neuron in the first hidden layer is:

$$y_j = f(u_j) = \frac{2}{1+e^{-2u_j}} - 1 \tag{3.5}$$

Equation 3.4 and Equation 3.5 are repeated sequentially for the remaining hidden layers. The output of the previous hidden layer is the input of the following hidden layer. The input and output of the neuron in hidden layers can be given as:

$$u_{ij} = b_{ij} + \sum_{i=1}^{m} w_{ij} y_{ij} \tag{3.6}$$

$$y_{ij} = f(u_{ij}) = \frac{2}{1+e^{-2u_{ij}}} - 1 \tag{3.7}$$

where i states as the number of neuron in the previous hidden layer and j states as the number of neuron in the next hidden layer. When only one hidden layer is considered the following expression is the input of the output layer.

$$v_k = b_{0k} + \sum_{j=1}^{m} w_{jk} y_j \tag{3.8}$$

Usually, linear activation functions are employed in the output layer. If linear activation function is used in the output layer, the outputs of the NN are:

$$z_k = f(v_k) = v_k \tag{3.9}$$

Equation 3.4 to Equation 3.9 form the implicit mathematical expression which establishes the connection between the input and output. In order to avoid presentation of these mathematical expressions in every application of the NN, in this thesis, a novel notation is created which implicitly describes the generation of NN outputs from inputs (Equation 3.4 to Equation 3.9). This new implicit NN notation is described in Figure 3.5.

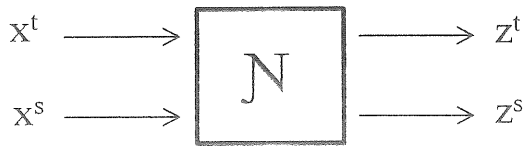


Figure 3.5. New notation of NN structure.

The implicit mathematical expression for the NN is:

$$z = (s, w, x^t, x^s) \tag{3.10}$$

where N stands for a NN. The argument s is the NN specification structure (number of layers, number of neurons, type of activation functions, etc.), w is the weights and biases, x is the input data, and z is the output of the NN. The superscript t represents training portion

of the data and the superscript s represents the simulation (testing) portion of the data. For a fixed (given) set of vector arguments, the NN represented by N fixes the output vector **z**.

The objective of the training is the MSE or SSE between outputs and target data. The MSE and SSE are given as follows where z is a function of N:

$$MSE = \frac{1}{2N}\sum_{i=1}^{k}(t_i - z_i)^2 \tag{3.11}$$

$$SSE = \sum_{i=1}^{k}(t_i - z_i)^2 \tag{3.12}$$

The iterations of the optimizer continue until a local minimum of the objective function is reached. The optimum values of weights and biases are assigned to NN and the NN is ready for simulation.

3.3. Third Method for Constrained Neural Networks - CNN-III

The purpose of the third scheme, depicted in Figure 3.3, is to use only the MATLAB's NN Toolbox training functions (i.e., MSE minimization algorithms such as `trainlm` which is tailored to NNs) and at the same time to handle equality and or inequality constraints, without using an external generic constrained optimizer. This obviously requires the employment of the Penalty Function technique and modification of the NN Toolbox functions in order to add the penalty term to the location where the MSE and its partial derivatives with respect to weights and biases are computed by the NN Toolbox without changing the other parts of algorithms of the NN Toolbox. The NN Toolbox admits a custom error function other than MSE only for some of its training functions. Therefore scheme CNN-III requires only the development and employment of the custom objective function such that it includes the constraint violations as a penalty as shown in Equation 3.3.

As mentioned before, the LM algorithm is a very powerful and fast method in the training of unconstrained NNs. The CNN-III scheme was first conceived to capitalize on the speed of the LM algorithm by using it directly with a simple addition of a penalty term for handling the constraints. However, it was recognized that MATLAB does not provide

means for developing custom error function for `trainlm` which invokes the LM method tailored for NNs. The LM algorithm, by default performs only with MSE or SSE as the objective function. Unfortunately, compared to `trainlm`, the other training functions of NN Toolbox do not have attractive performances. Furthermore, it was also recognized that a simple modification of a custom error function would not be sufficient and that customization of the gradient (with respect to weights and biases) computation function of the NN Toolbox was also necessary. Since CNN-I and CNN-II demonstrated the expected performance very well and served the main purpose of the thesis, CNN-III scheme was just presented as an idea without any implementation which may be of interest in a future work.

3.4. Some Application Details of CNN-I

As mentioned in Section 3.1, particularly the NN building and topology setting functions of the NN Toolbox of MATLAB are employed in CNN-I scheme. These employed functions of the NN Toolbox are listed in Table 3.1.

Table 3.1. NN Toolbox functions employed in CNN-I.

`newff`
`sim`
`getx`
`setx`

The `net = newff(•)` statement calls the `newff` function with input arguments denoted by `(•)` creates a feedforward backpropagation NN the structure of which is stored in the structured MATLAB array `net`. The list of the input arguments, `(•)`, can be found in the user guide of MATLAB's NN Toolbox (Mathworks, 2011). The input arguments of the `newff` function used in CNN-I are only the input and output data arrays, and the sizes of the hidden layers.

The `sim(net,(•))` function of the NN Toolbox simulates the NN topology of which is present in the structured array `net`. In CNN-I, only the input data array is used as the argument `(•)`. The output of the `sim` function is the NN output (prediction) array.

The `x = getx(net)` function of the NN Toolbox extracts the NN weights and biases as a single vector **x**. Since, no built-in training of the NN is employed; the extracted weights and biases are only the initialized values. By default, MATLAB's NN Toolbox uses Nguyen-Widrow initialization method (Mathworks, 2011). Thus, in CNN-I, the well-established weight and bias initialization algorithm is used just to provide good initial guesses for the optimizer since the weights and biases are the only decision variables, **x**, in the optimization problem.

The MATLAB's NN Toolbox arranges the weights and biases in the NN layers into a single-column vector as shown by Figure 3.6.

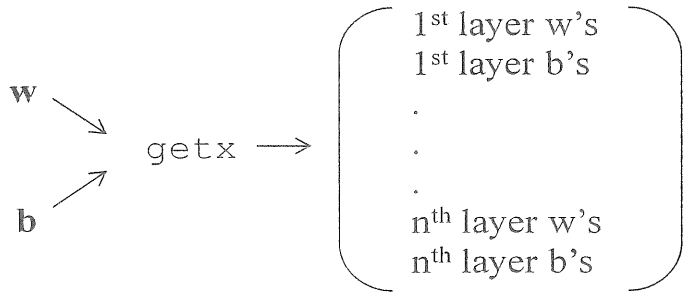


Figure 3.6. Output order of the `getx` function.

The `net = setx(net,x)` function of the NN Toolbox assigns the weight and bias values from a column vector, **x**, to the NN structure defined in the structured array `net`. Thus, the values in the decision variable vector received from the optimizer, **x**, is properly distributed to the values of the weights and biases in accordance with the network structure hidden in the array `net`.

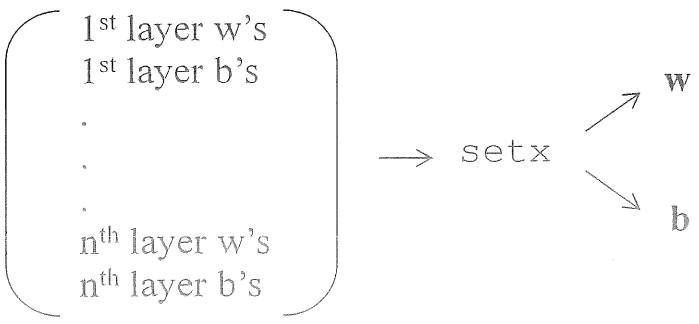


Figure 3.7. Input order of the `setx` function.

The MATLAB's NN Toolbox arranges the elements of the column vector into weights and biases in the NN layers as shown by Figure 3.7.

The first and the last steps of the CNN-I and CNN-II schemes, which include preprocessing, creation and configuration of the NN topology, and postprocessing, are identical to each other. The CNN-I application starts with the preprocessing of the data. First, the data is divided into training and simulation sets by employing two alternatives. One of them is the `crossvalind` function which randomly mixes and divides the data into two sets. The other alternative is the simple non-random separation, where first portion of the data are taken as training set, and remaining part is taken as simulation set. Then, the scaling of data to the activation functions range is performed by two alternative functions; `mapminmax`, and `mapstd`. The NN Toolbox uses the `mapminmax` function for scaling as a default, and it is recommended to use. The preprocessing step is depicted in Figure 3.8.

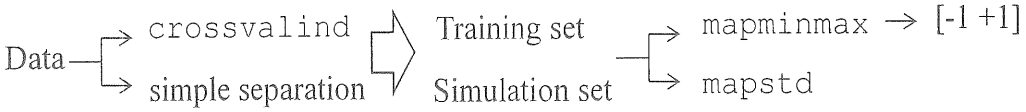


Figure 3.8. Data preprocessing.

Next step in the CNN-I algorithm is the creation and configuration of the NN where the employment of NN Toolbox begins. Network is created by `newff` function. Number of layers, number of neurons and type of the activation functions are all determined by this function. The NN is initialized by the NN Toolbox automatically which employs the `initnw` function. The `initnw` function uses the Nguyen-Widrow (NW) initialization algorithm if the required conditions are satisfied, if not (e.g., NW cannot be used with linear activation function), it employs the `rands` function. Initialized weights and biases are assigned to a vector by the `getx` function in order to use the initialized weights and biases as a judicious initial guess in the optimizer. The `getx` starts the assignment of the weights and biases from the first layer and continues sequentially until the last layer. First it assigns the weights, and then the biases into the vector for each layer.

CNN-I scheme continues with the training of the NN. As described in Section 3.1, constrained NN training can be made either directly (by implementation of the constraints explicitly) or indirectly (by implementation of the penalty function technique). When a direct implementation of constraints is desired, the constrained optimizers (e.g., `fmincon`, `ga`, and `patternsearch`) are employed. If an indirect constraint implementation is desired, unconstrained optimizers are employed (e.g., `fminunc`, `simulannealbnd`, and `fminsearch`). Both of the constrained or unconstrained optimizers need a custom made objective function. Also, constrained optimizers (e.g., `fmincon`) may need a custom made constraint function.

The custom objective function takes the weights and biases as the input. First, the current values of the weights and biases sent by the optimizer are assigned to the NN by the `setx` function. Then, the output of the NN is obtained by the `sim` function. The objective (MSE/SSE) is computed between the outputs and the target data. Finally, regularization term can be added to the objective (MSE/SSE). In the indirect constraint implementation method (the penalty function technique), constraint violations are also calculated and added to the objective.

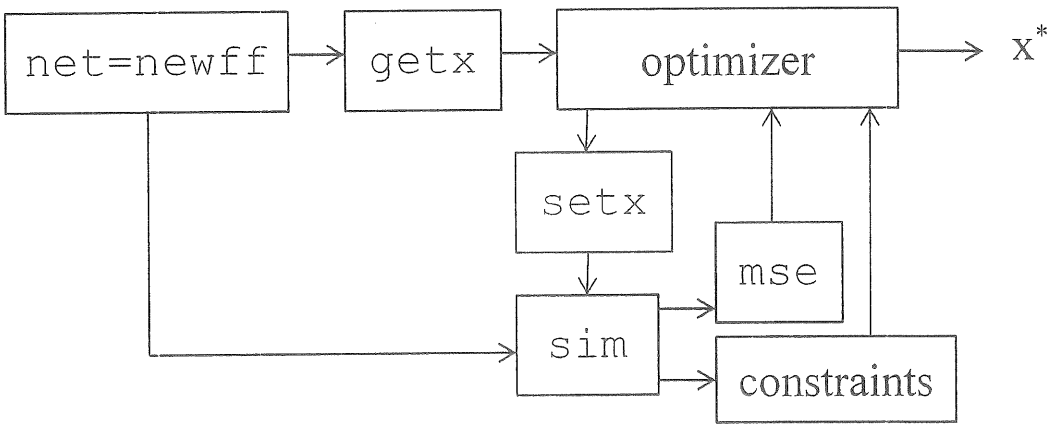


Figure 3.9. Optimizer interface of the CNN-I scheme.

Some of the constrained optimizers (e.g., `fmincon`) need a constraint function for direct implementation of constraints. The constraint function takes the weights and biases as the input, and first, it assigns them to the NN by the `setx` function. Then, the outputs of the NN are calculated by the `sim` function. If the data are scaled at preprocessing, the

scaling of the NN outputs and the input data need to be reversed before the calculation of the constraints.

The optimum weights and biases are assigned to the NN after the training step by the `setx` function. The NN is now ready for simulation with optimum weights and biases obtained in the training step. If the data are scaled at preprocessing, the outputs need to be unscaled.

3.5. Some Application Details of CNN-II

In the CNN-II algorithm, none of the NN Toolbox functions are employed. All of the necessary functions for creation and configuration of the NN are coded explicitly. The NN block (Figure 3.5) which establishes the connection between inputs and outputs is formed with custom made functions. These functions, given the NN topology (**s**) and weights and biases, (**w**) map the given inputs (**x**) to outputs (**z**), as expressed implicitly by Equation 3.13.

$$z = (s, w, x) \tag{3.13}$$

The CNN-II algorithm is similar to the CNN-I. It starts with the preprocessing of the data. Data are divided into training and simulation sets by `crossvalind` function or simple separation. Then, the data are scaled into activation functions range by `mapminmax`, or `mapstd`.

The algorithm continues with the creation and configuration of the NN. The number of layers, the number of neurons in the layers, and the type of activation functions are all determined by MATLAB's structured arrays. Then, the NN is initialized by the custom made initialization function. There are three alternative methods used for generating initial weights and biases. The first alternative uses the `rand` function of MATLAB. Second alternative employs the `rands` function of the NN Toolbox. Third alternative uses the Nguyen-Widrow initialization algorithm. Third alternative utilizes the `calcnw` function of the NN Toolbox, and the `rands` function in case the conditions do not satisfy the NW requirements.

After the initial weights and biases are obtained, they need to be assigned in a vector form to feed them as an initial guess to the optimizer. Custom made weights and biases extraction function is employed for assigning weights and biases from the NN to a vector, and this function performs just like the `getx` function of NN Toolbox as depicted in Figure 3.6. Its output is the same as the output of the `getx` function.

At the next step, the training of the NN is accomplished. The training procedure employed are the same as CNN-I. Constraint implementation can be made directly or indirectly (penalty function technique) by constrained optimization functions (e.g., `fmincon`, `ga`, and `patternsearch`), and unconstrained optimization functions (e.g., `fminunc`, `simulannealbnd`, and `fminsearch`), respectively. All the optimizers need custom objective function. As in CNN-I, the objective function takes weights and biases as an input. First, the output of the network is calculated by the custom made NN simulation function. Then, the SSE or MSE are calculated between outputs and targets. Finally, the regularization term can be added to the objective. In the indirect constraint implementation method (penalty function technique), constraint violations are calculated and added to the objective. The constrained NN training by CNN-II scheme is depicted in Figure 3.2, and constrained NN training with the given NN block in Equation 3.10 is depicted in Figure 3.10.

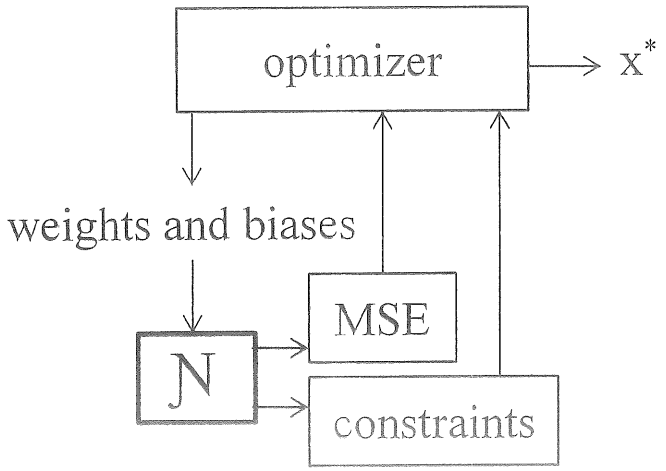


Figure 3.10. CNN-II training phase.

The custom made NN simulation function takes weights, biases and training data as the inputs. Weights and biases (decision variables) come from the optimizer in a vector form, and they are distributed to the NN by custom made weights and biases setting function as depicted in Figure 3.7. The inputs and outputs of the function are the same and in the same order as the `setx` function of the NN Toolbox. After the assignment of the weights and biases to the NN, the calculation of the outputs is accomplished as described implicitly by Equation 3.13.

In the direct implementation of constraints, as in CNN-I, the constraint function is employed. In the constraint function, first, the outputs are obtained by the custom-made simulation function with the current weights, biases (coming from the optimizer's constraint interface) and the training data.

It must be noted that even though a generic feedforward NN does not use the input data directly in computation of the objective (MSE) (i.e., the input data do not bypass the NN), the CNN may require the input data directly in evaluation of the constraint functions (input data enter as well as bypass the NN to enter the constraint functions).

If the data set is scaled at the preprocessing step, the output and input data are to be unscaled before computation of the constraint functions. The computation of the constraint functions is equivalent to the calculation of the constraint violations.

The optimum weights and biases are assigned to the NN by weights and biases setting function after the constrained or unconstrained optimizer locates the optimum weights and biases that minimize the objective (MSE/SSE) and constraint violations. At this point, the NN is ready for simulation. The outputs of the NN need to be unscaled, if the data are scaled at the preprocessing step.

3.6. Some Details on Optimization and Constraint Handling

The optimizers employed in the CNN-I and CNN-II are categorized as constrained optimization functions (e.g., `fmincon`, `ga`, and `patternsearch`), and unconstrained optimization functions (e.g., `fminunc`, `simulannealbnd`, and `fminsearch`).

Among these functions, `fmincon` and `fminunc` are the most powerful functions that are gradient based and that are mostly used in the thesis. Some of the option settings are the same for all optimizers and they are assigned by the `optimset` command. Each problem has its own characteristics. These options should thus be adjusted considering the characteristics of the optimization problem.

The function `fmincon` finds the minimum of constrained nonlinear multivariable function and has four alternative algorithms (Coleman *et al.*, 2007):

- *Interior-Point:* The algorithm computes constraints at all iterations. It is a large-scale algorithm.
- *SQP:* The algorithm computes constraints at all iterations. It is a medium-scale algorithm.
- *Active-Set:* The algorithm may violate the constraints during iterations. It is a medium-scale algorithm.
- *Trust-Region-Reflective:* The algorithm satisfies the constraints at all iterations. It is a large-scale algorithm. In order to use this algorithm in large-scale format, the gradients of objective function and constraints must be provided.

The SQP algorithm is the mostly employed option in the thesis. It is slower than Interior-Point but it generally performs better than the others and also satisfies all the constraints at all iterations. When the problem is infeasible, then `fmincon` attempts to minimize the maximum constraint violation.

3.6.1. Penalty Function Technique

The idea of the penalty function method is to transform a constrained optimization problem into an unconstrained optimization problem. A constrained optimization problem is given as:

$$min_x f(x) \tag{3.14}$$

$$s.t.\ h(x) = 0 \tag{3.15}$$

$$g(x) \leq 0 \tag{3.16}$$

In the penalty function method, the equality and inequality constraints are added to the objective function as a penalty term only when they are violated. This can be achieved with the following penalty function form:

$$min_x\, P(f(x), \boldsymbol{h}(x), \boldsymbol{g}(x)) \tag{3.17}$$

where,

$$P(f, \boldsymbol{h}, \boldsymbol{g}) = f(x) + \mu \| \boldsymbol{h}(x) \| + \mu \sum_{k=1}^{m} max\,(0, g_k(x)) \tag{3.18}$$

3.6.2. Constraint Aggregation

In CNN-I and CNN-II, constraints functions depend on the data values (either input data or NN target values). Therefore all of the constraints are to be evaluated for each data point at every iteration of the optimizer as exposed by Equation 3.19 and Equation 3.20.

$$h_i((\boldsymbol{s}, \boldsymbol{w}, \boldsymbol{x}^t), \boldsymbol{x}^t) = 0 \quad i = 1, \ldots, m \tag{3.19}$$

$$g_j((\boldsymbol{s}, \boldsymbol{w}, \boldsymbol{x}^t), \boldsymbol{x}^t) \leq 0 \quad j = 1, \ldots, k \tag{3.20}$$

In consequence, for large problems with many data points, there can be enormous number of constraint evaluations in each iteration of the optimizer. When m equality constraints and k inequality constraints are considered for n data points (rows), there are $(m \times n + k \times n)$ number of constraint evaluations for each call of the constraint function by the optimization algorithm during the training phase. The number of constraint evaluations can be decreased by the use of a constraint aggregation technique (Akman and Okay, 2003). A general optimization problem is given as:

$$min_x\, f(\boldsymbol{x}) \tag{3.21}$$

$$s.t.\ \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0} \tag{3.22}$$

$$\boldsymbol{g}(\boldsymbol{x}) \leq \boldsymbol{0} \tag{3.23}$$

$$\boldsymbol{x}^L \leq \boldsymbol{x} \leq \boldsymbol{x}^U \tag{3.24}$$

where, $\mathbf{x}$ is the vector of N continuous decision variables between lower ($\mathbf{x}^L$) and upper bounds ($\mathbf{x}^U$). f is the scalar objective function. $\mathbf{h}$ is the vector of N equality constraints, and $\mathbf{g}$ is the vector of N inequality constraints.

The equality constraints can be handled with the following modification:

$$h(x) \leq 0, \ -h(x) \leq 0 \tag{3.25}$$

The side constraints (bound), if required, can be reworked as well:

$$(-x + x^U) \leq 0, \ and \ (x - x^U) \leq \ 0 \tag{3.26}$$

With these modifications, the entire set of equality constraints and side constraints, together with the original inequality constraints, can be lumped into an augmented vector of inequality constraints, $\mathbf{C(x)} \leq \mathbf{0}$, and the optimization problem defined by Equation 3.21 to Equation 3.24 can be written as:

$$min_x \, f(x) \tag{3.27}$$

$$s.t. \ \ C(x) \leq 0 \tag{3.28}$$

Constraint aggregation technique lumps the constraints defined by Equation 3.28 into a single and equivalent constraint.

Kreisselmeier-Steinhauser (KS) function is an attractive aggregation function which can be used for large-scale difficult optimization problems. The KS function that underestimates the feasible region formed by inequality constraints is given by Equation 3.29 and the KS function that overestimates the feasible region is given in Equation 3.30.

$$KS(x, \rho) = \frac{1}{\rho} ln\left(\sum_{j=1}^{N} exp\ (\rho C_j(x))\right) \leq 0 \tag{3.29}$$

$$KS(x, \rho) = \frac{1}{\rho} ln\left(\frac{1}{N}\sum_{j=1}^{N} exp\ (\rho C_j(x))\right) \leq 0 \tag{3.30}$$

where, the parameter ρ is an approximation parameter for the KS function with the range $0 < \rho < \infty$. With this exponential aggregation, the optimization problem defined by Equation 3.27 and Equation 3.28 can be written as:

$$min_x f(x) \tag{3.31}$$

$$s.t.\ \ KS(x, \rho) \leq 0 \tag{3.32}$$

where, this constraint-aggregated optimization problem has only single inequality constraint. Thus, the optimization problem can be solved with one aggregated constraint. Therefore, the computational load in constraint handling can be dramatically decreased. (Akman and Okay, 2003)

3.7. Notation

As described in the previous chapters, the exact mathematical definition of the CNN problems may be very convoluted. A common definition needs to be formed in order to abbreviate the problem definition. Hence, a NN block is formed in Section 3.2 which is described in Figure 3.11.



Figure 3.11. Implicit representation of the block of NN functions.

where, x is input data (training or testing), and z is output (training or testing) of the NN. The symbol N implicitly describes the NN structure. The implicit mathematical expression of the NN block is given as:

$$z = (s, w, x) \tag{3.33}$$

where N stands for the NN. The argument s is the NN specification structure (number of layers, number of neurons, type of activation functions, etc.), **w** is the weights and biases, **x** is the input data, and **z** is the output of the NN.

The training of a NN can be described as an optimization problem. The notation for the NN block is very functional in expressing the optimization problem, since the explicit

form of objective and constraints may be very protracted. Therefore, the training of a NN, with, for instance, the SSE objective can be described as:

$$\min_{w} f(w) = SSE(\ (s, w, x^t), y^t) \tag{3.34}$$

$$s.t.\ h((s, w, x^t), x^t) = 0 \tag{3.35}$$

$$g((s, w, x^t), x^t) \leq 0 \tag{3.36}$$

where, superscript t means training data. The implementation of constraints can be made indirectly by the penalty function technique in the NN training task. The optimization problem form of the training of a NN using penalty function can be described as:

$$\min_{w} f(w) = SSE(\ (s, w, x^t), y^t) + \mu \|h((s, w, x^t), x^t)\| + \mu \sum_k max(0, g_k((s, w, x^t), x^t)) \tag{3.37}$$

The regularization term can be added to the objective in the direct implementation of the constraints as:

$$\min_{w} f(w) = SSE(\ (s, w, x^t), y^t)\ + \delta \|w\| \tag{3.38}$$

$$s.t.\ h((s, w, x^t), x^t) = 0 \tag{3.39}$$

$$g((s, w, x^t), x^t) \leq 0 \tag{3.40}$$

The training of a NN with constraints using penalty function and regularization can be described as:

$$\min_{w} f(w) = SSE(\ (s, w, x^t), y^t) + \mu \|h((s, w, x^t), x^t)\| + \mu \sum_k max(0, g_k((s, w, x^t), x^t)) + \delta \|w\| \tag{3.41}$$

4. APPLICATIONS OF CONSTRAINED NEURAL NETWORKS

In this chapter, CNN-I and CNN-II methods, which are elaborated in Chapter 3, are applied to several problems in chemical engineering and other fields. In all of the examples, first the problem definition is given and generation of the data is described. Then, problem is transformed to a NN problem and the NN is trained by the CNN-I and CNN-II methods with/without penalty function technique. For comparison, the NN is also trained by MATLAB's NN Toolbox using LM method.

Throughout the thesis CNN-I-D will be used to indicate the application of CNN-I with direct implementation of the constraints (via `fmincon`), CNN-I-P will be used to indicate the application of CNN-I with implementation of the constraints via the penalty function methods (via `fminunc`), CNN-II-D will be used to indicate the application of CNN-II with direct constraint implementation (via `fmincon`), and CNN-II-P will be used to indicate the application of CNN-II with indirect constraint implementation by the penalty function methods (via `fminunc`).

4.1. Synthetic Time Series

In this section two different sets of synthetic time series are considered. These time series come from different literatures that concentrate on the theory and application of the causality in binary or multivariable time series.

4.1.1. Synthetic Linear Time Series

The first model of synthetic series (Ding *et al.*, 2006) consists of three time series which has causal influence from Y_t to X_t that is indirect and mediated by Z_t. The model is given as:

$$X_t = 0.8X_{t-1} - 0.5X_{t-2} + 0.4Z_{t-1} + \epsilon_t \tag{4.1}$$

$$Y_t = 0.9Y_{t-1} - 0.8Y_{t-2} + \xi_t \tag{4.2}$$

$$Z_t = 0.5Z_{t-1} - 0.2Z_{t-2} + 0.5Y_{t-1} + \eta_t \tag{4.3}$$

where, $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ are time series. ϵ, ξ, and η are Gaussian white noises. Subscript t denotes the time step. When the right hand sides of the equations are considered, there are six variables, which are X_{t-1}, X_{t-2}, Y_{t-1}, Y_{t-2}, Z_{t-1}, and Z_{t-2}. The three variables X_t, Y_t, and Z_t depend on these six variables. Thus, X_t, Y_t, and Z_t are taken as outputs of the NN, and X_{t-1}, X_{t-2}, Y_{t-1}, Y_{t-2}, Z_{t-1}, and Z_{t-2} are taken as inputs of the NN. The NN problem is depicted in Figure 4.1.

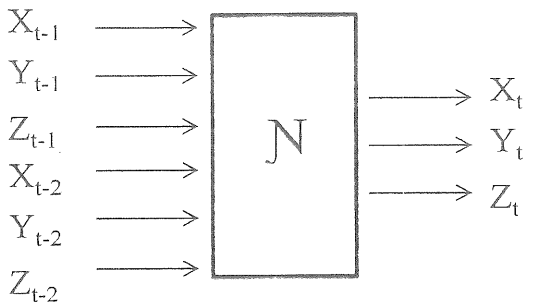


Figure 4.1. NN model of Synthetic Linear Time Series.

The problem can be described by the notation given in the Section 3.7.

$$z = (s, w, x) \tag{4.4}$$

where,

$$z = [X_t \quad Y_t \quad Z_t] \tag{4.5}$$

is the output of the NN which have components X_t, Y_t, and Z_t.

The constraint of the problem is defined as:

$$X \geq -0.1 \tag{4.6}$$

However, the inequality constraints should be in the following form:

$$g((s, w, x), x) \leq 0 \tag{4.7}$$

Thus, the inequality constraint can be described as:

$$g = -X - 0.1 \leq 0 \tag{4.8}$$

The training of the NN with regularization and direct implementation of constraints can be given as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \delta\|w\| \tag{4.9}$$

$$s.t.\ g \leq 0 \tag{4.10}$$

This optimization problem of NN training was performed by CNN-I and CNN-II, with/without penalty function technique (CNN-I-P/CNN-I-D and CNN-II-P/CNN-II-D). The NN structure consists of six inputs meaning six input layers, 12 neurons in the single hidden layer, where all neurons have hyperbolic tangent sigmoid functions, and three neurons in the output layer all having linear activation functions. The total number of weights and biases are 108 and 15, respectively. These make the number of decision variables in the optimization problem 123.

The data set was generated from Equation 4.1 to Equation 4.3 between $0 \leq t \leq 100$. Thus, each one of the series has 96 data points after removing out the time delay points. 60% of the data (58 data points) were used in the training step, and the remaining 40% (38 data points) were used in the testing step. The training options for the direct implementation of constraints, g, by CNN-I-D and CNN-II-D, and for the unconstrained NN Toolbox are the optimizer selection (`fmincon` for CNN-I-D and CNN-II-D, `trainbr` for NN Toolbox), and regularization parameter (δ = 0.0001). Regularization was used to avoid overlearning. As the error function, SSE was used since `trainbr` performs only with SSE.

For CNN-I-D and CNN-II-D, initial guesses for the decision variables (NN weights and biases, shown as w) were obtained from the custom made initialization function as

using NW initialization method that was explained in Section 3.5. With 58 data points, the total number of inequality constraints (Equation 4.6) imposed on X series becomes 58. Therefore, the optimization problem given by Equation 4.9 above has 123 decision variables, 58 inequality constraints in the form Equation 4.10.

The maximum number of epochs allowed for the `trainbr` of the NN Toolbox was set to 1000. Other optional parameters of the `trainbr` of the NN Toolbox were set such that the training will not usually stop before maximum epochs allowed (e.g., min gradient = 1×10^{-12}, max mu = 1×10^{10}). For CNN-I and CNN-II, the optimization termination criteria were set as TolX = 1×10^{-6}, TolFun = 1×10^{-9}, TolCon = 1×10^{-2}, and MaxIter = 100.

The outputs of CNN-I-D (by direct constraint implementation) and NN-Toolbox are given in Figure 4.2 and Figure 4.3. The horizontal line shows the constraint limit and vertical line separates the training (left) and simulation/testing (right) parts.

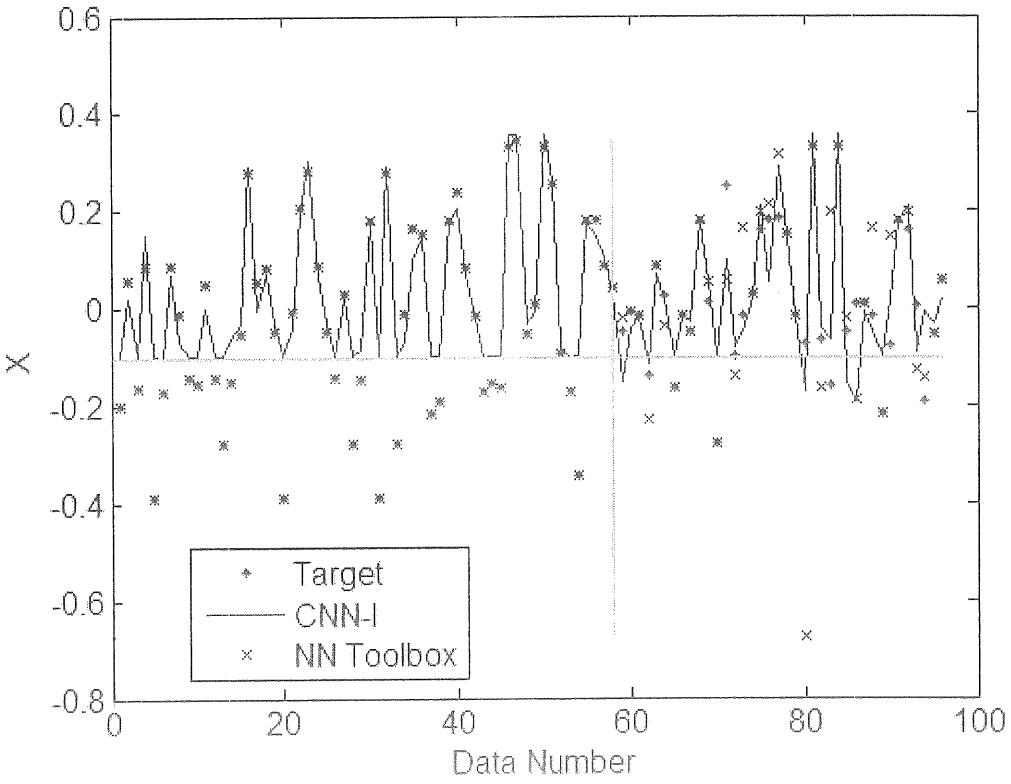


Figure 4.2. Output **X** of CNN-I-D for the Synthetic Linear Time Series.

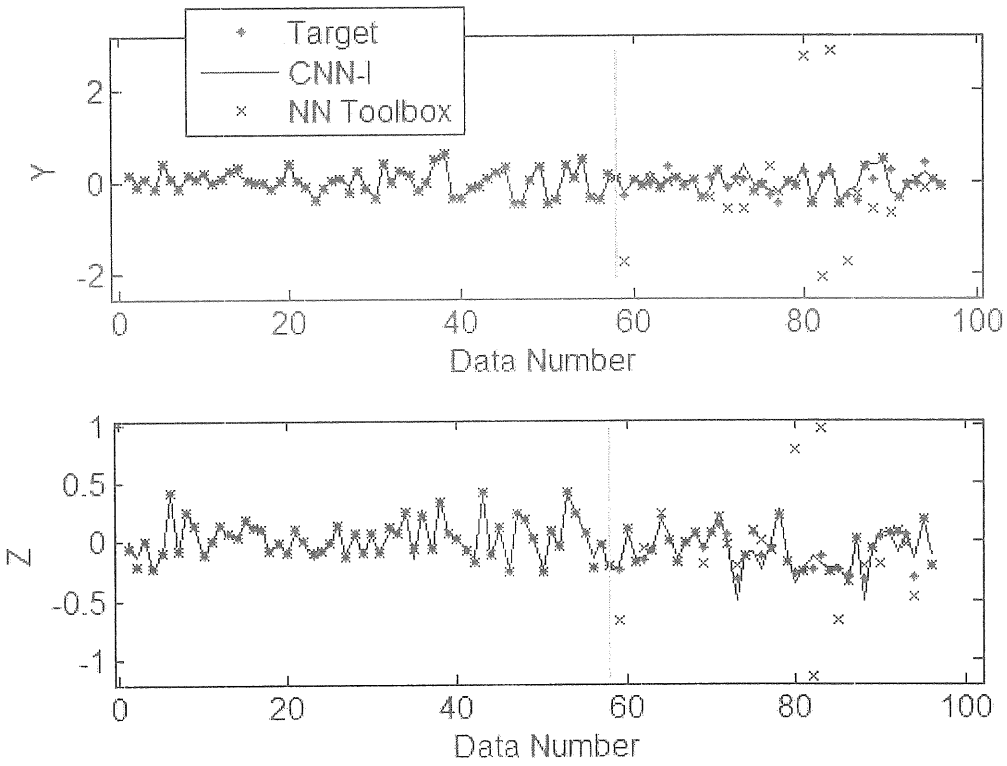


Figure 4.3. Outputs **Y** and **Z** of CNN-I-D for the Synthetic Linear Time Series.

The regression coefficients (R^2) of the NN Toolbox are one for all outputs X, Y, and Z in the training set, respectively and 0.70, 0.36, and 0.45 for outputs X, Y, and Z in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.62. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set well but its predictions are not so well in the testing set. On the other hand, the regression coefficients of CNN-I-D are 0.92, 0.98, and 0.99 for outputs X, Y, and Z in the training set, respectively, and 0.83, 0.76, and 0.90 for the outputs X, Y, and Z in the testing set, respectively. The overall regression coefficient of CNN-I-D considering all data is 0.90. This shows that CNN-I-D performs better than the NN Toolbox especially in the testing set even under inequality constraint. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The constraint violation norm, $\|-\mathbf{X}-0.1\|$, is 0.14 for CNN-I-D and 0.94 for the NN Toolbox.

The outputs of CNN-II-D (by direct constraint implementation) and NN-Toolbox are given in Figure 4.4 and Figure 4.5. The horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

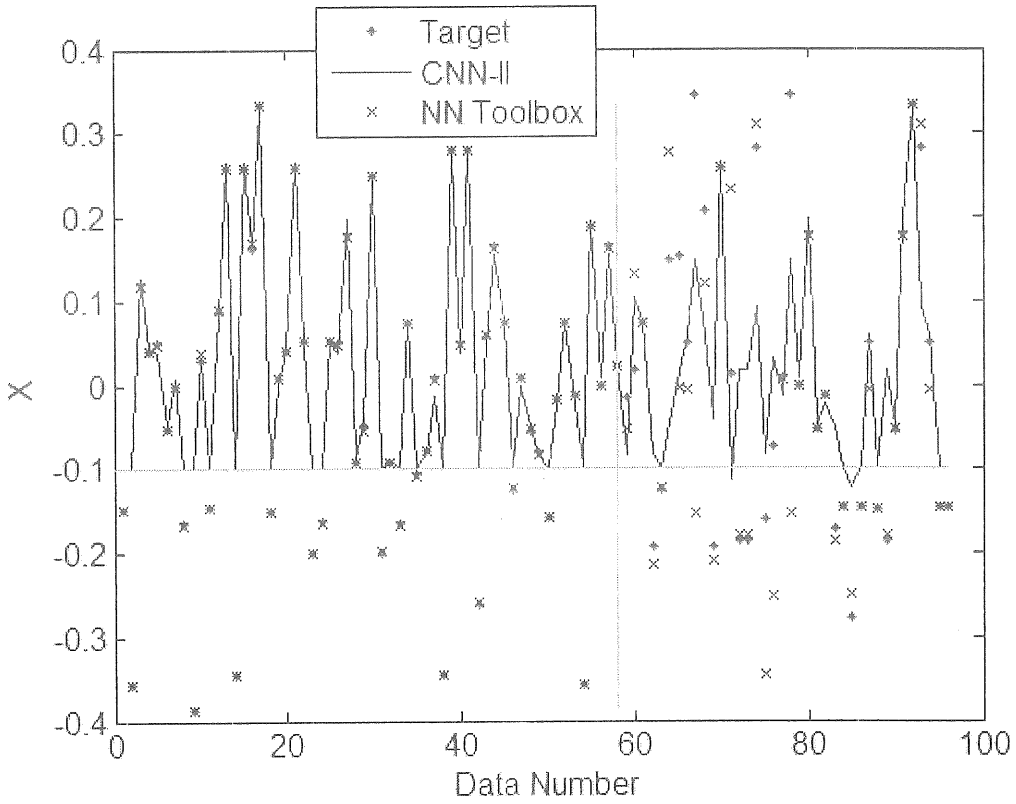


Figure 4.4. Output **X** of CNN-II-D for the Synthetic Linear Time Series.

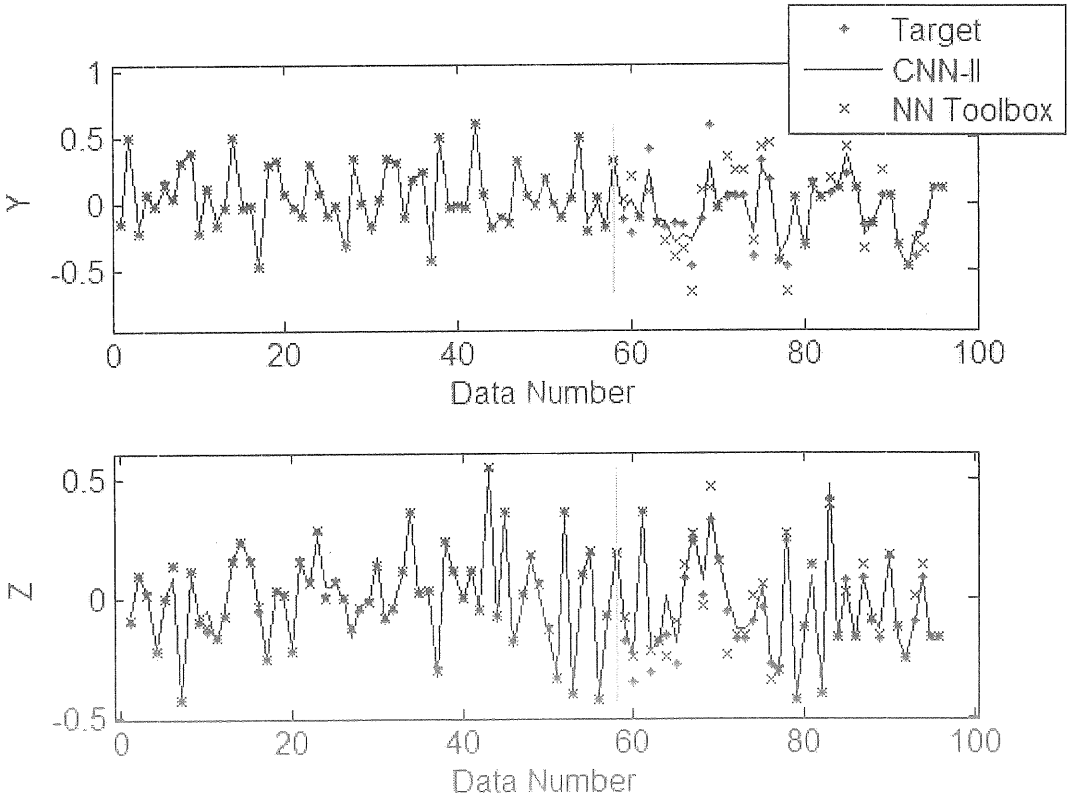


Figure 4.5. Outputs **Y** and **Z** of CNN-II-D for the Synthetic Linear Time Series.

The regression coefficients (R^2) of the NN Toolbox are 0.99, 0.99, and 0.99 for outputs X, Y, and Z in the training set, respectively and 0.73, 0.80, and 0.95 for outputs X, Y, and Z in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.92. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set well but its predictions are not so well in the testing set. On the other hand, the regression coefficients of CNN-II-D are 0.92, 0.99, and 0.99 for outputs X, Y, and Z in the training set, respectively, and 0.78, 0.92, and 0.97 for the outputs X, Y, and Z in the testing set, respectively. The overall regression coefficient of CNN-II-D considering all data is 0.95. This shows that CNN-II-D performs better than the NN Toolbox especially in the testing set even under inequality constraint. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The constraint violation norm, $\|-\mathbf{X}-0.1\|$, is 0.026 for CNN-II-D and 0.76 for the NN Toolbox.

In the NN literature, it is customary to provide a predicted versus data graph for all NN outputs, annotated with the y=x line and its regression coefficient (R^2) value. However, in this thesis, no such graphs are presented since the presence of inequality or equality constraints makes these graphs pointless. In order to present this inconsistency, the regression plots are given only for the direct constraint implementation case of CNN-II-D.

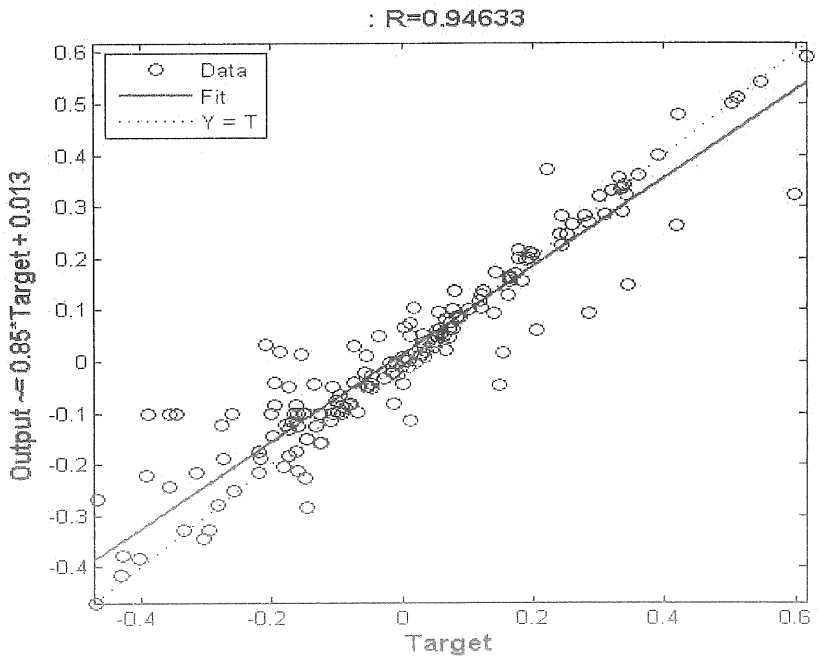


Figure 4.6. Regression plot of the complete data set for CNN-II-D case.

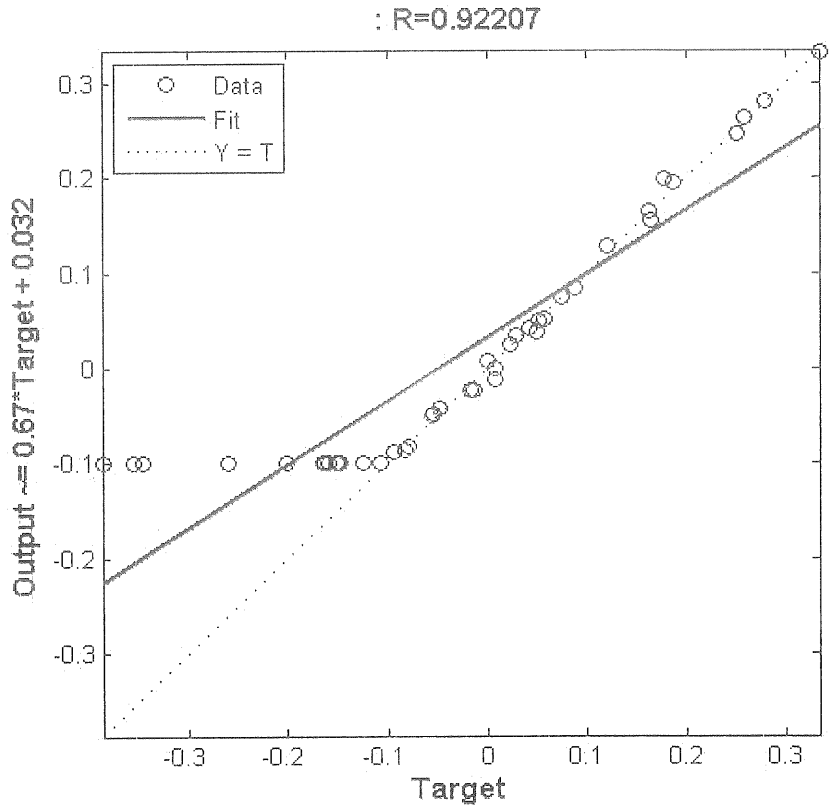


Figure 4.7. Regression plot of output X in CNN-II-D case for training data (effect of the inequality constraint $\mathbf{X} \geq -\mathbf{0.1}$ on the NN outputs is clearly visible).

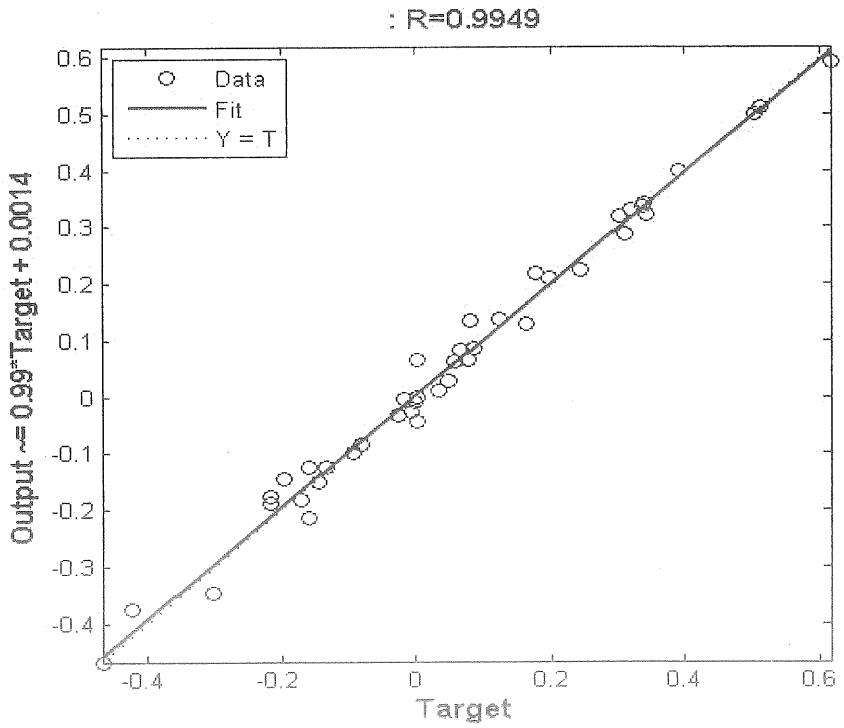


Figure 4.8. Regression plot of output Y in CNN-II-D case for training data.

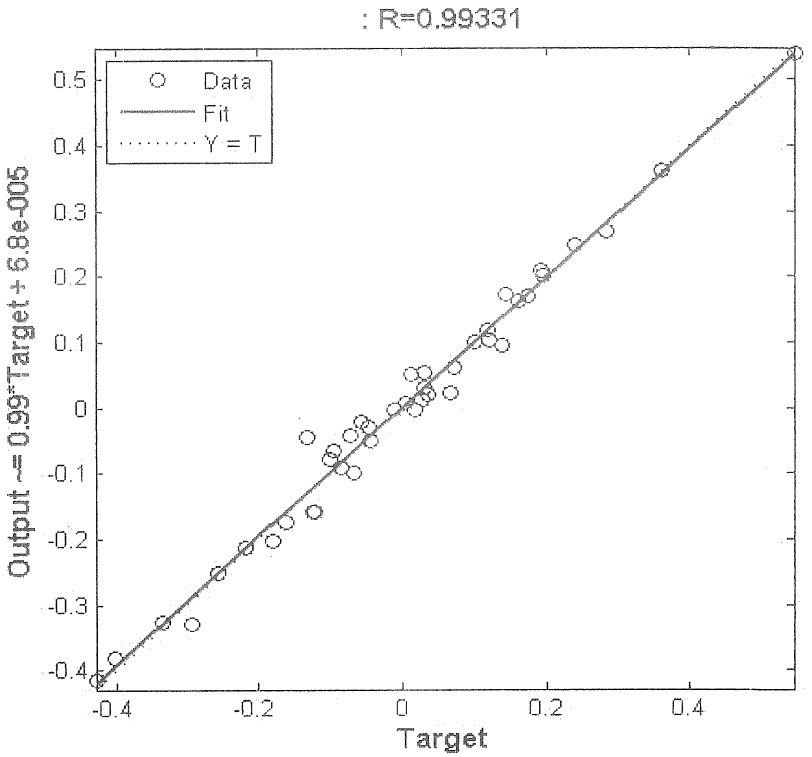


Figure 4.9. Regression plot of output Z in CNN-II-D case for training data.

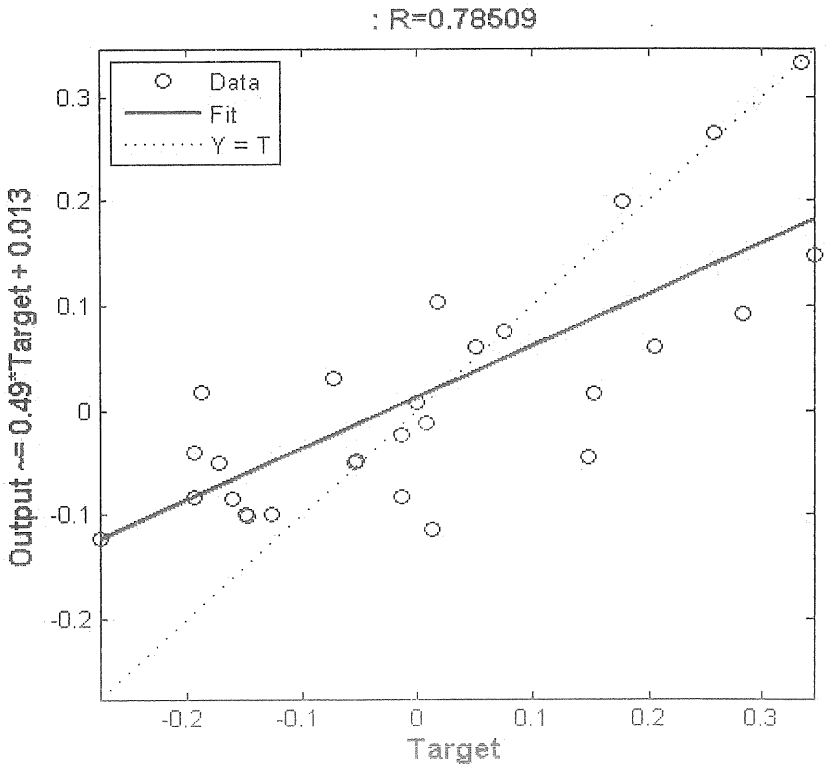


Figure 4.10. Regression plot of output X in CNN-II-D case for test data (effect of the inequality constraint $X \geq -0.1$ on the NN outputs is vaguely visible).

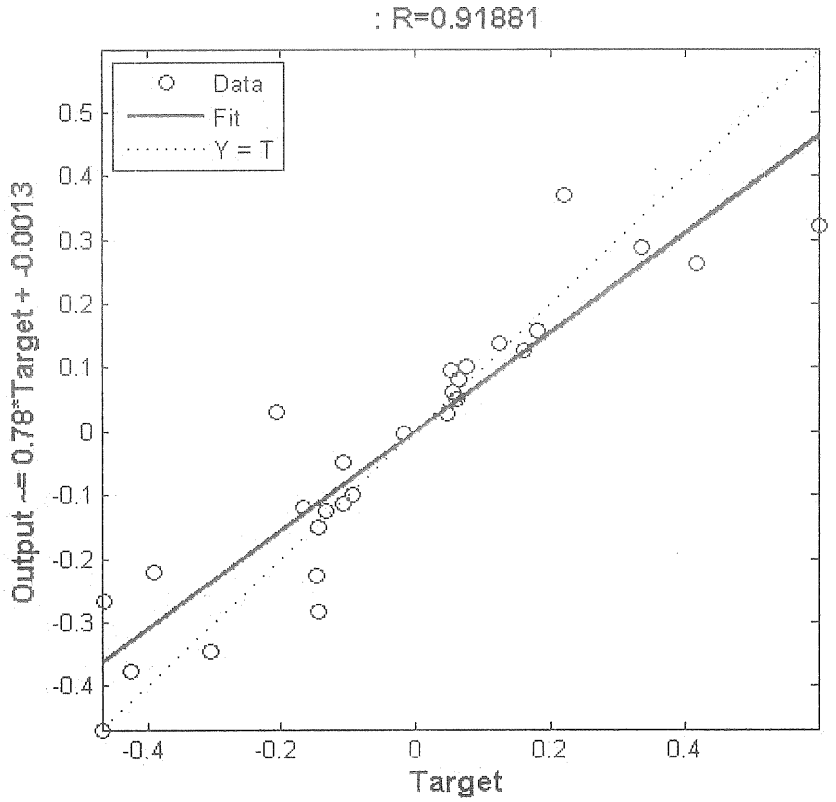


Figure 4.11. Regression plot of output Y in CNN-II-D case for test data.

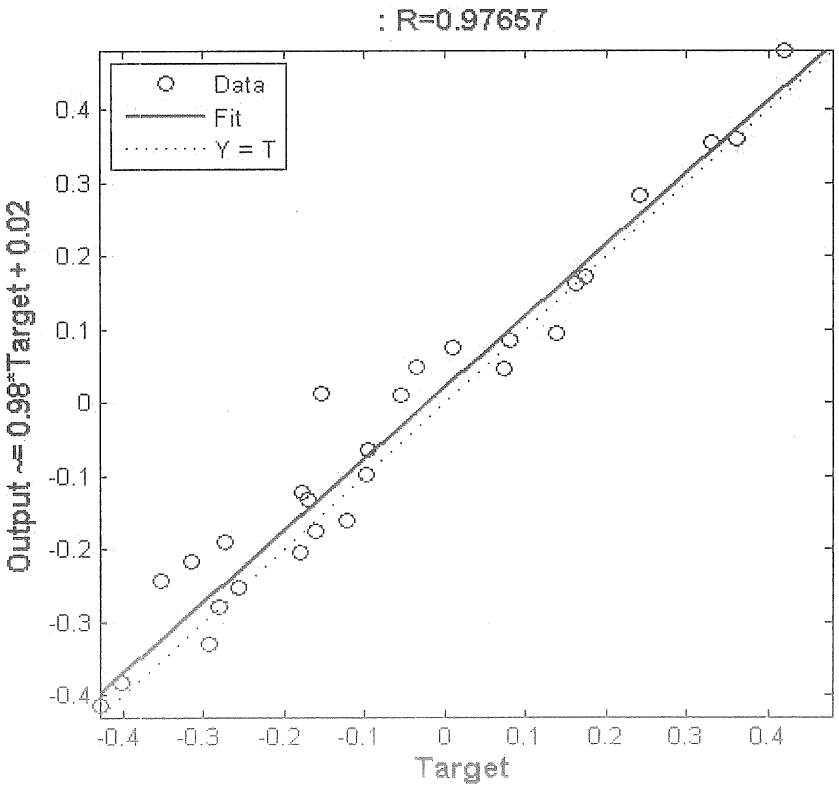


Figure 4.12. Regression plot of output Z in CNN-II-D case for test data.

As seen in Figure 4.7 and Figure 4.10, which are regression plots of constrained output X, the NN outputs stay above the constraint limit of $\mathbf{X} \geq -0.1$ (exactly for the training phase but vaguely for the testing phase). Although many outputs did not match the targets which are under the $\mathbf{X} \geq -0.1$ limit, the regression coefficients are quite high; 0.92 in the training part and 0.78 in the test part. Therefore, in this thesis, from this point on, no such regression graphs will be given since the presence of inequality or equality constraints makes these graphs pointless and since the R^2 values under constrained NN training do not provide reliable information.

The constrained NN problem can also be solved by using the penalty function technique. The optimization problem of constrained NN training with penalty function technique can be described as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \mu \sum_k max(0, g_k) + \delta \|w\| \qquad (4.11)$$

where,

$$g = -X - 0.1 \leq \mathbf{0} \qquad (4.12)$$

The NN structure is the same as in the direct constraint implementation case. The training options for the indirect implementation of constraints (penalty function technique), by CNN-I-P and CNN-II-P, and for the unconstrained NN Toolbox are the optimizer selection (`fminunc` for CNN-I-P and CNN-II-P, `trainbr` for NN Toolbox), regularization parameter (δ = 0.0001), and penalty parameter (μ = 10). This optimization problem has also 123 decision variables, however no constraints.

For CNN-I-P and CNN-II-P, initial guesses for the decision variables were obtained from the custom made initialization function by using NW initialization method that was explained in Section 3.5 as in the direct constraint implementation case. All other optimization (`fmincon`) related optional parameters are the same as in the direct constraint handling case described above.

The outputs of CNN-I-P (by penalty function technique) and NN-Toolbox are given in Figure 4.13 and Figure 4.14. A horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

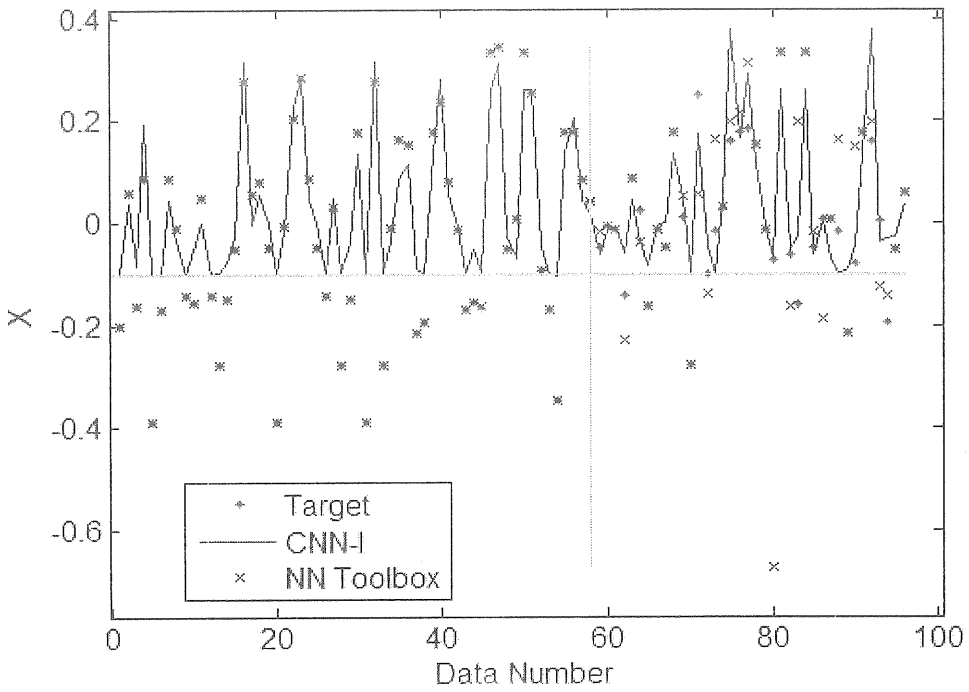


Figure 4.13. Output **X** of CNN-I-P with penalty function for Synthetic Linear Time Series.

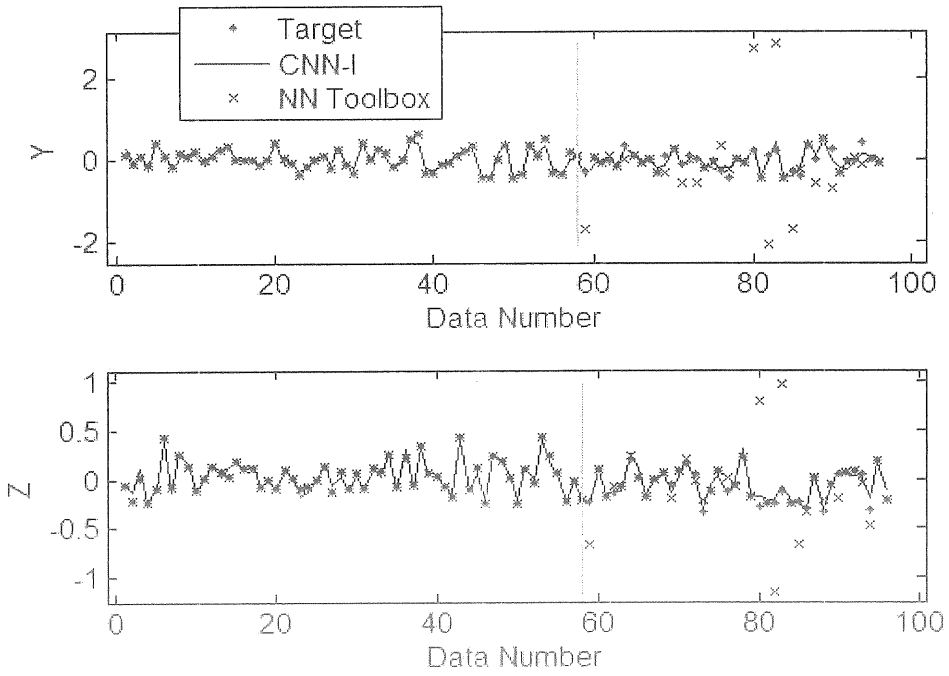


Figure 4.14. Outputs **Y** and **Z** of CNN-I-P with penalty function for the Synthetic Linear Time Series.

The regression coefficients (R^2) of the NN Toolbox are one for all outputs X, Y, and Z in the training set, respectively and 0.69, 0.35, and 0.45 for outputs X, Y, and Z in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.62. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set well but its predictions are not so well in the testing set. On the other hand, the regression coefficients of CNN-I-P are 0.91, 0.94, and 0.97 for outputs X, Y, and Z in the training set, respectively, and 0.82, 0.83, and 0.94 for the outputs X, Y, and Z in the testing set, respectively. The overall regression coefficient of CNN-I-P considering all data is 0.90. This shows that CNN-I-P under the penalty function method performs better than the NN Toolbox especially in the testing set even under inequality constraint. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The constraint violation norm, $\|-\mathbf{X}-0.1\|$, is 0.004 for CNN-I-P and 0.94 for the NN Toolbox.

The outputs of CNN-II-P (by penalty function technique) and NN-Toolbox are given in Figure 4.15 and Figure 4.16. A horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

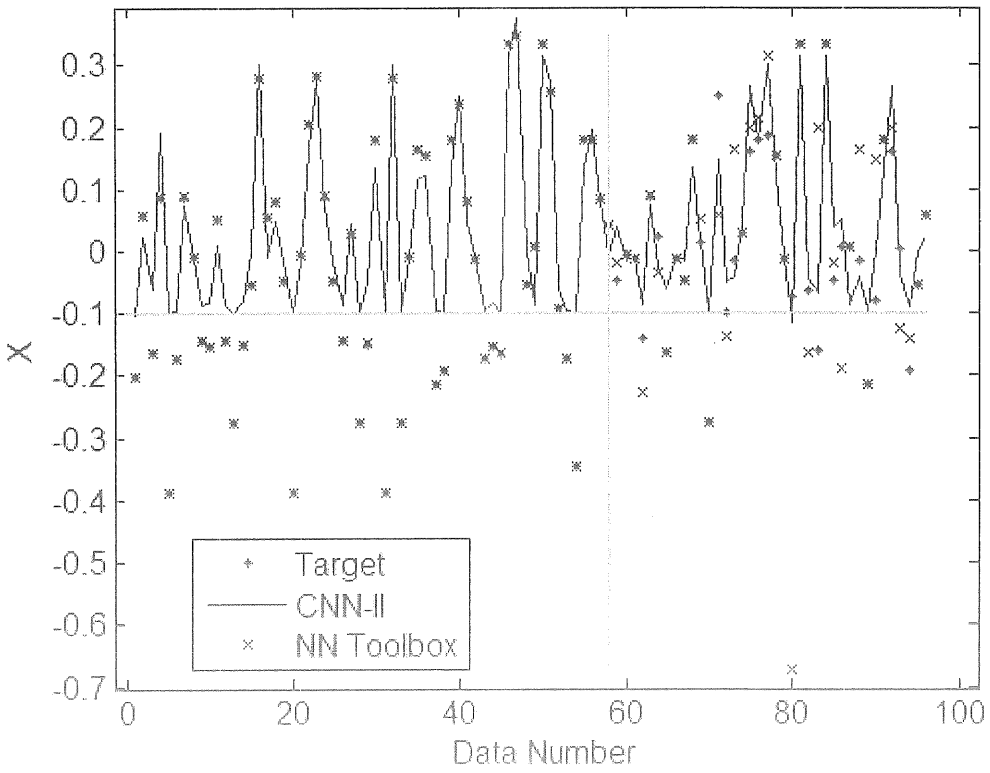


Figure 4.15. Output **X** of CNN-II-P with penalty function for Synthetic Linear Time Series.

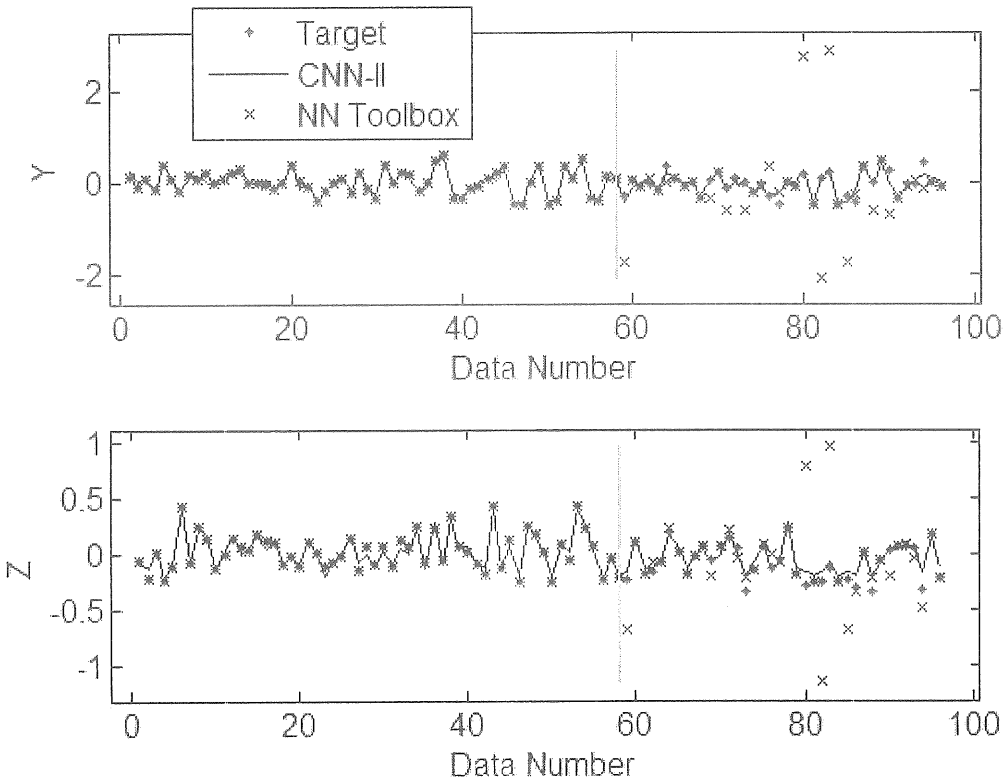


Figure 4.16. Outputs **Y** and **Z** of CNN-II-P with penalty function for the Synthetic Linear Time Series.

The regression coefficients (R^2) of the NN Toolbox are one for all outputs X, Y, and Z in the training set, respectively and 0.69, 0.35, and 0.45 for outputs X, Y, and Z in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.62. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set well but its predictions are not so well in the testing set. On the other hand, the regression coefficients of CNN-II-P are 0.92, 0.96, and 0.97 for outputs X, Y, and Z in the training set, respectively, and 0.89, 0.87, and 0.95 for the outputs X, Y, and Z in the testing set, respectively. The overall regression coefficient of CNN-II-P considering all data is 0.93. This shows that CNN-II-P under the penalty function method performs better than the NN Toolbox especially in the testing set even under inequality constraint. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The constraint violation norm, $\|-\mathbf{X}-0.1\|$, is 0.016 for CNN-II-P and 0.94 for the NN Toolbox.

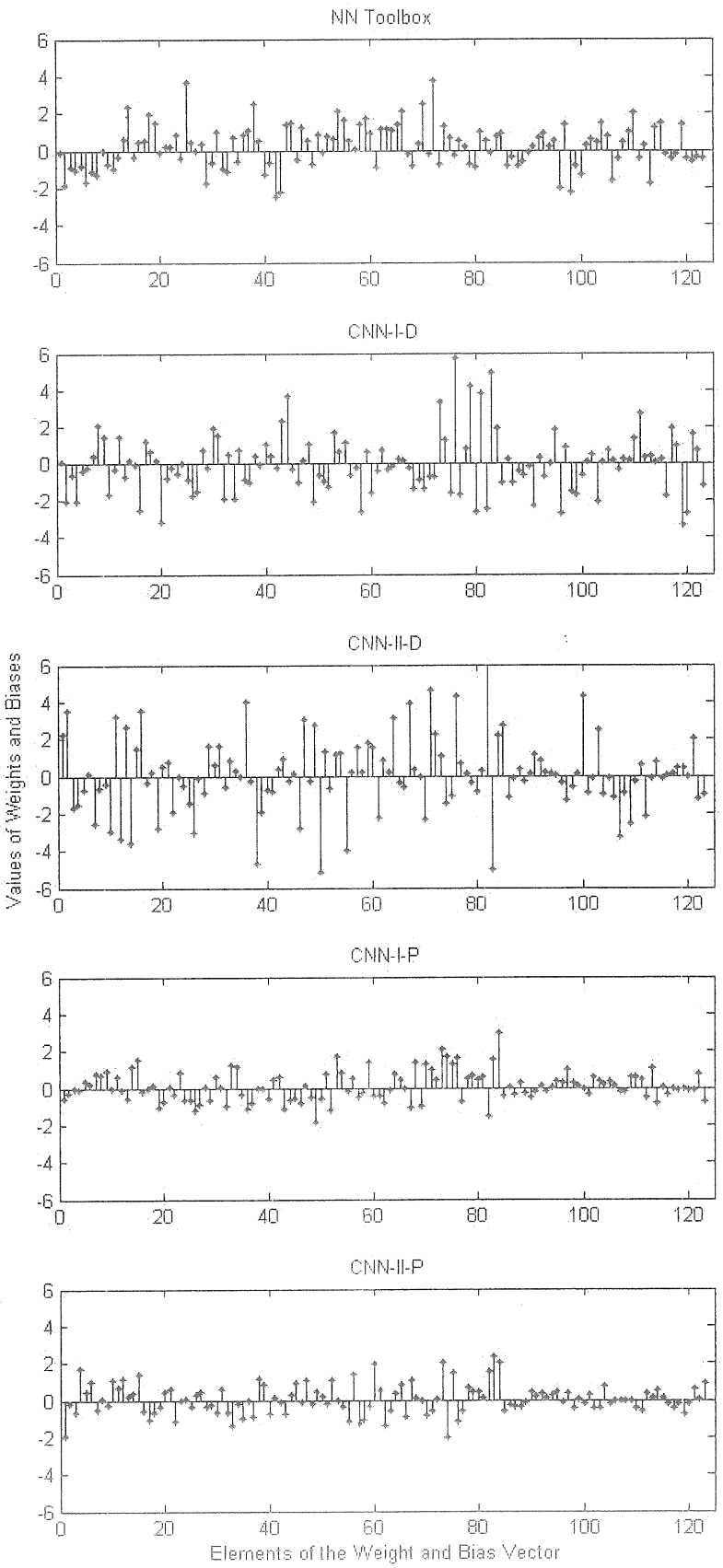


Figure 4.17. Weights and biases of the NN Toolbox and CNN methods for Synthetic Linear Time Series.

Figure 4.17 shows and compares the weights and biases of the NN Toolbox after training with `trainbr`, and the optimal decision variables (weights and biases) obtained via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-I and CNN-II. It is seen that NN Toolbox weights and biases are more homogeneous than CNN methods. The presence of less homogeneous decision variables (NN weights and biases) in the CNN methods can be attributed to possible preferential activation of some of the neurons which primarily take role in satisfying the constraints.

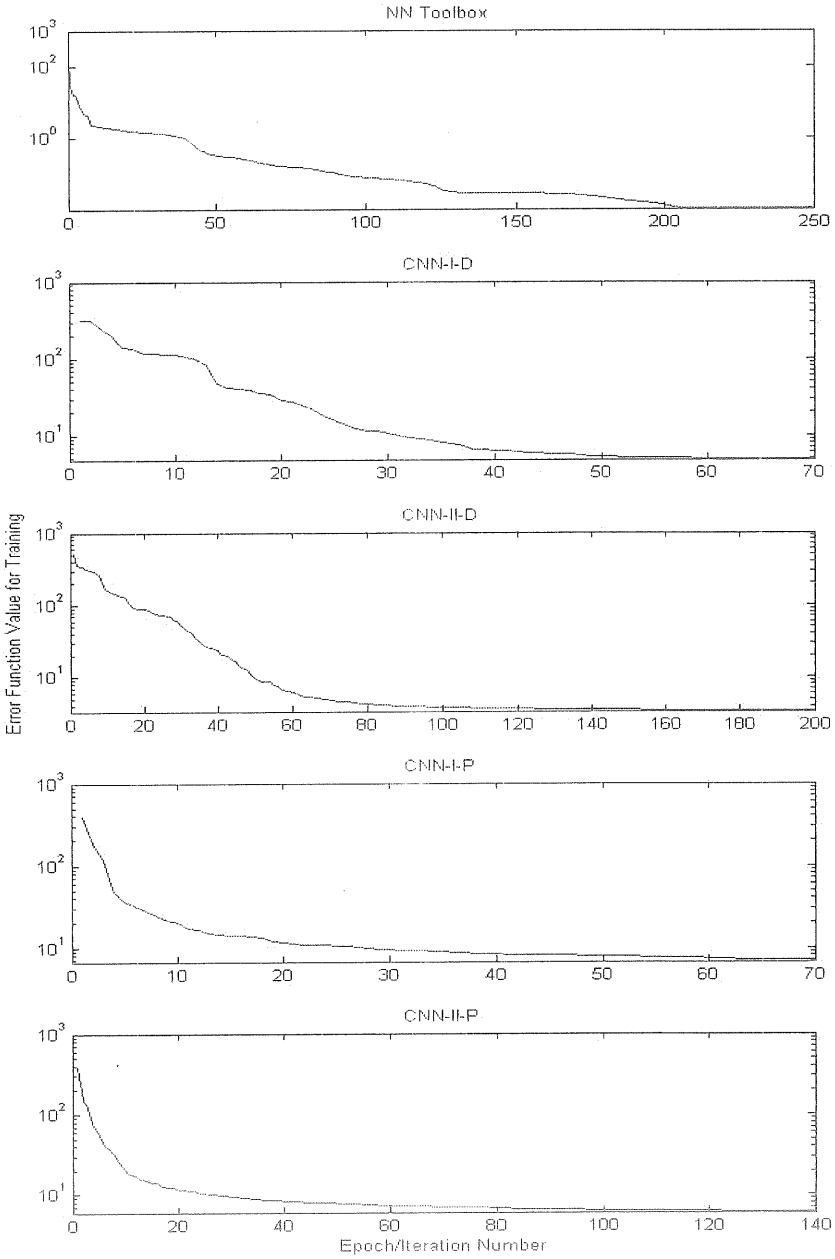


Figure 4.18. Progress of SSE in training by NN Toolbox, CNN-I, and CNN-II for the Synthetic Linear Time Series.

Figure 4.18 shows the progress of SSE as a function of epochs for the NN Toolbox and as a function of iterations for optimization via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-I and CNN-II. These show that CNN methods approach the minimum in small number of iterations comparable to the NN Toolbox. In Figure 4.18, the function values on the y-axis are the pure SSE values as reported by the NN Toolbox used with `trainbr` (Bayesian regularization), however, for CNN-I, and CNN-II, the function values on the y-axes are the SSE values including the regularized term values given by Equation 2.26 in Section 2.5.2.

The CPU times as measured by the MATLAB's `tic` and `toc` functions are compared in Figure 4.19.

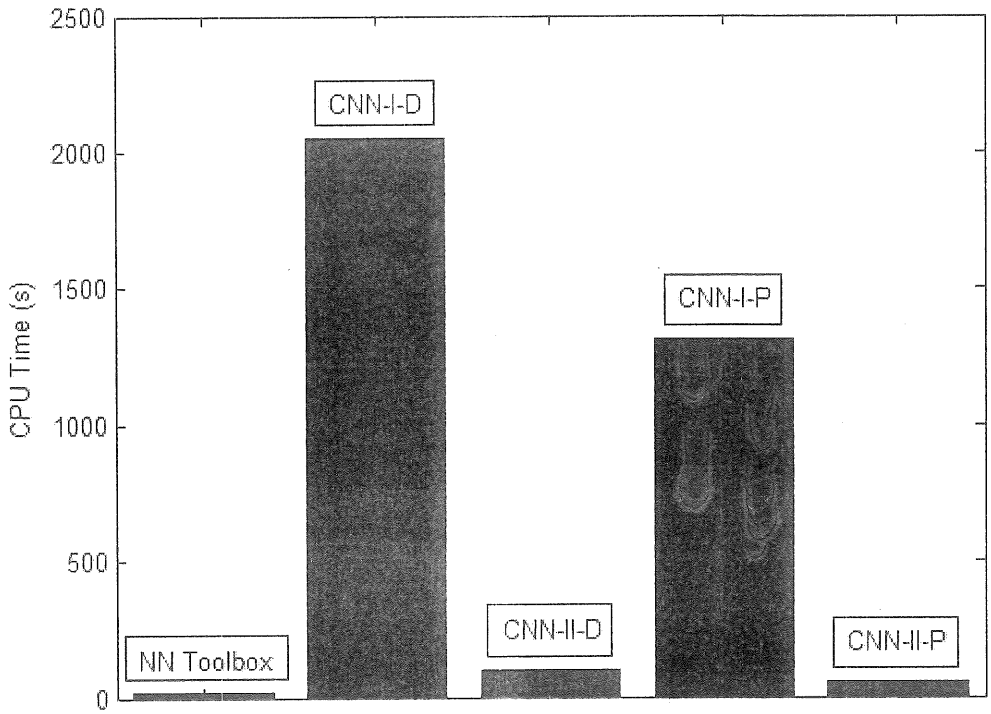


Figure 4.19. Training CPU times for the Synthetic Linear Time Series.

Table 4.1 summarizes the comparison among NN Toolbox, CNN-I-D (direct constraint implementation), CNN-II-D (direct constraint implementation), CNN-I-P (penalty function implementation), and CNN-II-P (penalty function implementation). Since the initialization of weights and biases are done randomly, the results may change from run to run; the NN Toolbox results listed in the table correspond to the run the results of which are given in Figure 4.4 and Figure 4.5.

Table 4.1. Comparison of NN Toolbox and CNN methods for the Synthetic Linear Time Series.

	NN Toolbox	CNN-I-D	CNN-II-D	CNN-I-P	CNN-II-P
Number of Parameters	123	123	123	123	123
Number of Constraints	-	58	58	-	-
Training SSE, overall	0.0023	0.7178	0.4691	1.1207	0.9483
Training SSE, X	0.0002	0.5130	0.4061	0.5807	0.5557
Training SSE, Y	0.0002	0.1606	0.0331	0.4487	0.3059
Training SSE, Z	0.0019	0.0443	0.0299	0.0912	0.0866
Test SSE, overall	2.0496	1.5495	0.9587	1.1133	0.8762
Test SSE, X	0.6851	0.2466	0.4878	0.2733	0.1781
Test SSE, Y	1.1840	1.0803	0.3742	0.7267	0.5323
Test SSE, Z	0.1805	0.2225	0.0968	0.1133	0.1658
Training Constraint Violation Norm	0.6355	0.0001	0.0001	0.0027	0.0016
Test Constraint Violation Norm	0.4159	0.1403	0.0262	0.0031	0.0165
Training CPU (s)	17	1668	101	1311	56
No of Epochs/Iterations	266	70	200	77	150

Since there is an inequality constraint on the output X, the SSE values given in the above table for the output X (and for the overall output set that includes the output X as well) should be appraised with care. However, from Figure 4.4 it is visually seen that values of X above –0.1 (values of X in the feasible region) fit to data points very well. Thus, if the SSE value of the output X is calculated based only on those data points that fall into the feasible region, it will be similar to the SSE values of the outputs Y and Z.

In summary, the performance of CNN-I and CNN-II are comparable. NN Toolbox cannot handle constraints. CNN-II is trained much faster than CNN-I since custom made

subroutines do not use input checking statements which are used heavily in the NN Toolbox as a MATLAB standard.

It should be noted that for NN Toolbox, CNN-I, and CNN-II, the initialization of weights and biases are done randomly according to the NW algorithm as explained in Section 2.4. Therefore, the SSE, constraint violation norm, and CPU values change from run to run. Therefore, the figures, SSE, and constraint violation norms, and CPU values presented in this chapter should be accepted only as representative values.

CNN-I and CNN-II both solve the same constrained NN learning (training) problems. Their SSE and constraint violation norm values are comparable. However, the CPU times of CNN-II are about two orders of magnitude lower than those of CNN-I due to code implementation as explained in Chapter 3. These observations, demonstrated with this time series example, are valid for all the other examples studied in this thesis. Therefore, considering the main purpose of the thesis, from this point on, the constrained NN results will be presented only with CNN-II, since the inclusion of CNN-I will not add any further value to the analyses.

4.1.2. Synthetic Nonlinear Time Series

The second model of synthetic series (Seth and Príncipe, 2010) consists of three nonlinear time series. The model is described as:

$$x_1(t+1) = \left(1 + \frac{x_1(t)}{1+x_1^2(t)}\right) \sin x_2(t) \tag{4.13}$$

$$x_2(t+1) = x_1(t) \exp\left(\frac{-x_1^2(t)-x_2^2(t)}{8}\right) + x_2(t) \cos x_2(t) + \frac{x_4^3(t)}{(1+x_4(t))^2+0.5\cos(x_1(t)+x_2(t))} \tag{4.14}$$

$$x_3(t+1) = \frac{x_1(t)}{1+0.5\sin x_2(t)} + \frac{x_2(t)}{1+0.5\sin x_1(t)} \tag{4.15}$$

where, x_1, x_2, and x_3 are the time series. x_4 is Gaussian white noise and t denotes the time step. There are three variables, $x_1(t+1)$, $x_2(t+1)$, and $x_3(t+1)$, which are taken as outputs of the NN. The outputs are dependent on the two variables $x_1(t)$ and $x_2(t)$, which are taken as inputs of the NN. The NN problem is depicted in Figure 4.20.

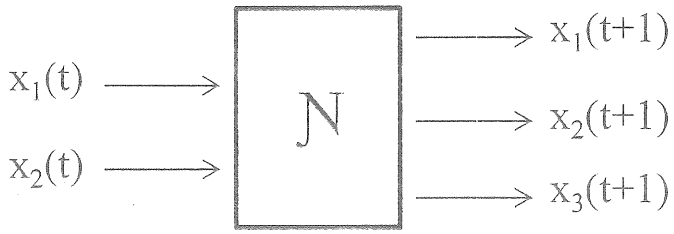


Figure 4.20. NN model of Synthetic Nonlinear Time Series.

The problem can be described as:

$$z = (s, w, x) \tag{4.16}$$

where,

$$z = [x_1\ x_2\ x_3] \tag{4.17}$$

The constraints of the problem are defined as:

$$x_1 \geq 0 \tag{4.18}$$

$$x_3 \leq 0 \tag{4.19}$$

However, the inequality constraints should be in the following form:

$$g((s, w, x), x) \leq 0 \tag{4.20}$$

Thus, the inequality constraints can be described as:

$$g_1 = -x_1 \leq 0 \tag{4.21}$$

$$g_2 = x_3 \leq 0 \tag{4.22}$$

The training of the NN with regularization and direct implementation of constraints can be given as:

$$\min_{w} f(w) = SSE(\ (s, w, x^t), y^t) + \delta\|w\| \tag{4.23}$$

$$s.t.\ g \leq 0 \tag{4.24}$$

This optimization problem of NN training was performed by CNN-II, with/without penalty function technique (CNN-II-P/CNN-II-D). The NN structure consists of two inputs meaning two input layers, 10 neurons in the single hidden layer, where all neurons have hyperbolic tangent sigmoid functions, and three neurons in the output layer all having linear activation functions. The total number of weights and biases are 50 and 13, respectively. These make the number of decision variables in the optimization problem 63.

The data set was generated from Equation 4.13 to Equation 4.15 between $0 \leq t \leq 100$. Thus, each one of the series has 100 data points. 60% of the data (60 data points) were used in the training step, and the remaining 40% (40 data points) were used in the testing step. The training options for the direct implementation of constraints, $\mathbf{g}$, by CNN-II-D, and for the unconstrained NN Toolbox are the optimizer selection (`fmincon` for CNN-II-D, `trainbr` for NN Toolbox), and regularization parameter ($\delta = 0.0001$). Regularization was used to avoid overlearning. As the error function, SSE was used since `trainbr` performs only with SSE.

For CNN-II-D, initial guesses for the decision variables (NN weights and biases, shown as $\mathbf{w}$) were obtained from the custom made initialization function as using NW initialization method that was explained in Section 3.5. With 60 data points, the total number of inequality constraints (Equation 4.18 and Equation 4.19) imposed on $\mathbf{x}_1$, and $\mathbf{x}_3$ output series becomes 120. Therefore, the optimization problem given by Equation 4.23 above has 63 decision variables, 120 inequality constraints in the form Equation 4.24.

The maximum number of epochs allowed for the `trainbr` of the NN Toolbox was set to 1000. Other optional parameters of the `trainbr` of the NN Toolbox were set such that the training will not usually stop before maximum epochs allowed (e.g., min gradient = 1×10^{-12}, max mu = 1×10^{10}). For CNN-II, the optimization termination criteria were set as TolX = 1×10^{-6}, TolFun = 1×10^{-9}, TolCon = 1×10^{-2}, and MaxIter = 120.

The outputs of CNN-II-D (by direct constraint implementation) and NN-Toolbox are given in Figure 4.21. The horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

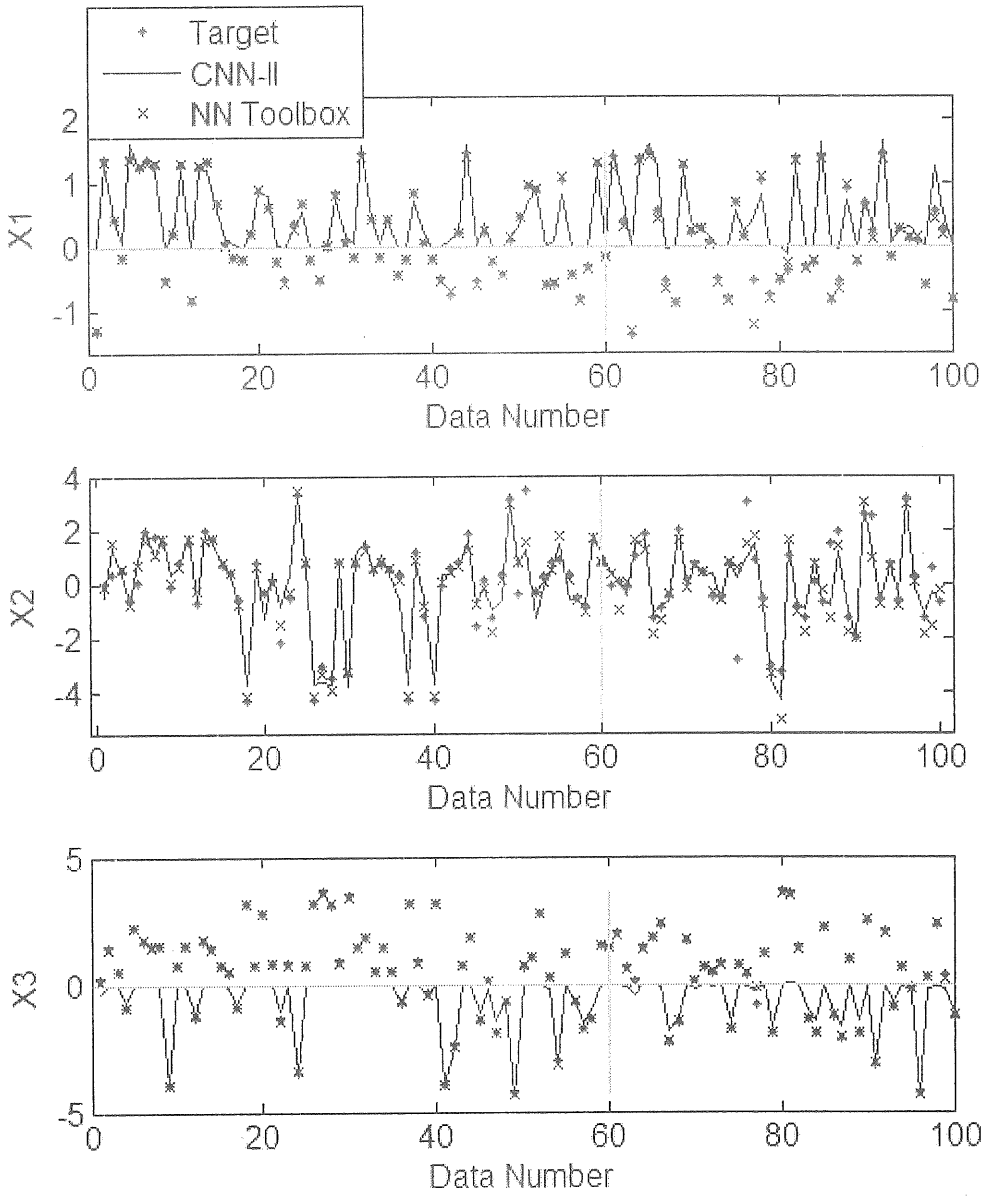


Figure 4.21. Outputs of CNN-II-D for Synthetic Nonlinear Time Series.

The regression coefficients (R^2) of the NN Toolbox are 0.99, 0.97, and 0.99 for outputs x_1, x_2, and x_3 in the training set, respectively and 0.99, 0.81, and 0.99 for outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.88. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set well and its predictions are also good in the testing set. On the other hand, the regression coefficients of CNN-II-D are 0.94, 0.94, and 0.86 for outputs x_1, x_2, and x_3 in the training set, respectively, and 0.90, 0.85, and 0.86 for the outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of CNN-II-D considering all data is 0.79. This shows that CNN-II-D performs

well both in the training and testing sets even under inequality constraints. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The norm, $\|-\mathbf{x}_1\|$, of the first constraint's violations is 0.11 for CNN-II-D and 3.84 for the NN Toolbox. The norm, $\|\mathbf{x}_3\|$, of the second constraint's violations is 0.06 for CNN-II-D and 14.4 for the NN Toolbox.

In the NN literature, it is customary to provide a predicted versus data graph for all NN outputs, annotated with the y=x line and its regression coefficient (R^2) value. However, in this thesis, no such graphs are presented since the presence of inequality or equality constraints makes these graphs pointless.

The constrained NN problem can also be solved by using the penalty function technique. The optimization problem of constrained NN training with penalty function technique can be described as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \mu \sum_k max(0, g_k) + \delta \|w\| \qquad (4.25)$$

where,

$$g_1 = -x_1 \leq 0 \qquad (4.26)$$

$$g_2 = x_3 \leq 0 \qquad (4.27)$$

The NN structure is the same as in the direct constraint implementation case. The training options for the indirect implementation of constraints (penalty function technique), by CNN-II-P, and for the unconstrained NN Toolbox are the optimizer selection (`fminunc` for CNN-II-P, `trainbr` for NN Toolbox), regularization parameter (δ = 0.0001), and penalty parameter (μ = 3). This optimization problem has also 63 decision variables, however no constraints.

For CNN-II-P, initial guesses for the decision variables were obtained from the custom made initialization function by using NW initialization method that was explained in Section 3.5 as in the direct constraint implementation case. All other optimization (`fmincon`) related optional parameters are the same as in the direct constraint handling case described above.

The outputs of CNN-II-P (by penalty function technique) and NN-Toolbox are given in Figure 4.22. A horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

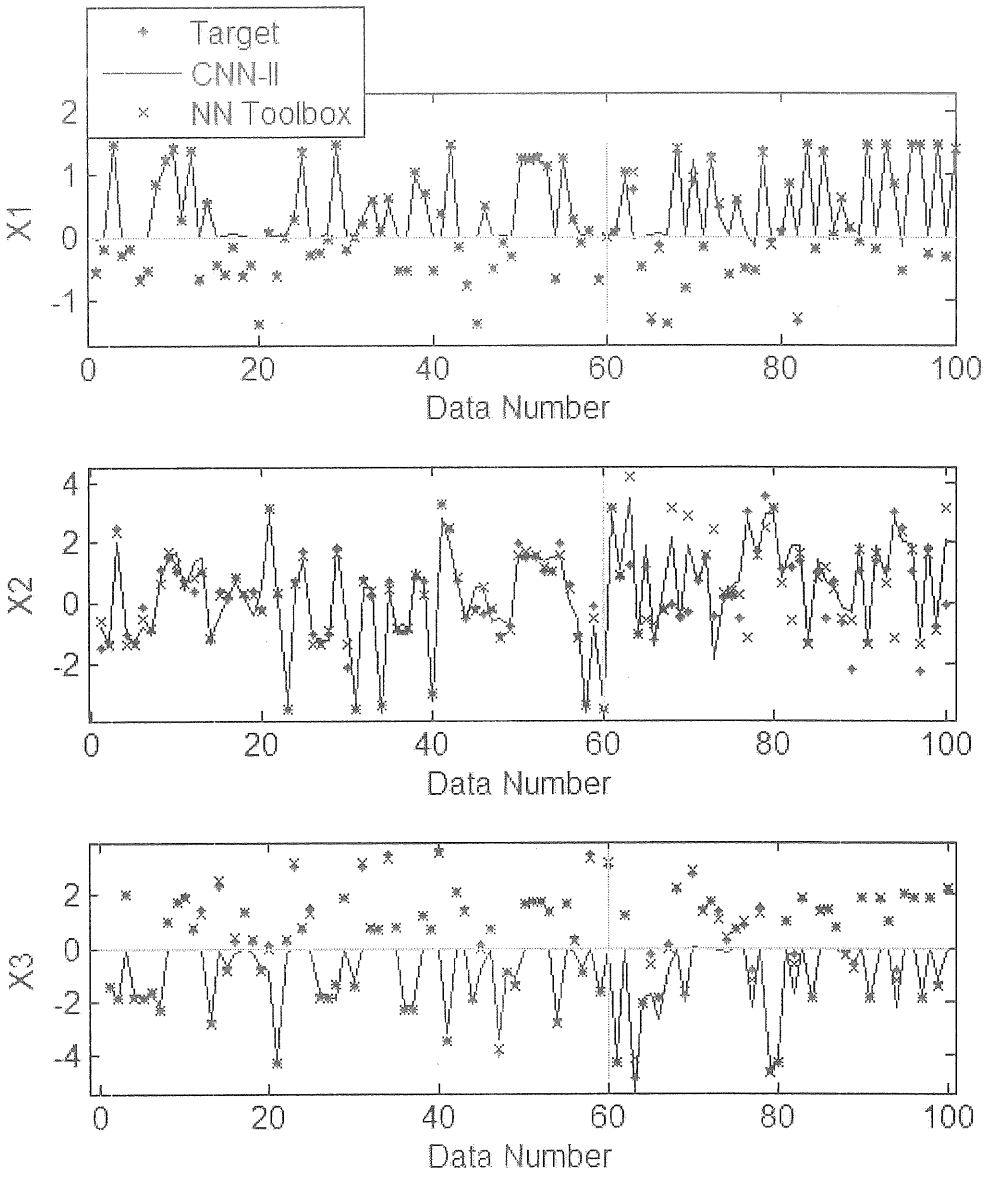


Figure 4.22. Outputs of CNN-II-P for Synthetic Nonlinear Time Series.

The regression coefficients (R^2) of the NN Toolbox are 0.99, 0.98, and 0.99 for outputs x_1, x_2, and x_3 in the training set, respectively and 0.99, 0.44, and 0.99 for outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.94. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set well and its predictions are also good

in the testing set. On the other hand, the regression coefficients of CNN-II-P are 0.92, 0.96, and 0.91 for outputs x_1, x_2, and x_3 in the training set, respectively, and 0.89, 0.82, and 0.94 for the outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of CNN-II-P considering all data is 0.81. This shows that CNN-II-P performs well both in the training and testing sets even under inequality constraints. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The norm, $\|-\mathbf{x}_1\|$, of the first constraint's violations is 0.22 for CNN-II-P and 4.09 for the NN Toolbox. The norm, $\|\mathbf{x}_3\|$, of the second constraint's violations is 0.02 for CNN-II-P and 13.2 for the NN Toolbox.

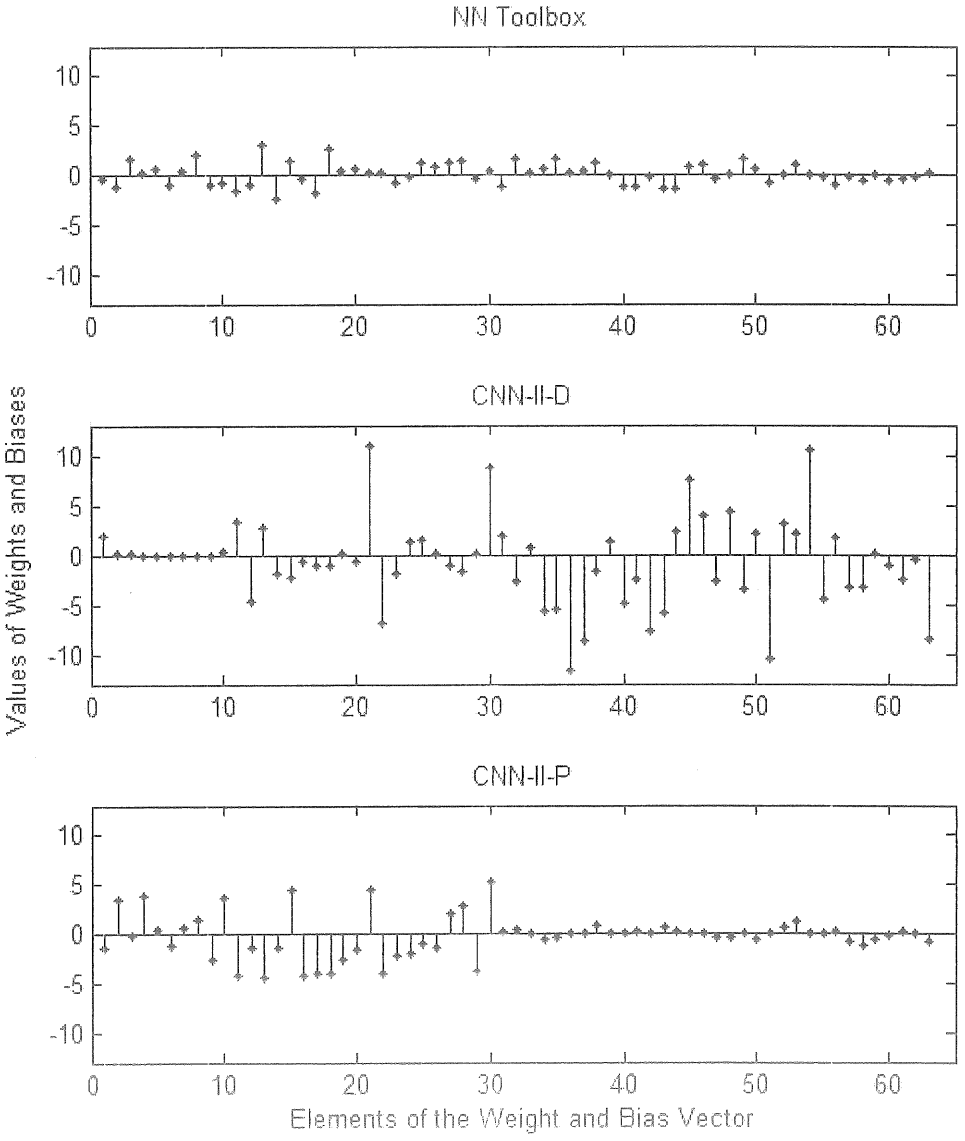


Figure 4.23. Weights and biases of the NN Toolbox and CNN-II for Synthetic Nonlinear Time Series.

Figure 4.23 shows and compares the weights and biases of the NN Toolbox after training with `trainbr` and the optimal decision variables (weights and biases) obtained via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. It is seen that NN Toolbox weights and biases are more homogeneous than CNN methods. The presence of less homogeneous decision variables (NN weights and biases) in the CNN methods can be attributed to possible preferential activation of some of the neurons which primarily take role in satisfying the constraints.

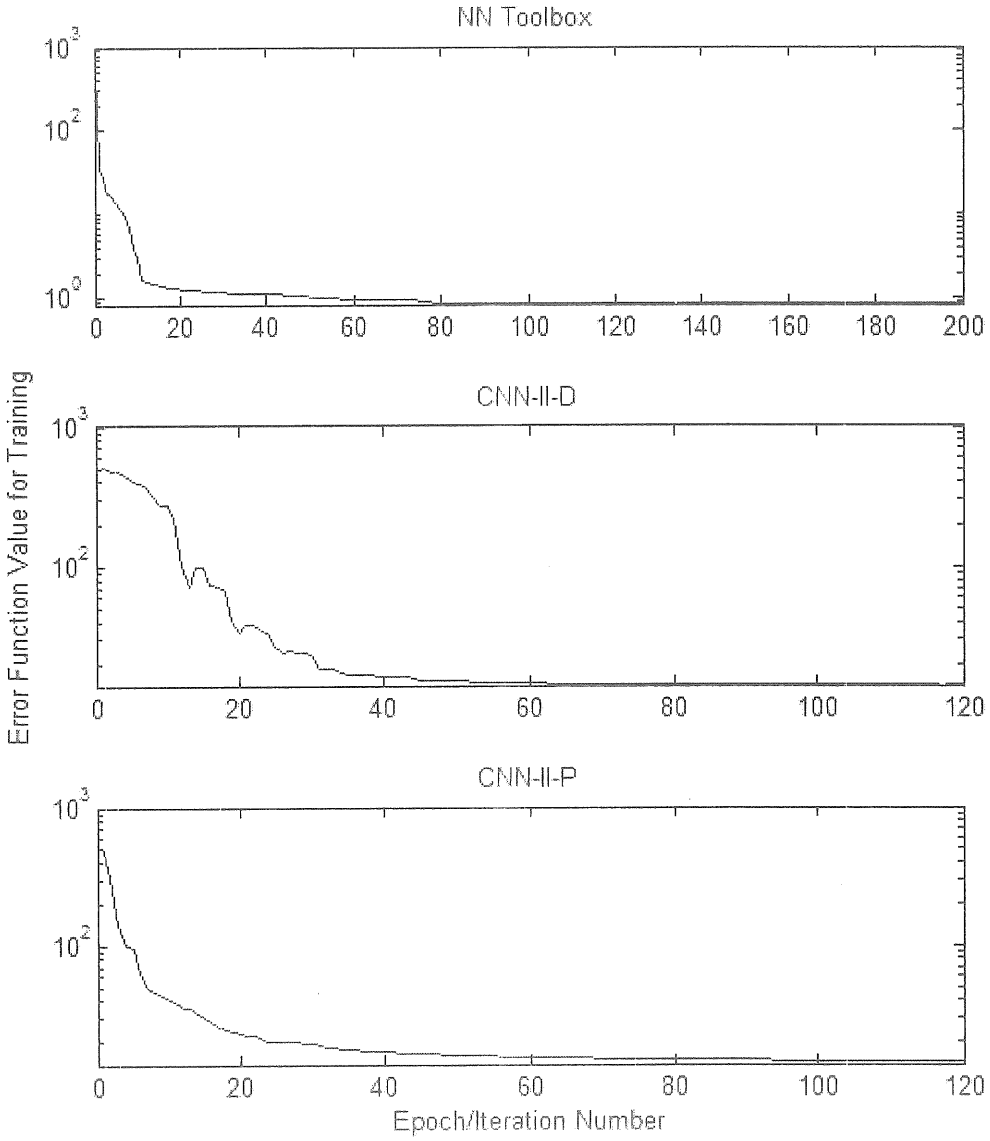


Figure 4.24. Progress of SSE in training by NN Toolbox and CNN-II for Synthetic Nonlinear Time Series.

Figure 4.24 shows the progress of SSE as a function of epochs for the NN Toolbox and as a function of iterations for optimization via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. These show that CNN methods approach the minimum in small number of iterations comparable to the NN Toolbox. In Figure 4.24, the function values on the y-axis are the pure SSE values as reported by the NN Toolbox used with `trainbr` (Bayesian regularization), however, for CNN-II, the function values on the y-axes are the SSE values including the regularized term values given by Equation 2.26 in Section 2.5.2.

The CPU times as measured by the MATLAB's `tic` and `toc` functions are compared in Figure 4.25.

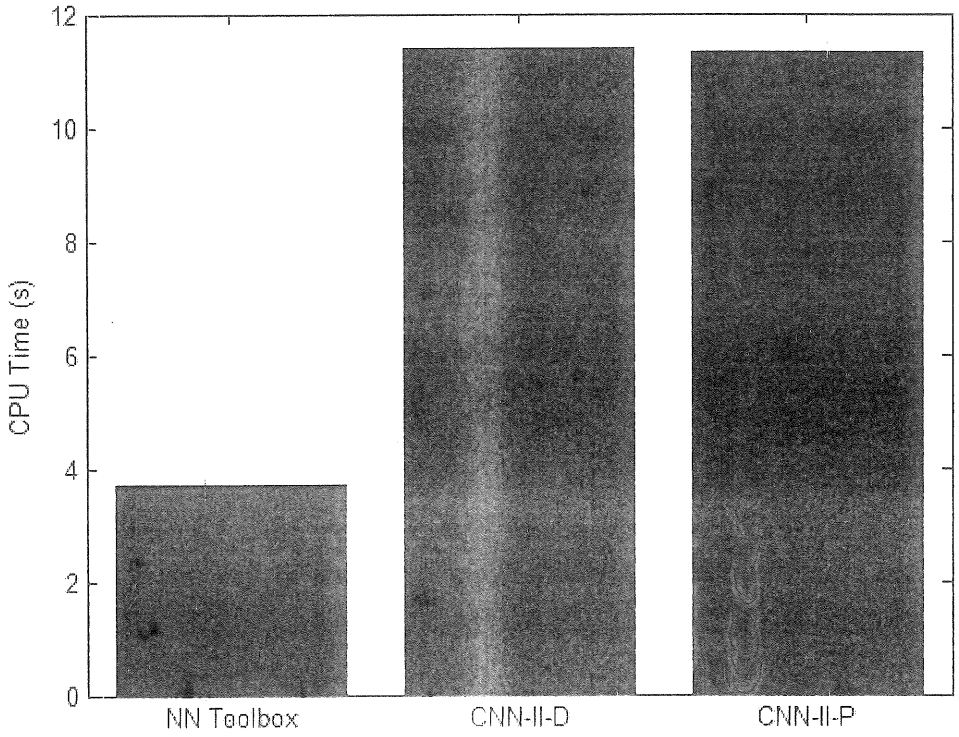


Figure 4.25. Training CPU times for Synthetic Nonlinear Time Series.

Table 4.2 summarizes the comparison among NN Toolbox, CNN-II-D (direct constraint implementation), and CNN-II-P (penalty function implementation). Since the initialization of weights and biases are done randomly, the results may change from run to run; the NN Toolbox results listed in the table correspond to the run the results of which are given in Figure 4.21.

Table 4.2. Comparison of NN Toolbox and CNN-II method for Synthetic Nonlinear Time Series.

	NN Toolbox	CNN-II-D	CNN-II-P
Number of Parameters	63	63	63
Number of Constraints	-	120	-
Training SSE, overall	12.4	167.5	142.8
Training SSE, X_1	0.06	7.30	10.10
Training SSE, X_2	12.2	21.3	11.4
Training SSE, X_3	0.1	138.8	121.3
Testing SSE, overall	39.5	111.4	112.2
Testing SSE, X_1	0.6	9.3	8.6
Testing SSE, X_2	38.2	26.5	35.8
Testing SSE, X_3	0.6	75.6	67.7
Training 1st Constraint Violation Norm	2.509	0.002	0.005
Testing 1st Constraint Violation Norm	2.909	0.105	0.219
Training 2nd Constraint Violation Norm	11.681	0	0.001
Testing 2nd Constraint Violation Norm	8.41	0.06	0.02
Training CPU (s)	3.7	11.4	11.3
Number of Epochs/Iterations	269	120	201

Since there are inequality constraints on the outputs x_1 and x_3, the SSE values given in the above table for the outputs x_1 and x_3 (and for the overall output set that includes the outputs x_1 and x_3 as well) should be appraised with care. However, from Figure 4.21 and Figure 4.22, it is visually seen that values of x_1 above zero (values of x_1 in the feasible region) and values of x_3 below zero (values of x_3 in the feasible region) fit to data points very well. Thus, if the SSE values of the outputs x_1 and x_3 are calculated based only on

those data points that fall into the feasible region, the SSE values will be similar to the SSE value of the output x_2.

In summary, NN Toolbox cannot handle constraints. It should be noted that for NN Toolbox and CNN-II, the initialization of weights and biases are done randomly according to the NW algorithm as explained in Section 2.4. Therefore, the SSE, constraint violation norm, and CPU values change from run to run. Therefore, the figures, SSE, and constraint violation norms, and CPU values should be accepted only as representative values.

4.2. Ternary Vapor-Liquid Equilibrium

In this example, a system of ternary vapor-liquid equilibrium (VLE) is considered. Isobaric vapor-liquid equilibrium data for ternary MTBE(1) + Ethanol(2) + Octane(3) are taken from Khamas (2008). The data consist of temperature (T) and corresponding liquid- and vapor-phase mole fractions. There are 122 data points. The NN is built to reflect the correlation between liquid phase mole fractions and temperature. The mole fractions of the liquid phase (x_1, x_2, and x_3) are taken as outputs and the temperature (T) is taken as input of the NN. Training functions of NN Toolbox learns the pattern in the training data very well but they give much unreliable estimation in the simulation data. The key issue in this example is that the sum of the mole fractions should add to unity, i.e., there is an implicit equality constraint imposed on the output mole fractions. Furthermore, mole fractions should be between zero and one, which implies requirement of inequality constraints. It should be noted that the data already satisfy these constraints since the reported mole fractions do not involve measurement errors. The NN is depicted in Figure 4.26.



Figure 4.26. NN model of ternary VLE.

The problem can be described as:

$$z = (s, w, x) \tag{4.28}$$

where,

$$z = [x_1 \quad x_2 \quad x_3] \tag{4.29}$$

Since the estimated mole fractions should be between zero and one, and since the sum of mole fractions should add up to unity, the constraints are defined as:

$$0 \leq x_1 \leq 1 \tag{4.30}$$
$$0 \leq x_2 \leq 1 \tag{4.31}$$
$$0 \leq x_3 \leq 1 \tag{4.32}$$
$$\sum_{i=1}^{3} x_i - 1 = 0 \tag{4.33}$$

However, the inequality constraints should be in the following form:

$$g((s, w, x), x) \leq 0 \tag{4.34}$$

Thus, the constraints can be described as:

$$g_1 = -x_1 \leq 0 \tag{4.35}$$
$$g_2 = x_1 - 1 \leq 0 \tag{4.36}$$
$$g_3 = -x_2 \leq 0 \tag{4.37}$$
$$g_4 = x_2 - 1 \leq 0 \tag{4.38}$$
$$g_5 = -x_3 \leq 0 \tag{4.39}$$
$$g_6 = x_3 - 1 \leq 0 \tag{4.40}$$
$$h = x_1 + x_2 + x_3 - 1 = 0 \tag{4.41}$$

The training of the NN with regularization and direct implementation of constraints can be given as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \delta\|w\| \tag{4.42}$$
$$s.t. \quad g \leq 0 \tag{4.43}$$
$$h = 0 \tag{4.44}$$

This optimization problem of NN training was performed by CNN-II, with/without penalty function technique (CNN-II-P/CNN-II-D). The NN structure consists of one input meaning one input layer, 50 neurons in the single hidden layer, where all neurons have hyperbolic tangent sigmoid functions, and three neurons in the output layer all having linear activation function. Total number of weights and biases are 200 and 53, respectively. These make the number of decision variables in the optimization problem 253.

The data have 122 rows. 70% of the data (86 data points) were used in the training step, and the remaining 30% (36 data points) were used in the testing step. The training options for the direct implementation of constraints, $\mathbf{g}$ and $\mathbf{h}$, by CNN-II-D, and for the unconstrained NN Toolbox are the optimizer selection (`fmincon` for CNN-II-D, `trainbr` for NN Toolbox), and regularization parameter (δ = 0.0001). Regularization was used to avoid overlearning. As the error function, SSE was used since `trainbr` performs only with SSE.

For CNN-II-D, initial guesses for the decision variables (NN weights and biases, shown as $\mathbf{w}$) were obtained from the custom made initialization function as using NW initialization method that was explained in Section 3.5. With 86 data points, the total number of inequality constraints (Equation 4.35 to Equation 4.40) imposed on $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ outputs becomes 516. The total number of equality constraints (Equation 4.41) is 86. Therefore, the optimization problem given by Equation 4.42 above has 253 decision variables, 516 inequality constraints in the form Equation 4.43 and 86 equality constraints in the form Equation 4.44.

The maximum number of epochs allowed for the `trainbr` of the NN Toolbox was set to 1500. Other optional parameters of the `trainbr` of the NN Toolbox were set such that the training will not usually stop before maximum epochs allowed (e.g., min gradient = 1×10^{-12}, max mu = 1×10^{10}). For CNN-II, the optimization termination criteria were set as TolX = 1×10^{-6}, TolFun = 1×10^{-9}, TolCon = 1×10^{-6}, and MaxIter = 200.

The outputs of CNN-II-D (by direct constraint implementation) and NN-Toolbox are given in Figure 4.27. The horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

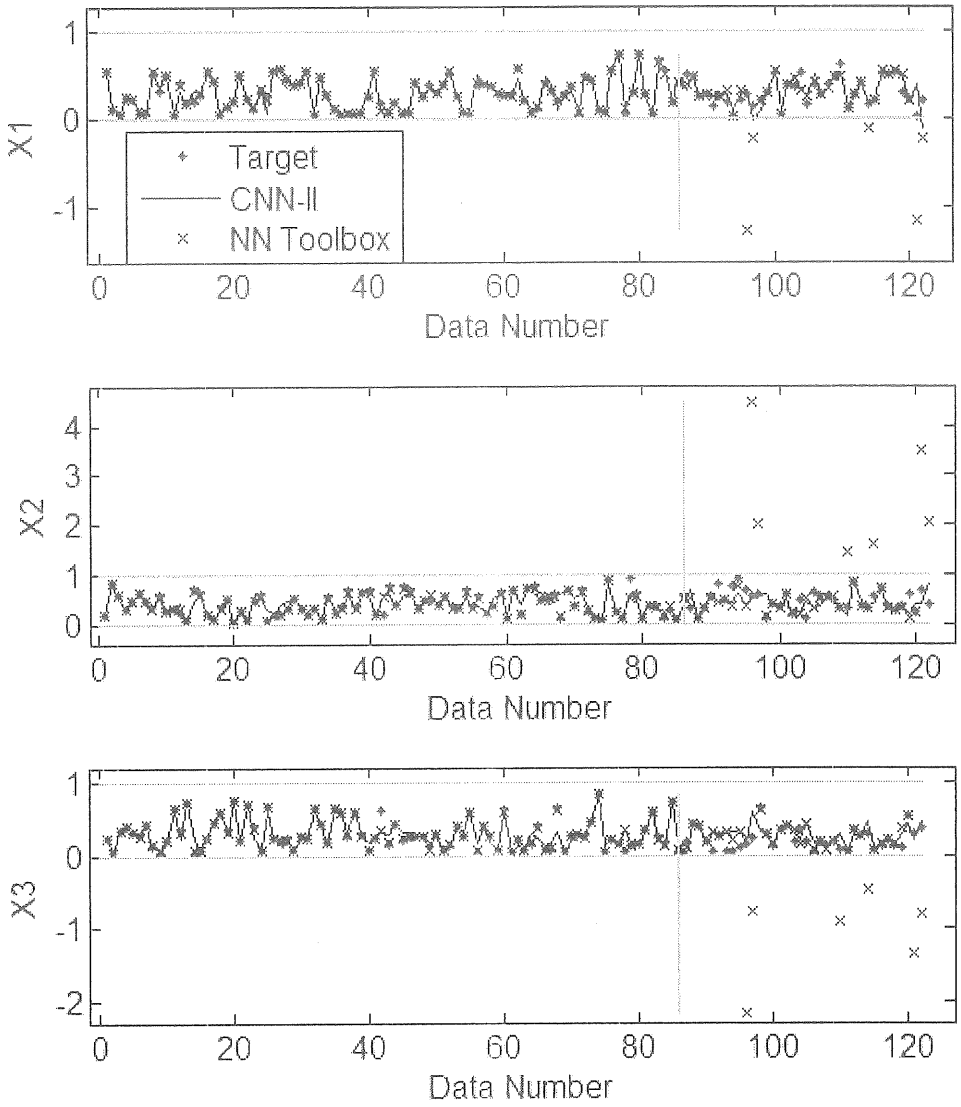


Figure 4.27. Outputs of CNN-II-D for ternary VLE.

The regression coefficients (R^2) of the NN Toolbox are 0.99, 0.96, and 0.97 for outputs x_1, x_2, and x_3 in the training set, respectively and 0.54, 0.26, and 0.17 for outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.54. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set very well, however its predictions are not good in the testing set, including many violations of the physical bounds ($0 \leq \mathbf{x} \leq 1$). On the other hand, the regression coefficients of CNN-II-D are 0.97, 0.85, and 0.93 for outputs x_1, x_2, and x_3 in the training set, respectively, and 0.75, 0.40, and 0.50 for the outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of CNN-II-D considering all data is 0.84. This shows that CNN-II-D performs well both in the training and testing sets even under inequality constraints, where violations of the

physical bounds ($0 \leq x \leq 1$) are almost negligible. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The norm, $\|-x_1\|$, of the first constraint's violations is 0.13 for CNN-II-D and 1.74 for the NN Toolbox. The norm, $\|x_1 - 1\|$, of the second constraint's violations is zero for CNN-II-D and zero for the NN Toolbox. The norm, $\|-x_2\|$, of the third constraint's violations is zero for CNN-II-D and zero for the NN Toolbox. The norm, $\|x_2 - 1\|$, of the fourth constraint's violations is zero for CNN-II-D and 4.55 for the NN Toolbox. The norm, $\|-x_3\|$, of the fifth constraint's violations is 0.01 for CNN-II-D and 2.98 for the NN Toolbox. The norm, $\|x_3 - 1\|$, of the sixth constraint's violations is zero for CNN-II-D and zero for the NN Toolbox. The norm $\|x_1 + x_2 + x_3 - 1\|$ for the equality constraint is 0.47 for CNN-II-D and 9.24 for the NN Toolbox.

Since there are negative valued outputs (mole fractions cannot be negative), the violations of equality constraints can be measured more appropriately by using absolute values of outputs in the form of Equation 4.45, which are depicted in Figure 4.28.

$$h = \sum_{i=1}^{3} |x_i| - 1 \tag{4.45}$$

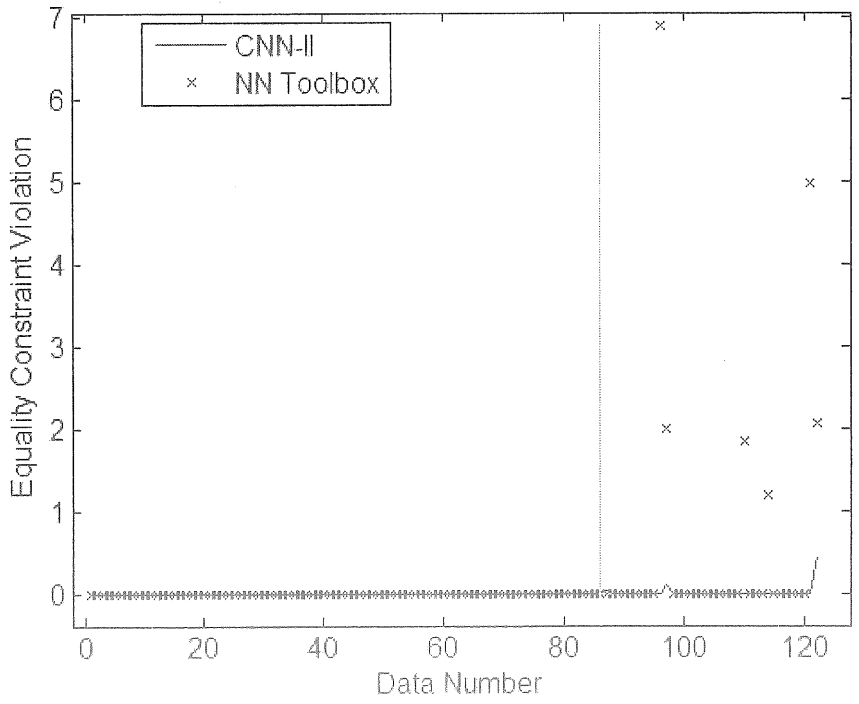


Figure 4.28. Violations of equality constraints for CNN-II-D for ternary VLE data.

In the NN literature, it is customary to provide a predicted versus data graph for all NN outputs, annotated with the y=x line and its regression coefficient (R^2) value. However, in this thesis, no such graphs are presented since the presence of inequality or equality constraints makes these graphs pointless.

The constrained NN problem can also be solved by using the penalty function technique. The optimization problem of constrained NN training with penalty function technique can be described as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \mu \sum_k max(0, g_k) + \mu \|h\| + \delta w \qquad (4.46)$$

where,

$$g_1 = -x_1 \leq 0 \qquad (4.47)$$

$$g_2 = x_1 - 1 \leq 0 \qquad (4.48)$$

$$g_3 = -x_2 \leq 0 \qquad (4.49)$$

$$g_4 = x_2 - 1 \leq 0 \qquad (4.50)$$

$$g_5 = -x_3 \leq 0 \qquad (4.51)$$

$$g_6 = x_3 - 1 \leq 0 \qquad (4.52)$$

$$h = x_1 + x_2 + x_3 - 1 = 0 \qquad (4.53)$$

The NN structure is the same as in the direct constraint implementation case. The training options for the indirect implementation of constraints (penalty function technique), by CNN-II-P, and for the unconstrained NN Toolbox are the optimizer selection (`fminunc` for CNN-II-P, `trainbr` for NN Toolbox), regularization parameter (δ = 0.0001), and penalty parameter (μ = 8). This optimization problem has also 253 decision variables, however no constraints.

For CNN-II-P, initial guesses for the decision variables were obtained from the custom made initialization function by using NW initialization method that was explained in Section 3.5 as in the direct constraint implementation case. All other optimization (`fmincon`) related optional parameters are the same as in the direct constraint handling case described above.

The outputs of CNN-II-P (by penalty function technique) and NN-Toolbox are given in Figure 4.29. A horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

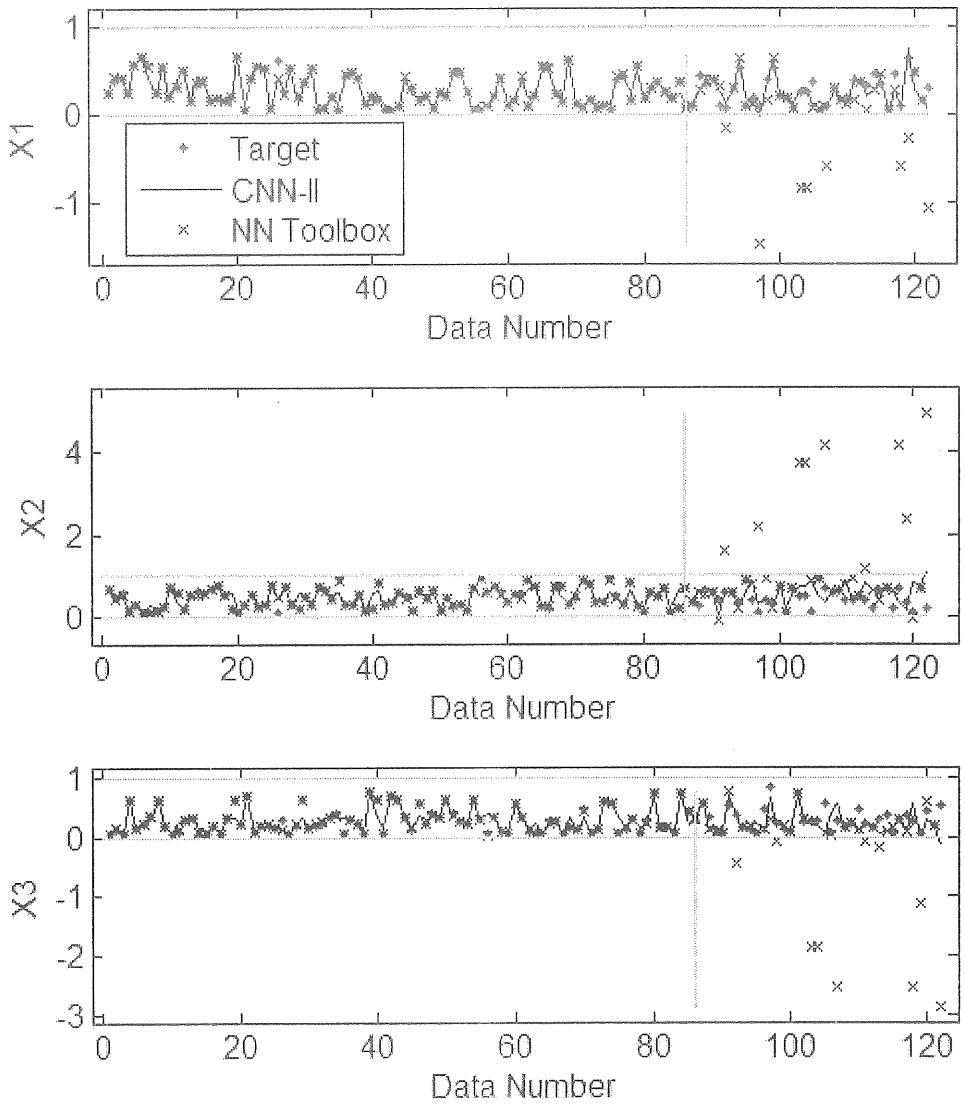


Figure 4.29. Outputs of CNN-II-P for ternary VLE.

The regression coefficients (R^2) of the NN Toolbox are 0.99, 0.98, and 0.99 for outputs x_1, x_2, and x_3 in the training set, respectively and 0.37, 0.06, and 0.07 for outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.38. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set, however its predictions are not good in the testing set, including many violations of the physical bounds ($0 \leq x \leq 1$) in all three mole fractions. On the other hand, the regression coefficients of CNN-II-P are 0.96, 0.81,

and 0.88 for outputs x_1, x_2, and x_3 in the training set, respectively, and 0.79, 0.38, and 0.40 for the outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of CNN-II-P considering all data is 0.73. This shows that CNN-II-P performs well both in the training and testing sets even under inequality constraints where violations of the physical bounds ($0 \leq \mathbf{x} \leq 1$) are basically negligible. Furthermore, it is impossible to put constraints to `trainbr` of NN Toolbox. The norm, $\|-\mathbf{x}_1\|$, of the first constraint's violations is 0.02 for CNN-II-P and 2.37 for the NN Toolbox. The norm, $\|\mathbf{x}_1 - 1\|$, of the second constraint's violations is zero for CNN-II-P and zero for the NN Toolbox. The norm, $\|-\mathbf{x}_2\|$, of the third constraint's violations is zero for CNN-II-P and 0.11 for the NN Toolbox. The norm, $\|\mathbf{x}_2 - 1\|$, of the fourth constraint's violations is 0.11 for CNN-II-P and 7.33 for the NN Toolbox. The norm, $\|-\mathbf{x}_3\|$, of the fifth constraint's violations is 0.15 for CNN-II-P and 5.45 for the NN Toolbox. The norm, $\|\mathbf{x}_3 - 1\|$, of the sixth constraint's violations is zero for CNN-II-P and zero for the NN Toolbox. The norm $\|\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 - 1\|$ for the equality constraint is 0.62 for CNN-II-P and 14.79 for the NN Toolbox.

The violations of equality constraints for CNN-II-P, as computed form of Equation 4.45, are shown in Figure 4.30.

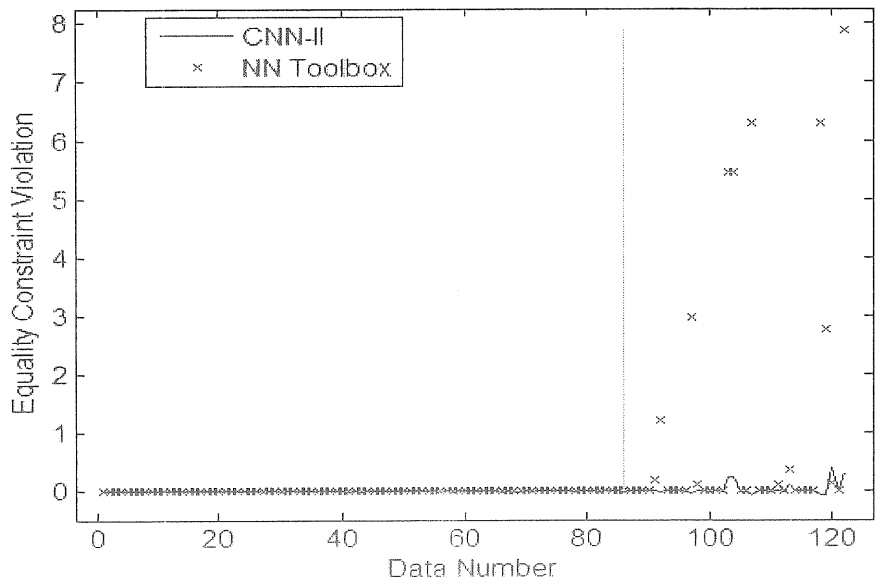


Figure 4.30. Violations of equality constraints for CNN-II-P for ternary VLE data.

Figure 4.31 shows and compares the weights and biases of the NN Toolbox after training with `trainbr` and the optimal decision variables (weights and biases) obtained

via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. It is seen that NN Toolbox weights and biases are more homogeneous than CNN methods. The presence of less homogeneous decision variables (NN weights and biases) in the CNN methods can be attributed to possible preferential activation of some of the neurons which primarily take role in satisfying the constraints.

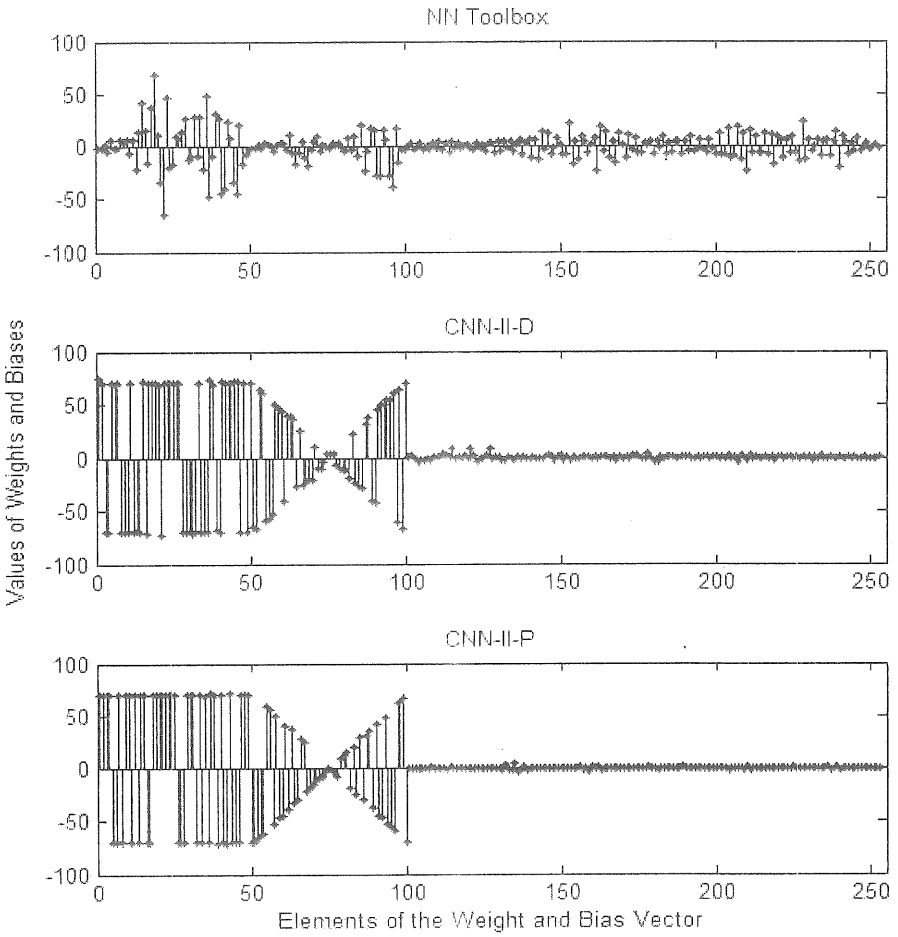


Figure 4.31. Weights and biases of the NN Toolbox and CNN-II for ternary VLE.

Figure 4.32 shows the progress of SSE as a function of epochs for the NN Toolbox and as a function of iterations for optimization via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. These show that CNN methods approach the minimum in small number of iterations comparable to the NN Toolbox. In Figure 4.32, the function values on the y-axis are the pure SSE values as reported by the NN Toolbox used with `trainbr` (Bayesian regularization), however, for CNN-II, the function values on the y-axes are the SSE values including the regularized term values given by Equation 2.26 in Section 2.5.2.

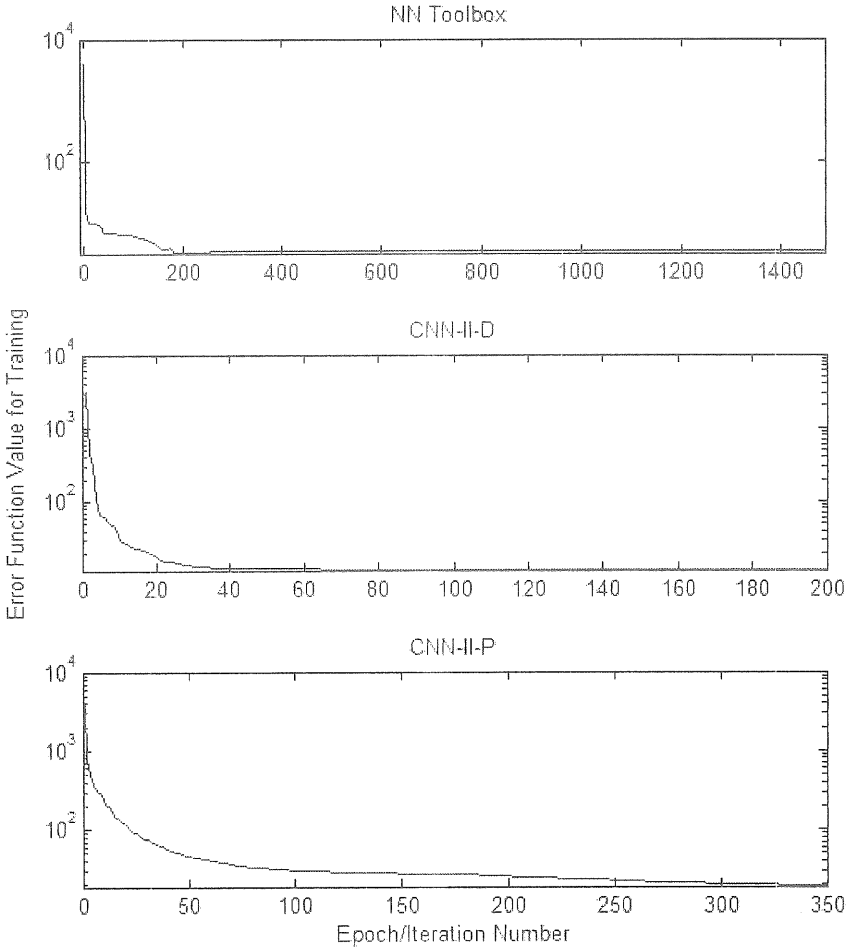


Figure 4.32. Progress of SSE in training by NN Toolbox and CNN-II for ternary VLE.

The CPU times as measured by the MATLAB's `tic` and `toc` functions are compared in Figure 4.33.

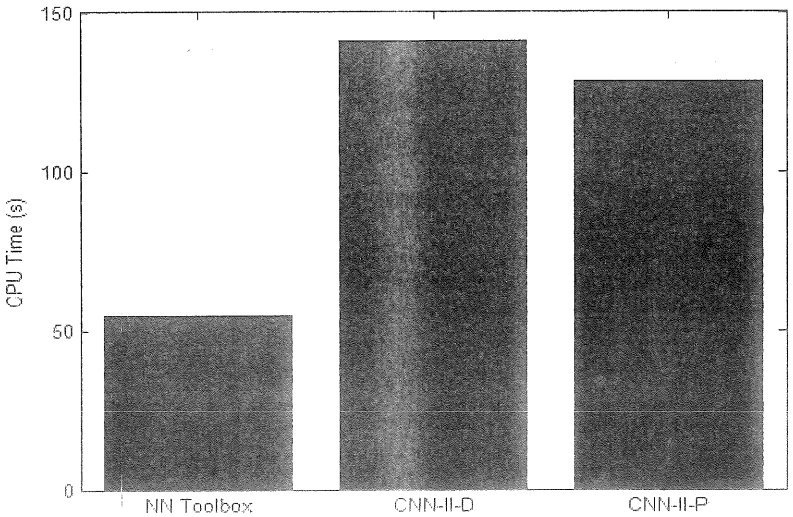


Figure 4.33. Training CPU times for ternary VLE.

Table 4.3. Comparison of NN Toolbox and CNN-II method for ternary VLE data.

	NN Toolbox	CNN-II-D	CNN-II-P
Number of Parameters	253	253	253
Number of Constraints	-	602	-
Training SSE, overall	0.55	1.91	2.79
Training SSE, x_1	0.02	0.18	0.21
Training SSE, x_2	0.33	1.19	1.73
Training SSE, x_3	0.19	0.52	0.84
Testing SSE, overall	47.88	2.88	6.44
Testing SSE, x_1	4.43	0.47	0.43
Testing SSE, x_2	30.98	1.64	4.22
Testing SSE, x_3	12.46	0.76	1.77
Training Overall Constraint Violation Norm	1.2×10^{-4}	4.1×10^{-7}	1.3×10^{-2}
Testing Overall Constraint Violation Norm	10.83	0.49	0.52
Training 1st Constraint ($\mathbf{g}_1$) Violation Norm	0	0	0
Testing 1st Constraint ($\mathbf{g}_1$) Violation Norm	1.74	0.13	0.02
Training 2nd Constraint ($\mathbf{g}_2$) Violation Norm	0	0	0
Testing 2nd Constraint ($\mathbf{g}_2$) Violation Norm	0	0	0
Training 3rd Constraint ($\mathbf{g}_3$) Violation Norm	0	0	0
Testing 3rd Constraint ($\mathbf{g}_3$)Violation Norm	0	0	0
Training 4th Constraint ($\mathbf{g}_4$) Violation Norm	0	0	0
Testing 4th Constraint ($\mathbf{g}_4$) Violation Norm	4.5	0	0.1
Training 5th Constraint ($\mathbf{g}_5$) Violation Norm	0	0	0
Testing 5th Constraint ($\mathbf{g}_5$) Violation Norm	2.98	0.01	0.15
Training 6th Constraint ($\mathbf{g}_6$) Violation Norm	0	0	0
Testing 6th Constraint ($\mathbf{g}_6$) Violation Norm	0	0	0
Training 7th Constraint ($\mathbf{h}$) Violation Norm	1.9×10^{-4}	2.1×10^{-6}	1.6×10^{-2}
Testing 7th Constraint ($\mathbf{h}$) Violation Norm	9.24	0.47	0.62
Training CPU (s)	54.5	141.1	128.5
Number of Epochs/Iterations	1500	200	351

Table 4.3 summarizes the comparison among NN Toolbox, CNN-II-D (direct constraint implementation), and CNN-II-P (penalty function implementation). Since the initialization of weights and biases are done randomly, the results may change from run to run; the NN Toolbox results listed in the table correspond to the run the results of which are given in Figure 4.27.

Since there are inequality and equality constraints on the outputs x_1, x_2, and x_3, the SSE values given in the above table for the outputs x_1, x_2, and x_3 should be appraised with care. However, from Figure 4.27 and Figure 4.29, it is visually seen that values of x_1 above zero and below one (values of x_1 in the feasible region), values of x_2 above zero and below one (values of x_2 in the feasible region), and values of x_3 above zero and below one (values of x_3 in the feasible region) fit to data points very well.

In summary, NN Toolbox cannot handle constraints. It should be noted that for NN Toolbox and CNN-II, the initialization of weights and biases are done randomly according to the NW algorithm as explained in Section 2.4. Therefore, the SSE, constraint violation norm, and CPU values change from run to run. Therefore, the figures, SSE, and constraint violation norms, and CPU values should be accepted only as representative values.

4.2.1. Reproducibility Analysis

Since the initialization of weights and biases are done randomly according to the NW algorithm as explained in Section 2.4, the results change from run to run. The distribution of the results that stem from this random initialization of weights and biases in the training phase and their statistical properties (mean values, μ, and standard deviations, σ) are provided in Figure 4.34 to Figure 4.36 for both the training and testing phases. These frequency-distribution plots were obtained by running the NN Toolbox, CNN-II-D, and CNN-II-P codes 200 times and by plotting the histograms with a bin of 20. Such reproducibility (sensitivity to random initialization) analysis will be presented only for this example of the thesis since the most significant variability was observed for this example due to the nature of the data.

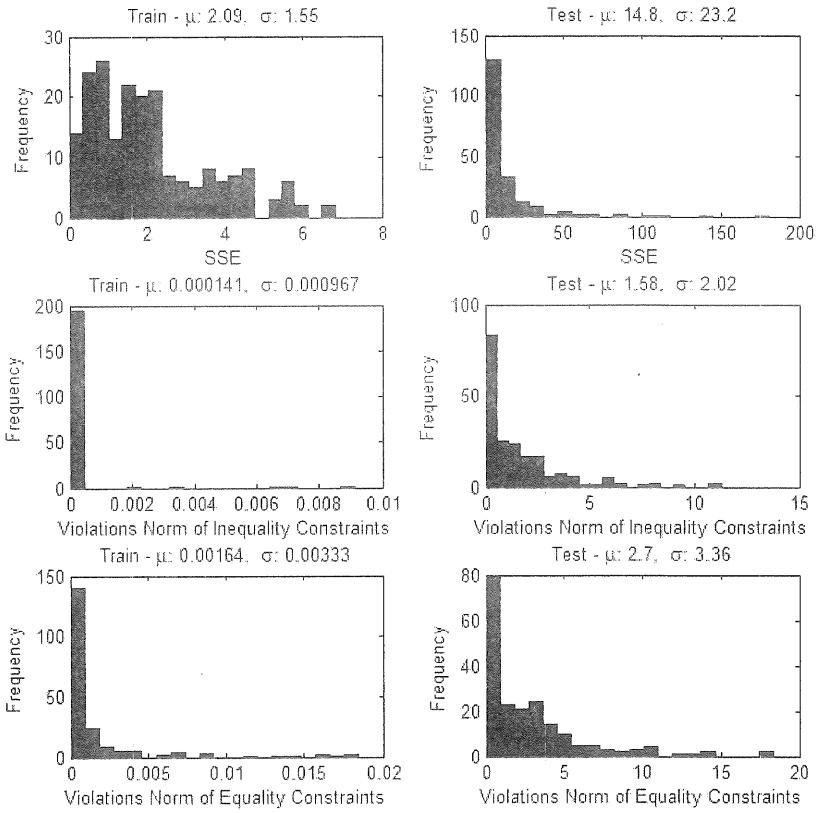


Figure 4.34. Sensitivity of NN Toolbox to random initializations.

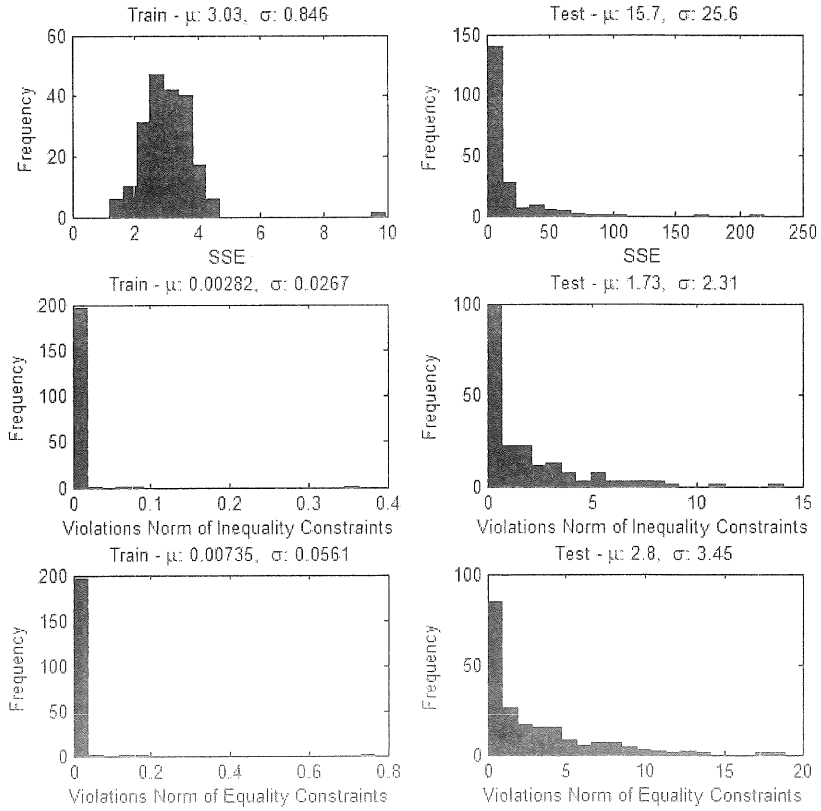


Figure 4.35. Sensitivity of CNN-II-D to random initializations.

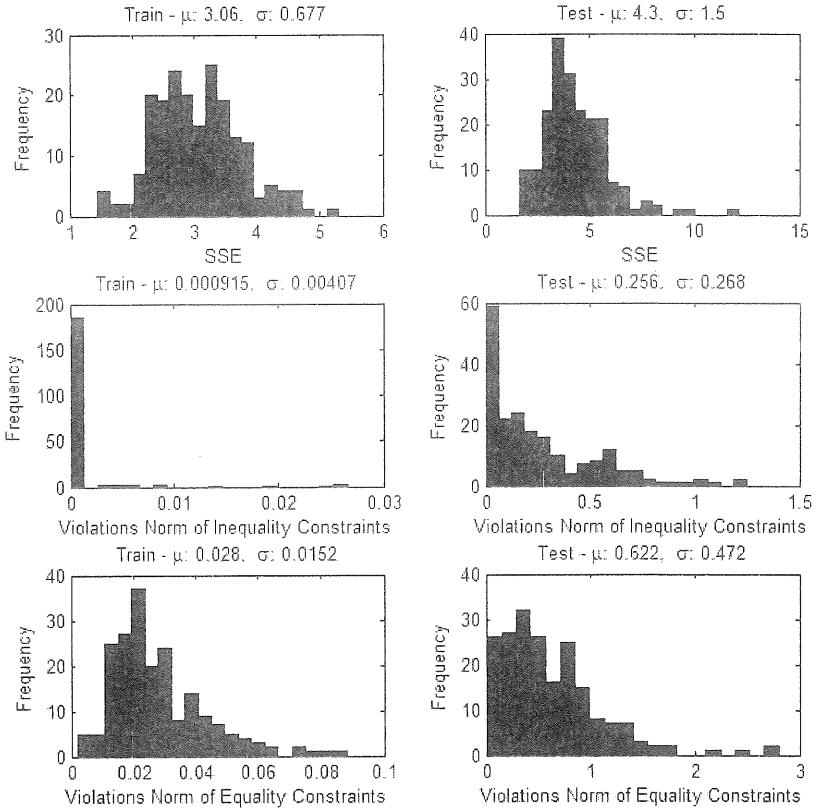


Figure 4.36. Sensitivity of CNN-II-P to random initializations.

In the 200 runs with random initializations of the NN Toolbox, CNN-II-D and CNN-II-P, the SSE, violations norm of inequality and equality constraints were collected both in the training and testing phases. Thus, the comparison can be made for the sensitivities in both regression and forecasting under equality and inequality constraints. The means (μ) of the SSE values are 2.09, 3.03, and 3.06 in the training phase and 14.8, 15.7, and 4.3 in the testing phase for the NN Toolbox, CNN-II-D, and CNN-II-P, respectively. The standard deviations (σ) of the SSE values are 1.55, 0.84, and 0.67 in the training phase and 23.2, 25.6, and 1.5 in the testing phase for the NN Toolbox, CNN-II-D, and CNN-II-P, respectively. The SSE values show the performance in fitting the data. The SSE values are expected to be lower for the NN Toolbox rather than CNN methods, since CNN methods have to satisfy the constraints besides fitting the data during training. However, CNN-II-P performed better in the testing phase with its average SSE and lower standard deviation values. This means that it will have performance of fitting probably more close to the average in each run. In other words, CNN-II-P has the lowest sensitivity, and thus uncertainty, to random initializations, even under constraints. The means of the norm of inequality constraint violations are 0.0001, 0.002, and 0.0009 in the training phase and

1.58, 1.73, and 0.25 in the testing phase for the NN Toolbox, CNN-II-D, and CNN-II-P, respectively. The standard deviations of the norm of inequality constraint violations are 0.0009, 0.02, and 0.004 in the training phase and 2.02, 2.31, and 0.26 in the testing phase for the NN Toolbox, CNN-II-D, and CNN-II-P, respectively. The means of the norm of equality constraint violations are 0.0016, 0.007 and 0.028 in the training phase and 2.7, 2.8, and 0.62 in the testing phase for the NN Toolbox, CNN-II-D, and CNN-II-P, respectively. The standard deviations of the norm of equality constraint violations are 0.0033, 0.05, and 0.015 in the training phase and 3.36, 3.45, and 0.47 in the testing phase for the NN Toolbox, CNN-II-D, and CNN-II-P, respectively. Since the data itself satisfy the inequality and equality constraints, the NN Toolbox may falsely be expected to perform well in satisfying the constraints as well as in fitting the data. However, the performance of the NN Toolbox (SSE) was good only in training phase but not in the testing phase, where it showed large deviations in its outputs from run to run (large standard deviation, large uncertainty, less reliable). The NN Toolbox, with histograms of high probability near zero but skewed to the right, exhibit probabilities of getting constraint violations much larger than zero (or an acceptable violation threshold). The NN Toolbox has fatter tails than CNNs. Thus, CNNs, especially CNN-II-P, have lower probabilities of yielding larger (extreme) violations, and thus their overall results are more reliable (lower standard deviation, lower uncertainty). The mean values of the results are important, however, the standard deviations have also a great importance in reproducibility assessment. It determines the probability of having a similar solution in another run. Therefore, CNNs, especially CNN-II-P, have thinner tails than NN Toolbox and have a higher probability to obtain similar solutions in each run. This sensitivity analysis also shows that `fmincon` (used with CNN-II-D) has more prone to premature stopping due to local minima compared to `fminunc` (used with CNN-II-P) since CNN-II-P has a much lower sensitivity to random initialization of the weights and biases.

4.3. Ternary Vapor-Liquid Equilibrium with Measurement Errors

In this section, the previous example of ternary vapor-liquid equilibrium (VLE) is considered again, however this time, hypothetical measurement errors (noises) are added to the data. The data consist of temperature (T) and corresponding liquid- and vapor-phase mole fractions. There are 122 data points. In the previous example, the data used were

exact, such that the sums of mole fractions add up to unity. However, in real life applications, there is measurement errors associated with the data, mostly in the form of white noise. In order to simulate this situation, white noise is added to the actual data. Thus, in the modified data set used in this section, the sums of mole fractions are not equal to one. The NN Toolbox fits the data containing measurement errors very well, and since it does not (it cannot) care about the implicit constraint that the mole fractions should add up to unity, it cannot provide outputs (mole fractions) that sum up to one. However, this constraint, as well as the bounds on the mole fractions ($0 \leq x \leq 1$), can be learned by the CNN methods developed in this thesis work.

All of the NN structures, constraints, and settings used for NN Toolbox, CNN-II-D, and CNN-II-P are exactly the same as those of the previous section. The only difference is that measurement errors represented by normal random numbers (with mean zero and standard deviation 0.02) are added to the data, such that the sums of mole fractions in the data range from 0.90 to 1.09.

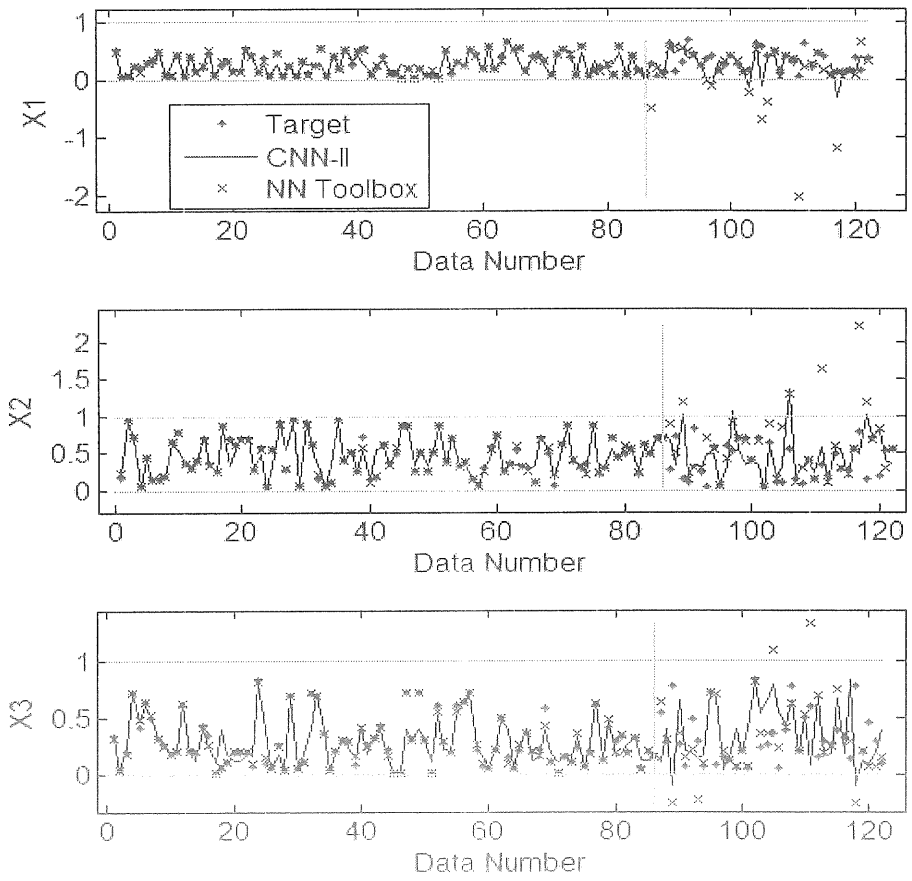


Figure 4.37. Outputs of CNN-II-D for ternary VLE with measurement errors.

The outputs of CNN-II-D (by direct constraint implementation) and NN-Toolbox are given in Figure 4.37. The horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

As in the previous example, the violations of equality constraints for CNN-II-D, as computed form of Equation 4.45, are shown in Figure 4.38.

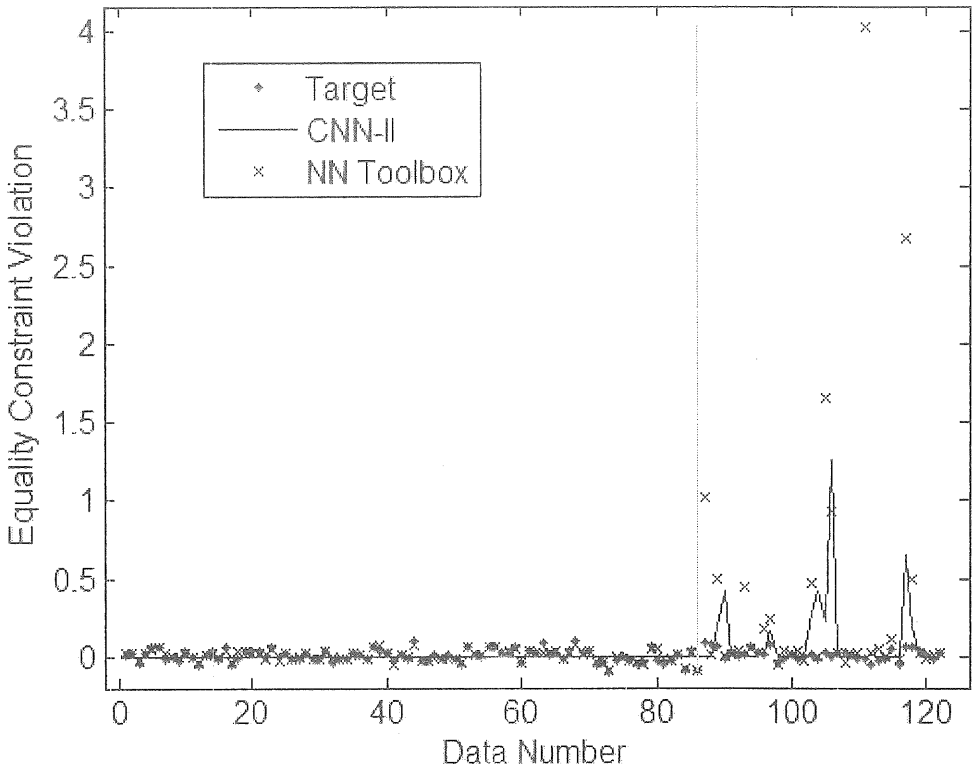


Figure 4.38. Violations of equality constraints for CNN-II-D for ternary VLE with measurement errors.

Figure 4.39 illustrates the training part of the violations of equality constraints ($\mathbf{h} = \sum x - 1 = 0$), the total number of which is equal to the row number of the data, for CNN-II-D. The NN Toolbox fits the data very well in the training part but there are small violations in the equality constraints since there is a white noise added to the data. However, CNN method can adapt to the noisy data very well, and at the same time, satisfy the requirement that mole fractions sum to one.

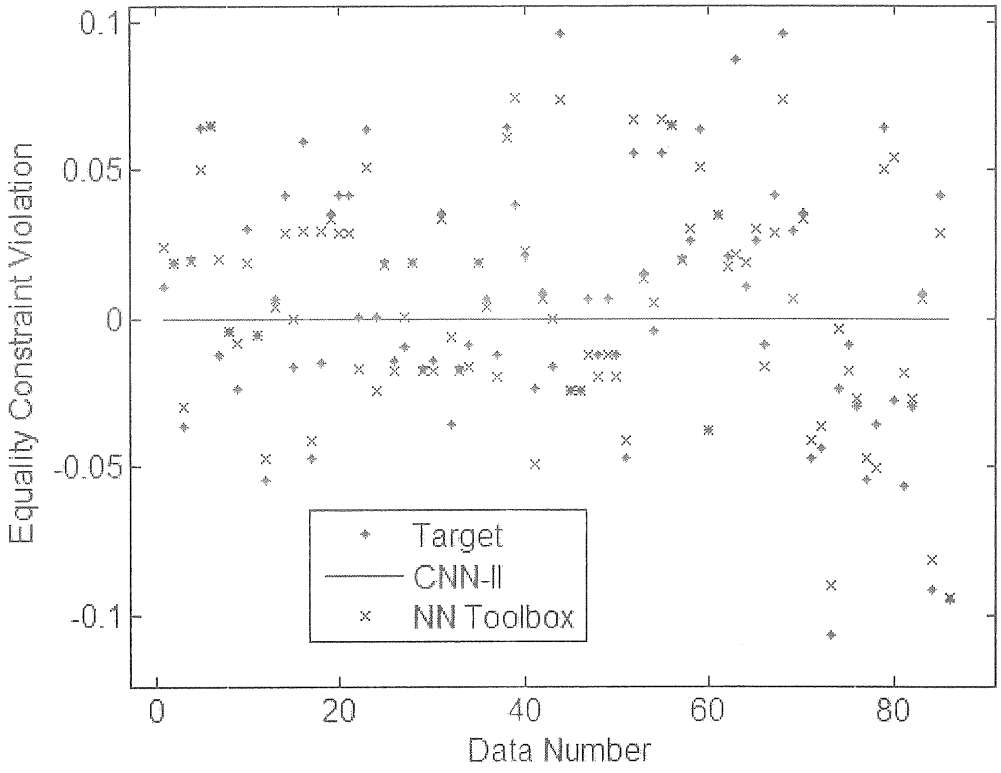


Figure 4.39. Violations of equality constraints for CNN-II-D for ternary VLE with measurement errors in the training phase.

The regression coefficients (R^2) of the NN Toolbox are 0.98, 0.99, and 0.98 for outputs x_1, x_2, and x_3 in the training set, respectively and 0.29, 0.32, and 0.25 for outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.58. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set very well, however its predictions are not good in the testing set, including many violations of the physical bounds ($0 \leq \mathbf{x} \leq 1$). On the other hand, the regression coefficients of CNN-II-D are 0.87, 0.92, and 0.88 for outputs x_1, x_2, and x_3 in the training set, respectively, and 0.28, 0.22, and –0.05 for the outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of CNN-II-D considering all data is 0.66. This shows that CNN-II-D performs good in the training set and better than NN Toolbox in the test set even under inequality and equality constraints, where violations of the physical bounds ($0 \leq \mathbf{x} \leq 1$ and $\sum x - 1=0$) are almost negligible. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The norm, $\|-\mathbf{x}_1\|$, of the first constraint's violations is 0.37 for CNN-II-D and 2.55 for the NN Toolbox. The norm, $\|\mathbf{x}_1 - 1\|$, of the second constraint's violations is zero

for CNN-II-D and zero for the NN Toolbox. The norm, $\|-x_2\|$, of the third constraint's violations is zero for CNN-II-D and zero for the NN Toolbox. The norm, $\|x_2 - 1\|$, of the fourth constraint's violations is 0.36 for CNN-II-D and 1.41 for the NN Toolbox. The norm, $\|-x_3\|$, of the fifth constraint's violations is 0.15 for CNN-II-D and 0.44 for the NN Toolbox. The norm, $\|x_3 - 1\|$, of the sixth constraint's violations is zero for CNN-II-D and 0.36 for the NN Toolbox. The norm $\|x_1 + x_2 + x_3 - 1\|$ for the equality constraint is 1.6 for CNN-II-D and 5.4 for the NN Toolbox.

The outputs of CNN-II-P (by penalty function technique) and NN-Toolbox are given in Figure 4.40. A horizontal line shows the constraint limit and vertical line separates the training (left) and testing (right) parts.

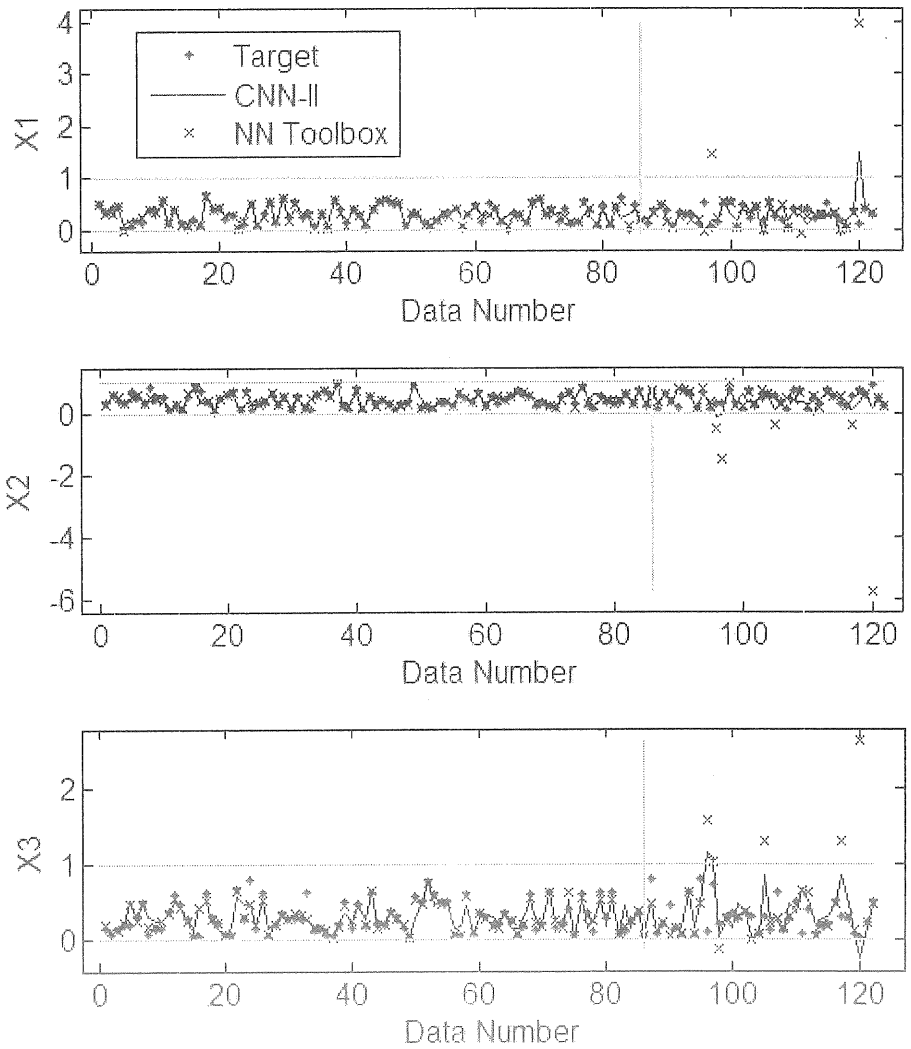


Figure 4.40. Outputs of CNN-II-P for ternary VLE with measurement errors.

The violations of equality constraints for CNN-II-P, as computed form of Equation 4.45, are shown in Figure 4.41.

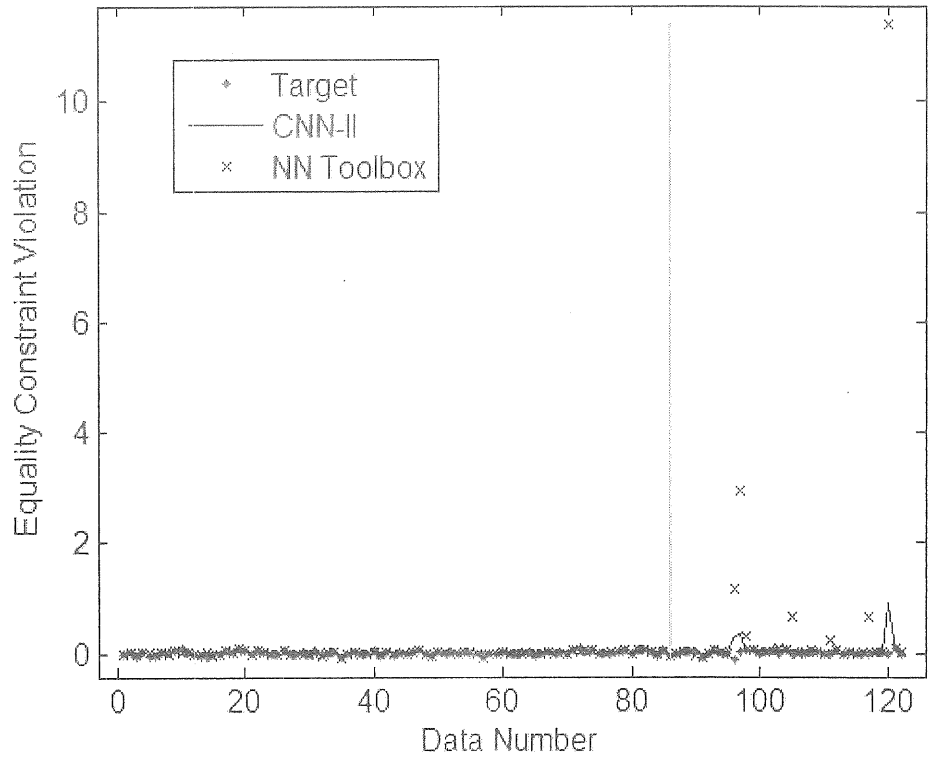


Figure 4.41. Violations of equality constraints for CNN-II-P for ternary VLE with measurement errors.

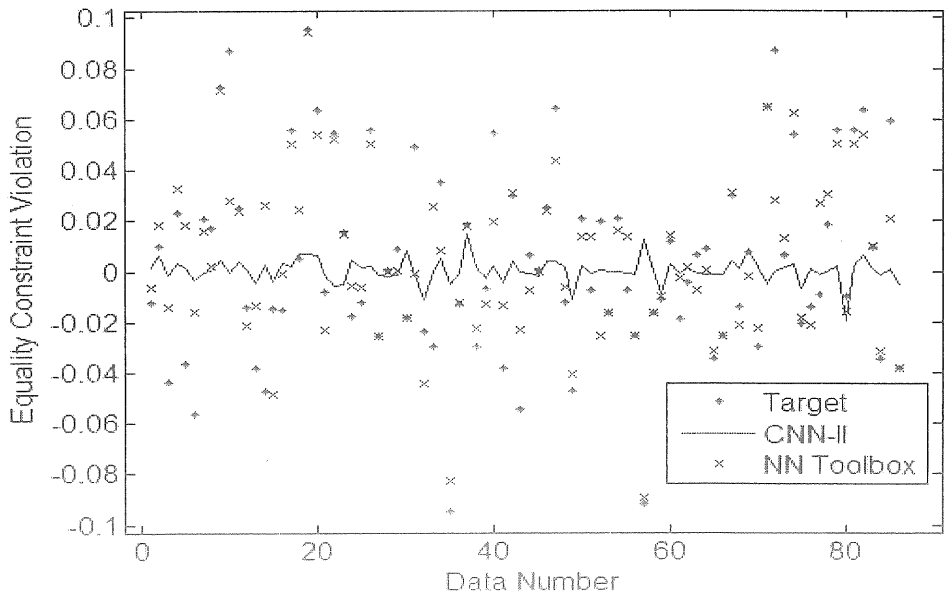


Figure 4.42. Violations of equality constraints for CNN-II-P for ternary VLE with measurement errors in the training phase.

Figure 4.42 illustrates the training part of the violations of equality constraints for CNN-II-P. The NN Toolbox fits the data very well in the training part but there are small violations in the equality constraints since there is a white noise added to the data. However, CNN methods can adapt to the noisy data very well, and at the same time, satisfy the requirement that mole fractions sum to one.

The regression coefficients (R^2) of the NN Toolbox are 0.95, 0.92, and 0.90 for outputs x_1, x_2, and x_3 in the training set, respectively and –0.13, –0.14, and 0.04 for outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.18. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set, however its predictions are not good in the testing set, including many violations of the physical bounds ($0 \leq \mathbf{x} \leq 1$) in all three mole fractions. On the other hand, the regression coefficients of CNN-II-P are 0.83, 0.81, and 0.82 for outputs x_1, x_2, and x_3 in the training set, respectively, and 0.09, 0.29, and 0.42 for the outputs x_1, x_2, and x_3 in the testing set, respectively. The overall regression coefficient of CNN-II-P considering all data is 0.62. This shows that CNN-II-P performs well both in the training and testing sets even under inequality and equality constraints where violations of the physical bounds ($0 \leq x \leq 1$ and $\sum x - 1 = 0$) are basically negligible. Furthermore, it is impossible to put constraints to `trainbr` of NN Toolbox. The norm, $\|-\mathbf{x}_1\|$, of the first constraint's violations is 0.14 for CNN-II-P and 0.09 for the NN Toolbox. The norm, $\|\mathbf{x}_1 - 1\|$, of the second constraint's violations is 0.48 for CNN-II-P and 2.98 for the NN Toolbox. The norm, $\|-x_2\|$, of the third constraint's violations is 0.18 for CNN-II-P and 6.02 for the NN Toolbox. The norm, $\|\mathbf{x}_2 - 1\|$, of the fourth constraint's violations is zero for CNN-II-P and zero for the NN Toolbox. The norm, $\|-\mathbf{x}_3\|$, of the fifth constraint's violations is 0.26 for CNN-II-P and 0.12 for the NN Toolbox. The norm, $\|\mathbf{x}_3 - 1\|$, of the sixth constraint's violations is 0.16 for CNN-II-P and 1.78 for the NN Toolbox. The norm $\|\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 - 1\|$ for the equality constraint is 1.02 for CNN-II-P and 11.86 for the NN Toolbox.

Figure 4.43 shows and compares the weights and biases of the NN Toolbox after training with `trainbr` and the optimal decision variables (weights and biases) obtained via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II.

It is seen that NN Toolbox weights and biases are more homogeneous than CNN methods. The presence of less homogeneous decision variables (NN weights and biases) in the CNN methods can be attributed to possible preferential activation of some of the neurons which primarily take role in satisfying the constraints.

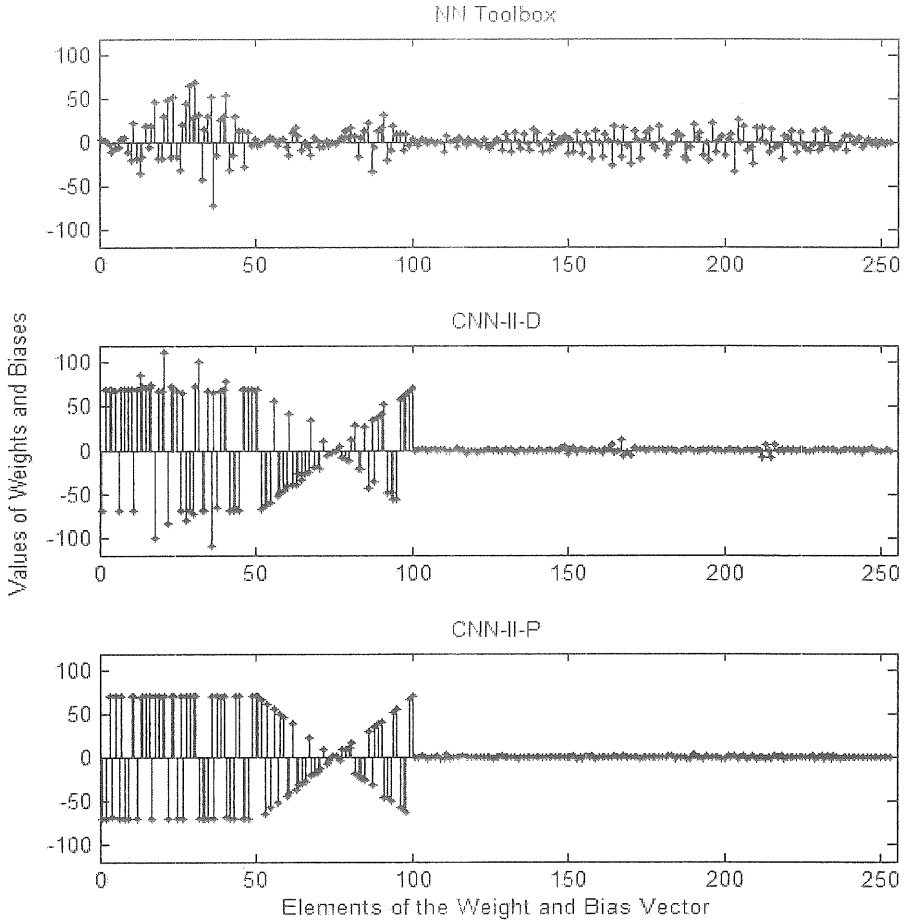


Figure 4.43. Weights and biases of the NN Toolbox and CNN-II for ternary VLE with measurement errors.

Figure 4.44 shows the progress of SSE as a function of epochs for the NN Toolbox and as a function of iterations for optimization via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. These show that CNN methods approach the minimum in small number of iterations comparable to the NN Toolbox. In Figure 4.44, the function values on the y-axis are the pure SSE values as reported by the NN Toolbox used with `trainbr` (Bayesian regularization), however, for CNN-II, the function values on the y-axes are the SSE values including the regularized term values given by Equation 2.26 in Section 2.5.2.

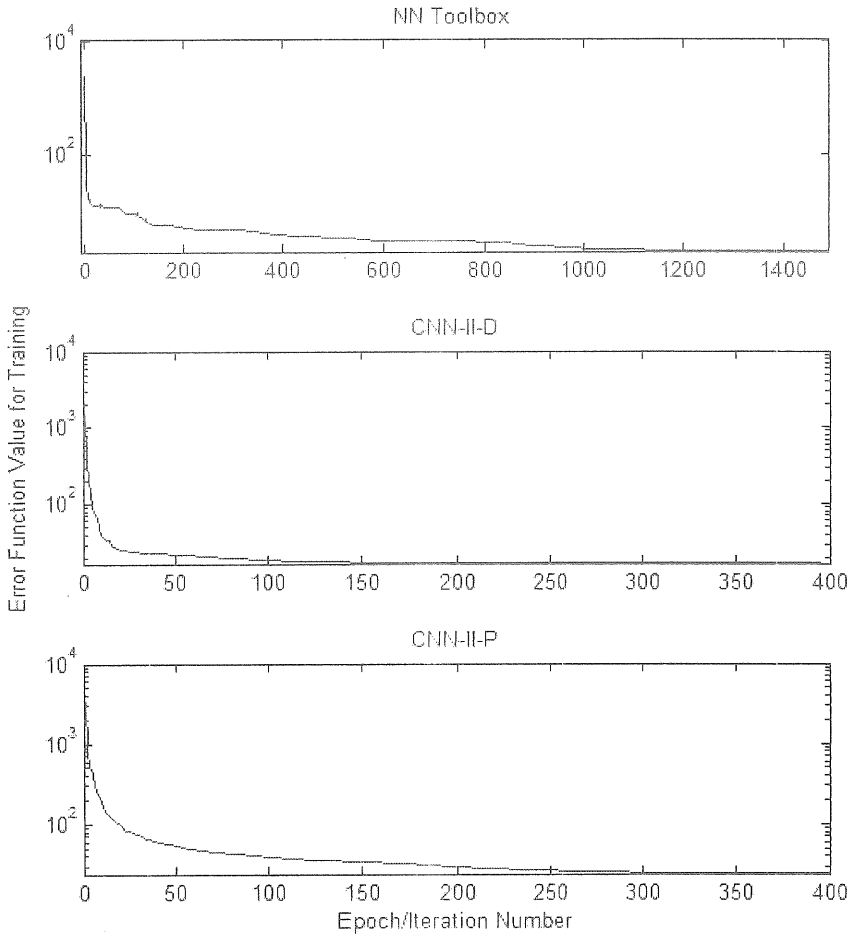


Figure 4.44. Progress of SSE in training by NN Toolbox and CNN-II for ternary VLE with measurement errors.

The CPU times as measured by the MATLAB's `tic` and `toc` functions are compared in Figure 4.45.

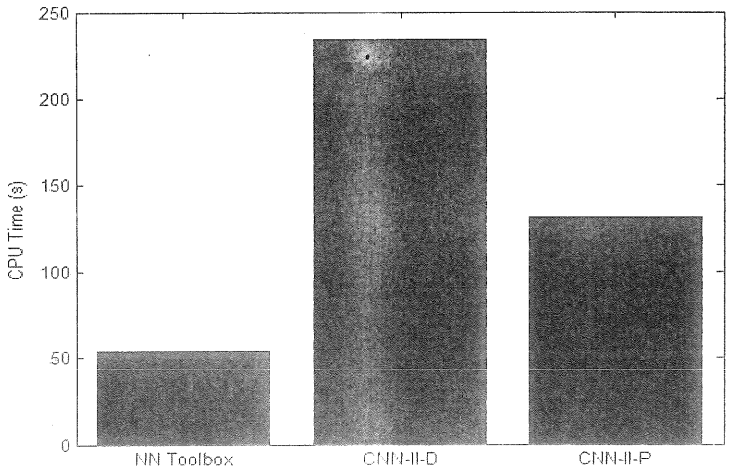


Figure 4.45. Training CPU times for ternary VLE with measurement errors.

Table 4.4. Comparison of NN Toolbox and CNN-II Method for ternary VLE with measurement errors.

	NN Toolbox	CNN-II-D	CNN-II-P
Number of Parameters	253	253	253
Number of Constraints	-	602	-
Training SSE, overall	0.28	2.42	3.64
Training SSE, x_1	0.06	0.69	0.89
Training SSE, x_2	0.11	0.85	1.50
Training SSE, x_3	0.10	0.87	1.24
Testing SSE, overall	24.2	11.3	8.7
Testing SSE, x_1	10.4	2.1	2.8
Testing SSE, x_2	8.9	4.4	2.8
Testing SSE, x_3	4.8	4.9	3.1
Training Overall Constraint Violation Norm	2.6×10^{-1}	1.7×10^{-5}	3.1×10^{-2}
Testing Overall Constraint Violation Norm	6.06	1.64	1.13
Training 1st Constraint ($\mathbf{g}_1$) Violation Norm	0	0	0
Testing 1st Constraint ($\mathbf{g}_1$) Violation Norm	2.55	0.37	0.14
Training 2nd Constraint ($\mathbf{g}_2$) Violation Norm	0	0	0
Testing 2nd Constraint ($\mathbf{g}_2$) Violation Norm	0	0	0.48
Training 3rd Constraint ($\mathbf{g}_3$) Violation Norm	0	0	0
Testing 3rd Constraint ($\mathbf{g}_3$)Violation Norm	0	0	0.18
Training 4th Constraint ($\mathbf{g}_4$) Violation Norm	0	0	0
Testing 4th Constraint ($\mathbf{g}_4$) Violation Norm	1.41	0.36	0
Training 5th Constraint ($\mathbf{g}_5$) Violation Norm	0	0	0
Testing 5th Constraint ($\mathbf{g}_5$) Violation Norm	0.43	0.15	0.26
Training 6th Constraint ($\mathbf{g}_6$) Violation Norm	0	0	0
Testing 6th Constraint ($\mathbf{g}_6$) Violation Norm	0.36	0	0.16
Training 7th Constraint ($\mathbf{h}$) Violation Norm	3.4×10^{-1}	2.7×10^{-5}	4.4×10^{-2}
Testing 7th Constraint ($\mathbf{h}$) Violation Norm	5.38	1.61	1.01
Training CPU (s)	53.5	234.6	131.2
Number of Epochs/Iterations	1500	395	417

Table 4.4 summarizes the comparison among NN Toolbox, CNN-II-D (direct constraint implementation), and CNN-II-P (penalty function implementation). Since the initialization of weights and biases are done randomly, the results may change from run to run; the NN Toolbox results listed in the table correspond to the run the results of which are given in Figure 4.37.

Since there are inequality and equality constraints on the outputs x_1, x_2, and x_3, the SSE values given in the above table for the outputs x_1, x_2, and x_3 should be appraised with care. However, from Figure 4.37 and Figure 4.40, it is visually seen that values of x_1 above zero and below one (values of x_1 in the feasible region), values of x_2 above zero and below one (values of x_2 in the feasible region), and values of x_3 above zero and below one (values of x_3 in the feasible region) fit to data points very well.

In summary, NN Toolbox cannot handle constraints. It should be noted that for NN Toolbox and CNN-II, the initialization of weights and biases are done randomly according to the NW algorithm as explained in Section 2.4. Therefore, the SSE, constraint violation norm, and CPU values change from run to run. Therefore, the figures, SSE, and constraint violation norms, and CPU values should be accepted only as representative values.

As seen in Figure 4.39 and Figure 4.42, white noises are added to the data so that the physical equality constraint ($\sum x=1$) is not contained in the data, in other words, the external information that the mole fractions must sum to one is not embedded into the data. Since it is impossible to implement constraints in the NN Toolbox, it performs well in the training phase and fits the noisy (wrong) data very well, however without satisfying the equality constraint. On the other hand, the external information (implicit constraints that $0 \leq \mathbf{x} \leq 1$ and $\sum x - 1 = 0$) are forced through imposed constraints during training phase in the CNN methods. This way, noise reconciliation and external information embedding are achieved simultaneously during the training in the CNN methods, as verified in Figure 4.39 and Figure 4.42.

4.4. Propylene Glycol Production in CSTR

In this example (Fogler, 2008), propylene glycol production in a continuously stirred tank reactor with a cooling coil is considered. Propylene glycol is produced by the hydrolysis of propylene oxide:

$$C_3H_6O + H_2O \xrightarrow{H_2SO_4} C_3H_8O_2 \qquad (4.54)$$

The reaction takes place readily at room temperature when catalyzed by sulfuric acid. Feed is the 2500 lb/hr (43.04 lb mol/hr) of propylene oxide (PO) to the reactor. The feed stream consists of (1) an equivolumetric mixture of propylene oxide (46.62 ft^3/hr) and methanol (46.62 ft^3/hr), and (2) water containing 0.1 wt% H_2SO_4. The volumetric flow rate of water is 233.1 ft^3/hr, which is 2.5 times the methanol-PO flow rate. The corresponding molar flow rates of methanol and water are 71.87 and 802.8 lb mol/hr, respectively. The water / propylene-oxide / methanol mixture undergoes a slight decrease in volume upon mixing (approximately 3%), but can be neglect in the calculations. The temperature of both feed streams is 58°F prior to mixing, but there is an immediate 17°F temperature rise upon mixing of the two feed streams caused by the heat of mixing. The entering temperature of all feed streams is thus taken to be 75°F. The process is depicted in Figure 4.46.

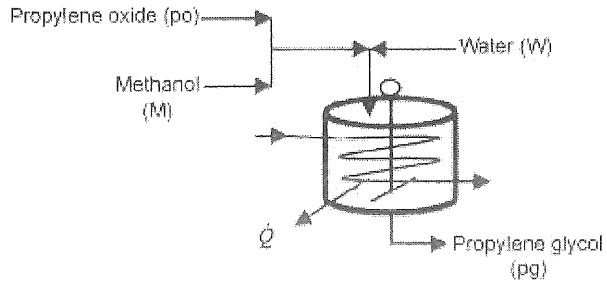


Figure 4.46. Propylene Glycol production in CSTR.

Propylene oxide is a rather low boiling substance (boiling point at one atmosphere is 93.7°F). For the mixture used in the problem, the operating temperature of the reactor cannot exceed 125°F (585°R) or the reactor will lose too much propylene oxide by vaporization through the vent system.

A cooling coil has been located in equipment storage for use in the hydration of propylene oxide. The cooling coil has 40 ft^2 of cooling surface and the cooling water flow

rate inside the coil is sufficiently large that a constant cooling temperature. A typical overall heat transfer coefficient for such a coil is 100 Btu/h-ft^2-°F.

The steady-state model of the reactor can be described by the following two nonlinear algebraic equations, which can be derived from the mole and energy balances including reaction rate expression:

$$f_1(X,T) = X - \frac{(2.084\times10^{12})\exp\left(-\frac{16306}{T}\right)}{1+(2.084\times10^{12})\exp\left(-\frac{16306}{T}\right)} = 0 \quad (4.55)$$

$$f_2(X,T) = X - \frac{403.3(T-535)+92.9(T-T_a)}{36400+7(T-528)} = 0 \quad (4.56)$$

where, X is the conversion and T is the temperature in °R, 528°R is the reference temperature and 535°R is the entering stream temperature.

There are three variables considering the material and energy balances and two equations. If the cooling temperature (T_a) is set to a fixed value, the degrees of freedom become zero and the solution of conversion and temperature can be obtained corresponding to the given cooling temperature. The input-output data for this system can be generated as follows: The effect of cooling temperature on the conversion and temperature is obtained by changing the cooling temperature and by recording the conversion (X) and temperature (T) values from the simultaneous solution of Equation 4.55 and Equation 4.56.

Due to the vaporization of propylene oxide for temperatures above 585°R (125°F), the model equations (Equation 4.55 and Equation 4.56) cannot represent the real system above vaporization temperature of propylene oxide. Considering this constraint, the data are generated for T_a values below 555°R, for which the conversions obtained fall below the limit of 0.63 at reactor temperature of 585°R. The data which contains exact information about the system defined by Equation 4.55 and Equation 4.56 can be learned very well by the NN Toolbox. However, in real life applications, there is measurement errors associated with the data, mostly in the form of white noise. In order to simulate this situation, white noise is added to the actual data. Thus, in the modified data set used in this section, the data set does not satisfy the Equation 4.55 and Equation 4.56 which are the mass and

energy balances of the system. The NN Toolbox fits the data containing measurement errors very well, yet, since it does not (it cannot) care about the mass and energy balances, it cannot provide reliable outputs (X and T). However, mass and energy balances can be enforced by training with CNN methods developed in this thesis work.

The measurement errors represented by normal random numbers (with mean zero and standard deviations of 1.8, 0.001, and 1.9 for the T_a, X, and T data, respectively) are added to the data, such that the first constraint (Equation 4.55) violations range from –0.064 to 0.075 and the second constraint (Equation 4.56) violations range from –0.034 to 0.047.

Thus, the NN is configured to reflect the relation between input (cooling temperature, T_a) and outputs (X and T). Mass and energy balances are considered as the equality constraints. By imposing constraints during training, CNNs can learn this external information and can carry this information learned in the training to forecasting phase.

The NN model is depicted in Figure 4.47.

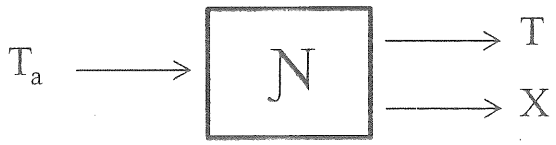


Figure 4.47. NN model of Propylene Glycol production.

The problem can be described as:

$$z = (s, w, x) \tag{4.57}$$

where,

$$z = [X \quad T] \tag{4.58}$$

The constraints of the process $h((s, w, x), x) = 0$ are defined as:

$$h_1 = X - \frac{(2.084\times10^{12})\exp\left(-\frac{16306}{T}\right)}{1+(2.084\times10^{12})\exp\left(-\frac{16306}{T}\right)} = 0 \tag{4.59}$$

$$h_2 = X - \frac{403.3(T-535)+92.9(T-T_a)}{36400+7(T-528)} = 0 \tag{4.60}$$

The training of the NN with regularization and direct implementation of constraints can be given as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \delta\|w\| \tag{4.61}$$

$$s.t. \quad h = 0 \tag{4.62}$$

This optimization problem of NN training was performed by CNN-II, with/without penalty function technique (CNN-II-P/CNN-II-D). The NN structure consists of one input (one input layer), 10 neurons in the single hidden layer, where all neurons have hyperbolic tangent sigmoid functions, and two neurons in the output layer all having linear activation functions. The total number of weights and biases are 30 and 12, respectively. These make the number of decision variables in the optimization problem 42.

The data have 111 rows. 50% of the data (56 data points) were used in the training step, and the remaining 50% (55 data points) were used in the testing step. The training options for the direct implementation of constraints, **g**, by CNN-II-D, and for the unconstrained NN Toolbox are the optimizer selection (`fmincon` for CNN-II-D, `trainbr` for NN Toolbox), and regularization parameter (δ = 0.0001). Regularization was used to avoid overlearning. As the error function, SSE was used since `trainbr` performs only with SSE.

For CNN-II-D, initial guesses for the decision variables (NN weights and biases, shown as **w**) were obtained from the custom made initialization function as using NW initialization method that was explained in Section 3.5. With 56 data points, the total number of inequality constraints (Equation 4.59 and Equation 4.60) imposed on output **X** becomes 112. Therefore, the optimization problem given by Equation 4.61 above has 42 decision variables and 112 inequality constraints in the form Equation 4.62.

The maximum number of epochs allowed for the `trainbr` of the NN Toolbox was set to 1000. Other optional parameters of the `trainbr` of the NN Toolbox were set such

that the training will not usually stop before maximum epochs allowed (e.g., min gradient = 1×10^{-12}, max mu = 1×10^{10}). For CNN-II, the optimization termination criteria were set as TolX = 1×10^{-5}, TolFun = 1×10^{-9}, TolCon = 1×10^{-2}, and MaxIter = 1000.

The outputs of CNN-II-D (by direct constraint implementation) and NN-Toolbox are given in Figure 4.48. The vertical line separates the training (left) and testing (right) parts.

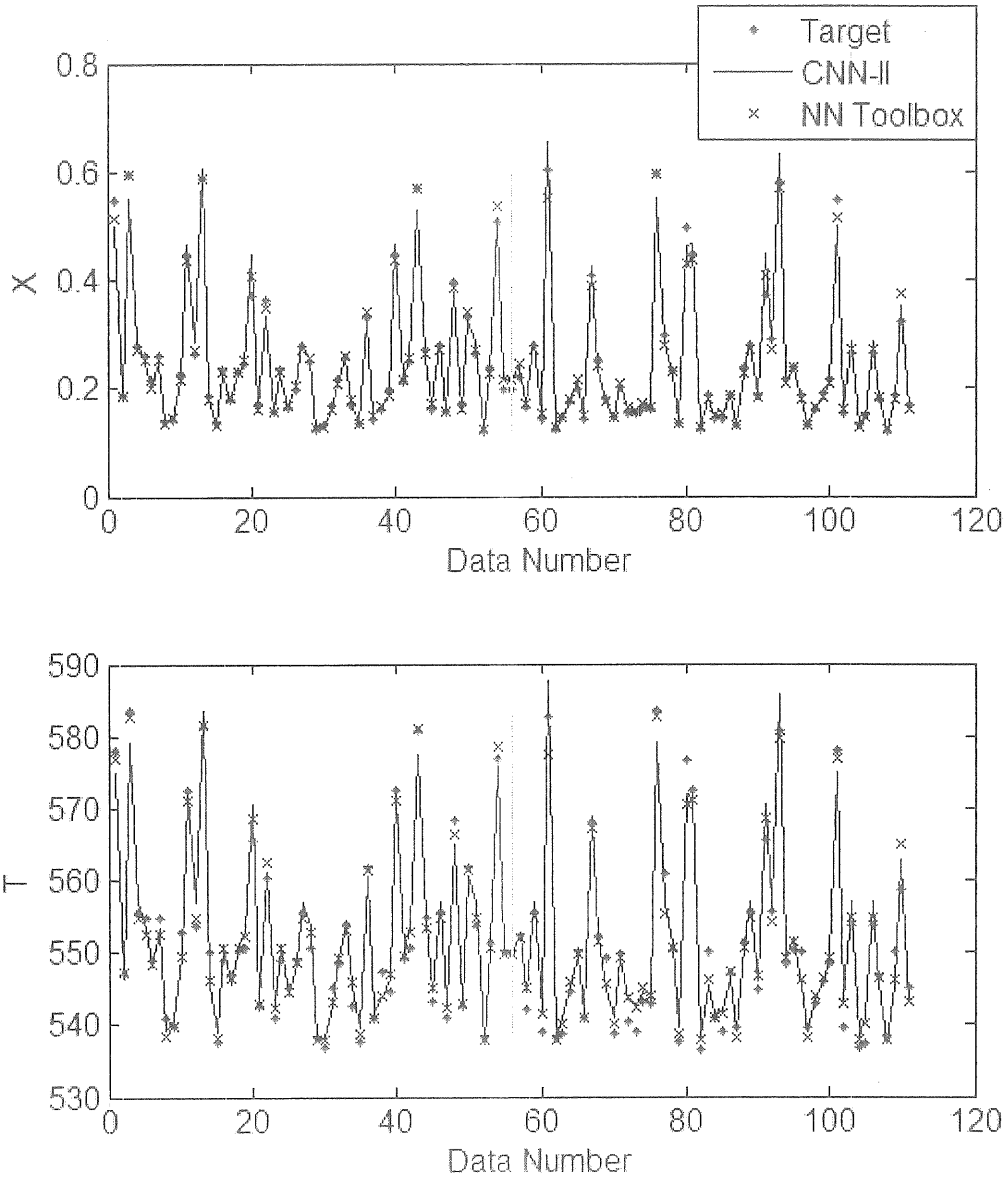


Figure 4.48. Outputs of CNN-II-D for Propylene Glycol production.

The violations of equality constraints for CNN-II-D are shown in Figure 4.49.

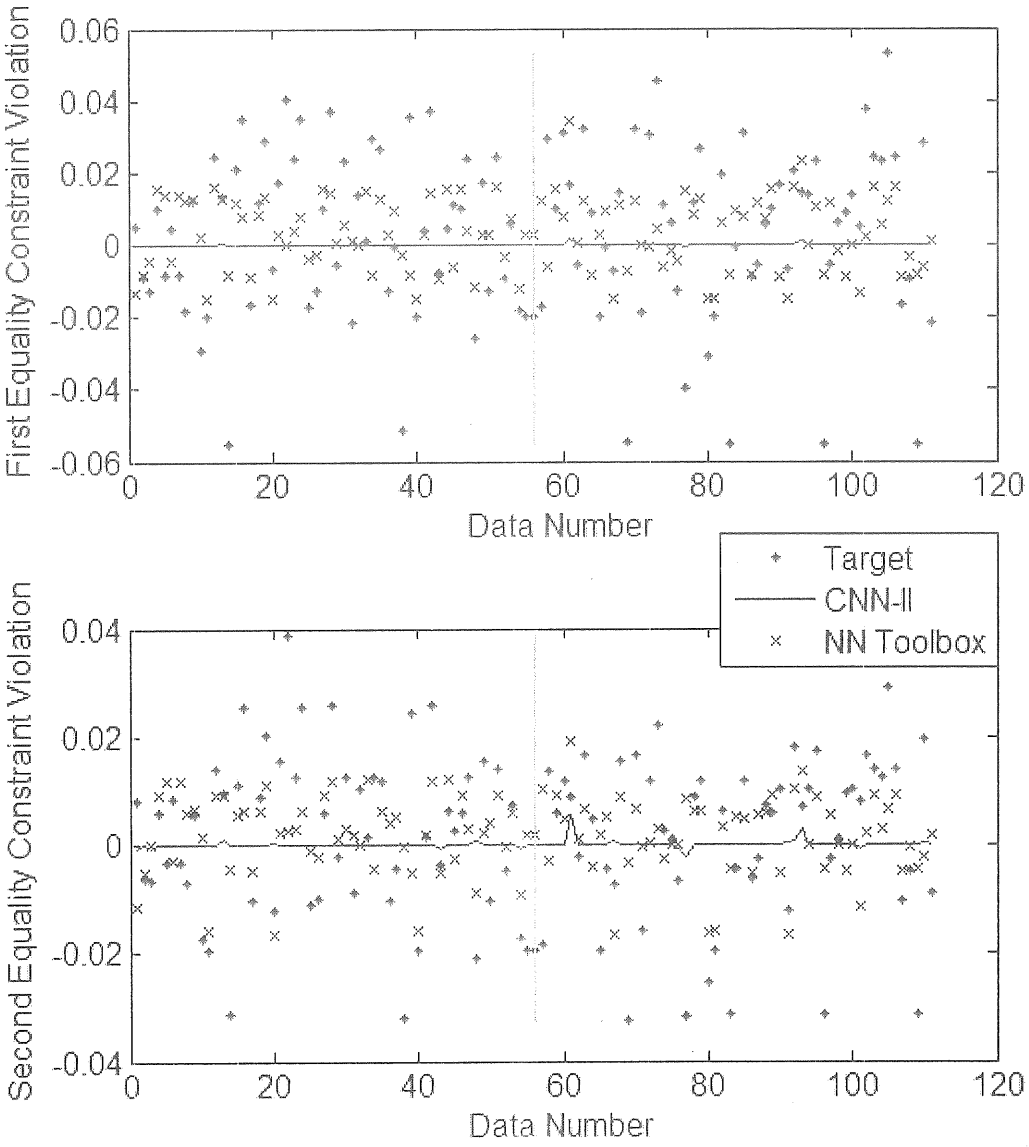


Figure 4.49. Violations of equality constraints for CNN-II-D for Propylene Glycol production.

The regression coefficients (R^2) of the NN Toolbox are 0.99 for outputs X and T in the both training and testing set. The overall regression coefficient of NN Toolbox considering all data is also 0.99. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set very well, and its predictions are also perfect in the testing set, however these predictions violate equality constraints ($\mathbf{h}_1$ and $\mathbf{h}_2$). On the other hand, the regression coefficients of CNN-II-D are 0.99 and 0.98 for outputs X and T in the both training and testing sets, respectively. The overall regression coefficient of CNN-II-D considering all data is 0.99. This shows that CNN-II-D performs well both in

the training and testing sets even under equality constraints, where violations of the equality constraints are almost zero. Furthermore, it is impossible to put equality constraints to `trainbr` of NN Toolbox. The norm of $\mathbf{h}_1$ of the constraint violations is 0.002 for CNN-II-D and 0.11 for the NN Toolbox. The norm of $\mathbf{h}_2$ of the constraint violations is 0.007 for CNN-II-D and 0.08 for the NN Toolbox.

The constrained NN problem can also be solved by using the penalty function technique. The optimization problem of constrained NN training with penalty function technique can be described as:

$$\min_{w} f(w) = SSE(\ (s, w, x^t), y^t) + \mu\|h\| + \delta\|w\| \tag{4.63}$$

where,

$$h = 0 \tag{4.64}$$

The NN structure is the same as in the direct constraint implementation case. The training options for the indirect implementation of constraints (penalty function technique), by CNN-II-P, and for the unconstrained NN Toolbox are the optimizer selection (`fminunc` for CNN-II-P, `trainbr` for NN Toolbox), regularization parameter (δ = 0.0001), and penalty parameter (μ = 8). This optimization problem has also 42 decision variables, however no constraints.

For CNN-II-P, initial guesses for the decision variables were obtained from the custom made initialization function by using NW initialization method that was explained in Section 3.5 as in the direct constraint implementation case. All other optimization (`fmincon`) related optional parameters are the same as in the direct constraint handling case described above.

The outputs of CNN-II-P (by penalty function technique) and NN-Toolbox are given in Figure 4.50. The vertical line separates the training (left) and testing (right) parts.

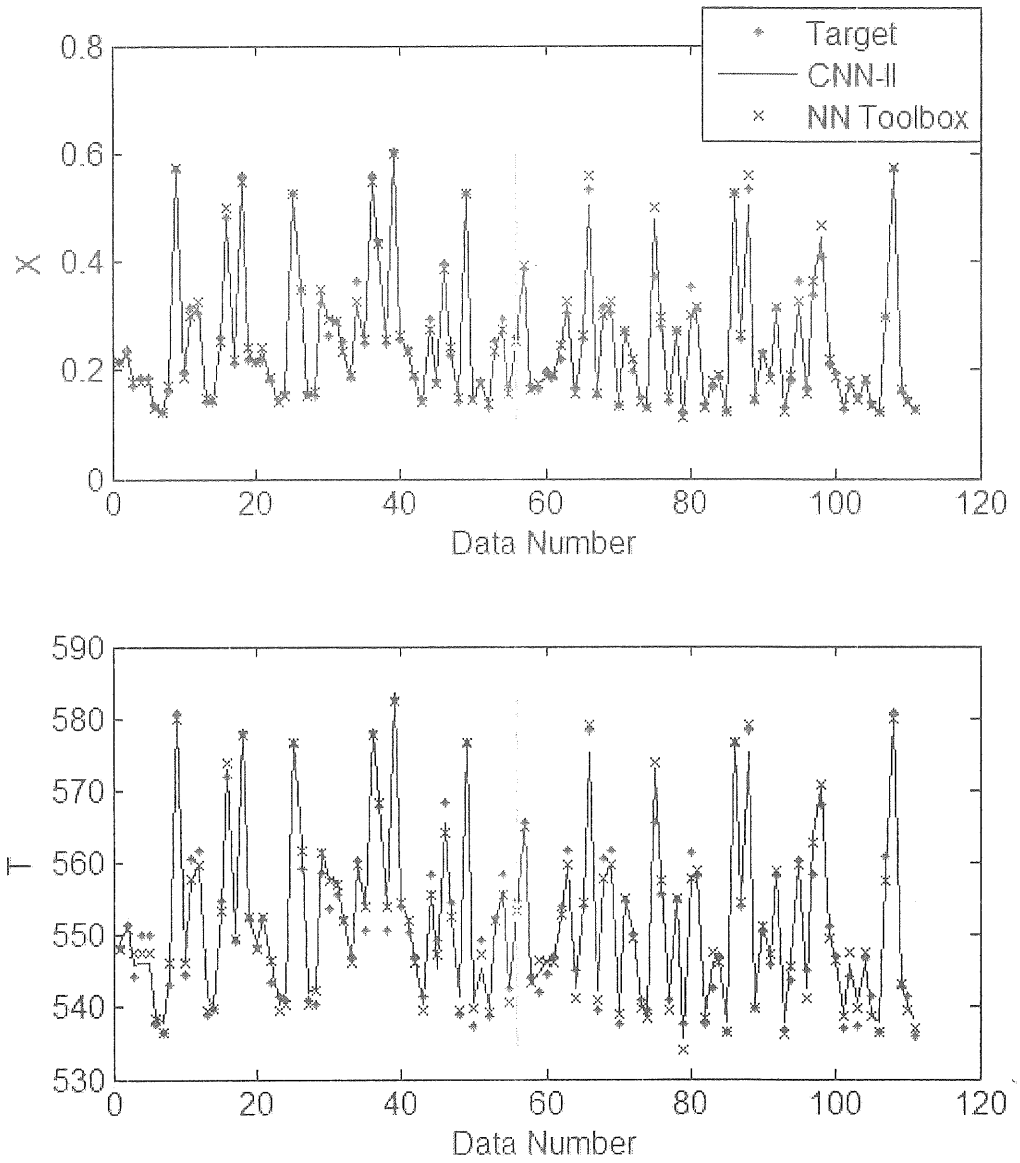


Figure 4.50. Outputs of CNN-II-P for Propylene Glycol production.

The regression coefficients (R^2) of the NN Toolbox are 0.99 for outputs X and T in the training set and 0.98 for outputs X and T in the testing set. The overall regression coefficient of NN Toolbox considering all data is 0.99. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set very well, and its predictions are also perfect in the testing set, however these predictions violate equality constraints ($\mathbf{h}_1$ and $\mathbf{h}_2$). On the other hand, the regression coefficients of CNN-II-P are 0.99 and 0.99 for outputs X and T in the training set, respectively, and 0.98 and 0.98 for the outputs X and T in the testing set, respectively. The overall regression coefficient of CNN-II-P considering all data is 0.99. This shows that CNN-II-P performs well both in the training and testing sets even under inequality constraints, where violations of the equality

constraints are almost zero. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The norm of $\mathbf{h}_1$ of the constraint violations is 0.025 for CNN-II-P and 0.15 for the NN Toolbox. The norm of $\mathbf{h}_2$ of the constraint violations is 0.013 for CNN-II-P and 0.093 for the NN Toolbox.

The violations of equality constraints for CNN-II-P are shown in Figure 4.51.

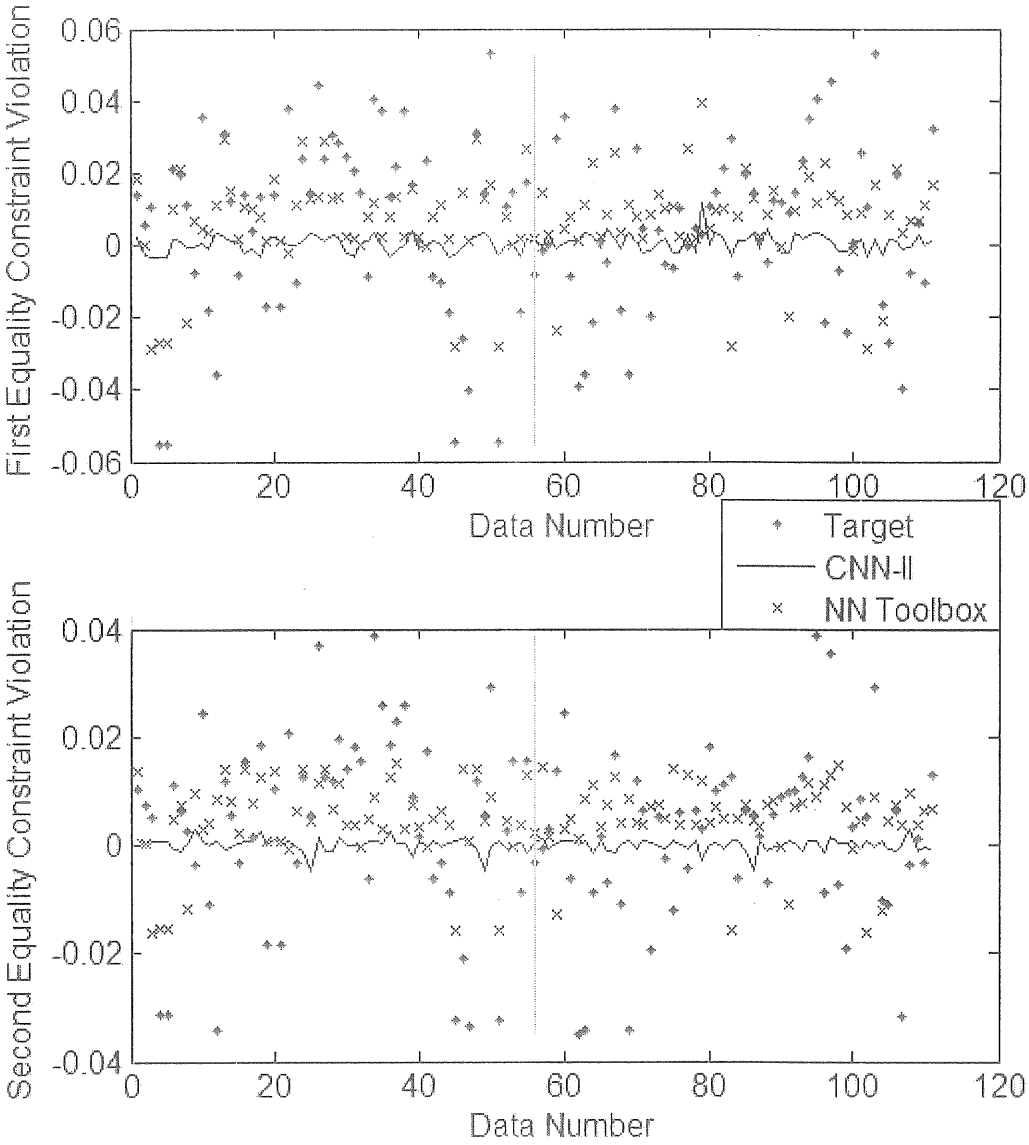


Figure 4.51. Violations of equality constraints for CNN-II-P for Propylene Glycol production.

Figure 4.52 shows and compares the weights and biases of the NN Toolbox after training with `trainbr` and the optimal decision variables (weights and biases) obtained

via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. It is seen that NN Toolbox weights and biases are more homogeneous than CNN methods. The presence of less homogeneous decision variables (NN weights and biases) in the CNN methods can be attributed to possible preferential activation of some of the neurons which primarily take role in satisfying the constraints.

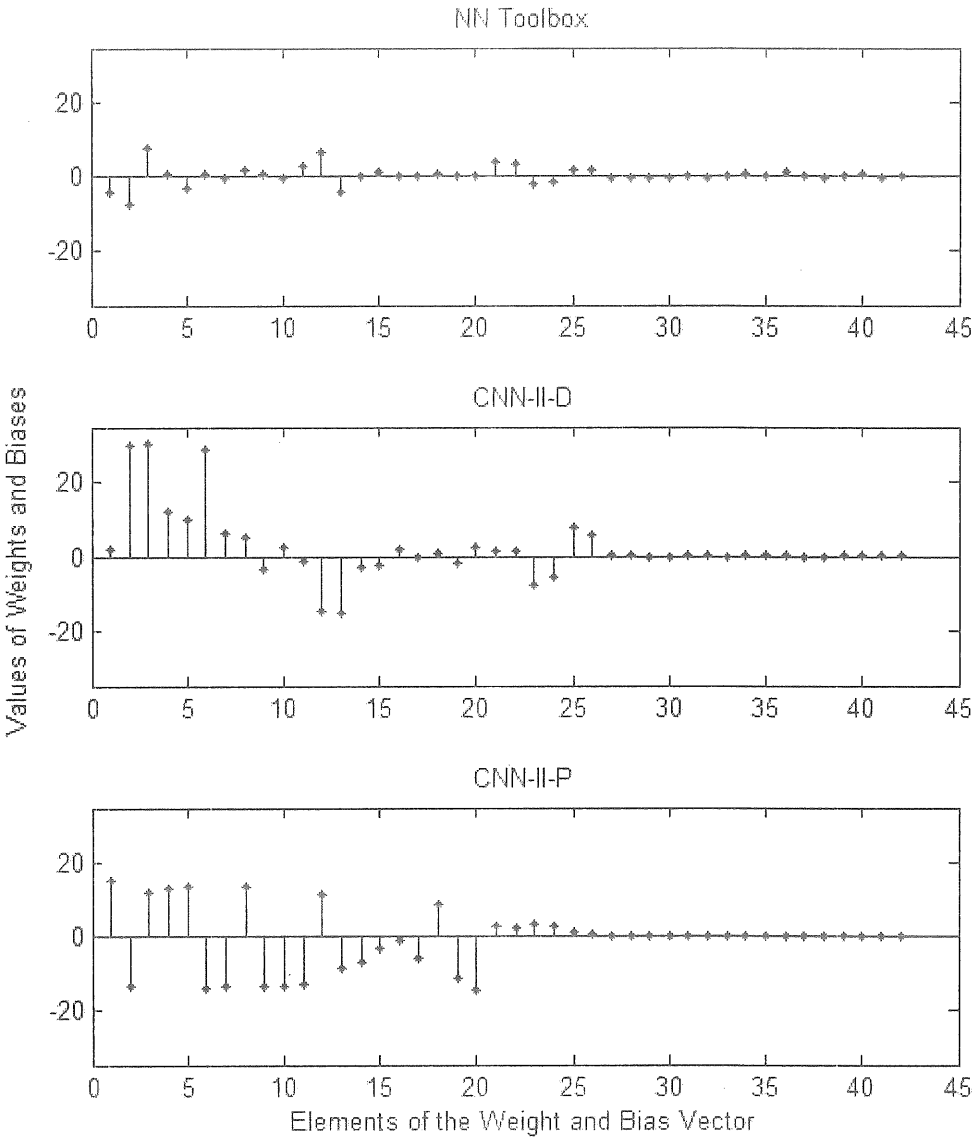


Figure 4.52. Weights and biases of the NN Toolbox and CNN-II for Propylene Glycol production.

Figure 4.53 shows the progress of SSE as a function of epochs for the NN Toolbox and as a function of iterations for optimization via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. These show that CNN methods approach

the minimum in small number of iterations comparable to the NN Toolbox. In Figure 4.53, the function values on the y-axis are the pure SSE values as reported by the NN Toolbox used with `trainbr` (Bayesian regularization), however, for CNN-II, the function values on the y-axes are the SSE values including the regularized term values given by Equation 2.26 in Section 2.5.2.

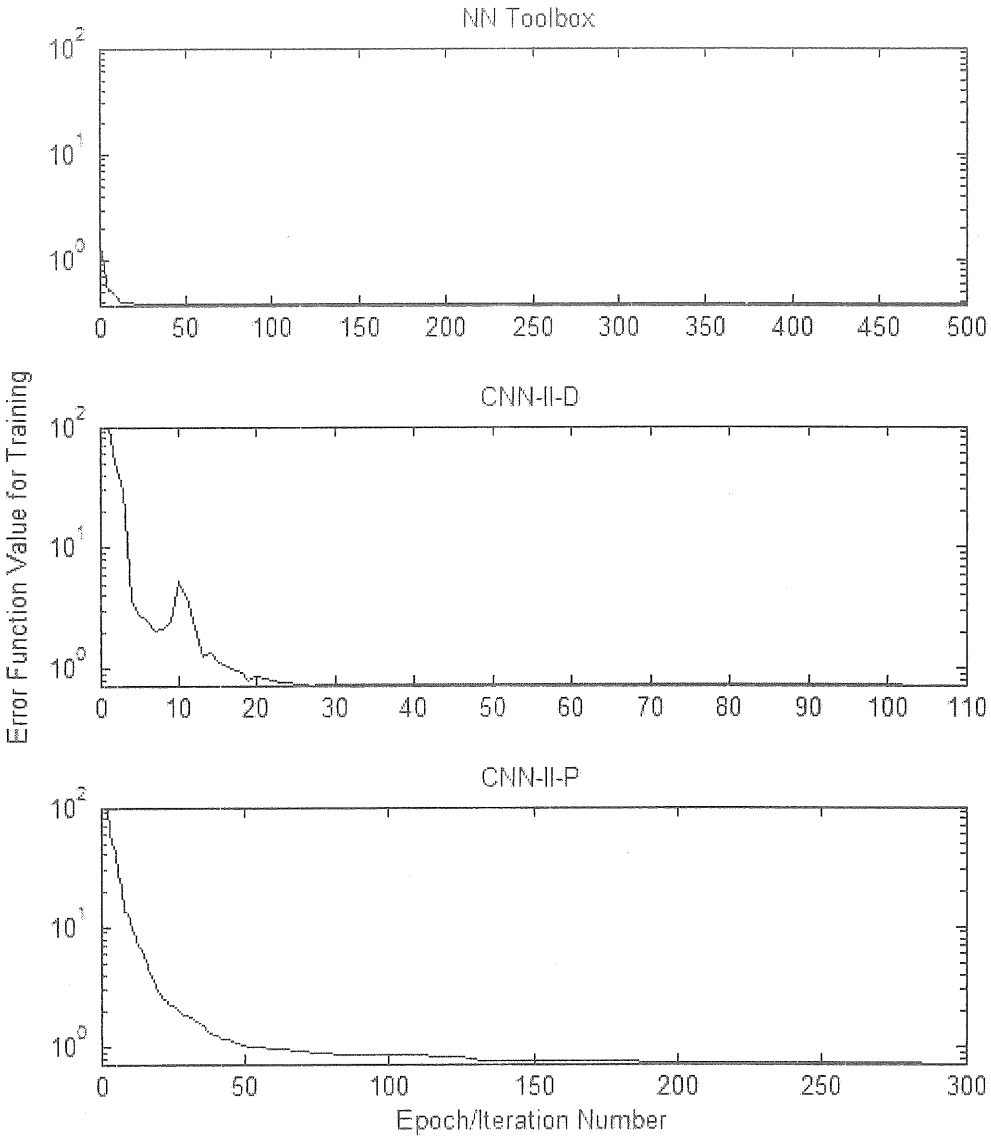


Figure 4.53. Progress of SSE in training by NN Toolbox and CNN-II for Propylene Glycol production.

The CPU times as measured by the MATLAB's `tic` and `toc` functions are compared in Figure 4.54.

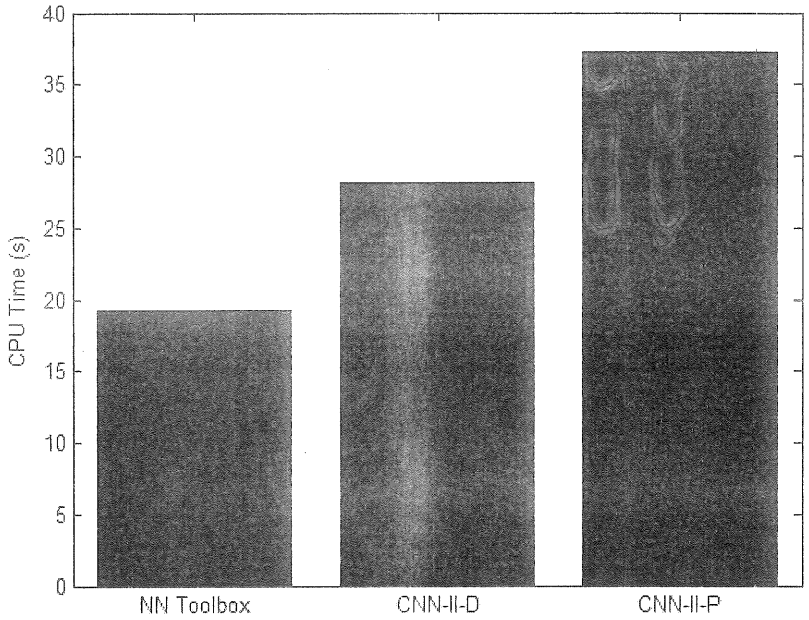


Figure 4.54. Training CPU times for Propylene Glycol production.

Table 4.5. Comparison of NN Toolbox and CNN-II method for Propylene Glycol production.

	NN Toolbox	CNN-II-D	CNN-II-P
Number of Parameters	42	42	42
Number of Constraints	-	56	-
Training SSE, overall	141.9	237.3	227.2
Training SSE, X	0.006	0.017	0.007
Training SSE, T	141.9	237.3	227.2
Testing SSE, overall	310.2	378.5	260.6
Testing SSE, X	0.016	0.021	0.024
Testing SSE, T	310.2	378.5	260.6
Total Constraint ($\mathbf{h}$) Violation Norm	0.136	0.007	0.028
Training Constraint ($\mathbf{h}_1$) Violation Norm	0.0748	0.0006	0.0156
Testing Constraint ($\mathbf{h}_1$) Violation Norm	0.0824	0.0022	0.0196
Training Constraint ($\mathbf{h}_2$) Violation Norm	0.0562	0.0006	0.0156
Testing Constraint ($\mathbf{h}_2$) Violation Norm	0.0565	0.0022	0.0196
Training CPU (s)	19.2	28.1	37.2
Number of Epochs/Iterations	503	102	284

Table 4.5 summarizes the comparison among NN Toolbox, CNN-II-D (direct constraint implementation), and CNN-II-P (penalty function implementation). Since the initialization of weights and biases are done randomly, the results may change from run to run; the NN Toolbox results listed in the table correspond to the run the results of which are given in Figure 4.48.

Since there is equality constraint on the outputs, the SSE values given in the above table for the output X should be appraised with care. However, from Figure 4.48 and Figure 4.50, it is visually seen that values of X and T (values of X and T in the feasible region) fit the data points very well.

In summary, NN Toolbox cannot handle constraints. It should be noted that for NN Toolbox and CNN-II, the initialization of weights and biases are done randomly according to the NW algorithm as explained in Section 2.4. Therefore, the SSE, constraint violation norm, and CPU values change from run to run. Therefore, the figures, SSE, and constraint violation norms, and CPU values should be accepted only as representative values.

In this problem, the data are obtained from the solutions of the algebraic equations (mass and energy balance of the system) for different T_a values. The noise is added to the data in order to simulate real life situation. The NN trained by NN Toolbox can only learn the information contained in the solutions of these algebraic equations and NN Toolbox constructs an empirical equation that implicitly describes only the mass and energy balance equations. However, the NN trained by CNN methods learns the information contained in the mass and energy balances together with the external information imposed as equality constraints. Since the data contain noise, NN Toolbox cannot satisfy the mass and energy balances. The model based learning can be accomplished with the CNN methods by employing the mass and energy balances as equality constraints.

4.5. Runaway in a Packed Bed Reactor

This problem (Fogler, 2008) is an example of runaway reaction in Packed Bed Reactor (PBR). The reaction of oxidation of Naphthalene to Phthalic Anhydride is given as:

$$2Naphthalene + 9O_2 \rightarrow 2Phthalic\ Anhydride + 4CO_2 + 4H_2O \qquad (4.65)$$

The oxidation is to be carried out in a PBR with cooling. The coolant temperature (T_a) is constant along the length of the reactor. The pressure is 1.00 atm and the partial pressure of naphthalene is 0.01 atm. The entering temperature is 625 K. The reactor is depicted in Figure 4.55.

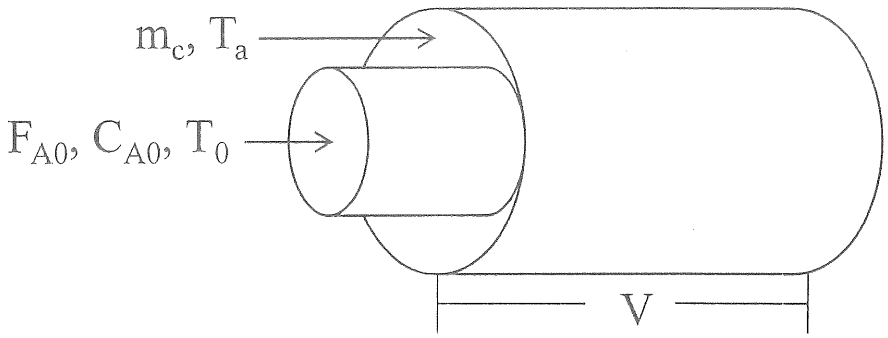


Figure 4.55. PBR.

where, m_c is cooling fluid mass flow, T_a is cooling fluid temperature, F_{A0} is mass flow rate of entering reactant, C_{A0} is the inlet concentration reactant, and T_0 is the inlet temperature. The temperature and concentration profiles along the reactor are obtained by solving the material and energy balances simultaneously. The material and energy balance differential equations are given as:

$$\frac{dC_A}{dV} = \frac{r_a}{\vartheta_0} \qquad (4.66)$$

$$\frac{dT}{dV} = \frac{Q_g - Q_r}{\sum F_i C_{P\,i}} \qquad (4.67)$$

$$r_a = AkC_A \qquad (4.68)$$

where, T and C_A are the reactor temperature and concentration of reactant as functions of the independent variable, the reactor volume V (axial position times the cross-sectional area), which varies between zero (at the inlet location) and total volume (at the effluent location), Q_g is the amount of heat generated in the reactor, Q_r is the amount of heat removed from the reactor, F is the mass flow of materials, C_P is the heat capacity of the materials, r_a is the reaction rate, A is a constant multiplier for the reaction rate equation, k

is the rate constant, and ϑ_0 is the entering volumetric flow rate. The detailed derivation and solution of these coupled Ordinary Differential Equations (ODEs) can be found in Fogler (2008).

The temperature runaway phenomenon takes place in the reactor with respect to the inlet concentration of reactant and feed temperature. The problem is designed to reflect this runaway behavior. The exit temperature and conversion data are produced for various inlet concentration data at fixed inlet temperature of 625 K. The critical value of inlet concentration is 0.01825 kmol/m^3 at this temperature. If the inlet concentration exceeds the critical point, the runaway reaction occurs in the reactor. When the mass and energy balance equations of reactor are solved, impracticable conversions that reach 100% are obtained above the critical inlet concentration. The conversion and exit temperature are plotted against the input concentration in Figure 4.56.

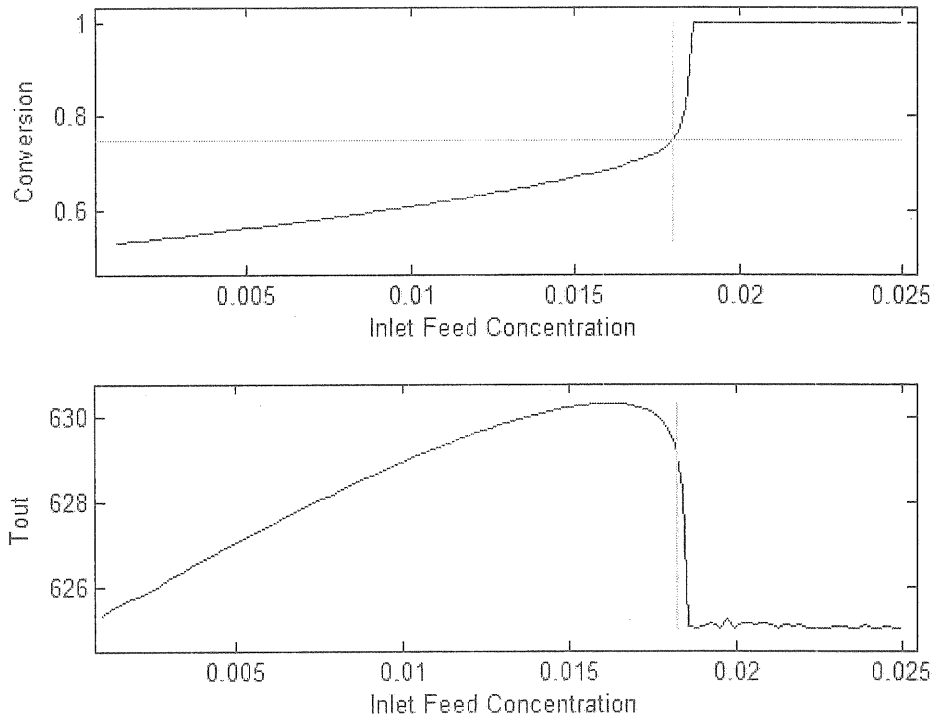


Figure 4.56. Input and output data for PBR.

The vertical line in the above figure stands for the critical initial concentration (0.01825 kmol/m^3) and the data in the right hand side of this line are impracticable runaway data. The horizontal line stands for the conversion values in the runaway behavior. If the conversion reaches 100%, the reactant in the reactor is depleted, and

therefore, the temperature in the reactor decreases to coolant temperature as the reaction stops. The outlet temperature does not exceed the coolant temperature in the runaway behavior. If the data including such runaway data are trained by the NN Toolbox algorithms, the impracticable output values of conversion are also learned. On the other hand, these situations can be eliminated by CNN methods by imposing suitable constraints. Therefore, a NN problem is formed to reflect the relation between input concentration and outputs of exit temperature and conversion. The NN model is depicted in Figure 4.57.

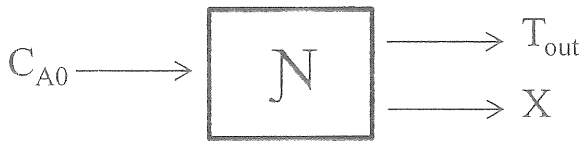


Figure 4.57. NN model of PBR.

The problem can be described as:

$$z = (s, w, x) \tag{4.69}$$

where,

$$z = [X \quad T_{out}] \tag{4.70}$$

The constraints of the problem are the conversion limit of 0.75 and the outlet temperature limit of 625.3 K that is barely above the coolant temperature at which the critical point of inlet concentration (0.01825 kmol/m^3) prevails. Thus, the conversion should be lower than 0.75 and, correspondingly, the outlet T should be higher than 625.3 K based on Figure 4.56.

$$X \leq 0.75 \tag{4.71}$$

$$T_{out} \geq 625.3 \tag{4.72}$$

However, the inequality constraints should be in the following form:

$$g((s, w, x), x) \leq 0 \tag{4.73}$$

Thus, the constraint can be described as:

$$g_1 = X - 0.75 \leq \mathbf{0} \tag{4.74}$$

$$g_2 = -T_{out} + 625.3 \leq \mathbf{0} \tag{4.75}$$

The training of the NN with regularization and direct implementation of constraints can be given as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \delta\|w\| \tag{4.76}$$

$$s.t. \quad g \leq \mathbf{0} \tag{4.77}$$

This optimization problem of NN training was performed by CNN-II, with/without penalty function technique (CNN-II-P/CNN-II-D). The NN structure consists of one input (one input layer), 10 neurons in the single hidden layer, where all neurons have hyperbolic tangent sigmoid functions, and two neurons in the output layer all having linear activation functions. The total number of weights and biases are 30 and 12, respectively. These make the number of decision variables in the optimization problem 42.

The data have 110 rows. 50% of the data (55 data points) were used in the training step, and the remaining 50% (55 data points) were used in the testing step. The training options for the direct implementation of constraints, **g**, by CNN-II-D, and for the unconstrained NN Toolbox are the optimizer selection (`fmincon` for CNN-II-D, `trainbr` for NN Toolbox), and regularization parameter (δ = 0.0001). Regularization was used to avoid overlearning. As the error function, SSE was used since `trainbr` performs only with SSE.

For CNN-II-D, initial guesses for the decision variables (NN weights and biases, shown as **w**) were obtained from the custom made initialization function as using NW initialization method that was explained in Section 3.5. With 55 data points, the total number of inequality constraints (Equation 4.74 and Equation 4.75) imposed on output **X** and $\mathbf{T}_{out}$ becomes 110. Therefore, the optimization problem given by Equation 4.76 above has 42 decision variables and 110 inequality constraints in the form Equation 4.77.

The maximum number of epochs allowed for the `trainbr` of the NN Toolbox was set to 1000. Other optional parameters of the `trainbr` of the NN Toolbox were set such that the training will not usually stop before maximum epochs allowed (e.g., min gradient = 1×10^{-12}, max mu = 1×10^{10}). For CNN-II, the optimization termination criteria were set as TolX = 1×10^{-4}, TolFun = 1×10^{-5}, TolCon = 1×10^{-2}, and MaxIter = 1000.

The outputs of CNN-II-D (by direct constraint implementation) and NN-Toolbox are given in Figure 4.58. The horizontal line shows the constraint limit ($X \leq 0.75$ and $T_{out} \geq 625.3$) and vertical line separates the training (left) and testing (right) parts.

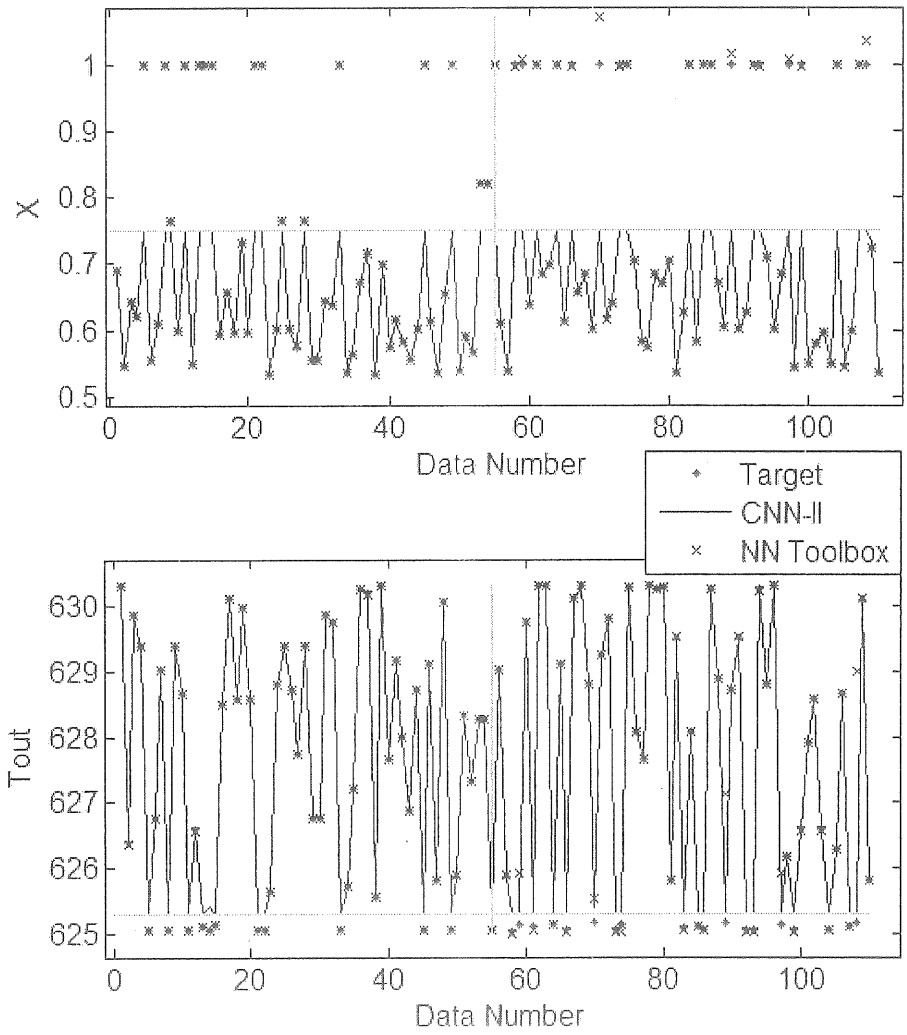


Figure 4.58. Outputs of CNN-II-D for PBR.

The regression coefficients (R^2) of the NN Toolbox are one for outputs X and T_{out} in the training set, and 0.99, and 0.96 for outputs X and T_{out} in the testing set, respectively.

The overall regression coefficient of NN Toolbox considering all data is 0.99. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set very well, and its predictions are also good in the testing set, however these predictions violate the maximum conversion and minimum outlet temperature limits ($\mathbf{X} \leq 0.75$ and $\mathbf{T}_{out} \geq 625.3$). On the other hand, the regression coefficients of CNN-II-D are 0.90 and 0.99 for outputs X and T_{out} in the training set, respectively, and 0.92 and 0.99 for the outputs X and T_{out} in the testing set, respectively. The overall regression coefficient of CNN-II-D considering all data is one. This shows that CNN-II-D performs well both in the training and testing sets even under inequality constraints, where violations of the conversion limit ($\mathbf{X} \leq 0.75$ and $\mathbf{T}_{out} \geq 625.3$) are almost zero. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The norm, $\|\mathbf{X} - 0.75\|$, of the constraint violations is 0.005 for CNN-II-D and 1.42 for the NN Toolbox. The norm, $\|625.3 - \mathbf{T}_{out}\|$, of the constraint violations is 0.001 for CNN-II-D and 1.23 for the NN Toolbox.

The constrained NN problem can also be solved by using the penalty function technique. The optimization problem of constrained NN training with penalty function technique can be described as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \mu \sum_k max(0, g_k) + \delta \|w\| \tag{4.78}$$

where,

$$g_1 = X - 0.75 \leq 0 \tag{4.79}$$

$$g_2 = -T_{out} + 625.3 \leq 0 \tag{4.80}$$

The NN structure is the same as in the direct constraint implementation case. The training options for the indirect implementation of constraints (penalty function technique), by CNN-II-P, and for the unconstrained NN Toolbox are the optimizer selection (`fminunc` for CNN-II-P, `trainbr` for NN Toolbox), regularization parameter (δ = 0.0001), and penalty parameter (μ = 8). This optimization problem has also 42 decision variables, however no constraints.

For CNN-II-P, initial guesses for the decision variables were obtained from the custom made initialization function by using NW initialization method that was explained

in Section 3.5 as in the direct constraint implementation case. All other optimization (`fmincon`) related optional parameters are the same as in the direct constraint handling case described above.

The outputs of CNN-II-P (by penalty function technique) and NN-Toolbox are given in Figure 4.59. A horizontal line shows the constraint limit ($X \leq 0.75$ and $T_{out} \geq 625.3$) and vertical line separates the training (left) and testing (right) parts.

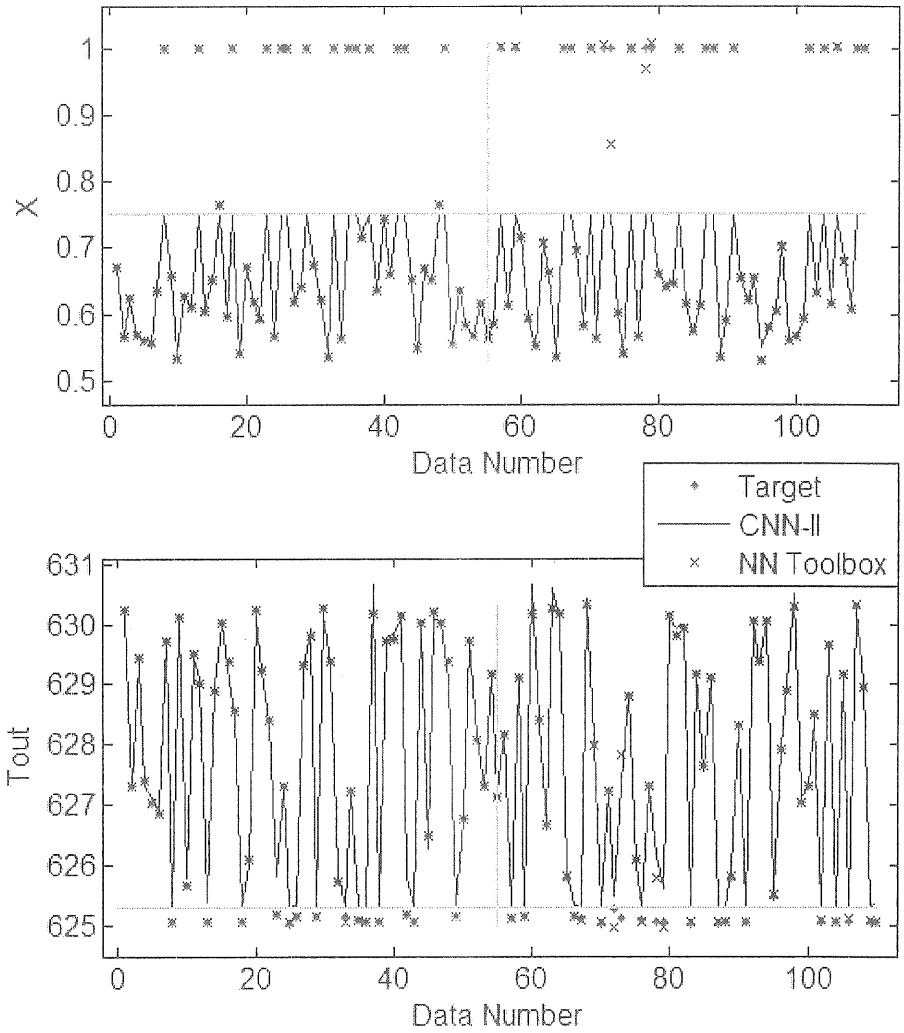


Figure 4.59. Outputs of CNN-II-P for PBR.

The regression coefficients (R^2) of the NN Toolbox are one and 0.99 for outputs X and T_{out} in the training set, respectively, and 0.99, and 0.98 for outputs X and T_{out} in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is 0.99. This shows that `trainbr` (even though the Bayesian regularization was

employed) learns the training set very well, and its predictions are also perfect in the testing set, however these predictions violate the maximum conversion and minimum outlet temperature limits ($\mathbf{X} \leq 0.75$ and $\mathbf{T}_{out} \geq 625.3$). On the other hand, the regression coefficients of CNN-II-P are 0.91 and 0.99 for outputs X and T_{out} in the training set, respectively, and 0.94 and 0.98 for the outputs X and T_{out} in the testing set, respectively. The overall regression coefficient of CNN-II-P considering all data is 0.99. This shows that CNN-II-P performs well both in the training and testing sets even under inequality constraints, where violations of the conversion and outlet temperature limits ($\mathbf{X} \leq 0.75$ and $\mathbf{T}_{out} \geq 625.3$) are almost zero. Furthermore, it is impossible to put inequality constraints to `trainbr` of NN Toolbox. The norm, $\|\mathbf{X} - 0.75\|$, of the constraint violations is 0.003 for CNN-II-P and 1.47 for the NN Toolbox. The norm, $\|625.3 - \mathbf{T}_{out}\|$, of the constraint violations is 0.0006 for CNN-II-P and 1.21 for the NN Toolbox.

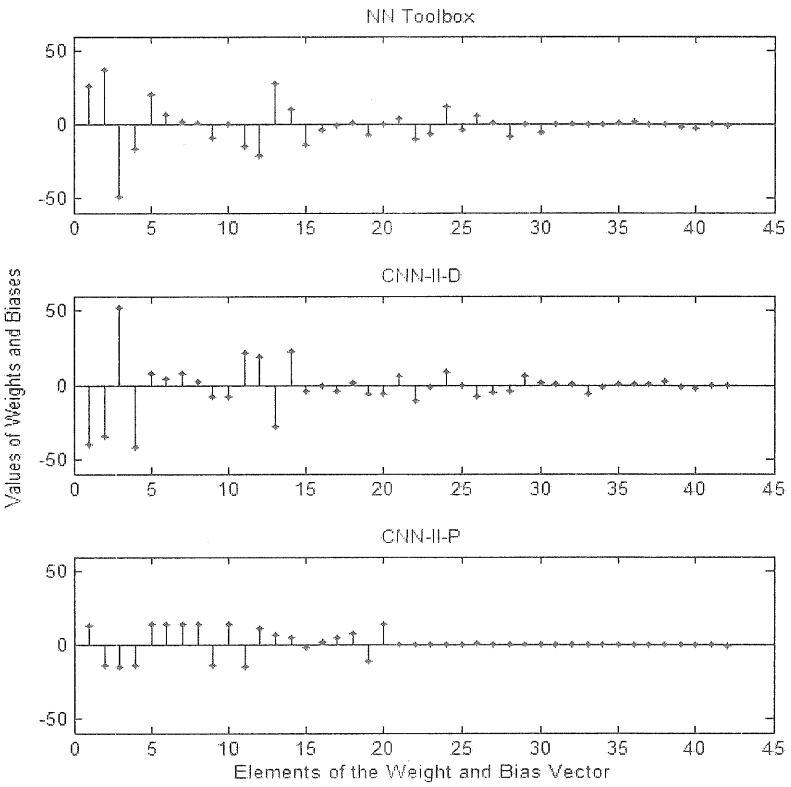


Figure 4.60. Weights and biases of the NN Toolbox and CNN-II for PBR.

Figure 4.60 shows and compares the weights and biases of the NN Toolbox after training with `trainbr` and the optimal decision variables (weights and biases) obtained via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II.

It is seen that NN Toolbox weights and biases are more homogeneous than CNN methods. The presence of less homogeneous decision variables (NN weights and biases) in the CNN methods can be attributed to possible preferential activation of some of the neurons which primarily take role in satisfying the constraints.

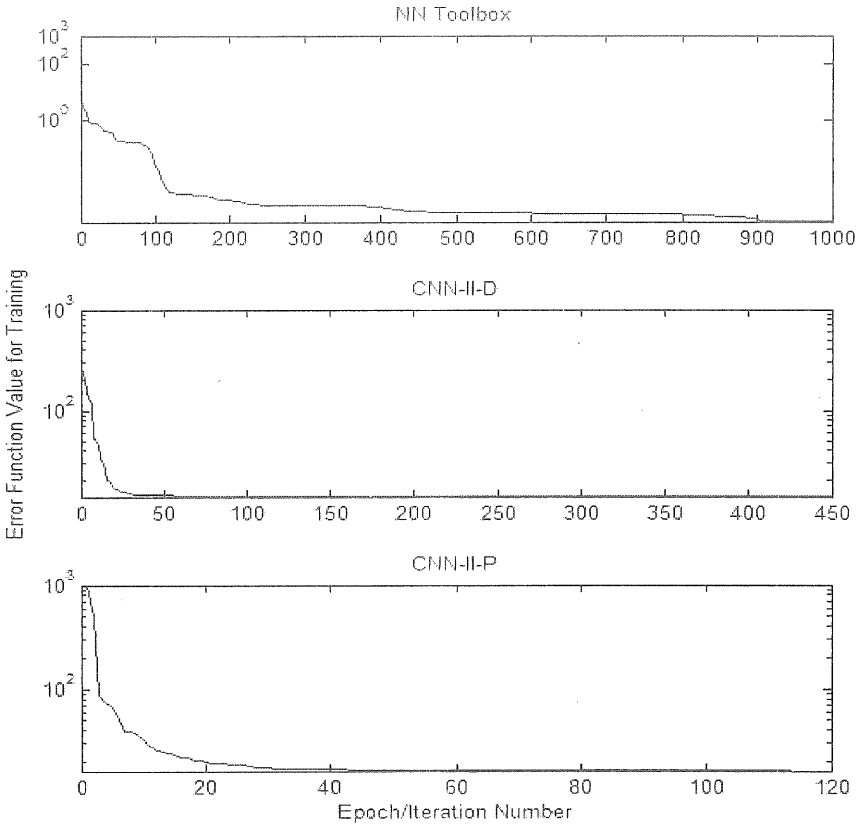


Figure 4.61. Progress of SSE in training by NN Toolbox and CNN-II for PBR.

Figure 4.61 shows the progress of SSE as a function of epochs for the NN Toolbox and as a function of iterations for optimization via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. These show that CNN methods approach the minimum in small number of iterations comparable to the NN Toolbox. In Figure 4.61, the function values on the y-axis are the pure SSE values as reported by the NN Toolbox used with `trainbr` (Bayesian regularization), however, for CNN-II, the function values on the y-axes are the SSE values including the regularized term values given by Equation 2.26 in Section 2.5.2.

The CPU times as measured by the MATLAB's `tic` and `toc` functions are compared in Figure 4.62.

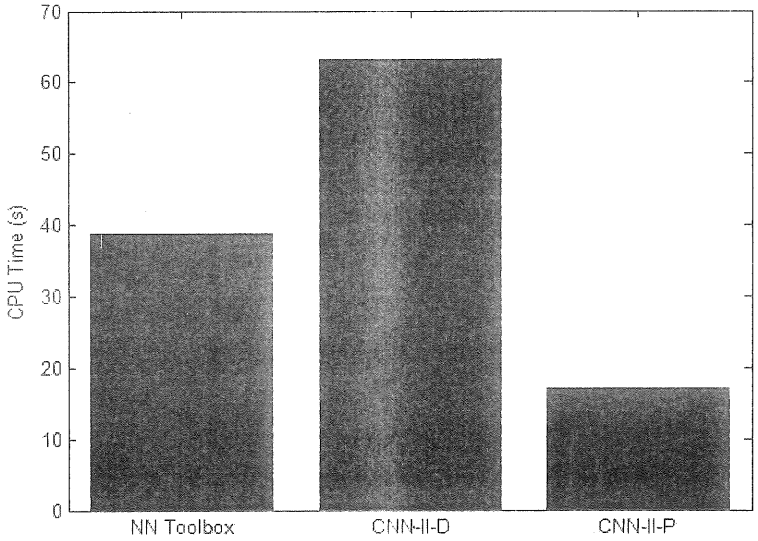


Figure 4.62. Training CPU times for PBR.

Table 4.6. Comparison of NN Toolbox and CNN-II method for PBR.

	NN Toolbox	CNN-II-D	CNN-II-P
Number of Parameters	42	42	42
Number of Constraints	-	110	-
Training SSE, overall	0.001	1.491	2.923
Training SSE, X	2.3×10^{-6}	7.5×10^{-1}	8.7×10^{-1}
Training SSE, T_{out}	0.001	0.731	2.049
Testing SSE, overall	19.9	1.9	9.1
Testing SSE, X	0.007	1.187	1.183
Testing SSE, T_{out}	19.91	0.80	7.94
Overall Constraint ($\mathbf{g}$) Violation Norm	1.882	0.005	0.003
Training Constraint ($\mathbf{g}_1$) Violation Norm	0.871	0.005	0.001
Testing Constraint ($\mathbf{g}_1$) Violation Norm	1.1224	0.0006	0.0029
Training Constraint ($\mathbf{g}_2$) Violation Norm	0.809	0.001	0
Testing Constraint ($\mathbf{g}_2$) Violation Norm	0.9327	0.0010	0.0006
Training CPU (s)	38.7	63.2	17.08
Number of Epochs/Iterations	1000	452	113

Table 4.6 summarizes the comparison among NN Toolbox, CNN-II-D (direct constraint implementation), and CNN-II-P (penalty function implementation). Since the initialization of weights and biases are done randomly, the results may change from run to

run; the NN Toolbox results listed in the table correspond to the run the results of which are given in Figure 4.58.

Since there is an inequality constraint on the outputs X and T_{out}, the SSE values given in the above table for the outputs X and T_{out} should be appraised with care. However, from Figure 4.58 and Figure 4.59, it is visually seen that values of X below 0.75 and T_{out} above 625.3 (values of X and T_{out} in the feasible region) fit the data points very well.

In summary, NN Toolbox cannot handle constraints. It should be noted that for NN Toolbox and CNN-II, the initialization of weights and biases are done randomly according to the NW algorithm as explained in Section 2.4. Therefore, the SSE, constraint violation norm, and CPU values change from run to run. Therefore, the figures, SSE, and constraint violation norms, and CPU values should be accepted only as representative values.

In this problem, the data are obtained from the solutions of the ODEs (mass and energy balance of the system) for different inlet concentrations. However, the constraint information (the impracticable conversions at the condition of runaway temperatures) cannot be contained in these equations. Thus, the NN trained by NN Toolbox can only learn the information contained in the solutions of these algebraic equations and NN Toolbox constructs an empirical equation that implicitly describes only the mass and energy balance equations. However, the NN trained by CNN methods learn the information contained in the mass and energy balances together with the external information imposed as inequality constraint.

4.6. Williams-Otto Plant

The process examined in this problem is initially proposed by Williams and Otto (1960). The Williams-Otto Plant (WOP) has been employed with small modifications as a tool for typical chemical process simulation in many journal articles and optimization textbooks (Edgar *et al.*, 2001). The process flowsheet is shown in Figure 4.63.

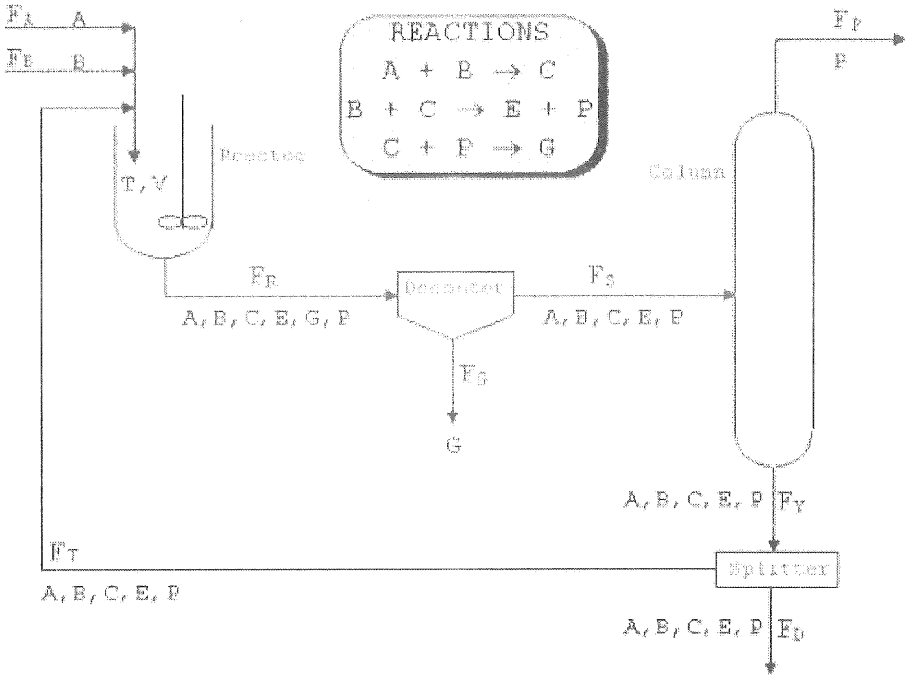


Figure 4.63. Williams-Otto Plant floowsheet.

F_A and F_B are fresh feed components of A and B, F_m is total stream flow rates (m: R, S, D, P, T, Y), $F_{m,i}$ is component flow rates (i: A, B, C, E, G, P), T is reactor temperature, and V is reactor volume.

Feed streams with pure species A and B are mixed with a recycle stream and enter a continuously stirred tank reactor, where three reactions indicated on the figure take place. C is an intermediate, P is the main product, and G is a waste product. The decanter separates the waste product G from reactants and other products. The distillation column separates product P. Due to the formation of an azeotrope, some of the product (10 wt% of the mass flowrate of component E in the column feed) is retained in the column bottoms. Most of this bottom product is recycled to the reactor, and the rest is used as fuel.

The plant model can be defined without an energy balance and further simplified by considering only isothermal reactions for the manufacture of compound P. The reactor model is given with these specifications as:

$$k_i = a_i \exp\left(-\frac{b_i}{T}\right)\frac{V\rho}{F_R^2} \tag{4.81}$$

$$R_1 = k_1 F_{RA} F_{RB} \tag{4.82}$$

$$R_2 = k_2 F_{RB} F_{RC} \tag{4.83}$$

$$R_3 = k_3 F_{RC} F_{RP} \tag{4.84}$$

where, ρ is density of the reaction mixture (ρ = 50 lb/ft^3), R is the reaction rate equations, k is the rate constant, a and b are the parameters of the rate constant equation.

Table 4.7. WOP rate equation constants.

i	a_i	b_i
1	$5.9755*10^9$	12000
2	$2.5962*10^{12}$	15000
3	$9.6283*10^{15}$	20000

The steady state component (mass) balances around the reactor are given as:

A: $$F_A + F_{TA} - F_{RA} - R_1 = 0 \tag{4.85}$$

B: $$F_B + F_{TB} - F_{RB} - R_1 - R_2 = 0 \tag{4.86}$$

C: $$F_{TC} - F_{RC} + 2R_1 - 2R_2 - R_3 = 0 \tag{4.87}$$

E: $$F_{TE} - F_{RE} + 2R_2 = 0 \tag{4.88}$$

G: $$F_{TG} - F_{RG} + \frac{3}{2}R_3 = 0 \tag{4.89}$$

P: $$F_{TP} - F_{RP} + R_2 - \frac{1}{2}R_3 = 0 \tag{4.90}$$

The decanter unit assumes a perfect separation between G and other components. The component (mass) balance equations can be written as:

$$F_{S,i} = F_{R,i} \quad i = A, B, C, E, P \tag{4.91}$$

$$F_{S,G} = 0 \tag{4.92}$$

$$F_{G,i} = 0 \quad i = A, B, C, E, P \tag{4.93}$$

$$F_{G,G} = F_{R,G} \tag{4.94}$$

The distillation column assumes a pure separation of product P in the overhead but also assumes that some of the product is retained in the bottoms due to formation of an azeotrope (10 wt% of the mass flowrate of component E in the column feed is taken as the

amount of P retained in the bottoms). Thus, the component (mass) balance equations can be written as:

$$F_{Y,i} = F_{S,i} \quad i = A, B, C, E \tag{4.95}$$

$$F_{P,i} = 0 \quad i = A, B, C, E, P \tag{4.96}$$

$$F_{P,P} = F_{S,P} - 0.1F_{S,E} \tag{4.97}$$

$$F_{Y,P} = F_{S,P} - F_{P,P} = 0.1F_{S,E} \tag{4.98}$$

$$F_{G,G} = F_{R,G} \tag{4.99}$$

The flow splitter component (mass) balance equations can be written as:

$$F_{T,i} = \alpha F_{Y,i} \quad i = A, B, C, E, P \tag{4.100}$$

$$F_{D,i} = (1 - \alpha)F_{Y,i} \quad i = A, B, C, E, P \tag{4.101}$$

where, α is split fraction. When considering the component balances of the process, there are five degrees of freedom. The specification of these five variables makes the degrees of freedom zero and a unique solution can be obtainable. These variables are V, T, α, F_A, and F_B. The values of the three fixed variables are given in Table 4.8. Of these five variables, F_A and F_B are the input of the system and directly related to the outputs of the process. In obtaining the data for the training of the NNs, three of the variables (V, T, α) were fixed and two variables (F_A and F_B) were perturbed randomly from their nominal values given in Table 4.8, and the WOP equations were solved using MATLAB's nonlinear algebraic equation solver `fsolve` 100 times for 100 different F_A, and F_B values.

Table 4.8. Parameter specifications of WOP.

T	653
V	100
α	0.74
F_A	15000
F_B	30000

The NN is formed to reflect the relation between inputs (F_A and F_B) and outputs of the process (F_P, F_G, and F_D). The data are obtained from the solution of algebraic equations

(mass balances) of the WOP. Therefore the data satisfy the mass balances exactly and these implicit balances embedded in the data can be learned by the NN Toolbox. In real life applications, the data contain measurement errors and thus do not satisfy the mass balances. In order to simulate the real situation, noise is added to the data. The measurement errors represented by normal random numbers (with mean zero and standard deviations of 100.4, 248.2, 296.1, 18.4, and 34.2 for the F_A, F_B, F_P, F_G, and F_D data, respectively) are added to the data, such that the equality constraint (mass balance, Equation 4.106) violations range from −1009 to 920.4. The generated NN is depicted in Figure 4.64.

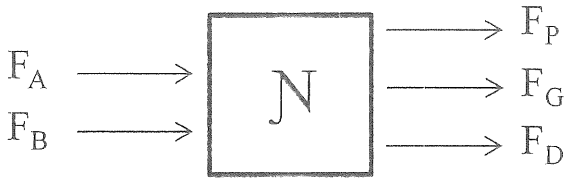


Figure 4.64. NN model of WOP.

The problem can be described as:

$$z = (s, w, x) \tag{4.102}$$

where,

$$z = [F_P \quad F_G \quad F_D] \tag{4.103}$$

The constraint of the problem is the mass balance of the process. The total outputs should be equal to the total inputs.

$$F_A + F_B = F_P + F_G + F_D \tag{4.104}$$

However, the equality constraints should be in the following form:

$$h((s, w, x), x) = 0 \tag{4.105}$$

The constraint of the plant is defined as:

$$\boldsymbol{h} = F_A + F_B - F_P - F_G - F_D = \mathbf{0} \tag{4.106}$$

The training of the NN with regularization and direct implementation of constraints can be given as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \delta\|w\| \tag{4.107}$$

$$s.t. \quad \boldsymbol{h} = \mathbf{0} \tag{4.108}$$

This optimization problem of NN training was performed by CNN-II, with/without penalty function technique (CNN-II-P/CNN-II-D). The NN structure consists of one input (one input layer), 20 neurons in the single hidden layer, where all neurons have hyperbolic tangent sigmoid functions, and three neurons in the output layer all having linear activation functions. The total number of weights and biases are 100 and 23, respectively. These make the number of decision variables in the optimization problem 123.

The data have 100 rows. 70% of the data (70 data points) were used in the training step, and the remaining 30% (30 data points) were used in the testing step. The training options for the direct implementation of constraints, **h**, by CNN-II-D, and for the unconstrained NN Toolbox are the optimizer selection (`fmincon` for CNN-II-D, `trainbr` for NN Toolbox), and regularization parameter (δ = 0.0001). Regularization was used to avoid overlearning. As the error function, SSE was used since `trainbr` performs only with SSE.

For CNN-II-D, initial guesses for the decision variables (NN weights and biases, shown as **w**) were obtained from the custom made initialization function as using NW initialization method that was explained in Section 3.5. With 70 data points, the total number of equality constraints (Equation 4.106) imposed on output **X** becomes 70. Therefore, the optimization problem given by Equation 4.107 above has 123 decision variables and 70 equality constraints in the form Equation 4.108.

The maximum number of epochs allowed for the `trainbr` of the NN Toolbox was set to 1000. Other optional parameters of the `trainbr` of the NN Toolbox were set such

that the training will not usually stop before maximum epochs allowed (e.g., min gradient = 1×10^{-12}, max mu = 1×10^{10}). For CNN-II, the optimization termination criteria were set as TolX = 1×10^{-3}, TolFun = 1×10^{-4}, TolCon = 1×10^{-2}, and MaxIter = 1000.

The outputs of CNN-II-D (by direct constraint implementation) and NN-Toolbox are given in Figure 4.65. The vertical line separates the training (left) and testing (right) parts.

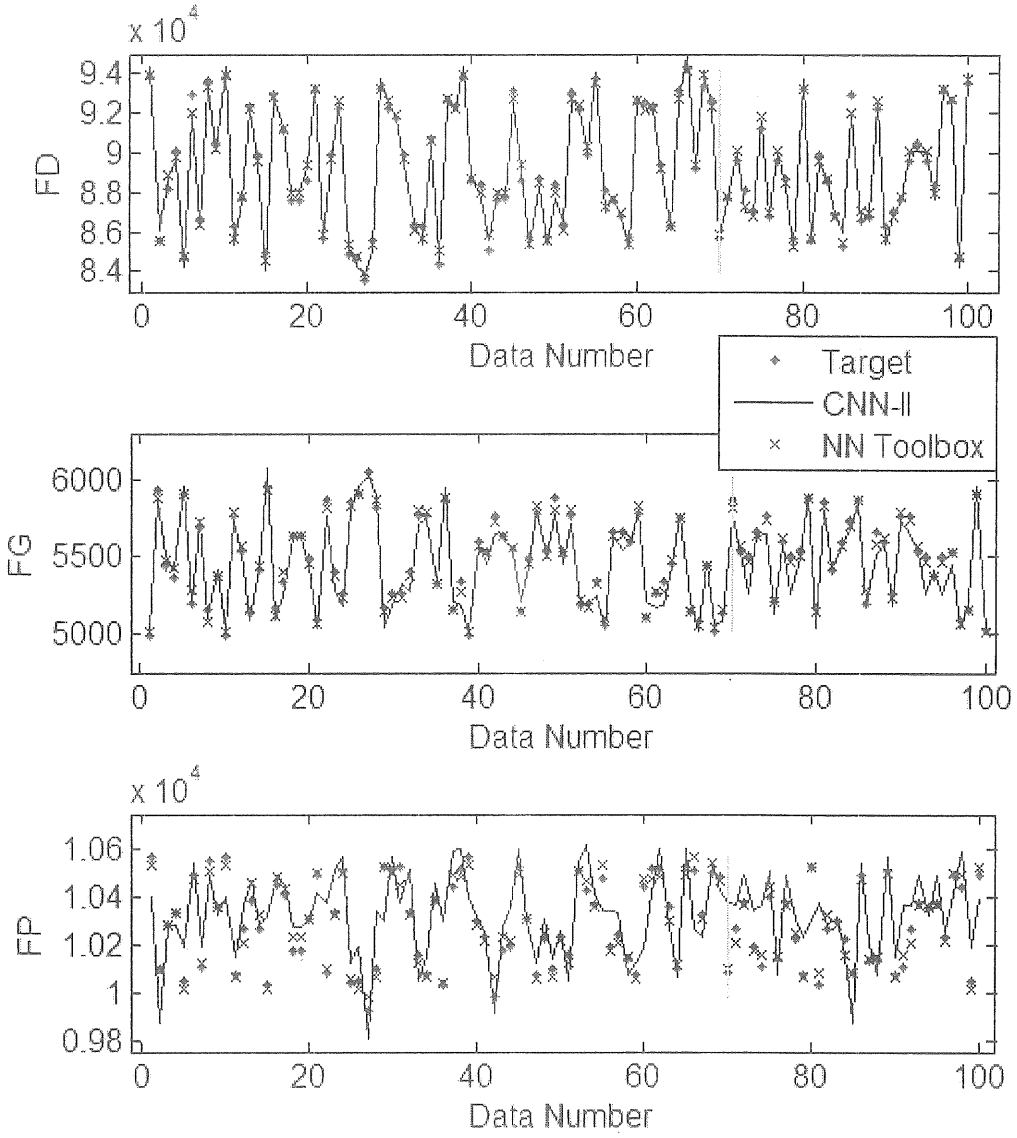


Figure 4.65. Outputs of CNN-II-D for WOP.

The violations of equality constraints for CNN-II-D are shown in Figure 4.66.

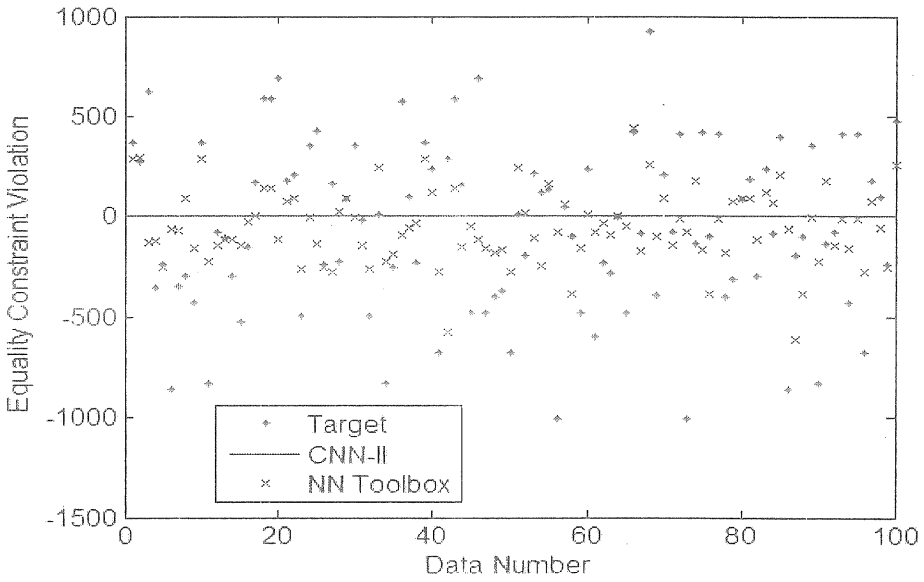


Figure 4.66. Violations of equality constraints for CNN-II-D for WOP.

The regression coefficients (R^2) of the NN Toolbox are 0.99, 0.99, and 0.98 for outputs F_D, F_G, and F_P in the training set, respectively, and 0.99, 0.99, and 0.98 for outputs F_D, F_G, and F_P in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is also 0.99. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set very well, and its predictions are also perfect in the testing set, however these predictions violate equality constraint (**h**). On the other hand, the regression coefficients of CNN-II-D are 0.99, 0.98, and 0.75 for outputs F_D, F_G, and F_P in the training set, respectively, and 0.99, 0.94, and 0.68 for outputs F_D, F_G, and F_P in the testing set, respectively. The overall regression coefficient of CNN-II-D considering all data is 0.99. This shows that CNN-II-D performs well both in the training and testing sets even under equality constraints, where violations of the equality constraints are almost zero. Furthermore, it is impossible to put equality constraints to `trainbr` of NN Toolbox. The norm of **h** of the constraint violations is 0.14 for CNN-II-D and 1921 for the NN Toolbox.

The constrained NN problem can also be solved by using the penalty function technique. The optimization problem of constrained NN training with penalty function technique can be described as:

$$\min_{w} f(w) = SSE(\,(s, w, x^t), y^t) + \mu\|\boldsymbol{h}\| + \delta\|\boldsymbol{w}\| \qquad (4.109)$$

where,

$$h = 0 \tag{4.110}$$

The NN structure is the same as in the direct constraint implementation case. The training options for the indirect implementation of constraints (penalty function technique), by CNN-II-P, and for the unconstrained NN Toolbox are the optimizer selection (`fminunc` for CNN-II-P, `trainbr` for NN Toolbox), regularization parameter (δ = 0.0001), and penalty parameter (μ = 5). This optimization problem has also 123 decision variables, however no constraints.

For CNN-II-P, initial guesses for the decision variables were obtained from the custom made initialization function by using NW initialization method that was explained in Section 3.5 as in the direct constraint implementation case. All other optimization (`fmincon`) related optional parameters are the same as in the direct constraint handling case described above.

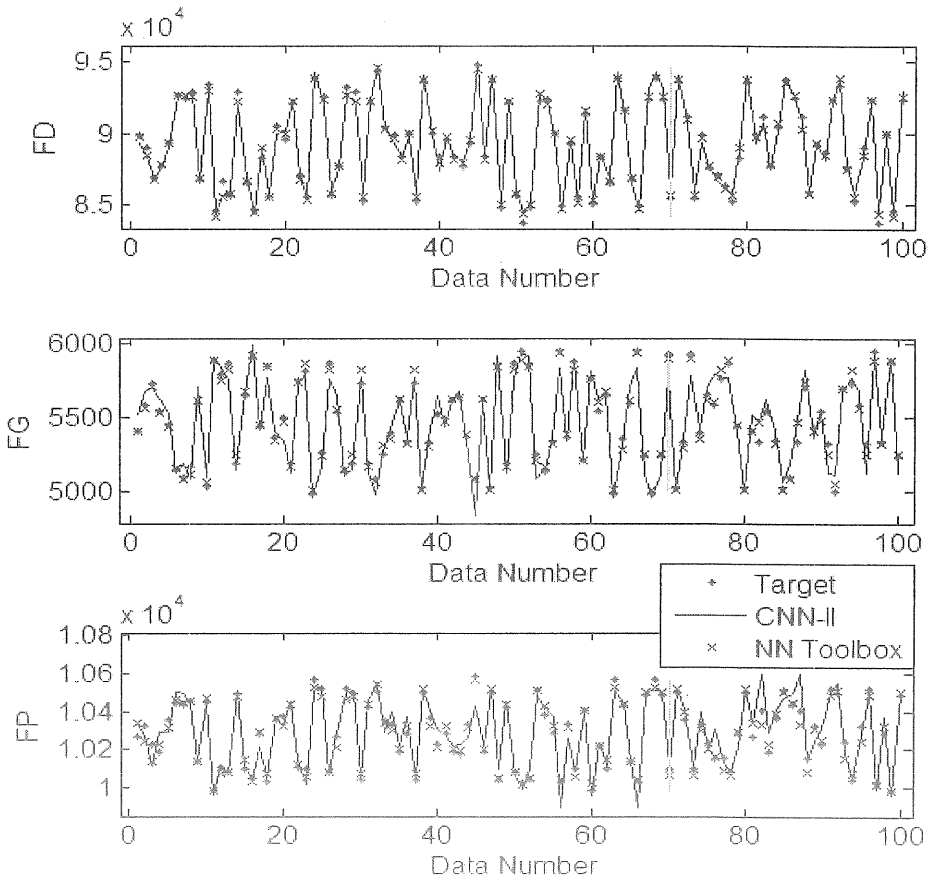


Figure 4.67. Outputs of CNN-II-P for WOP.

The outputs of CNN-II-P (by penalty function technique) and NN-Toolbox are given in Figure 4.67. The vertical line separates the training (left) and testing (right) parts.

The regression coefficients (R^2) of the NN Toolbox are 0.99, 0.99, and 0.98 for outputs F_D, F_G, and F_P in the training set, respectively, and 0.99, 0.98, and 0.96 for outputs F_D, F_G, and F_P in the testing set, respectively. The overall regression coefficient of NN Toolbox considering all data is also 0.99. This shows that `trainbr` (even though the Bayesian regularization was employed) learns the training set very well, and its predictions are also perfect in the testing set, however these predictions violate equality constraint (**h**). On the other hand, the regression coefficients of CNN-II-P are 0.99, 0.97, and 0.96 for outputs F_D, F_G, and F_P in the training set, respectively, and 0.99, 0.94, and 0.92 for outputs F_D, F_G, and F_P in the testing set, respectively. The overall regression coefficient of CNN-II-P considering all data is 0.99. This shows that CNN-II-P performs well both in the training and testing sets even under equality constraints, where violations of the equality constraints are very small. Furthermore, it is impossible to put equality constraints to `trainbr` of NN Toolbox. The norm of **h** of the constraint violations is 501 for CNN-II-D and 2262 for the NN Toolbox.

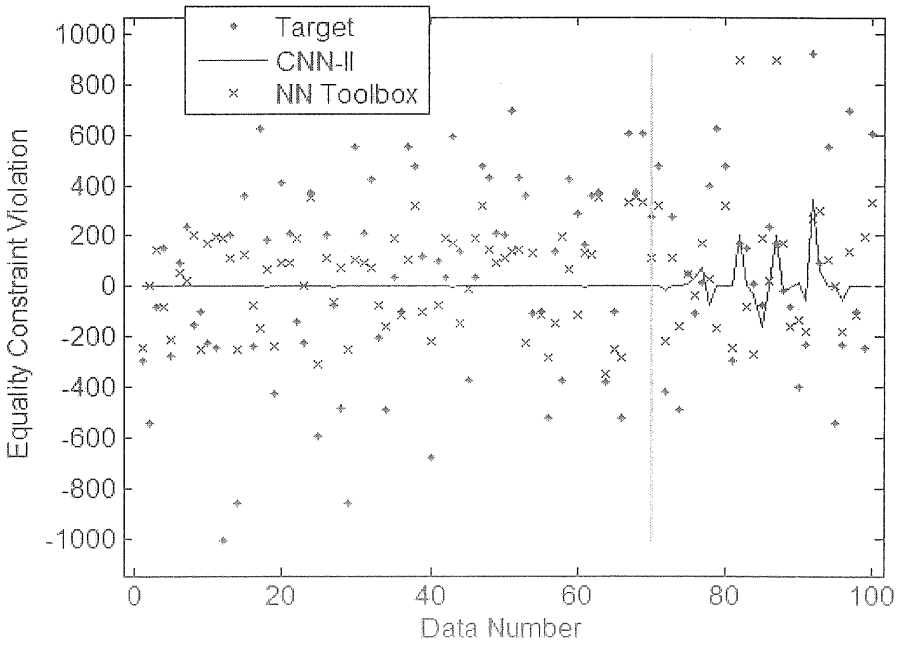


Figure 4.68. Violations of equality constraints for CNN-II-P for WOP.

The violations of equality constraints for CNN-II-P are shown in Figure 4.68.

Figure 4.69 shows and compares the weights and biases of the NN Toolbox after training with `trainbr` and the optimal decision variables (weights and biases) obtained via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. It is seen that NN Toolbox weights and biases are more homogeneous than CNN methods. The presence of less homogeneous decision variables (NN weights and biases) in the CNN methods can be attributed to possible preferential activation of some of the neurons which primarily take role in satisfying the constraints.

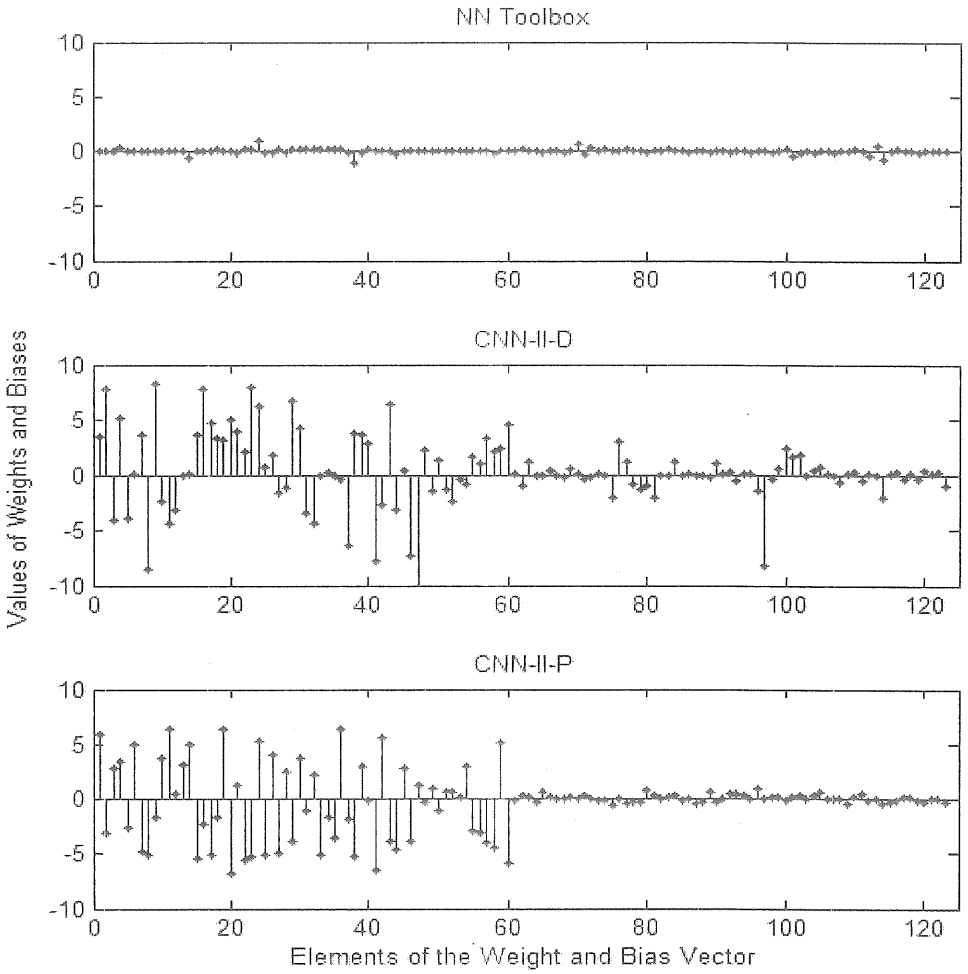


Figure 4.69. Weights and biases of the NN Toolbox and CNN-II for WOP.

Figure 4.70 shows the progress of SSE as a function of epochs for the NN Toolbox and as a function of iterations for optimization via `fmincon` for direct and `fminunc` for indirect constraint implementations of CNN-II. These show that CNN methods approach the minimum in small number of iterations comparable to the NN Toolbox. In Figure 4.70, the function values on the y-axis are the pure SSE values as reported by the NN Toolbox

used with `trainbr` (Bayesian regularization), however, for CNN-II, the function values on the y-axes are the SSE values including the regularized term values given by Equation 2.26 in Section 2.5.2.

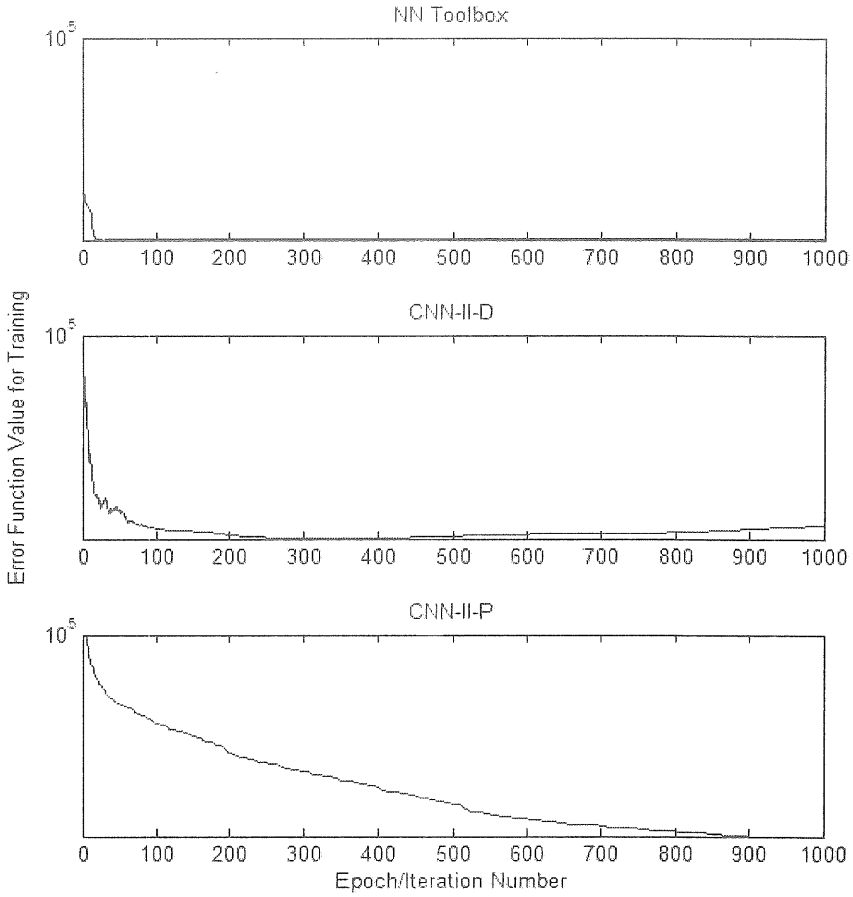


Figure 4.70. Progress of SSE in training by NN Toolbox and CNN-II for WOP.

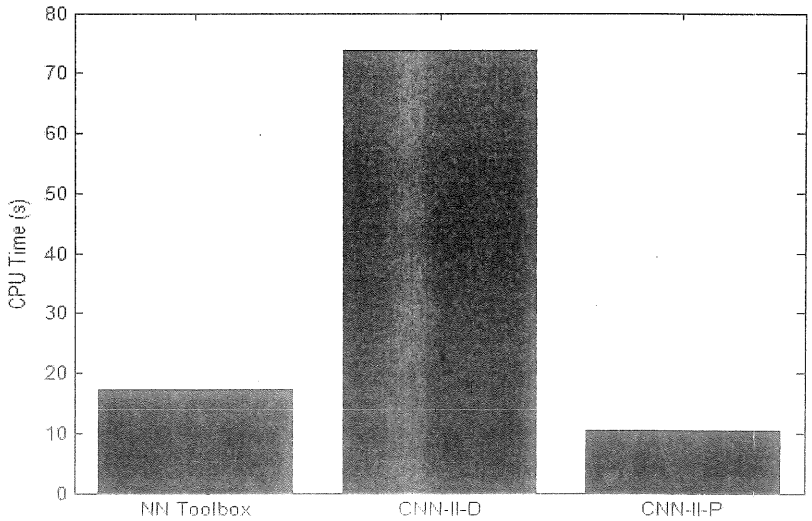


Figure 4.71. Training CPU times for WOP.

The CPU times as measured by the MATLAB's `tic` and `toc` functions are compared in Figure 4.71.

Table 4.9 summarizes the comparison among NN Toolbox, CNN-II-D (direct constraint implementation), and CNN-II-P (penalty function implementation). Since the initialization of weights and biases are done randomly, the results may change from run to run; the NN Toolbox results listed in the table correspond to the run the results of which are given in Figure 4.65.

Table 4.9. Comparison of NN Toolbox and CNN-II method for WO Plant.

	NN Toolbox	CNN-II-D	CNN-II-P
Number of Parameters	123	123	123
Number of Constraints	-	70	-
Training SSE, overall	9.9×10^{6}	1.7×10^{7}	1.2×10^{7}
Training SSE, F_D	9.7×10^{6}	1.5×10^{7}	1.2×10^{7}
Training SSE, F_G	9.1×10^{4}	3.2×10^{5}	4.2×10^{5}
Training SSE, F_P	8.6×10^{4}	1.1×10^{6}	2.1×10^{5}
Testing SSE, overall	4.5×10^{6}	7.8×10^{6}	5.5×10^{6}
Testing SSE, F_D	4.4×10^{6}	6.9×10^{2}	5.1×10^{6}
Testing SSE, F_G	3.1×10^{4}	3.4×10^{5}	3.1×10^{5}
Testing SSE, F_P	3.1×10^{4}	5.7×10^{5}	1.1×10^{5}
Training Constraint (**h**) Violation Norm	1568.1	0.1	6.4
Testing Constraint (**h**) Violation Norm	1109.6	0.1	501.4
Training CPU (s)	14.6	303.8	95.1
Number of Epochs/Iterations	1000	1000	901

Since there is an equality constraint on the outputs, the SSE values given in the above table for the outputs should be appraised with care. However, from Figure 4.65 and Figure 4.67, it is visually seen that values of outputs (values of outputs in the feasible region) fit the data points very well.

In summary, NN Toolbox cannot handle constraints. It should be noted that for NN Toolbox and CNN-II, the initialization of weights and biases are done randomly according

to the NW algorithm as explained in Section 2.4. Therefore, the SSE, constraint violation norm, and CPU values change from run to run. Therefore, the figures, SSE, and constraint violation norms, and CPU values should be accepted only as representative values.

In this problem, the data were obtained from the solutions of the algebraic equations (mass balances of WOP) for different F_A and F_B values. The noise was added to the data in order to simulate realistic situation. The NN trained by NN Toolbox can only learn the information contained in the solutions of these algebraic equations together with the noise added to the data. However, the NN trained by CNN methods learns the information contained in the mass balances via the external information imposed as an equality constraint. Since the data contain noise, NN Toolbox cannot satisfy the mass balance of the system.

5. CONCLUSIONS AND RECOMMENDATIONS

In this thesis, novel methods of training Neural Networks (NNs) under constraints are developed and they are called as the Constrained Neural Networks (CNNs). The usage of existing unconstrained NN tools in the constrained applications can be problematic. Thus, the necessity of considering constraints for using NNs in real-life applications is overcome by the CNNs. Almost every data taken from real-life applications contain measurement errors that causes violations of the constraints originating from the first principles of conservation of mass and energy. Therefore, the importance of CNNs increases with increasing importance of satisfaction of constraints and minimization of their perhaps unavoidable violations. Moreover, CNNs can also be seen as the "constrained-learning" methods, and thus, alternative areas of usage can be much wider than the above-stated scope if they are considered in other fields.

The NN algorithm consists of mainly three steps; preprocessing, training, and post-processing. The CNN methods consider mainly the training part of the NN algorithm, since the constraint implementation can be made only at the training phase. The training of the NN is actually an optimization problem and this optimization problem is solved by well known unconstrained optimization tools, such as the Levenberg-Marquardt (LM) method. If the training of a NN is seen as an optimization problem, the constraint implementation is possible by changing the unconstrained optimization tool with a constrained optimization tool or using alternative techniques (e.g., penalty function technique) with unconstrained optimization tools. The main idea in the CNN algorithms is to integrate the constrained optimization tools to NN algorithms. Thus, the constraint implementation can be made during the training and the system with constraints can be simulated after the training phase. Since the most popular NN tool is the NN Toolbox of MATLAB, the CNN methods developed in this thesis were compared with this unconstrained tool.

The equality/inequality constraint implementations on/between the outputs of a NN during training step were made in two alternative schemes. In the first alternative scheme, CNN-I, some functions of the MATLAB's NN Toolbox were employed. The NN building and topology functions were utilized as building blocks directly without any modification.

However, the training functions were not used. Instead, an unconstrained or constrained generic optimizer was employed from the MATLAB's Optimization Toolbox for the training of the CNN. In the training step, there were two alternative methods used for constraint implementation. The first one (CNN-I-D) involved the direct constraint implementation tools, employing a constrained optimizer (such as `fmincon`). The second one (CNN-I-P) involved the indirect constraint implementation tools (i.e., penalty function technique) employing an unconstrained optimizer (such as `fminunc`). For indirect implementation of the equality and/or inequality constraints in the NN training task, the constraint violations were included as a penalty term in the augmented objective function.

In the second alternative scheme, CNN-II, all the necessary functions of NN Toolbox employed in the CNN-I were coded explicitly. The algorithm of the CNN-II is same as CNN-I however the functions are custom made. Hence, the desired modifications can be made easily in the CNN-II and also the execution speed of the CNN-II is much higher than the CNN-I since it does not contain any input checking statements.

Alternative third scheme was proposed as CNN-III, with the aim of employing the training functions of NN Toolbox directly, especially the `trainlm` function, by defining custom-made error function. However, `trainlm` function does not accept an error function other than `mse`. Thus, this method was not implemented in this thesis since the performances of the other training methods that accept custom-made error function were not favorable.

The two CNN methods, CNN-I and CNN-II, were applied on six examples. Most of them are in the field of chemical engineering. Some of the example data were taken from articles, and some of the data were generated from model equations taken from the examples of various sources. In each example, the definition of the problem statement, data generation steps, and the logic of the created NN were described in detail. Then, for comparison, both the CNNs and NN Toolbox were trained for the example data. Regularization was performed in the all examples in order to prevent the overlearning. In the first example, a synthetic linear time series, both the CNN-I and CNN-II were employed and, when the solutions were compared, it was seen that the solutions were comparable. Thus, in the remaining examples, only the CNN-II was employed since CNN-

I is too slow due to the input-argument checking statements borrowed from the NN Toolbox. Among many optimizers, `fmincon` was employed for the direct constraint implementation and `fminunc` was employed for the indirect constraint implementation since they are the best performing constrained and unconstrained optimizers available in the Optimization Toolbox of MATLAB.

The same NN topology was used for both NN Toolbox and CNNs. However, it was disadvantageous for the CNNs since CNNs needs additional neurons to learn the constraint information together with the relation between inputs and outputs.

Besides the fitting performance of the NNs, several optimization performance criteria were observed in the solutions. First, the figures of outputs were given. In these figures, the fitting performance of the NN and the inequality constraint violations were demonstrated visually. For CNNs, the inequality constraint violations in the training phase were almost zero, and very small in the testing phase for all of the examples. Furthermore, the fitting performance was very well for the outputs in the feasible regions.

The regression coefficients of the outputs were also given. The regression coefficients were meaningless for the constrained outputs. In order to show this fact, the regression plots were given only for the synthetic linear time series example. The example included inequality constraint on one output and the effect of the constraint can be seen in regression plots. There was poor fitting below the constraint limit but the regression coefficient was above 0.90. Therefore, regression coefficients were not a good indicator of performance.

In order to demonstrate the constraint violation performance, inequality constraint violations were depicted in the NN output figures and equality constraint violations were depicted as separate figures. Additionally, the norms of the constraint violations were tabulated for the numerical comparison. CNNs were successful in achieving good constraint violation performance; the equality and/or inequality constraint violations were within tolerable limits even in the testing phase.

The weights and biases of the NN Toolbox and CNNs were also compared visually. The weights and biases of the NN Toolbox were more homogeneous than the CNNs. This means that the activations of the neurons were more like each other in the NN Toolbox. However, some of the magnitudes of weights and biases were higher than others in the CNNs; meaning that these corresponding neurons were more active than others. Therefore, it may be concluded that they played an important role in satisfying the constraints. Additionally, the weights and biases of the indirect constraint implementation case, which were obtained from `fminunc`, were more homogeneous than the direct constraint implementation case, which utilized `fmincon`. This situation stem from the algorithm behind `fmincon` that gives the first struggle to satisfying the constraints.

The performances of the NNs were also compared in terms of the SSE progress with optimization iterations/epochs. Generic optimizers found the local minima in small number of iterations than the training function (the LM method) of NN Toolbox. However, the comparison in terms of CPU times showed that the `trainbr` function of NN Toolbox, which uses the NN-tailored LM method with Bayesian regularization, is faster than the generic optimizers.

In the first example, two models of synthetic time series were considered. The data were generated from the linear and non-linear models. In the linear time series model, one inequality constraint was used on one output, and in the non-linear time series model, two inequality constraints were used on two outputs. Looking at the constraint violations in both CNN-II-D and CNN-II-P, and the fitting and testing performances in the feasible regions, it can be concluded that the thesis' aim was achieved.

The second example is from the area of thermodynamics. The data of ternary vapor-liquid equilibrium were modeled by the CNNs and NN Toolbox. It includes both inequality and equality constraints. Inequality constraints, which are bounds of mole fractions, were implicitly included in the data and equality constraint was the equality of the sum of mole fractions to unity. The equality constraint was also satisfied by the data itself. The relation between outputs and inputs were learned successfully together with the equality and inequality constraints where the violations were lower than the NN Toolbox. In the third example, the same data were utilized with the addition of noise to simulate realistic data

sampling case. Since the data contain measurement errors, the physical equality constraint was not satisfied by the data, in other words, the external information that the mole fractions must sum to one was not embedded into the data. Since it is impossible to implement constraints in the NN Toolbox, it performed well in the training phase and fitted the noisy (wrong) constraint-violating data very well, however without satisfying the equality constraint. On the other hand, the external information (implicit constraints that $0 \leq x \leq 1$ and $\sum x - 1 = 0$) were forced through imposed constraints during training phase in the CNN methods. This way, noise reconciliation and external information embedding were achieved simultaneously during the training in the CNN methods. Additionally, there were deviations between solutions of the CNNs and NN Toolbox in this ternary VLE problem. Small deviations between the solutions of the other examples were also observed. Thus, reproducibility analysis was made for the second example to see the deviation between the results obtained from run to run. In this analysis, it was seen that the deviations for the CNNs were low, yet the deviations in the NN Toolbox were high; yielding the conclusion that CNNs were more reliable and less sensitive to random initialization of weights and biases.

In the fourth problem, the data were obtained from the solutions of the algebraic equations (mass and energy balances of the system) of a continuously stirred tank reactor (CSTR) for different coolant temperature values. The white noise was added to the data to emulate measurement errors in order to simulate a real-life situation. The NN trained by NN Toolbox can only learn the information contained in the solutions of these algebraic equations and constructs an empirical equation that implicitly describes only the mass and energy balance equations corrupted by the white noise. However, the NN trained by the CNN methods learns the information contained in the mass and energy balances together with the external information imposed as equality constraints. Since the data are corrupted by the noise, NN Toolbox cannot satisfy the true mass and energy balances. On the other hand, the constraint learning can be accomplished with the CNN methods by employing the mass and energy balances as equality constraints.

In the fifth problem, the data were obtained from the solutions of the ODEs (mass and energy balance of the system) of a packed-bed reactor (PBR) for different inlet concentrations. However, the constraint information (the impracticability of reported

conversions at the condition of runaway temperatures) cannot be contained in these equations. Thus, the NN trained by NN Toolbox can only learn the information contained in the data obtained by the solution of the ODEs and constructs an empirical equation that implicitly describes only the mass and energy balance equations. However, the NN trained by the CNN methods can learn the information contained in the mass and energy balances together with the external information imposed as inequality constraint, and thus predicts practicable conversion values that are not at the condition of runaway temperatures.

In the sixth problem, the data were obtained from the solutions of the algebraic equations (mass balances) of the Williams Otto Plant (WOP) for different feed stream flow rate values of components A and B. The noise was added to the data in order to simulate realistic situation. The NN trained by NN Toolbox can only learn the information contained in the corrupted data, i.e., together with the noise added to the data. Since the data contain noise, NN Toolbox cannot satisfy the mass balance of the system. However, the NN trained by CNN methods learns the information contained in the mass balances via the external information imposed as an equality constraint.

Therefore, the CNNs successfully achieve the aim of the thesis as the results of the examples demonstrate. The CNNs can learn the data as well as constraints. Even in the testing phase, the constraints learnt by the CNNs are only rarely and mildly violated. The results prove that via the CNNs, one can implement equality and/or inequality constraints on/between the outputs of a feedforward NN during its training phase so that constraint violations of the model constructed by the NN are within the tolerable limits both in the training and testing phases.

5.1 Recommendations

In this thesis work, the CNN methods were developed to implement the equality and/or inequality constraints on/between the outputs of a feedforward NN during its training phase. These methods constitute a base to the future developments. First, they can be applied not only to feedforward NNs but to all known types of NNs as long as the NN training task is comprehended as an optimization problem.

The scope of the thesis was to develop the CNN methods. The effect of parameters used in the CNNs was not studied. The parameters taken in the thesis were adjusted by trial and error method. For example, the effect of penalty parameter in the indirect constraint implementation case can be studied and also the effect of penalty parameter on the distribution of weights and biases can be investigated.

Two methods of constraint implementation were used in the CNNs. More advanced penalty function techniques such as the exact penalty function technique can be implemented in the methods. The use of the exact penalty function technique may reduce the dependence of the optimal solution (optimal weights and biases) to the choice of the penalty parameter.

Additionally, the usage of regularization in the CNNs is beneficial in reducing overlearning (memorization) problem. The regularization method employed in the CNNs was described with details in Section 2.5.2 which is simpler with respect to the Bayesian regularization method. Bayesian regularization can be implemented in the CNN methods.

Since the NN training is data based, the constraints must be augmented for each row of the data. This explodes the number of constraints and puts a significant burden on the constrained optimizer for direct implementation of CNNs. As briefly introduced in Section 3.6.2, constraint aggregation technique, which nonlinearly and differentially aggregates the constraints into a single but highly nonlinear constraint, may be exploited and compared with the indirect implementation of the constrains via the penalty function method.

The MATLAB is functional using the NN Toolbox for building and pre-/post-processing a NN. Considering the robustness and variety of the optimization solvers of the GAMS software, a combination of MATLAB and GAMS may be beneficial by employing the MATLAB-GAMS interface. Furthermore, CNNs may be coded entirely in GAMS language to tackle constrained learning problems that involve integer or binary variables since GAMS has powerful Mixed-Integer Nonlinear Programming (MINLP) solvers.

APPENDIX: SOME SELECTED MATLAB CODES

A.1. Source Code of Direct Constraint Implementation of CNN-I for Section 4.1.1 of Synthetic Linear Time Series

(i) Main Code

```
clc; close all; clear all; format short; NNTWARN OFF;

global net I_Train T_Train N_Inputs N_Outputs
global is ts
global cnnfval

%-------------------------DATA PREPROCESSING----------------------------
[A]=xlsread('1Data.xls','gcex2');
x=A(:,1); y=A(:,2); z=A(:,3);
xn=x(3:end,:)'; yn=y(3:end,:)'; zn=z(3:end,:)';
xt1=x(2:(end-1),:)'; xt2=x(1:(end-2),:)';
yt1=y(2:(end-1),:)'; yt2=y(1:(end-2),:)';
zt1=z(2:(end-1),:)'; zt2=z(1:(end-2),:)';
%Series1 Data=>
% 1  2  3
% x  y  z
%           1    2    3    4    5    6                1  2  3
%INPUT==>>[Xt-1 Xt-2 Yt-1 Yt-2 Zt-1 Zt-2] TARGET==>>[Xt Yt Zt]

input = [xt1;xt2;yt1;yt2;zt1;zt2]; target = [xn; yn; zn];
N_Inputs=size(input,1)
N_Outputs=size(target,1)
DatWin=size(input,2)

%------------------------Data selection---------------------------------
r=randi(DatWin,DatWin,1);
input=input(:,r);
target=target(:,r);
[Itr Isp] = crossvalind('HoldOut', DatWin, 0.4);
I_Train(:,:)    = [input(:,Itr)];
I_Simulate(:,:) = [input(:,Isp)];
T_Train         = [target(:,Itr)];
T_Simulate      = [target(:,Isp)];
TrWin=size(I_Train,2)

% % %--- Second Alternative
% % %--- Randomly create training & testing sets
% % [Itr Isp] = crossvalind('HoldOut', DatWin, 0.5);
% % % NoOfTrainData=count(Itr==1)
% % % NoOfTestData=count(Isp==1)
% %
% % I = [DatWin(Itr,:)];
% % T = [DatWin(Itr,:)];
% % TrWin=length(I);
% %
```

```
% % % Adjustments for training & testing window sizes
% % I_Train(:,:)     = [DatWin(Itr,:)];
% % I_Simulate(:,:) = [DatWin(Isp,:)];
% % T_Train          = [DatWin(Itr,:)];
% % T_Simulate       = [DatWin(Isp,:)];
% % %---END Randomly create training & testing sets

% % %----- Third Alternative
% % TrWin=95
% % I_Train(:,:)     = [input(:,1:TrWin)];
% % I_Simulate(:,:) = [input(:,(TrWin+1):DatWin)];
% % T_Train          = [target(:,1:TrWin)];
% % T_Simulate       = [target(:,(TrWin+1):DatWin)];

% %-----------------Scaling Data-----------------------------------------
[I_Train,is]=mapminmax(I_Train);
[T_Train,ts]=mapminmax(T_Train);
[I_Simulate]=mapminmax('apply',I_Simulate,is);
[T_Simulate]=mapminmax('apply',T_Simulate,ts);
% %--- Scaling by mapstd
% [I_Train,is]=mapstd(I_Train);
% [T_Train,ts]=mapstd(T_Train);
% [I_Simulate]=mapstd('apply',I_Simulate,is);
% [T_Simulate]=mapstd('apply',T_Simulate,ts);
%---------------------------------------------------------------------------

%------------------CONFIGURING NETWORK-----------------------------------
net = newff(repmat([-1 1],N_Inputs,1), [12 N_Outputs],...
           {'tansig' 'purelin'},'trainbr','','sse');
%---------------------------------------------------------------------------
unetIW=net.IW
unetLW=net.LW
unetB=net.b
%------------------PRE-TRAINING NETWORK (OPTIONAL)----------------------
net.trainParam.epochs   = 1000;
net.trainParam.min_grad = 1.e-12;
net.initFcn='initnw';

x0=getx(net);

tic
netbr = train(net,I_Train,T_Train);
toc

sNTTxTr = sim(netbr,I_Train);
sNTTxSim = sim(netbr,I_Simulate);

figure
plotregression([T_Train T_Simulate],[sNTTxTr sNTTxSim])
figure
plotregression(T_Train(1,:),sNTTxTr(1,:))
figure
plotregression(T_Train(2,:),sNTTxTr(2,:))
figure
plotregression(T_Train(3,:),sNTTxTr(3,:))
figure
plotregression(T_Simulate(1,:),sNTTxSim(1,:))
figure
plotregression(T_Simulate(2,:),sNTTxSim(2,:))
figure
```

```
plotregression(T_Simulate(3,:),sNTTxSim(3,:))
%-------------------------------------------------------------------------
%----------------------NETWORK TRAINING-----------------------------------
cnnfval = [];
options=optimset('Display','iter','MaxFunEvals',500000,'MaxIter',1000,...
                 'Algorithm','sqp',...
                 'TolFun',1.E-9,'TolX',1.E-6,'TolCon',1.E-2,...
                 'PlotFcns',{@optimplotfval},...
                 'outputfcn',@outfun);
tic
[x,fval,exitflag,output] =
fmincon(@CNN1ObjFct,x0,[],[],[],[],[],[],@CNN1Cons,options)
toc
%-------------------------------------------------------------------------

%------------------SIMULATION OF NETWORK AND PLOTS----------------------
net=setx(net,x);
sNT = sim(net,I_Train);
sNTsim = sim(net,I_Simulate);
sNTbr = sim(netbr,I_Train);
sNTsimbr = sim(netbr,I_Simulate);
%-------------------Unscaling the Data-----------------------------------
I_Train = mapminmax('reverse',I_Train,is);
T_Train = mapminmax('reverse',T_Train,ts);
I_Simulate = mapminmax('reverse',I_Simulate,is);
T_Simulate = mapminmax('reverse',T_Simulate,ts);
sNT = mapminmax('reverse',sNT,ts);
sNTsim = mapminmax('reverse',sNTsim,ts);
sNTbr = mapminmax('reverse',sNTbr,ts);
sNTsimbr = mapminmax('reverse',sNTsimbr,ts);
% I_Train = mapstd('reverse',I_Train,is);
% T_Train = mapstd('reverse',T_Train,ts);
% I_Simulate = mapstd('reverse',I_Simulate,is);
% T_Simulate = mapstd('reverse',T_Simulate,ts);
% sNT = mapstd('reverse',sNT,ts);
% sNTsim = mapstd('reverse',sNTsim,ts);
% sNTbr = mapstd('reverse',sNTbr,ts);
% sNTsimbr = mapstd('reverse',sNTsimbr,ts);
%-------------------------------------------------------------------------
AllOutput=[sNT sNTsim];
AllOutputbr=[sNTbr sNTsimbr];
AllInput=[I_Train I_Simulate];
AllTarget=[T_Train T_Simulate];

figure
plot(AllTarget(1,:),'r.'); hold on; plot(AllOutput(1,:),'b'); hold on;
plot(AllOutputbr(1,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(1,:))
max(AllOutputbr(1,:))],'LineWidth',1,'Color','c');
line([0 DatWin],[-0.1 -0.1],'LineWidth',1,'Color','c');
legend('Target','CNN-I','NN Toolbox',2,'Location','Best');
set(legend,'FontSize',12);
xlabel('Data Number','FontSize',12);ylabel('X','FontSize',12);
set(gca,'FontSize',12)
figure
subplot(2,1,1)
plot(AllTarget(2,:),'r.'); hold on; plot(AllOutput(2,:),'b'); hold on;
plot(AllOutputbr(2,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(2,:))
max(AllOutputbr(2,:))],'LineWidth',1,'Color','c');
```

```
xlabel('Data Number','FontSize',12);ylabel('Y','FontSize',12);
legend('Target','CNN-I','NN Toolbox',2,'Location','Best');
set(legend,'FontSize',12);
set(gca,'FontSize',12)
subplot(2,1,2)
plot(AllTarget(3,:),'r.'); hold on; plot(AllOutput(3,:),'b'); hold on;
plot(AllOutputbr(3,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(3,:))
max(AllOutputbr(3,:))],'LineWidth',1,'Color','c');
xlabel('Data Number','FontSize',12);ylabel('Z','FontSize',12);
set(gca,'FontSize',12)

consviotr=-sNT(1,:)-0.1;
consviosim=-sNTsim(1,:)-0.1;
consvioTxtr=-sNTbr(1,:)-0.1;
consvioTxsim=-sNTsimbr(1,:)-0.1;

Norm_CNNI=norm(max(0,[consviotr consviosim]))
Norm_Tx=norm(max(0,[consvioTxtr consvioTxsim]))

Norm_consviotr=norm(max(0,consviotr))
Norm_consviosim=norm(max(0,consviosim))
Norm_consvioTxtr=norm(max(0,consvioTxtr))
Norm_consvioTxsim=norm(max(0,consvioTxsim))

figure
plotregression(AllTarget,AllOutput)
figure
plotregression(T_Train(1,:),sNT(1,:))
figure
plotregression(T_Train(2,:),sNT(2,:))
figure
plotregression(T_Train(3,:),sNT(3,:))
figure
plotregression(T_Simulate(1,:),sNTsim(1,:))
figure
plotregression(T_Simulate(2,:),sNTsim(2,:))
figure
plotregression(T_Simulate(3,:),sNTsim(3,:))

sseTxtr = sum(sum((T_Train-sNTbr).^2,2),1)
sum((T_Train-sNTbr).^2,2)
sseTxsim = sum(sum((T_Simulate-sNTsimbr).^2,2),1)
sum((T_Simulate-sNTsimbr).^2,2)
sseCNNtr = sum(sum((T_Train-sNT).^2,2),1)
sum((T_Train-sNT).^2,2)
sseCNNsim = sum(sum((T_Simulate-sNTsim).^2,2),1)
sum((T_Simulate-sNTsim).^2,2)
```

(ii) Objective Function Code

```
function obj = CNN1ObjFct(x)

global net I_Train T_Train N_Inputs N_Outputs
global is ts

Weights=x;
net=setx(net,Weights);
```

```
Outputs = sim(net,I_Train);

err=(Outputs-T_Train);

obj=sse(err);
%obj=mse(err);

%-------------- Regularization ----------------------------------------
RegParam=1.E-4;
N=length(Weights);
obj = obj + RegParam*sum(Weights.^2)/N;
%-END- Regularization
```

(iii) Constraint Function Code

```
function [c,ceq] = CNN1Cons(x)

global net I_Train T_Train N_Inputs N_Outputs
global is ts

% x=x'; %for the ga function

Weights=x;
net=setx(net,Weights);
Outputs = sim(net,I_Train);

%--- Unscale to compute constraints
UScaleOutputs = mapminmax('reverse',Outputs,ts);
% UScaleOutputs = mapstd('reverse',Outputs,ts);
% %-END- Unscale to compute constraints

% Compute nonlinear inequalities at x.
c = [-UScaleOutputs(1,:)-0.1];

% Compute nonlinear equalities at x.
ceq = [];


% %--- Sum of all constraint (h=0) violations: single constraint but
nondifferentiable
% ceq=max(max(0,abs(ceq)));
% %-END Sum of all constraint (h=0) violations: single constraint but
nondifferentiable

% %--- Constraint Aggregation of all constraints (h=0) & differentiable
% %    h <= 0 & -h <= 0
% R=2;
% ceq=(1/R)*log( sum(exp(R*ceq)) + sum(exp(-R*ceq)) );
% %-END Constraint Aggregation


% %--- Sum of all constraint (g<0) violations: single constraint but
nondifferentiable
% c=sum(max(0,c));
% %-END Sum of all constraint (g<0) violations: single constraint but
nondifferentiable

% %--- Maximum of all constraint (g<0) violations: single constraint
```

```
% c=max((max(0,c)));
% %-END Maximum of all constraint (g<0) violations: single constraint

% %--- Constraint Aggregation of all constraints (g<0) & differentiable
% R=50;
% c=(1/R)*log(sum(exp(R*c)));
% %-END Constraint Aggregation

% size(c)
```

(iv) Output Function Code

```
function stop = outfun(x,optimValues,state)

global cnnfval

stop = false;

     switch state
         case 'init'
             hold on
         case 'iter'
           cnnfval = [cnnfval; optimValues.fval];
         case 'done'
             hold off
         otherwise
     end
 end
```

A.2. Source Code of Indirect Constraint Implementation of CNN-I for Section 4.1.1 of Synthetic Linear Time Series

(i) Main Code

```
clc; close all; clear all; format short; NNTWARN OFF;

global net I_Train T_Train N_Inputs N_Outputs
global is ts
global cnnfval

%---------------------------DATA PREPROCESSING-----------------------------
[A]=xlsread('1Data.xls','gcex2');
x=A(:,1); y=A(:,2); z=A(:,3);
xn=x(3:end,:)'; yn=y(3:end,:)'; zn=z(3:end,:)';
xt1=x(2:(end-1),:)'; xt2=x(1:(end-2),:)';
yt1=y(2:(end-1),:)'; yt2=y(1:(end-2),:)';
zt1=z(2:(end-1),:)'; zt2=z(1:(end-2),:)';
%Makale 1 Data=>
% 1  2  3
% x  y  z
%           1    2    3    4    5    6                1  2  3
%INPUT==>>[Xt-1 Xt-2 Yt-1 Yt-2 Zt-1 Zt-2] TARGET==>>[Xt Yt Zt]
```

```
input = [xt1;xt2;yt1;yt2;zt1;zt2]; target = [xn; yn; zn];
N_Inputs=size(input,1)
N_Outputs=size(target,1)
DatWin=size(input,2)

%-------------------------Data selection---------------------------------
r=randi(DatWin,DatWin,1);
input=input(:,r);
target=target(:,r);
[Itr Isp] = crossvalind('HoldOut', DatWin, 0.4);
I_Train(:,:)    = [input(:,Itr)];
I_Simulate(:,:) = [input(:,Isp)];
T_Train         = [target(:,Itr)];
T_Simulate      = [target(:,Isp)];
TrWin=size(I_Train,2)

% % %--- Second Alternative
% % %--- Randomly create training & testing sets
% % [Itr Isp] = crossvalind('HoldOut', DatWin, 0.5);
% % % NoOfTrainData=count(Itr==1)
% % % NoOfTestData=count(Isp==1)
% %
% % I = [DatWin(Itr,:)];
% % T = [DatWin(Itr,:)];
% % TrWin=length(I);
% %
% % % Adjustments for training & testing window sizes
% % I_Train(:,:)    = [DatWin(Itr,:)];
% % I_Simulate(:,:) = [DatWin(Isp,:)];
% % T_Train         = [DatWin(Itr,:)];
% % T_Simulate      = [DatWin(Isp,:)];
% % %---END Randomly create training & testing sets

% % %----- Third Alternative
% % TrWin=95
% % I_Train(:,:)    = [input(:,1:TrWin)];
% % I_Simulate(:,:) = [input(:,(TrWin+1):DatWin)];
% % T_Train         = [target(:,1:TrWin)];
% % T_Simulate      = [target(:,(TrWin+1):DatWin)];

% %-----------------Scaling Data-------------------------------------------
[I_Train,is]=mapminmax(I_Train);
[T_Train,ts]=mapminmax(T_Train);
[I_Simulate]=mapminmax('apply',I_Simulate,is);
[T_Simulate]=mapminmax('apply',T_Simulate,ts);
% %--- Scaling by mapstd
% [I_Train,is]=mapstd(I_Train);
% [T_Train,ts]=mapstd(T_Train);
% [I_Simulate]=mapstd('apply',I_Simulate,is);
% [T_Simulate]=mapstd('apply',T_Simulate,ts);
%-----------------------------------------------------------------------

%-------------------CONFIGURING NETWORK---------------------------------
net = newff(repmat([-1 1],N_Inputs,1), [12 N_Outputs],...
          {'tansig' 'purelin'},'trainbr','','sse');
%-----------------------------------------------------------------------
unetIW=net.IW
unetLW=net.LW
unetB=net.b
%-------------------PRE-TRAINING NETWORK (OPTIONAL)---------------------
```

```
net.trainParam.epochs    = 1000;
net.trainParam.min_grad = 1.e-12;
net.initFcn='initnw';

x0=getx(net);

tic
netbr = train(net,I_Train,T_Train);
toc

sNTTxTr = sim(netbr,I_Train);
sNTTxSim = sim(netbr,I_Simulate);

figure
plotregression([T_Train T_Simulate],[sNTTxTr sNTTxSim])
figure
plotregression(T_Train(1,:),sNTTxTr(1,:))
figure
plotregression(T_Train(2,:),sNTTxTr(2,:))
figure
plotregression(T_Train(3,:),sNTTxTr(3,:))
figure
plotregression(T_Simulate(1,:),sNTTxSim(1,:))
figure
plotregression(T_Simulate(2,:),sNTTxSim(2,:))
figure
plotregression(T_Simulate(3,:),sNTTxSim(3,:))
%-------------------------------------------------------------------------
%----------------------NETWORK TRAINING-----------------------------------

cnnfval = [];
options =
optimset('LargeScale','off','MaxFunEvals',500000,'MaxIter',200,...
                    'Display','iter','TolFun',1.E-
12,'HessUpdate','bfgs',...
                    'PlotFcns',{@optimplotfval},...
                    'outputfcn',@outfun);
tic
[x,fval,exitflag,output] = fminunc(@CNN1ObjFct, x0, options)
toc
%-------------------------------------------------------------------------

%------------------SIMULATION OF NETWORK AND PLOTS----------------------
net=setx(net,x);
sNT = sim(net,I_Train);
sNTsim = sim(net,I_Simulate);
sNTbr = sim(netbr,I_Train);
sNTsimbr = sim(netbr,I_Simulate);
%------------------Unscaling the Data-----------------------------------
I_Train = mapminmax('reverse',I_Train,is);
T_Train = mapminmax('reverse',T_Train,ts);
I_Simulate = mapminmax('reverse',I_Simulate,is);
T_Simulate = mapminmax('reverse',T_Simulate,ts);
sNT = mapminmax('reverse',sNT,ts);
sNTsim = mapminmax('reverse',sNTsim,ts);
sNTbr = mapminmax('reverse',sNTbr,ts);
sNTsimbr = mapminmax('reverse',sNTsimbr,ts);
% I_Train = mapstd('reverse',I_Train,is);
% T_Train = mapstd('reverse',T_Train,ts);
% I_Simulate = mapstd('reverse',I_Simulate,is);
```

```
% T_Simulate = mapstd('reverse',T_Simulate,ts);
% sNT = mapstd('reverse',sNT,ts);
% sNTsim = mapstd('reverse',sNTsim,ts);
%-------------------------------------------------------------------------
AllOutput=[sNT sNTsim];
AllOutputbr=[sNTbr sNTsimbr];
AllInput=[I_Train I_Simulate];
AllTarget=[T_Train T_Simulate];

figure
plot(AllTarget(1,:),'r.'); hold on; plot(AllOutput(1,:),'b'); hold on;
plot(AllOutputbr(1,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(1,:))
max(AllOutputbr(1,:))],'LineWidth',1,'Color','c');
line([0 DatWin],[-0.1 -0.1],'LineWidth',1,'Color','c');
legend('Target','CNN-I','NN Toolbox',2,'Location','Best');
set(legend,'FontSize',12);
xlabel('Data Number','FontSize',12);ylabel('X','FontSize',12);
set(gca,'FontSize',12)
figure
subplot(2,1,1)
plot(AllTarget(2,:),'r.'); hold on; plot(AllOutput(2,:),'b'); hold on;
plot(AllOutputbr(2,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(2,:))
max(AllOutputbr(2,:))],'LineWidth',1,'Color','c');
xlabel('Data Number','FontSize',12);ylabel('Y','FontSize',12);
legend('Target','CNN-I','NN Toolbox',2,'Location','Best');
set(legend,'FontSize',12);
set(gca,'FontSize',12)
subplot(2,1,2)
plot(AllTarget(3,:),'r.'); hold on; plot(AllOutput(3,:),'b'); hold on;
plot(AllOutputbr(3,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(3,:))
max(AllOutputbr(3,:))],'LineWidth',1,'Color','c');
xlabel('Data Number','FontSize',12);ylabel('Z','FontSize',12);
set(gca,'FontSize',12)

consviotr=-sNT(1,:)-0.1;
consviosim=-sNTsim(1,:)-0.1;
consvioTxtr=-sNTbr(1,:)-0.1;
consvioTxsim=-sNTsimbr(1,:)-0.1;

Norm_CNNI=norm(max(0,[consviotr consviosim]))
Norm_Tx=norm(max(0,[consvioTxtr consvioTxsim]))

Norm_consviotr=norm(max(0,consviotr))
Norm_consviosim=norm(max(0,consviosim))
Norm_consvioTxtr=norm(max(0,consvioTxtr))
Norm_consvioTxsim=norm(max(0,consvioTxsim))

figure
plotregression(AllTarget,AllOutput)
figure
plotregression(T_Train(1,:),sNT(1,:))
figure
plotregression(T_Train(2,:),sNT(2,:))
figure
plotregression(T_Train(3,:),sNT(3,:))
figure
plotregression(T_Simulate(1,:),sNTsim(1,:))
```

```
figure
plotregression(T_Simulate(2,:),sNTsim(2,:))
figure
plotregression(T_Simulate(3,:),sNTsim(3,:))

sseTxtr = sum(sum((T_Train-sNTbr).^2,2),1)
sum((T_Train-sNTbr).^2,2)
sseTxsim = sum(sum((T_Simulate-sNTsimbr).^2,2),1)
sum((T_Simulate-sNTsimbr).^2,2)
sseCNNtr = sum(sum((T_Train-sNT).^2,2),1)
sum((T_Train-sNT).^2,2)
sseCNNsim = sum(sum((T_Simulate-sNTsim).^2,2),1)
sum((T_Simulate-sNTsim).^2,2)
```

(ii) Objective Function Code

```
function obj = CNN1ObjFct(x)

global net I_Train T_Train N_Inputs N_Outputs
global is ts

Weights=x;
net=setx(net,Weights);
Outputs = sim(net,I_Train);

err=(Outputs-T_Train);

obj=sse(err);
%obj=mse(err);

%----------------------Penalty Section-----------------------------------
%--- Unscale to compute constraints
UScaleOutputs = mapminmax('reverse',Outputs,ts);
% UScaleOutputs = mapstd('reverse',Outputs,ts);
% %-END- Unscale to compute constraints

r=10; % Penalty Parameters for two outputs
g1 = [-UScaleOutputs(1,:)-0.1]; % inequality constraints
% h = [cons]; % equality constraint

obj = obj + r * sum(sum(max(0,g1),2),1); % Only ineq cons.
% obj = obj + r * (h*h')^0.5; % Eq. cons.
% obj = obj + r * ((h*h')^0.5 + sum(sum(max(0,g1),2),1)); % ineq+eq cons.
%-------------------------------------------------------------------------

%-------------- Regularization ----------------------------------------
RegParam=1.E-4;
N=length(Weights);
obj = obj + RegParam*sum(Weights.^2)/N;
%-END- Regularization
```

(iii) Output Function Code

```
function stop = outfun(x,optimValues,state)
```

```
global cnnfval

stop = false;

     switch state
         case 'init'
             hold on
         case 'iter'
           cnnfval = [cnnfval; optimValues.fval];
         case 'done'
             hold off
         otherwise
     end
 end
```

A.3. Source Code of Direct Constraint Implementation of CNN-II for Section 4.1.1 of Synthetic Linear Time Series

(i) Main Code

```
clc; close all; clear all; format short g; warning off;

global N_HiddenLayers N_Inputs N_NeuronsHiddenLayers N_Outputs
TransferFunctions
global TrWin I_Train T_Train
global is ts
global cnnfval

%-------------------------DATA PREPROCESSING-----------------------------
[A]=xlsread('1Data.xls','gcex2');
x=A(:,1); y=A(:,2); z=A(:,3);
xn=x(3:end,:)'; yn=y(3:end,:)'; zn=z(3:end,:)';
xt1=x(2:(end-1),:)'; xt2=x(1:(end-2),:)';
yt1=y(2:(end-1),:)'; yt2=y(1:(end-2),:)';
zt1=z(2:(end-1),:)'; zt2=z(1:(end-2),:)';
%Makale 1 Data=>
% 1  2  3
% x  y  z
%           1    2    3    4    5    6                1  2  3
%INPUT==>>[Xt-1 Xt-2 Yt-1 Yt-2 Zt-1 Zt-2] TARGET==>>[Xt Yt Zt]

input = [xt1;xt2;yt1;yt2;zt1;zt2]; target = [xn; yn; zn];
N_Inputs=size(input,1)
N_Outputs=size(target,1)
DatWin=size(input,2)

%-------------------------Data selection---------------------------------
r=randi(DatWin,DatWin,1);
input=input(:,r);
target=target(:,r);
[Itr Isp] = crossvalind('HoldOut', DatWin, 0.4);
I_Train(:,:)     = [input(:,Itr)];
I_Simulate(:,:)  = [input(:,Isp)];
T_Train          = [target(:,Itr)];
T_Simulate       = [target(:,Isp)];
```

```
TrWin=size(I_Train,2)

% % %--- Second Alternative
% % %--- Randomly create training & testing sets
% % [Itr Isp] = crossvalind('HoldOut', DatWin, 0.5);
% % % NoOfTrainData=count(Itr==1)
% % % NoOfTestData=count(Isp==1)
% %
% % I = [DatWin(Itr,:)];
% % T = [DatWin(Itr,:)];
% % TrWin=length(I);
% %
% % % Adjustments for training & testing window sizes
% % I_Train(:,:)     = [DatWin(Itr,:)];
% % I_Simulate(:,:) = [DatWin(Isp,:)];
% % T_Train          = [DatWin(Itr,:)];
% % T_Simulate       = [DatWin(Isp,:)];
% % %---END Randomly create training & testing sets

% % %----- Third Alternative
% % TrWin=95
% % I_Train(:,:)     = [input(:,1:TrWin)];
% % I_Simulate(:,:) = [input(:,(TrWin+1):DatWin)];
% % T_Train          = [target(:,1:TrWin)];
% % T_Simulate       = [target(:,(TrWin+1):DatWin)];

% %-----------------Scaling Data-----------------------------------------
[I_Train,is]=mapminmax(I_Train);
[T_Train,ts]=mapminmax(T_Train);
[I_Simulate]=mapminmax('apply',I_Simulate,is);
[T_Simulate]=mapminmax('apply',T_Simulate,ts);
% %--- Scaling by mapstd
% [I_Train,is]=mapstd(I_Train);
% [T_Train,ts]=mapstd(T_Train);
% [I_Simulate]=mapstd('apply',I_Simulate,is);
% [T_Simulate]=mapstd('apply',T_Simulate,ts);
%---------------------------------------------------------------------------

%------------------CONFIGURING NETWORK-----------------------------------
net = newff(repmat([-1 1],N_Inputs,1), [12 N_Outputs],...
          {'tansig' 'purelin'},'trainbr','','sse');
%---------------------------------------------------------------------------
unetIW=net.IW
unetLW=net.LW
unetB=net.b
%------------------PRE-TRAINING NETWORK (OPTIONAL)----------------------
net.trainParam.epochs   = 1000;
net.trainParam.min_grad = 1.e-12;
net.initFcn='initnw';

tic
[net,tr] = train(net,I_Train,T_Train);
toc
xxx=tr.epoch;
yyy=tr.perf;

sNTTxTr = sim(net,I_Train);
sNTTxSim = sim(net,I_Simulate);

figure
```

```
plotregression([T_Train T_Simulate],[sNTTxTr sNTTxSim])
figure
plotregression(T_Train(1,:),sNTTxTr(1,:))
figure
plotregression(T_Train(2,:),sNTTxTr(2,:))
figure
plotregression(T_Train(3,:),sNTTxTr(3,:))
figure
plotregression(T_Simulate(1,:),sNTTxSim(1,:))
figure
plotregression(T_Simulate(2,:),sNTTxSim(2,:))
figure
plotregression(T_Simulate(3,:),sNTTxSim(3,:))

%-------------------NETWORK CONFIGURATION AND INITIALIZATION-----------
N_NeuronsHiddenLayers=[12 N_Outputs]; % Each number correspond to a layer
and the corresponding neuron number in this layer
N_HiddenLayers=length(N_NeuronsHiddenLayers);
TransferFunctions={'TANSIG' 'PURLIN'}; %Transfer Functions can be TANSIG
LOGSIG TANHYP PURLIN

[W, B]=UANNETInitialization;

Weights0 = UANNETGetWeights(W,B);

%-------------------------------------------------------------------------

%---------------------NETWORK TRAINING-----------------------------------
x0=Weights0;

cnnfval = [];
options=optimset('Display','iter','MaxFunEvals',500000,'MaxIter',200,...
                 'Algorithm','sqp',...
                 'TolFun',1.E-9,'TolX',1.E-6,'TolCon',1.E-2,...
                 'PlotFcns',{@optimplotfval},...
                 'outputfcn',@outfun);
tic
[x,fval,exitflag,output] =
fmincon(@UANNetFct,x0,[],[],[],[],[],[],@UANNetCons,options)
toc
%-------------------------------------------------------------------------

%-----------------------------PLOTS---------------------------------------

Weights=x;

%-----------------------Simulation of Network-----------------------------
Inputs = I_Train;
sNT = UANNetSim(Inputs, Weights);
sNTbr = sim(net,Inputs);
%---
Inputs = I_Simulate;
sNTsim = UANNetSim(Inputs, Weights);
sNTsimbr = sim(net,Inputs);
%-------------------------------------------------------------------------
%------------------Unscaling the Data-------------------------------------
I_Train = mapminmax('reverse',I_Train,is);
T_Train = mapminmax('reverse',T_Train,ts);
I_Simulate = mapminmax('reverse',I_Simulate,is);
T_Simulate = mapminmax('reverse',T_Simulate,ts);
```

```
sNT = mapminmax('reverse',sNT,ts);
sNTsim = mapminmax('reverse',sNTsim,ts);
sNTbr = mapminmax('reverse',sNTbr,ts);
sNTsimbr = mapminmax('reverse',sNTsimbr,ts);
% I_Train = mapstd('reverse',I_Train,is);
% T_Train = mapstd('reverse',T_Train,ts);
% I_Simulate = mapstd('reverse',I_Simulate,is);
% T_Simulate = mapstd('reverse',T_Simulate,ts);
% sNT = mapstd('reverse',sNT,ts);
% sNTsim = mapstd('reverse',sNTsim,ts);
%-------------------------------------------------------------------------
AllOutput=[sNT sNTsim];
AllOutputbr=[sNTbr sNTsimbr];
AllInput=[I_Train I_Simulate];
AllTarget=[T_Train T_Simulate];

figure
plot(AllTarget(1,:),'r.'); hold on; plot(AllOutput(1,:),'b'); hold on;
plot(AllOutputbr(1,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(1,:))
max(AllOutputbr(1,:))],'LineWidth',1,'Color','c');
line([0 DatWin],[-0.1 -0.1],'LineWidth',1,'Color','c');
legend('Target','CNN-II','NN Toolbox',2,'Location','Best');
set(legend,'FontSize',12);
xlabel('Data Number','FontSize',12);ylabel('X','FontSize',12);
set(gca,'FontSize',12)
figure
subplot(2,1,1)
plot(AllTarget(2,:),'r.'); hold on; plot(AllOutput(2,:),'b'); hold on;
plot(AllOutputbr(2,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(2,:))
max(AllOutputbr(2,:))],'LineWidth',1,'Color','c');
xlabel('Data Number','FontSize',12);ylabel('Y','FontSize',12);
legend('Target','CNN-II','NN Toolbox',2,'Location','Best');
set(legend,'FontSize',12);
set(gca,'FontSize',12)
subplot(2,1,2)
plot(AllTarget(3,:),'r.'); hold on; plot(AllOutput(3,:),'b'); hold on;
plot(AllOutputbr(3,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(3,:))
max(AllOutputbr(3,:))],'LineWidth',1,'Color','c');
xlabel('Data Number','FontSize',12);ylabel('Z','FontSize',12);
set(gca,'FontSize',12)

consviotr=-sNT(1,:)-0.1;
consviosim=-sNTsim(1,:)-0.1;
consvioTxtr=-sNTbr(1,:)-0.1;
consvioTxsim=-sNTsimbr(1,:)-0.1;

Norm_CNNI=norm(max(0,[consviotr consviosim]))
Norm_Tx=norm(max(0,[consvioTxtr consvioTxsim]))

Norm_consviotr=norm(max(0,consviotr))
Norm_consviosim=norm(max(0,consviosim))
Norm_consvioTxtr=norm(max(0,consvioTxtr))
Norm_consvioTxsim=norm(max(0,consvioTxsim))

figure
plotregression(AllTarget,AllOutput)
figure
```

```
plotregression(T_Train(1,:),sNT(1,:))
figure
plotregression(T_Train(2,:),sNT(2,:))
figure
plotregression(T_Train(3,:),sNT(3,:))
figure
plotregression(T_Simulate(1,:),sNTsim(1,:))
figure
plotregression(T_Simulate(2,:),sNTsim(2,:))
figure
plotregression(T_Simulate(3,:),sNTsim(3,:))

sseTxtr = sum(sum((T_Train-sNTbr).^2,2),1)
sum((T_Train-sNTbr).^2,2)
sseTxsim = sum(sum((T_Simulate-sNTsimbr).^2,2),1)
sum((T_Simulate-sNTsimbr).^2,2)
sseCNNtr = sum(sum((T_Train-sNT).^2,2),1)
sum((T_Train-sNT).^2,2)
sseCNNsim = sum(sum((T_Simulate-sNTsim).^2,2),1)
sum((T_Simulate-sNTsim).^2,2)
```

(ii) Objective Function Code

```
function obj = UANNetFct(Weights)

global TrWin I_Train T_Train
global is ts

Inputs = I_Train;
Outputs = UANNetSim(Inputs, Weights);

obj = sum(sum((T_Train-Outputs).^2,2),1);

%-------------- Regularization -----------------------------------------
RegParam=1.E-4;
N=length(Weights);
% obj = 1/N*obj + RegParam*sum(Weights.^2);
obj = obj + RegParam*sum(Weights.^2)/N;
%-END- Regularization
```

(iii) Constraints Function Code

```
function [c,ceq] = UANNetCons(x)

global TrWin I_Train T_Train
global is ts

Weights = x;
Inputs = I_Train;
Outputs = UANNetSim(Inputs, Weights);

% %------ Unscale to compute constraints -------------------------------
UScaleOutputs = mapminmax('reverse',Outputs,ts);
% UScaleOutputs = mapstd('reverse',Outputs,ts);
% %--- END --- Unscale to compute constraints ---------------------------
```

```
% Compute nonlinear inequalities at x.
c = [-UScaleOutputs(1,:)-0.1];

% Compute nonlinear equalities at x.
ceq = [];

% %--- Sum of all constraint (h=0) violations: single constraint but
nordifferentiable
% ceq=max(max(0,abs(ceq)));
% %-END Sum of all constraint (h=0) violations: single constraint but
nondifferentiable

% %--- Constraint Aggregation of all constraints (h=0) & differentiable
% %    h <= 0 & -h <= 0
% R=2;
% ceq=(1/R)*log( sum(exp(R*ceq)) + sum(exp(-R*ceq)) );
% %-END Constraint Aggregation

% %--- Sum of all constraint (g<0) violations: single constraint but
nondifferentiable
% c=sum(max(0,c));
% %-END Sum of all constraint (g<0) violations: single constraint but
nondifferentiable

% %--- Maximum of all constraint (g<0) violations: single constraint
% c=max((max(0,c)));
% %-END Maximum of all constraint (g<0) violations: single constraint

% %--- Constraint Aggregation of all constraints (g<0) & differentiable
% R=50;
% c=(1/R)*log(sum(exp(R*c)));
% %-END Constraint Aggregation

% size(c)
```

A.4. Source Code of Indirect Constraint Implementation of CNN-II for Section 4.1.1 of Synthetic Linear Time Series

(i) Main Code

```
clc; close all; clear all; format short g; warning off;

global N_HiddenLayers N_Inputs N_NeuronsHiddenLayers N_Outputs
TransferFunctions
global TrWin I_Train T_Train
global is ts
global cnnfval

%-------------------------DATA PREPROCESSING-----------------------------
[A]=xlsread('1Data.xls','gcex2');
x=A(:,1); y=A(:,2); z=A(:,3);
xn=x(3:end,:)'; yn=y(3:end,:)'; zn=z(3:end,:)';
xt1=x(2:(end-1),:)'; xt2=x(1:(end-2),:)';
yt1=y(2:(end-1),:)'; yt2=y(1:(end-2),:)';
```

```
zt1=z(2:(end-1),:)'; zt2=z(1:(end-2),:)';
%Makale 1 Data=>
% 1  2  3
% x  y  z
%           1    2    3    4    5    6                1  2  3
%INPUT==>>[Xt-1 Xt-2 Yt-1 Yt-2 Zt-1 Zt-2] TARGET==>>[Xt Yt Zt]

input = [xt1;xt2;yt1;yt2;zt1;zt2]; target = [xn; yn; zn];
N_Inputs=size(input,1)
N_Outputs=size(target,1)
DatWin=size(input,2)

%-------------------------Data selection---------------------------------
r=randi(DatWin,DatWin,1);
input=input(:,r);
target=target(:,r);
[Itr Isp] = crossvalind('HoldOut', DatWin, 0.4);
I_Train(:,:)    = [input(:,Itr)];
I_Simulate(:,:) = [input(:,Isp)];
T_Train         = [target(:,Itr)];
T_Simulate      = [target(:,Isp)];
TrWin=size(I_Train,2)

% % %--- Second Alternative
% % %--- Randomly create training & testing sets
% % [Itr Isp] = crossvalind('HoldOut', DatWin, 0.5);
% % % NoOfTrainData=count(Itr==1)
% % % NoOfTestData=count(Isp==1)
% %
% % I = [DatWin(Itr,:)];
% % T = [DatWin(Itr,:)];
% % TrWin=length(I);
% %
% % % Adjustments for training & testing window sizes
% % I_Train(:,:)    = [DatWin(Itr,:)];
% % I_Simulate(:,:) = [DatWin(Isp,:)];
% % T_Train         = [DatWin(Itr,:)];
% % T_Simulate      = [DatWin(Isp,:)];
% % %---END Randomly create training & testing sets

% % %----- Third Alternative
% % TrWin=95
% % I_Train(:,:)    = [input(:,1:TrWin)];
% % I_Simulate(:,:) = [input(:,(TrWin+1):DatWin)];
% % T_Train         = [target(:,1:TrWin)];
% % T_Simulate      = [target(:,(TrWin+1):DatWin)];

% %------------------Scaling Data-----------------------------------------
[I_Train,is]=mapminmax(I_Train);
[T_Train,ts]=mapminmax(T_Train);
[I_Simulate]=mapminmax('apply',I_Simulate,is);
[T_Simulate]=mapminmax('apply',T_Simulate,ts);
% %--- Scaling by mapstd
% [I_Train,is]=mapstd(I_Train);
% [T_Train,ts]=mapstd(T_Train);
% [I_Simulate]=mapstd('apply',I_Simulate,is);
% [T_Simulate]=mapstd('apply',T_Simulate,ts);
%------------------------------------------------------------------------

%-------------------CONFIGURING NETWORK----------------------------------
```

```
net = newff(repmat([-1 1],N_Inputs,1), [12 N_Outputs],...
          {'tansig' 'purelin'},'trainbr','','sse');
%-------------------------------------------------------------------------
unetIW=net.IW
unetLW=net.LW
unetB=net.b
%-------------------PRE-TRAINING NETWORK (OPTIONAL)----------------------
net.trainParam.epochs   = 1000;
net.trainParam.min_grad = 1.e-12;
net.initFcn='initnw';

tic
[net,tr] = train(net,I_Train,T_Train);
toc
xxx=tr.epoch;
yyy=tr.perf;

sNTTxTr = sim(net,I_Train);
sNTTxSim = sim(net,I_Simulate);

figure
plotregression([T_Train T_Simulate],[sNTTxTr sNTTxSim])
figure
plotregression(T_Train(1,:),sNTTxTr(1,:))
figure
plotregression(T_Train(2,:),sNTTxTr(2,:))
figure
plotregression(T_Train(3,:),sNTTxTr(3,:))
figure
plotregression(T_Simulate(1,:),sNTTxSim(1,:))
figure
plotregression(T_Simulate(2,:),sNTTxSim(2,:))
figure
plotregression(T_Simulate(3,:),sNTTxSim(3,:))
%--------------------NETWORK CONFIGURATION AND INITIALIZATION------------
N_NeuronsHiddenLayers=[12 N_Outputs]; % Each number correspond to a layer
and the corresponding neuron number in this layer
N_HiddenLayers=length(N_NeuronsHiddenLayers);
TransferFunctions={'TANSIG' 'PURLIN'}; %Transfer Functions can be TANSIG
LOGSIG TANHYP PURLIN

[W, B]=UANNETInitialization;

Weights0 = UANNETGetWeights(W,B);

%-------------------------------------------------------------------------

%---------------------NETWORK TRAINING-----------------------------------
x0=Weights0;

cnnfval = [];
options =
optimset('LargeScale','off','MaxFunEvals',500000,'MaxIter',150,...
                    'Display','iter','TolFun',1.E-
12,'HessUpdate','bfgs',...
                    'PlotFcns',{@optimplotfval},...
                    'outputfcn',@outfun);
tic
[x,fval,exitflag,output] = fminunc(@UANNetFct, x0, options)
toc
```

```
%-------------------------------------------------------------------------

%-----------------------------PLOTS---------------------------------------

Weights=x;

%------------------------Simulation of Network----------------------------
Inputs = I_Train;
sNT = UANNetSim(Inputs, Weights);
sNTbr = sim(net,Inputs);
%---
Inputs = I_Simulate;
sNTsim = UANNetSim(Inputs, Weights);
sNTsimbr = sim(net,Inputs);
%-------------------------------------------------------------------------
%-------------------Unscaling the Data------------------------------------
I_Train = mapminmax('reverse',I_Train,is);
T_Train = mapminmax('reverse',T_Train,ts);
I_Simulate = mapminmax('reverse',I_Simulate,is);
T_Simulate = mapminmax('reverse',T_Simulate,ts);
sNT = mapminmax('reverse',sNT,ts);
sNTsim = mapminmax('reverse',sNTsim,ts);
sNTbr = mapminmax('reverse',sNTbr,ts);
sNTsimbr = mapminmax('reverse',sNTsimbr,ts);
% I_Train = mapstd('reverse',I_Train,is);
% T_Train = mapstd('reverse',T_Train,ts);
% I_Simulate = mapstd('reverse',I_Simulate,is);
% T_Simulate = mapstd('reverse',T_Simulate,ts);
% sNT = mapstd('reverse',sNT,ts);
% sNTsim = mapstd('reverse',sNTsim,ts);
%-------------------------------------------------------------------------
AllOutput=[sNT sNTsim];
AllOutputbr=[sNTbr sNTsimbr];
AllInput=[I_Train I_Simulate];
AllTarget=[T_Train T_Simulate];

figure
plot(AllTarget(1,:),'r.'); hold on; plot(AllOutput(1,:),'b'); hold on;
plot(AllOutputbr(1,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(1,:))
max(AllOutputbr(1,:))],'LineWidth',1,'Color','c');
line([0 DatWin],[-0.1 -0.1],'LineWidth',1,'Color','c');
legend('Target','CNN-II','NN Toolbox',2,'Location','Best');
set(legend,'FontSize',12);
xlabel('Data Number','FontSize',12);ylabel('X','FontSize',12);
set(gca,'FontSize',12)
figure
subplot(2,1,1)
plot(AllTarget(2,:),'r.'); hold on; plot(AllOutput(2,:),'b'); hold on;
plot(AllOutputbr(2,:),'kx');
line([TrWin TrWin],[min(AllOutputbr(2,:))
max(AllOutputbr(2,:))],'LineWidth',1,'Color','c');
xlabel('Data Number','FontSize',12);ylabel('Y','FontSize',12);
legend('Target','CNN-II','NN Toolbox',2,'Location','Best');
set(legend,'FontSize',12);
set(gca,'FontSize',12)
subplot(2,1,2)
plot(AllTarget(3,:),'r.'); hold on; plot(AllOutput(3,:),'b'); hold on;
plot(AllOutputbr(3,:),'kx');
```

```
line([TrWin TrWin],[min(AllOutputbr(3,:))
max(AllOutputbr(3,:))],'LineWidth',1,'Color','c');
xlabel('Data Number','FontSize',12);ylabel('Z','FontSize',12);
set(gca,'FontSize',12)

consviotr=-sNT(1,:)-0.1;
consviosim=-sNTsim(1,:)-0.1;
consvioTxtr=-sNTbr(1,:)-0.1;
consvioTxsim=-sNTsimbr(1,:)-0.1;

Norm_CNNI=norm(max(0,[consviotr consviosim]))
Norm_Tx=norm(max(0,[consvioTxtr consvioTxsim]))

Norm_consviotr=norm(max(0,consviotr))
Norm_consviosim=norm(max(0,consviosim))
Norm_consvioTxtr=norm(max(0,consvioTxtr))
Norm_consvioTxsim=norm(max(0,consvioTxsim))

figure
plotregression(AllTarget,AllOutput)
figure
plotregression(T_Train(1,:),sNT(1,:))
figure
plotregression(T_Train(2,:),sNT(2,:))
figure
plotregression(T_Train(3,:),sNT(3,:))
figure
plotregression(T_Simulate(1,:),sNTsim(1,:))
figure
plotregression(T_Simulate(2,:),sNTsim(2,:))
figure
plotregression(T_Simulate(3,:),sNTsim(3,:))

sseTxtr = sum(sum((T_Train-sNTbr).^2,2),1)
sum((T_Train-sNTbr).^2,2)
sseTxsim = sum(sum((T_Simulate-sNTsimbr).^2,2),1)
sum((T_Simulate-sNTsimbr).^2,2)
sseCNNtr = sum(sum((T_Train-sNT).^2,2),1)
sum((T_Train-sNT).^2,2)
sseCNNsim = sum(sum((T_Simulate-sNTsim).^2,2),1)
sum((T_Simulate-sNTsim).^2,2)
```

(ii) Objective Function Code

```
function obj = UANNetFct(Weights)

global TrWin I_Train T_Train
global is ts

Inputs = I_Train;
Outputs = UANNetSim(Inputs, Weights);

obj = sum(sum((T_Train-Outputs).^2,2),1);
%----------------------Penalty Section-------------------------------------
%--- Unscale to compute constraints
UScaleOutputs = mapminmax('reverse',Outputs,ts);
% UScaleOutputs = mapstd('reverse',Outputs,ts);
% %-END- Unscale to compute constraints
```

```
r=10; % Penalty Parameters for two outputs
g1 = -UScaleOutputs(1,:) - 0.1; % inequality constraints
% h = [cons]; % equality constraint

obj = obj + r * sum(max(0,g1),2); % Only ineq cons.
% obj = obj + r * (h*h')^0.5; % Eq. cons.
%obj = obj + r * ((h*h')^0.5 + sum(max(0,g1),2)); % ineq+eq cons.
%-------------------------------------------------------------------------

%-------------- Regularization ---------------------------------------------
RegParam=1.E-4;
N=length(Weights);
obj = obj + RegParam*sum(Weights.^2)/N;
%-END- Regularization
```

A.5. Source Codes of Common Functions Employed in CNN-II

(i) Simulation Function Code

```
function [Outputs] = UANNetSim(Inputs, Weights)

global N_HiddenLayers N_Inputs N_NeuronsHiddenLayers N_Outputs
TransferFunctions

[W, B] = UANNETSetWeights(Weights);

n=size(Inputs,2);
for i=1:N_HiddenLayers
B{i}=repmat(B{i},1,n);
end

OutputOfNeuron=Inputs;
for i=1:N_HiddenLayers
InputOfNeuron = W{i} * OutputOfNeuron + B{i};
if TransferFunctions{i}=='TANSIG'
OutputOfNeuron = 2./(1+exp(-2*InputOfNeuron))-1;
else if TransferFunctions{i}=='TANHYP'
OutputOfNeuron = sinh(InputOfNeuron)./cosh(InputOfNeuron);
else if TransferFunctions{i}=='LOGSIG'
OutputOfNeuron = 1./(1+exp(-InputOfNeuron));
else if TransferFunctions{i}=='PURLIN'
OutputOfNeuron = InputOfNeuron;
end
end
end
end
end

Outputs=OutputOfNeuron;
```

(ii) Function Code for Retreiving Weights and Biases

```
function Weights = UANNETGetWeights(W,B)
```

```
global N_HiddenLayers N_Inputs N_NeuronsHiddenLayers N_Outputs
TransferFunctions

Layer=[N_Inputs N_NeuronsHiddenLayers];
Weights=[];
for i=2:(N_HiddenLayers+1)
for j=1:Layer(i-1)
Weights = [Weights; W{i-1}(:,j)];
end
Weights = [Weights; B{i-1}(:,1)];
end
```

(iii) Function Code for Setting Weights and Biases

```
function [W, B]=UANNETSetWeights(Weights)

global N_HiddenLayers N_Inputs N_NeuronsHiddenLayers N_Outputs
TransferFunctions

Layer=[N_Inputs N_NeuronsHiddenLayers];
k=0;
for i=2:(N_HiddenLayers+1)
for j=1:Layer(i-1)
W{i-1}(:,j) = Weights(k+1:k+Layer(i),1);
k=k+Layer(i);
end
B{i-1}(:,1) = Weights(k+1:k+Layer(i),1);
k=k+Layer(i);
end
```

(iv) Initialization Function Code

```
function [W, B]=UANNETInitialization

global N_HiddenLayers N_Inputs N_NeuronsHiddenLayers N_Outputs
TransferFunctions

temp=[N_Inputs N_NeuronsHiddenLayers];
%%%%%----------Alternative 1-----------------------------------------
% c=1;
% for i=1:N_HiddenLayers
% W{i}=(rand(temp(i+1),temp(i))-0.5)*c;
% B{i}=(rand(temp(i+1),1)-0.5)*c;
% end
%%%%%---------------------------------------------------------------

%%%%%--------------Alternative 2-----------------------------------
% for i=1:N_HiddenLayers
% W{i}=rands(temp(i+1),temp(i));
% B{i}=rands(temp(i+1),1);
% end
%%%%%---------------------------------------------------------------

%%%%%--------------Alternative 3-----------------------------------
temp2=[-1 1]; N=[-2 2];
```

```
for j=1:(N_HiddenLayers-1)
pr=repmat(temp2,temp(j),1);
[W{j},B{j}]=calcnw(pr,temp(j+1),N);
end
W{N_HiddenLayers}=rands(temp(N_HiddenLayers+1),temp(N_HiddenLayers));
B{N_HiddenLayers}=rands(temp(N_HiddenLayers+1),1);

%=========================================================================
function [w,b]=calcnw(pr,s,n)
%CALCNW Calculates Nugyen-Widrow initial conditions.
%
%  PR
%  S - Number of neurons.
%  N - Active region of transfer function N = [Nmin Nmax].

% Special case: No inputs
r = size(pr,1);
if (r == 0) || (s == 0)
  w = zeros(s,r);
  b = zeros(s,1);
  return
end

% Fix nonfinite pr
i = find(~isfinite(pr(:,1)+pr(:,2)));
pr(i,:) = repmat([0 1],length(i),1);

% Remove constant inputs
R = r;
ind = find(pr(:,1) ~= pr(:,2));
r = length(ind);
pr = pr(ind,:);

% Special case: No variable inputs
if (r == 0)
  w = zeros(s,R);
  b = zeros(s,1);
  return
end

% Nguyen-Widrow Method
% Assume inputs and net inputs range in [-1 1].
% -------------------

wMag = 0.7*s^(1/r);
wDir = randnr(s,r);
w = wMag*wDir;

if (s==1)
  b = 0;
else
  b = wMag*linspace(-1,1,s)'.*sign(w(:,1));
end

% --------------------

% Conversion of net inputs of [-1 1] to [Nmin Nmax]
x = 0.5*(n(2)-n(1));
y = 0.5*(n(2)+n(1));
w = x*w;
```

```
b = x*b+y;

% Conversion of inputs of PR to [-1 1]
x = 2./(pr(:,2)-pr(:,1));
y = 1-pr(:,2).*x;
xp = x';
b = w*y+b;
w = w.*xp(ones(1,s),:);

% Replace constant inputs
ww = w;
w = zeros(s,R);
w(:,ind) = ww;

end
```

(Beale *et al.*, 2010)

```
%==========================================================
%%%%-------------------------------------------------------------------------

display('Initialized Weights and Biases')
j=length(N_NeuronsHiddenLayers);
for m=1:j
    W{m}
    B{m}
end
W
B
end
```

(v) Output Function Code

```
function stop = outfun(x,optimValues,state)

global cnnfval

stop = false;

     switch state
         case 'init'
             hold on
         case 'iter'
           cnnfval = [cnnfval; optimValues.fval];
         case 'done'
             hold off
         otherwise
     end
 end
```

REFERENCES

Akman, U., and N. Okay, 2003, "Index Tracking with Constraint Aggregation", *Applied Economic Letters,* Vol. 10, pp. 913-916.

Beale, M., M. Hagan and H. Demuth, 2010, *Neural Network Toolbox 7 User's Guide.*

Coleman, T., M. Branch and A. Grace, 2007, *Optimization Toolbox User's Guide.*

Ding, M., Y. Chen and S. L. Bressler, 2006, "Granger Causality: Basic Theory and Application to Neuroscience", *Handbook of Time Series Analysis: Recent Theoretical Developments and Applications*, Wiley-VCH Verlag GmbH and Co. KGaA, Weinheim, Germany.

Edgar, T. F., D. M. Himmelblau, and L. S. Lasdon, 2001, *Optimization of Chemical Processes*, McGraw Hill, New York, 2nd edition.

Fogler, H. S., 2008, *Elements of Chemical Reaction Engineering*, Prentice Hall, NJ, 4th Edition, pp. 526-532.

Fogler, H. S., 2008, *Elements of Chemical Reaction Engineering*, Prentice Hall, NJ, 4th Edition, pp. 567.

Khamas, R., 2008, *Isobaric Vapor-Liquid Equilibria of Enilosag Ta Smetsys Sevitidda 101.3 aPk*, M.S. Thesis, University of Technology.

Pavelka A. and A. Procházka, 2004, "Algorithms for Initialization of Neural Network Weights", *Sborník Príspevku 12th Rocníku Konference MATLAB 2004*, Vol. 2, pp. 453-459.

Samarasinghe, S., 2007, *Neural Networks for Applied Sciences and Engineering*, Taylor and Francis, New York, USA.

Sohan, S. and J. C. Príncipe, 2010, "A Test of Granger Non-causality Based on Nonparametric Conditional Independence", *ICPR '10 Proceedings of the 2010 20th International Conference on Pattern Recognition*, pp. 2620-2623.

Williams, T. J. and R. E. Otto, 1960, "A Generalized Chemical Processing Model for the Investigation of Computer Control", *American Institute of Electrical Engineers Transactions*, Vol. 79, pp. 458.