Utilizing Dual Neural Networks as a Tool for Training, Optimization, and Architecture Conversion

by

David Shawn Hunter

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 4, 2013

Keywords: Neural Network BMLP DNN Conversion

Approved by

Bogdan M. Wilamowski, Chair, Alumni Professor of Electrical and Computer Engineering & Director of AMSTC
Michael Baginski, Associate Professor of Electrical and Computer Engineering
Vitaly Vodyanoy, Professor of Physiology and Director of Biosensor Laboratory
Pradeep Lall, Thomas Walter Professor, Department of Mechanical Engineering & Director of CAVE^3

Abstract

Very little time has been devoted to the application of Dual Neural Networks and advances that they might produce by utilizing them for conversion between network architectures. By leveraging the efficiencies of the various networks, one can begin to draw some conclusions about the unleashed power of network conversion. If we could harness the advantages of multiple network architectures and somehow combine them into one network, we could make great advances in ANNs. By introducing the DNN as a tool for training, optimization, and architecture conversion, we find that this newly presented architecture is key to unlocking the strengths of other network architectures. Results in this study show that DNN networks have significantly higher overall success rates compared to BMLP and MLP networks. In fact, the DNN architecture had either the highest or the second highest success rate in all experiments. With the conversion methods presented in this study, not only do we now have a path for network conversion between BMLP, DNN, and MLP architectures, but also a means for training networks that were previously untrainable.

Acknowledgments

I would like to first of all thank my wonderful wife, Kimberly, for her patience and support throughout this entire process. I would also like to thank my parents who have given me such great support and encouragement. I dedicate this work to them and my wonderful children who have sacrificed time with “Dad” so that I could accomplish this work.

I would also like to express my deep appreciation to Dr. Bogdan Wilamowski for sharing his knowledge and passion for the field with me. He not only inspired me, but has guided me through this entire process. There is no doubt that I would not be where I am today without him.

Table of Contents

Abstract ii

Acknowledgments iii

List of Figures viii

List of Tables xii

List of Abbreviations xiv

Chapter 1 1

Chapter 2 5

2.1 ANN Training Algorithms 5

2.1.1 EBP Algorithm 6

2.1.2 Levenberg Marquardt Algorithm 6

2.1.3 Neuron by Neuron Algorithm 7

2.2 ANN Algorithm Optimization 8

2.2.1 EBP Optimizations 8

2.2.2 Levenberg Marquardt Optimizations 12

2.2.3 NBN Optimizations 15

2.3 ANN Topologies 16

2.3.1 MLP 16

2.3.2 BMLP 18
2.3.3 FCC 19
2.3.4 DNN 20
2.4 Benchmark Data Sets Used for Training and Comparison 23
2.4.1 Benchmark #1A – Simple 3-D Surface 23
2.4.2 Benchmark #1B – 3-D Surface 24
2.4.3 Benchmark #2 – Two Spiral Classification 25
2.4.4 Benchmark #3 – Parity-N Problems 27
2.4.5 Benchmark #4 – Checker-N Problems 28
2.4.6 Additional Benchmarks 29
2.5 Summary of Algorithms, Optimization, Topologies, and Benchmarks 34
Chapter 3 36
3.1 Digital Approach 36
3.2 Digital Implementation with ANNs 38
3.3 A Puzzle and a Discovery 40
Chapter 4 42
4.1 Architecture Conversion Overview 43
4.2 BMLP to DNN Conversion Process 44
4.2.1 BMLP to DNN Architecture Conversion 44
4.2.2 BMLP to DNN Weight Conversion 54

4.2.3 BMLP to DNN Conversion Examples 56
4.3 DNN to BMLP Conversion Process 61
4.3.1 DNN to BMLP Architecture Conversion 61
4.3.2 DNN to BMLP Weight Conversion 72
4.3.3 DNN to BMLP Conversion Examples 75
4.4 DNN to MLP Conversion Process 80
4.4.1 DNN to MLP Architecture Conversion 81
4.4.2 DNN to MLP Weight Conversion 83
4.4.3 DNN to MLP Conversion 91
4.5 MLP to DNN Conversion Process 95
4.5.1 MLP to DNN Architecture Conversion 95
4.5.2 MLP to DNN Weight Conversion 97
4.5.3 MLP to DNN Conversion 98
4.6 Conversion Summary 103
Chapter 5 105
5.1 Efficiency Comparison 105
5.2 Experimental Training Results 108
Chapter 6 121
Bibliography 124
Appendices 128

Appendix A 128

Appendix B 132

Appendix C 136

Appendix D 141

Appendix E 147

Appendix F 149

Appendix G 156

Appendix H 161

List of Figures

Figure 1.1: MLP Network for Parity-11 Problem 2

Figure 1.2: DNN Network for Parity-11 Problem 3

Figure 2.1: MLP network with 2-5-1 architecture 17

Figure 2.2: MLP network with 2-3-2-1 architecture 17

Figure 2.3: BMLP network with 2=3=2=1 architecture 18

Figure 2.4: FCC network with 2=1=1=1 architecture 20

Figure 2.5: The different parts and functions of the human brain 21

Figure 2.6: DNN network with 2-1(2)-1(1)-1 architecture 21

Figure 2.7: DNN network with 2-3(3)-2(1)-1 architecture 22

Figure 2.8: Simple 3-D Surface 24

Figure 2.9: 3-D Surface 25

Figure 2.10: Two Spiral Classification – (a) Graph; and (b) MATLAB code 26

Figure 2.11: Checker-3 Problem – (a) Graph; and (b) MATLAB code 29

Figure 3.1: XOR module with ANNs as digital units (3 neurons) 36

Figure 3.2: Parity-4 by combining XOR units (7 neurons) 37

Figure 3.3: Parity-8 by combining XOR units (15 neurons) 38

Figure 3.4: Parity-4 using MLP ANN architecture (5 neurons) 39

Figure 3.5: Parity-8 using MLP ANN architecture (9 neurons) 39

Figure 3.6: Parity-4 solution using only 4 neurons 40

Figure 3.7: Parity-8 solution using only 6 neurons 40

Figure 4.1: (a) 2=3=2=1 BMLP and (b) 2-1(1)-1 DNN Architectures 43

Figure 4.2: Conversion relationship between BMLP, DNN, and MLP 43

Figure 4.3: BMLP 2=1=1 to DNN 2-1(1)-1 45

Figure 4.4: BMLP 2=1=1=1 to DNN 46

Figure 4.5: BMLP X=2=3=2=1 network 47

Figure 4.6: DNN network equivalent to BMLP in Figure 4.5 48

Figure 4.7: BMLP X=2=3=2=1 to DNN X-2(6)-3(3)-2(1)-1 49

Figure 4.8: BMLP 2=3=2=1 Network 51

Figure 4.9: DNN network equivalent to BMLP in Figure 4.8 53

Figure 4.10: X=2=3=2=1 BMLP to X-2(6)-3(3)-2(1)-1DNN conversion with calculations 53

Figure 4.11: Weight Translation of networks in Figure 4.3 55

Figure 4.12: Weight Translation of networks in Figure 4.4 56

Figure 4.13: 2=3=2=2=1 BMLP network for the Simple 3-D Surface 57

Figure 4.14: DNN equivalent network for the network in Figure 4.13 58

Figure 4.15: 2=3=2=1 BMLP Network for 3-D Surface Benchmark 59

Figure 4.16: DNN equivalent network for the network in Figure 4.15 59

Figure 4.17: 11=2=1=1 BMLP Network for Parity-11 Benchmark 60

Figure 4.18: DNN equivalent network for the network in Figure 4.17 60

Figure 4.19: DNN 2-1(1)-1 to BMLP 2=1=1 62

Figure 4.20: DNN 2-1(2)-1(1)-1 to BMLP 2=1=1=1 63

Figure 4.21: 2-3(3)-2(1)-1 DNN conversion Step 1 65

Figure 4.22: 2-3(3)-2(1)-1 DNN conversion Steps 2 and 3 65

Figure 4.23: 2-3(3)-2(1)-1 DNN conversion Step 4 66

Figure 4.24: 2-3(3)-2(1)-1 DNN conversion Step 2 repeated..... 67

Figure 4.25: 2-3(3)-2(1)-1 DNN conversion Step 4 repeated..... 68

Figure 4.26: 2-3(3)-2(1)-1 DNN conversion Step 2 repeated..... 68

Figure 4.27: 2-3(3)-2(1)-1 DNN conversion Step 4 repeated..... 69

Figure 4.28: X-2(6)-3(3)-2(1)-1 DNN conversion Step 1 69

Figure 4.29: DNN conversion Steps 2 and 3 (a) and 4 (b) 70

Figure 4.30: DNN conversion Steps 2 and 3 (a) and 4 (b) repeated..... 71

Figure 4.31: DNN conversion Steps 2 and 3 (a) and 4 (b) repeated..... 72

Figure 4.32: Weight Conversion for Network in Figure 4.19..... 74

Figure 4.33: Weight Conversion for Network in Figure 4.20..... 74

Figure 4.34: 2-3(5)-2(3)-2(1)-1 DNN network for the Simple 3-D Surface..... 76

Figure 4.35: BMLP equivalent network for the network in Figure 4.34 77

Figure 4.36: 2-3(3)-2(1)-1 DNN network for the 3-D Surface 78

Figure 4.37: BMLP equivalent network for the network in Figure 4.36 78

Figure 4.38: 11-2(3)-2(1)-1 DNN network for Parity-11 79

Figure 4.39: BMLP equivalent network for the network in Figure 4.38 80

Figure 4.40: 2-1(1)-1 DNN to 2-2-1 MLP conversion 81

Figure 4.41: 2-1(2)-1(1)-1 DNN to 2-3-2-1 MLP conversion 82

Figure 4.42: Graph of bipolar activation function 84

Figure 4.43: 2-1(1)-1 DNN to 2-2-1 MLP weight conversion..... 85

Figure 4.44: 2-1(2)-1(1)-1 DNN to 2-3-2-1 MLP weight conversion 86

Figure 4.45: 2-3(5)-2(3)-2(1)-1 DNN network for Simple 3-D Surface Problem 87

Figure 4.46: 2-8-5-3-1 MLP network for the network in Figure 4.45 89

Figure 4.47: 2-3(5)-2(3)-2(1)-1 DNN network for Simple 3-D Surface Problem 92

Figure 4.48: 2-8-5-3-1 MLP equivalent network for network in Figure 4.47 93

Figure 4.49: DNN 2-3(3)-2(1)-1 network for 3-D Surface Benchmark 94

Figure 4.50: MLP equivalent network for network in Figure 4.49 94

Figure 4.51: 2-2-1 MLP to 2-1(1)-1 DNN conversion 96

Figure 4.52: 2-3-2-1 MLP to 2-1(2)-1(1)-1 DNN conversion 96

Figure 4.53: 2-2-1 MLP to 2-1(1)-1 DNN weight conversion 98

Figure 4.54: 2-3-2-1 MLP to 2-1(2)-1(1)-1 DNN weight conversion 98

Figure 4.55: 2-8-5-3-1 MLP network for Simple 3-D Surface Problem 99

Figure 4.56: 2-3(5)-2(3)-2(1)-1 DNN network for Simple 3-D Surface Problem 101

Figure 5.1: Efficiency comparison of various neural network architectures 106

Figure 5.3: Equivalent networks for 3-D Surface 112

Figure 5.4: Equivalent networks for Simple 3-D Surface 113

Figure 5.5: Equivalent networks for Parity-11 114

Figure 5.6: Equivalent networks for Checker-3 115

Figure 6.1: Success rates comparison for training the two-spiral patterns [10] 122

List of Tables

Table 2.1: Simple 3-D Surface Data Set 24

Table 2.2: Example Parity-N data sets – bipolar and unipolar 27

Table 4.1: Variables used for BMLP to DNN conversion 50

Table 4.2: Parameters for DNN Network equivalent to BMLP in Figure 4.3 52

Table 4.3: Parameters for the conversion in Figure 4.10 54

Table 4.4: Topology for the network in Figure 4.45 88

Table 4.5: Weights for the network in Figure 4.45 88

Table 4.6: Topology for the network in Figure 4.46 90

Table 4.7: Initial conversion weights for the network in Figure 4.46 90

Table 4.8: Final weights for the network in Figure 4.46 91

Table 4.9: Topology for the network in Figure 4.55 100

Table 4.10: Weights for the network in Figure 4.55 100

Table 4.11: Topology for the network in Figure 4.56 102

Table 4.12: Beginning weights for the network in Figure 4.56 102

Table 4.13: Final weights for the network in Figure 4.56 103

Table 5.1: Comparison of neural network efficiency with required weights 107

Table 5.2: Comparison of equivalent architectures on 3-D Surface 112

Table 5.3: Comparison of equivalent architectures on Simple 3-D Surface 113

Table 5.4: Comparison of equivalent architectures on Parity-11 Benchmark 114

Table 5.5: Comparison of equivalent architectures on Checker-3 Benchmark 116

Table 5.6: Minimal Architecture Comparison on 3-D Surface Benchmark 117

Table 5.7: Minimal Architecture Comparison on Simple 3-D Surface Benchmark..... 118

Table 5.8: Minimal Architecture Comparison on Parity-11 Benchmark..... 118

Table 5.9: Minimal Architecture Comparison on Checker-3 Benchmark..... 119

Table 5.10: Summary of Winners..... 120

List of Abbreviations

ACN	Arbitrarily Connected Network
ANN	Artificial Neural Network
BMLP	Bridged Multi-Layer Perceptron
DNN	Dual Neural Network
EBP	Error Back Propagation
FCC	Fully Connected Cascade
GNA	Gauss–Newton Algorithm
LM	Levenberg Marquardt
MLP	Multi-Layer Perceptron
NBLM	Neighborhood Based Levenberg Marquardt
NNT	Neural Network Trainer
RPROP	Resilient Propagation
SSE	Sum of Squares Error
XOR	Exclusive-OR

Chapter 1

Introduction

An enormous amount of research has been devoted to artificial neural network (ANN) research over the past several decades. Much research has focused on training algorithms such as Error Back Propagation (EBP) [1] [2], the Levenberg Marquardt (LM) algorithm [3] [4], and the Neuron by Neuron (NBN) algorithm [5] [6] [7]. Other research has seized opportunities to optimize training algorithms and network architecture. This research has produced improvements in algorithms such as momentum [8] and flat-spot elimination [9] for EBP. We have also seen the architecture progression from the multi-layer Perceptron (MLP) to the bridged multi-layer Perceptron (BMLP) to the fully connected cascade (FCC) architecture [10] and finally to dual neural networks (DNN) [11]. All of this research has produced many advances in the methods used to train ANNs as well as improvements in architecture design which have significantly improved efficiency [11] [12] [13]. However, little time has been devoted to conversion between network architectures and any advances that this might produce.

By looking at the efficiencies of the various networks, one can begin to draw some conclusions about network conversion. Research has shown that for the Parity-N problem an MLP network with one hidden layer requires N+1 neurons [14]. So, for the Parity-11 problem, twelve neurons would be required in an 11-11-1 MLP architecture. The number of neurons required for this same problem can be reduced by one if the MLP

network has two hidden layers in an 11-5-5-1 MLP architecture. Figure 1.1 shows the network for the Parity-11 problem using the 11-5-5-1 MLP architecture. With further testing, we find that this is the smallest MLP network that can be trained for the Parity-11 problem.

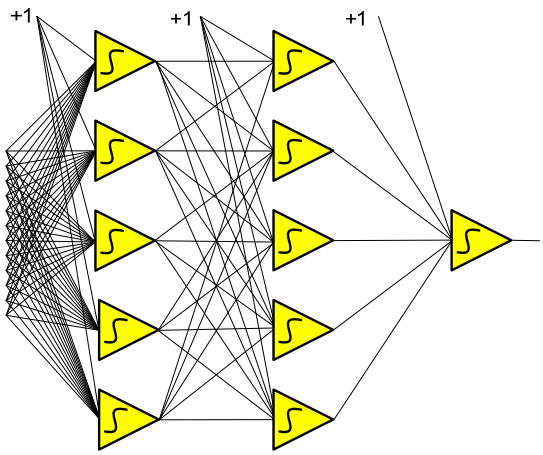


Figure 1.1: MLP Network for Parity-11 Problem

If we move away from the mainstream architectures that utilize only sigmoidal activation functions and look at a DNN architecture which utilizes neurons with both sigmoidal and linear activation functions, we discover something quite surprising. We find that if we use two linear neurons, one in each hidden layer, in a traditional MLP type architecture, that we can reduce the number of required neurons for the Parity-11 problem from 11 to 6. This DNN network can be seen in Figure 1.2 and has an 11-1(1)-2(1)-1 architecture where the number in parentheses represents the number of linear neurons in that hidden layer. As seen in this network, an analytical solution was found for the Parity-11 problem. By applying the input patterns for this network, one can see that it yields the correct solution for each input pattern.

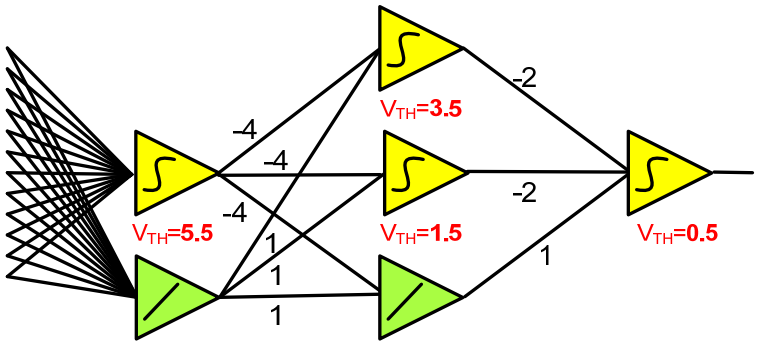


Figure 1.2: DNN Network for Parity-11 Problem

Analytically, the ANN seen in Figure 1.2 is not capable of solving the Parity-11 problem. So, what made this possible? As I studied the networks in Figures 1.1 and 1.2, I could only conclude that what made the network in Figure 1.2 so much more powerful was the addition of linear neurons. Simply changing the activation function of two neurons from sigmoidal to linear transformed a traditional MLP network capable of solving Parity-6 into a DNN capable of solving Parity-11. These results led me to ask several important questions:

1) Can this type of result be duplicated with other network architectures?
2) Is there a way to directly convert between DNN and other architectures?
3) Can the use of DNNs assist with the training of other network architectures?

Pondering these questions let me to consider that further investigation into DNNs was warranted. Therefore, I decided to focus my research on answering these questions. With this end in mind, the future chapters will aid one in understanding how we can utilize DNNs as a tool for training, optimization, and architecture conversion.

Chapter 2 will discuss the current state of ANN research with regards to training, optimization, architectures, and the many benchmarks used to test and analyze ANNs. Chapter 3 will explain the motivation for my research of DNNs. Chapter 4 will cover

conversion methods between architectures. Chapter 5 will compare the results of the various network architectures. Finally, Chapter 6 will provide a summary and conclusion of the research.

Chapter 2

Overview of ANN Research

ANNs are currently used in many different applications. We likely see them every day, but don't realize that they are there. When you watch the weather forecast on the news or view it on the Internet or you mobile phone, few of us realize that ANNs were likely used to generate the forecast [15] [16] [17]. In industry, ANNs are used to control induction [18] [19] [20], permanent magnet [21] [22], and stepper motors [23]. Additionally, they are used in robotics [24], motion control [25], battery control [26], job scheduling [27], and networking [28]. Some more advanced ANNs are used in highly complex, dynamic systems such as oil wells [29] [30]. Although their uses vary, you can see that ANNs are found almost everywhere.

As mentioned in the introduction, much research has been performed on ANNs over the past several decades. So, what is the current state of ANN research with regards to training, optimization, and architectures? Each of these areas will be addressed separately in this chapter.

2.1 ANN Training Algorithms

The three main training algorithms that exist today are the Error Back Propagation (EBP) algorithm, the Levenberg Marquardt (LM) algorithm, and the Neuron by Neuron (NBN) algorithm. Each of these algorithms has benefits and drawbacks and they have all seen modifications and improvements over the years. A discussion of each algorithm

with its benefits and drawbacks will be discussed in this section while the optimizations to each algorithm will be discussed in the next section of this chapter.

2.1.1 EBP Algorithm

The EBP algorithm is probably the most widely used and popular algorithm for ANN training. EBP is a supervised learning method based upon a generalization of the delta rule. This algorithm was developed in 1969 by Arthur E. Bryson and Yu-Chi Ho [31], but was not applied in the context of neural networks until 1974 [1] [2]. Its "re-discovery" in 1974 caused a so-called "renaissance" in the field of ANN research. The benefit of this algorithm is that it has the ability to reduce learning errors to very small values. However, this algorithm has many drawbacks [10]. Generally speaking, EBP is a very inefficient algorithm that is computationally intensive. Although it can be very successful at training an ANN to a give training set, there is no guarantee that the resulting network has good generalization abilities. Another drawback of this algorithm is that its solution search process only follows the gradient, leaving it vulnerable to being trapped in local minima.

2.1.2 Levenberg Marquardt Algorithm

The Levenberg Marquardt algorithm is a second order algorithm that interpolates between the Gauss–Newton (GNA) and EBP algorithms. The LM algorithm applies a Jacobian matrix to evaluate the change of the gradient. This algorithm has two main benefits. First, it typically provides much better results than the EBP algorithm. And second, it is very fast and more efficient than the EBP algorithm. However, the LM algorithm also has several drawbacks. First, computation of the Jacobian matrix can become an impediment if the number of input patterns exceeds a few hundred [4].

Second, this algorithm only finds a local minimum, not a global minimum. So, in situations where there are multiple minima, the algorithm will only find a solution if the initial guess is close to the final solution. Third, the algorithm functions in such a way that it only works on MLP network architectures. It does not have the ability to function on arbitrarily connected networks. Finally, for very large network architectures which have a large number of neurons and weights, the computational load is very heavy and can be taxing to even the most current computers.

In spite of its drawbacks, the LM algorithm was a major step forward in ANN training. It not only provided more efficient results in faster times, it also gave researchers an additional training algorithm option when the widely used EBP algorithm failed to converge to a solution.

2.1.3 Neuron by Neuron Algorithm

One of the biggest drawbacks of both the EBP and LM algorithms is that both only work on MLP architectures. Even the very popular MATLAB Neural Network Toolbox's [32] first and second order training algorithms only work on MLP networks. This was a very limiting factor which gave researchers no other choice but to use MLP architectures and any results obtained were usually less than satisfactory [33]. Both the architectural and size limitations were solved by the development of the Neuron by Neuron (NBN) algorithm [5] [6] [7]. A fully functioning software package that implements the NBN algorithm along with EBP, LM, and other variations is available online as the Neural Network Trainer [34] [35]. This particular algorithm has two main benefits. First, it can work on all of the previously discussed network architectures: MLP, BMLP, FCC, and DNN. Second, it supports several types of neuron activation functions

such as unipolar, bipolar, and linear. The NNT software allows a user to define new types of activation functions. The only know drawback of this algorithm is that it has some of the same vulnerabilities as the LM algorithm in that it can get stuck in local minima and like other algorithms, is not guaranteed to produce a solution.

2.2 ANN Algorithm Optimization

The previous section discussed the development of the three main training algorithms used today in ANN research. As with most things, engineers are always looking for ways to improve things. This is true of the EBN, LM, and NBN algorithms. This section will discuss the optimizations developed over the years for the mentioned algorithms as well as architectural optimizations.

2.2.1 EBP Optimizations

The EBP algorithm is the oldest of the algorithms that we have discussed, so it will probably come as no surprise that this algorithm probably has the most optimizations, or enhancements. Over the many years that EBP has existed, many improvements have been developed to solve some of the algorithm's inherent problems. We will discuss some of the more well known optimizations.

2.2.1.1 EBP with Momentum

In this optimization to EBP, the momentum term is added to the weight update rule to speed up the process of learning. The momentum term is added to the weight update equation to prevent the system from converging to a saddle point or local minimum. The momentum, typically given the Greek letter alpha, is a value between 0 and 1. The momentum term, which is the product of alpha and the change in weight that occurred in the previous weight update, is added to the current weight update. By adding

this momentum term, weight changes can be kept on a faster and more even path to a solution [36].

Addition of this momentum term to the weight update equation doesn't come without risks. One must be careful when setting this parameter. While a high alpha can help increase the speed of convergence, it can also cause instability in the system with a risk of overshooting the minimum. At the other end of the spectrum, a low alpha is not guaranteed to ovoid local minima and it will slow the training process. While the addition of momentum improves EBP, the selection of alpha is more of an art than a science.

2.2.1.2 EBP with Stochastic Learning Rate

As a deterministic decent gradient algorithm, EBP has a limitation on speed and convergence. These limitations are mainly due to large plateaus on the surface of the error function as well as the potential presence of several local minima on that same surface. As the size of the network and number of weights increase, these problems are magnified. To overcome these problems, researchers have suggested adding a stochastic, or random, process to the learning algorithm [37].

In 1988, Kolen suggested restarting the entire learning process with random weights every time the algorithm failed to converge at a solution [38]. Starting over every time the algorithm fails to converge is very impractical and could be very inefficient. Instead, other researchers suggested that the stochastic element be added in real-time in parallel with the deterministic weight changes rather than at the end of a training cycle. The idea behind this procedure is to somehow avoid or get out of local minima and achieve better solutions [39] [40]. It was shown that any of the stochastic modifications improved either the convergence quality or speed.

2.2.1.3 EBP with Flat-Spot Elimination

If an ANN has high neuron gain or has neuron states that are well defined and far from the thresholds, back-propagation convergence is typically very slow. In both of these cases, the gradient calculated for the back-propagation algorithm are very small making it difficult for errors to propagate back through the network and effect meaningful change in weights. Under these circumstances, "flat-spots" are encountered and the output of the network can be entirely wrong while producing a small SSE. At least three different methods were employed by different researchers to eliminate these "flat-spots" and increase the speed of convergence. One method added an offset to the activation function [41]. Another used a scaled linear approximation of the sigmoidal function for the error calculation [42]. The last method uses a logic OR in the calculation. Depending on whether the error is large or small, one of two effective gradient calculations is used [9]. All three modifications yielded improvement in convergence.

2.2.1.4 RPROP

RPROP, or Resilient PROPagation, was developed to deal with the inherent issues introduced with EBP with momentum. As you will remember, the momentum parameter is chosen by the individual training the network. If this parameter is too small or too large, it will cause convergence issues. The selection of this parameter value became more of an art than a science and was different for every network.

To counter the problems just listed, RPROP introduces an individual weight update value which solely determines the change in that weight. This adaptive update value for each weight changes during the learning process based upon its local sight on the error function. In general terms, the weight update value is adjusted as outlined here.

If the partial derivative of the weight changes signs, then the previous update was too large and jumped over a local minimum. In this case, the weight update value is decreased. On the other hand, if the partial derivative of the weight does not change signs, then the previous update was too small and the weight update value is slightly increased.

Once the weight update values are all determined, the actual weight update follows this guideline. If the derivative is positive, meaning increasing error, then the weight is decreased by the weight update value. If the derivative is negative, meaning decreasing error, then the weight is summed with the update value. The only exception to this update is when the partial derivative changes signs, meaning that a local minimum was jumped. In this instance, the previous weight change is reversed and the process continues [43].

The success of this algorithm stems from the fact that there is individual and direct weight adaptation during each training cycle. Essentially, it eliminates the error caused by the "what's good for one is good for all" mentality.

2.2.1.5 QUICKPROP

The QUICKPROP algorithm [44] is similar to the RPROP algorithm discussed in the last section. In developing this algorithm, Fahlman makes two risky assumptions: 1) He assumes that the error versus weight curve is an upward pointing parabola; and 2) The changes in each individual weight do not affect the other weights. The goal of the algorithm is to move downward in the parabola until the minimum is reached. To do this the error derivative calculated during the previous iteration are stored and compared to the current error derivative. For each weight, we use the previous and current error slopes

coupled with the weight-change between the points at which these slopes were measured. This information allows us to determine a parabola for that weight. With two known points on the parabola, we are now able to jump directly to the minimum point of this parabola.

In practice, the algorithm deals with three different cases: 1) If the current slope is somewhat smaller and in the same direction as the previous slope, then the weight is moving in the right direction. How much the weight is changed depends on the ratio of slope change to weight change in the last iteration; 2) If the current slope is the opposite direction (sign changes) of the previous slope, it means that the previous weight change caused the weight to jump over the minimum. The weight is now on the opposite side of the parabola. In this case, the weight is changed in the opposite direction in an amount that puts it somewhere in between the previous two weights; and 3) In the final case, the current slope is either the same or larger than the previous slope. In this case, a parameter called the "maximum growth factor" is implemented to prevent a large step in the wrong direction. This factor is multiplied by the previous step size and the weight is moved by that amount.

While QUICKPROP is a significant optimization over standard EBP, it still can suffer from the flat-spot problem. Thus we see that no algorithm is perfect.

2.2.2 Levenberg Marquardt Optimizations

Like the EBP algorithm, the LM algorithm has also seen a number of optimizations, or enhancements. Over the many years that the LM algorithm has been used in ANN training, many improvements have been developed to solve some of the algorithm's problems. We will discuss some of the more well known optimizations.

2.2.2.1 Neighborhood Based Levenberg Marquardt

The LM algorithm has been extensively applied as a neural-network training method. In addition to only operating on MLP networks, it requires a very large overhead in memory and number of operations when the network to be trained has a large number of adaptive weights. The Neighborhood Based Levenberg Marquardt (NBLM) algorithm was inspired by the way that biological brains seem to act; letting different groups of neurons specialize in different tasks [45].

This algorithm optimization works by dividing the ANN into different groups or neighborhoods and considers each group an independent learning unit. The algorithm then allows weight adaptation to take place in each neighborhood independent of other neighborhoods. The process for the algorithm has three steps:

1) Define the network structure and initial weights, and then assign neighborhoods.
2) Select a neighborhood to be trained and then train it with the LM algorithm.
3) Evaluate SSE to see if the training error was reached. If not, repeat steps 2 and 3.

The results of the NBLM algorithm showed a significant reduction in memory and time requirements for computations. This was most noticeable in very large networks. The gains in training times were found to considerably depend upon the neighborhood size and selection; however, no general guidelines for selecting neighborhoods have been outlined by the developers [46].

2.2.2.2 Modified NBLM

Several years following the development of the NBLM algorithm, its original developer worked with other researchers to make additional improvements to the algorithm. The researchers made one simple modification. They implemented a locally

adaptive learning coefficient for the LM algorithm in each defined neighborhood. The results of testing this modified algorithm showed statistically significant improvements in learning times over the original NBLM algorithm [47].

2.2.2.3 Improved Computation for LM

As previously discussed, one of the major limitations of the LM algorithm is how taxing the computational requirements are. In this modification to the LM algorithm, the researchers developed a method to compute a Quasi-Hessian matrix and gradient vector directly without utilizing Jacobian matrix multiplication and storage. This new method solves the memory limitation problem for LM training [6].

To accomplish this exploit, only elements in the upper/lower triangular array need to be calculated. Therefore, the Quasi-Hessian matrix, $\boldsymbol{Q}$, can be calculated as an approximation of Hessian matrix:

$$\boldsymbol{H} \approx \boldsymbol{Q} = \boldsymbol{J}^T\boldsymbol{J} \quad \textbf{(1)}$$

Similarly, the gradient vector can be calculated as follows:

$$\mathbf{g} = \mathbf{J}^T\mathbf{e} \quad \textbf{(2)}$$

Huge computational savings are achieved as a result of the Quasi-Hessian matrix, $\boldsymbol{Q}$, and gradient vector, $\boldsymbol{g}$, being calculated directly without the necessity of calculating and storing Jacobian matrix, $\boldsymbol{J}$ [6].

2.2.2.4 LM for Arbitrarily Connected Neural Networks

Another major limitation of the LM algorithm is that it only worked for MLP networks. As mentioned previously, a C++ implementation of the NNT [35] [48] was developed to assist researchers in ANN training research. This tool implements two different versions of the LM algorithm: 1) Traditional forward-backward computation;

and 2) A newly developed forward-only computation. Additionally, these new implementations can not only handle MLP networks, but also arbitrarily connected networks (ACN).

This is a momentous step forward for the LM algorithm as researchers are no longer tied to using MLP networks when utilizing the LM algorithm for training and testing purposes.

2.2.3 NBN Optimizations

The NBN algorithm is relatively young in comparison to the EBP and LM algorithms. In spite of its youth, this algorithm has also seen a few improvements and optimizations. In addition to its traditional forward-backward computation algorithm, two additional optimizations have been developed.

2.2.3.1 NBN Forward Only

The designers of the NBN algorithm [34] [49]and NNT implemented a modification to the standard NBN algorithm that improves its speed. This modification utilizes a forward-only computation capable of handling ACNs. Due to the reduced computational requirements, the NBN forward only algorithm is significantly faster than the standard NBN algorithm. This increased speed is even more noticeable on ANNs with multiple output neurons.

2.2.3.2 NBN Improved Algorithm

The most recent optimization to the NBN algorithm is based upon the NBN forward only algorithm just described. This optimization focuses on improving computational speed. To accomplish this goal, the algorithm is designed to only invert the Hessian matrix one time per iteration, reducing computational overhead.

Theoretically and in experiments, this algorithm is able to compute faster than the LM and NBN algorithms which need to invert the Hessian matrix several times per iteration. Additionally, minor modifications were made to improve convergence [34].

2.3 ANN Topologies

There are four different ANN architectures that are commonly used today. They are MLP, BMLP, FCC, and DNN. Each of these architectures has advantages and disadvantages. While the architectures themselves have not changed over time, a significant amount of research has been done to help determine optimal network size. We will discuss each architecture separately.

2.3.1 MLP

A MLP network is a feedforward ANN that maps a set of input patterns to a set of output patterns. It is made up of a given number of hidden layers and an output layer. All inputs are connected to all neurons in the first hidden layer and all neurons outputs in the hidden layers are connected to the inputs of neurons in the next layer. Figure 2.1 shows a MLP network with one hidden layer with a 2-5-1 architecture. Figure 2.2 shows a MLP network with two hidden layers with a 2-3-2-1 architecture.

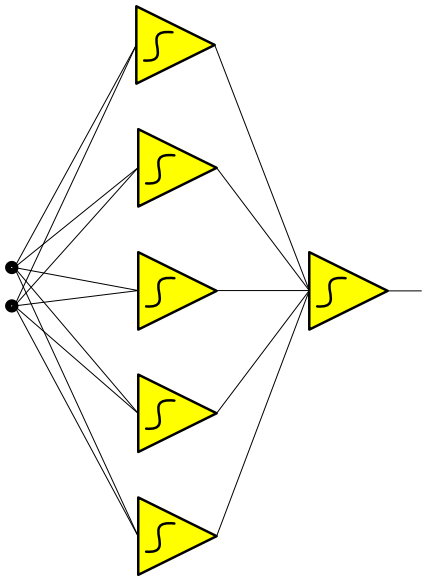


Figure 2.1: MLP network with 2-5-1 architecture

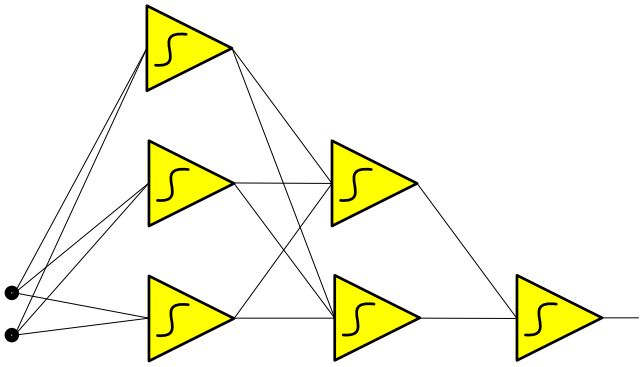


Figure 2.2: MLP network with 2-3-2-1 architecture

As will all networks, the first question one usually asks is, “What is the power of this network?” Or, in other words, how big of a problem is a given network able to solve. For purposes of this comparison, we will use the Parity-N problem as a benchmark for comparing the networks. For a MLP network with one hidden layer, like that seen in Figure 2.1, the total number of neurons, J, required to solve the Parity-N problem is:

$$\mathbf{J = N + 1} \qquad \mathbf{(3)}$$

Although only two inputs are shown, we can conclude that the network in Figure 2.1 is capable of solving the Parity-5 problem [11] [10]. Essentially, the number of neurons in the first and only hidden layer is equal to the parity number, N.

For a MLP network with multiple hidden layers, the largest Parity-N problem that a network can solve is defined by:

$$\mathbf{N = (h_1 + 1) + (h_2 + 1) + \ldots + (h_n + 1) - 1} \qquad \mathbf{(4)}$$

where h_n represents the number of neurons in the n^{th} hidden layer and it is assumed that there is a single output neuron. Based upon this, we can conclude that the MLP network in Figure 2.2 is capable of solving Parity-6.

2.3.2 BMLP

A BMLP network is also a feedforward ANN that maps a set of input patterns to a set of output patterns. It is made up of a given number of hidden layers and an output layer. All inputs are connected to all neurons in the entire network. Additionally, all outputs from neurons in the hidden layers are fully connected to all neurons in forward layers, including the output layer. Figure 2.3 shows a BMLP network with two hidden layers and a 2=3=2=1 architecture.

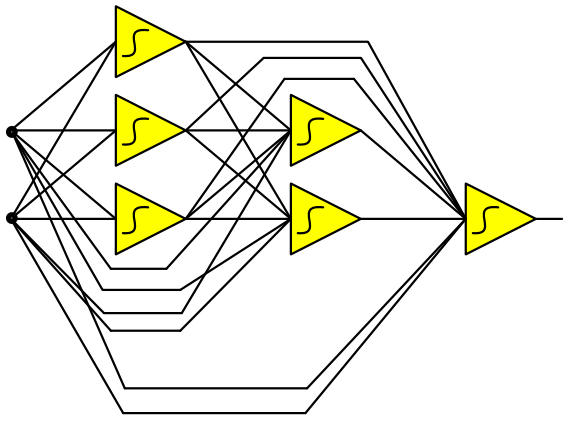


Figure 2.3: BMLP network with 2=3=2=1 architecture

For a BMLP network with multiple hidden layers, the largest Parity-N problem that a network can solve is defined by [10] [14]:

$$\mathbf{N = 2(h_1 + 1)(h_2 + 1) \ldots (h_n + 1) - 1} \quad \mathbf{(5)}$$

where h_n represents the number of neurons in the n^{th} hidden layer and it is assumed that there is a single output neuron. Based upon this, we can conclude that the BMLP network in Figure 2.3 is capable of solving Parity-23. As can be seen by comparing the networks in Figures 2.2 and 2.3, the only difference is that bridged connections are added to the network in Figure 2.3. These bridged network connections significantly increase the power of this network by nearly a factor of 4.

2.3.3 FCC

The FCC network is actually a special case of the BMLP network with only a single neuron in each hidden layer. Once again, all connections are fully connected to all neurons. Figure 2.4 shows a FCC network with two hidden layers and a 2=1=1=1 architecture. Research has shown that the largest Parity-N problem that a FCC network can solve is defined by:

$$\mathbf{N = 2^n - 1} \quad \mathbf{(6)}$$

where n is the total number of neurons in the network [10] [11] [14]. Based upon this, we can conclude that the FCC network in Figure 2.4 is capable of solving Parity-7. Notice that this answer also agrees with equation (5) above. This is due to the fact that FCC is a special case of BMLP and therefore the calculation in equation (5) also applies to FCC. However, the calculation in equation (6) is simplified and easier to use.

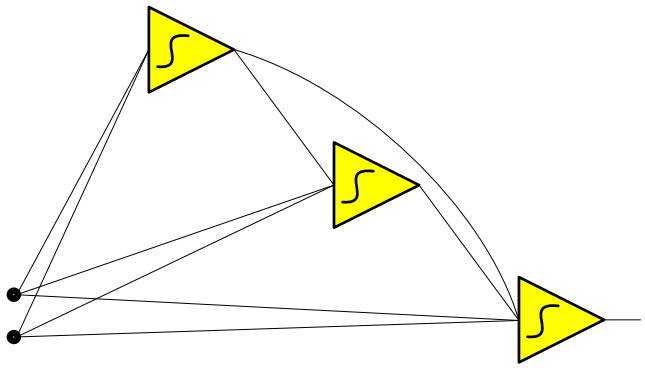


Figure 2.4: FCC network with 2=1=1=1 architecture

2.3.4 DNN

A majority of research to date looks at neural networks as if they only contain a single type of neuron. What if we looked at ANNs differently and allowed them to have different types of neurons, each with a different function. The use of 2 different types of neurons in the same architecture falls under a class of architectures called Dual Neural Networks (DNN). Many researchers have looked to DNNs to solve specific problems [50] [51]. Figure 2.5 shows a diagram of the human brain, outlining the different sections of the brain and the functions or specialization that occurs in that section of the brain. If we apply this methodology on a small scale to DNNs by using only two different types of neurons, we can construct a very powerful and versatile network structure. A majority of DNNs to date in the literature are very specialized, using a myriad of different components such as fuzzy devices, summers, logic blocks, and countless others.

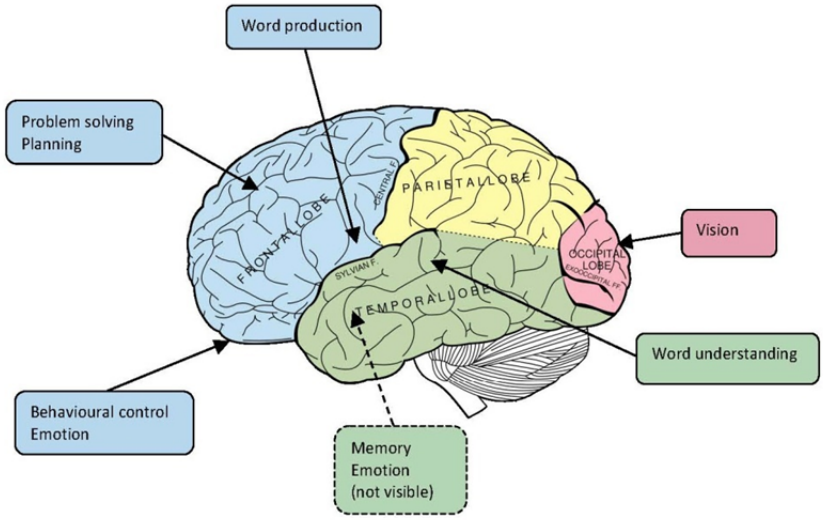


Figure 2.5: The different parts and functions of the human brain

This research will focus on a simple DNN design that utilizes standard linear and non-linear neurons. Figure 2.6 shows a simple DNN network with a 2-1(2)-1(1)-1 architecture. Note that the numbers in parentheses represent the number of linear neurons in that hidden layer.

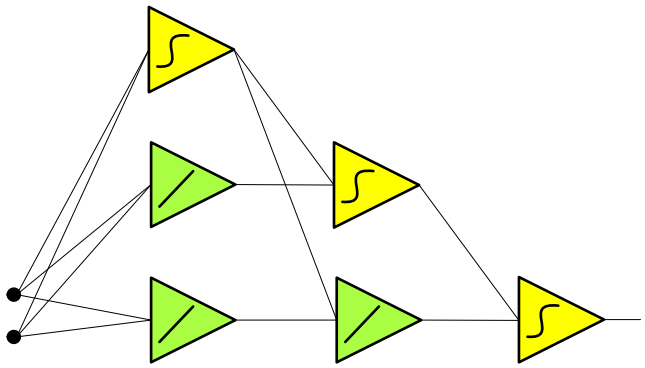


Figure 2.6: DNN network with 2-1(2)-1(1)-1 architecture

Extensive research has not been performed by outside researchers on this specific architecture, however, based upon its architecture; we can draw some conclusions regarding the power of the network. One may notice that the network in Figure 2.6 looks

very similar to the FCC network in Figure 2.4. Likewise, if you look at the DNN network

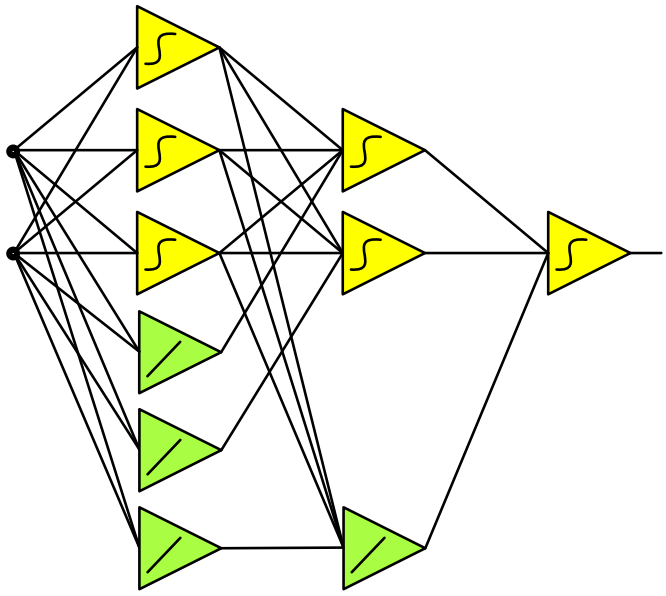


Figure 2.7: DNN network with 2-3(3)-2(1)-1 architecture

in Figure 2.7 that has a 2-3(3)-2(1)-1 architecture, you will notice that it is also very similar to the BMLP network in Figure 2.3. In both cases, we see that the bridged connections are removed from the BMLP and FCC networks and they are replaced with linear neurons and non-bridged connections.

For purposes of determining the power of these networks for the Parity-N problem, we will make the assumption that we can treat the networks in Figures 2.6 and 2.7 like BMLP networks and use equation (5) to determine their power. In doing this, we will only consider the non-linear neurons in each hidden layer for purposes of calculations. By doing this, we find that the DNN network in Figure 2.6 is capable of solving Parity-7 and the network in Figure 2.7 is capable of solving Parity-23. This can be verified by training both networks with the specified parity data set. What is not know is if these DNNs possess even greater power than the BMLP equation shows. However,

we do know that the BMLP equation does give us a minimum power value for DNN networks [11].

2.4 Benchmark Data Sets Used for Training and Comparison

When performing research with Neural Networks, one must have pre-defined sets of data to use for training, testing, and comparison. These benchmark data sets must provide a vigorous and robust means for validating and comparing Neural Networks. This section will discuss many of the benchmarks used in this and other research. In order to compare results between different Neural Network architectures, there must be benchmark data sets used for comparison and validation. The general method is to select a benchmark data set and train a given network architecture to that set. Then, perform the network conversion. Following the conversion, validate the newly created network with the same data set. If the network conversion is correct, the results for all inputs will be identical on both networks. The conversions described later in this work will use one of four baseline data sets: 1) Simple 3-D Surface; 2) 3-D Surface; 3) Two Spiral Classification; 4) Parity-N problems; and 5) Checker-N problem.

2.4.1 Benchmark #1A – Simple 3-D Surface

The Surface seen in Figure 2.8 results from a set of 25 ordered pairs obtained from the function:

$$z = \gamma e^{-[\alpha(x-x_0)^2+\beta(y-y_0)^2]} \quad \textbf{(7)}$$

where $x,y \in (0, 1, 2, 3, 4)$, x_0=4, y_0=3,and γ=4. This particular data set yields a rough 3-D surface.

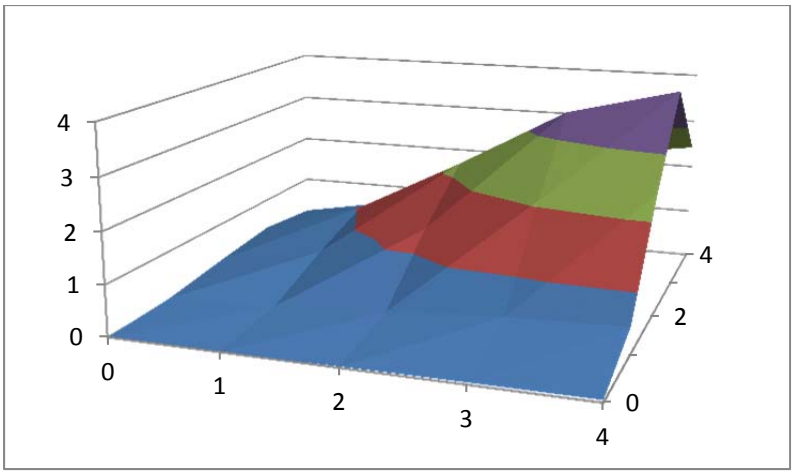


Figure 2.8: Simple 3-D Surface

Y Axis Value \ X Axis Value	0	1	2	3	4
0	0.00403	0.01152	0.02439	0.03825	0.044436
1	0.04911	0.14034	0.29709	0.46594	0.54134
2	0.22009	0.62895	1.3315	2.0882	2.4261
3	0.36287	1.037	2.1952	3.4428	4
4	0.22009	0.62895	1.3315	2.0882	2.4261

Table 2.1: Simple 3-D Surface Data Set

2.4.2 Benchmark #1B – 3-D Surface

The Surface seen in Figure 2.9 results from a set of 1600 ordered pairs obtained from function 7. However, in this implementation, $x,y \in (0, 0.2564, 0.5128, \ldots, 9.7436, 10)$, $x_0=9$, $y_0=5$,and $\gamma=1$. As this surface is created from a more extensive data set, it provides a much more rigorous challenge for training. The data set consists of 1600 ordered pairs with unique outputs derived from a mathematical function.

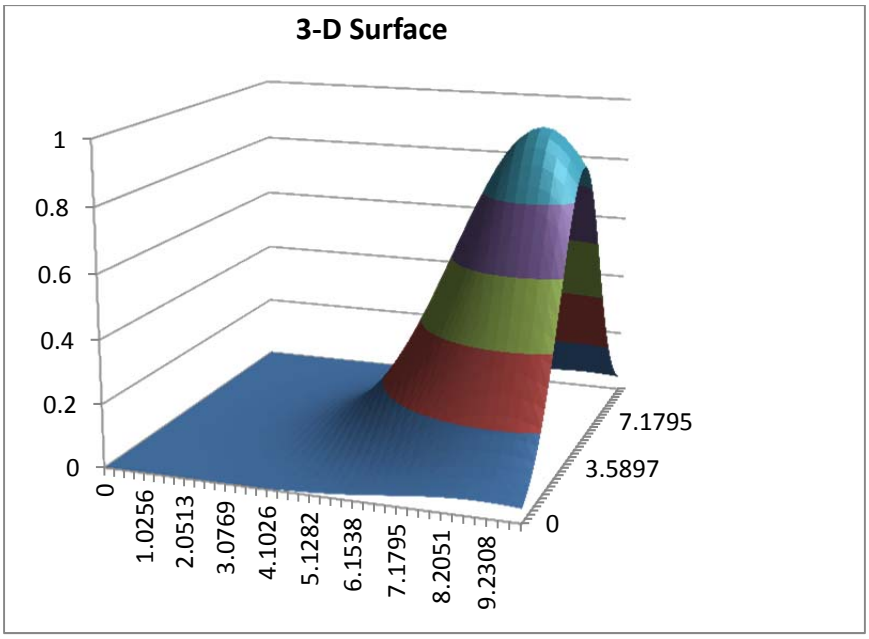
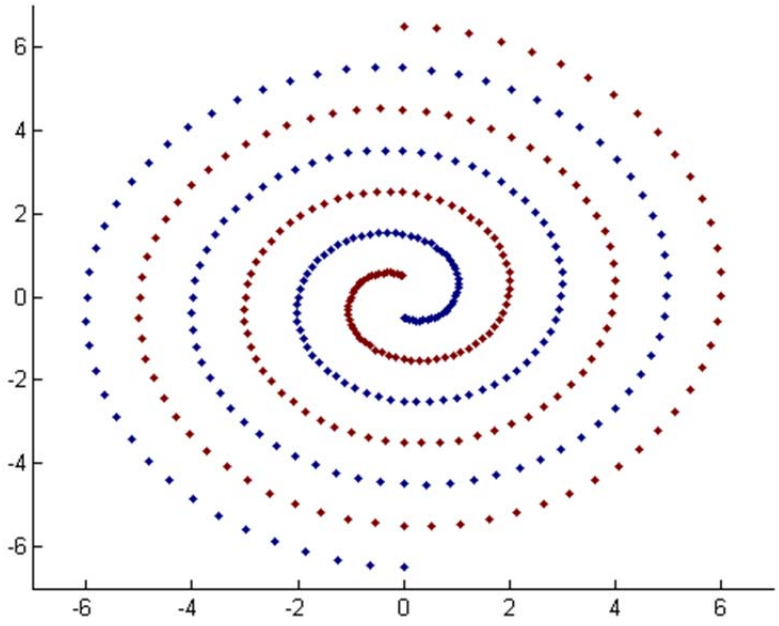


Figure 2.9: 3-D Surface

2.4.3 Benchmark #2 – Two Spiral Classification

The Two Spiral Classification seen in Figure 2.10a results from a set of data consisting of two input values (X and Y) and an output (Z) which is either a +1 or -1 classification. This set of data creates two spirals that are intertwined. The data set consists of 382 ordered pairs with either a +1 or -1 which was generated from the MATLAB code seen in Figure 2.10b. Values for Z are classifications of either -1 (blue) or +1 (red).



(a)

```
clear all; format compact; format short;
m=2; %multiplier for number of patterns if m=1 then np=194
n=m*96;
j=0;
for i = 0:n
   angle = i*3.1415926/(m*16.0);
   radius = 6.5*(104*m-i)/(104*m);
   x = radius*sin(angle);
   y = radius*cos(angle);
   j=j+1;
   a(j,:)=[x,y,1]
   j=j+1;
   a(j,:)=[-x,-y,-1]
end
figure(1); clf;
scatter(a(:,1),a(:,2),7,a(:,3),'filled');
axis([-7 7 -7 7]);
whos
```

(b)

Figure 2.10: Two Spiral Classification – (a) Graph; and (b) MATLAB code

2.4.4 Benchmark #3 – Parity-N Problems

Parity-N Problems consists of a set of N inputs and a single output. All inputs and the output are either unipolar (0 or 1) or bipolar (-1 or +1). The goal of the parity problem is to calculate the parity bit for any given set of inputs. The resulting output will be +1 if the number of +1 inputs is odd and will be 0 or -1 if the number of +1 inputs is even. There is no way to visualize the parity problem, but example data sets for Parity-3 and Parity-5 can be found in Table 2.2. The parity calculation is used frequently in today's computers and has become one of the standards for Neural Network training.

Parity-3 Problem (bipolar)			
Input 1	Input 2	Input 3	Parity Bit
-1	-1	-1	-1
-1	-1	1	1
-1	1	-1	1
-1	1	1	-1
1	-1	-1	1
1	-1	1	-1
1	1	-1	-1
1	1	1	1

Parity-4 Problem (unipolar)				
Input 1	Input 2	Input 3	Input 4	Parity Bit
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Parity-5 Problem (bipolar)					
Input 1	Input 2	Input 3	Input 4	Input 5	Parity Bit
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	1
-1	-1	-1	1	-1	1
-1	-1	-1	1	1	-1
-1	-1	1	-1	-1	1
-1	-1	1	-1	1	-1
-1	-1	1	1	-1	-1
-1	-1	1	1	1	1
-1	1	-1	-1	-1	1
-1	1	-1	-1	1	-1
-1	1	-1	1	-1	-1
-1	1	-1	1	1	1
-1	1	1	-1	-1	-1
-1	1	1	-1	1	1
-1	1	1	1	-1	1
-1	1	1	1	1	-1
1	-1	-1	-1	-1	1
1	-1	-1	-1	1	-1
1	-1	-1	1	-1	-1
1	-1	-1	1	1	1
1	-1	1	-1	-1	-1
1	-1	1	-1	1	1
1	-1	1	1	-1	1
1	-1	1	1	1	-1
1	1	-1	-1	-1	-1
1	1	-1	-1	1	1
1	1	-1	1	-1	1
1	1	-1	1	1	-1
1	1	1	-1	-1	1
1	1	1	-1	1	-1
1	1	1	1	-1	-1
1	1	1	1	1	1

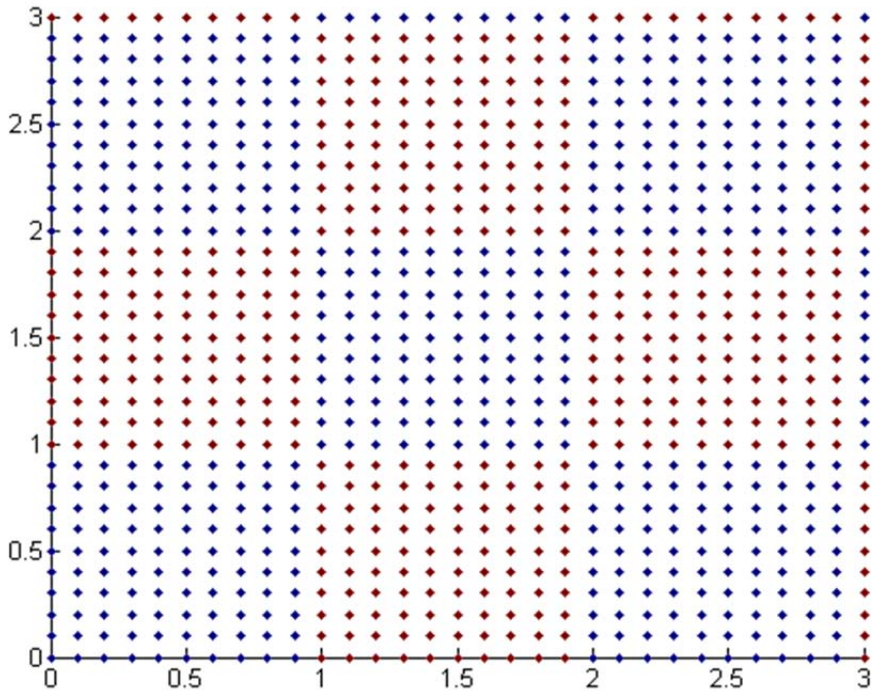
Table 2.2: Example Parity-N data sets – bipolar and unipolar

2.4.5 Benchmark #4 – Checker-N Problems

The Checker-N problem consists of a 10N x 10N grid similar to a checker board. The checker board consists of N^2 squares, each containing 100 data points. For a given ordered pair, the output is calculated with the following equation:

$$d(x, y) = [\lfloor x \rfloor + \lfloor y \rfloor \ mod\ 2] - 1 \quad \textbf{(7)}$$

where x and y are respectively (0, 0.1, 0.2, …, N) and $\lfloor . \rfloor$ is the floor operator [52]. Figure 2.11 shows a visual representation of the Checker-3 problem which contains 961 data points. Note that blue diamonds represent a +1 output while red diamonds represent a -1 output.



(a)

```
clear all; format compact; format short;
m=10
n=3;
np=m*n
j=0;
for x = 0:np
  for y = 0:np
    z=2*mod(floor(x*0.1) + floor(y*0.1),2) -1
    j=j+1;
    a(j,:)=[x*0.1,y*0.1,z]
  end
end
figure(1); clf;
scatter(a(:,1),a(:,2),3,a(:,3)'filled');
axis([0 3 0 3]);
whos
```

(b)

Figure 2.11: Checker-3 Problem – (a) Graph; and (b) MATLAB code

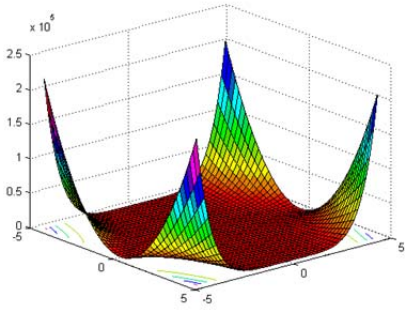
2.4.6 Additional Benchmarks

The previous four benchmarks are only a small portion of the many benchmarks that exist and are used in NN research. In his research on efficient optimization algorithms, Pham [48] utilizes many additional test functions. Thirteen additional test functions are outlined below. For functions that can be limited to 2 variables, graphs of the function are shown below the respective function.

1) Beale function

$$F = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$

(8)



2) Biggs Exp6 function

$$F = \sum_{i=1}^{n} [x_{i+2} e^{t_i x_i} - x_{i+3} e^{-t_i x_{i+1}} + x_{i+5} e^{t_i x_{i+4}} - y_i]^2$$

(9)

3) Box function

$$F = \sum_{i=1}^{n} [e^{-ax_i} - e^{-ax_{i+1}} - x_{i+2}(e^{-a} - e^{-10a})]^2$$

(10)

4) Colville function

$$F = \sum_{i=1}^{n} [100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2 + (x_{i+2} - 1)^2$$

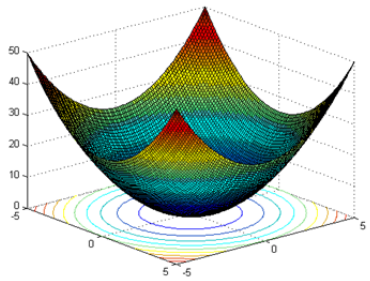
$$+ 10.1((x_{i+1} - 1)^2 + (x_{i+3} - 1)^2) + 90(x_{i+2}^2 - x_{i+3})^2$$

$$+ 19.8(x_{i+1} - 1)(x_{i+3} - 1)]$$

(11)

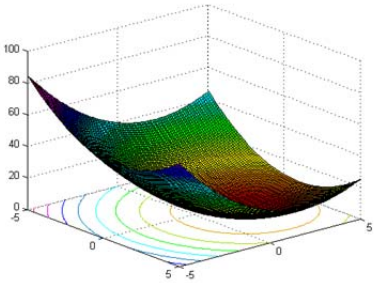
5) De Jong function

$$F = \sum_{i=1}^{n} x_i^2 \tag{12}$$



6) De Jong function with a moved axis

$$F = \sum_{i=1}^{n} (x_i - a_i)^2 \tag{13}$$

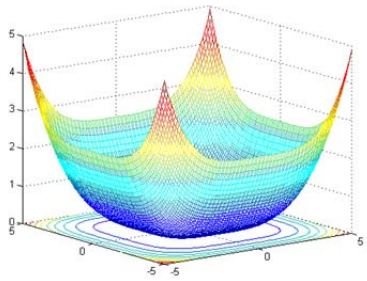


7) Powell function

$$F = \sum_{i=1}^{n} [(x_i + 10x_{i+1})^2 + 5(x_{i+2} - x_{i+3})^2 + (x_{i+1} - 2x_{i+2})^4 + 10(x_i - x_{i+3})^4] \tag{14}$$

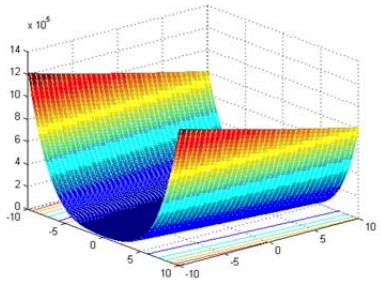
8) Quadruple function

$$F = \sum_{i=1}^{n} (\frac{x_i}{4})^4 \quad \textbf{(15)}$$



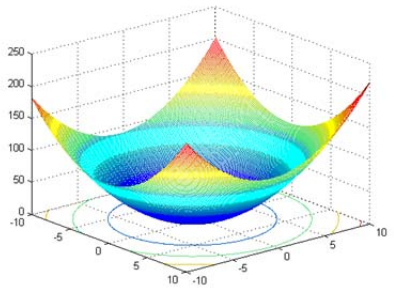
9) Rosenbrock function

$$F = \sum_{i=1}^{n} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \quad \textbf{(16)}$$



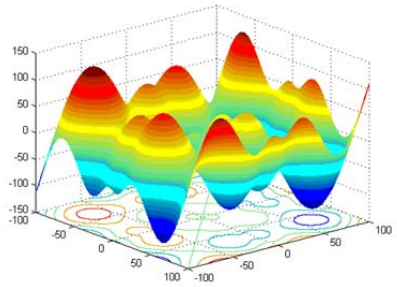
10) Step function

$$F = \sum_{i=1}^{n} |x_i + 0.5|^2 \quad \textbf{(17)}$$



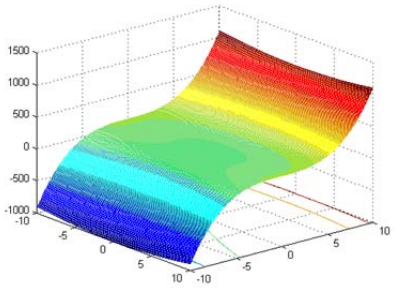
11) Schwefel function

$$F = 418.9829n - \sum_{i=1}^{n} \left(x_i sin\sqrt{|x_i|} \right) \quad \textbf{(18)}$$



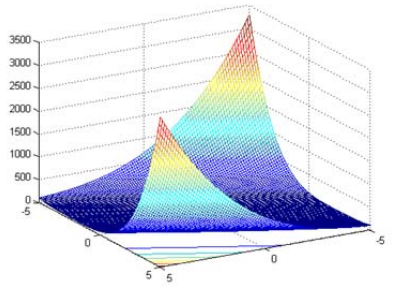
12) Sum of different power function

$$F = \sum_{i=1}^{n} |x_i|^{i+1} \tag{19}$$



13) Zarakov function

$$F = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{n} 0.5ix_i\right)^4 \tag{20}$$



2.5 Summary of Algorithms, Optimization, Topologies, and Benchmarks

The previous sections have been dedicated to understanding and outlining the current state of ANN research with regards to training, optimization, architectures, and benchmarks. In the training section, we discussed the main methods used for training

ANNs today, namely EBP, LM, and NBN. In the optimization section, we discussed optimizations made to the three mainstream algorithms and the advantages that these optimizations bring to bear. In the architecture section, the current state of each of the four main architectures was discussed. Time was spent discussing the power and efficiency of each network architecture and the Parity-N benchmark was used as a baseline to compare all four architectures. In the benchmark section, we discussed a small portion of the many benchmarks that are used for ANN training and testing.

Now, with an understanding of where research has brought ANNs over the past several decades, we are ready to look at the motivation for this work and the role of DNNs in architectural conversion.

Chapter 3

Motivation of Research

The exclusive-OR (XOR) and parity-N problems are used very frequently in digital systems. For example, a chain of XOR operators is used to convert Gray code to binary code. Also, parity-N circuits are essential for error detection and correction such as generation of a parity bit and checksums. Likewise, digital addition and multiplication require parity-N circuits. Additionally, parity-N circuits are often used in digital transmission systems to detect errors and they are also used in digital memory to detect hardware failures [14].

3.1 Digital Approach

In approaching XOR problems in a digital world, we are able to substitute a neuron for a digital logic gate or unit. Figure 3.1 shows and simple XOR module which utilizes 3 neurons.

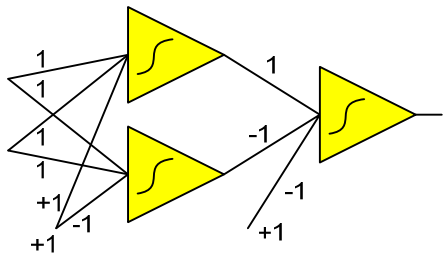


Figure 3.1: XOR module with ANNs as digital units (3 neurons)

To solve larger Parity-N problems in a digital world, the XOR module seen in

Figure 3.1 can be duplicated and combined to solve Parity-4, Parity-8, Parity-16, and so on. Each time the parity number is doubled and the number of required neurons is two times the parity number minus 1, or 2N-1.

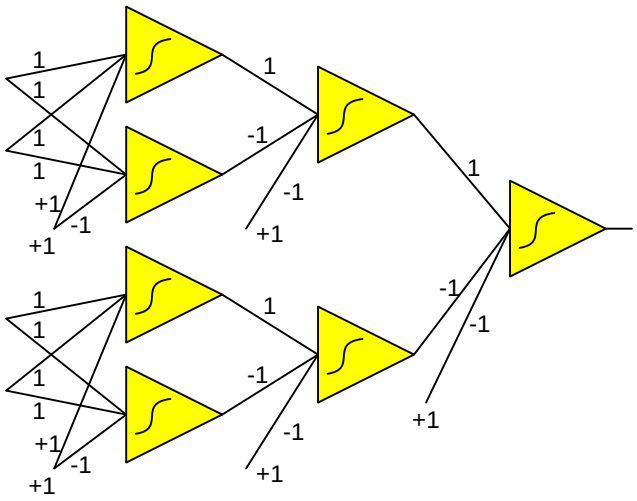


Figure 3.2: Parity-4 by combining XOR units (7 neurons)

Figure 3.2 shows a digital solution for the Parity-4 problem. In this solution, two XOR modules are placed side-by-side and their outputs are combined with the addition of a single output neuron. Notice that this network requires 2N-1, or 7 neurons. Similarly, figure 3.3 shows the digital solution for the Parity-8 problem. In this case, we placed two of the Parity-4 modules side-by-side and combined the outputs with a single output neuron. This network required 2N-1, or 15 neurons.

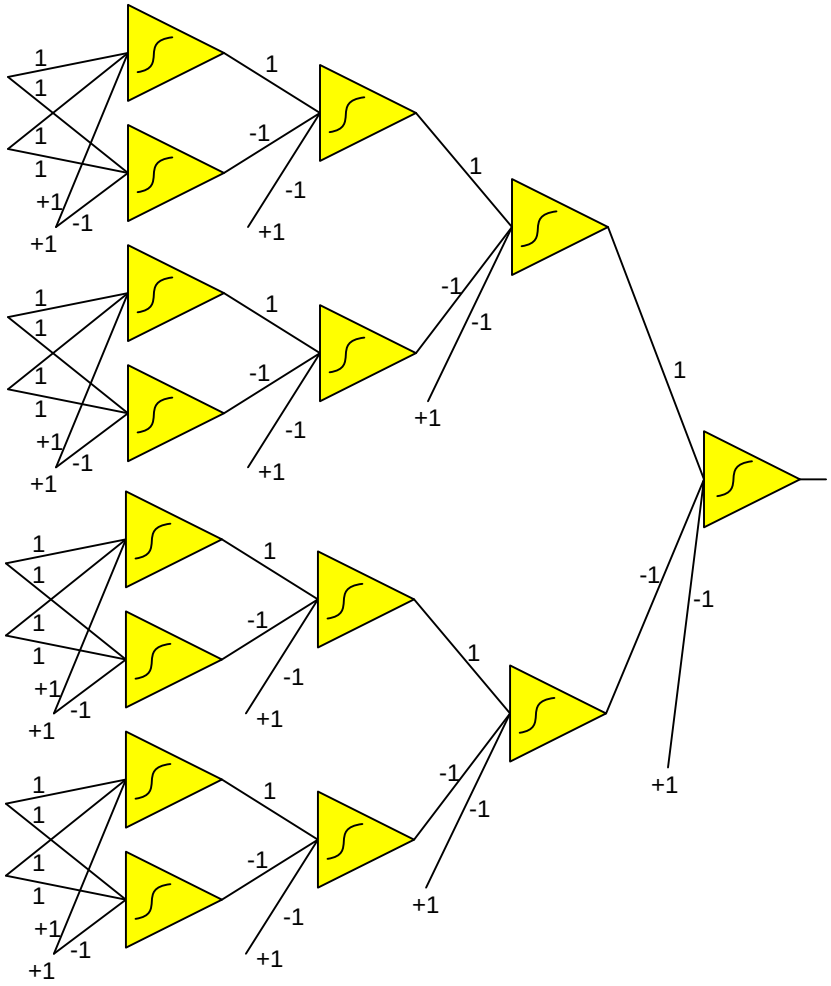


Figure 3.3: Parity-8 by combining XOR units (15 neurons)

3.2 Digital Implementation with ANNs

A significant research effort has been made for many decades toward optimizing the design of threshold logic networks. At the same time, many researchers were trying to solve the XOR and parity-N problems with artificial neurons that use non-linear activation functions.

In 1961, Minnink showed that solving the parity-N problem using threshold networks with one hidden layer required N hidden threshold units plus one output unit [53]. Since this time, the standard for solving a Parity-N problem using a MLP with one

hidden layer required N+1 neurons. Figures 3.4 and 3.5 show the networks for Parity-4 and Parity-8, respectively. Using the power of the non-linear activation function, the number of neurons required for Parity-4 and Parity-8 were reduced from 7 (digital approach) to 5 (ANN approach) and 15 (digital approach) to 9 (ANN approach), respectively.

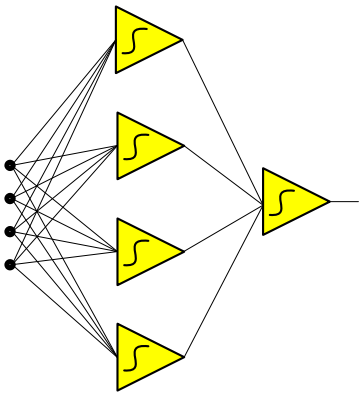


Figure 3.4: Parity-4 using MLP ANN architecture (5 neurons)

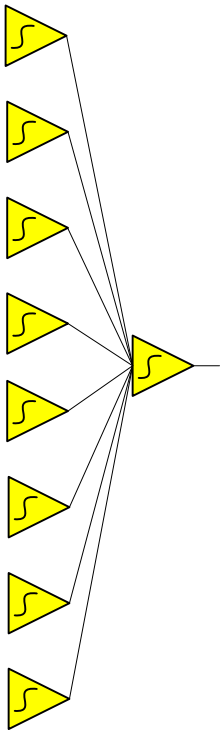


Figure 3.5: Parity-8 using MLP ANN architecture (9 neurons)

The reduction in the number of required neurons for Parity-N resulting from use of ANNs was significant. The requirement of N+1 neurons for a single hidden layer MLP network to solve the Parity-N problem has been the standard for over 50 years.

3.3 A Puzzle and a Discovery

The N+1 threshold discussed in section 3.2 has never really been overcome. However, in various experiments, I discovered that I was able to solve the Parity-N problem with fewer than N+1 neurons. Figures 3.6 and 3.7 show solutions for Parity-4 and Parity-8, respectively. Note that in Figure 3.7, it is assumed that all 8 inputs are connected to each of the neurons in the first hidden layer. These input connections are not shown to simplify the drawings.

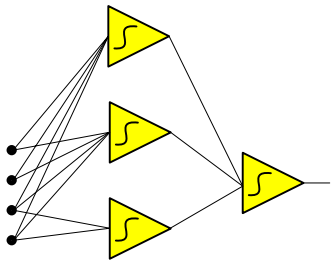


Figure 3.6: Parity-4 solution using only 4 neurons

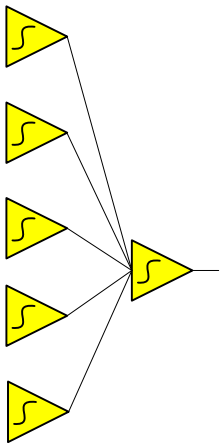


Figure 3.7: Parity-8 solution using only 6 neurons

Although the networks in Figures 3.6 and 3.7 should theoretically not be able to solve the Parity-4 and Parity-8 problems, I found that I was able to train these networks to do just that. The success rates for training these networks were extremely low; however, I was able to obtain solutions. These results violated the 50+ year old standard and raised the question, "How is it possible to solve the Parity-N problem with fewer than N+1 neurons in a MLP network?" I went forward, determined to answer this question.

After analyzing the training results for the Parity-4 and Parity-8 solutions, I found that at least one of the neurons in the first hidden layer of each MLP network was operating in its linear region. This was identified by very small input weights and a very large output weight on the linear-acting neuron. This was a significant discovery!

Once I realized that the N+1 threshold could be overcome by having one or more neurons operate in their linear region, I began to look at the potential advances that could come from creating a network with both linear and non-linear neurons. This type of network is the Dual Neural Network (DNN) discussed previously. I suspected that this type of network could provide significant advances in ANN research. I proceeded to investigate how a DNN could be used as a tool for training, optimization, and architectural conversion.

Chapter 4

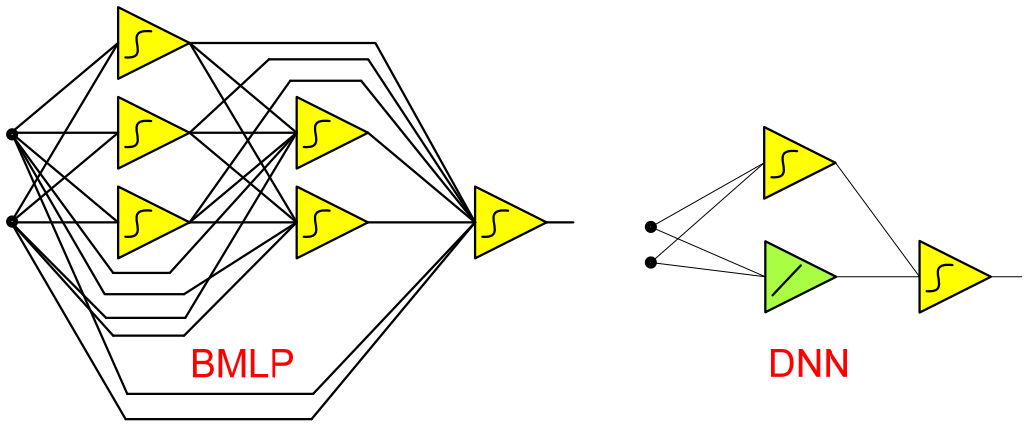
Network Architecture Conversion

Researchers to date have dedicated very little time and research to conversion between different neural network architectures. Generally, research has focused on the advantages of a particular neural network architecture over another. Many architectures have been developed with very specific problems or tasks in mind, leaving them very specialized and incapable of being used for solutions outside the realm of that specific research. However, when one explores the possibility of converting a particular neural network architecture to a different architecture which yields an identical result, a world of possibilities is opened.

For example, research has shown that some neural network architectures are easier to train, others require less time to train, and others are more likely to yield a solution to a training set [10]. Some architectures are more efficient, using a smaller number of neurons, and others less efficient, requiring more neurons for a solution [11].

Figure 4.1 shows an example of a 2=3=2=1 BMLP network and a 2-1(1)-1 DNN network. Please note that the networks in Figure 4.1 are not equivalent networks. Notice that in a BMLP network, all inputs are connected to all neurons in the network. In addition to this, the output of each neuron is connected to all neurons in forward layers. In the DNN network, notice that there are no cross-layer or bridged connections. The inputs are only connected to neurons in the first hidden layer and linear neurons are used

in each hidden layer to pass inputs to forward layers. This chapter will explore and discuss conversions between three different neural network architectures: BMLP, DNN, and MLP. All NN training will be performed with software developed by Yu and Wilamowski [35].



(a) (b)

Figure 4.1: (a) 2=3=2=1 BMLP and (b) 2-1(1)-1 DNN Architectures

4.1 Architecture Conversion Overview

After a significant amount of research, it was discovered that the DNN architecture is the key to conversion between BMLP and MLP architectures. While there is no direct conversion from MLP to BMLP or from BMLP to MLP, both of these architectures can be converted to and from the DNN architecture. Figure 4.2 shows three equivalent networks: 1) 2=1=1=1 BMLP; 2) 2-1(2)-1(1)-1 DNN; and 3) 2-3-2-1 MLP.

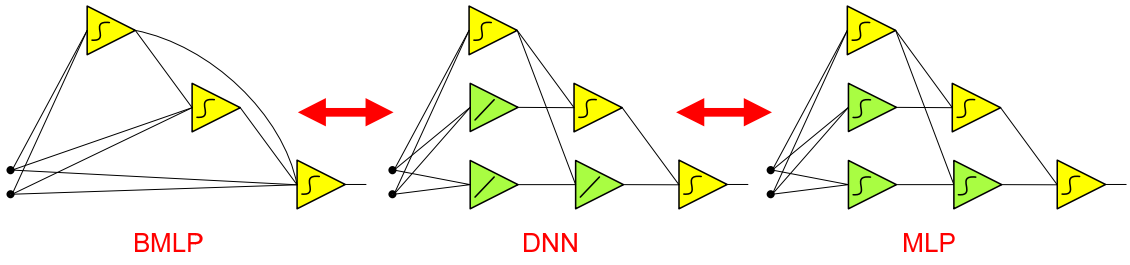


Figure 4.2: Conversion relationship between BMLP, DNN, and MLP

Note that if I train any one of the networks in Figure 4.2 to a given data set, I can convert the architecture and weights to either of the other two equivalent networks. If I start with a DNN network, then I can directly convert to either BMLP or MLP. If I start with a BMLP or MLP network and I desire to convert to the other, I must first convert to the DNN architecture and then to the desired architecture. In other words, the DNN architecture provides the path for conversion between BMLP and MLP.

With an understanding of the conversion relationship between BMLP, DNN, and MLP, it is now important to understand the process to complete these conversions. The next sections will cover the following conversions:

1) BMLP to DNN conversion
2) DNN to BMLP conversion
3) DNN to MLP conversion
4) MLP to DNN conversion

4.2 BMLP to DNN Conversion Process

This section will focus on the conversion from the BMLP architecture to an equivalent DNN architecture. A specific process is employed to convert from the BMLP architecture to the DNN architecture. This conversion has two different parts: 1) The architecture conversion; and 2) The weight conversion. We will look at each part of the conversion separately.

4.2.1 BMLP to DNN Architecture Conversion

One of the purposes of the BMLP to DNN architecture conversion is to eliminate cross-layer or bridged connections found in the BMLP architecture. To do this, linear neurons are inserted into the DNN architecture in each hidden layer to pass signals across

layers without creating bridged connections. For example, if a bridged connection in a BMLP network crossed 4 hidden layers, the DNN network would have 4 linear neurons, one in each hidden layer, to pass the signals to the destination neuron.

4.2.1.1 BMLP to DNN Examples

To understand how this architectural conversion works, it is beneficial for us to first look at several examples. Figures 4.3 and 4.4 show examples of simple BMLP networks converted to DNN networks.

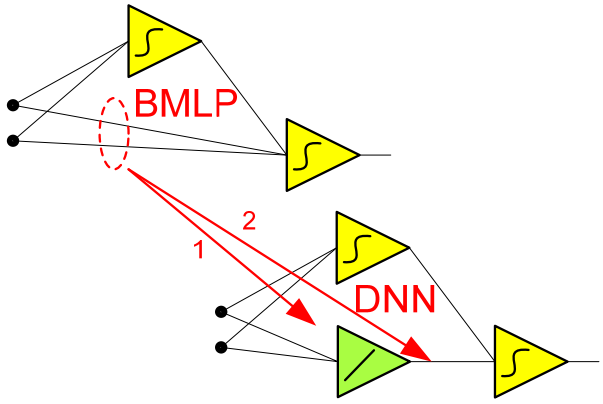


Figure 4.3: BMLP 2=1=1 to DNN 2-1(1)-1

Notice how the bridged connections are eliminated by inserting linear neurons. To create the DNN in Figure 4.3, the bridged connections going from the two inputs to the output neuron are replaced by connections to a linear neuron (1) in parallel with the bipolar neuron in the first hidden layer. The sum of the inputs to this linear neuron is then passed to the output neuron by a single connection (2) from the output of the linear neuron to the input of the output neuron. And this simple architecture conversion is complete.

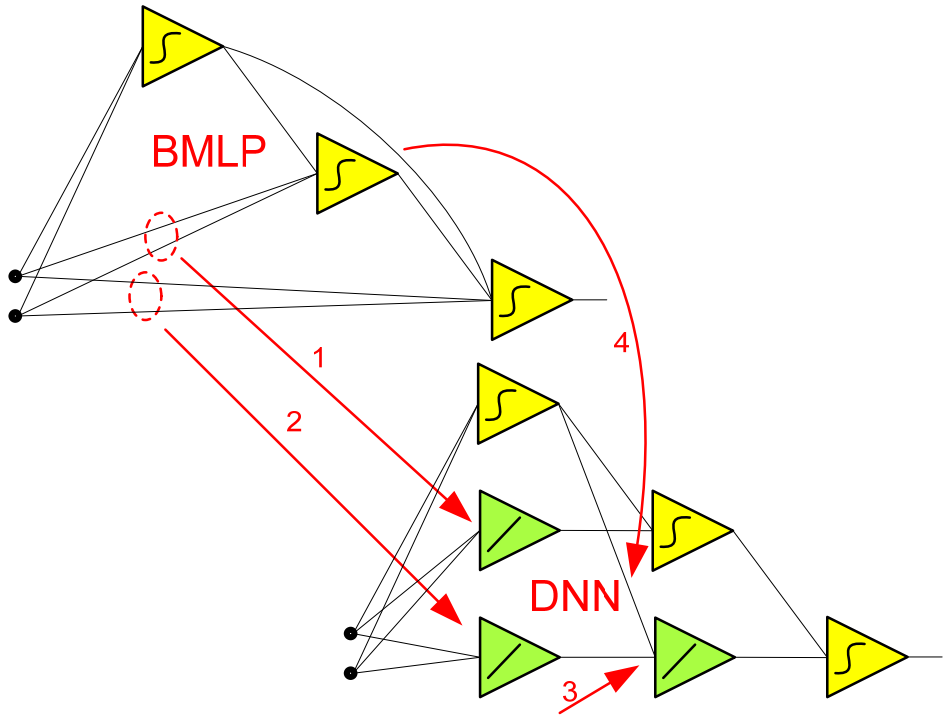


Figure 4.4: BMLP 2=1=1=1 to DNN

To create the DNN in Figure 4.4, the bridged connections going from the two inputs to the neuron in the second hidden layer and the output neuron had to be replaced. First, the two bridged inputs to the second hidden layer neuron are replaced by connections to a linear neuron (1) in parallel with the bipolar neuron in the first hidden layer. The sum of the inputs to this linear neuron is then passed to the second hidden layer neuron by a single connection.

Next, the two bridged inputs to the output neuron are replaced by connections to a linear neuron (2) in parallel with the bipolar neuron in the first hidden layer. The sum of the inputs to this linear neuron is then passed to another linear neuron (3) in parallel with the bipolar neuron in the second hidden layer. The sum of inputs to this linear neuron (3) is passed to the output neuron via a single connection.

Finally, the bridged connection from the output of the first hidden layer neuron to the input of the output neuron is replaced (4) by a connection from the output of the first hidden layer neuron to the input of the linear neuron (3) in layer 2. Thus, the output neuron receives input from the first hidden layer neuron via a linear neuron (3) in layer two rather than by a bridged connection. And, now, the architecture conversion is complete.

The previous two examples were quite simple and straight forward; however, let us look at a more complex conversion. Let us consider the BMLP network with an X=2=3=2=1 architecture found in Figure 4.5. As the number of inputs to the network is not important to it structure, we will ignore them in this example with the understanding that all inputs are connected to every neuron in the BMLP network.

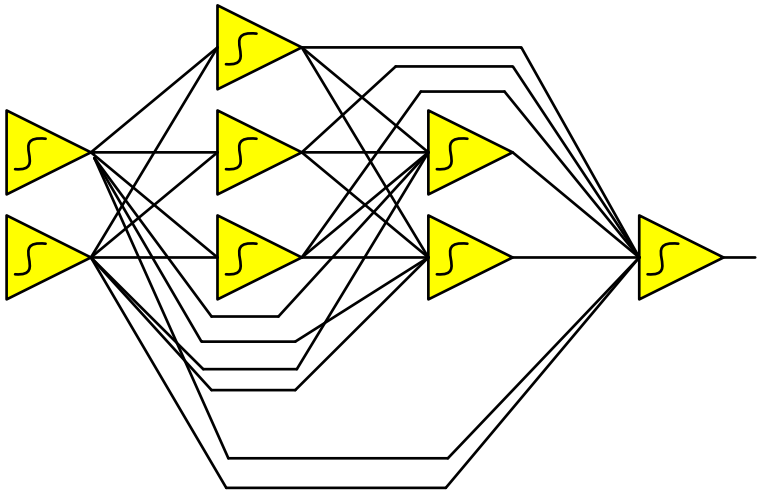


Figure 4.5: BMLP X=2=3=2=1 network

Using the same method as in the previous examples, the network in Figure 4.5 can be converted to the DNN network in Figure 4.6. It is understood that all inputs to the network will be connected to all neurons in the first hidden layer of the DNN network in Figure 4.6.

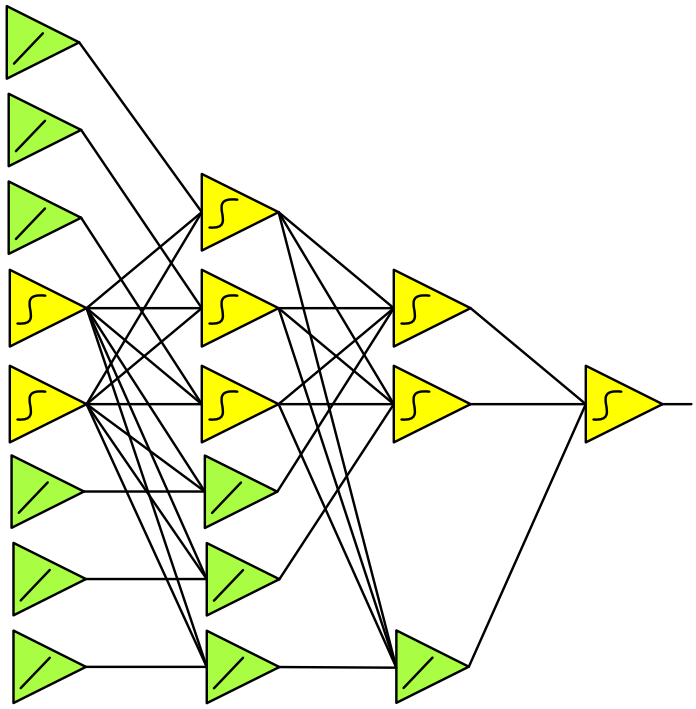


Figure 4.6: DNN network equivalent to BMLP in Figure 4.5

Let us now break down this more complex example from Figures 4.5 and 4.6 and determine how we arrived with the network in Figure 4.6. In doing this, we will reference the two networks side-by-side in Figure 4.7. Remember, it is understood that all inputs to the network will be connected to all neurons in the first hidden layer of the DNN network and all inputs are connected to all neurons in the BMLP network.

To create the DNN in Figure 4.7, the bridged connections going from the inputs to the neuron in the second and third hidden layers and the output neuron had to be replaced. First, the three bridged inputs to the second hidden layer neurons are replaced by connections to linear neurons (1) in parallel with the bipolar neurons in the first hidden layer. The sum of the inputs to this linear neuron is then passed to the second hidden layer neuron by a single connection.

Next, the bridged inputs to the two bipolar neurons in the third hidden layer are

replaced by connections to two linear neurons (2) in parallel with the bipolar neurons in the first hidden layer. The sum of the inputs to these linear neurons is then passed to two additional linear neurons (3) in parallel with the bipolar neurons in the second hidden layer. The sum of inputs to these linear neurons (3) is passed to the two bipolar neurons in the third hidden layer via a single connection (4).

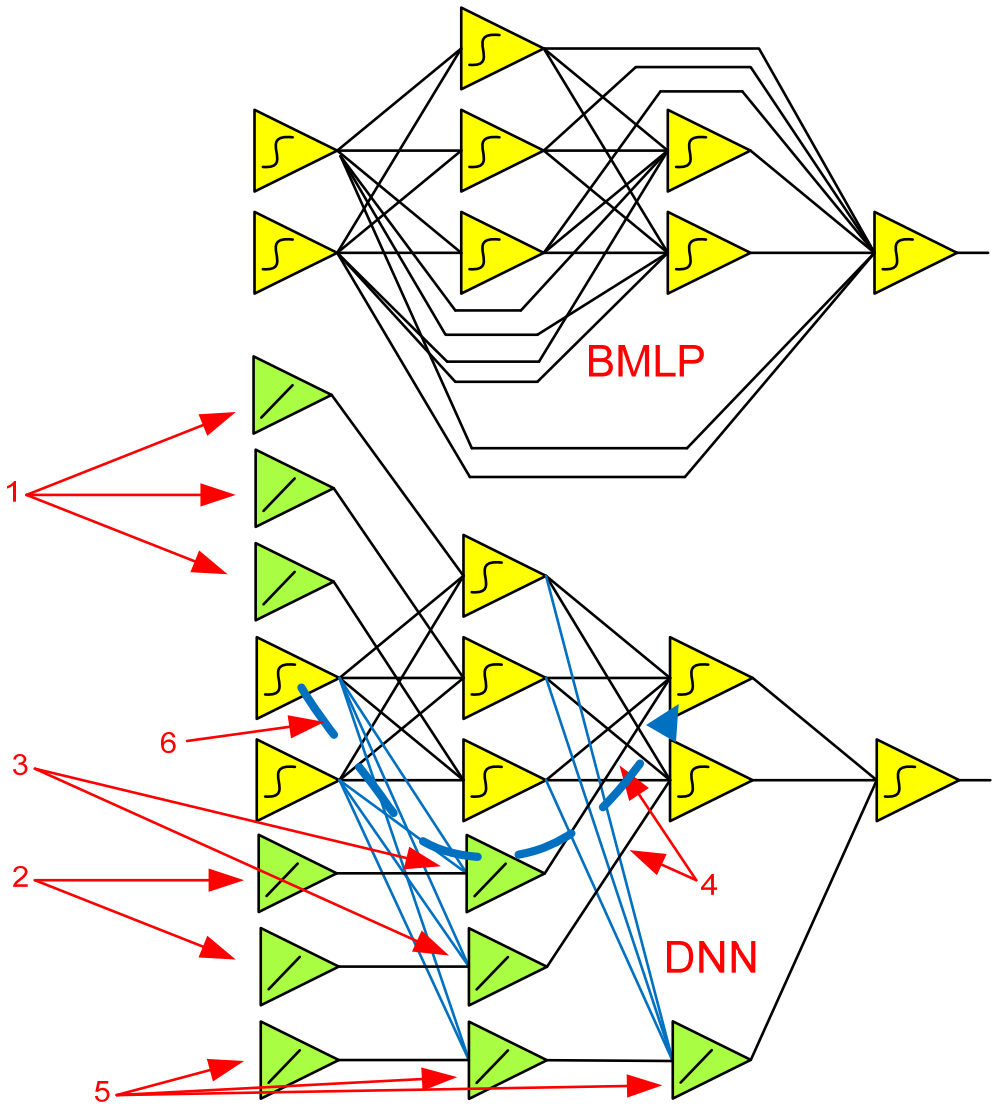


Figure 4.7: BMLP X=2=3=2=1 to DNN X-2(6)-3(3)-2(1)-1

Then, the bridged inputs going to the output neuron are replaced by a series of 3

linear neurons, one in each of the first three hidden layers (5). The sum of the inputs to the first linear neuron in this series is passed to each of the future neurons by a single connection between the neurons in each layer. Thus, the output neuron receives input via three linear neurons in series rather than by bridged connections.

Finally, we deal with the connections between hidden layers. Instead of using bridged connections to go between layers, connections are routed to the linear neuron in next hidden layer that feeds the destination neuron. These connections are blue in Figure 4.7. Notice how the bridged connection from one bipolar neuron in the first hidden layer to a bipolar neuron in the third hidden layer is replaced with a connection to a linear neuron in the second hidden layer. This path is highlighted with a blue dotted line (6). Now, the architecture conversion is complete.

4.2.1.2 BMLP to DNN Architecture Conversion Methodology

The previous three examples show progressively more complicated conversions. After significant analysis, standardized methods have been developed with simple formulas that determine the number of non-linear and linear neurons in each layer. Table 4.1 contains the variables used for BMLP to DNN architecture conversion. As seen in this table, when converting from the BMLP architecture to the DNN architecture, the DNN architecture will have the same total number of layers and the same number of

Variable	Description
K	The total number of layers in both the BMLP and DNN networks.
n_L	Number of non-linear neurons in the L^{th} layer of both the BMLP and DNN networks where L is from 1 to K
h_L	Number of linear neurons in the L^{th} layer of the DNN network where L is from 1 to K. This is a calculated value.

Table 4.1: Variables used for BMLP to DNN conversion

non-linear neurons in each hidden layer. The only parameter that must be calculated is the number linear neurons in each hidden layer of the DNN architecture. From analysis, the number of linear neurons in each hidden layer is equal to the number of bipolar neurons in all future layers combined. This value can be calculated with Equation 21.

$$h_L = (\sum_{x=L}^{K} n_x) - n_L \tag{21}$$

Note that the number of inputs to the network has no effect on the architectural conversion. For this reason, I have limited the number of inputs in the example networks to two in order to simplify the drawings and make them less cluttered by connections between neurons. Now, with all of the conversion tools at our disposal let us now consider a more complex circuit like that found in Figure 4.8. This figure shows a BMLP network with a 2=3=2=1 architecture. As seen in the figure, there are two inputs, three

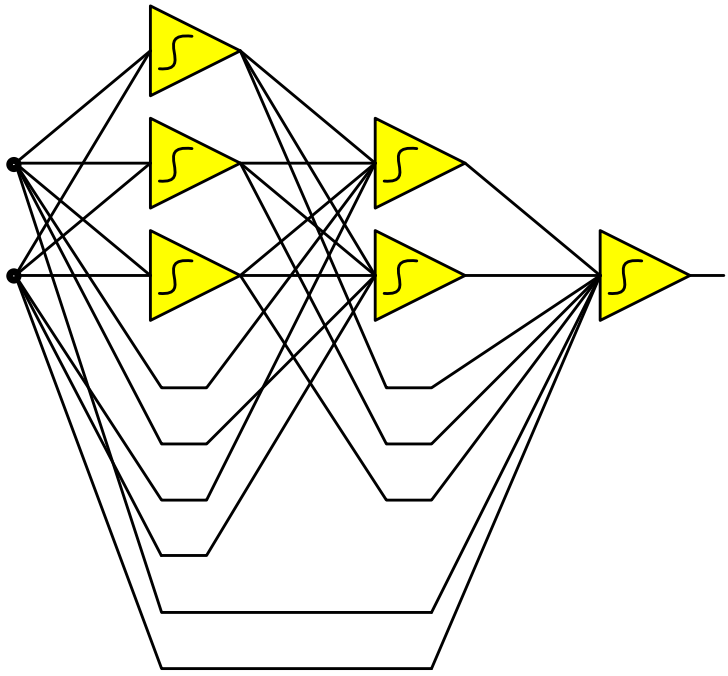


Figure 4.8: BMLP 2=3=2=1 Network

non-linear neurons in the first hidden layer, two non-linear neurons in the second hidden layer, and one non-linear neuron in the output layer. Now, with an understanding of Table 4.1 and Equation 21, we can calculate the architecture of the DNN network which is equivalent to the BMLP network in Figure 4.8. The calculations for h_1, h_2, and h_3 are:

$$h_1 = (\sum_{x=1}^{3} n_x) - n_1 = 3$$

$$h_2 = (\sum_{x=2}^{3} n_x) - n_2 = 1$$

$$h_3 = (\sum_{x=3}^{3} n_x) - n_3 = 0$$

Remember that the number of non-linear neurons in each layer, nL, is the same for both architectures. A summary of our results can be found in Table 4.2 and the corresponding DNN network can be found in Figure 4.9.

Layer	$\boldsymbol{n_L}$	$\boldsymbol{h_L}$
1	3	3
2	2	1
3	1	0

Table 4.2: Parameters for DNN Network equivalent to BMLP in Figure 4.3

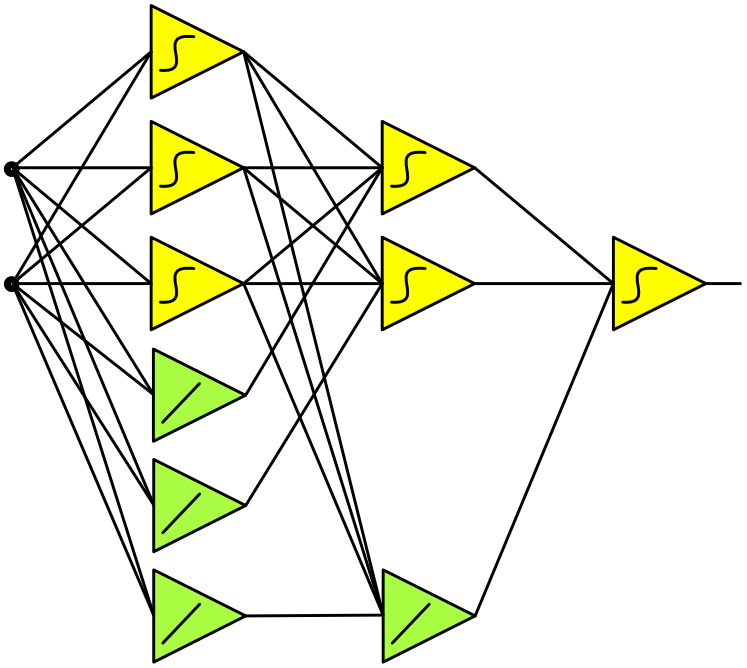


Figure 4.9: DNN network equivalent to BMLP in Figure 4.8

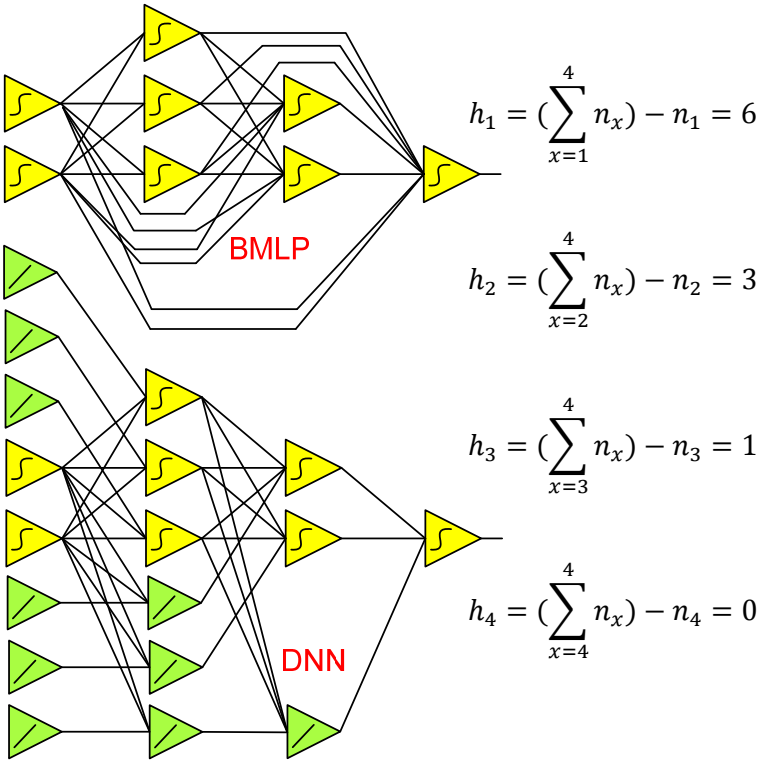


Figure 4.10: X=2=3=2=1 BMLP to X-2(6)-3(3)-2(1)-1DNN conversion with calculations

Let us look at the conversion example seen in Figure 4.7 again. This time, we will use Equation 21 and Table 4.1 to validate the conversion. Figure 4.10 shows the BMLP and DNN networks in Figure 4.7 along with the calculations for the number of linear neurons in each hidden layer. Table 4.3 summarizes all the parameters for the DNN network. Thus we see that the conversion shown in Figure 4.7 is accurate.

Layer	n_L	h_L
1	2	6
2	3	3
3	2	1
4	1	0

Table 4.3: Parameters for the conversion in Figure 4.10

4.2.1.3 BMLP to DNN Architecture Conversion Summary

This section has focused on the BMLP to DNN architecture conversion. As we saw, the number of layers in the network and the number of non-linear neurons in each layer did not change. It is only necessary to calculate the number of linear neurons for each hidden layer and understand how they are placed and connected. Notice that in all cases, no non-linear neurons will be added to the output layer. With an understanding of the architecture conversion, we must now gain an understanding of the weight conversion.

4.2.2 BMLP to DNN Weight Conversion

With an understanding of the BMLP to DNN architecture conversion, we are now prepared to discuss the weight conversion. This weight conversion follows three rules:

1) All weights for non-bridged connections remain the same.
2) Weights for bridged connections remain the same, but are transferred to the new network connection going to the linear neuron in the next hidden layer.

3) The weight on the output of every linear neuron is 1.

These rules may be a little bit confusing, so it is easiest to illustrate the application of the three rules on networks that we have looked at previously in this chapter. Let's look at the network conversion in Figures 4.3 and 4.4. We will use these same networks to illustrate the weight conversion. To simplify the network diagrams for illustrative purposes, unique variable weights will be assigned so that it is easy to see how the weights are converted. Figure 4.11 shows the weight conversion for the network in Figure 4.3. Figure 4.12 shows the weight conversion for the network in Figure 4.4.

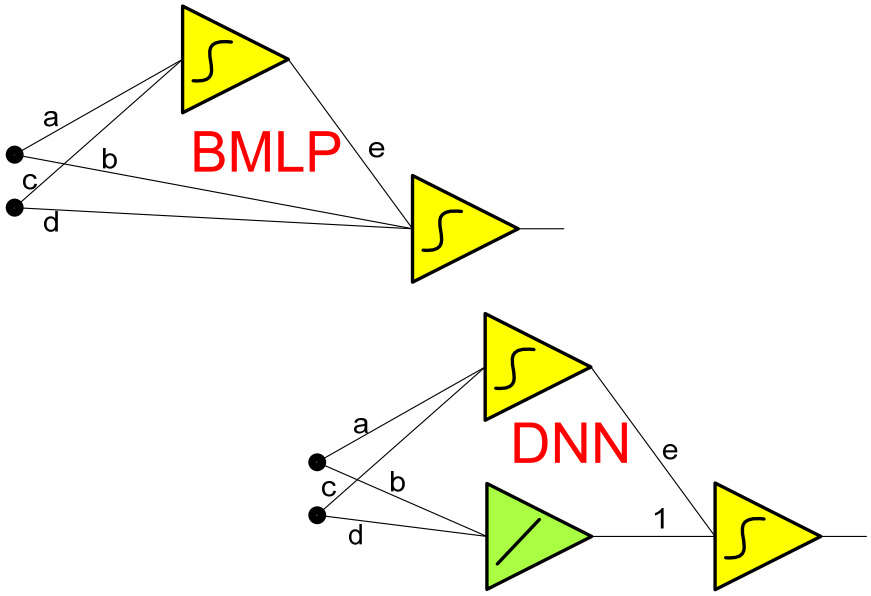


Figure 4.11: Weight Translation of networks in Figure 4.3

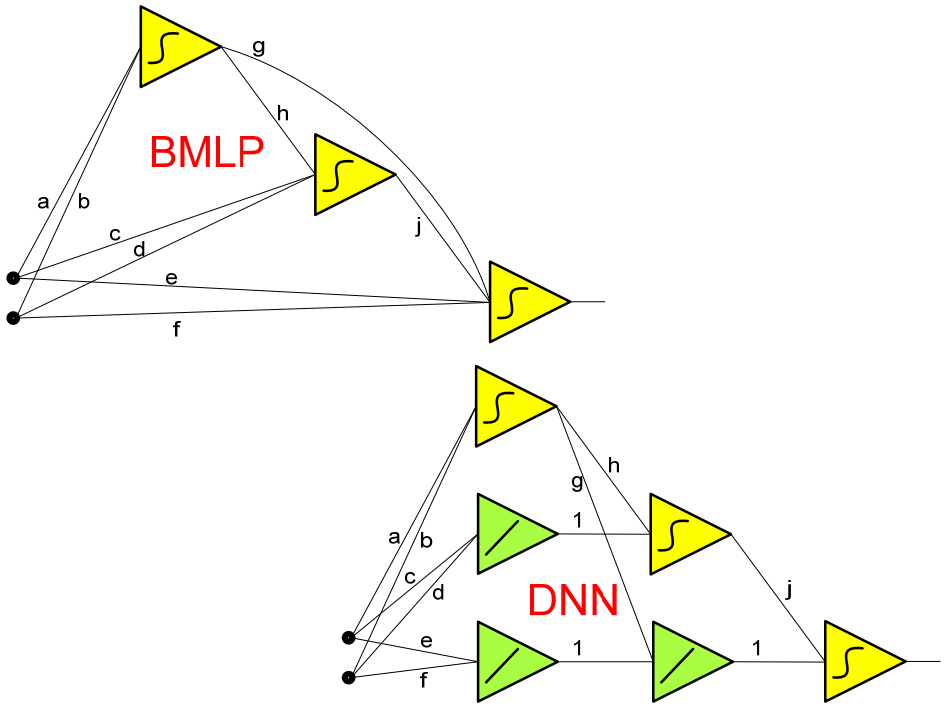


Figure 4.12: Weight Translation of networks in Figure 4.4

As can be seen in the examples illustrated in Figures 4.11 and 4.12, the weight conversion during the BMLP to DNN conversion is simple and straight forward. There is now a clear path for conversion between BMLP network architectures and the DNN network architecture. The opposite conversion will be dealt with later in this chapter.

4.2.3 BMLP to DNN Conversion Examples

The process for converting a BMLP network to an equivalent DNN network is rather simple and straight forward with the information that we have just covered. This section will focus on an overview of the conversion tests performed as part of this research and the implications of these tests.

4.2.3.1 BMLP to DNN Using Simple 3-D Surface Benchmark

The first conversion test was performed using the Simple 3-D Surface benchmark. A BMLP network was built with two architectures: 2=3=2=2=1 and 2=2=3=2=1. In both

cases, the BMLP network was trained to SSE ≤ 0.01. Once the training was completed, each network was converted to the DNN architecture using the standard conversion process. The DNN equivalent architectures gave identical output results with no variation. Figure 4.13 shows the trained 2=3=2=2=1 BMLP. Figure 4.14 shows the DNN equivalent network.

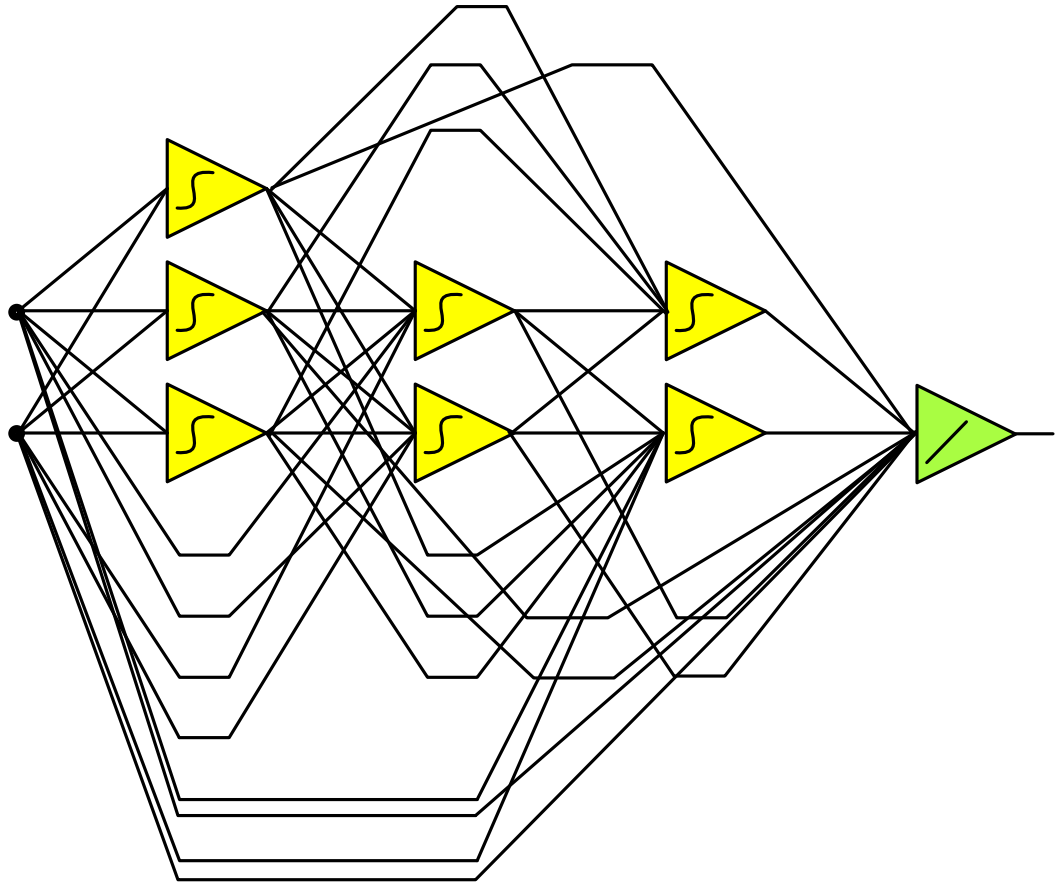


Figure 4.13: 2=3=2=2=1 BMLP network for the Simple 3-D Surface

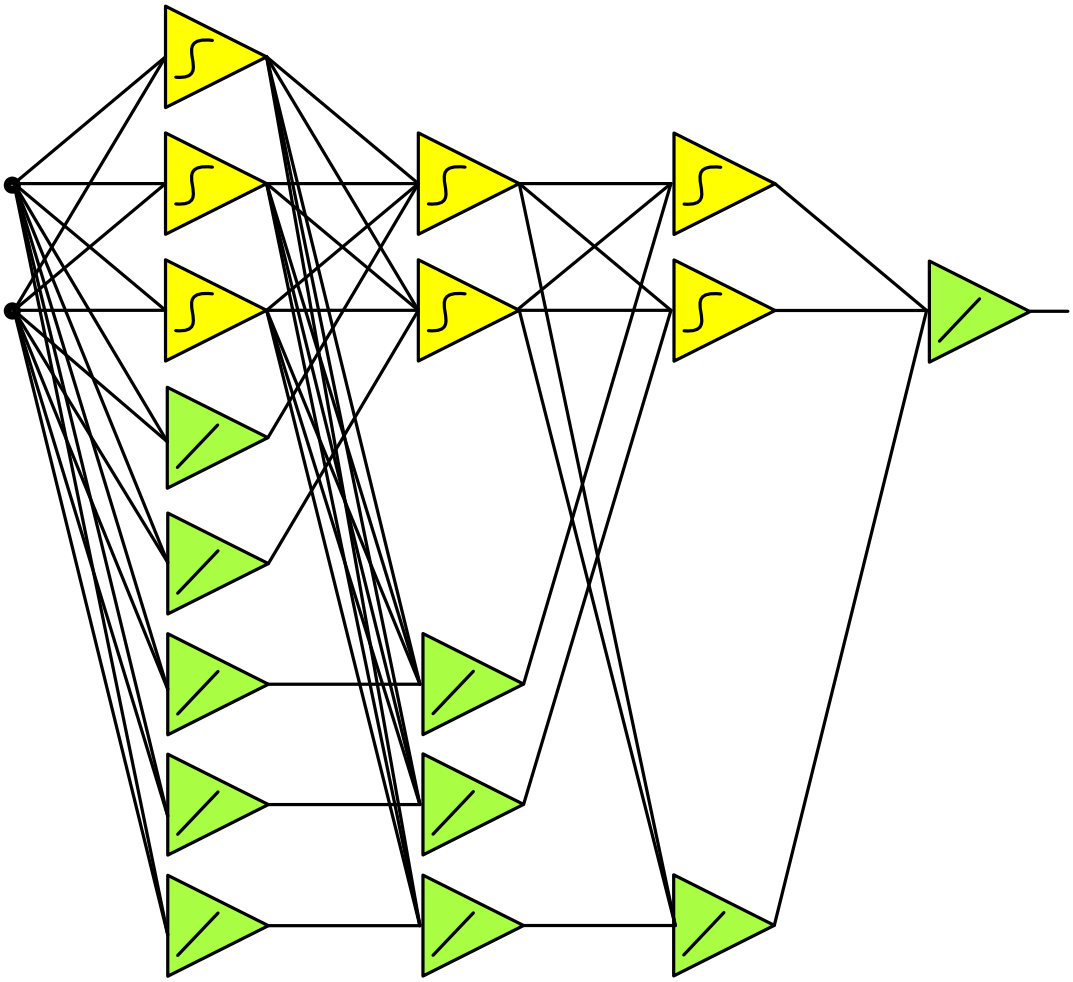


Figure 4.14: DNN equivalent network for the network in Figure 4.13

As mentioned previously, the networks in Figures 4.13 and 4.14 yielded identical results to the benchmark data set.

4.2.3.2 BMLP to DNN Using 3-D Surface Benchmark

The second conversion test was performed using the 3-D Surface benchmark. This time, a BMLP network was built with the architecture 2=3=2=1. The BMLP network was trained to SSE ≤ 0.01. Once the training was completed, the network was converted to the DNN architecture using the standard conversion process. Figure 4.15 shows the BMLP network and Figure 4.16 shows the DNN equivalent. Note that the

DNN equivalent architecture gave identical output results with no variation.

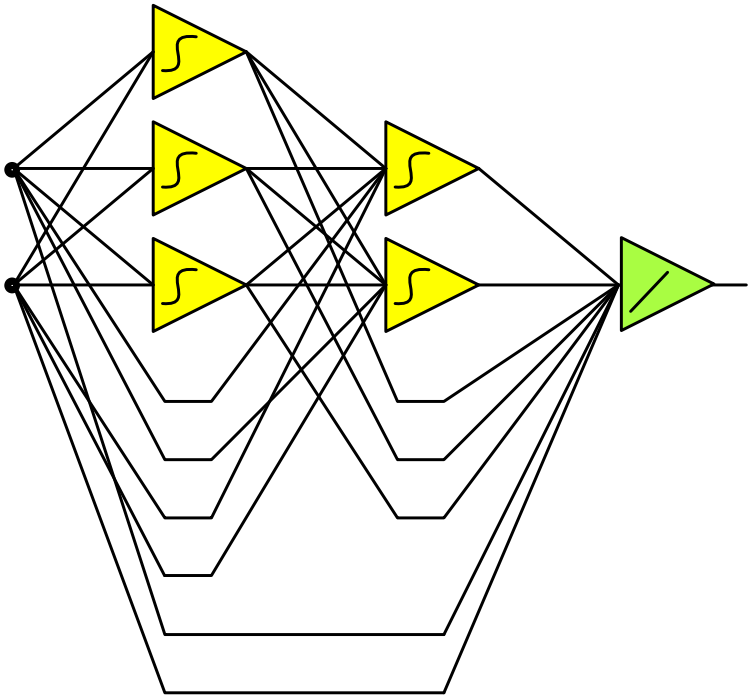


Figure 4.15: 2=3=2=1 BMLP Network for 3-D Surface Benchmark

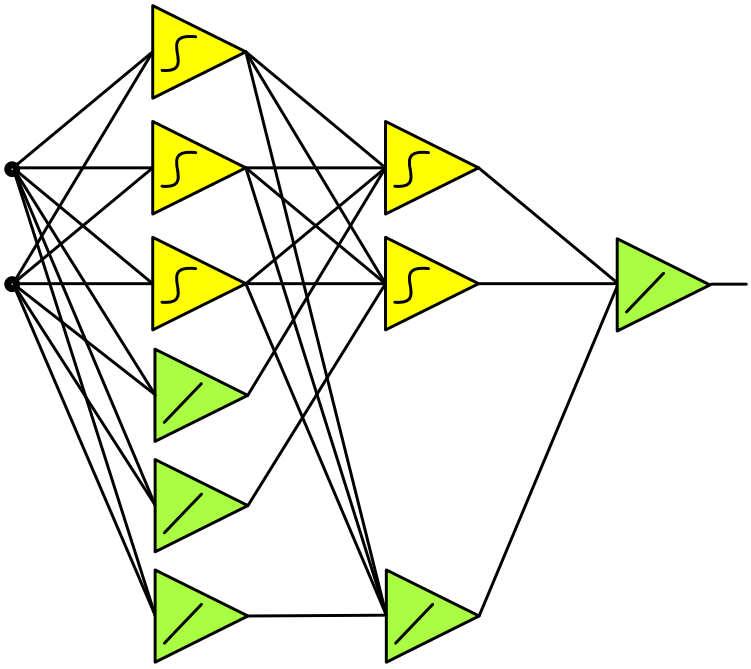


Figure 4.16: DNN equivalent network for the network in Figure 4.15

4.2.3.3 BMLP to DNN Using Parity-N Benchmark

The third conversion test was performed using the Parity-N benchmark. The BMLP network was built for Parity-11 using an 11=2=1=1 architecture. The BMLP network was trained to SSE ≤ 0.01. Once the training was completed, the network was converted to the DNN architecture using the standard conversion process. Figure 4.17 shows the BMLP network and Figure 4.18 shows the DNN equivalent. Note that the DNN equivalent architecture gave identical output results with no variation.

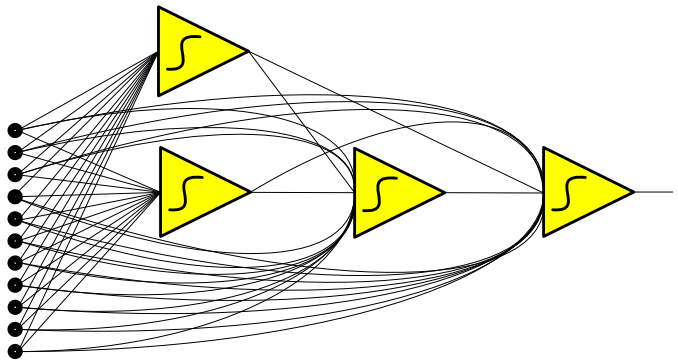


Figure 4.17: 11=2=1=1 BMLP Network for Parity-11 Benchmark

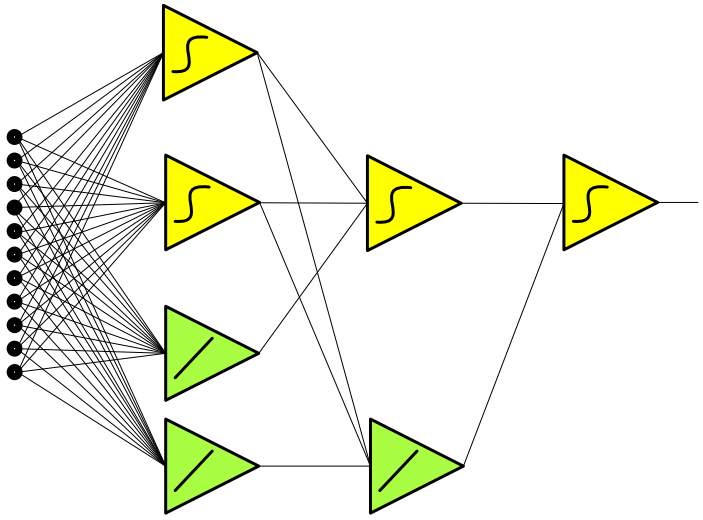


Figure 4.18: DNN equivalent network for the network in Figure 4.17

4.2.3.4 BMLP to DNN Conversion Summary

The results in Section 4.2.3 are significant because we have shown with multiple data sets and multiple BMLP architectures that we can train a given BMLP network to any given data set and then convert that trained network to an equivalent DNN architecture. This equivalent architecture will give us identical results to those obtained with the original BMLP network.

4.3 DNN to BMLP Conversion Process

This section will focus on the conversion from the DNN architecture to an equivalent BMLP architecture. Under certain circumstances, it may be desirable to minimize the total number of neurons in a NN with the trade-off of added bridged connections. A specific process is employed to convert from the DNN architecture to the BMLP architecture. This conversion has two different parts: 1) The architecture conversion; and 2) The weight conversion. We will look at each part of the conversion separately.

4.3.1 DNN to BMLP Architecture Conversion

One of the purposes of the DNN to BMLP architecture conversion is to minimize the total number of neurons in the NN architecture. To do this, linear neurons are removed from the DNN architecture and are replaced with bridged connections. For example, if a series of three linear neurons (1 in each hidden layer prior to the output layer) feeds the sum of the inputs to the output neuron, the three linear neurons are removed and a bridged connection is made from each input to the output neuron. Additionally, connections from the output of non-linear neurons in the hidden layers to linear neurons in the next hidden layer are replaced by bridged connections to the

destination non-linear neuron.

4.3.1.1 DNN to BMLP Examples

To understand how this architectural conversion works, it is beneficial for us to first look at several examples. We will revisit many of the networks presented previously. Figures 4.19 and 4.20 show examples of simple DNN networks converted to BMLP networks.

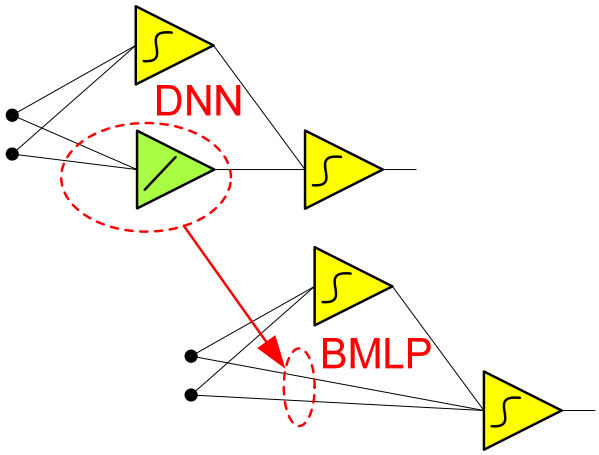


Figure 4.19: DNN 2-1(1)-1 to BMLP 2=1=1

Notice in Figure 4.19 how the linear neuron was removed and the two connections to its inputs were simply extended to the input of the output neuron. Although this is a very simple example, it shows that we are able to simply remove linear neurons and create bridged connections to replace them and the inputs that they receive.

Notice that the network in Figure 4.20 is more complex than the one in Figure 4.19. Its conversion will take more time. First, a single linear neuron in the first hidden layer is removed (1) and the two connections it received are replaced with bridged connections to the bipolar neuron in the second hidden layer. Next, two linear neurons which pass outputs to the output neuron are removed (2) and the connections coming

from the input are replaced with bridged connections to the output neuron.

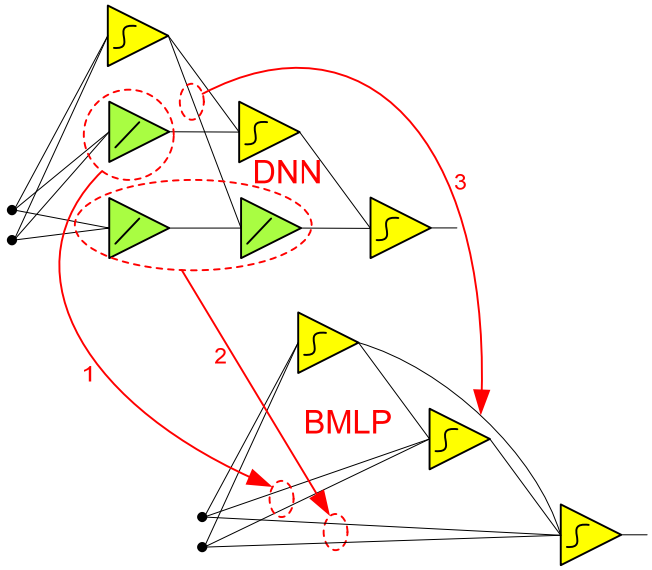


Figure 4.20: DNN 2-1(2)-1(1)-1 to BMLP 2=1=1=1

With the removal of the last two neurons (2), the output from the bipolar neuron in the first hidden layer to the linear neuron in the second hidden layer must be replaced with a bridged connection (3). With the placement of this last bridged connection, the conversion from DNN to BMLP is complete.

4.3.1.2 DNN to BMLP Architecture Conversion Methodology

The previous two examples show the basic concepts for converting from the DNN architecture to the BMLP architecture. Referring back to Table 4.1 and what we learned earlier in this chapter, we know the following about the BMLP network:

1) The BMLP network will have the same number of layers as the DNN network.
2) The number of non-linear neurons in each of the hidden layers will remain the same.
3) The output neuron(s) will remain the same.

With an understanding of the points outlined above, we can now outline a

standardized methodology for the DNN to BMLP conversion. As noted above, the main idea behind this conversion is to remove all linear neurons and replace them with bridged connections. The process for this conversion is as follows:

1) Identify all of the linear neuron series in the network.
2) Of the linear neuron series remaining in the network, work on the series with the highest number of neurons in it. Trace the series from the inputs to the final destination neuron.
3) Identify inputs to linear neurons in the series that come from neurons outside of the series. Replace these inputs with bridged connections from the source neuron to the final destination neuron.
4) After all tangential connections to the series are removed, remove all linear neurons in the series and replace them with bridged connections from all inputs that went to the first linear neuron in the series to the final destination neuron.
5) Repeat steps 2-4 until there are no linear neurons remaining in the network.

These steps may seem a little confusing, so let us look at two examples on the more complex networks found in Figures 4.9 and 4.10. We will start with Figure 4.9 which is smaller and will apply the steps outlined above in order. Figure 4.21 shows the network in Figure 4.9 with all linear neuron series identified with blue arrows.

In Figure 4.22, we perform steps 2 and 3 on the largest linear neuron series. The three outside inputs to the linear neuron series are identified by dotted blue lines. These connections are removed and replaced with the three bridged connections which are red.

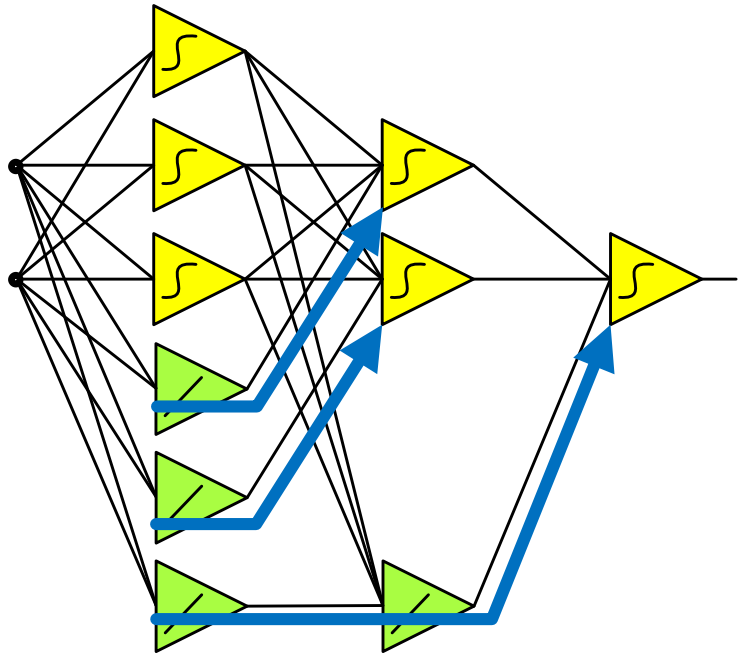


Figure 4.21: 2-3(3)-2(1)-1 DNN conversion Step 1

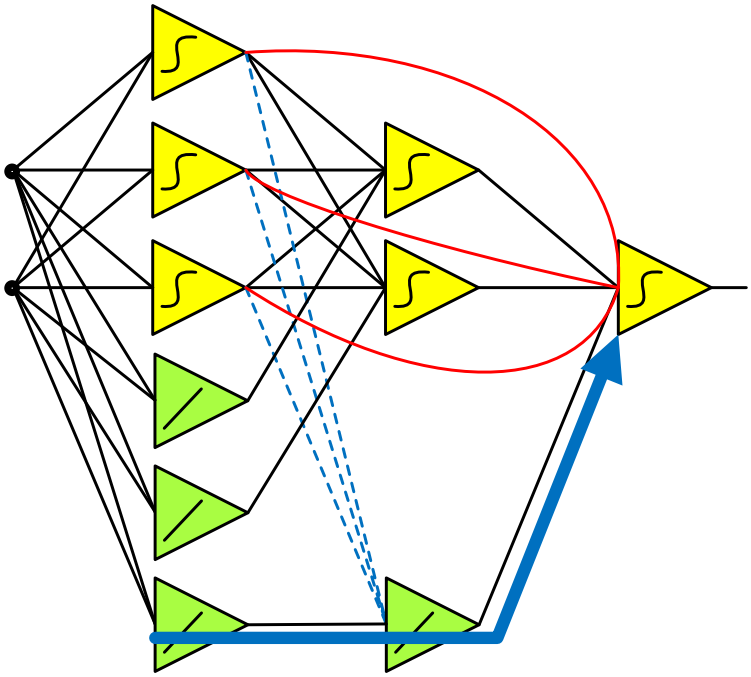


Figure 4.22: 2-3(3)-2(1)-1 DNN conversion Steps 2 and 3

In Figure 4.23, we perform step 4. Here, we remove the two linear neurons in that series along with the associated connections. We then replace them with two bridged connections from the two inputs to the output neuron. These new connections are red.

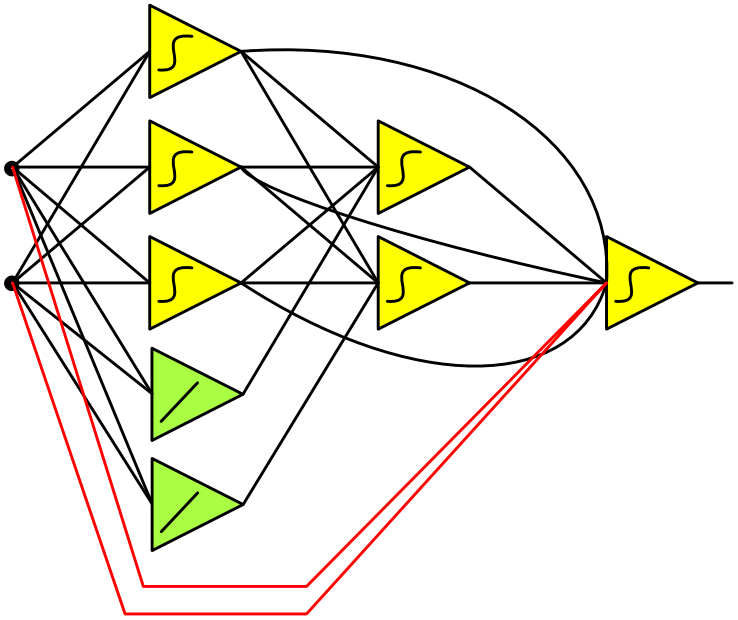


Figure 4.23: 2-3(3)-2(1)-1 DNN conversion Step 4

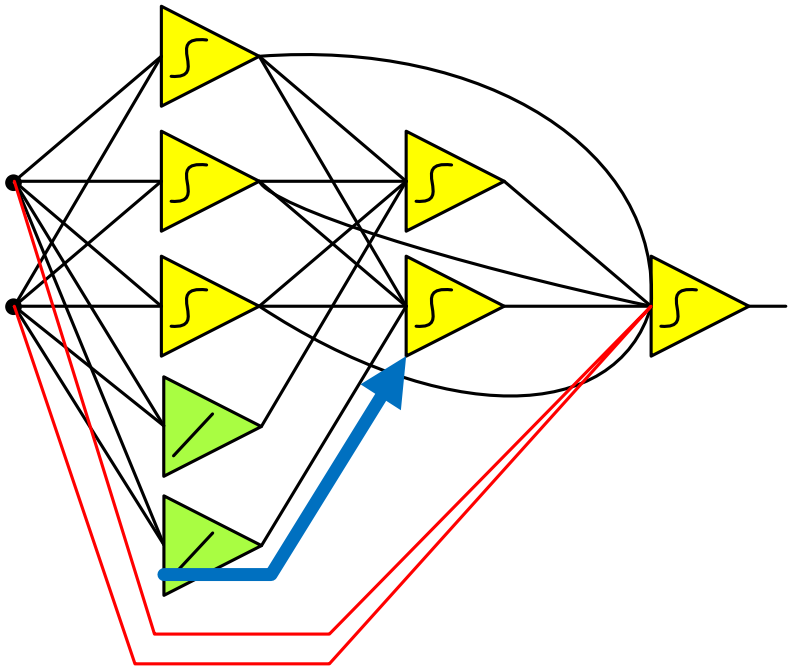


Figure 4.24: 2-3(3)-2(1)-1 DNN conversion Step 2 repeated

I will now repeat steps 2 through 4 for the remaining two linear neurons. As neither of the two remaining linear neurons has outside connections, we are able to skip step 3 and only do steps 2 and 4. Figure 4.24 identifies the linear neuron series we will work on. Figure 4.25 shows replacement of the linear neuron with bridged connections.

We will now repeat steps 2 and 4 one more time to complete the conversion. Figure 4.26 identifies the linear neuron series we will work on. Figure 4.27 shows replacement of the linear neuron with bridged connections. As always, the new connections are red. This step is now complete.

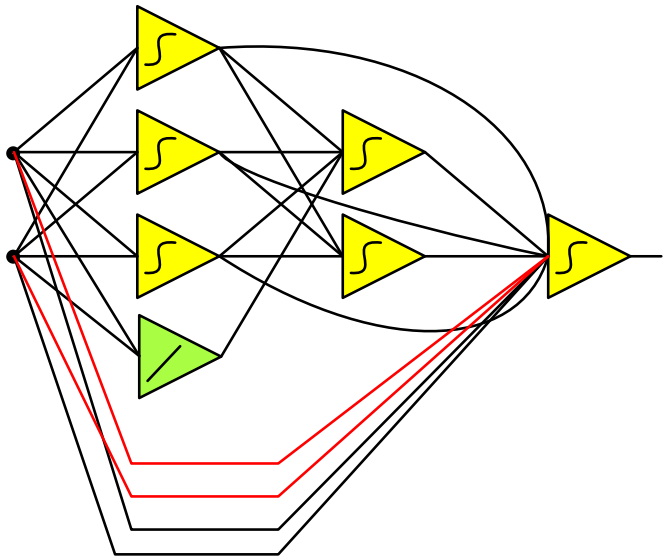


Figure 4.25: 2-3(3)-2(1)-1 DNN conversion Step 4 repeated

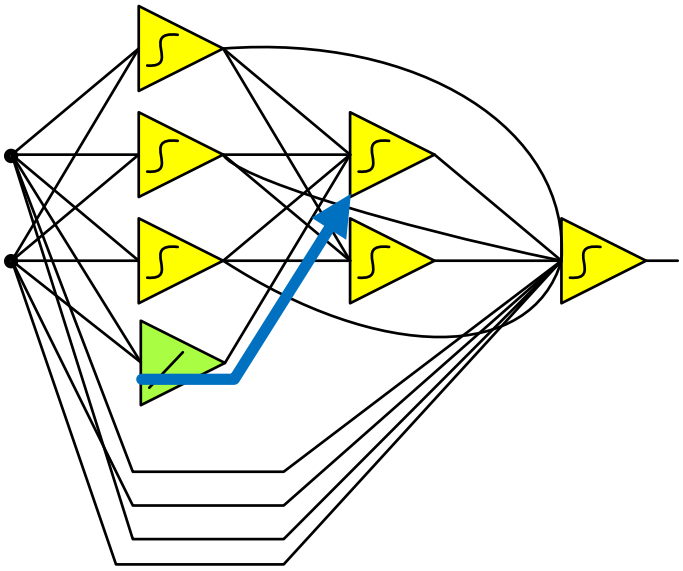


Figure 4.26: 2-3(3)-2(1)-1 DNN conversion Step 2 repeated

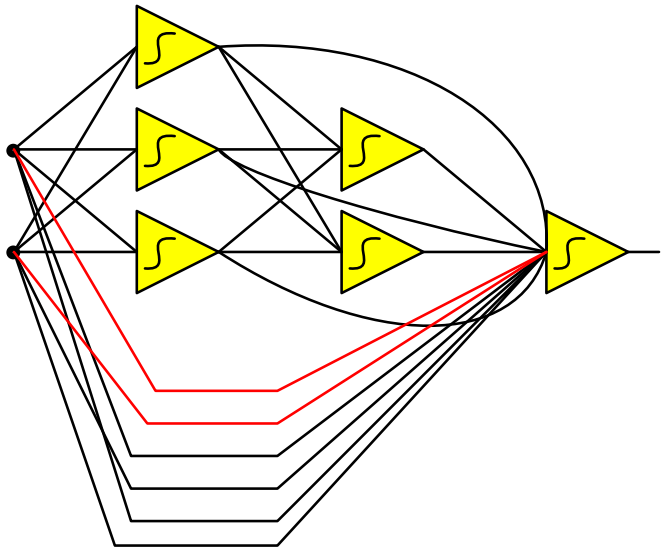


Figure 4.27: 2-3(3)-2(1)-1 DNN conversion Step 4 repeated

Let us now look at a more complex network which is found in Figure 4.10. Figure 4.28 shows the network in Figure 4.10 with all of the linear neuron series identified with blue arrows. For simplicity, it is assumed that all inputs are connected to all neurons in the first hidden layer.

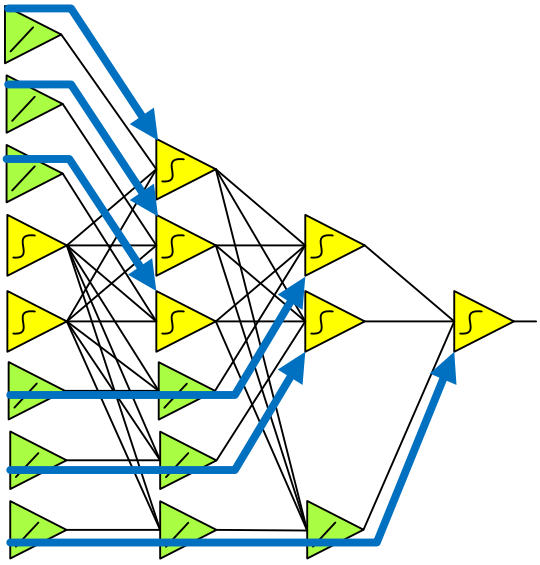
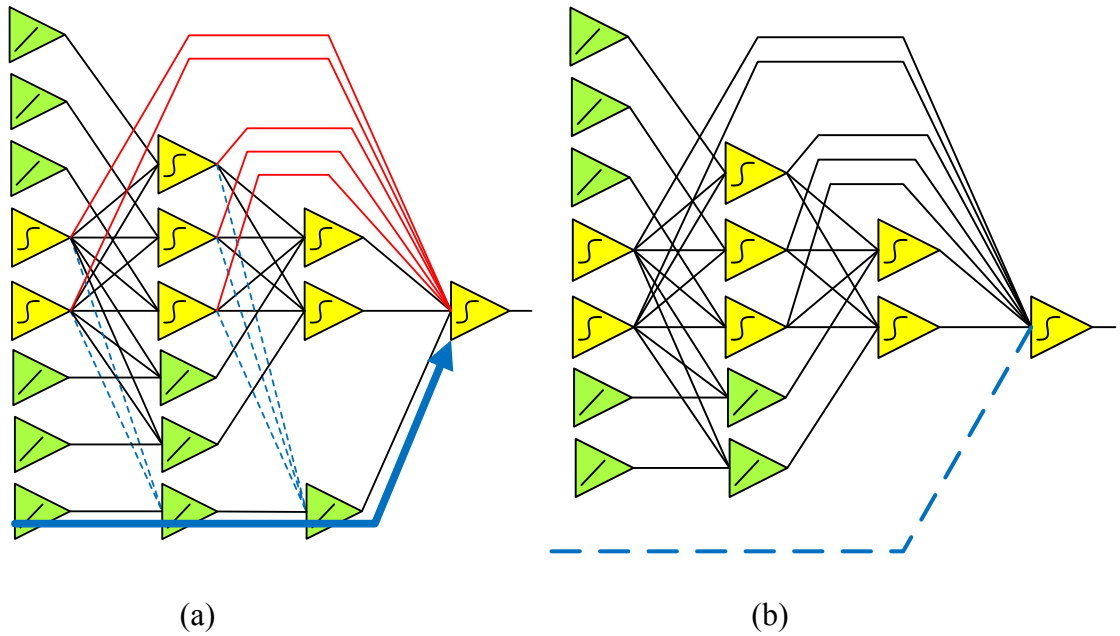


Figure 4.28: X-2(6)-3(3)-2(1)-1 DNN conversion Step 1

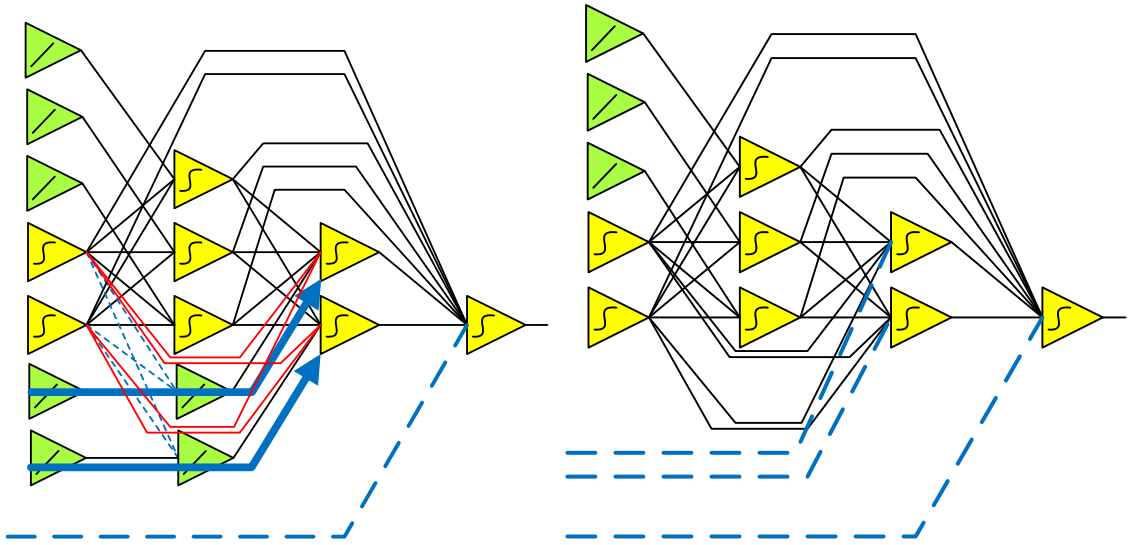
Note that during step 1 shown in Figure 4.28, we identified six linear neuron series. There is one series with 3 linear neurons, two series with 2 linear neurons each, and three series with a single linear neuron each. In Figure 4.29a, we perform steps 2 and 3 and in Figure 4.29b, we perform step 4 on the largest linear neuron series. The five outside inputs to the linear neuron series are identified by dotted blue lines. These connections are removed and replaced with the three bridged connections which are red. The X bridged input connections are shown as one single, thick dotted blue line.



(a) (b)

Figure 4.29: DNN conversion Steps 2 and 3 (a) and 4 (b)

After finishing the first iteration of conversion, we are ready to repeat steps 2 through 4 on the remaining linear neuron series. Since the next largest series is two linear neurons and there are two of these series, we can work on both in parallel. Refer to

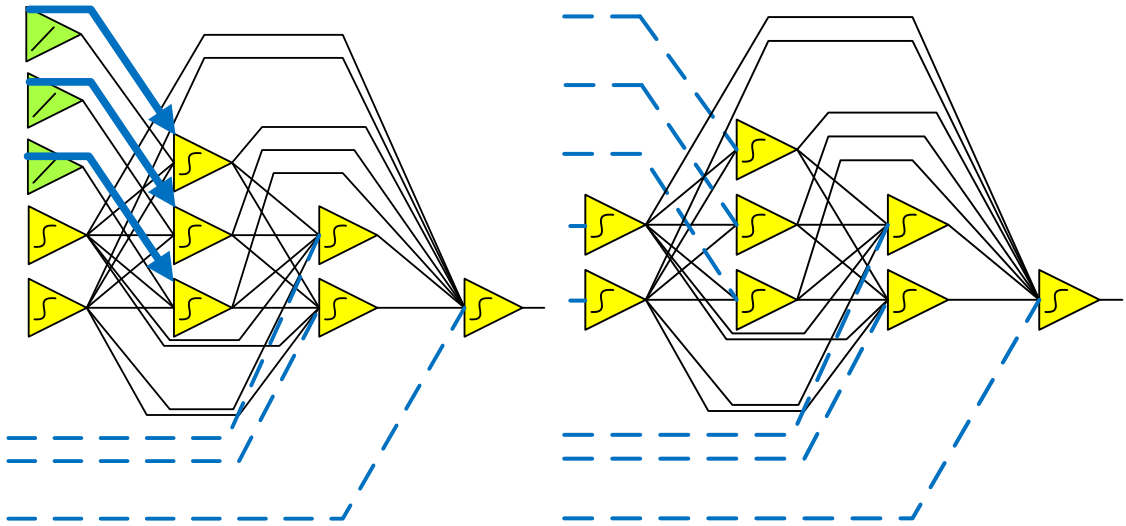


(a) (b)

Figure 4.30: DNN conversion Steps 2 and 3 (a) and 4 (b) repeated

Figure 4.28 to identify the two series that we will work on. Figure 4.30a will show steps 2 and 3 on these series. Figure 4.30b will show step 4 on the same series.

After finishing the second iteration of conversion, we are ready to repeat steps 2 through 4 on the final three linear neuron series. As the remaining three linear neuron series are all a single neuron, we can work on them in parallel. Refer to Figure 4.28 to identify the three series that we will work on. Figure 4.31a will show steps 2. Step 3 is not needed as there are no inputs from outside neurons. Figure 4.31b will show step 4 on the same set of series. The conversion from DNN to BMLP is now complete for this network. This particular network took three iterations of the conversion process to complete.



(a) (b)

Figure 4.31: DNN conversion Steps 2 and 3 (a) and 4 (b) repeated

4.3.1.3 DNN to BMLP Architecture Conversion Summary

This section has focused on the DNN to BMLP architecture conversion. As we saw, the number of layers in the network and the number of non-linear neurons in each layer did not change. It was only necessary to remove all linear neurons from the DNN network hidden layers and replace them with the appropriate bridged connections. With an understanding of the architecture conversion, we must now gain an understanding of the weight conversion.

4.3.2 DNN to BMLP Weight Conversion

With an understanding of the DNN to BMLP architecture conversion, we are now prepared to discuss the weight conversion. This weight conversion follows these rules:

1) All weights to non-linear neurons in the first hidden layer remain the same.
2) All weights from the outputs of non-linear neurons to the output neuron or the inputs of non- linear neurons in the next hidden layer remain the same.

3) All weights from the outputs of non-linear neurons in any hidden layer to the inputs of linear neurons in the next hidden layer are multiplied by all subsequent linear neuron output weights through to the destination neuron. The result of this operation provides the new weight for the bridged connection created in the architecture conversion.
4) All input weights that go to linear neurons in the first hidden layer are multiplied by all subsequent linear neuron output weights through to the destination neuron. The result of this operation provides the new weights for the bridged connections created in the architecture conversion.

The last two rules may be somewhat confusing, so it is easiest to illustrate the application of the rules on networks that we have looked at previously in this chapter. Let's look at the network conversion in Figures 4.19 and 4.20. We will use these same networks to illustrate the weight conversion. To simplify the network diagrams for illustrative purposes, unique variable weights will be assigned so that it is easy to see how the weights are converted. Figure 4.32 shows the weight conversion for the network in Figure 4.19. Figure 4.33 shows the weight conversion for the network in Figure 4.20. In each of these three figures, arrows will show how the weight conversions are performed. Each arrow will be numbered to show which step of the conversion process is being performed on the specified weights. When a weight conversion requires an initial weight to be multiplied by one or more factors, the dot operator will be used to signify the multiplication operation. Dotted lines with an arrow will show the contribution of the additional factors to the initial weight. The converted weight will be displayed as a product. For example, "a" times "g" would be displayed as a·g.

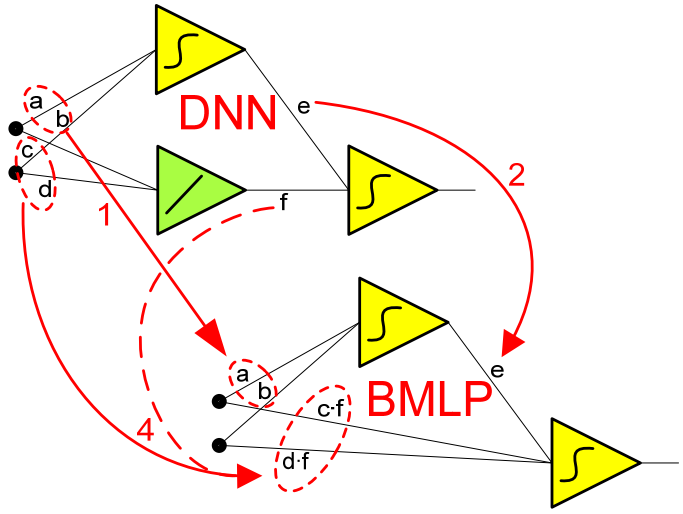


Figure 4.32: Weight Conversion for Network in Figure 4.19

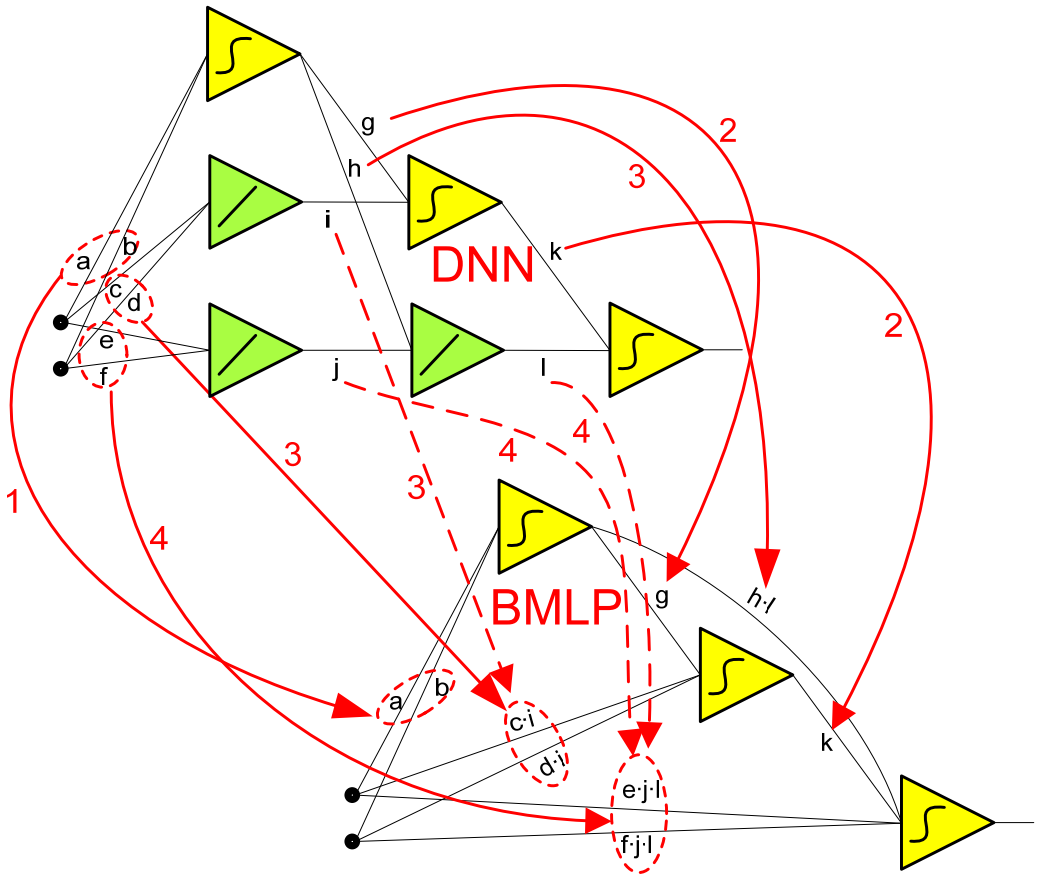


Figure 4.33: Weight Conversion for Network in Figure 4.20

As can be seen in the examples illustrated in Figures 4.32 and 4.33, the weight conversion during the DNN to BMLP conversion is more complex than the BMLP to DNN weight conversion. With this conversion understood, there is now a clear path for conversion between the DNN network architectures and the BMLP network architecture.

4.3.3 DNN to BMLP Conversion Examples

The process for converting a DNN network to an equivalent BMLP network is more complex than the previous conversion, but is accomplished through a systematic process. This section will focus on an overview of the conversion tests performed as part of this research and the implications of these tests.

For the DNN to BMLP conversion tests, I used the same input data sets as were used in the BMLP to DNN conversion. Only, in all cases, the DNN network was constructed first, trained, and then converted to the equivalent BMLP architecture.

4.3.3.1 DNN to BMLP Using Simple 3-D Surface Benchmark

The first conversion test was performed using the Simple 3-D Surface benchmark. A DNN network was built with two architectures: BMLP 2=3=2=2=1 equivalent and BMLP 2=2=3=2=1 equivalent. In both cases, the DNN network was trained to SSE ≤ 0.01. Once the training was completed, each network was converted to the BMLP architecture using the conversion process outlined earlier in this chapter. The BMLP equivalent architectures gave identical output results with no variation.

Figure 4.34 shows the trained 2-3(5)-2(3)-2(1)-1 DNN network and Figure 4.35 shows the BMLP equivalent. It is important to note that both the DNN and BMLP architectures had different weights than the architectures in Section 4.3.2. This is due to the fact that the DNN network was trained to the benchmark and then converted to a

BMLP. Since the origin network was different, the weights are different. Note that the BMLP equivalent architecture gave identical output results with no variation.

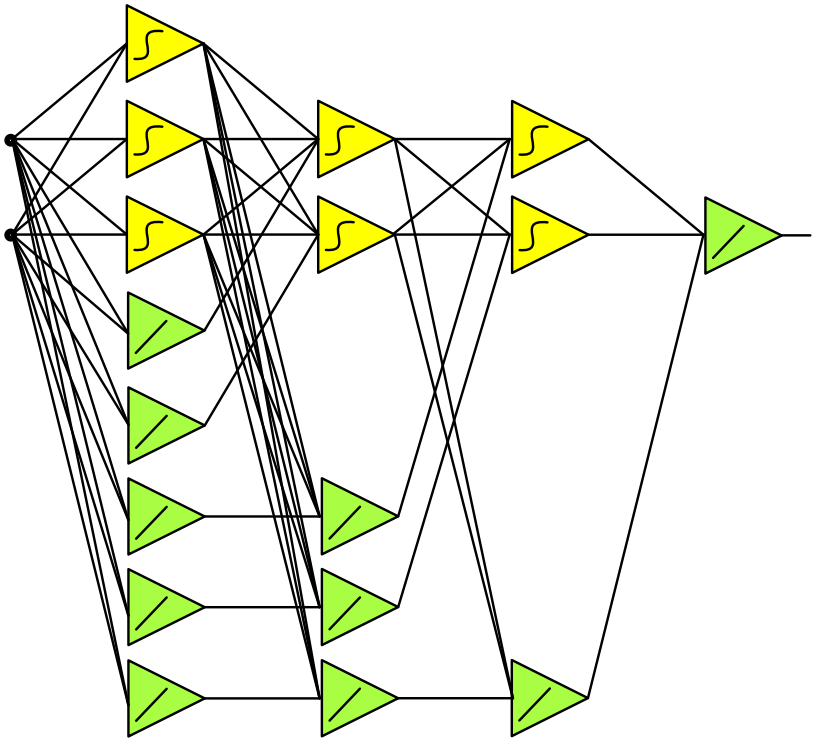


Figure 4.34: 2-3(5)-2(3)-2(1)-1 DNN network for the Simple 3-D Surface

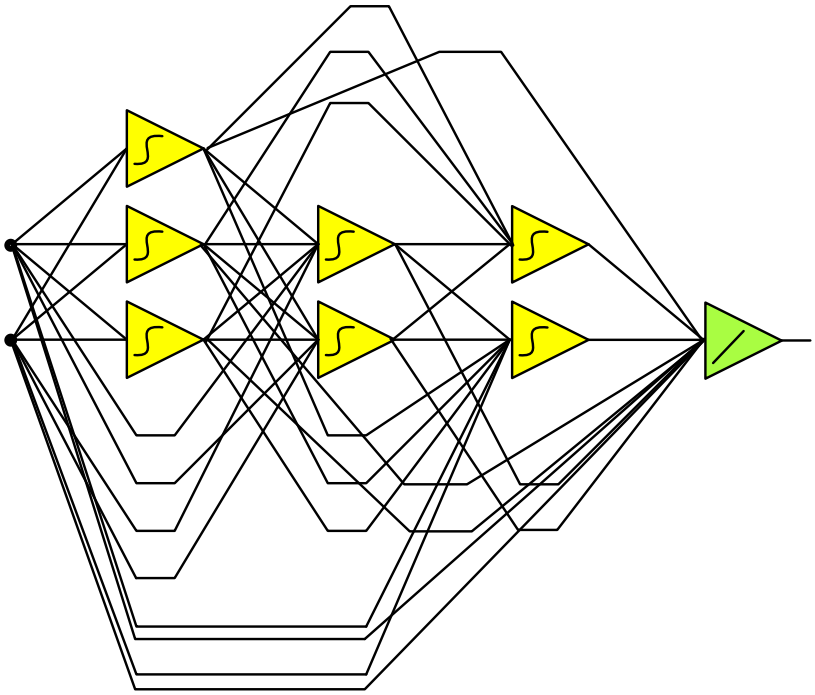


Figure 4.35: BMLP equivalent network for the network in Figure 4.34

4.3.3.2 DNN to BMLP Using 3-D Surface Benchmark

The second conversion test was performed using the 3-D Surface benchmark. This time, a DNN network was built with the BMLP 2=3=2=1 equivalent architecture. The DNN network was trained to SSE $\leq$ 0.01. Once the training was completed, the network was converted to the BMLP architecture using the standard conversion process. The BMLP equivalent architecture gave identical output results with no variation.

Figure 4.36 shows the DNN network and Figure 4.37 shows the BMLP equivalent network. Once again, the DNN and BMLP architectures had different weights than the architectures in Section 4.3.2.

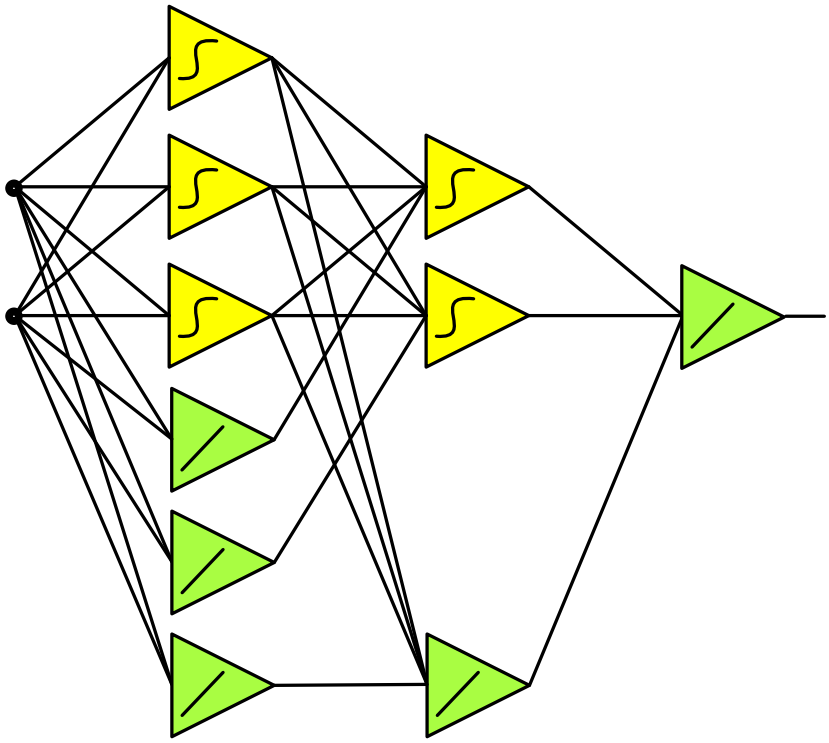


Figure 4.36: 2-3(3)-2(1)-1 DNN network for the 3-D Surface

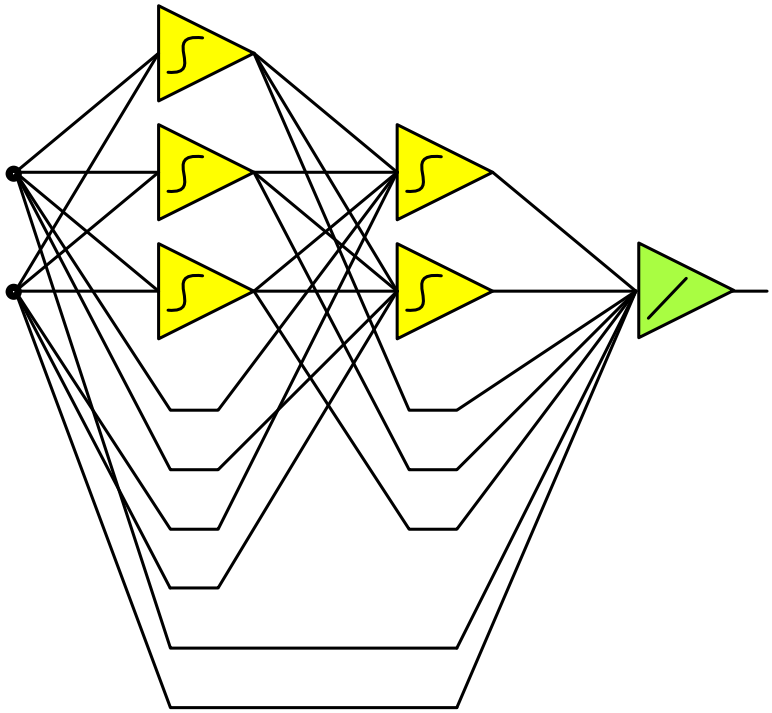


Figure 4.37: BMLP equivalent network for the network in Figure 4.36

4.3.3.3 DNN to BMLP Using Parity-N Benchmark

The third conversion test was performed using the Parity-N data set. The DNN network was built for Parity-11 using a BMLP 11=2=2=1 equivalent architecture. This network was then trained to SSE ≤ 0.01. Once training was completed, the network was converted to the BMLP architecture using the standard two-step conversion process. The BMLP equivalent architectures gave identical output results with no variation.

Figure 4.38 shows the DNN network and Figure 4.39 shows the BMLP equivalent network. Note that the BMLP equivalent architecture gave identical output results with no variation. As with the previous two tests, the DNN and BMLP architectures had different weights than the architectures in Section 4.3.2.

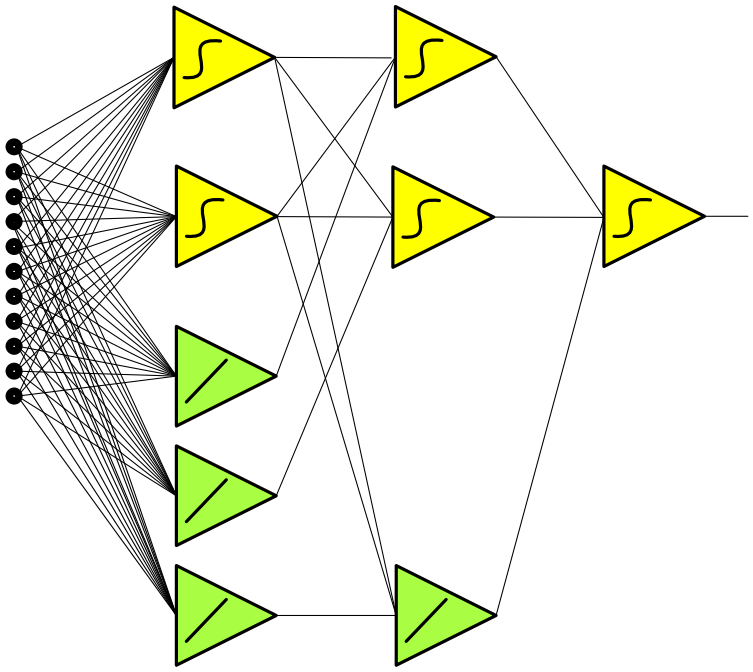


Figure 4.38: 11-2(3)-2(1)-1 DNN network for Parity-11

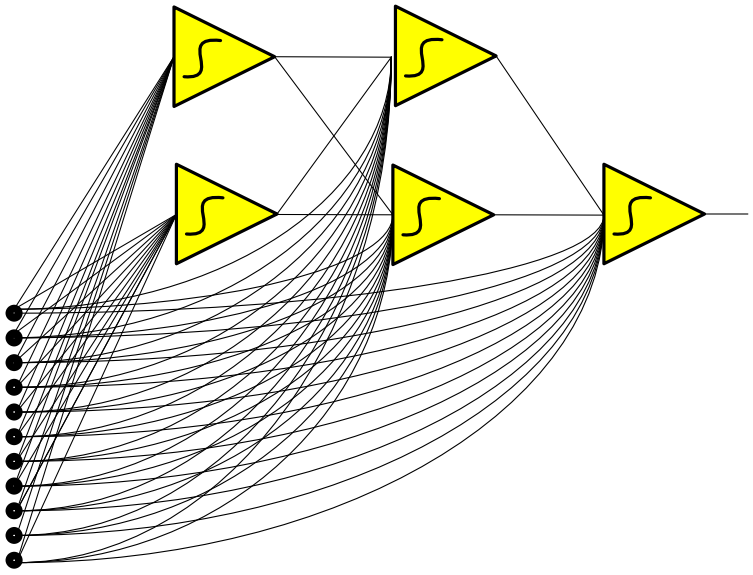


Figure 4.39: BMLP equivalent network for the network in Figure 4.38

4.3.3.4 DNN to BMLP Conversion Summary

The results in this section are significant because we have shown with multiple data sets and DNN architectures that we can train a DNN network to a given data set and then convert that trained network to an equivalent BMLP architecture.

It has now been demonstrated that we are able to convert back and forth from BMLP and DNN. This gives us many advantages, some of which will be discussed later.

4.4 DNN to MLP Conversion Process

This section will focus on the conversion from the DNN architecture to an equivalent MLP architecture. The DNN to MLP conversion is not nearly as straight forward as the other conversions that we have looked at thus far. And, in some circumstances, the conversion may not be successful. As with previous conversions, a specific process is employed to convert from the DNN architecture to the MLP

architecture. This conversion has two different parts: 1) The architecture conversion; and 2) The weight conversion. We will look at each part of the conversion separately.

4.4.1 DNN to MLP Architecture Conversion

The architectural conversion from DNN to MLP is actually very simple. Once the DNN network is trained, it can be converted to the MLP architecture by simply replacing all linear neurons in the DNN network with bipolar neurons. We will color these bipolar neurons the same color as the linear neurons to remind us that these neurons will have weight adjustments that help them to operate in their linear region.

4.4.1.1 DNN to MLP Examples

As mentioned above, the architectural conversion from DNN to MLP is very simple. No connections are changed, only the linear neurons are replaced with bipolar neurons. Let's take a look at a few examples. Figure 4.40 and Figure 4.41 show examples of the DNN to MLP architectural conversion. Notice that the networks look identical with the exception of the linear neurons being changed to bipolar neurons.

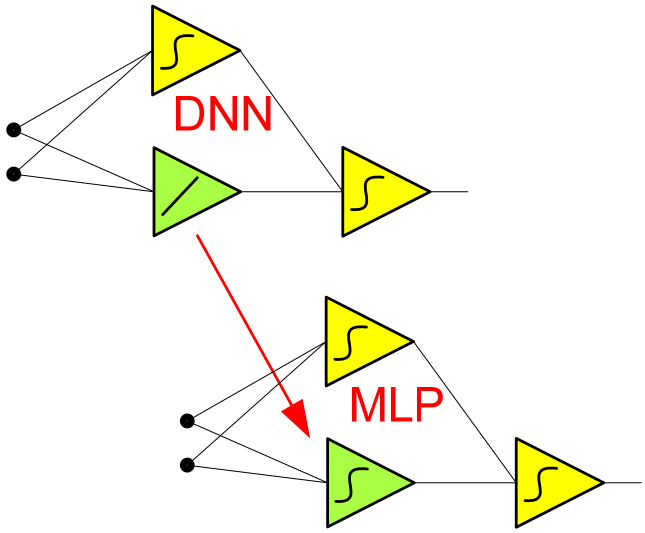


Figure 4.40: 2-1(1)-1 DNN to 2-2-1 MLP conversion

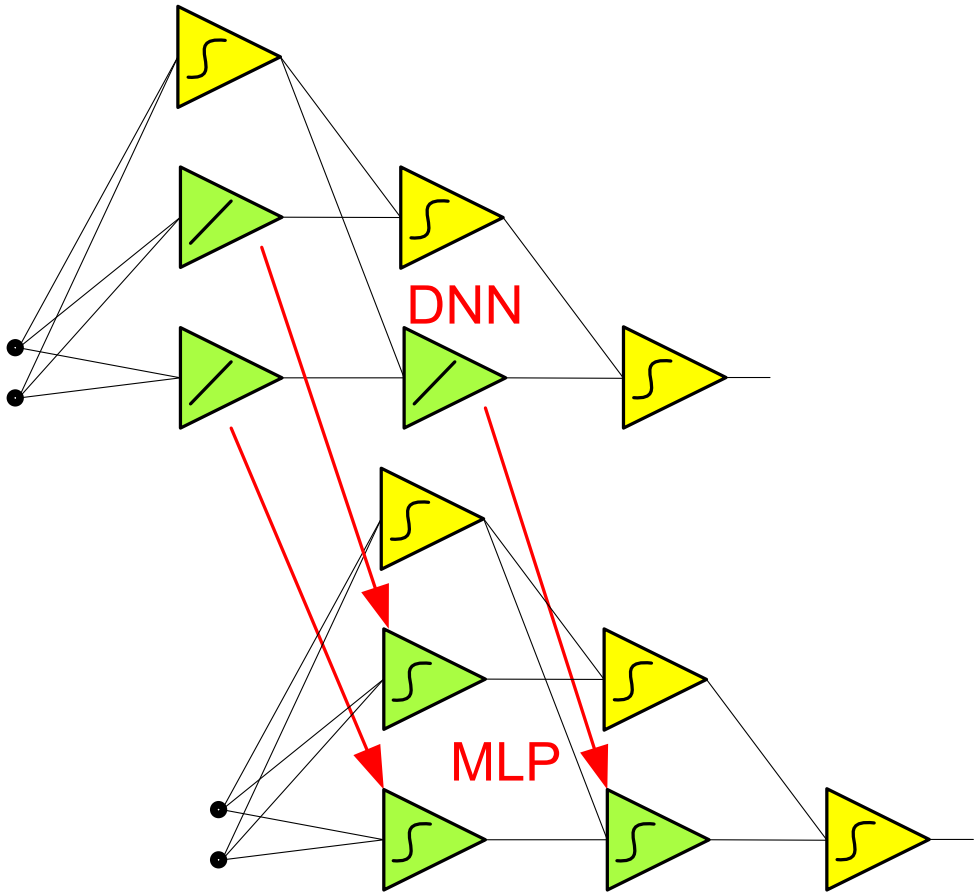


Figure 4.41: 2-1(2)-1(1)-1 DNN to 2-3-2-1 MLP conversion

4.4.1.2 DNN to MLP Architecture Conversion Summary

This brief section has focused on the DNN to MLP architecture conversion. As we saw, the MLP architecture looks identical to the DNN architecture except for the linear neurons being changed to bipolar neurons. Remember that the color of these newly placed bipolar neurons is the same as the linear neurons to remind us that these neurons will have weight adjustments that help them to operate in their linear region. With an understanding of the architecture conversion, the remainder of this section will focus on the weight conversion which is much more complex.

4.4.2 DNN to MLP Weight Conversion

With an understanding of the DNN to MLP architecture conversion, we are now prepared to discuss the weight conversion. After a significant amount of testing, it was determined that all inputs into linear acting bipolar neurons had to be reduced by a factor $\geq 10^6$ to yield identical results to the DNN network. As this is not realistically feasible, an alternate solution was developed for determining the weights of the MLP network. This weight conversion follows these rules:

1) All weights to non-linear neurons in the first hidden layer remain the same.
2) All weights from the outputs of non-linear neurons to the output neuron or the inputs of non- linear neurons in the next hidden layer remain the same.
3) All weights to the inputs of linear neurons in the first hidden layer are divided by the given factor (generally 100).
4) All weights from non-linear neurons in the hidden layers to inputs of linear neurons in the next hidden layer are divided by the given factor.
5) Weights for connections between one linear neuron's output to the input of a linear neuron in the next hidden layer remains unchanged.
6) Weights for connections between the one linear neuron's output to the input of a non-linear neuron in the next hidden layer or an output neuron are multiplied by the given factor (i.e. x·100 if the factor was 100 and the weight was "x").

This weight conversion yields a MLP network that is relatively close to a final solution, but has a SSE >> 0.01. The output function for the bipolar neuron is:

$$f_b(net) = \tanh(1 \times net) + 0.01 \times net \qquad (22)$$

A graph of this activation function can be seen in Figure 4.42. Due to the nature of the

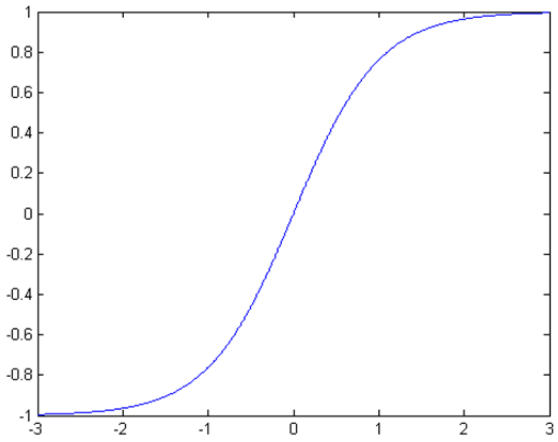


Figure 4.42: Graph of bipolar activation function

tangent hyperbolic function, its linear region is very small. Therefore, we need small net values (typically ≤ 0.1) in order to get a nearly linear output. Testing showed that dividing inputs to the neurons by a value ≥ 100 was sufficient. Outputs could then be multiplied by the same factor where appropriate.

Performance of these six steps will be demonstrated by revisiting the networks in Figure 4.40 and Figure 4.41. Once again, we will use unique variables for the weights so that we can see how the conversion takes place. Figure 4.43 and Figure 4.44 show the weight conversions for the networks in Figure 4.40 and Figure 4.41, respectively. These figures will not have arrows showing weight movements as the weights will remain in the same physical location, but will either be left alone, divided by a factor, or multiplied by a factor. For simplicity, when several factors are multiplied together, only the product will be shown. For example, if our conversion factor is 100 and we are performing step 6 on variable "x", the converted weight will be displayed as "x·100" rather than "x·1·100."

Once the weight conversion is complete, the MLP network is ready for final training iterations with the NNT software [34]. With the starting weights obtained by

using steps 1-6 outlined above, experimentation has shown that it usually takes <50 training iterations with the Neuron-by-Neuron algorithm [6] [7].

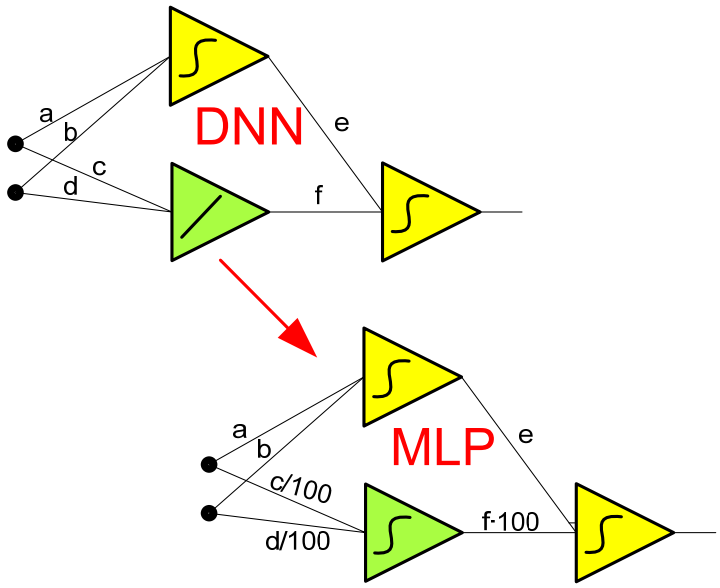


Figure 4.43: 2-1(1)-1 DNN to 2-2-1 MLP weight conversion

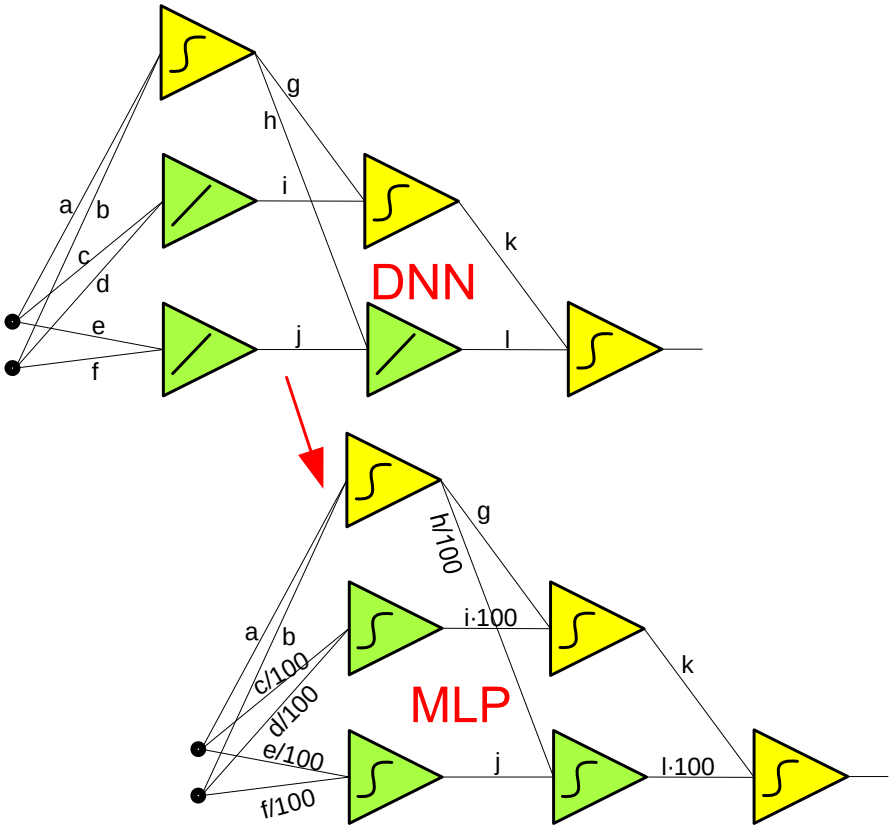


Figure 4.44: 2-1(2)-1(1)-1 DNN to 2-3-2-1 MLP weight conversion

To illustrate the entire process of weight conversion, we will re-examine the network in Figure 4.35. For this example, the DNN network in Figure 4.45 will be trained to SSE ≤ 0.01 and it will not be converted to the standardized form. This means that the linear neurons will have bias weights and their output weights will not be equal to 1. Please note that the inputs and neurons in Figure 4.45 are given sequential node numbers starting with the inputs. Therefore, the inputs are nodes 1 and 2 and the neurons are numbered nodes 3 through 19 going top to bottom and left to right. These node numbers will be used when we look at the topology map for this network.

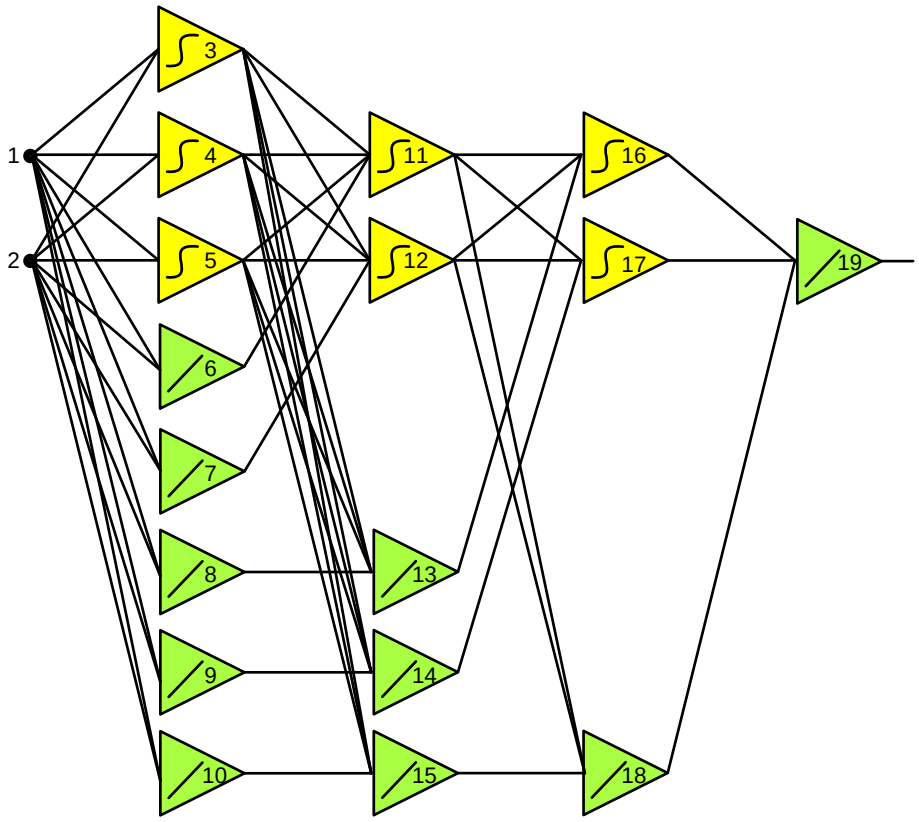


Figure 4.45: 2-3(5)-2(3)-2(1)-1 DNN network for Simple 3-D Surface Problem

The topology map and weights in Tables 4.4 and 4.5 warrant further explanation. Using the network in Figure 4.45 and Table 4.4 for explanation, let us first address the topology map. Each neuron or input in Figure 4.45 is given a node number as explained previously. The network in Figure 4.45 has two inputs and 17 neurons. In the topology portion of Table 4.4, each neuron is listed sequentially in the left-hand column with the lowest numbered neuron at the top and the highest numbered neuron at the bottom. Along with the neuron number, you will also find its type: unipolar, bipolar, or linear. In the columns to the right, you see listed which nodes provide inputs to that neuron. Please remember that node numbers can be either inputs or neuron outputs.

Topology				
Neuron	**Nodes connected to neuron**			
bipolar 3	1	2		
bipolar 4	1	2		
bipolar 5	1	2		
linear 6	1	2		
linear 7	1	2		
linear 8	1	2		
linear 9	1	2		
linear 10	1	2		
bipolar 11	3	4	5	6
bipolar 12	3	4	5	7
linear 13	3	4	5	8
linear 14	3	4	5	9
linear 15	3	4	5	10
bipolar 16	11	12	13	
bipolar 17	11	12	14	
linear 18	11	12	15	
linear 19	16	17	18	

Table 4.4: Topology for the network in Figure 4.45

Neuron Bias	**Weights from connection to neuron**			
3.08025801	0.6126632	-2.39058		
0.82199443	-1.862036	-0.705177		
1.53836839	-0.566183	-0.636743		
-0.9956325	-0.536091	1.480373		
-0.6684818	-0.595857	-1.10077		
-0.9683538	-0.019494	-1.019197		
-1.5016464	0.7368931	0.5822719		
-0.6546572	-1.557863	-2.056673		
1.09760861	-0.296337	-0.699554	-1.127874	-1.416979
0.91290076	-0.119499	0.7418785	-0.154638	-0.523773
0.52186636	0.0234138	-0.988214	0.4395666	-0.842445
-0.3563915	1.6641668	-0.575217	0.4944395	-1.298709
1.34918296	-0.808509	0.324148	-3.253097	0.2072503
-0.3881175	0.7695723	-0.739663	-0.6214	
-0.3561072	0.0590711	0.1192312	1.8024681	
0.35572164	-2.245134	-0.544116	-3.168171	
-0.3251274	0.2428015	3.4537729	-3.794651	

Table 4.5: Weights for the network in Figure 4.45

To familiarize ourselves with the topology map and weight table, let's examine Table 4.4 and Table 4.5. By looking at Table 4.4, I can see that node 3 is a bipolar neuron and is listed as "bipolar 3" in the topology map. I can now look at the correlating cell in Table 4.5 to find that the bias weight for neuron 3 is 3.08025801. Similarly, I can look at node 15 and see that it is a linear neuron listed as "linear 15" in topology map in Table 4.4. In this same topology map, I can see that neuron 15 receives inputs from nodes 3, 4, 5, and 10. Each of these connections going to the input of neuron 15 has a weight associated with it. If I wanted to know what the input weight coming from node 10 into neuron 15, I would locate the corresponding cell in the weight table in Table 4.5 and see that this weight is 0.2072503. While this may seem tedious at first, it gets much easier with a little practice. Note that it is important to understand the topology map and the weight table in order to understand the weight conversion process about to be described.

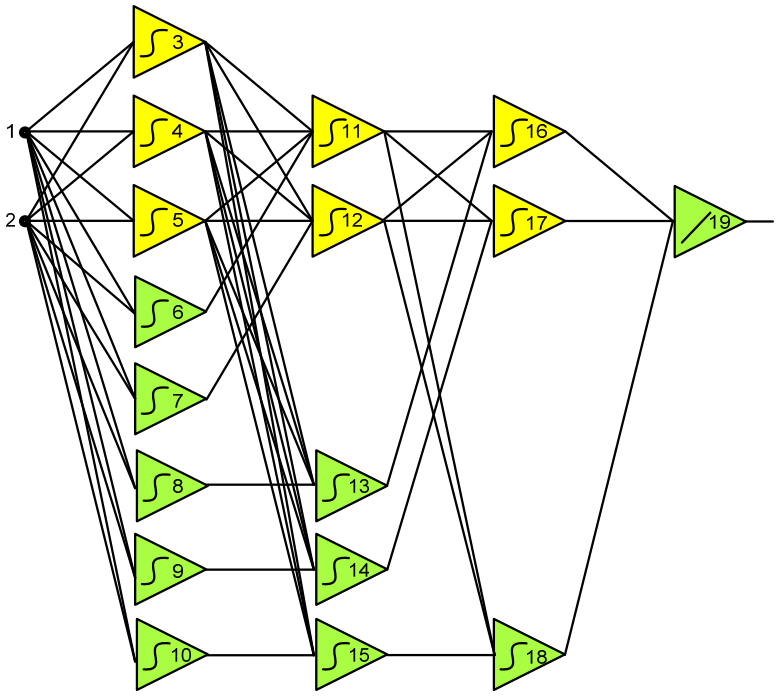


Figure 4.46: 2-8-5-3-1 MLP network for the network in Figure 4.45

Topology				
Neuron	**Nodes connected to neuron**			
bipolar 3	1	2		
bipolar 4	1	2		
bipolar 5	1	2		
bipolar 6	1	2		
bipolar 7	1	2		
bipolar 8	1	2		
bipolar 9	1	2		
bipolar 10	1	2		
bipolar 11	3	4	5	6
bipolar 12	3	4	5	7
bipolar 13	3	4	5	8
bipolar 14	3	4	5	9
bipolar 15	3	4	5	10
bipolar 16	11	12	13	
bipolar 17	11	12	14	
bipolar 18	11	12	15	
linear 19	16	17	18	

Table 4.6: Topology for the network in Figure 4.46

Neuron Bias	Weights from connection to neuron			
3.08025801	0.6126632	-2.39058		
0.82199443	-1.862036	-0.705177		
1.53836839	-0.566183	-0.636743		/100
-0.009956325	-0.005361	0.0148037		
-0.006684818	-0.005959	-0.011008		x100
-0.009683538	-0.000195	-0.010192		
-0.015016464	0.0073689	0.0058227		x1
-0.006546572	-0.015579	-0.020567		
1.09760861	-0.296337	-0.699554	-1.127874	-141.6979
0.91290076	-0.119499	0.7418785	-0.154638	-52.37735
0.52186636	0.0234138	-0.988214	0.4395666	-0.842445
-0.35639153	1.6641668	-0.575217	0.4944395	-1.298709
1.34918296	-0.808509	0.324148	-3.253097	0.2072503
-0.38811753	0.7695723	-0.739663	-62.14001	
-0.35610724	0.0590711	0.1192312	180.24681	
0.35572164	-2.245134	-0.544116	-3.168171	
-0.32512742	0.2428015	3.4537729	-379.4651	

Table 4.7: Initial conversion weights for the network in Figure 4.46

Neuron Bias	Weights from connection to neuron			
4.34861146	0.4308709	-2.391354		
0.56880824	-1.993203	-1.137935		
2.43241213	-1.521003	0.1752029		
-3.89975667	-0.559942	2.0931741		
-0.6707688	-0.440732	-1.162265		
-0.96690361	-0.052703	-0.992139		
-2.13978938	0.9667528	-0.419782		
0.23989112	-1.639442	-0.111844		
1.61316693	0.9173422	-0.571508	-0.237162	-4.293259
0.77067376	-0.602085	0.8525756	0.1313983	-0.376507
0.52020742	0.044344	-0.983806	0.4045055	-0.831085
0.46269498	1.2281474	-1.203796	1.2096349	-2.289386
0.60310117	-5.1284	0.6672689	-3.62143	1.5179751
-0.34602684	0.8856006	-0.722258	-0.560782	
0.19584711	1.7527249	0.2180854	2.9040219	
1.63388368	-3.019219	-0.557369	-4.838114	
0.18648368	0.3942708	4.320831	-5.077064	

Table 4.8: Final weights for the network in Figure 4.46

In this example, the DNN weights were converted to MLP weights using a factor of 100. In some cases, you may need to increase the factor to obtain a final result. As seen in Table 4.7, all weights highlighted in orange were divided by the factor. All weights highlighted in green were multiplied by 100. Finally, weights highlighted in blue were multiplied by 1, or left unchanged. This conversion gave us the initial training weights for the MLP network. After 15 iterations of training using the Neuron-by-Neuron algorithm, an SSE = 0.0098 was achieved and the resulting weights are found in Table 4.8. Although the results of the MLP are not identical to the DNN network for any given input in the data set, the result is acceptable because we achieved SSE $\leq$ 0.01.

4.4.3 DNN to MLP Conversion

Converting from a DNN architecture to an equivalent MLP architecture proved to

be much more difficult than other conversion. Since the MLP architecture does not have any bridged connections across layers, it is necessary for the MLP architecture to include at least one neuron in each hidden layer that is functioning in its linear region.

Let us look at two simple examples of this conversion. For the purposes of these examples, only the DNN to MLP conversion will be demonstrated.

4.4.3.1 DNN to MLP Using Simple 3-D Surface Benchmark

Figure 4.47 shows a 2-3(5)-2(3)-2(1)-1 DNN network trained for the Simple 3-D Surface benchmark. This network was trained to SSE=0.00950321. Figure 4.48 the MLP equivalent network obtained with the conversion process described previously. Note that the output neuron is not changed and remains a linear neuron per the conversion procedure. After the initial weight conversion, the network in Figure 4.48 received final training with the NNT software. This training was successful and yielded SSE=0.00449923 in 15 iterations.

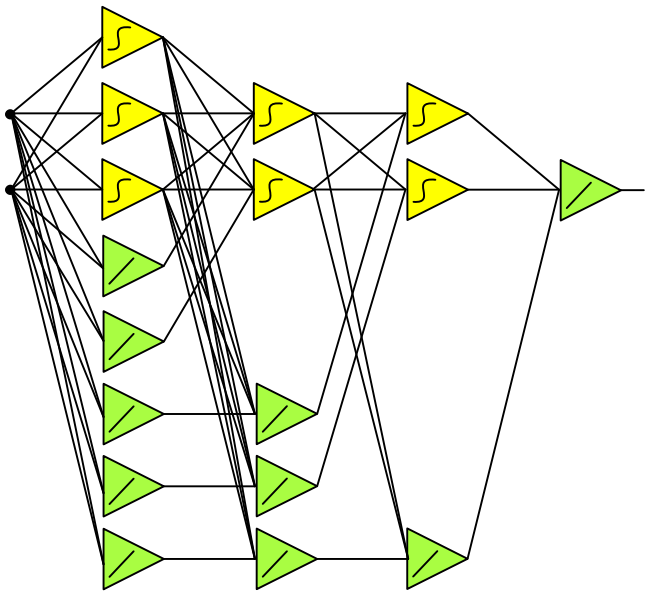


Figure 4.47: 2-3(5)-2(3)-2(1)-1 DNN network for Simple 3-D Surface Problem

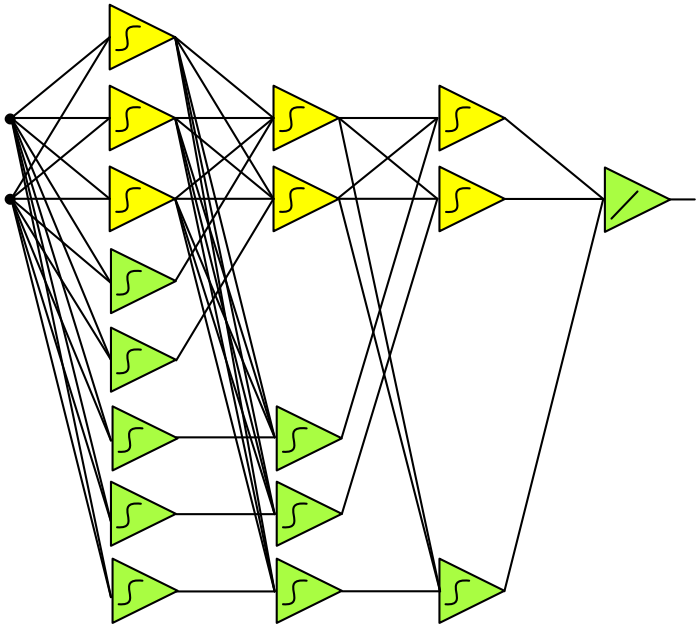


Figure 4.48: 2-8-5-3-1 MLP equivalent network for network in Figure 4.47

4.4.3.2 DNN to MLP Using 3-D Surface Benchmark

Figure 4.49 is identical to Figure 4.36 presented earlier in this chapter and is placed here for ease of comparison during this example. This figure shows a DNN 2-3(3)-2(1)-1 architecture for the 3-D Surface problem. This network was trained to SSE= 0.00968703. Figure 4.50 shows the MLP equivalent network obtained with the conversion process described above. Note that the output neuron is not changed and remains a linear neuron per the conversion procedure. After the initial weight conversion, the network in Figure 4.50 received final training with the NNT software. This training was successful and yielded SSE= 0.00972139 in 59 iterations.

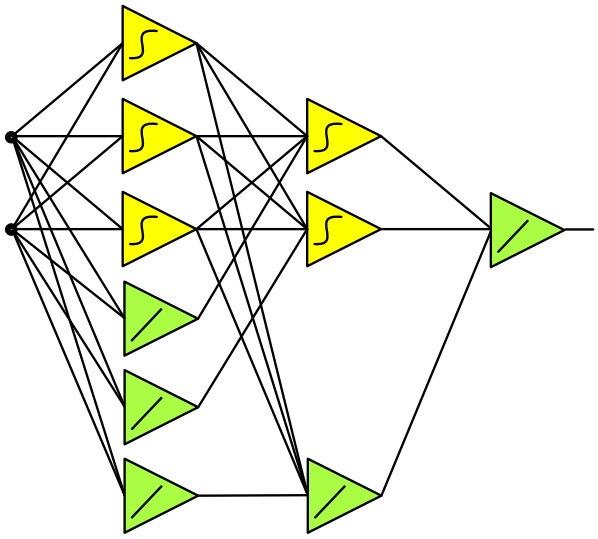


Figure 4.49: DNN 2-3(3)-2(1)-1 network for 3-D Surface Benchmark

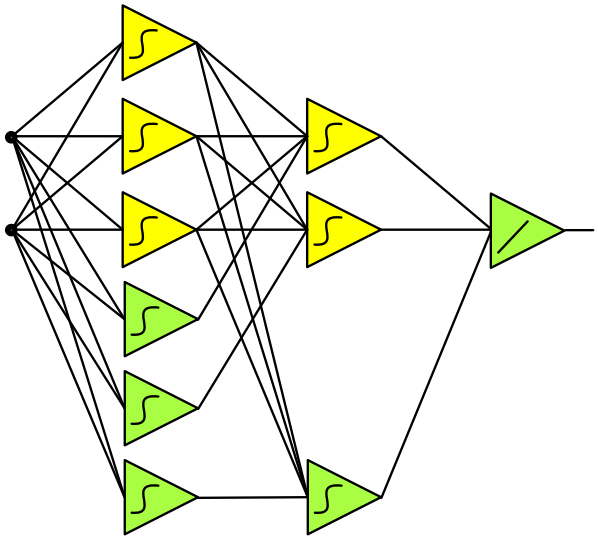


Figure 4.50: MLP equivalent network for network in Figure 4.49

4.4.3.3 DNN to MLP Conversion Summary

These results are momentous because it has been shown that a DNN network can be trained and then converted to a MLP architecture. This is important because training deep MLP networks has been very challenging and typically has a very low success rate. While it may not work in all cases, the method just described provides another option for

training challenging MLP networks. Additionally, the success rate of training deep MLP networks can be significantly increased by using the new method just described.

4.5 MLP to DNN Conversion Process

This section will focus on the conversion from the MLP architecture to an equivalent DNN architecture. The MLP to DNN conversion is slightly less complex than the DNN to MLP conversion in the previous section. As with previous conversions, a specific process is employed to convert from the MLP architecture to the DNN architecture. This conversion has two different parts: 1) The architecture conversion; and 2) The weight conversion. We will look at each part of the conversion separately.

4.5.1 MLP to DNN Architecture Conversion

The architectural conversion from MLP to DNN is actually very simple. Once the MLP network is trained, it can be converted to the DNN architecture by simply replacing all linear-acting bipolar neurons in the MLP network with linear neurons. Remember that the linear-acting bipolar neurons are the same color as the linear.

4.5.1.1 MLP to DNN Examples

As mentioned above, the architectural conversion from MLP to DNN is very simple. No connections are changed, only the linear-acting bipolar neurons are replaced with linear neurons. Let's take a look at a few examples. Figure 4.51 and Figure 4.52 show examples of the MLP to DNN architectural conversion. Notice that the networks look identical with the exception of the linear-acting bipolar neurons being changed to linear neurons.

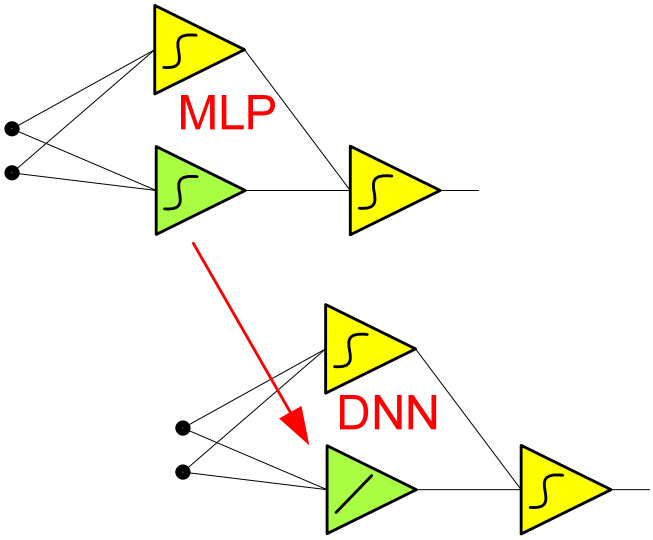


Figure 4.51: 2-2-1 MLP to 2-1(1)-1 DNN conversion

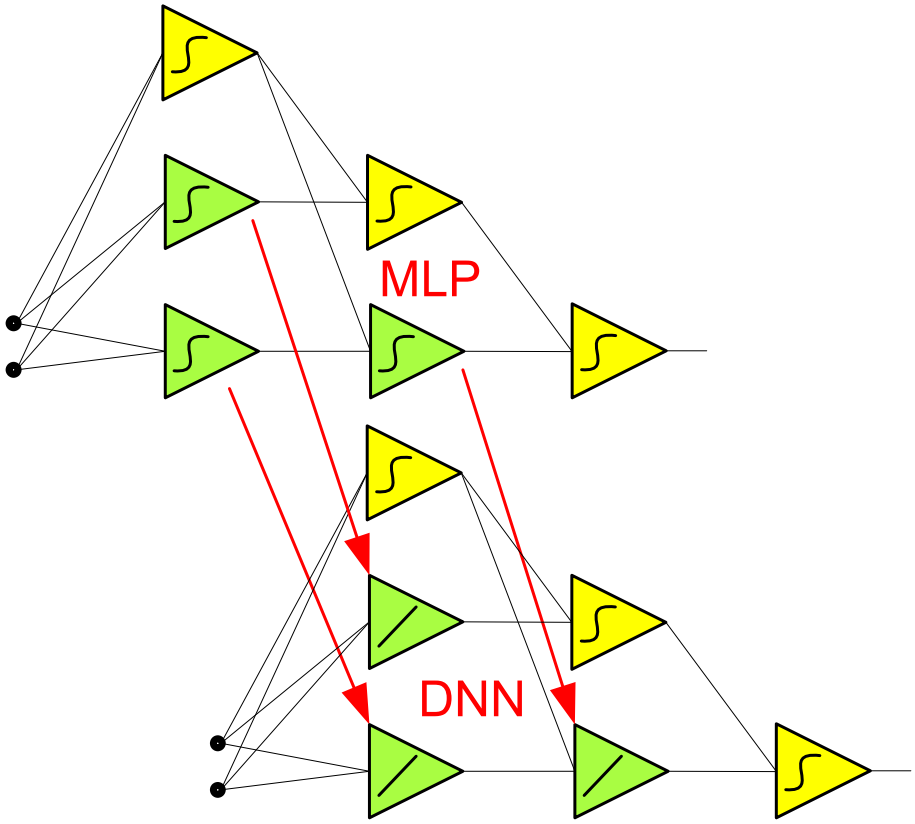


Figure 4.52: 2-3-2-1 MLP to 2-1(2)-1(1)-1 DNN conversion

4.5.1.2 MLP to DNN Architecture Conversion Summary

This brief section has focused on the MLP to DNN architecture conversion. As

we saw, the DNN architecture looks identical to the MLP architecture except for the linear-acting bipolar neurons being changed to linear neurons. Remember that the color of these linear-acting bipolar neurons is the same as the linear. With an understanding of the architecture conversion, the remainder of this chapter will focus on the weight conversion which is much more complex.

4.5.2 MLP to DNN Weight Conversion

With an understanding of the MLP to DNN architecture conversion, we are now prepared to discuss the weight conversion. This weight conversion follows these rules:

1) All weights to non-linear neurons in the first hidden layer remain the same.
2) All weights from the outputs of non-linear neurons to the output neuron or the inputs of non-linear neurons in the next hidden layer remain the same.
3) All output weights for linear-acting bipolar neurons remain the same.

This weight conversion yields a DNN network that is relatively close to a final solution, but has a SSE > 0.01.

Performance of these three steps will be demonstrated on the networks in Figure 4.53 and Figure 4.54. Once again, we will use unique variables for the weights so that we can see how the conversion takes place. As before, these figures will not have arrows showing weight movements as the weights will remain in the same physical location and will remain unchanged.

Once the weight conversion is complete, the DNN network is ready for final training iterations with the NNT software [34]. With the starting weights obtained by using steps 1-3 outlined above, experimentation has shown that it usually takes <50 training iterations with the Neuron-by-Neuron algorithm [6] [7].

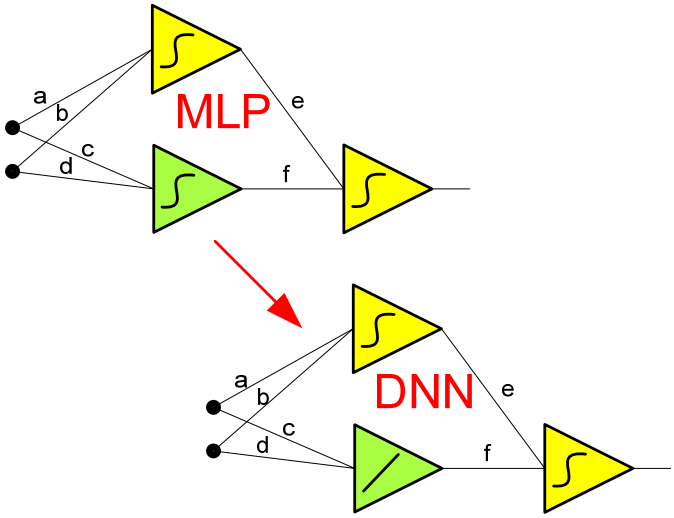


Figure 4.53: 2-2-1 MLP to 2-1(1)-1 DNN weight conversion

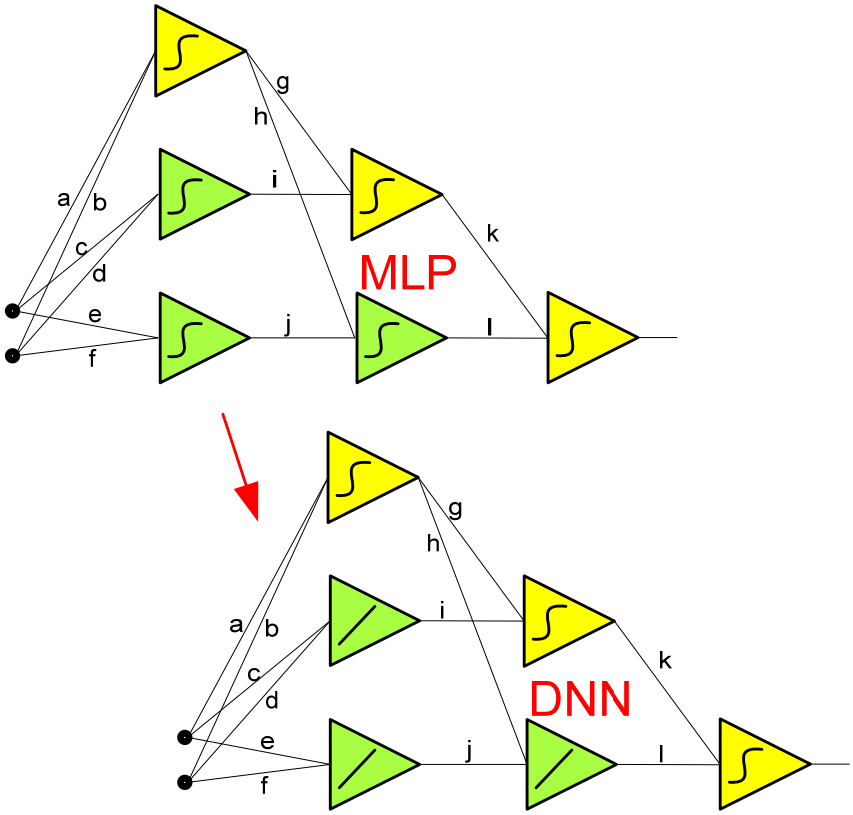


Figure 4.54: 2-3-2-1 MLP to 2-1(2)-1(1)-1 DNN weight conversion

4.5.3 MLP to DNN Conversion

Let us look at an example of this conversion. For the purposes of this example,

only the MLP to DNN conversion will be demonstrated.

To illustrate the entire process of weight conversion, we will examine the network in Figure 4.55. This network was trained for the Simple 3-D Surface benchmark. The resulting topology map can be found in Table 4.9 and the resulting weights can be found in Table 4.10.

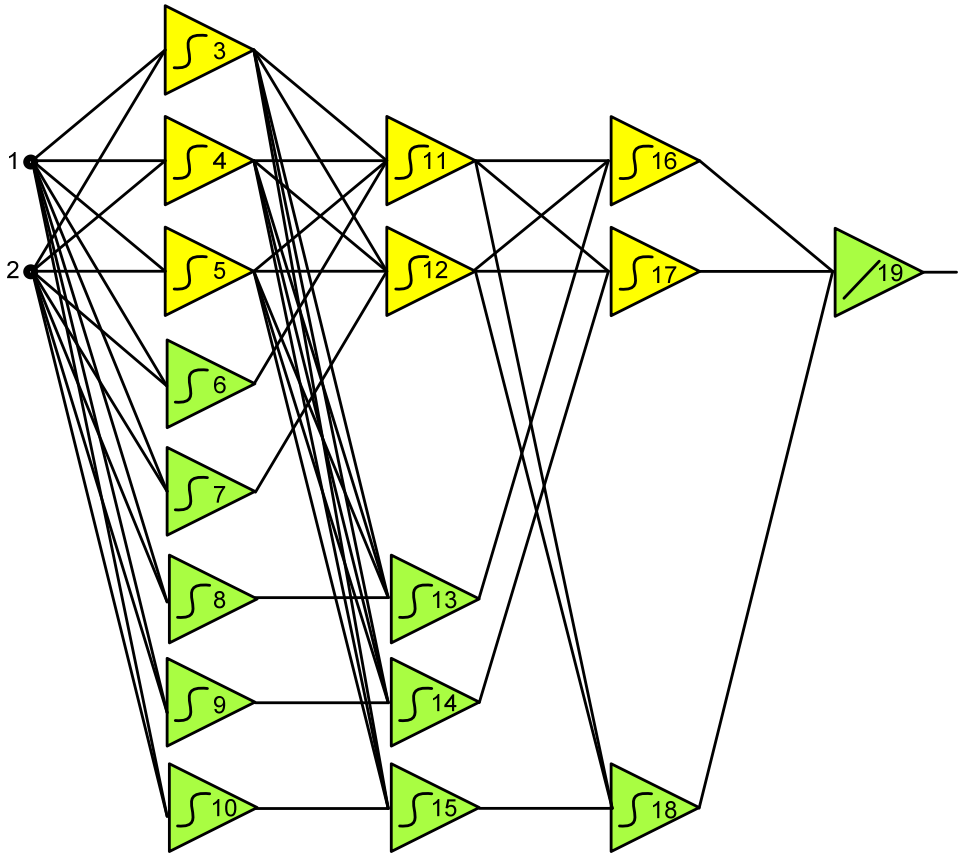


Figure 4.55: 2-8-5-3-1 MLP network for Simple 3-D Surface Problem

Topology				
Neuron	**Nodes connected to neuron**			
bipolar 3	1	2		
bipolar 4	1	2		
bipolar 5	1	2		
bipolar 6	1	2		
bipolar 7	1	2		
bipolar 8	1	2		
bipolar 9	1	2		
bipolar 10	1	2		
bipolar 11	3	4	5	6
bipolar 12	3	4	5	7
bipolar 13	3	4	5	8
bipolar 14	3	4	5	9
bipolar 15	3	4	5	10
bipolar 16	11	12	13	
bipolar 17	11	12	14	
bipolar 18	11	12	15	
linear 19	16	17	18	

Table 4.9: Topology for the network in Figure 4.55

Neuron Bias	**Weights from connection to neuron**			
2.48386872	0.3391859	-2.446145		
0.86353479	-0.800207	-0.337705		
1.51179365	-0.427785	-0.626862		
0.013593	0.0856393	0.049224		
0.2802055	0.2875856	0.0970668		
0.11637587	-0.052247	0.3852626		
0.125717	-0.075481	-0.231028		
-0.18270813	-0.247846	-0.21535		
2.5086595	-0.198985	-0.685744	-1.137666	195.99979
1.26875625	-0.11896	0.737556	-0.153597	196.00021
0.23606937	-0.329665	0.0423351	-0.003764	1.9625221
-0.14593281	-0.237823	-0.387408	-0.186387	2.0362219
0.00418045	-0.013606	-0.356509	-0.381559	1.7635307
-1.21451414	0.7665509	-0.734969	196.00011	
2.51404962	0.2812977	0.1215542	195.99918	
0.00176091	0.0031047	-0.075732	1.4589998	
12.91383034	0.1040237	4.150778	195.99519	

Table 4.10: Weights for the network in Figure 4.55

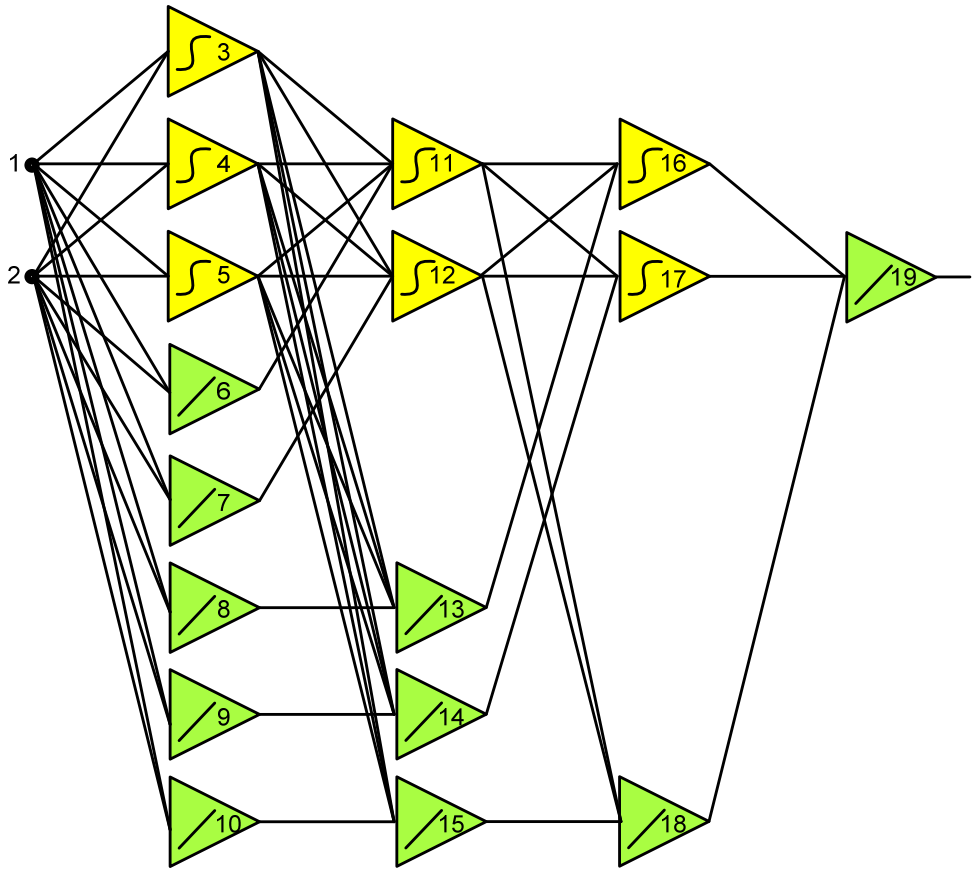


Figure 4.56: 2-3(5)-2(3)-2(1)-1 DNN network for Simple 3-D Surface Problem

Table 4.11 shows the topology map for the DNN network and Table 4.12 shows the converted weights obtained by step 3. After the initial weight conversion, the network in Figure 4.56 received final training with the NNT software.

Topology				
Neuron	**Nodes connected to neuron**			
bipolar 3	1	2		
bipolar 4	1	2		
bipolar 5	1	2		
linear 6	1	2		
linear 7	1	2		
linear 8	1	2		
linear 9	1	2		
linear 10	1	2		
bipolar 11	3	4	5	6
bipolar 12	3	4	5	7
linear 13	3	4	5	8
linear 14	3	4	5	9
linear 15	3	4	5	10
bipolar 16	11	12	13	
bipolar 17	11	12	14	
linear 18	11	12	15	
linear 19	16	17	18	

Table 4.11: Topology for the network in Figure 4.56

Neuron Bias	**Weights from connection to neuron**			
2.48386872	0.3391859	-2.446145		
0.86353479	-0.800207	-0.337705		
1.51179365	-0.427785	-0.626862		
0.013593	0.0856393	0.049224		
0.2802055	0.2875856	0.0970668		
0.11637587	-0.052247	0.3852626		
0.125717	-0.075481	-0.231028		
-0.18270813	-0.247846	-0.21535		
2.5086595	-0.198985	-0.685744	-1.13767	195.99979
1.26875625	-0.11896	0.737556	-0.15360	196.00021
0.23606937	-0.329665	0.0423351	-0.00376	1.9625221
-0.14593281	-0.237823	-0.387408	-0.18639	2.0362219
0.00418045	-0.013606	-0.356509	-0.38156	1.7635307
-1.21451414	0.7665509	-0.734969	196.00011	
2.51404962	0.2812977	0.1215542	195.99918	
0.00176091	0.0031047	-0.075732	1.45900	
12.91383034	0.1040237	4.150778	195.99519	

Table 4.12: Beginning weights for the network in Figure 4.56

Neuron Bias	Weights from connection to neuron			
4.14626735	0.2568368	-3.006063		
2.60073766	-0.10758	-0.674673		
2.51577592	-1.355735	-0.349683		
0.67166225	0.6315569	0.1414477		
0.63301109	0.2482514	-0.015002		
0.22037034	0.0502236	0.4690399		
0.68221011	0.0083547	0.3357026		
-0.20082532	-0.691805	-2.152607		
2.53580054	-0.554188	-0.712023	-1.137542	196.00541
1.28312069	-0.108948	0.7404038	-0.148668	196.00129
0.66138032	-0.201117	0.0193988	0.0754367	2.0358455
-0.71014083	0.3395287	0.020528	0.1942554	3.8971835
-2.16988419	-1.220909	2.8424076	-1.033485	0.999659
-1.19716736	0.7821705	-0.717134	196.0018	
2.48804015	0.1618998	0.0987929	196.01029	
0.00821911	-0.221949	0.4109369	3.6250751	
12.9342263	0.1981816	5.3071404	196.01165	

Table 4.13: Final weights for the network in Figure 4.56

In this example, the MLP weights required no changes. As seen in Table 4.12, all weights highlighted in yellow remained the same, giving us a starting point for training. After 65 iterations of training using the Neuron-by-Neuron algorithm, an SSE = 0.00967125 was achieved and the resulting weights are found in Table 4.13. Although the results of the DNN network are not identical to the MLP network for any given input in the data set, the result is acceptable because we achieved SSE $\leq$ 0.01.

4.6 Conversion Summary

As shown in the previous sections of this chapter, we are now able to convert between BMLP, DNN, and MLP networks. As we discovered, the DNN architecture provides the key for conversion between BMLP and MLP. If I start with a DNN network, then I can directly convert to either BMLP or MLP. If I start with a BMLP or MLP

network and I desire to convert to the other, I must first convert to the DNN architecture and then to the desired architecture.

With the ability to convert between the different NN architectures, one has many new options when it comes to NN training, especially for those hard to train networks.

Chapter 5

Comparison of Training Success Rate and Efficiency

The conversion methods described in Chapter 4 provides additional means to increase the odds of obtaining a solution to a NN problem. Normally, NN problems are solved by selecting a NN architecture, compiling a training set, and then training the NN to the desired SSE. Unfortunately, this method can often yield lower success rates.

To improve the odds of obtaining a solution, consider training multiple NN architectures in parallel for the same problem. For example, you could select a BMLP architecture and also a DNN architecture. Apply the same training set to both architectures and train them in parallel. Not only do the odds of finding a solution increase, but you very well may end up with solutions for both architectures. If only one NN yields a solution, you have the option to use that architecture or convert it to a different architecture. If both architectures yield solutions, you are able to choose the best solution. If desired, you can convert to a different architecture using the methods described in Chapter 4.

5.1 Efficiency Comparison

When researchers compare NN efficiency, they generally look at the power of the network, or in other words, how many neurons are required to solve a given problem. While this is a good comparison mechanism, it does not tell the entire story. For example, Figure 5.1 shows an efficiency comparison of MLP, BMLP, FCC, and DNN network

architectures based upon the Parity-N benchmark. If we are to simply judge our networks based upon the number of neurons required to solve a given Parity-N problem, we would likely conclude that the FCC architecture is the most efficient and the MLP architecture is the least efficient. We could also conclude that the new DNN architecture presented in this research falls somewhere in the middle close to BMLP with two hidden layers. And, this would be somewhat disappointing!

Granted, Figure 5.1 is an important piece of data when comparing NN architectures and it surely should not be ignored, however, this data only correlates the number of neurons with the network power or efficiency. To get a more full picture

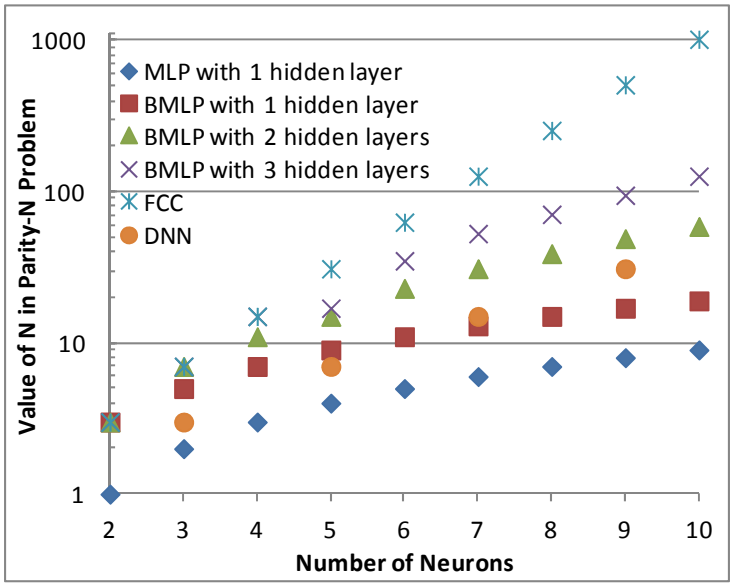


Figure 5.1: Efficiency comparison of various neural network architectures

of a network's true capabilities, we must also consider other aspects of its architecture such as the number and type of connections. Table 5.1 shows these same network architectures with an additional piece of data. In addition to showing the number of

neurons required to solve the given Parity-N problem, this table shows us the number of weights in the architecture which is also equal to the number of connections in the network.

Architecture	Parity-3		Parity-7		Parity-15		Parity-31		Parity-63	
	# of neurons	# of weights	# of neurons	# of weights	# of neurons	# of weights	# of neurons	# of weights	# of neurons	# of weights
MLP 1 hidden layer	4	16	8	64	16	256	32	1024	64	4096
BMLP 1 hidden layer	3	14	5	44	9	152	17	560	33	2144
BMLP 2 hidden layers	N/A		3	27	5	88	7	239	11	739
BMLP 3 hidden layers	N/A		N/A		4	70	6	203	8	530
FCC	2	9	3	27	4	70	5	170	6	399
DNN using 1 linear neuron per hidden layer	3	7	5	15	7	27	9	34	11	83

Table 5.1: Comparison of neural network efficiency with required weights

By more thoroughly analyzing the data presented in Table 5.1, one can easily see the drastic impact that the NN architecture has on the number of weights and connections in the network. For example, if we look at the Parity-63 column, we see that an MLP network requires 4096 connections, a BMLP network with one hidden layer requires, 2144 connection, a FCC network requires 399 connections, and the DNN network only requires 83 connections. Immediately, this gives us a much different picture than what we saw in Figure 5.1.

Rather than solely basing your judgment on either the number of neurons required or the number of connections in the architecture, an alternative method for judging

efficiency may be used. By summing the number of neurons required to solve a given problem with the number of weights or connections in that architecture, a fuller picture of network efficiency is seen. In this way, we take into account the entire solution, including any advantages or disadvantages that the architecture may have. As we know, the number of weights in an architecture has a direct effect on training times and success rates. Applying our new method to the data in Table 5.1, we can conclude that the new DNN architecture introduced in this research is truly more efficient than the others.

5.2 Experimental Training Results

Experimental results were obtained by comparing the BMLP, DNN, and MLP equivalent architectures on given benchmarks. Simulations were run using the NNT software [34] discussed previously. The user interface for the NNT can be seen in Figure 5.2 below.

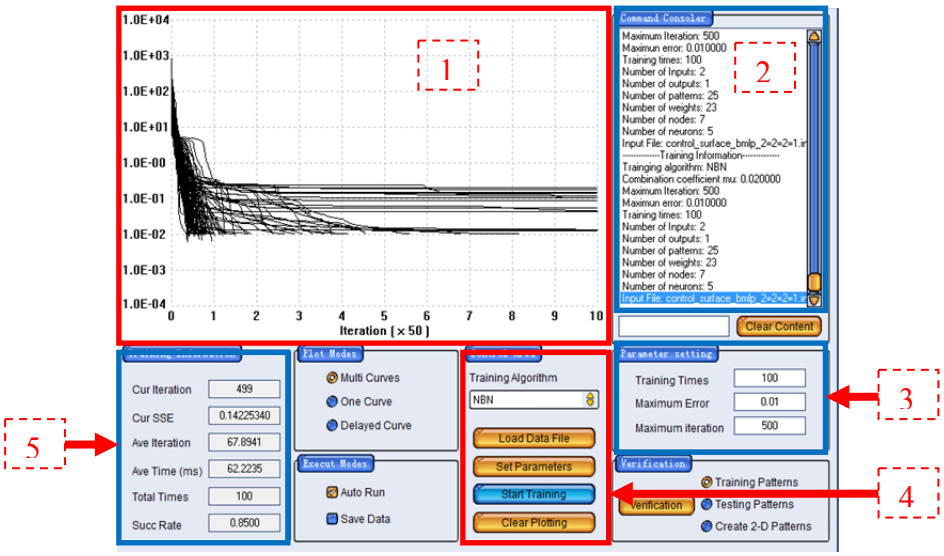


Figure 5.2: NNT User Interface

There are five main areas in the NNT user interface that are important to understand. In order to better understand the experimental results, it is important to

understand the tools used. The NNT was selected as the tool of choice because of its flexibility, statistical display, error plot display, and its breadth of available training algorithms. Each area of the NNT user interface and its functionality will be discussed separately.

The first area, labeled "1" in Figure 5.2 is the error plot. In this area, the error is plotted for each training iteration. The vertical axis displays error while the horizontal axis displays the training iteration number. The error plot seen in this figure is a result of 500 training iterations and we see all 500 training curves. Notice that some curves end when they reach the desired error, 1.0E-02, and others extend across the screen, never reaching the desired error level.

The second area, labeled "2" in Figure 5.2 is the console area. This area displays information pertaining to the current and previous training runs. This information includes the name of the network training file, topology information (i.e. number of inputs, outputs, weights, training patterns, nodes, and neurons), the desired maximum error, the maximum number of iterations allowed in each training cycle, the number of times to perform the training, and algorithm specific training parameters.

The third area, labeled "3" in Figure 5.2 is where certain algorithm independent training parameters are set. "Training Times" is the total number of times you want to train the given network. "Maximum error" is the desired maximum error for training. When the current training error is less than or equal to this value, the current training cycle ends. "Maximum iterations" sets the maximum number of training iterations that can be performed in any cycle before training ends. If the maximum number of iterations is reached before total training error reaches a value less than or equal to "Maximum

error," the current training cycle ends.

The fourth area, labeled "4" in Figure 5.2 is the control area. In the drop-down menu at the top, we are able to select the desired training algorithm. The button labeled "Load Data File" allows us to select and load our desired network topology file. This is the network that we want to train. The button labeled "Set Parameters" allows us to set algorithm specific parameters that affect how the chosen algorithm works. The button labeled "Start To Train" starts the training process. The button labeled "Clear Plotting" clears the error plots displayed in area 1. One would typically clear the plotting area in between training runs.

The fifth area, labeled "5" in Figure 5.2 displays training information from the current training runs. Remember that a training run may train a given network many times as defined by "Training Times" in area 3. Each training cycle can only perform up to a give number of iterations as defined by "Maximum iterations" in area 3. The top two values displayed in the training information area tell us about the current training cycle. "Cur Iteration" displays the current training iteration in the current cycle. This value counts up for each consecutive training iteration until the desired error is reached or the maximum number of iterations is reached, whichever comes first. "Cur SSE" displays the Sum of Squares Error for the current iteration. The bottom 4 parameters show statistical data for the training run. "Ave Iteration" gives the average number of iterations needed to train the network to the specified error. "Av Time (ms)" gives the average number of milliseconds required to train the network to the specified error. "Total Times" shows how many times the software successfully trained the network to the specified error. "Succ Rate" displays the rate at which the software was successful in training the

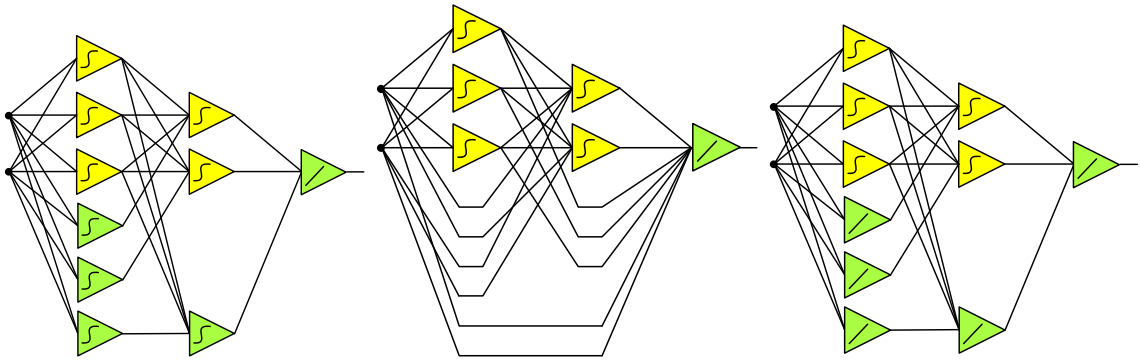
network. This value is equal to “Total Times” divided by “Training Times.”

All experiments were run using the NBN algorithm and all initial weights for training were randomized to non-zero values between +1 and -1. Training trials were run 100 times for each network up to a maximum of 500 training iterations in each trial. The desired SSE was set to 0.01. Many training trials were run for each network, attempting to optimize the training parameters. Final results shown for each network architecture were the optimal results obtained during experimentation.

The first set of experiments will focus on comparing equivalent networks, as defined in Chapter 4, against benchmarks to see which of the equivalent networks yields the highest success rate. This range of experiments will help determine if, all things being equal (i.e. networks are functionally equivalent), one particular architecture is consistently superior to the others with regards to training success across a range of benchmarks. In essence, this helps determine if architecture superiority is dependent on the benchmark. The second set of experiments will focus on comparing a minimally sized network against the same benchmarks to see which network yields the best success rate. This range of experiments will serve as a control group for comparison to the first set. Again, we will see if a particular architecture is consistently superior to the others with regards to training success when network size, not network equivalence, is the varied parameter.

The 3-D Control Surface Benchmark which has 1600 training patterns was used as the first benchmark to compare the success rate, average number of iterations required, and the average training time for equivalent BMLP, DNN, and MLP networks. Equivalent network architectures were setup per the architectural conversion process

outlined in Chapter 4. The BMLP network used a 2=3=2=1 architecture. The equivalent DNN network used a 2-3(3)-2(1)-1 architecture. And the equivalent MLP network used a 2-6-3-1 architecture. Figure 5.3 shows the three equivalent networks side-by-side for ease of comparison and review. Table 5.2 summarizes the training results. The supporting training data may be found in Appendix A.



2-6-3-1 MLP 2=3=2=1 BMLP 2-3(3)-2(1)-1 DNN

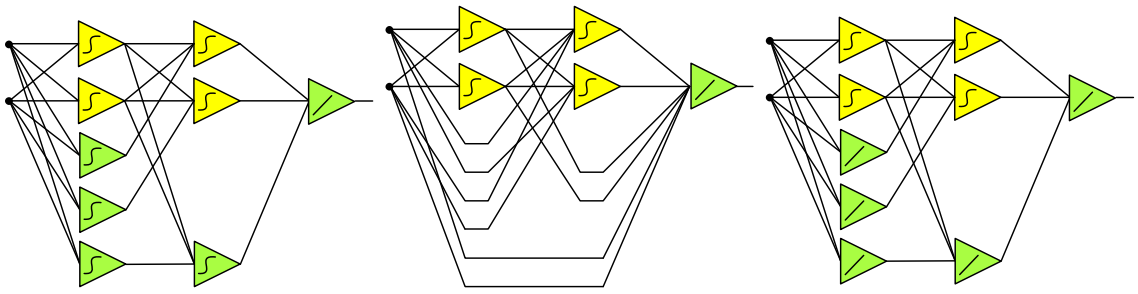
Figure 5.3: Equivalent networks for 3-D Surface

Architecture	Success Rate	Average Iterations	Average Training Time (ms)
MLP 2-6-3-1	94.00%	202.47	5997.37
BMLP 2=3=2=1	15.00%	275.73	5356.27
DNN 2-3(3)-2(1)-1	19.00%	303.11	4000.95

Table 5.2: Comparison of equivalent architectures on 3-D Surface

The Simple 3-D Control Surface Benchmark which has 25 training patterns was used as the second benchmark to compare the success rate, average number of iterations required, and the average training time for equivalent BMLP, DNN, and MLP networks.

Again, equivalent network architectures were setup per the architectural conversion process outlined in Chapter 4. The BMLP network used a 2=2=2=1 architecture. The equivalent DNN network used a 2-2(3)-2(1)-1 architecture. And the equivalent MLP network used a 2-5-3-1 architecture. Figure 5.4 shows the three equivalent networks side-by-side while Table 5.3 summarizes the results. The corresponding training data which supports the results in the table may be found in Appendix B.



2-5-3-1 MLP 2=2=2=1 BMLP 2-2(3)-2(1)-1 DNN

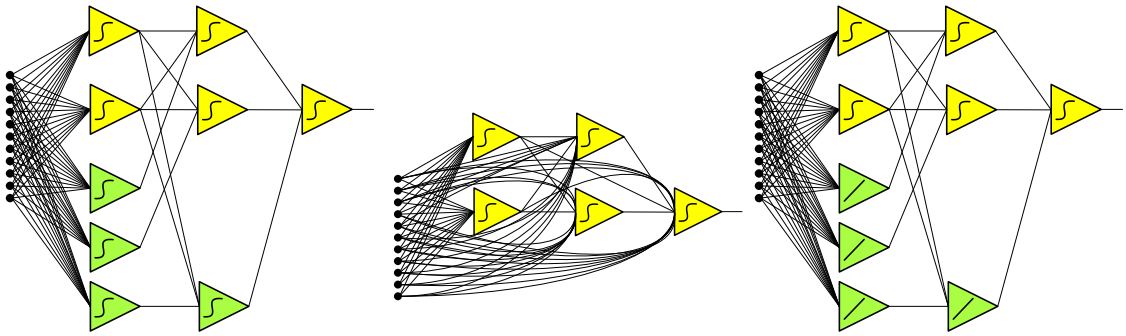
Figure 5.4: Equivalent networks for Simple 3-D Surface

Architecture	Success Rate	Average Iterations	Average Training Time (ms)
MLP 2-5-3-1	100.00%	35.3	46.55
BMLP 2=2=2=1	86.00%	59.38	50.28
DNN 2-2(3)-2(1)-1	90.00%	96.71	96.23

Table 5.3: Comparison of equivalent architectures on Simple 3-D Surface

The Parity-11 Benchmark which has 2048 training patterns was used as a third benchmark to compare the success rate, average number of iterations required, and the

average training time for equivalent BMLP, DNN, and MLP networks. Once again, equivalent network architectures were setup per the architectural conversion process outlined in Chapter 4. The BMLP network used a 11=2=2=1 architecture. The equivalent DNN network used a 11-2(3)-2(1)-1 architecture. And the equivalent MLP network used a 11-5-3-1 architecture. Figure 5.5 shows the three equivalent networks side-by-side. Table 5.4 summarizes the results. Training results supporting the data summarized in Table 5.4 can be found in Appendix C



11-5-3-1 MLP 11=2=2=1 BMLP 11-2(3)-2(1)-1 DNN

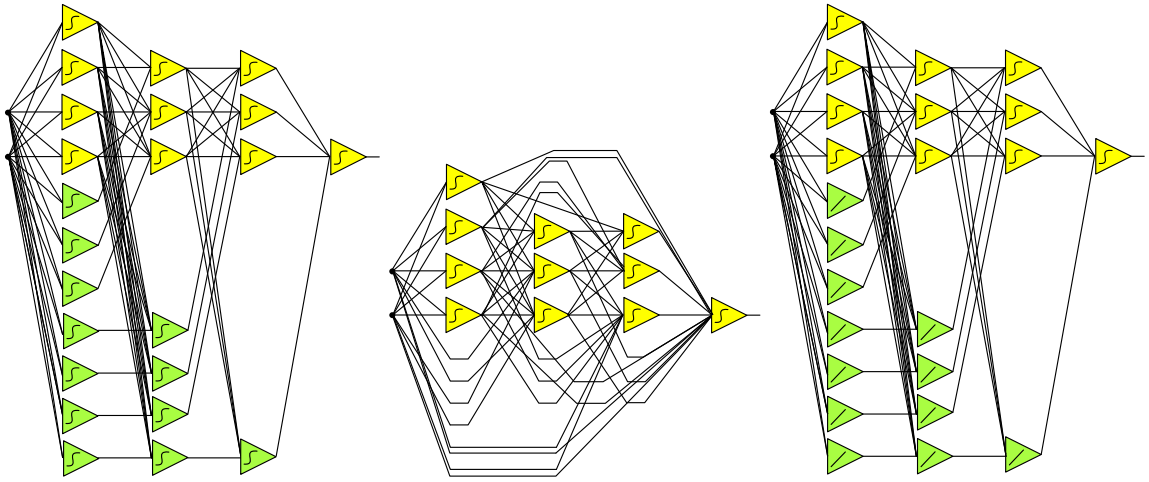
Figure 5.5: Equivalent networks for Parity-11

Architecture	Success Rate	Average Iterations	Average Training Time (ms)
MLP 11-5-3-1	1.00%	492	66706.00
BMLP 11=2=2=1	3.00%	201.33	5455.00
DNN 11-2(3)-2(1)-1	5.00%	275.40	11956.40

Table 5.4: Comparison of equivalent architectures on Parity-11 Benchmark

The Checker-N Benchmark which has 961 training patterns for the 3x3 grid was

used as the fourth benchmark to compare the success rate, average number of iterations required, and the average training time for equivalent BMLP, DNN, and MLP networks. Once again, equivalent network architectures were setup per the architectural conversion process outlined in Chapter 4. The BMLP network used a 2=4=3=3=1 architecture. The DNN network used a 2-4(7)-3(4)-3(1)-1 architecture. And the MLP network used a 2-11-7-4-1 architecture. Figure 5.6 shows the three equivalent networks side-by-side. Table 5.5 summarizes the results. In this instance, the BMLP network had the highest success rate. The MLP network failed to converge and the DNN network had a 1% success rate. The data supporting these results can be found in Appendix D.



2-11-7-4-1 MLP 2=4=3=3=1 BMLP 2-4(7)-3(4)-3(1)-1 DNN

Figure 5.6: Equivalent networks for Checker-3

Architecture	Success Rate	Average Iterations	Average Training Time (ms)
MLP 2-11-7-4-1	0.00%	N/A	N/A
BMLP 2=4=3=3=1	10.00%	256.10	21045.40
DNN 2-4(7)-3(4)-3(1)-1	1.00%	141.00	2262.00

Table 5.5: Comparison of equivalent architectures on Checker-3 Benchmark

In this sequence of experiments, where equivalent networks were compared, we found that the MLP architecture had the highest success rate with the 3-D Surfaces, but came in last on both the Parity and the Checker benchmarks. The BMLP architecture performed the best on the Checker benchmark and came in second on the Parity and 3-D Surfaces benchmarks. The DNN architecture yielded the best results for the Parity benchmark and yielded the second best results for the 3-D Surfaces and Checker benchmarks. These results show that no single architecture is superior across all benchmarks, solidifying the need for different architectures and the ability to convert between them.

The next set of experiments focuses on comparing a minimally sized network from each of the three architectures against four different benchmarks. The goal is to find which architecture yields the highest success rate for the given benchmark. Once the best network architecture is found, one can convert to either of the two other architectures if desired.

The 3-D Control Surface Benchmark which has 1600 training patterns was used

again as a benchmark to compare the success rate, average number of iterations required, and the average training time for the BMLP, DNN, and MLP networks. Minimal network architectures were used for each architectural. The BMLP network used a 2=3=3=1 architecture. The DNN network used a 2-3(4)-3(1)-1 architecture. And the MLP network used a 2-3-3-1 architecture. Table 5.6 summarizes the results. The supporting training data may be found in Appendix E.

Architecture	Success Rate	Average Iterations	Average Training Time (ms)
MLP 2-3-3-1	32.00%	257.56	4050.38
BMLP 2=3=3=1	36.00%	290.47	7923.31
DNN 2-3(4)-3(1)-1	45.00%	265.47	10599.71

Table 5.6: Minimal Architecture Comparison on 3-D Surface Benchmark

The Simple 3-D Control Surface Benchmark which has 25 training patterns was used as a benchmark to compare the success rate, average number of iterations required, and the average training time for the BMLP, DNN, and MLP networks. Minimal network architectures were used for each architectural. The BMLP network used a 2=2=2=1 architecture. The DNN network used a 2-2(3)-2(1)-1 architecture. And the MLP network used a 2-2-2-1 architecture. Table 5.7 summarizes the results. The corresponding training data which supports the results in the table may be found in Appendix F.

Architecture	Success Rate	Average Iterations	Average Training Time (ms)
MLP 2-2-2-1	74.00%	73.42	63.64
BMLP 2=2=2=1	86.00%	59.38	50.28
DNN 2-2(3)-2(1)-1	88.00%	96.71	96.23

Table 5.7: Minimal Architecture Comparison on Simple 3-D Surface Benchmark

The Parity-11 Benchmark which has 2048 training patterns was used as a benchmark to compare the success rate, average number of iterations required, and the average training time for the BMLP, DNN, and MLP networks. Minimal network architectures were used for each architectural. The BMLP network used a 11=2=2=1 architecture. The DNN network used a 11-2(3)-2(1)-1 architecture. And the MLP network used a 11-2-2-1 architecture. Table 5.8 summarizes the results. Training results supporting the data summarized in Table 5.8 can be found in Appendix G.

Architecture	Success Rate	Average Iterations	Average Training Time (ms)
MLP 11-2-2-1	2.00%	298.5	10749.00
BMLP 11=2=2=1	3.00%	201.33	5455.00
DNN 11-2(3)-2(1)-1	5.00%	275.40	11956.40

Table 5.8: Minimal Architecture Comparison on Parity-11 Benchmark

The Checker-N Benchmark with 961 training patterns for the 3x3 grid was used

as a benchmark to compare the success rate, average number of iterations required, and the average training time for the BMLP, DNN, and MLP networks. Once again, minimal network architectures were used for each architectural. The BMLP network used a 2=4=3=3=1 architecture. The DNN network used a 2-4(7)-3(4)-3(1)-1 architecture. And the MLP network used a 2-4-3-3-1 architecture. Table 5.9 summarizes the results. In this instance, the BMLP network had the highest success rate. The MLP network failed to converge and the DNN network had a 1% success rate. The data supporting these results can be found in Appendix H.

Architecture	Success Rate	Average Iterations	Average Training Time (ms)
MLP 2-4-3-3-1	0.00%	N/A	N/A
BMLP 2=4=3=3=1	9.00%	256.10	21045.40
DNN 2-4(7)-3(4)-3(1)-1	1.00%	141.00	2262.00

Table 5.9: Minimal Architecture Comparison on Checker-3 Benchmark

In this sequence of experiments where minimizing network size was the main objective, the MLP architecture came in last for all benchmarks. The BMLP architecture yielded the best result, once again, on the Checker benchmark and the second best result on the others. The DNN architecture had the best training results for all benchmarks, except for the Checker benchmark where it came in second. Table 5.10 summarizes the results of all tests, noting winners with a green highlight and second place with a yellow highlight.

	MLP	BMLP	DNN	
3D Surface	94%	15%	19%	Equivalent Networks
	32%	36%	45%	Minimal Networks
Simple 3D Surface	100%	86%	90%	Equivalent Networks
	74%	86%	88%	Minimal Networks
Parity-N	1%	3%	5%	Equivalent Networks
	2%	3%	5%	Minimal Networks
Checker-N	Failed to Converge	10%	1%	Equivalent Networks
	Failed to Converge	9%	1%	Minimal Networks

Table 5.10: Summary of Winners

By reviewing Table 5.10, one can easily see that the DNN architecture placed first or second in all experiments. Additionally, it has four wins to the other networks' two wins each. While the DNN architectures tend to have slightly higher average training times, this is overshadowed by its overall superior success rate. One would willingly take a longer training time over a failure to converge during the training process. While the DNN architecture does not always yield the highest success rate, it did converge in all test cases showing that it is a valuable tool for training and optimization.

The newly introduced DNN architecture proves to have success rate advantages both when compared to equivalent networks as well as when it is pitted against other minimally sized networks. These are two qualities that are highly desirable in any NN architecture.

Chapter 6

Conclusion

Over the past several decades, we have seen many advances in the field of ANNs, including the architecture progression from the multi-layer Perceptron (MLP) to the bridged multi-layer Perceptron (BMLP) to fully connected cascade (FCC) architecture [10] and finally to dual neural networks (DNN) [11]. This research has produced many advances in training algorithms as well as improvements in architecture design which have significantly improved efficiency [11] [12] [13]. However, until now, little time has been devoted to conversion between network architectures and any advances that this might produce.

As I noted in the introduction, I discovered that if you use two linear neurons, one in each hidden layer, in a traditional MLP architecture, that we can reduce the number of required neurons for the Parity-11 problem from 11 to 6. My conclusion was that somehow this DNN design significantly increases the power of a network. I wondered, "What other advantages do DNNs offer?" This let me to investigate DNNs as a tool for training, optimization, and network conversion.

The DNN architecture presented in this study offers advances in training, optimization, and network conversion. One of its biggest advances can be seen in the area of deep neural networks. Figure 6.1 shows a success rate comparison between BMLP and MLP networks. Notice that as the number of hidden layers increases, the success rate for

the BMLP networks steadily increase until they are essentially 100%. During this same time, as the networks get deeper, the MLP success rate drops to 0%.This highlights a fundamental problem with Deep MLP networks. They are very hard, if not impossible to train – at least until now!

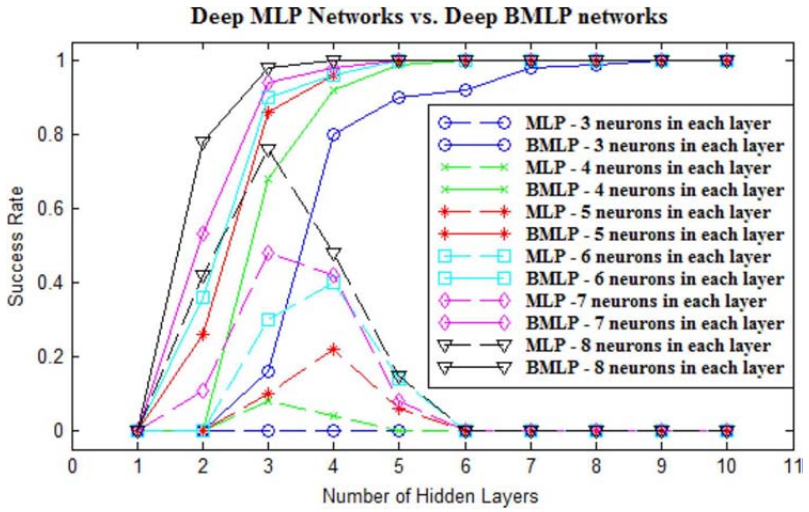


Figure 6.1: Success rates comparison for training the two-spiral patterns [10]

With the conversion methods presented in this study, we now have a path for network conversion between BMLP, DNN, and MLP architectures. That means that we now have a training solution for deep MLP networks. As seen in the experimental results in the previous chapter, DNN networks have significantly higher overall success rates compared to BMLP and MLP networks. In fact, the DNN architecture had either the highest or the second highest success rate in all experiments. In some cases, the MLP network failed to converge while one or both of the other architectures yielded convergence. In these cases, we can simply train either a DNN or BMLP network and then perform the conversion to MLP if that architecture is desired.

The utilization of DNNs as a tool for training, optimization, and network conversion solves some of the most difficult problems faced in today's ANN research.

This research provides a method for improved ANN training, optimization of ANN architectures, and a mechanism for ANN architecture conversion using DNNs. These new advances will revolutionize ANN research.

Bibliography

[1] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, pp. 533-536, 1986.

[2] P. J. Werbos, "Back-propagation: Past and future," in *Proceedings of International Conference on Neural Networks*, San Diego, 1988.

[3] K. Levenberg, "A method for the solution of certain problems in least squares," *Quarterly of Applied Mathematics,* vol. 2, pp. 164-168, 1944.

[4] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Transactions on Nerural Networks,* vol. 5, no. 6, pp. 989-993, 1994.

[5] B. M. Wilamowski, N. J. Cotton, O. Kaynak and G. Dundar, "Computing gradient vector and jacobian matrix in arbitrarily connected neural networks," *IEEE Transactions on Industrial Electronics,* vol. 55, no. 10, pp. 3784-3790, 2008.

[6] B. M. Wilamowski and H. Yu, "Improved Computation for Levenberg Marquardt Training," *IEEE Transactions on Neural Networks,* vol. 21, no. 6, pp. 930-937, 2010.

[7] B. M. Wilamowski and H. Yu, "Neural Network Learning Without Backpropagation," *IEEE Transactions on Neural Networks,* vol. 21, no. 11, pp. 1793 - 1803, 2010.

[8] V. V. Phansalkar and P. S. Sastry, "Analysis of the back-propagation algorithm with momentum," *IEEE Transactions on Neural Networks,* vol. 5, no. 3, pp. 505-506, 1994.

[9] B. M. Wilamowski and L. Torvik, "Modification of gradient computation in the back-propagation algorithm," in *ANNIE'93 - Artificial Neural Networks in Engineering*, St. Louis, 1993.

[10] D. S. Hunter, H. Yu, M. S. Pukish, J. Kolbusz and B. M. Wilamowski, "Selection of Proper Neural Network Sizes and Architectures -- A Comparative Study," *IEEE Transactions on Industrial Informatics,* vol. 8, no. 2, pp. 228-240, 2012.

[11] D. S. Hunter and B. M. Wilamowski, "Parallel Multi-Layer Neural Network Architecture with Improved Efficiency," in *Proceedings of the 4th International Conference on Human System Interaction*, Yokohama, 2011.

[12] S. Trenn, "Mulit-layer Perceptrons: Approximation Order and Necessary Number of Hidden Units," *IEEE Transactions on Neural Networks,* vol. 19, no. 5, pp. 836-844, 2008.

[13] B. M. Wilamowski, "Neural Network Architectures and Learning Algorithms," *IEEE Industrial Electronics Magazine,* pp. 56-63, December 2009.

[14] B. M. Wilamowski, D. S. Hunter and A. Malinowski, "Solving parity-n problems with feedforward neural networks," in *Proceedings of IJCNN'03 -- International Joing Conference on Neural Networks*, Portland, 2003.

[15] M. Jianwen, Hasibagan, Buheosr and Z. Zijiang, "The classification of AVHRR thermal infrared data and ground weather temperature data by using neural network," in *Proceedings of the 2003 IEEE International Geoscience and Remote Sensing Symposium*, Toulouse, 2003.

[16] S. Mandal, J. Choudhury, S. Bhadra and D. De, "Growth Estimation with Artificial Neural Network Considering Weather Parameters Using Factor and Principal Component Analysis," in *Proceedings of the 10th International Conference on Information Technology*, Rourkela, 2007.

[17] Y. Quan, "Research on weather forecast based on neural networks," in *Proceedings of the 3rd World Congress on Intelligent Control and Automation*, Beijing, 2000.

[18] V. N. Ghate and S. V. Dudul, "Cascade neural-network-based fault classifier for three-phase induction motor," *IEEE Transactions on Industrial Electronics,* vol. 58, no. 5, pp. 1555-1563, 2011.

[19] T. Orlowska-Kowaslska, M. Dybkowski and K. Szabat, "Adaptive sliding-mode neuro-fuzzy control of the two-mass induction motor drive without mechanical sensors," *IEEE Transactions on Industrial Electronics,* vol. 57, no. 2, pp. 553-564, 2010.

[20] M. Pucci and M. Cirrincione, "Neural MPPT control of wind generators with induction machines without speed sensors," *IEEE Transactions on Industrial Electronics,* vol. 58, no. 1, pp. 37-47, 2011.

[21] F. F. M. El-Sousy, "Hybrid H∞ based wavelet-neural-network tracking control for permanent-magnet synchronous motor servo drives," *IEEE Transactions on Industrial Electronics,* vol. 57, no. 9, pp. 3157-3166, 2010.

[22] C. Xia, C. Guo and T. Shi, "A neural-network-identifier and fuzzy-controller-based algorithm for dynamic decoupling control of permanent-magnet spherical motor," *IEEE Transactions on Inductrial Electronics,* vol. 57, no. 8, pp. 2868-2878, 2010.

[23] Q. N. Le and J. W. Jeon, "Neural-network-based low-speed-damping controller for stepper motor with an FPGA," *IEEE Transactions on Industrial Electronics,* vol. 57, no. 9, pp. 3167-3180, 2010.

[24] C. -F. Juang, Y. -C. Chang and C. -M. Hsiao, "Evolving gaits of a hexapod robot by recurrent neural networks with symbiotic species-based particle swarm optimization," *IEEE Transactions on Industrial Electronics,* vol. 58, no. 7, pp. 3110-3119, 2011.

[25] C. -C. Tsai, H. -C. Huang and S. -C. Lin, "Adaptive neural network control of a self-balancing two-wheeled scooter," *IEEE Transactions on Industrial Electronics,* vol. 57, no. 4, pp. 1420-1428, 2010.

[26] M. Charkhgard and M. Farrokhi, "State-of-charge estimation for lithium-ion

batteries using neural networks and EKF," *IEEE Transactions on Industrial Electronics,* vol. 57, no. 12, pp. 4178-4187, 2010.

[27] A. Yahyaoui, N. Fnaiech and F. Fnaiech, "A Suitable Initialization Procedure for Speeding a Neural Network Job-Shop Scheduling," *IEEE Transactions on Industrial Electronics,* vol. 58, no. 3, pp. 1052-1060, 2011.

[28] V. Machado, A. Neto and J. D. de Melo, "A neural network multiagent architecture applied to industrial networks for dynamic allocation of control strategies using standard function blocks," *IEEE Transactions on Industrial Electronics,* vol. 57, no. 5, pp. 1823-1834, 2010.

[29] C. -H. Lu, "Wavelet fuzzy nerual networks for identification and predictive control of dynamic systems," *IEEE Transactions on Industrial Electronics,* vol. 58, no. 7, pp. 3046-3058, 2011.

[30] B. M. Wilamowski and O. Kaynak, "Oil well diagnosis by sensing terminal characteristics of the induction motor," *IEEE Transactions on Industrial Electronics,* vol. 47, no. 5, pp. 1100-1107, 2000.

[31] A. E. Bryson and Y.-C. Ho, Applied optimal control: Optimization, Estimation, and Control, Waltham: Blaisdell Publishing Company, 1969.

[32] H. Demuth and M. Beale, Neural Network Toolbox For Use with Matlab, Natick: The MathWorks, Inc., 1993.

[33] B. M. Wilamowski, "Challenges in applications of computational intelligence in industrial electronics," in *Proceedings of International Symposium on Industrial Electronics*, Bari, 2010.

[34] B. M. Wilamowski and H. Yu, "NNT - Neural Networks Trainer," [Online]. Available: http://www.eng.auburn.edu/~wilambm/nnt/index.htm. [Accessed 1 February 2012].

[35] H. Yu and B. M. Wilamowski, "C++ Implementation of Neural Networks Trainer," in *Proceedings of 13th IEEE Intelligent Engineering Systems Conference*, Barbados, 2009.

[36] S. I. Gallant, Neural Network Learning and Expert Systems, Cambridge: MIT Press, 1993.

[37] A. Salvetti and B. M. Wilamowski, "Introducing stochastic processes within the backpropagation algorithm for improved convergence," in *Proceedings of ANNIE'94 -- Artificial Neural Networks and Intelligent Engineering*, St. Louis, 1994.

[38] J. F. Kolen, "Faster Learning Through a Probabilistic Approximation Algorithm," in *Proceedings of IEEE Internation Conference on Neural Networks*, San Diego, 1988.

[39] S. J. Hanson, "Behavioral Diversity, Search and Stochastic Connectionist Systems," in *Quantitative Analysis of Behavior: Neural Network Models of Conditioning and Action*, Harvard Press, 1990.

[40] A. Von Lehmen, E. Paek, P. Liao, A. Marrakchi and J. Patel, "Factors influencing learning by backpropagation," in *Proceedings of IEEE International Conference on Neural Networks* , San Diego, 1988.

[41] A. Van Ooten and B. Nienhuis, "Improving the convergence of the back-propagation

algorithm," *Neural Networks,* vol. 5, pp. 465-471, 1992.

[42] R. Parekh, K. Balakrishnan and V. Honavar, "An empirical comparison of flat-spot elimination techniques in back-propagation networks," in *Proceedings of Third Workshop on Neural Networks*, Auburn University, 1992.

[43] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proceedings of the International Conference on Neural Networks*, San Francisco, 1993.

[44] S. E. Fahlman, "Faster-Learning Variations on Back-Propagation: An Empirical Study," in *In Proceedings of Connectionist Models Summer School* , Pittsburgh, 1988.

[45] G. Lera and M. Pinzolas, "A quasilocal Levenberg-Marquardt algorithm," in *Proceedings of International Joint Conference on Neural Networks*, Anchorage, 1998.

[46] G. Lera and M. Pinzolas, "Neighborhood based Levenberg-Marquardt algorithm for neural network training," *IEEE Transactions on Neural Networks,* vol. 13, no. 5, pp. 1200-1203, 2002.

[47] A. Toledo, M. Pinzolas, J. Ibarrola and G. Lera, "Improvement of the neighborhood based Levenberg-Marquardt algorithm by local adaptation of the learning coefficient," *IEEE Transactions on Neural Networks,* vol. 16, no. 4, pp. 988 - 992, 2005.

[48] N. Pham, H. Yu and B. Wilamowski, "Neural Network Trainer through Computer Networks," in *24th IEEE International Conference on Advanced Information Networking and Applications*, Perth, 2010.

[49] H. Yu and B. M. Wilamowski, "Efficient and Reliable Training of Neural Networks," in *Proceedings of the 2nd International Conference on Human System Interaction*, Catania, 2009.

[50] S. Liu, "A Simplified Dual Neural Network for Quadratic Programming With Its KWTA Application," *IEEE Transactions on Neural Networks,* vol. 17, no. 6, pp. 1500-1510, 2006.

[51] R. Prakash and B. Paul, "Dual nature hidden layers neural networks ‘a novel paradigm of neural network architecture," in *Proceedings of International Conference of the IEEE Engineering in Medicine and Biology Society*, Istanbul, 2001.

[52] J. D. Hewlett, *Methods for Improving Generalization and Convergence in Artificial Neural Classifiers,* Auburn: Auburn University Dissertation, 2011.

[53] R. C. Minnick, "Linear-input logic," *IRE Transactions on Electronics,* Vols. EC-10, pp. 6-16, 1961.

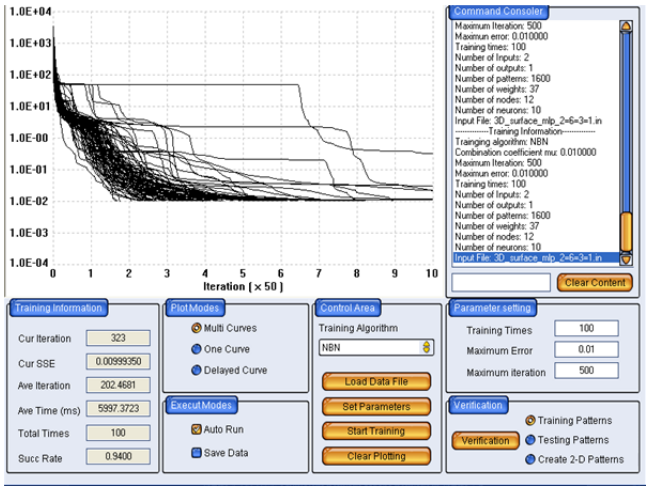
[54] N. D. Pham, *An Efficient Optimization Algorithm with Quasi Gradient Search,* Auburn: Auburn University Dissertation, 2011.

Appendices

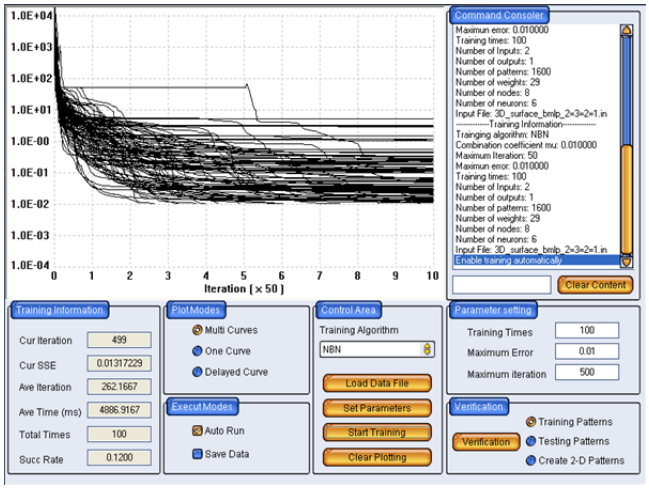
Appendix A

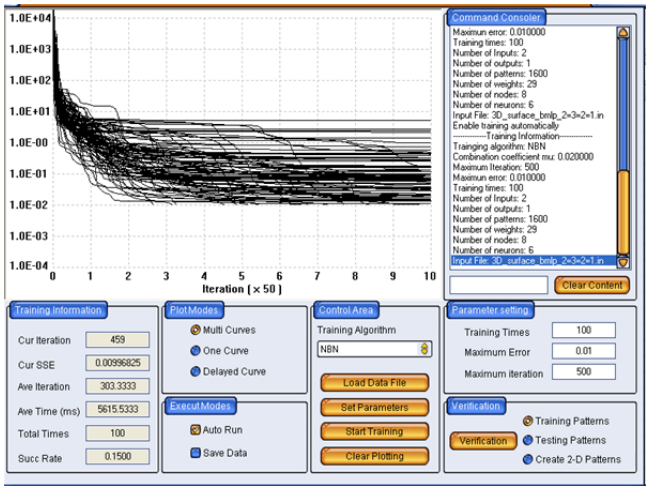
Supporting Data for 3-D Benchmark testing of equivalent networks

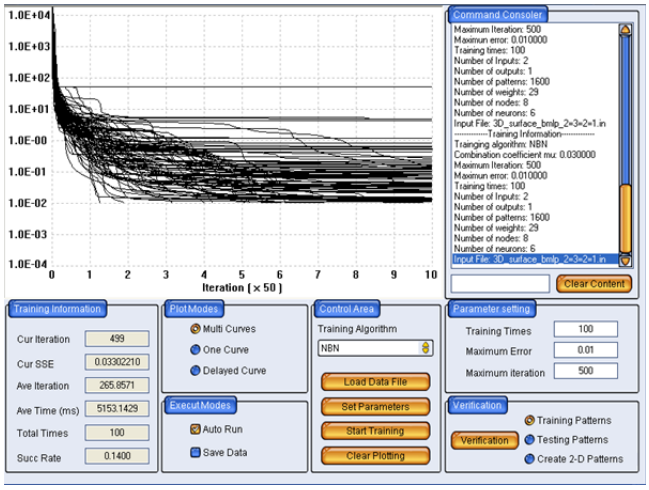
Best training data for MLP 2-6-3-1 network

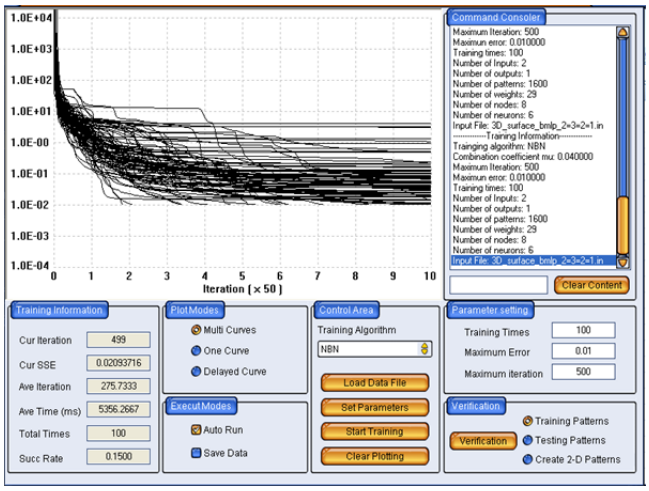


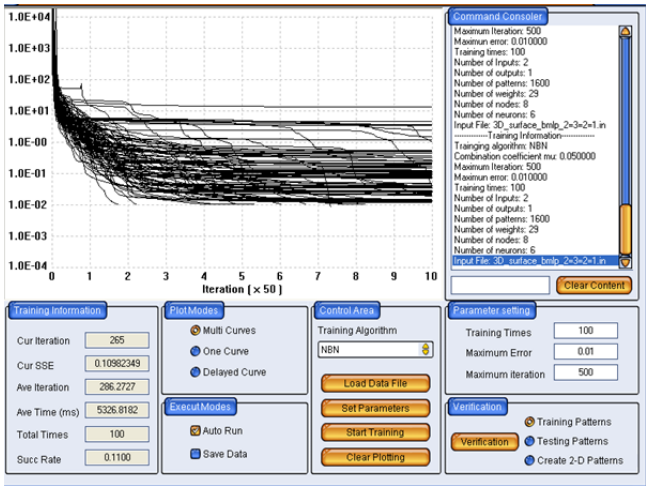
Best training data for BMLP 2=3=2=1 network



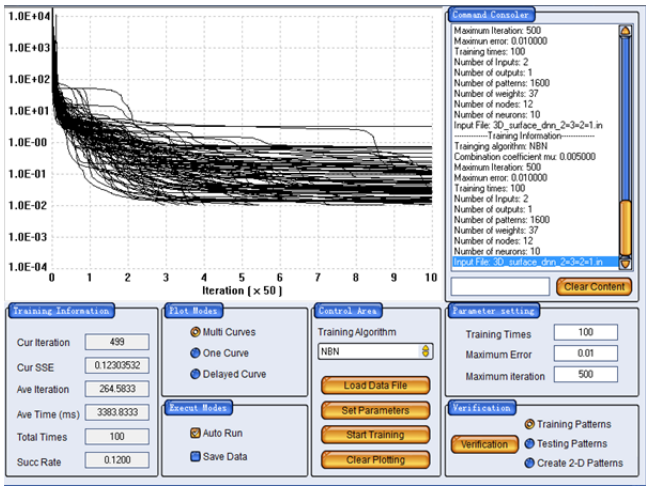

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 29
Number of nodes: 8
Number of neurons: 6
Input File: 3D_surface_bmlp_2×3×2×1.in
Enable training automatically
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.020000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 29
Number of nodes: 8
Number of neurons: 6
Input File: 3D_surface_bmlp_2×3×2×1.in
Clear Content
Training Information
Cur Iteration 459
Cur SSE 0.00996825
Ave Iteration 303.3333
Ave Time (ms) 5615.5333
Total Times 100
Succ Rate 0.1500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximun Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

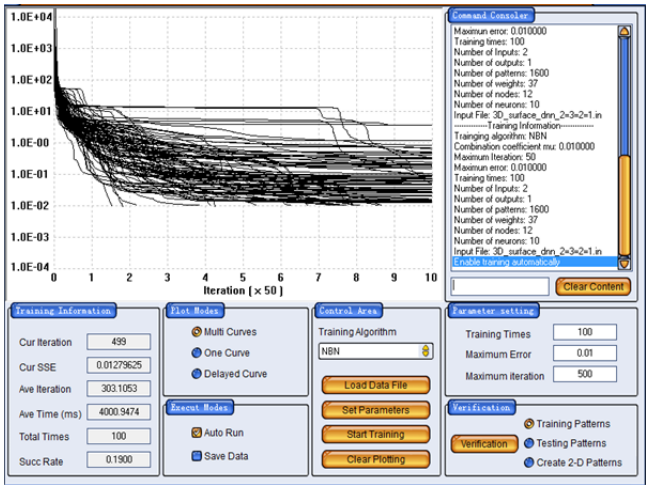

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 29
Number of nodes: 8
Number of neurons: 6
Input File: 3D_surface_bmlp_2×3×2×1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.030000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 29
Number of nodes: 8
Number of neurons: 6
Input File: 3D_surface_bmlp_2×3×2×1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.03302210
Ave Iteration 265.8571
Ave Time (ms) 5153.1429
Total Times 100
Succ Rate 0.1400
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximun Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

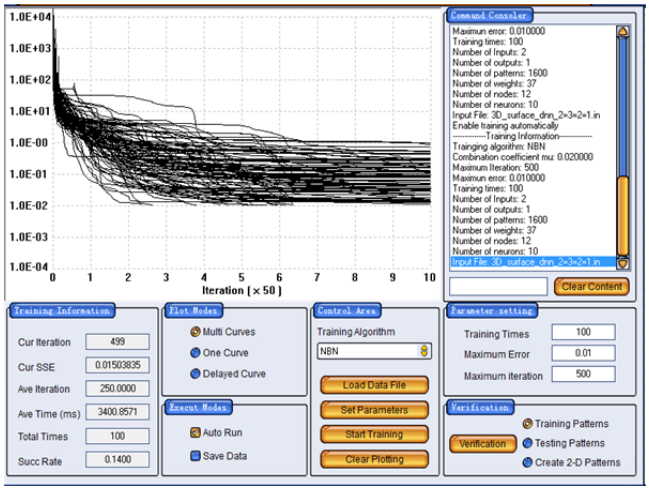

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 29
Number of nodes: 8
Number of neurons: 6
Input File: 3D_surface_bmlp_2×3×2×1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.040000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 29
Number of nodes: 8
Number of neurons: 6
Input File: 3D_surface_bmlp_2×3×2×1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.02093716
Ave Iteration 275.7333
Ave Time (ms) 5356.2667
Total Times 100
Succ Rate 0.1500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximun Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

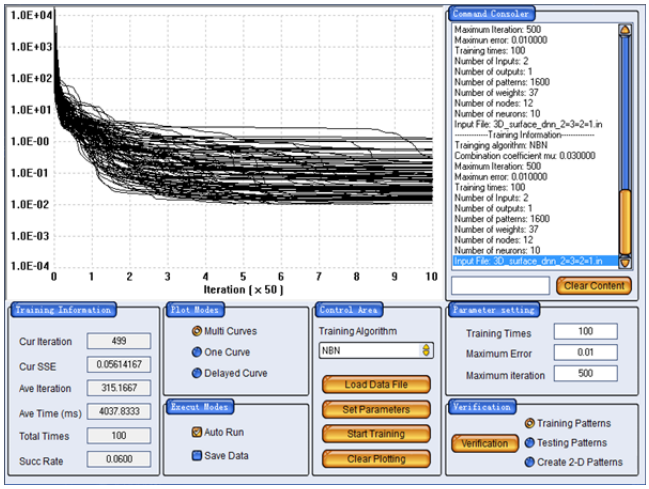


Best training data for DNN 2-3(3)-2(1)-1 network





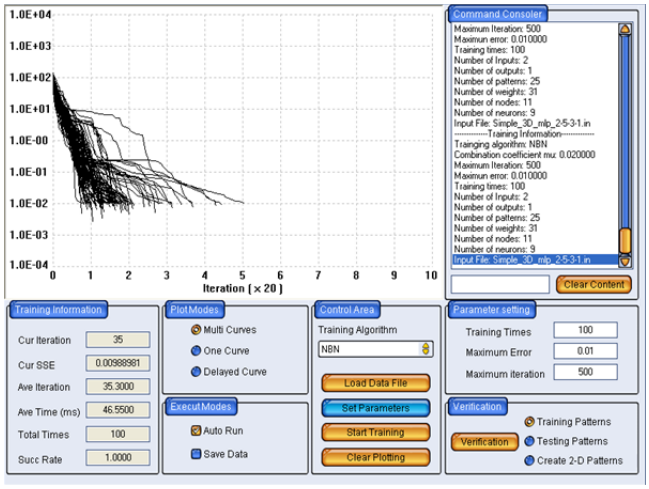

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 37
Number of nodes: 12
Number of neurons: 10
Input File: 3D_surface_dnn_2=3=2=1.in
Enable training automatically
----------Training Information----------
Training algorithm: NBN
Combination coefficient mu: 0.020000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 37
Number of nodes: 12
Number of neurons: 10
Input File: 3D_surface_dnn_2=3=2=1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.01503835
Ave Iteration 250.0000
Ave Time (ms) 3400.8571
Total Times 100
Succ Rate 0.1400
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 37
Number of nodes: 12
Number of neurons: 10
Input File: 3D_surface_dnn_2=3=2=1.in
----------Training Information----------
Training algorithm: NBN
Combination coefficient mu: 0.030000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 1600
Number of weights: 37
Number of nodes: 12
Number of neurons: 10
Input File: 3D_surface_dnn_2=3=2=1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.05614167
Ave Iteration 315.1667
Ave Time (ms) 4037.8333
Total Times 100
Succ Rate 0.0600
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

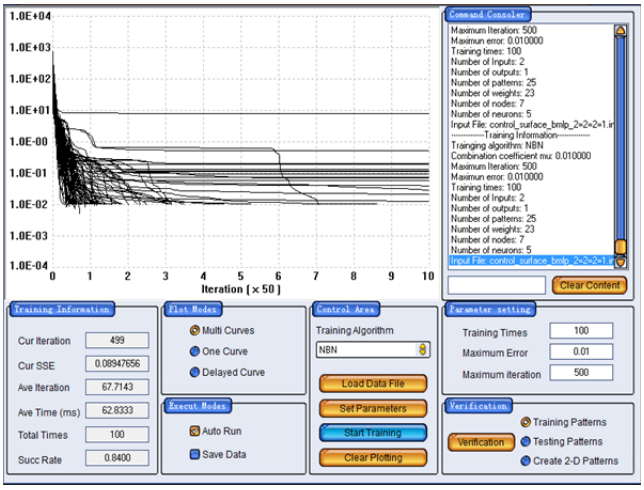
Appendix B

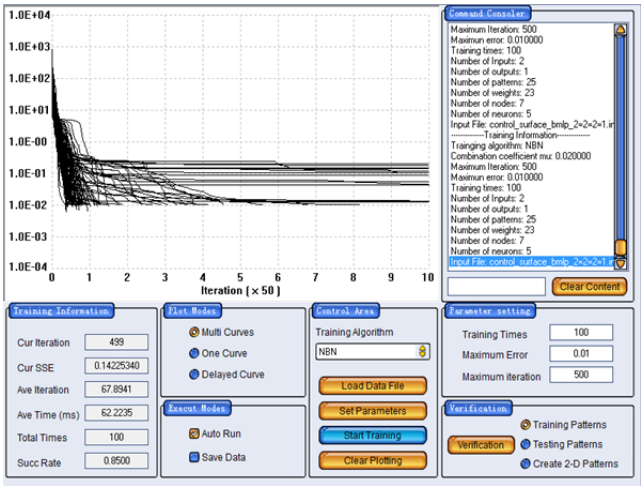
Supporting Data for Simple 3-D Benchmark testing of equivalent networks

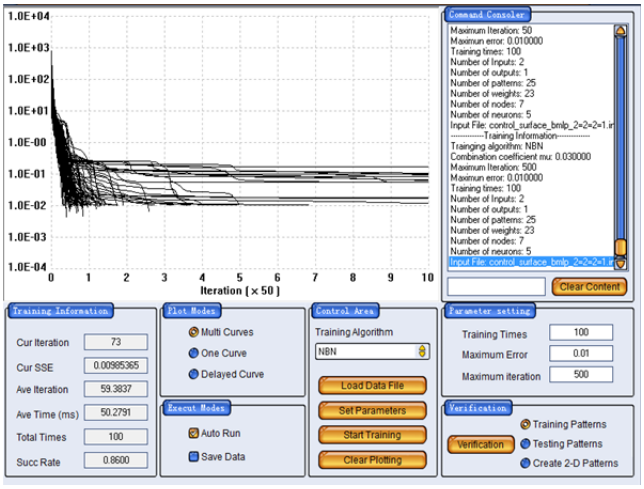
Best training data for MLP 2-5-3-1 network

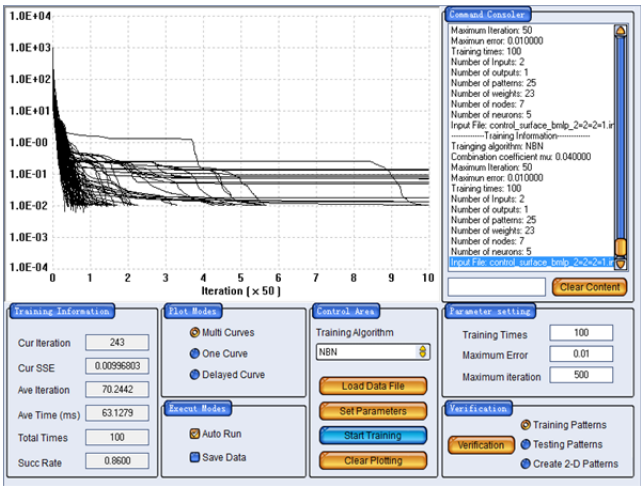


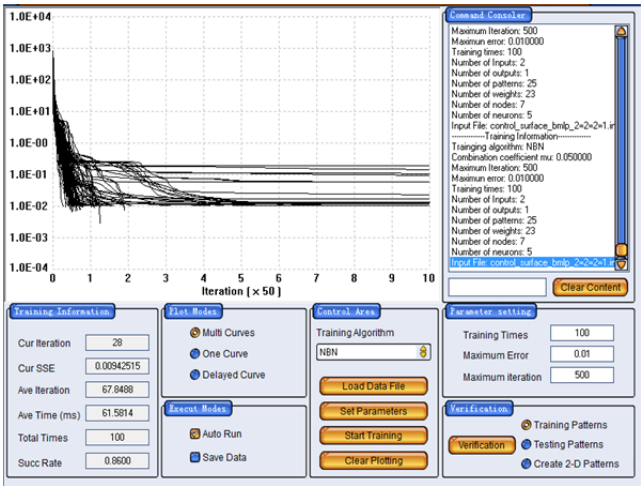
Best training data for BMLP 2=2=2=1 network



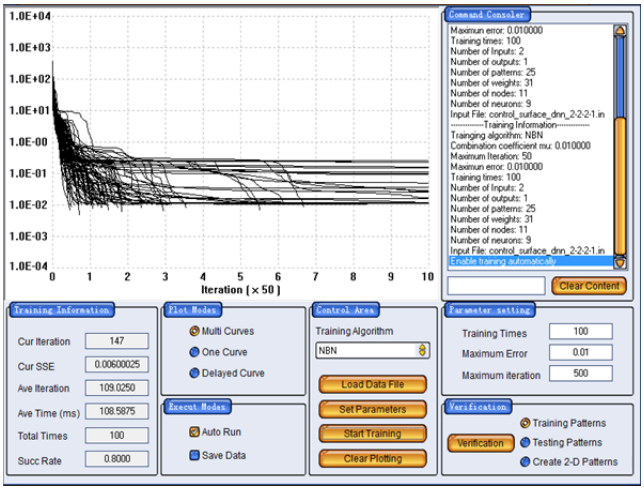

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 23
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_bmlp_2=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.020000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 23
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_bmlp_2=2=2=1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.14225340
Ave Iteration 67.8941
Ave Time (ms) 62.2235
Total Times 100
Succ Rate 0.8500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

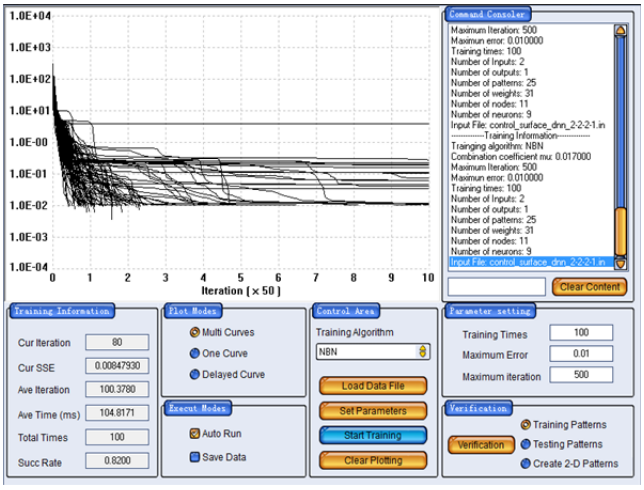

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 23
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_bmlp_2=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.030000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 23
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_bmlp_2=2=2=1.in
Clear Content
Training Information
Cur Iteration 73
Cur SSE 0.00985365
Ave Iteration 59.3837
Ave Time (ms) 50.2791
Total Times 100
Succ Rate 0.8600
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

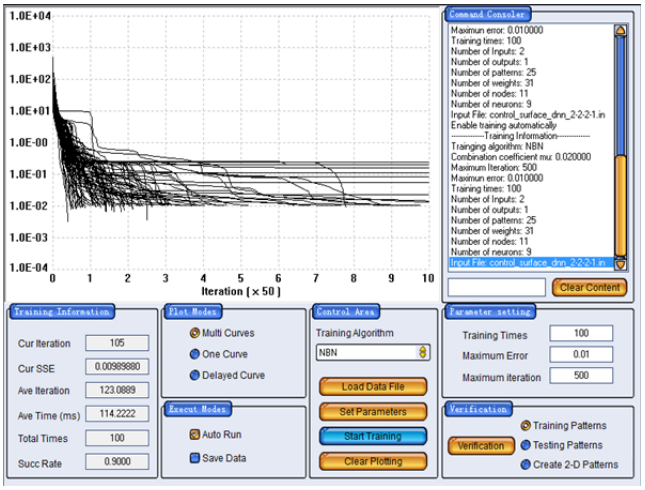

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 23
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_bmlp_2=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.040000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 23
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_bmlp_2=2=2=1.in
Clear Content
Training Information
Cur Iteration 243
Cur SSE 0.00996803
Ave Iteration 70.2442
Ave Time (ms) 63.1279
Total Times 100
Succ Rate 0.8600
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

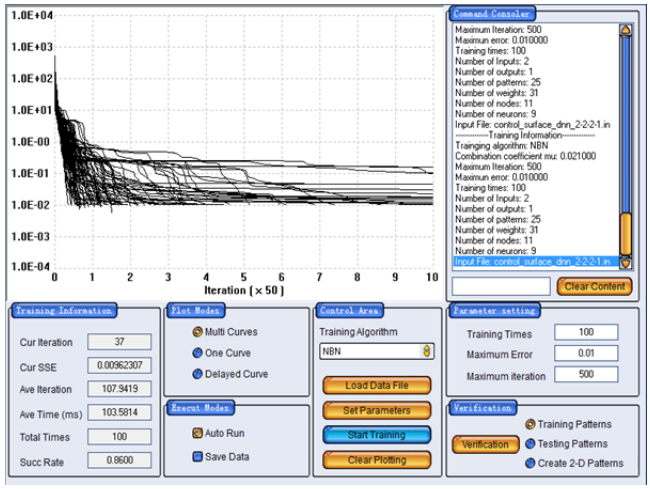


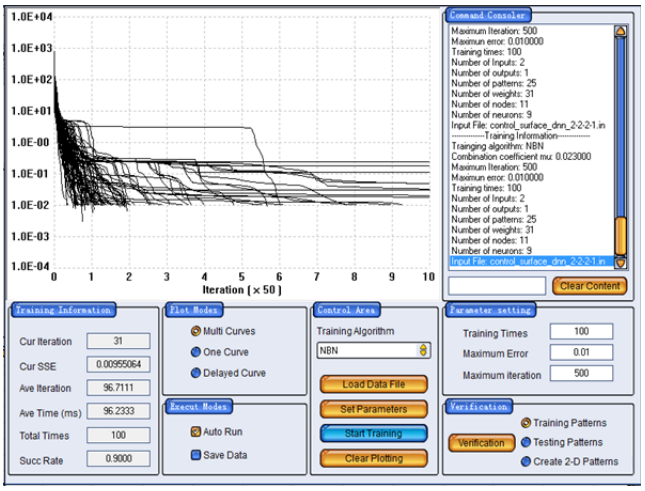
Best training data for DNN 2-2(3)-2(1)-1 network






1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Enable training automatically
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.020000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 105
Cur SSE 0.00989880
Ave Iteration 123.0889
Ave Time (ms) 114.2222
Total Times 100
Succ Rate 0.9000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

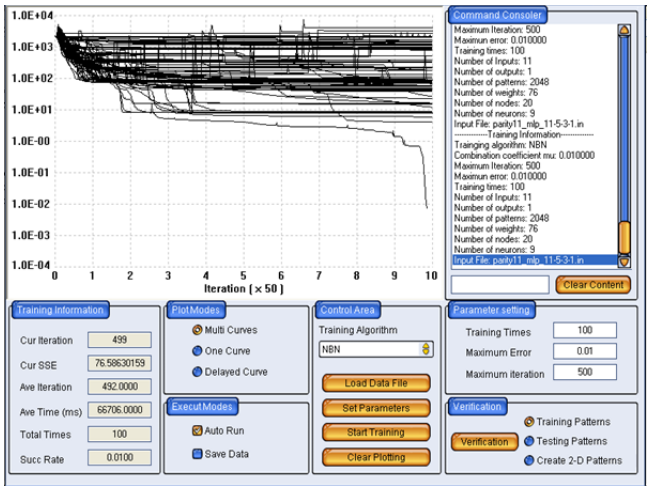

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.021000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 37
Cur SSE 0.00962307
Ave Iteration 107.9419
Ave Time (ms) 103.5814
Total Times 100
Succ Rate 0.8600
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

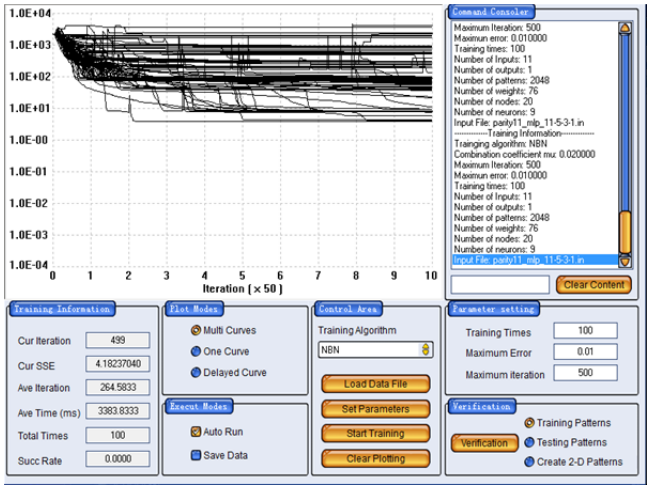

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.023000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 31
Cur SSE 0.00955064
Ave Iteration 96.7111
Ave Time (ms) 96.2333
Total Times 100
Succ Rate 0.9000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

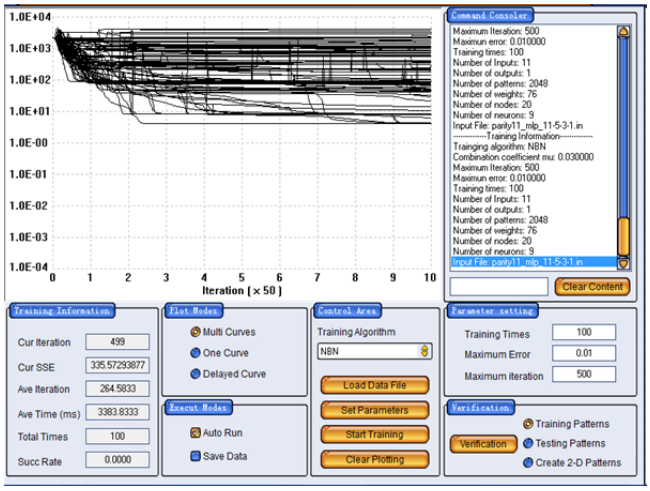
Appendix C

Supporting Data for Parity-11 Benchmark testing of equivalent networks

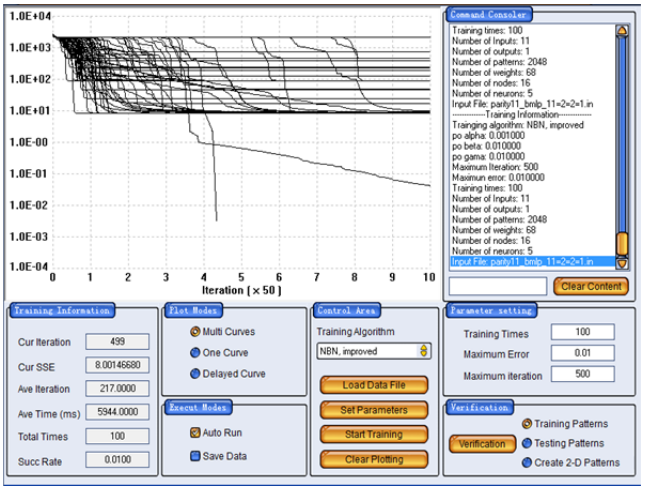
Best training data for MLP 11-5-3-1 network

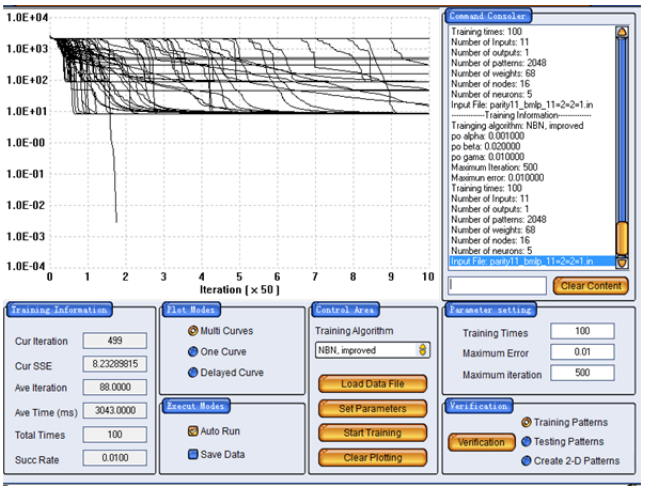


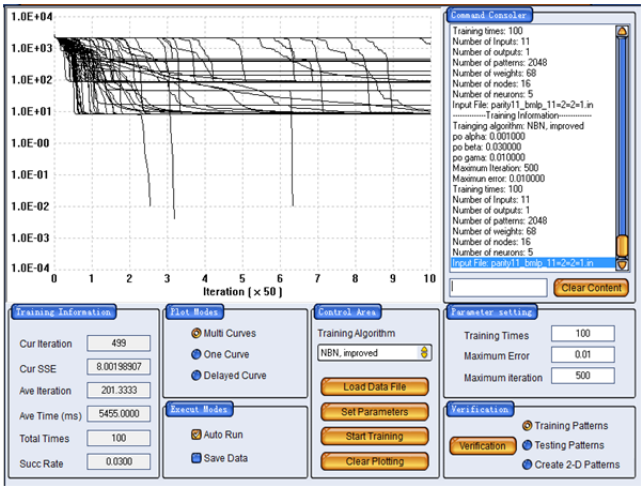


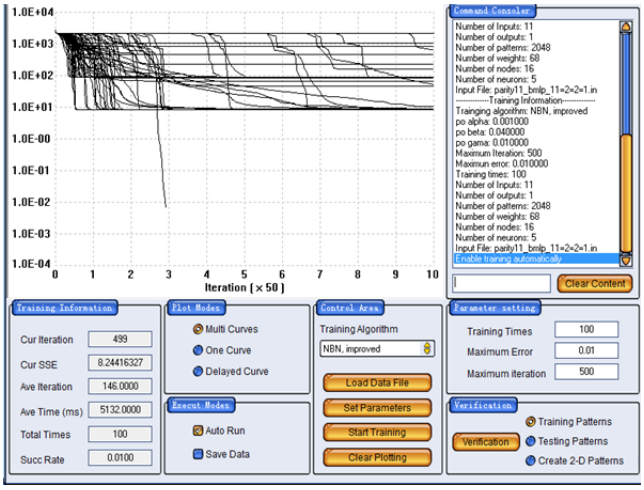


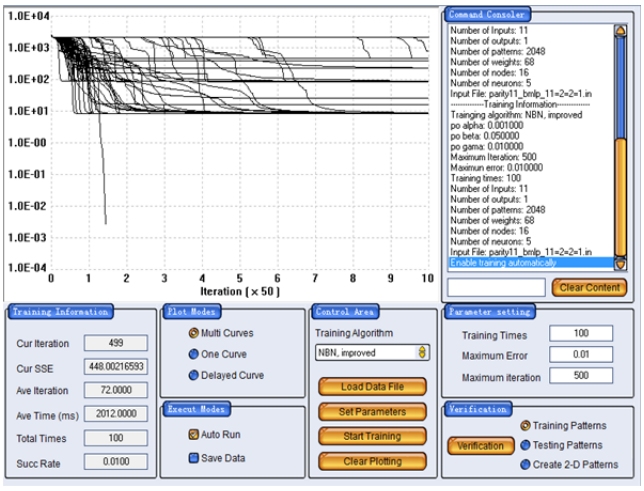
Best training data for BMLP 11=2=2=1 network



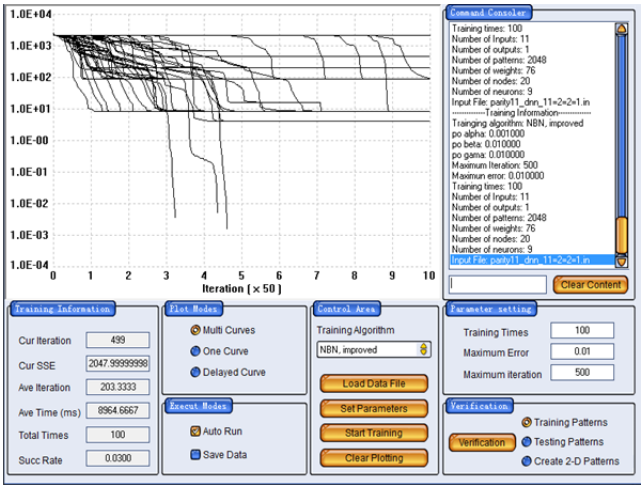


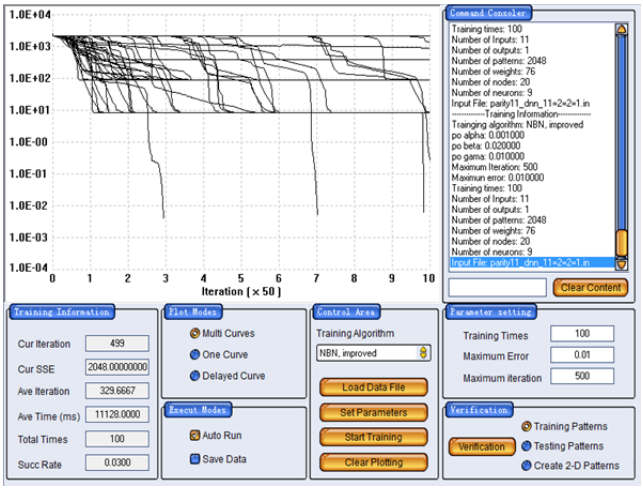

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.030000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2=2=1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 8.00198907
Ave Iteration 201.3333
Ave Time (ms) 5455.0000
Total Times 100
Succ Rate 0.0300
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

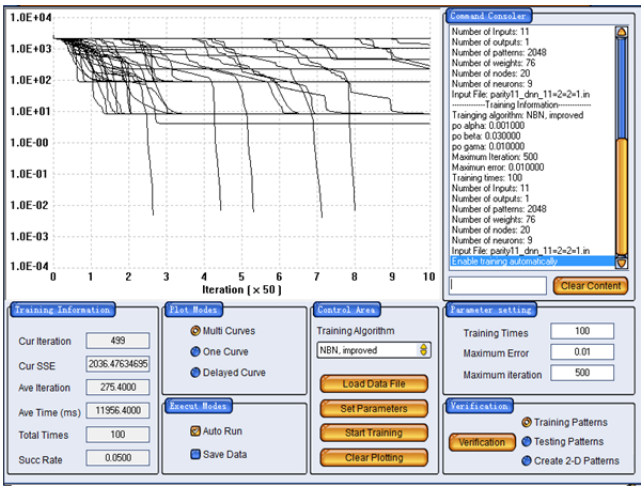

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.040000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2=2=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 499
Cur SSE 8.24416327
Ave Iteration 146.0000
Ave Time (ms) 5132.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

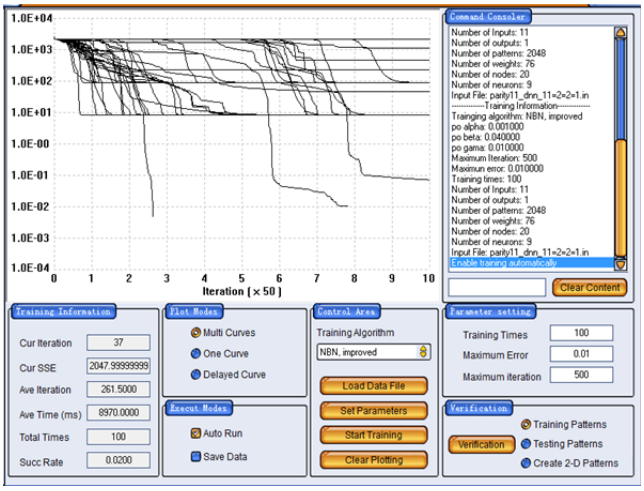

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.050000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2=2=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 499
Cur SSE 448.00216593
Ave Iteration 72.0000
Ave Time (ms) 2012.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

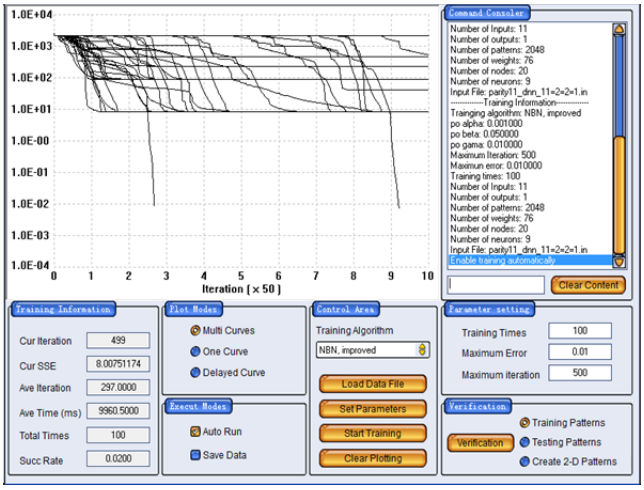
Best training data for DNN 11-2(3)-2(1)-1 network







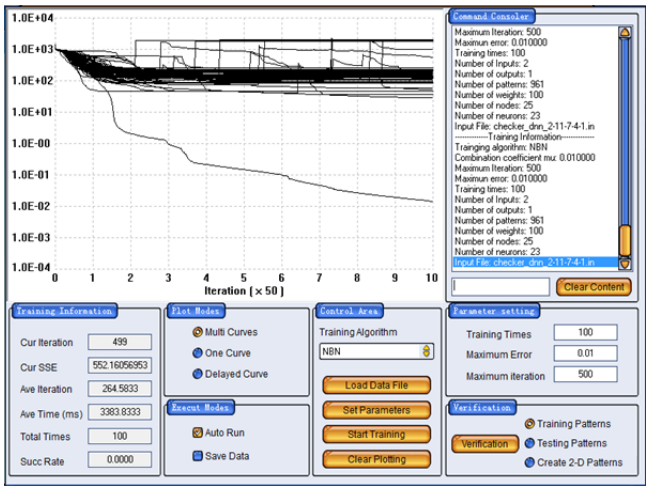

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.040000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 37
Cur SSE 2047.99999999
Ave Iteration 261.5000
Ave Time (ms) 8970.0000
Total Times 100
Succ Rate 0.0200
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Training Patterns
Verification
Testing Patterns
Create 2-D Patterns

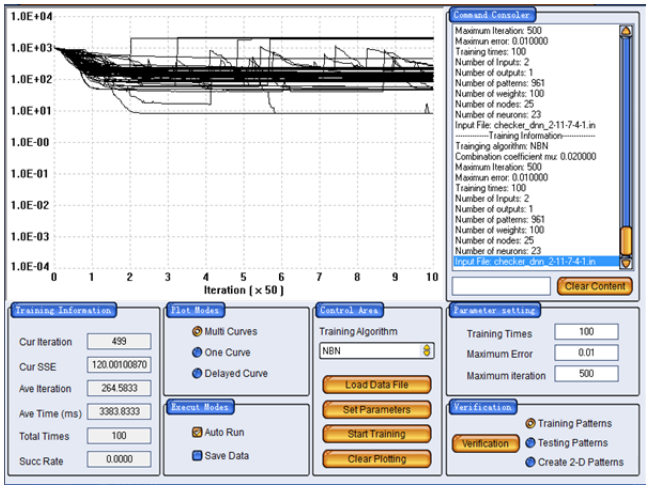

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.050000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 499
Cur SSE 8.00751174
Ave Iteration 297.0000
Ave Time (ms) 9960.5000
Total Times 100
Succ Rate 0.0200
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Training Patterns
Verification
Testing Patterns
Create 2-D Patterns

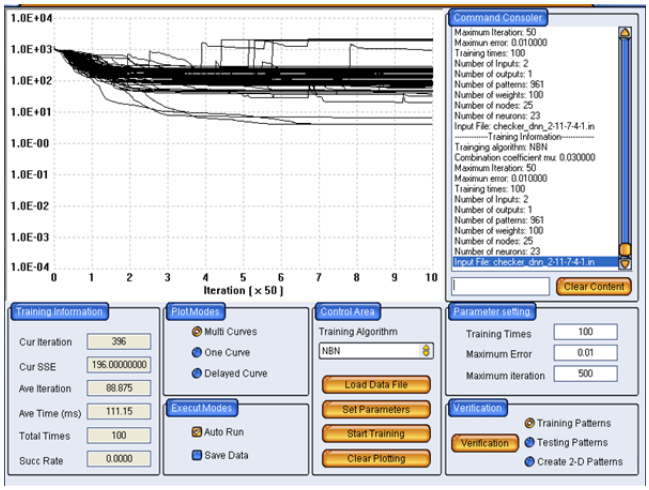
Appendix D

Supporting Data for Checker-3 Benchmark testing of equivalent networks

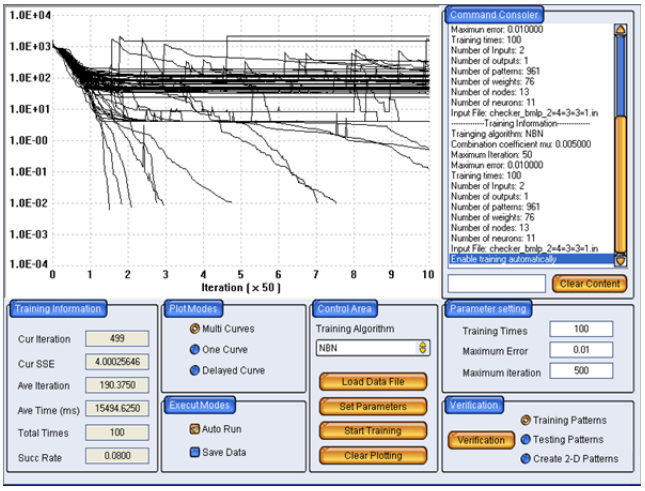
Best training data for MLP 2-11-7-4-1 network

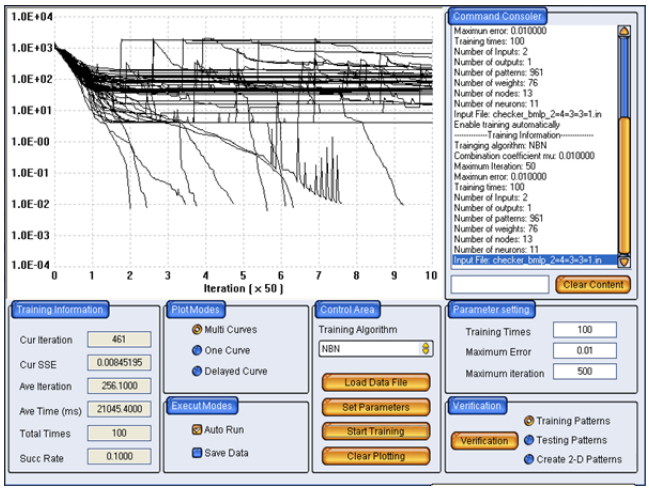


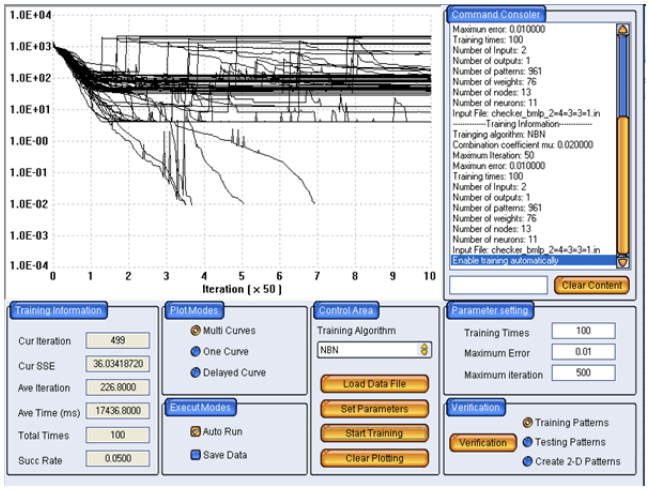


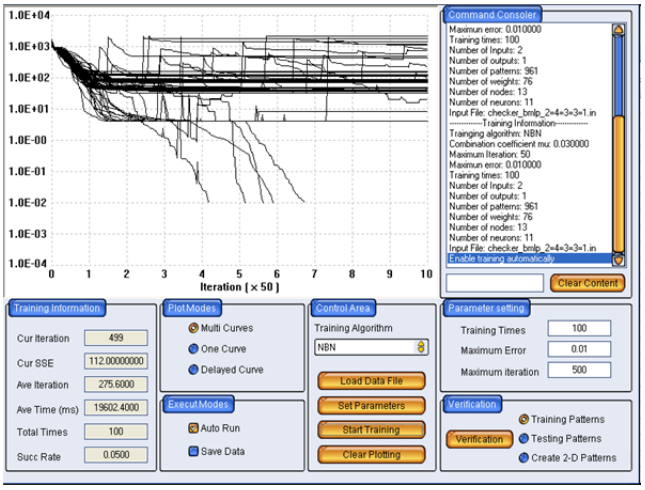


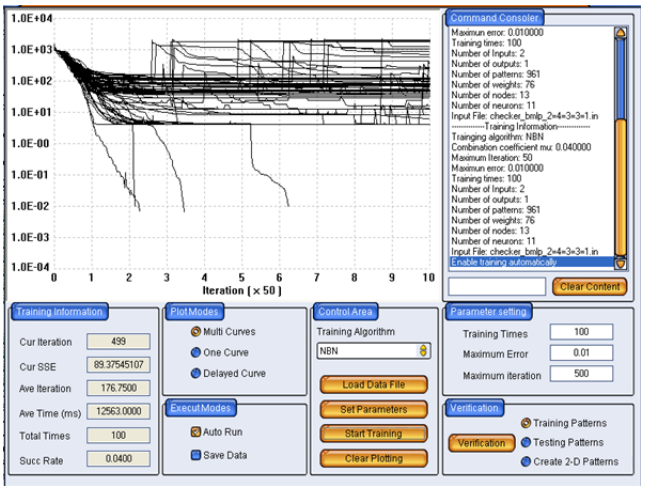
Best training data for BMLP 2=4=3=3=1 network

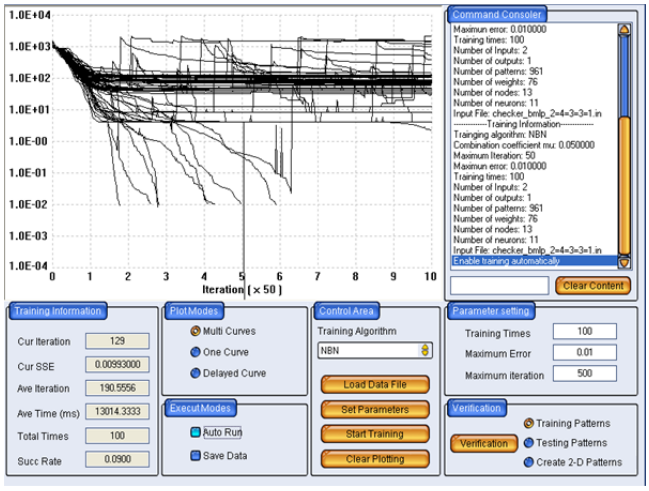


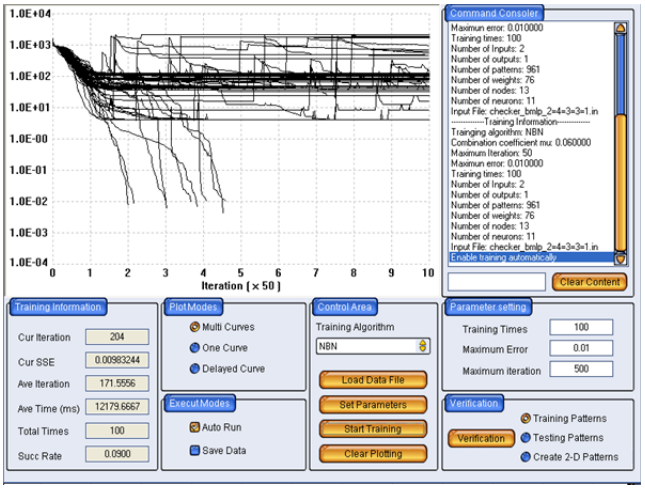


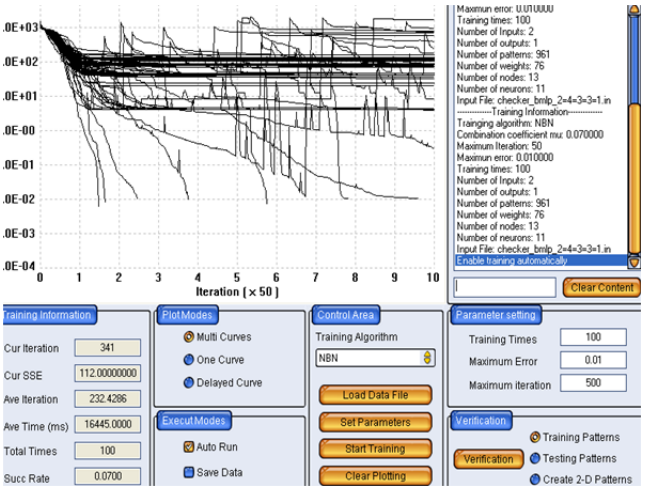

Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.020000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 499
Cur SSE 36.03418720
Ave Iteration 226.8000
Ave Time (ms) 17436.8000
Total Times 100
Succ Rate 0.0500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.030000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 499
Cur SSE 112.00000000
Ave Iteration 275.6000
Ave Time (ms) 19602.4000
Total Times 100
Succ Rate 0.0500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

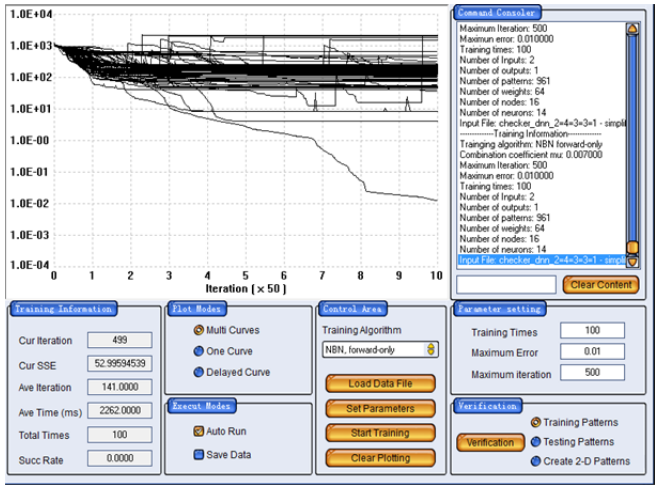

Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.040000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 499
Cur SSE 89.37545107
Ave Iteration 176.7500
Ave Time (ms) 12563.0000
Total Times 100
Succ Rate 0.0400
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

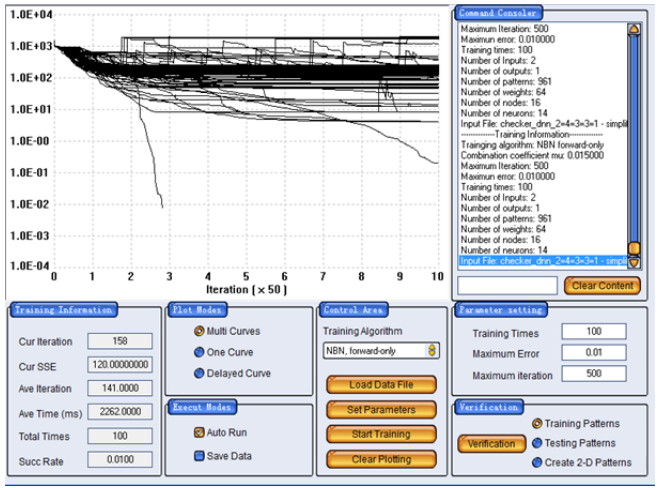

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.050000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 129
Cur SSE 0.00993000
Ave Iteration 190.5556
Ave Time (ms) 13014.3333
Total Times 100
Succ Rate 0.0900
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

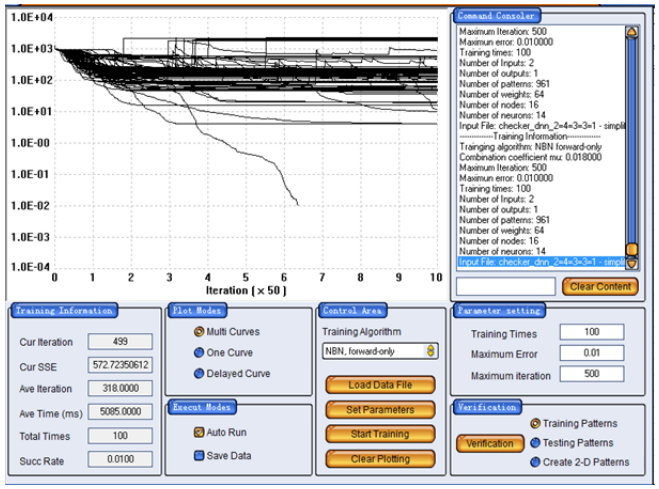

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.060000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 204
Cur SSE 0.00983244
Ave Iteration 171.5556
Ave Time (ms) 12179.6667
Total Times 100
Succ Rate 0.0900
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

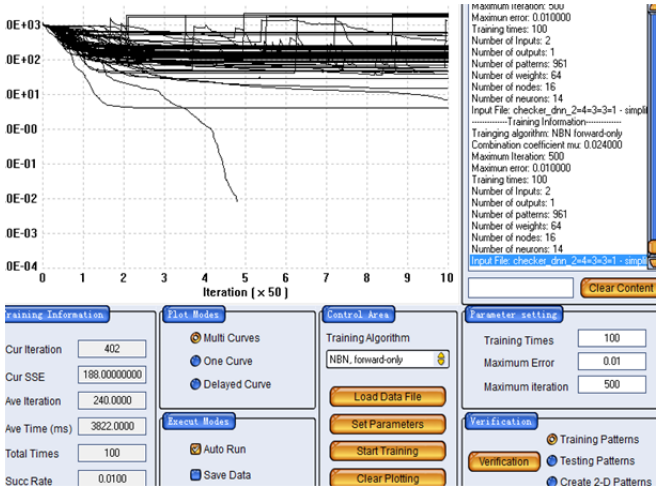

.0E+03
.0E+02
.0E+01
.0E-00
.0E-01
.0E-02
.0E-03
.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.070000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 341
Cur SSE 112.00000000
Ave Iteration 232.4286
Ave Time (ms) 16445.0000
Total Times 100
Succ Rate 0.0700
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

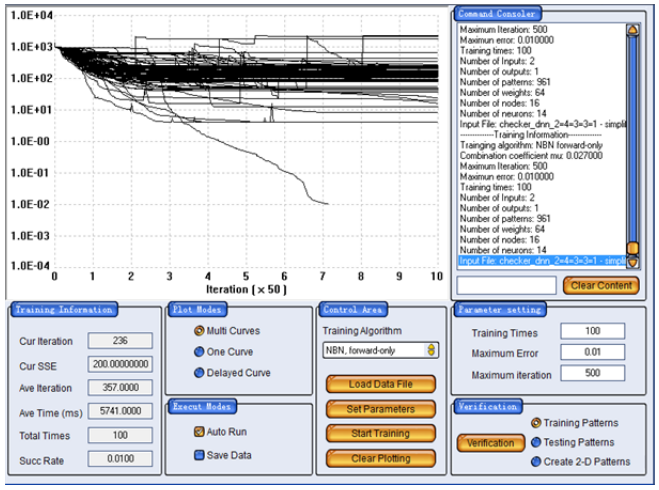
Best training data for DNN 2-4(7)-3(4)-3(1)-1 network

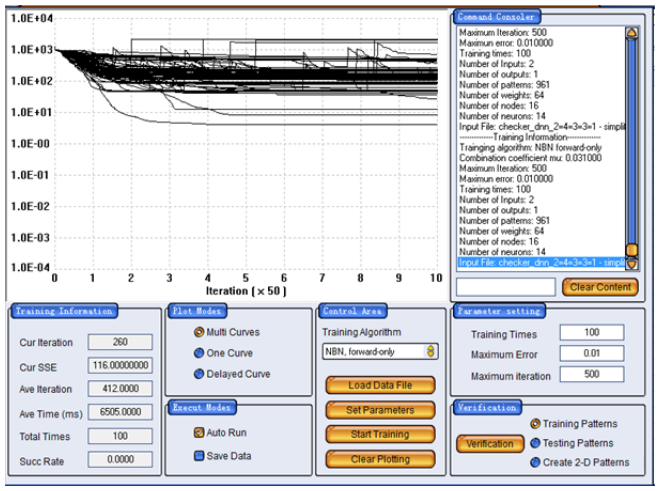







0E+03
0E+02
0E+01
0E-00
0E-01
0E-02
0E-03
0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.024000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpl
Clear Content
Training Information
Cur Iteration 402
Cur SSE 188.00000000
Ave Iteration 240.0000
Ave Time (ms) 3822.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

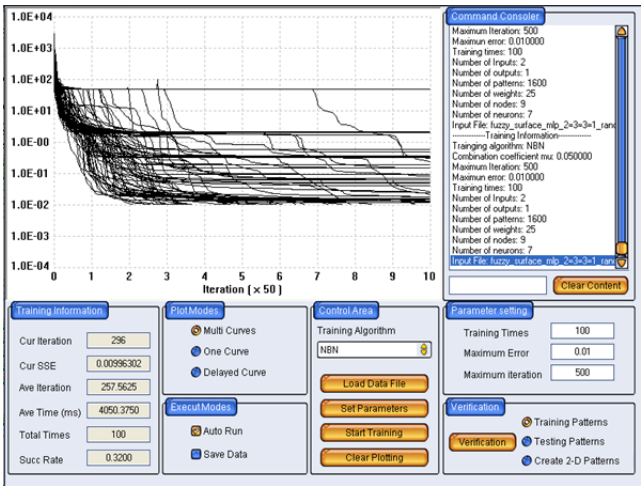

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.027000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
Clear Content
Training Information
Cur Iteration 236
Cur SSE 200.00000000
Ave Iteration 357.0000
Ave Time (ms) 5741.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.031000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpl
Clear Content
Training Information
Cur Iteration 260
Cur SSE 116.00000000
Ave Iteration 412.0000
Ave Time (ms) 6505.0000
Total Times 100
Succ Rate 0.0000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

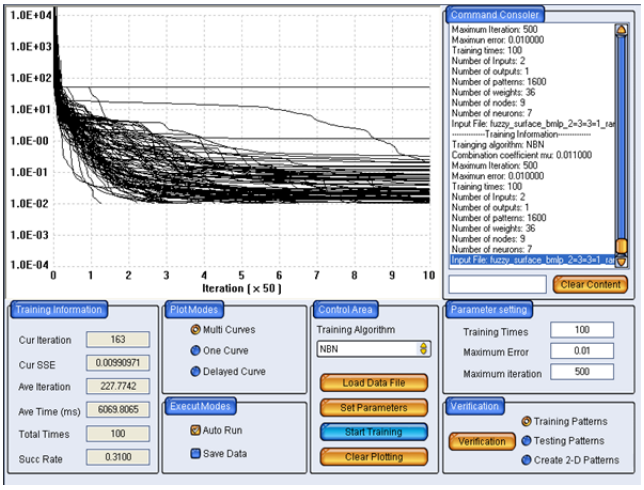
Appendix E

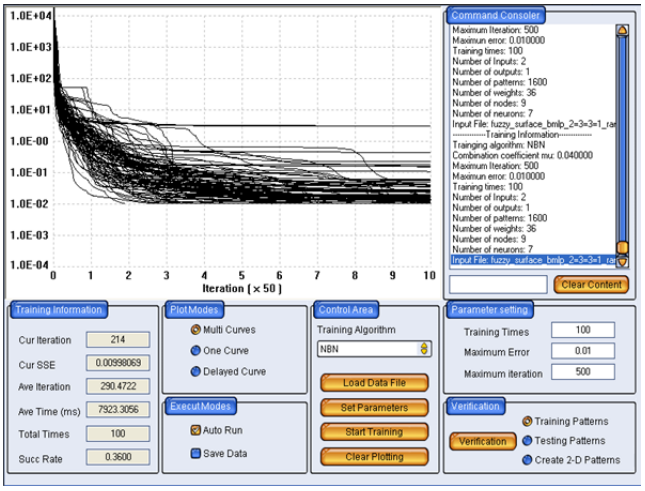
Supporting Data for 3-D Benchmark testing of minimal networks

Best training data for MLP 2-3-3-1 network

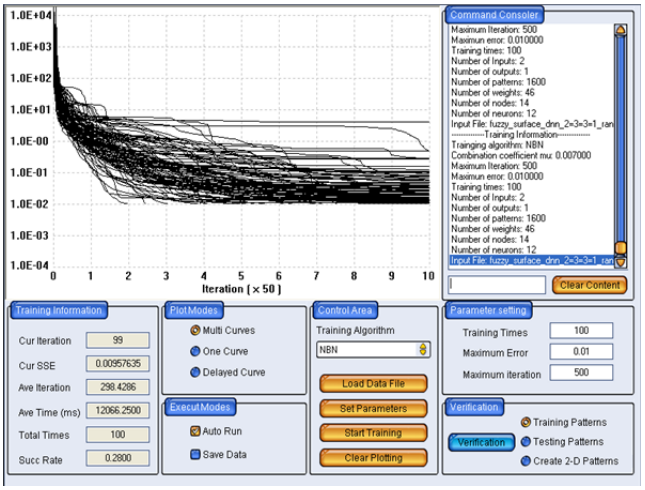


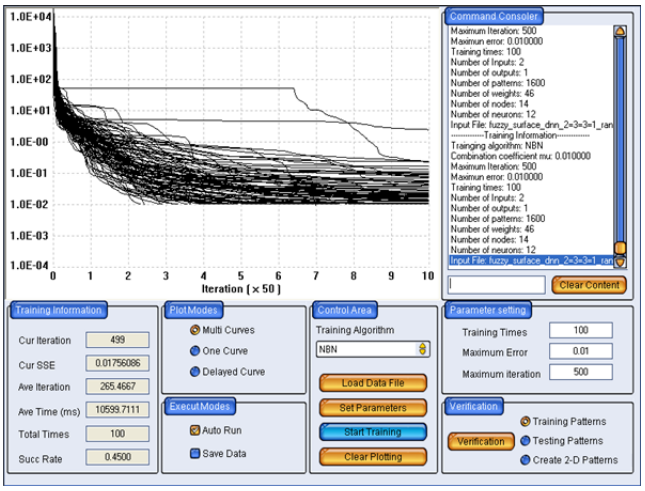
Best training data for BMLP 2=3=3=1 network





Best training data for DNN 2-3(4)-3(1)-1 network

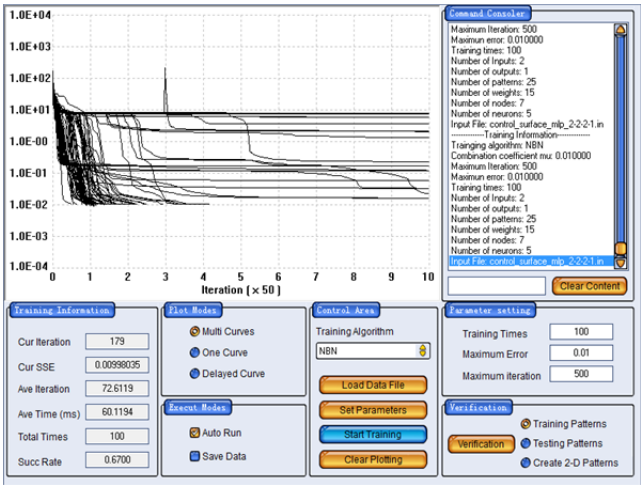


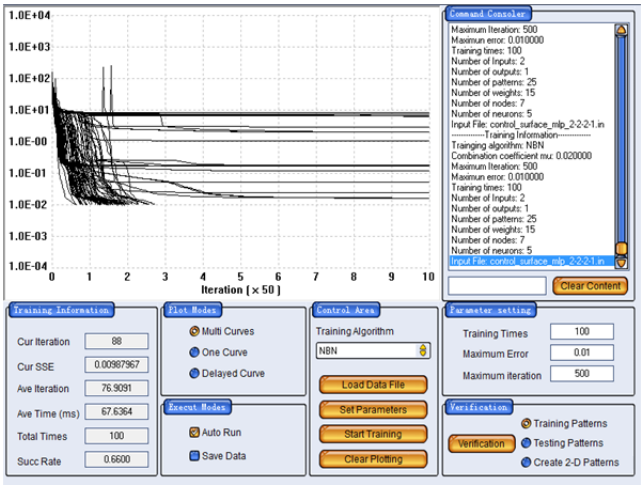


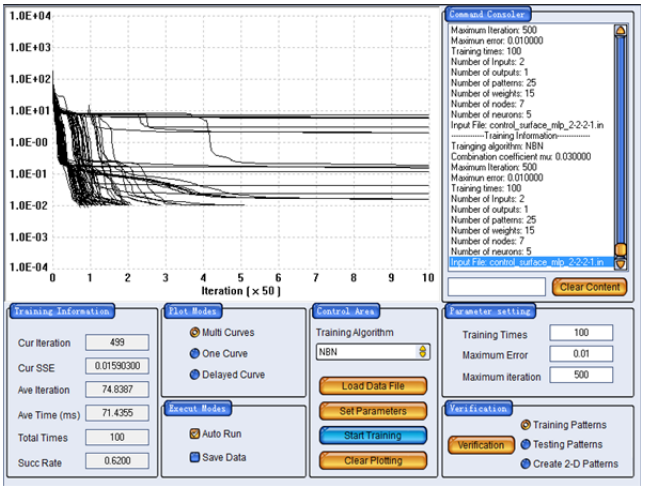
Appendix F

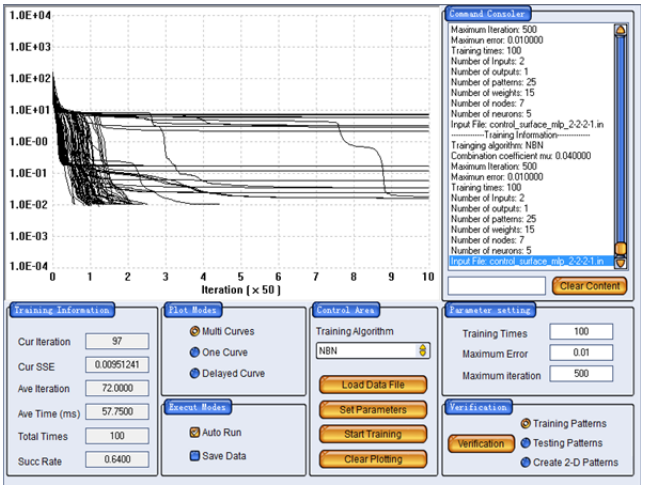
Supporting Data for Simple 3-D Benchmark testing of minimal networks

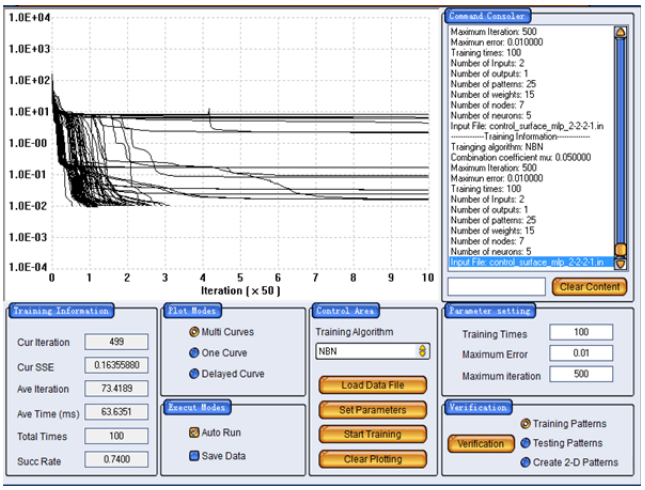
Best training data for MLP 2-2-2-1 network



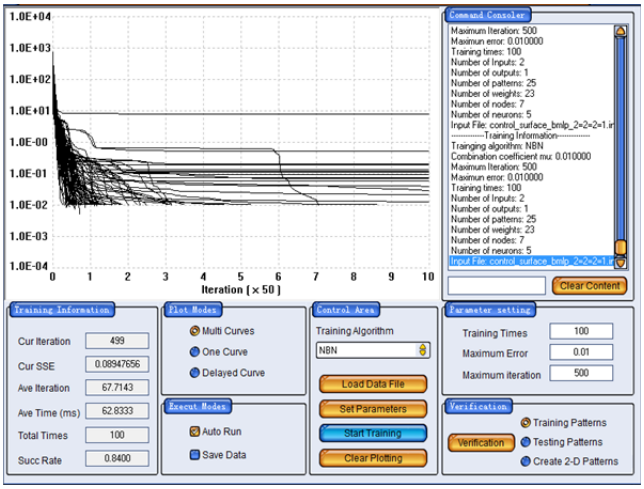


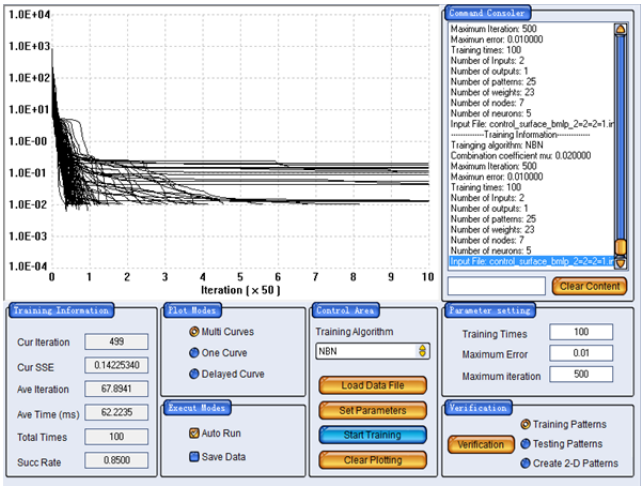

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 15
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_mlp_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.030000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 15
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_mlp_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.01590300
Ave Iteration 74.8387
Ave Time (ms) 71.4355
Total Times 100
Succ Rate 0.6200
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

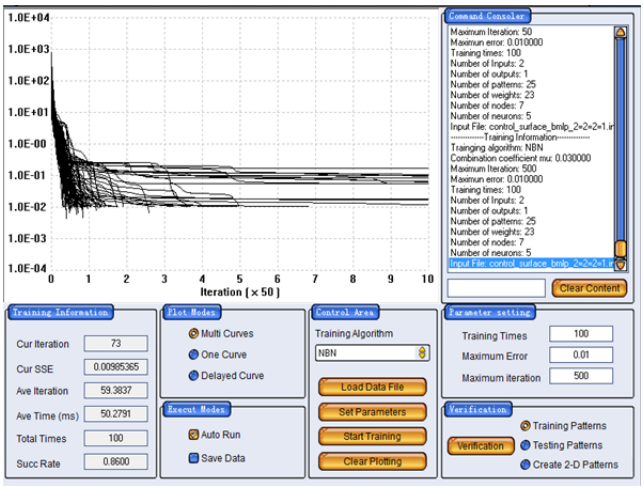

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 15
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_mlp_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.040000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 15
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_mlp_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 97
Cur SSE 0.00951241
Ave Iteration 72.0000
Ave Time (ms) 57.7500
Total Times 100
Succ Rate 0.6400
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

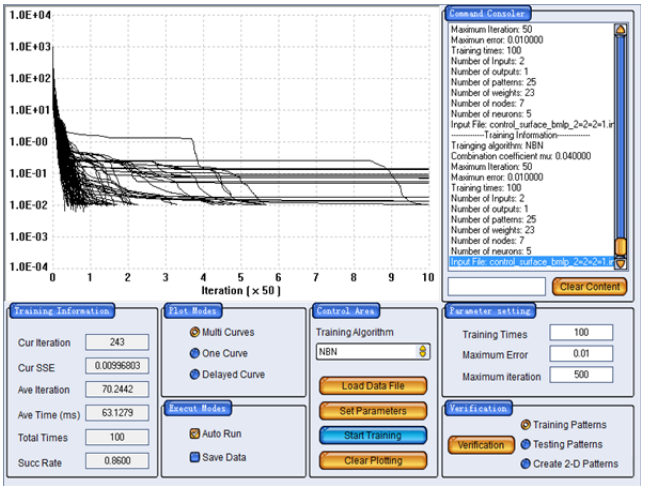

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 15
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_mlp_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.050000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 15
Number of nodes: 7
Number of neurons: 5
Input File: control_surface_mlp_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.16355880
Ave Iteration 73.4189
Ave Time (ms) 63.6351
Total Times 100
Succ Rate 0.7400
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

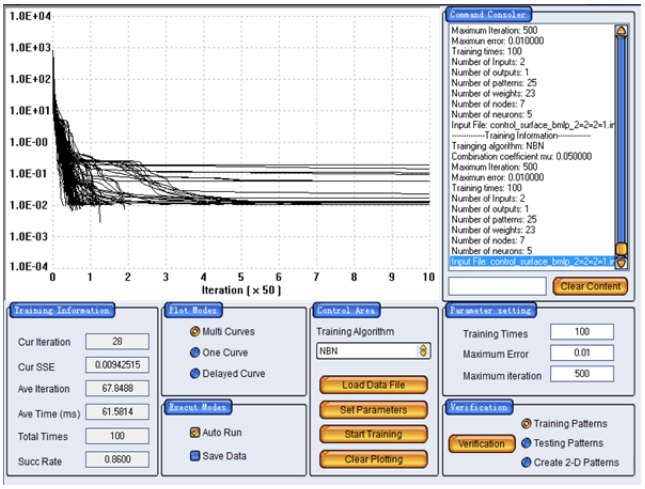
Best training data for BMLP 2=2=2=1 network



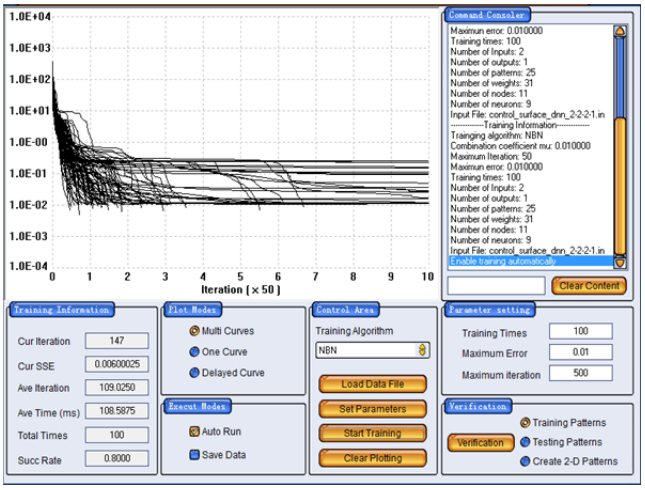


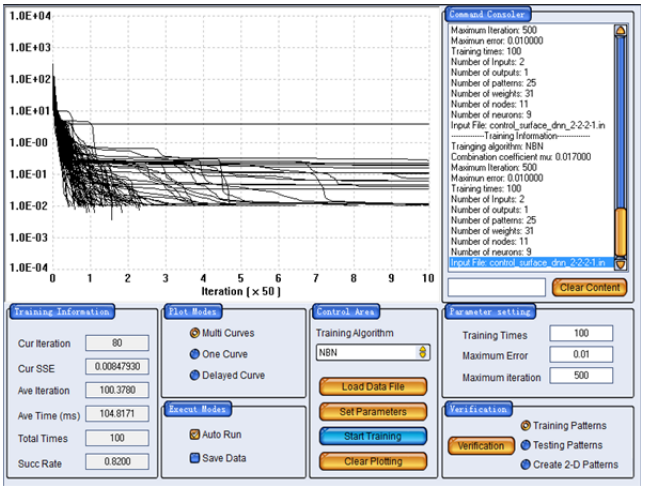


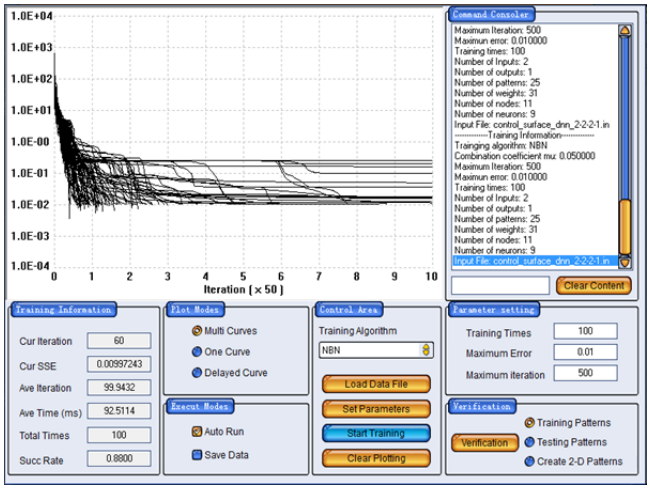


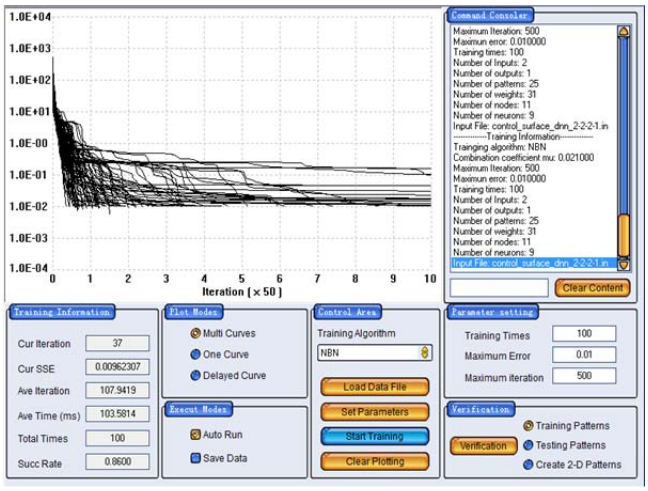


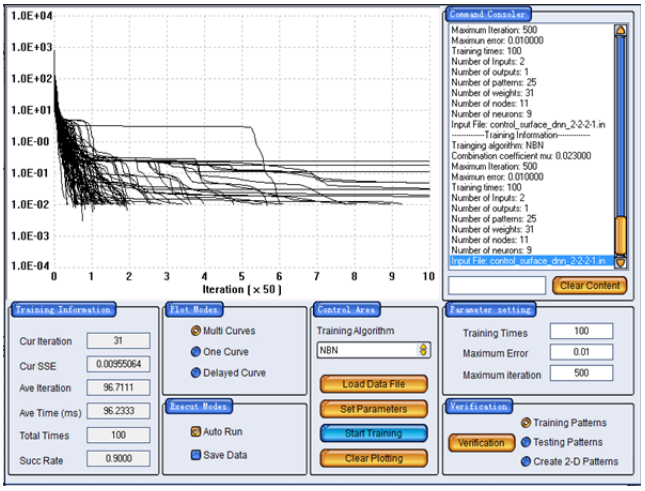
Best training data for DNN 2-2(3)-2(1)-1 network

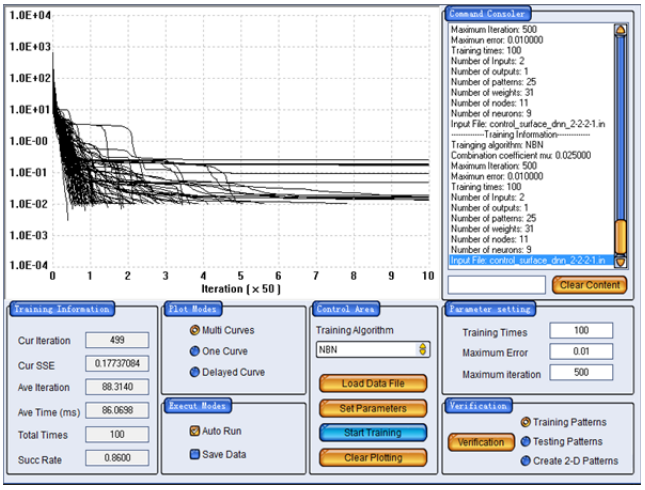


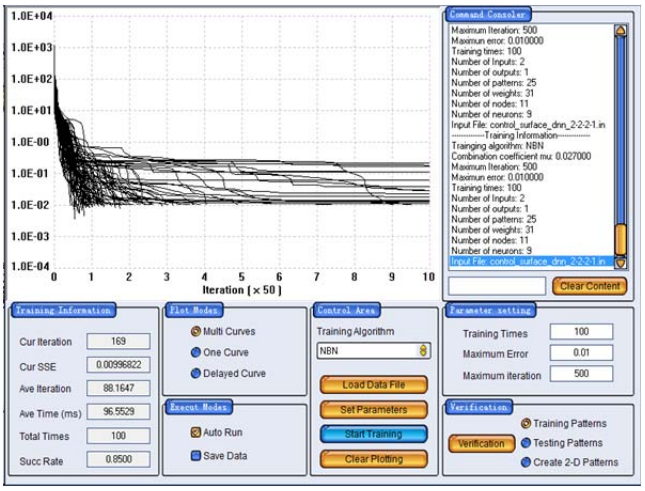

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.017000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 80
Cur SSE 0.00847930
Ave Iteration 100.3780
Ave Time (ms) 104.8171
Total Times 100
Succ Rate 0.8200
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

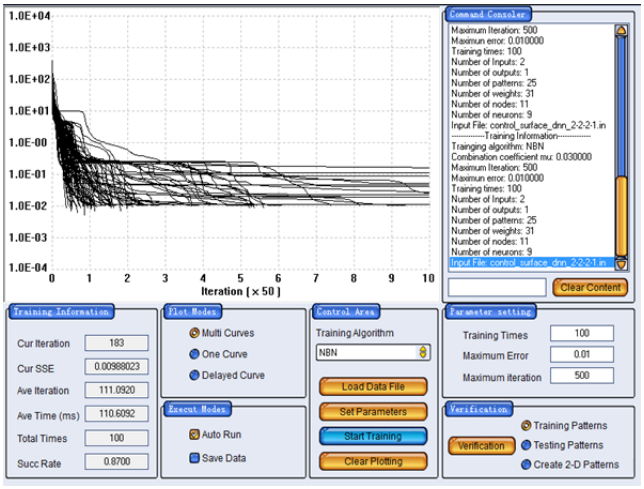

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.050000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 60
Cur SSE 0.00997243
Ave Iteration 99.9432
Ave Time (ms) 92.5114
Total Times 100
Succ Rate 0.8800
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

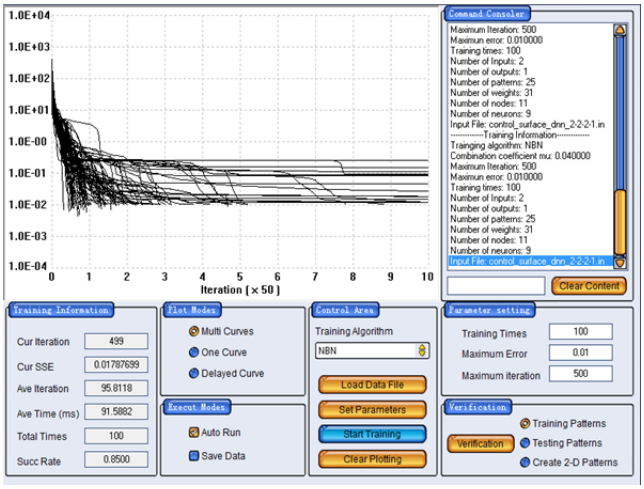

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.021000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 37
Cur SSE 0.00962307
Ave Iteration 107.9419
Ave Time (ms) 103.5814
Total Times 100
Succ Rate 0.8600
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

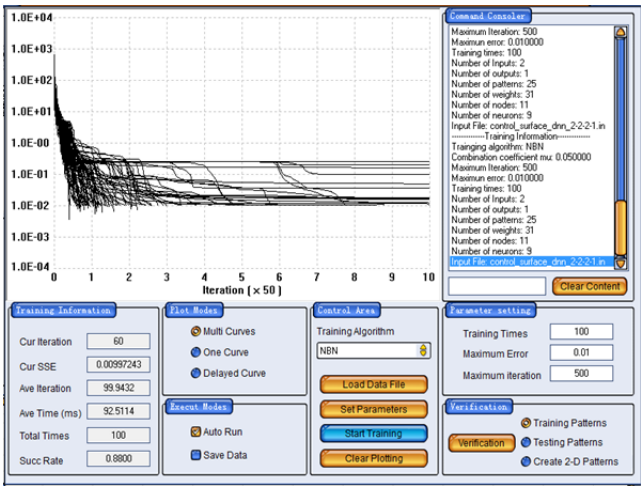

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.023000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 31
Cur SSE 0.00955064
Ave Iteration 96.7111
Ave Time (ms) 96.2333
Total Times 100
Succ Rate 0.9000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.025000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.17737084
Ave Iteration 88.3140
Ave Time (ms) 86.0698
Total Times 100
Succ Rate 0.9600
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.027000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 169
Cur SSE 0.00996822
Ave Iteration 88.1647
Ave Time (ms) 96.5529
Total Times 100
Succ Rate 0.8500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.030000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 183
Cur SSE 0.00988023
Ave Iteration 111.0920
Ave Time (ms) 110.6092
Total Times 100
Succ Rate 0.8700
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

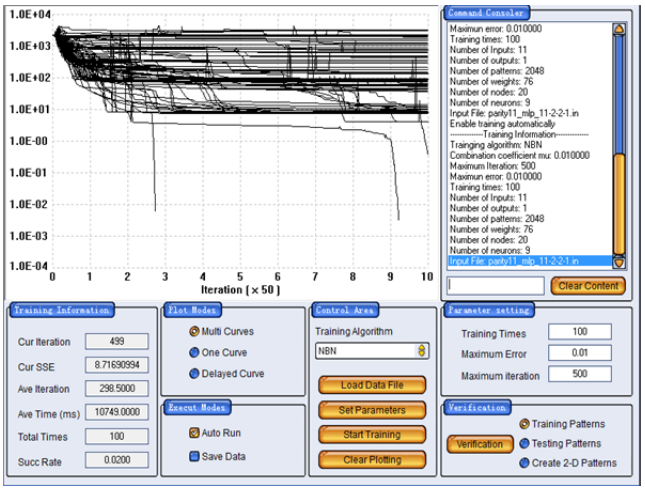

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.040000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 0.01787699
Ave Iteration 95.8118
Ave Time (ms) 91.5882
Total Times 100
Succ Rate 0.8500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

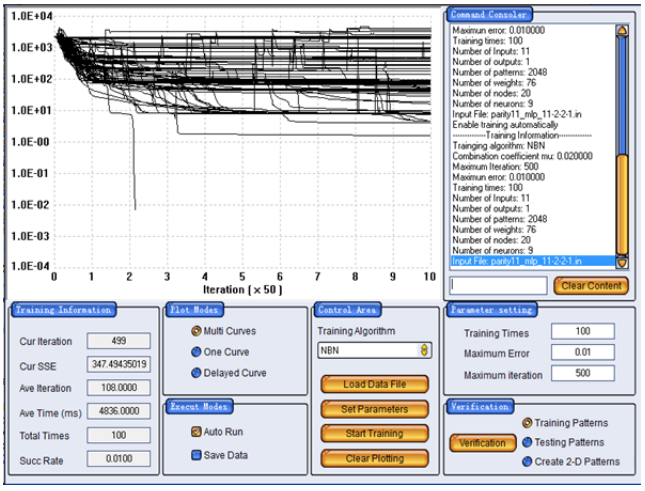

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.050000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 25
Number of weights: 31
Number of nodes: 11
Number of neurons: 9
Input File: control_surface_dnn_2-2-2-1.in
Clear Content
Training Information
Cur Iteration 60
Cur SSE 0.00997243
Ave Iteration 99.9432
Ave Time (ms) 92.5114
Total Times 100
Succ Rate 0.8800
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

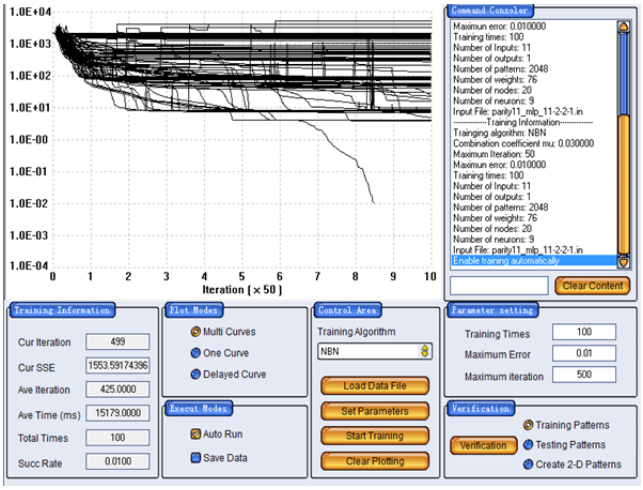
Appendix G

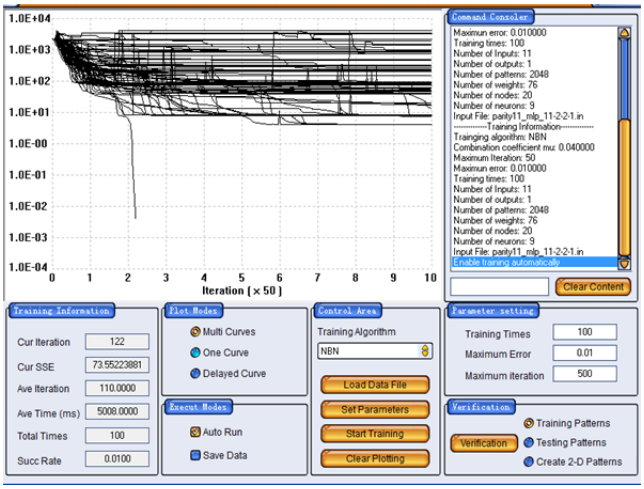
Supporting Data for Parity-11 Benchmark testing of minimal networks

Best training data for MLP 11-2-2-1 network

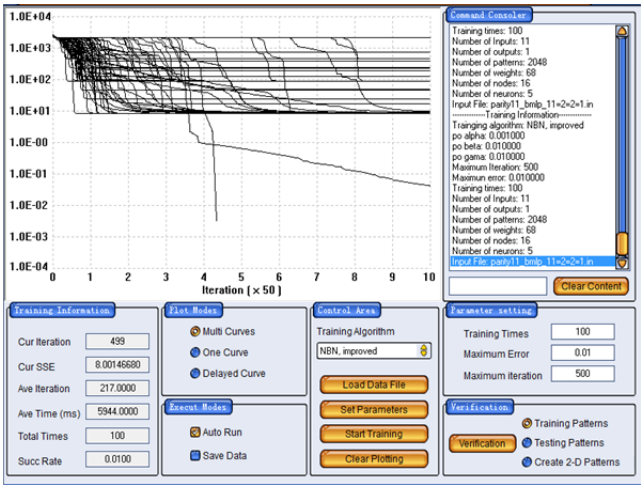


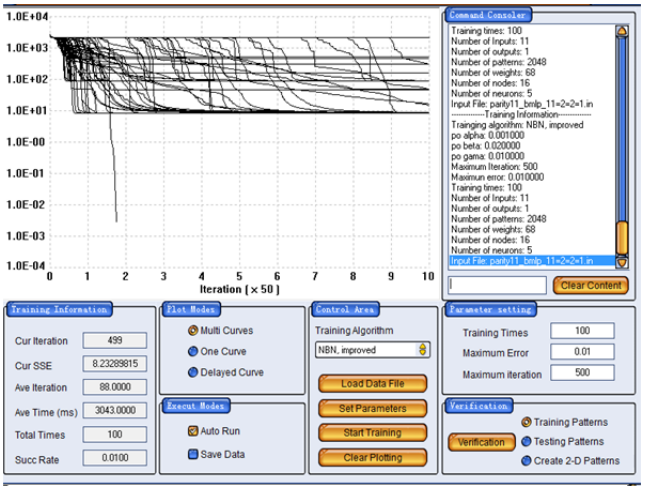


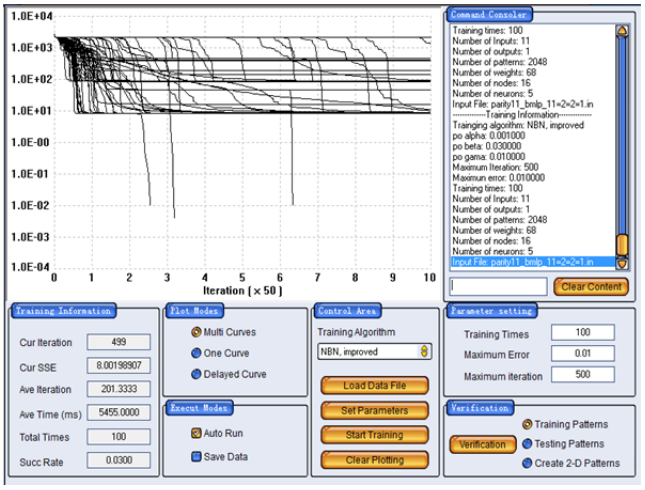


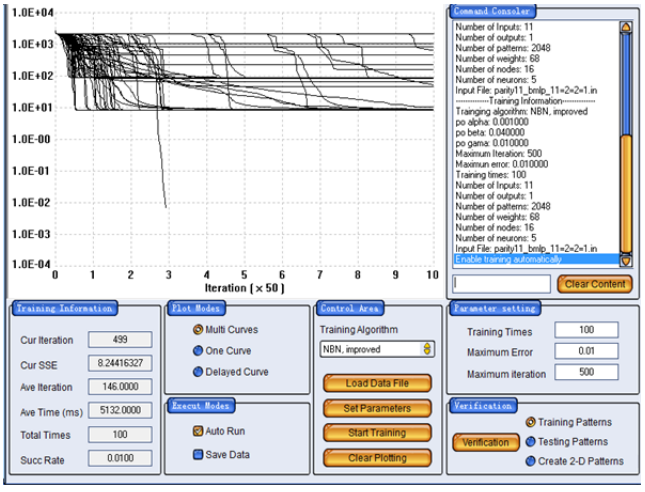


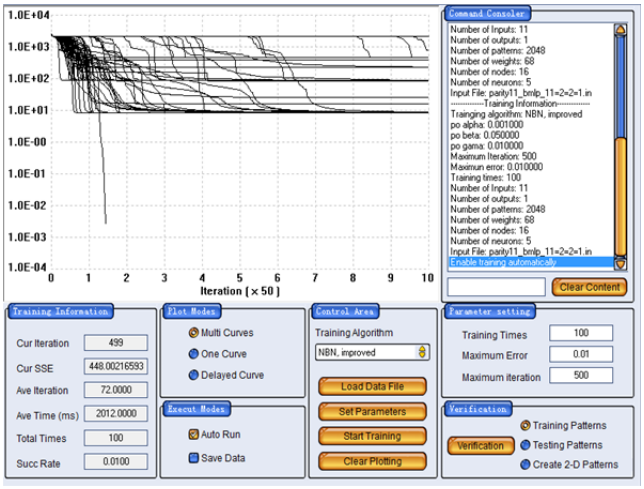
Best training data for BMLP 11=2=2=1 network



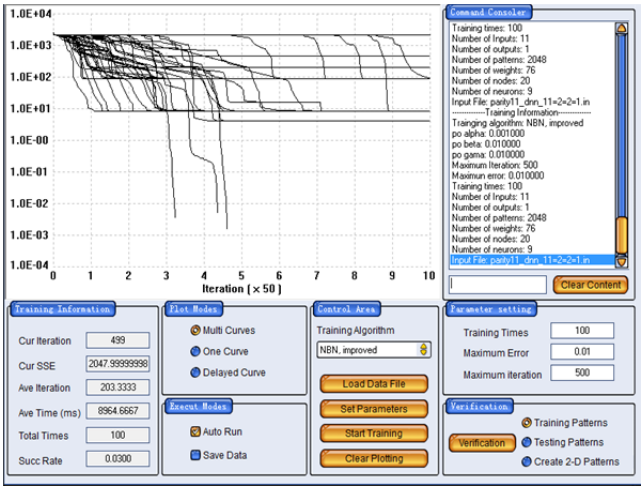

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2×2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.020000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2×2=1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 8.23289815
Ave Iteration 88.0000
Ave Time (ms) 3043.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

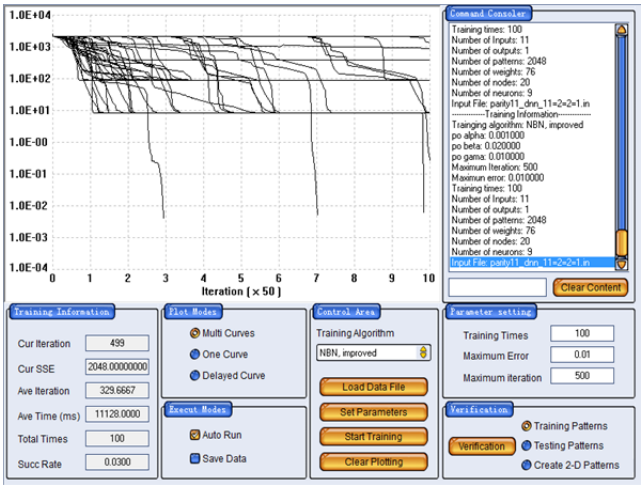

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2×2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.030000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2×2=1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 8.00198907
Ave Iteration 201.3333
Ave Time (ms) 5455.0000
Total Times 100
Succ Rate 0.0300
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

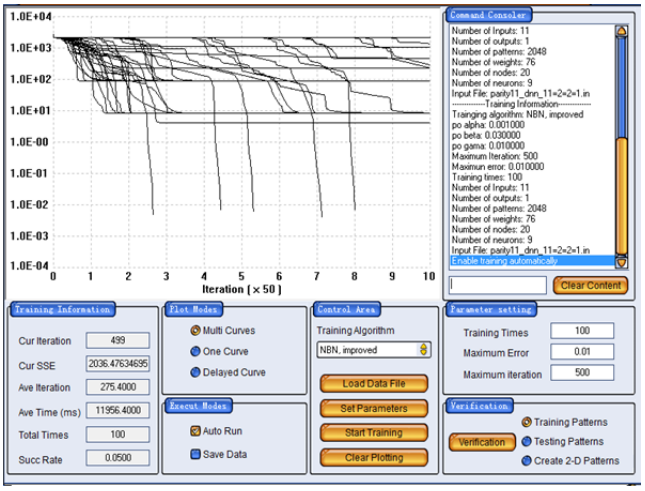

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2×2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.040000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 68
Number of nodes: 16
Number of neurons: 5
Input File: parity11_bmlp_11=2×2=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 499
Cur SSE 8.24416327
Ave Iteration 146.0000
Ave Time (ms) 5132.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

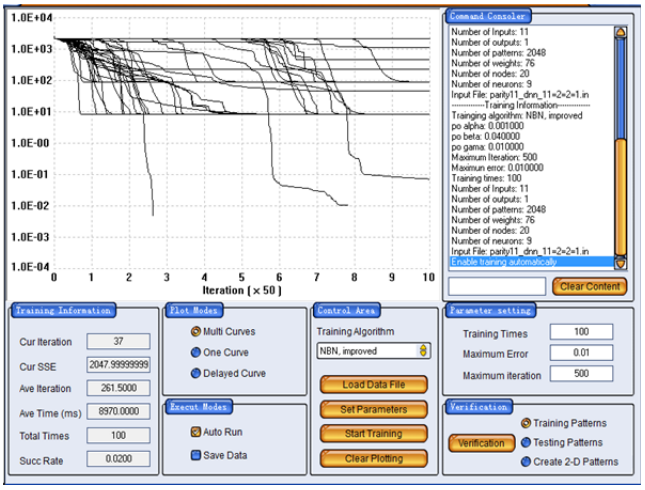


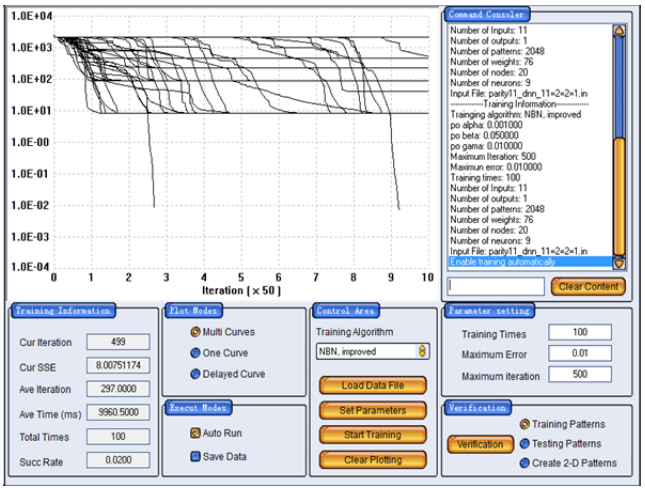
Best training data for DNN 11-2(3)-2(1)-1 network






1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.030000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 499
Cur SSE 2036.47634695
Ave Iteration 275.4000
Ave Time (ms) 11956.4000
Total Times 100
Succ Rate 0.0500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

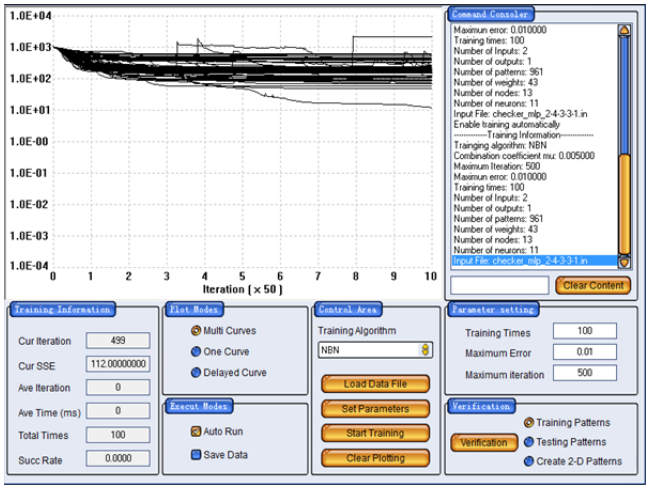

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.040000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 37
Cur SSE 2047.99999999
Ave Iteration 261.5000
Ave Time (ms) 8970.0000
Total Times 100
Succ Rate 0.0200
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

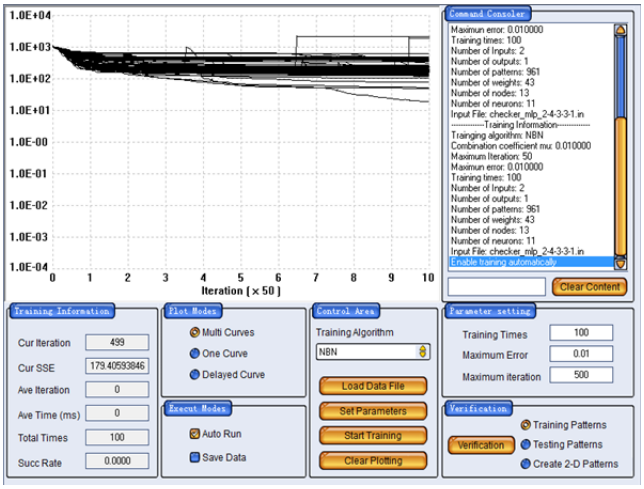

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
----------Training Information----------
Trainging algorithm: NBN, improved
po alpha: 0.001000
po beta: 0.050000
po gama: 0.010000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 11
Number of outputs: 1
Number of patterns: 2048
Number of weights: 76
Number of nodes: 20
Number of neurons: 9
Input File: parity11_dnn_11=2=2=1.in
Enable training automatically
Clear Content
Training Information
Cur Iteration 499
Cur SSE 8.00751174
Ave Iteration 297.0000
Ave Time (ms) 9960.5000
Total Times 100
Succ Rate 0.0200
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, improved
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

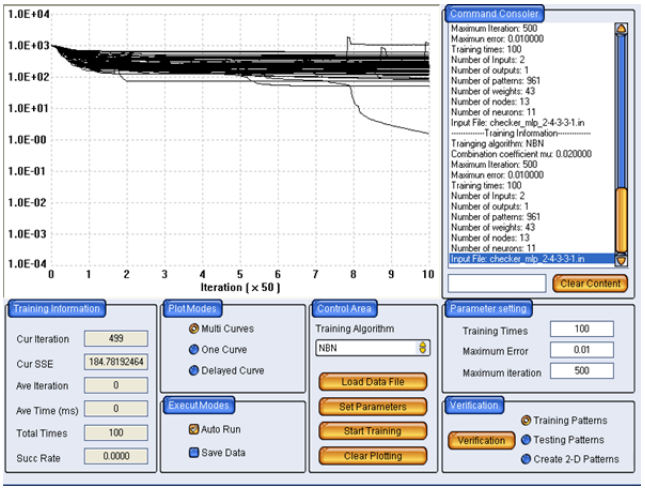
Appendix H

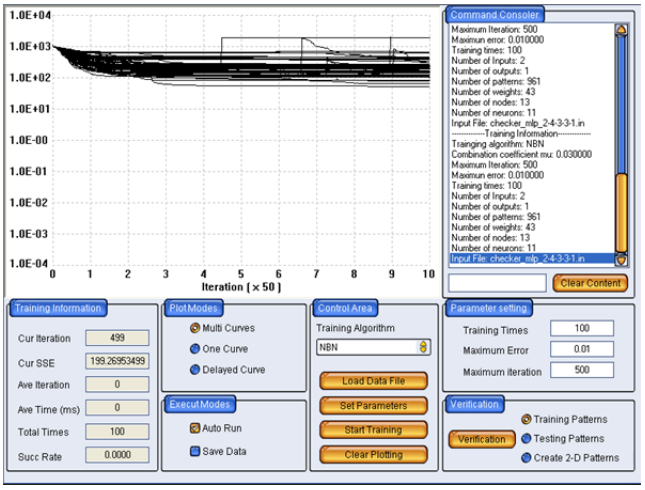
Supporting Data for Checker-N Benchmark testing of minimal networks

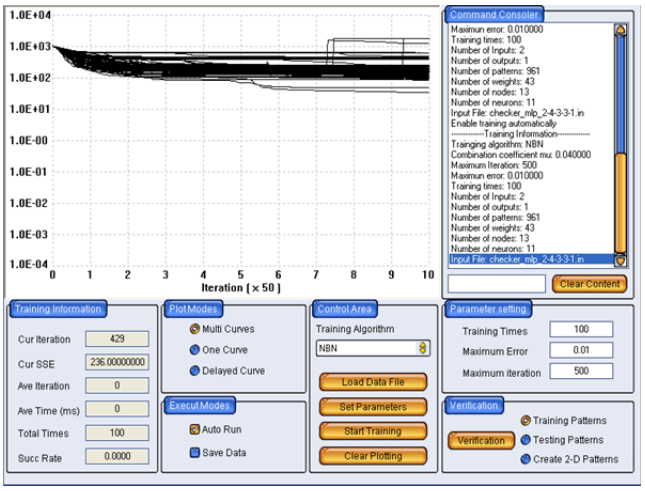
Best training data for MLP 2-4-3-3-1 network

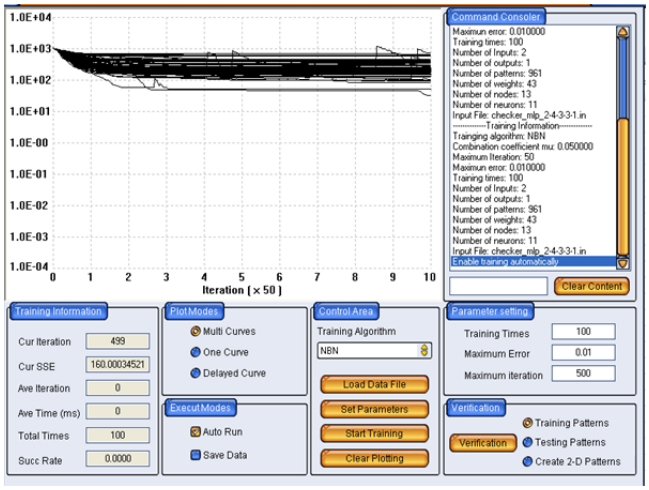




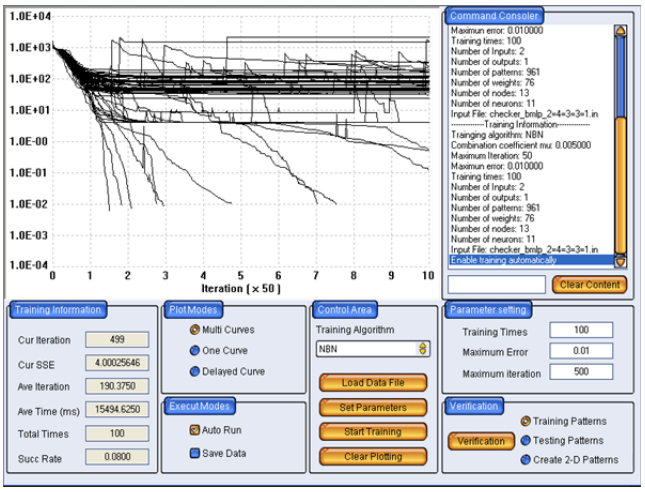

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 43
Number of nodes: 13
Number of neurons: 11
Input File: checker_mlp_2-4-3-3-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.020000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 43
Number of nodes: 13
Number of neurons: 11
Input File: checker_mlp_2-4-3-3-1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 184.78192464
Ave Iteration 0
Ave Time (ms) 0
Total Times 100
Succ Rate 0.0000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

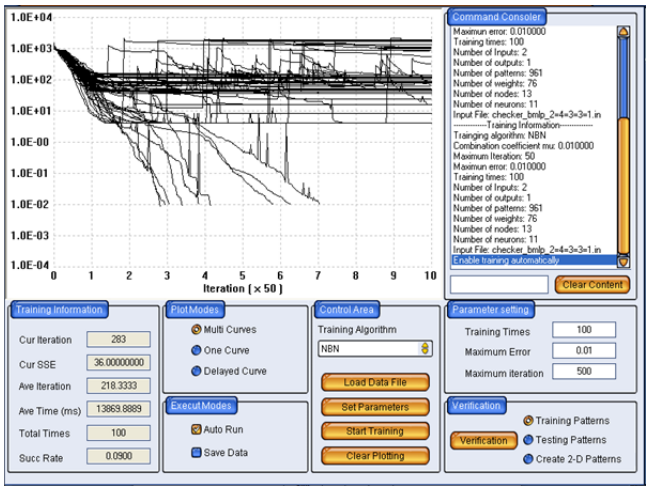

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 43
Number of nodes: 13
Number of neurons: 11
Input File: checker_mlp_2-4-3-3-1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.030000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 43
Number of nodes: 13
Number of neurons: 11
Input File: checker_mlp_2-4-3-3-1.in
Clear Content
Training Information
Cur Iteration 499
Cur SSE 199.26953499
Ave Iteration 0
Ave Time (ms) 0
Total Times 100
Succ Rate 0.0000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

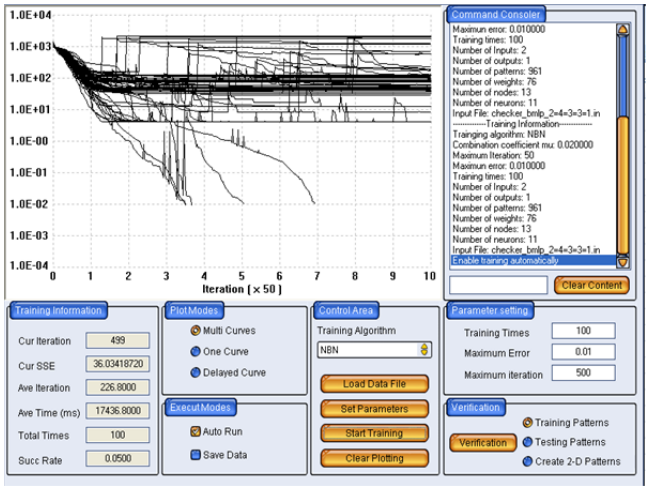

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (x 50)
Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 43
Number of nodes: 13
Number of neurons: 11
Input File: checker_mlp_2-4-3-3-1.in
Enable training automatically
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.040000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 43
Number of nodes: 13
Number of neurons: 11
Input File: checker_mlp_2-4-3-3-1.in
Clear Content
Training Information
Cur Iteration 429
Cur SSE 236.00000000
Ave Iteration 0
Ave Time (ms) 0
Total Times 100
Succ Rate 0.0000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

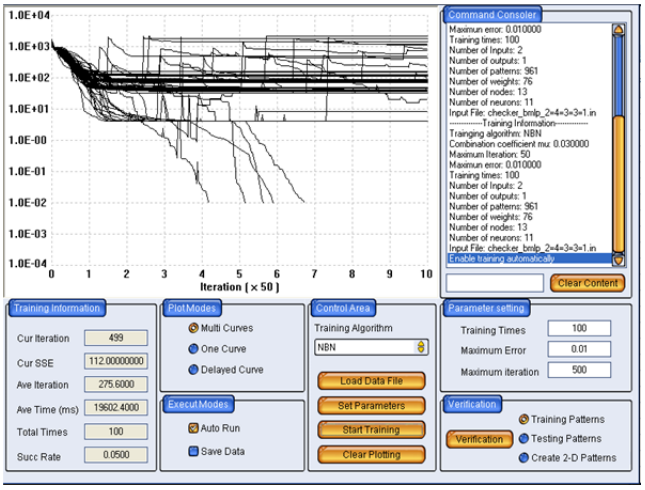


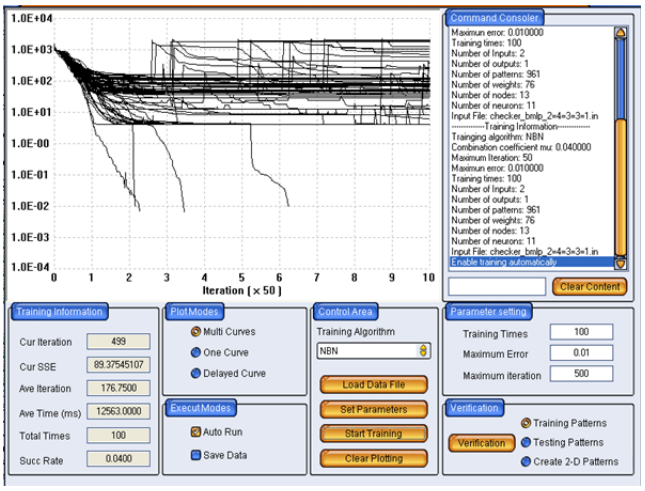
Best training data for BMLP 2=4=3=3=1 network

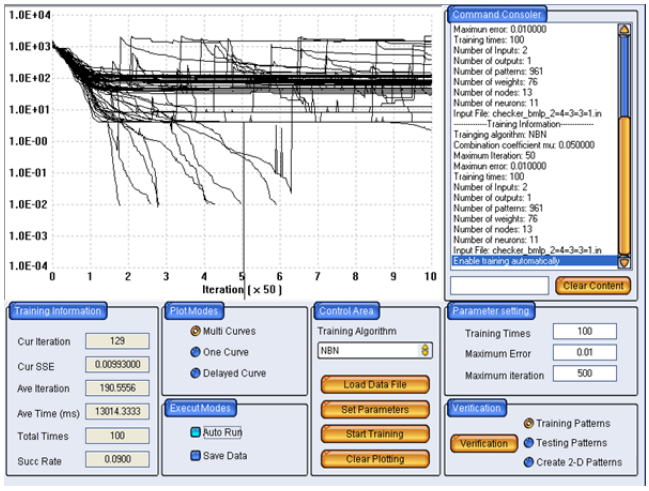


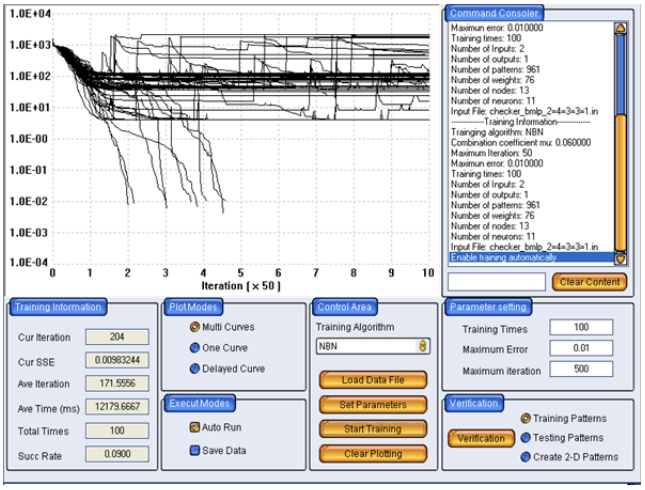


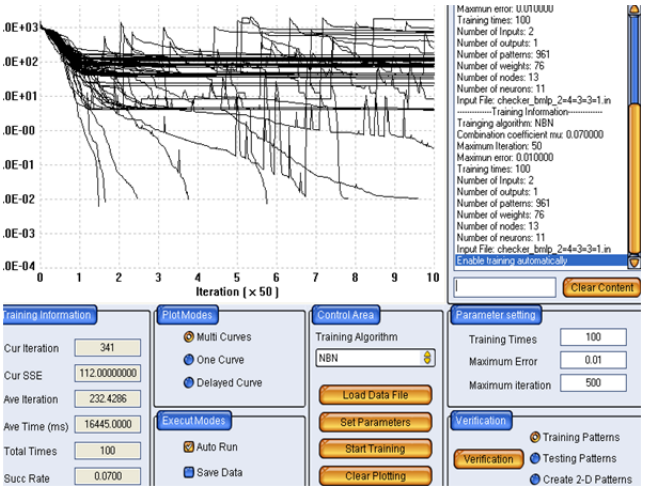

Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.020000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 499
Cur SSE 36.03418720
Ave Iteration 226.8000
Ave Time (ms) 17436.8000
Total Times 100
Succ Rate 0.0500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.030000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 499
Cur SSE 112.00000000
Ave Iteration 275.6000
Ave Time (ms) 19602.4000
Total Times 100
Succ Rate 0.0500
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

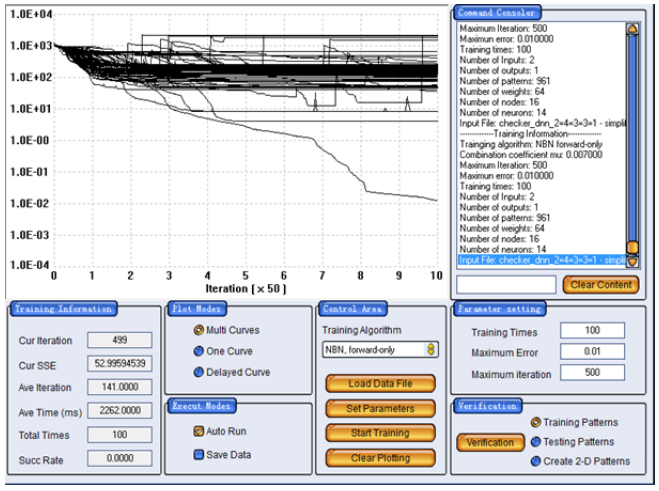

Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.040000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 499
Cur SSE 89.37545107
Ave Iteration 176.7500
Ave Time (ms) 12563.0000
Total Times 100
Succ Rate 0.0400
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

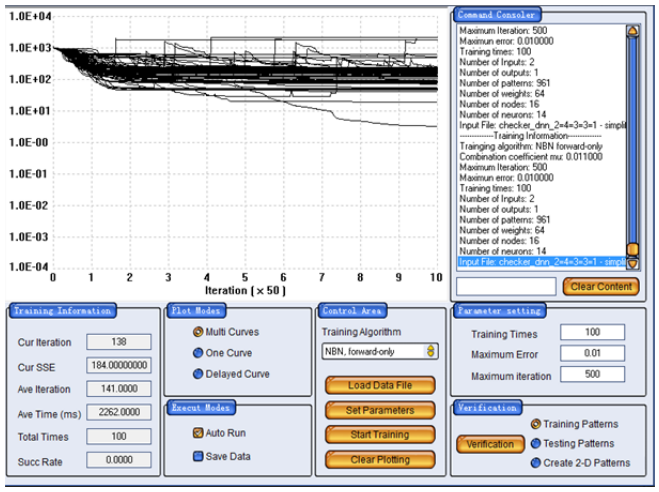

Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.050000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 129
Cur SSE 0.00993000
Ave Iteration 190.5556
Ave Time (ms) 13014.3333
Total Times 100
Succ Rate 0.0900
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

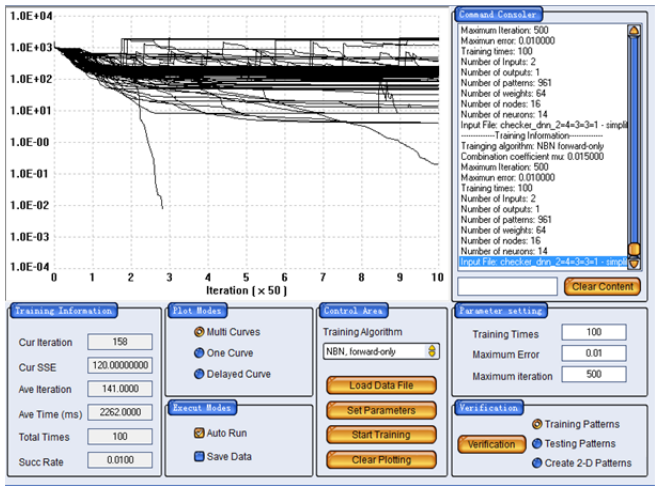

Command Consoler
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.060000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 204
Cur SSE 0.00983244
Ave Iteration 171.5556
Ave Time (ms) 12179.6667
Total Times 100
Succ Rate 0.0900
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

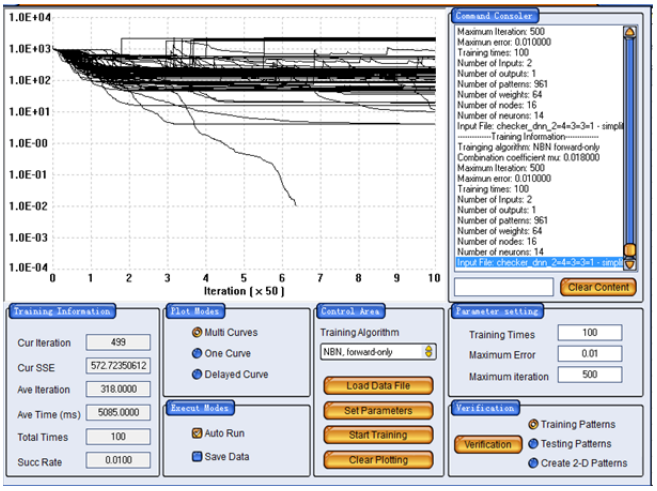

Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
----------Training Information----------
Trainging algorithm: NBN
Combination coefficient mu: 0.070000
Maximum Iteration: 50
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 76
Number of nodes: 13
Number of neurons: 11
Input File: checker_bmlp_2=4=3=3=1.in
Enable training automatically
Clear Content
Iteration (x 50)
Training Information
Cur Iteration 341
Cur SSE 112.00000000
Ave Iteration 232.4286
Ave Time (ms) 16445.0000
Total Times 100
Succ Rate 0.0700
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

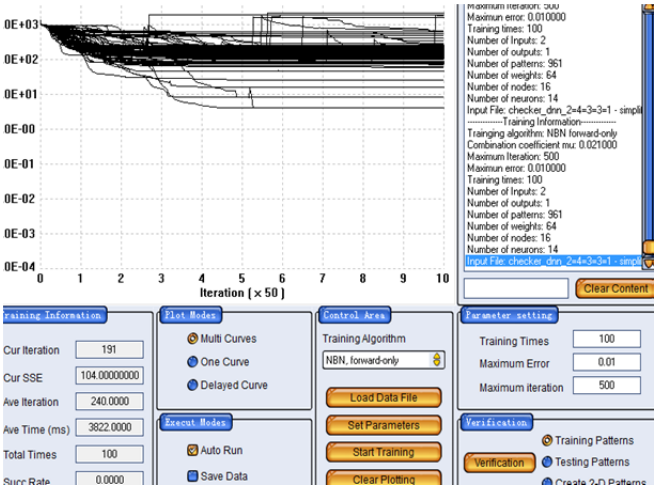
Best training data for DNN 2-4(7)-3(4)-3(1)-1 network

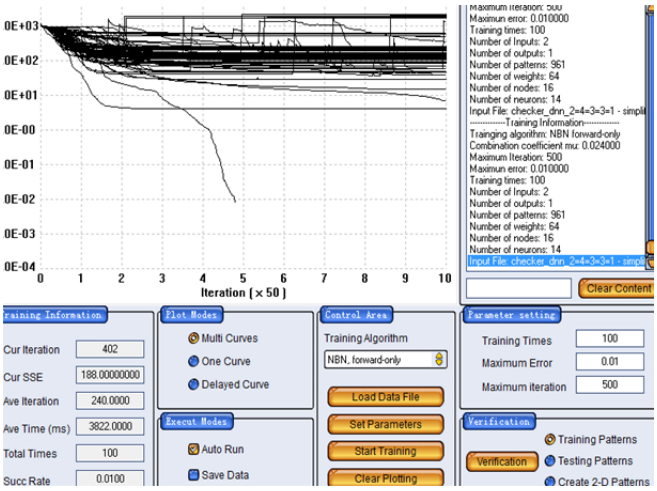


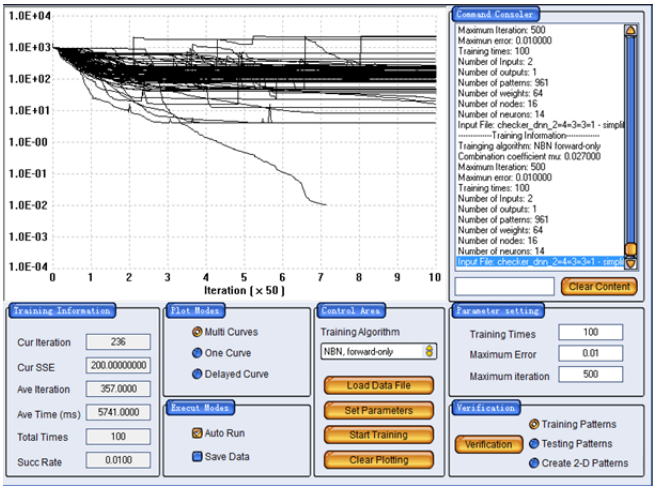


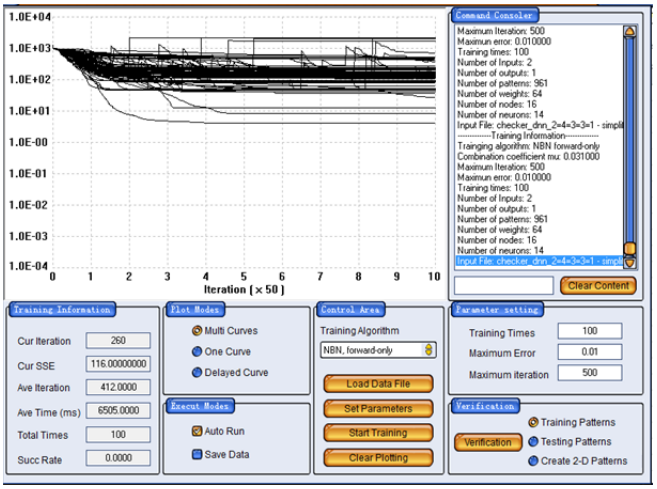


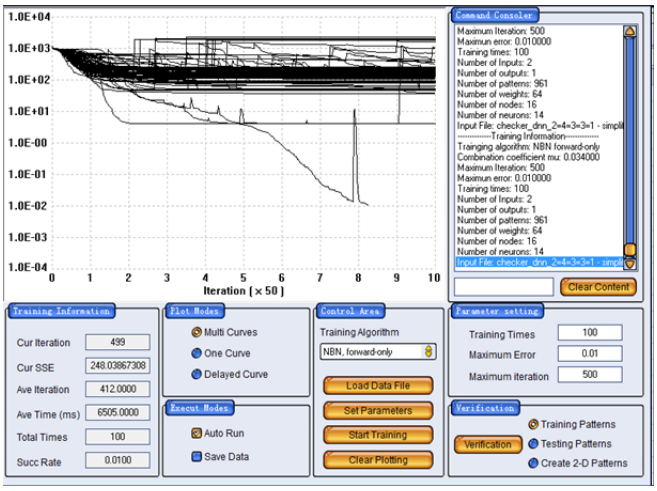

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.018000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
Clear Content
Training Information
Cur Iteration 499
Cur SSE 572.72350612
Ave Iteration 318.0000
Ave Time (ms) 5085.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns

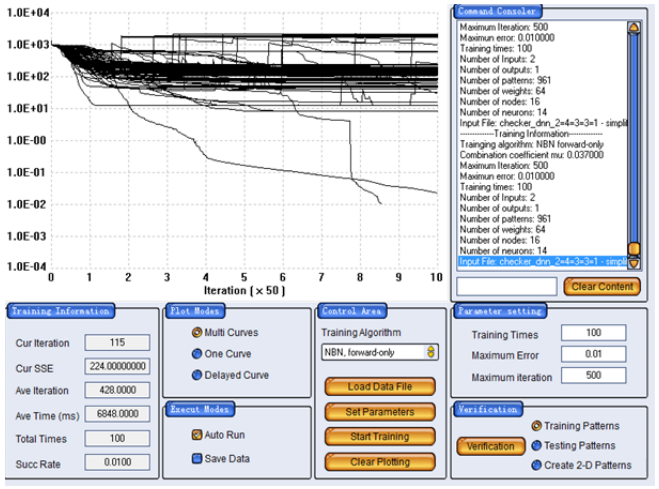

0E+03
0E+02
0E+01
0E-00
0E-01
0E-02
0E-03
0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.021000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
Clear Content
Cur Iteration 191
Cur SSE 104.00000000
Ave Iteration 240.0000
Ave Time (ms) 3822.0000
Total Times 100
Multi Curves
One Curve
Delayed Curve
Auto Run
Save Data
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Training Patterns
Testing Patterns
Verification

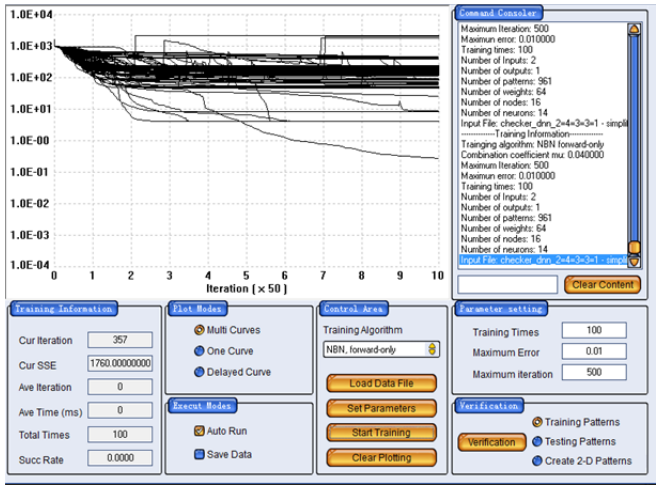

0E+03
0E+02
0E+01
0E-00
0E-01
0E-02
0E-03
0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.024000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
Clear Content
Cur Iteration 402
Cur SSE 188.00000000
Ave Iteration 240.0000
Ave Time (ms) 3822.0000
Total Times 100
Succ Rate 0.0100
Multi Curves
One Curve
Delayed Curve
Auto Run
Save Data
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Training Patterns
Testing Patterns
Verification

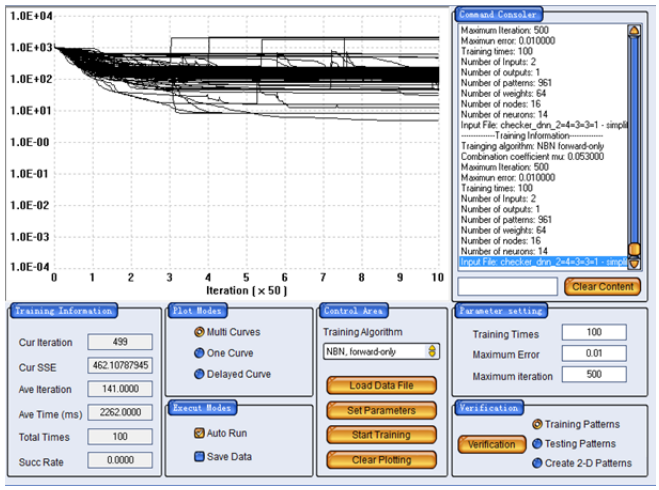

1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.027000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
Clear Content
Training Information
Cur Iteration 236
Cur SSE 200.00000000
Ave Iteration 357.0000
Ave Time (ms) 5741.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.031000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
Clear Content
Training Information
Cur Iteration 260
Cur SSE 116.00000000
Ave Iteration 412.0000
Ave Time (ms) 6505.0000
Total Times 100
Succ Rate 0.0000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.034000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simpli
Clear Content
Training Information
Cur Iteration 499
Cur SSE 248.03867308
Ave Iteration 412.0000
Ave Time (ms) 6505.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simple
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.037000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simple
Clear Content
Training Information
Cur Iteration 115
Cur SSE 224.00000000
Ave Iteration 428.0000
Ave Time (ms) 6848.0000
Total Times 100
Succ Rate 0.0100
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simple
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.040000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simple
Clear Content
Training Information
Cur Iteration 357
Cur SSE 1760.00000000
Ave Iteration 0
Ave Time (ms) 0
Total Times 100
Succ Rate 0.0000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns


1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E-00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
0 1 2 3 4 5 6 7 8 9 10
Iteration (× 50)
Command Consoler
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simple
----------Training Information----------
Trainging algorithm: NBN forward-only
Combination coefficient mu: 0.053000
Maximum Iteration: 500
Maximun error: 0.010000
Training times: 100
Number of Inputs: 2
Number of outputs: 1
Number of patterns: 961
Number of weights: 64
Number of nodes: 16
Number of neurons: 14
Input File: checker_dnn_2=4=3=3=1 - simple
Clear Content
Training Information
Cur Iteration 499
Cur SSE 462.10787945
Ave Iteration 141.0000
Ave Time (ms) 2262.0000
Total Times 100
Succ Rate 0.0000
Plot Modes
Multi Curves
One Curve
Delayed Curve
Execut Modes
Auto Run
Save Data
Control Area
Training Algorithm
NBN, forward-only
Load Data File
Set Parameters
Start Training
Clear Plotting
Parameter setting
Training Times 100
Maximum Error 0.01
Maximum iteration 500
Verification
Verification
Training Patterns
Testing Patterns
Create 2-D Patterns