INCREMENTAL NEURAL NETWORK CONSTRUCTION ALGORITHMS FOR TRAINING MULTILAYER PERCEPTRONS

by

Oya Aran

B.S., in Computer Engineering, Boğaziçi University, 2000

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

Boğaziçi University
2002

INCREMENTAL NEURAL NETWORK CONSTRUCTION ALGORITHMS FOR TRAINING MULTILAYER PERCEPTRONS

APPROVED BY:

Assoc. Prof. Ethem Alpaydın (Thesis Supervisor)	....
Assoc. Prof. Levent Akın	....
Prof. Günhan Dündar	....

DATE OF APPROVAL: 12.06.2002

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Assoc. Prof. Ethem Alpaydın. His guidance, encouragement, ideas and availability has made this thesis possible. Beginning from the Machine Learning course that I have taken at the last year of my undergraduate study, he is the person who guided me in determining my research direction.

I would also like to thank Assoc Prof. Levent Akın and Prof. Günhan Dündar for their suggestions on my thesis.

My special thanks must go to the behind-the-scenes people: my friends, my colleagues, and my family. I especially want to thank Onur Karakuş, without his support, patience and encouragement it would be harder to complete this thesis. I want to thank all the members of OliveOil group, without the short breaks full of fun and discussion, all day work would be unbearable. For sharing my stress especially at the very last phase of this thesis, I also want to thank my friend and colleague Onur Dikmen.

Finally, for their love and endless trust, I want to thank my family. Despite the kilometers between us, I always feel their love with me.

ABSTRACT

INCREMENTAL NEURAL NETWORK CONSTRUCTION ALGORITHMS FOR TRAINING MULTILAYER PERCEPTRONS

The problem of determining the architecture of a multilayer perceptron together with the disadvantages of the standard backpropagation algorithm, directed the research towards algorithms that determine not only the weights but also the structure of the network necessary for learning the data.

In this work we propose two algorithms: the Constructive Algorithm using Statistical Tests (CAST), and Constructive Algorithm with Multiple Operators using Statistical Tests (MOST). The first one constructs a single hidden layer network by adding hidden nodes one by one. The algorithm checks the difference between the errors of the current and candidate networks and decides whether to select the candidate network or not by using a statistical test for comparing the accuracies of the two networks. The networks that are constructed by MOST can have more than one hidden layer. The algorithm uses node removal, addition and layer addition and determines the number of nodes in layers by heuristics.

To our knowledge, MOST is the only algorithm that constructs a multilayer perceptron with multiple hidden layers with multiple units per layer.

The results of the algorithms are promising and near optimal.

ÖZET

YAPAY SİNİR AĞLARININ ÇOK KATMANLI ALGILAYICILAR İÇİN ARTIMLI OLUŞTURULMASI

Çok katmanlı algılayıcıların yapılarının belirlenmesi problemi ve standart geri yayılım algoritmasının sorunları bu konudaki çalısmaları bu yapıyı öğrenme sırasında belirleyebilen algoritmalar üstünde yoğunlaştırdı.

Bu çalışmada, çok katmanlı algılayıcıların yapılarını otomatik olarak belirleyen iki algoritma öneriyoruz: Birinci algoritma, CAST, saklı üniteleri birer birer ekleyerek tek saklı katmandan oluşan bir yapı oluşturur. İki yapıyı karşılaştırırken istatistiksel testleri kullanır. İkinci algoritma, MOST, birden fazla saklı katmanı olan ya da hiç saklı katmanı olmayan yapılar oluşturabilir. Üniteleri birer birer eklemek yerine, bir ya da daha fazla ünite çıkarmayı, eklemeyi ya da yeni bir saklı katman eklemeyi dener ve istatistiksel testleri kullanarak bir seçim yapar.

MOST algoritması ünite ekleme ve çıkarmaya hem de birden fazla saklı katmana izin veren tek algoritmadır.

Önerilen algoritmaların kullanılan standart veri kümeleri üzerindeki sonuçları ümit verici ve optimal çözüme yakındır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii
ABSTRACT iv
ÖZET v
LIST OF FIGURES viii
LIST OF TABLES xiii
LIST OF SYMBOLS/ABBREVIATIONS xv
1. INTRODUCTION 1
1.1. Outline of Thesis 3
2. MULTILAYER PERCEPTRONS 4
2.1. Linear Perceptron 5
2.1.1. Regression 5
2.1.2. Classification 6
2.2. Multilayer Perceptron 7
2.2.1. Regression 8
2.2.2. Classification 9
3. CONSTRUCTIVE ALGORITHMS FOR FINDING THE NEURAL NETWORK ARCHITECTURE 11
3.1. Training Techniques 11
3.1.1. Weight Freezing 12
3.1.2. Re–training the Whole Network Continuing from Previous Weights 12
3.1.3. Re–training the Whole Network from Scratch 13
3.2. Constructive Algorithms 13
3.2.1. Projection Pursuit Regression 13
3.2.2. Resource Allocating Network 14
3.2.3. Group Method of Data Handling 14
3.2.4. Upstart Algorithm 15
3.2.5. Cascade Correlation 15
3.2.6. Dynamic Node Creation 16
3.2.7. Constructive Algorithm for Real Valued Examples 18

3.2.8. Feedforward Neural Network Construction Using Cross Validation 18
4. PROPOSED METHODS 19
4.1. Modified Dynamic Node Creation 19
4.2. Incremental Neural Network Construction Using 5×2 cv F Test 21
4.2.1. Constructive Algorithm Using Statistical Tests 22
4.2.2. Applying Multiple Operators Using Statistical Tests 24
5. EXPERIMENTS 28
5.1. Datasets 28
5.1.1. Regression Datasets 28
5.1.2. Classification Datasets 29
5.2. Results 29
5.2.1. MLP Results 30
5.2.2. Modified DNC Results 31
5.2.3. CAST Results 37
5.2.4. MOST Results 38
5.2.5. Overall Comparison 41
6. CONCLUSIONS 46
APPENDIX A: PARAMETERS USED IN THE ALGORITHMS 47
APPENDIX B: MLP RESULTS 50
APPENDIX C: MODIFIED DNC RESULTS 56
APPENDIX D: CAST RESULTS 62
APPENDIX E: MOST RESULTS 65
APPENDIX F: CROSS VALIDATION RESULTS 68
REFERENCES 75

LIST OF FIGURES

Figure 2.1. A single perceptron 4

Figure 2.2. A linear perceptron 5

Figure 2.3. A multilayer perceptron 8

Figure 3.1. Types of constructive algorithms 13

Figure 3.2. Cascade-correlation architecture 15

Figure 3.3. The Cascade-correlation algorithm 16

Figure 3.4. Detecting the flattening of the error curve in DNC 17

Figure 3.5. The Dynamic Node Creation algorithm 18

Figure 4.1. Detecting the flattening of the error curve in modified DNC 20

Figure 4.2. The Modified DNC algorithm 22

Figure 4.3. The CAST algorithm 23

Figure 4.4. The MOST algorithm 26

Figure 4.5. Operator set used in the algorithm 27

Figure 4.6. Deciding between two architectures 27

Figure 5.1. The effect of the number of hidden units for `sin` dataset (dashed: test error, continous: training error) 31

Figure 5.2. The error graph during training of `sin` dataset in MLP 32

Figure 5.3. The effect of the number of hidden units for `ocr` dataset (dashed: test error, continous: training error) 33

Figure 5.4. The error graph during training of `ocr` dataset in MLP 33

Figure 5.5. The error graph during training of `sin` dataset with the DNC algorithm 34

Figure 5.6. The error graph during training of `sin` dataset with the modified DNC algorithm 34

Figure 5.7. A successful learning of `sin` dataset 35

Figure 5.8. A plot of a network that stucks in local minima of `sin` dataset . . 35

Figure 5.9. The error graph during training of `ocr` dataset with the DNC algorithm 36

Figure 5.10. The error graph during training of `ocr` dataset with the modified DNC algorithm 36

Figure 5.11. The error graph and the number of hidden nodes found for different confidence levels in `sin` dataset with CAST algorithm 37

Figure 5.12. The error graph and the number of hidden nodes found for different confidence levels in `ocr` dataset with CAST algorithm 38

Figure 5.13. Search for sin dataset in MOST algorithm 39

Figure 5.14. Search for ocr dataset in MOST algorithm 39

Figure 5.15. Search for penDigits dataset in MOST algorithm 40

Figure B.1. The effect of the number of hidden units for california dataset . 50

Figure B.2. The error graph during training of california dataset in MLP . . 50

Figure B.3. The effect of the number of hidden units for boston dataset 51

Figure B.4. The error graph during training of boston dataset in MLP 51

Figure B.5. The effect of the number of hidden units for pum8fh dataset 52

Figure B.6. The error graph during training of pum8fh dataset in MLP 52

Figure B.7. The effect of the number of hidden units for pum8nh dataset 53

Figure B.8. The error graph during training of pum8nh dataset in MLP 53

Figure B.9. The effect of the number of hidden units for optDigits dataset . . 54

Figure B.10. The error graph during training of optDigits dataset in MLP . . 54

Figure B.11. The effect of the number of hidden units for penDigits dataset . . 55

Figure B.12. The error graph during training of penDigits dataset in MLP . . 55

Figure C.1. The error graph during training of california dataset with the DNC algorithm 56

Figure C.2. The error graph during training of `california` dataset with the modified DNC algorithm 56

Figure C.3. The error graph during training of `boston` dataset with the DNC algorithm 57

Figure C.4. The error graph during training of `boston` dataset with the modified DNC algorithm 57

Figure C.5. The error graph during training of `pum8fh` dataset with the DNC algorithm 58

Figure C.6. The error graph during training of `pum8fh` dataset with the modified DNC algorithm 58

Figure C.7. The error graph during training of `pum8nh` dataset with the DNC algorithm 59

Figure C.8. The error graph during training of `pum8nh` dataset with the modified DNC algorithm 59

Figure C.9. The error graph during training of `optDigits` dataset with the DNC algorithm 60

Figure C.10. The error graph during training of `optDigits` dataset with the modified DNC algorithm 60

Figure C.11. The error graph during training of `penDigits` dataset with the DNC algorithm 61

Figure C.12. The error graph during training of `penDigits` dataset with the modified DNC algorithm 61

Figure D.1. The error graph and the number of hidden nodes found for different confidence levels in `california` dataset in CAST algorithm 62

Figure D.2. The error graph and the number of hidden nodes found for different confidence levels in `boston` dataset in CAST algorithm 62

Figure D.3. The error graph and the number of hidden nodes found for different confidence levels in `pum8fh` dataset in CAST algorithm 63

Figure D.4. The error graph and the number of hidden nodes found for different confidence levels in `pum8nh` dataset in CAST algorithm 63

Figure D.5. The error graph and the number of hidden nodes found for different confidence levels in `optDigits` dataset in CAST algorithm 64

Figure D.6. The error graph and the number of hidden nodes found for different confidence levels in `penDigits` dataset in CAST algorithm 64

Figure E.1. Search for `california` dataset in MOST algorithm 65

Figure E.2. Search for `boston` dataset in MOST algorithm 65

Figure E.3. Search for `pum8fh` dataset in MOST algorithm 66

Figure E.4. Search for `pum8nh` dataset in MOST algorithm 66

Figure E.5. Search for `optDigits` dataset in MOST algorithm 67

LIST OF TABLES

Table 4.1. An example for determining the number of hidden units 26

Table 5.1. Datasets used in experiments 30

Table 5.2. 5×2 cv results for sin dataset 42

Table 5.3. 5×2 cv F test results for sin dataset 43

Table 5.4. 5×2 cv results for ocr dataset 43

Table 5.5. 5×2 cv F test results for ocr dataset 44

Table 5.6. Overall results 45

Table 5.7. Newman-Keuls range test results 45

Table A.1. Parameter set used in sin dataset 47

Table A.2. Parameter set used in california dataset 47

Table A.3. Parameter set used in boston dataset 48

Table A.4. Parameter set used in pum8fh dataset 48

Table A.5. Parameter set used in pum8nh dataset 48

Table A.6. Parameter set used in ocr dataset 49

Table A.7. Parameter set used in optDigits dataset 49

Table A.8. Parameter set used in `penDigits` dataset 49

Table F.1. 5×2 cv results for `california` dataset 68

Table F.2. 5×2 cv F test results for `california` dataset 69

Table F.3. 5×2 cv results for `boston` dataset 69

Table F.4. 5×2 cv F test results for `boston` dataset 70

Table F.5. 5×2 cv results for `pum8fh` dataset 70

Table F.6. 5×2 cv F test results for `pum8fh` dataset 71

Table F.7. 5×2 cv results for `pum8nh` dataset 71

Table F.8. 5×2 cv F test results for `pum8nh` dataset 72

Table F.9. 5×2 cv results for `optDigits` dataset 72

Table F.10. 5×2 cv F test results for `optDigits` dataset 73

Table F.11. 5×2 cv results for `penDigits` dataset 73

Table F.12. 5×2 cv F test results for `penDigits` dataset 74

LIST OF SYMBOLS/ABBREVIATIONS

C_e	Cutoff for average squared error
Δ_r	Trigger slope
E_t	Average squared error at epoch t
$\bar{E}_t$	Average of average squared errors in epochs $t-w$ and t
o_i	The computed intermediate value of ith output
r_i	The desired value of ith output
t	Current epoch
t_0	The epoch when the last node was added to the hidden layer
v_{ij}	Weight from hidden unit j to output i
w	Width of window over which trigger slope is determined
w_{jk}	Weight from input k to hidden unit j
x_k	k^{th} dimension of the input
y_i	The computed value of ith output
α	Momentum constant
ϵ	Noise
η	Learning rate
ANN	Artificial Neural Network
`boston`	Boston housing dataset
`california`	California housing dataset
CARVE	Constructive Algorithm for Real Valued Examples
CAST	Constructive Algorithm using Statistical Tests
DNC	Dynamic Node Creation
exp	Exponential function
GMDH	Group Method of Data Handling
LP	Linear perceptron
MLP	Multilayer perceptron
MOST	Multiple Operators using Statistical Tests

`ocr`	OCR dataset
`optDigits`	Optical Digits dataset
`penDigits`	Pen Digits dataset
PPR	Projection Pursuit Regression
`pum8fh`	Pumadyn 8fh dataset
`pum8nh`	Pumadyn 8nh dataset
RAN	Resource Allocationg Network
sig	Sigmoid function
`sin`	Sine dataset
threshold	Threshold value

1. INTRODUCTION

Artificial neural networks [1, 2] provide us a great flexibility in learning different real life problems. Many of these problems can be divided into two topics, *classification* and *regression*. In classification problems, the input is assigned to a single class among a number of classes. In regression problems, there is a continous mapping between the input(s) and the output(s). In either case, the main aim is to find the underlying function of the given data.

Multilayer networks with fixed topology trained using standard *backpropagation* based on *gradient descent* are the most common use of neural network models (The details of ANN and back propagation will be given in Chapter 2). These networks are only useful with the appropriate network architecture. The standard back propagation algorithm finds the network weights using gradient descent procedure but the network architecture is found by trial and error. The optimal architecture is a network large enough to learn the underlying function, and as small as possible to generalize well. A network smaller than the optimal architecture can not learn the problem, but on the other hand a larger network will overlearn the data with a poor generalization performance. The generalization performance of neural networks can be viewed as the *bias/variance dilemma* [3]. The trade of between the bias and the variance is the key factor in the generalization performance of a neural network. A small network will have a high bias and will fail to learn the underlying function generating the data. If we use a large network, then the bias will be close to zero (it will even be zero if there are enough number of units in the network; in that case, the network will interpolate the data) but then a high variance will be introduced. The optimal architecture is the one that balances the bias and the variance so that the network can generalize the data, ignoring the noise.

The problem of finding the optimal architecture and constructing the network can be considered as a state space search [4]. The basic steps of state space search are: The state space, the initial state, the termination of search, and the search strategy.

The *state space* is the different neural network architectures. The architecture of a neural network is defined by:

- The number of hidden layers in the network
- The number of hidden units in each layer
- The connectivity graph specifying how the input, output and hidden units are connected
- Activation functions of hidden units
- Parameters of the whole network

The *initial state* is determined by the nature of the algorithm used. In constructive approach for neural network construction, the algorithm starts with small networks and constructs the network by adding nodes and connections [4, 5]. The simplest network is the network with no hidden units. In problems where prior information exists, the initial state can be different. Constructive approach will be examined in detail in Chapter 3. In contrast to the constructive approach, the algorithm starts with a large network and removes the unnecessary nodes and connections in the destructive approach (also called the pruning approach) [6].

The search must be terminated (reaching to a *goal state*) when the generalization performance of the network begins to decrease. Some algorithms continue until all training examples are correctly classified but these algorithms fail to learn noisy data and generate larger networks.

The search strategy determines how to reach to the goal state starting with the initial state in general. In particular, it determines the next state and how to move to the next state from the current state. In the constructive approach, the network of the next state is always larger than the network of current state. It is the opposite in pruning approach where the network of the next state is always smaller than the network of current state.

In most of the search problems, the *state transition mapping* is determined by the problem, not by the search startegy. But in regression and classification problems, the problem itself has no restrictions, besides the minimization of the error term, on the possible state transitions.

There are two types of state transition mappings: single-valued and multi-valued. In single-valued case, there is only one next state. The disadvantage of this type of algorithms is that, finding a good architecture for any kind of problem can be impossible since there is only one possible next state. In the multi-valued case, there are candidate next states and the algorithm chooses one among the candidates.

Constructive approach is generally preferred to the pruning approach for neural network construction. The advantages of constructive approach over pruning approach are:

- Specifying the initial network is easier.
- Computation time is less since it starts from smaller networks.
- They are more likely to find smaller networks.

1.1. Outline of Thesis

In Chapter 2, the basics of multilayer perceptrons and their training is explained. Detailed information on the constructive approach and a detailed literature survey on constructive algorithms is given in Chapter 3. The proposed methods are explained in Chapter 4. The results of the experiments are given in Chapter 5. The conclusions of the thesis and possible future work are in Chapter 6.

2. MULTILAYER PERCEPTRONS

Our work on artificial neural networks concentrates on multilayer perceptron type neural networks. The perceptron is the basic processing element (Figure 2.1).

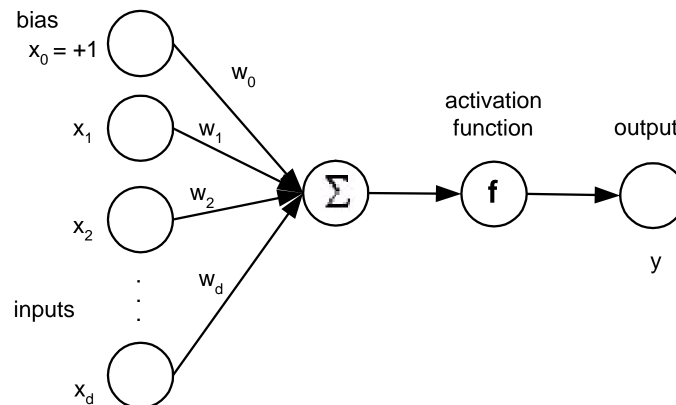


Figure 2.1. A single perceptron

In the simplest case, it is the weighted sum of its inputs.

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \tag{2.1}$$

where y is the output unit, x_j is the jth dimension of the input vector, w_j is the weight associated with the input x_j and w_0 is the weight associated with the *bias* unit, which is always +1. If the weighted sum needs to be activated when it exceeds some threshold value, then an activation function, f, can be used.

$$y = f\left(\sum_{j=1}^{d} w_j x_j + w_0\right) \tag{2.2}$$

The back propagation algorithm based on *gradient descent* is used in learning the weights. *Gradient descent* is applied in order to minimize the error between the network output values and target values for these outputs. The algorithm starts with initial weights, which are randomly assigned in an interval around zero, and updates

the weights based on the derivative of the error function until the network error is acceptable. The update procedure is done by the *delta rule*. The key idea behind the *delta rule* is to use *gradient descent* to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples [2].

2.1. Linear Perceptron

In linear preceptron, the input units, which are fed from environment, are connected directly to output units (Figure 2.2).

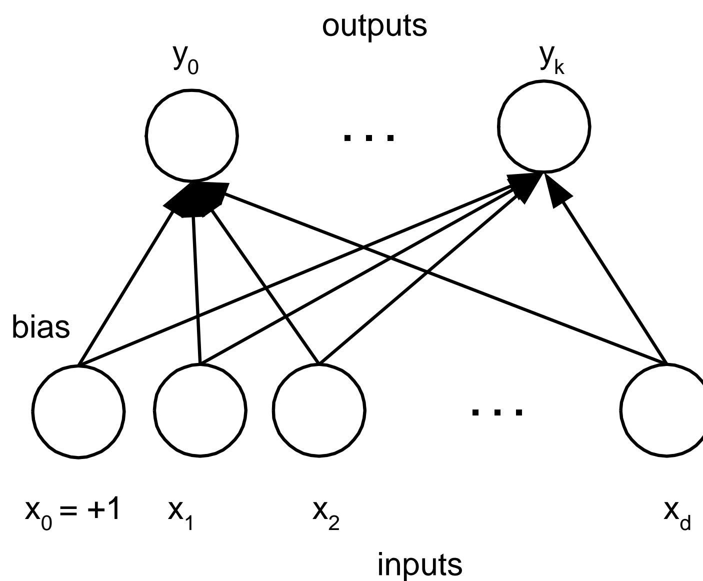


Figure 2.2. A linear perceptron

2.1.1. Regression

In regression, the output is the linear combination of input units (Equation 2.1). In the case where $d = 1$, Equation 2.1 becomes

$$y(x) = wx + w_0 \tag{2.3}$$

which is the equation of a line with w as the slope and w_0 is the intercept. The perceptron with one input and one output is used to implement linear regression. If $d > 1$, the line becomes an hyperplane and multivariate regression can be implemented. So the Equation 2.1 can be written in vector form as:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 = \sum_{j=1}^{d} w_j x_j + w_0 \tag{2.4}$$

where $\boldsymbol{x}$ is the d-dimensional input vector, $\boldsymbol{w}^T$ is the d-dimensional weight vector and w_0 is the weight of the bias.

To minimize the sum of squared error

$$E = \sum_t (r^t - y^t)^2 \tag{2.5}$$

the derivative of the error function is computed

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_j} \tag{2.6}$$

and the update rule becomes

$$\Delta w_j = \eta \sum_t (r^t - y^t) x_j^t \tag{2.7}$$

for $j = 0, \ldots, d$.

2.1.2. Classification

In classification with K classes, there are K perceptrons where $y_i^t = 1$ if $x^t \in C_i$ and $y_i^t = 0$ otherwise. The value of an output unit is the linear combination of the input units, with an activation function applied (Equation 2.2). *Softmax* function is

used as the activation function.

$$o_i = \boldsymbol{w}_i^T \boldsymbol{x} + w_0 = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} \tag{2.8}$$

for $i = 1, \ldots, K$.

$$y_i = \frac{\exp(o_i)}{\sum_{l=1}^{k} \exp(o_l)} \tag{2.9}$$

In classification, the error is the cross-entropy

$$E = -\sum_t \sum_i r_i^t \log y_i^t \tag{2.10}$$

and

$$\Delta w_{ij} = \eta \sum_t (r_i^t - y_i^t) x_j^t \tag{2.11}$$

for $i = 1, \ldots, K$ and $j = 1, \ldots, d$.

2.2. Multilayer Perceptron

In a multilayer perceptron, there are one or more hidden layers between the input and the output layers. Each unit is connected to the units in the preceeding layer (Figure 2.3).

The value of a hidden unit is the linear combination of the input units, with an activation function applied (Equation 2.2). *Sigmoid* function is used as the activation function. The computation of h_k for a two layer network is given in Equation 2.12.

$$h_k = sigmoid(\boldsymbol{w}_k^T \boldsymbol{x} + w_{k0}) = \frac{1}{1 + \exp(-\sum_{j=1}^{d} w_{kj} x_j + w_{k0})} \tag{2.12}$$

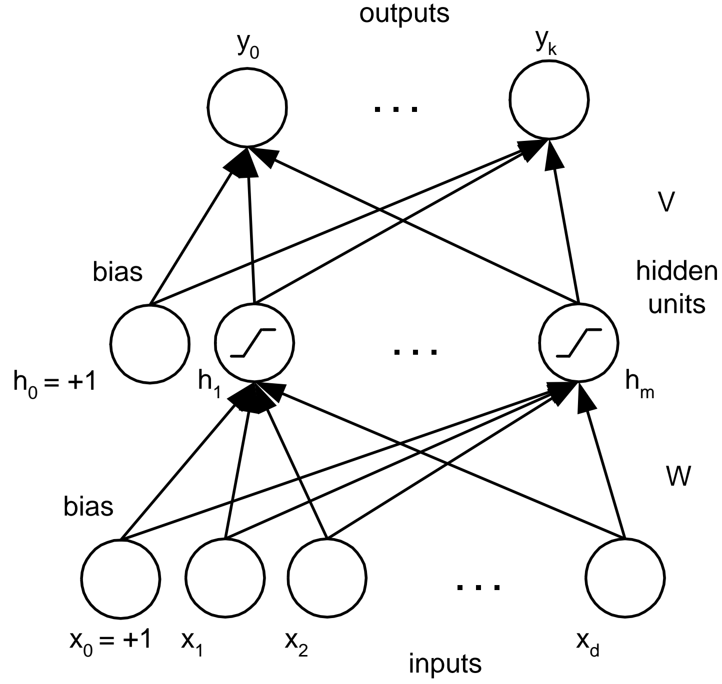


Figure 2.3. A multilayer perceptron

To find the update rule, the partial derivative of the error function is computed. For networks with two layers, the partial derivative is

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial h_k}\frac{\partial h_k}{\partial w_{kj}}. \tag{2.13}$$

2.2.1. Regression

The value of output units for nonlinear regression with K outputs is calculated as

$$y_i^t = \sum_{k=1}^{P} v_{ik} h_k^t + v_{i0} \tag{2.14}$$

for $i = 1, \ldots, K$ where P is the number of hidden units.

When we calculate the partial derivatives of the error function

$$E = \sum_t \sum_i (r_i^t - y_i^t)^2 \tag{2.15}$$

the update rules become

$$\Delta v_{ik} = \eta \sum_t (r_i^t - y_i^t)^2 h_k^t \tag{2.16}$$

$$\Delta w_{kj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ik} \right] h_k^t (1 - h_k^t) x_j^t. \tag{2.17}$$

2.2.2. Classification

The value of output units for classification with K outputs is calculated as

$$o_i^t = \sum_{k=1}^{P} v_{ik} h_k^t + v_{i0} \tag{2.18}$$

for $i = 1, \ldots, K$.

$$y_i^t = \frac{\exp(o_i^t)}{\sum_l \exp(o_l^t)} \tag{2.19}$$

When we calculate the partial derivatives of the error function,

$$E = -\sum_t \sum_i r_i^t log(y_i^t) \tag{2.20}$$

the update rules become

$$\Delta v_{ik}^t = \eta \sum_t (r_i^t - y_i^t)^2 h_k^t \tag{2.21}$$

$$\Delta w_{kj}^t = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ik} \right] h_k^t (1 - h_k^t) x_j^t \tag{2.22}$$

The equations for networks with more than two layers can be derived similarly.

3. CONSTRUCTIVE ALGORITHMS FOR FINDING THE NEURAL NETWORK ARCHITECTURE

The problem of determining the architecture of a neural network together with the disadvantages of the backpropagation algorithm, directed the research towards algorithms that determine not the weights and parameters but also the structure of the network necessary for learning the data. The problems in back propagation are as follows [4, 7]:

- Back propagation algorithm is computationally expensive. So, choosing the architecture by trial and error results in expensive training.
- *Catastrophic interference* is a problem when learning new data. The storage of new information causes the previous information to be lost.
- *Moving target problem* causes an inefficiency in learning. Each unit in the network tries to adjust its weights so that it can have an important and useful role in the overall network. But the lack of communication between hidden units causes slow training.

Constructive algorithms can solve some of the problems above but also introduce some other problems. The main disadvantage of the constructive algorithms is their weakness on noisy data. There are some problems that the standard backpropagation algorithm fails to learn but constructive algorithms are successful. But when there is high noise in the data, the constructive algorithms are generally not successful. The different training techniques used in constructive algorithms try to overcome these problems.

3.1. Training Techniques

Neural networks with fixed architecture are only trained once. However, in constructive methods, every time the architecture is changed, the training must be re-

peated. Since, the computational cost of these repeated trainings are high, some techniques are applied to reduce both time and space complexity.

3.1.1. Weight Freezing

The simple way of training the new network is to train only the newly added unit. The assumption in this approach is that, the existing units in the network are already trained and are useful in obtaining the target function. Each new hidden unit is trained until its contribution to network error is minimized and then is added to the network. Cascade-correlation algorithm (Section 3.2.5) uses weight freezing in training the network [7].

Although this kind of training reduces the time and space complexity of the whole process, research on weight freezing [8, 9] show that, in general, it fails to find the desired solution. When an extra degree of freedom is introduced by adding a new unit in the network, freezing the existing weights only allows finding the solution in an affine subset of the weight space [8].

On the other hand, the algorithms using weight freezing can sometimes find good solutions with a high generalization power, also with a huge decrease in the learning time.

3.1.2. Re–training the Whole Network Continuing from Previous Weights

When a new unit is added to the network, its appropriate weights are initialized to random values and the training of the whole network continues with the old weights. The idea here is that, the information learned so far will be useful in reaching the final network. Dynamic node creation algorithm (Section 3.2.6) uses this kind of training [8]. This kind of training decreases the training time but it can sometimes cause the algorithm to get stuck in local minima.

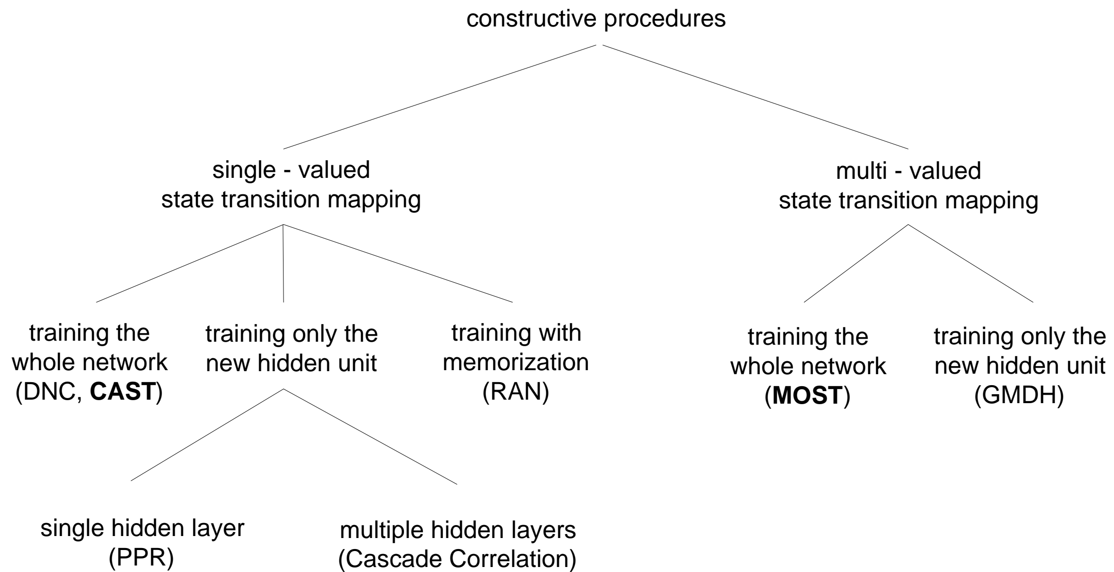


Figure 3.1. Types of constructive algorithms

3.1.3. Re–training the Whole Network from Scratch

When a new architecture is selected, the training of the new network is done with all the weights initialized to random values. This kind of training is the same with the standard backpropagation training. These algorithms have the advantages and disadvantages of backpropagation.

3.2. Constructive Algorithms

A taxonomy of constructive algorithms [4] is given in Figure 3.1

3.2.1. Projection Pursuit Regression

Projection Pursuit Regression (PPR), proposed in [10], is a statistical technique for multivariate data analysis using a two layer feedforward network with linear output units. A single-valued state transition mapping is used. The hidden units are increased one by one and added to the same hidden layer. The weights of the existing nodes are

frozen and only the weights of new unit are trained. The output equations are:

$$y_n(x) = \sum_{j=1}^{n} g_j(\boldsymbol{a}_j^T \boldsymbol{x}) \tag{3.1}$$

or in other variants of PPR

$$y_n(x) = \sum_{j=1}^{n} w_j g_j(\boldsymbol{a}_j^T \boldsymbol{x}) \tag{3.2}$$

where $\boldsymbol{a}_j^T$ is the projection vector, $\boldsymbol{x}$ is the input vector, and g_i are called *smoothers* in the statistics literature. The similarity between single hidden layer networks and PPR can easily be seen.

3.2.2. Resource Allocating Network

Besides the training techniques discussed in Section 3.1, to reduce the computation time, *memorization* can be used during training. The training patterns are memorized and the algorithm decides for the next state based on this memorization. In Resource Allocating Network proposed in [11], the allocation of a new unit is done when an unusual pattern is presented to the network. If the network performs well on the presented pattern, the parameters of the network are adjusted using standard gradient descent. If the network performs poorly on the presented pattern, then a new unit is allocated to correct the response of the presented pattern. In RAN, radial basis functions are used in the hidden units.

3.2.3. Group Method of Data Handling

Group Method of Data Handling (GMDH) type of algorithms [12] use a multi-valued state transition mapping. The number of incoming connections to a hidden unit is fixed but the sources of these incoming connections can change. It can be any combination of input units and other hidden units. The algorithm selects the next architecture among these different combinations.

3.2.4. Upstart Algorithm

The Upstart algorithm, proposed in [13], is a constructive algorithm for binary classification problems. The algorithm starts without hidden nodes and tries to separate the two classes of data. If separation is not possible, then corrector nodes are added. The generated network is very much similar to a network with one hidden layer by defining all the corrector nodes as hidden units.

3.2.5. Cascade Correlation

Cascade-correlation method, proposed in [7], constructs a network with multiple hidden layers. This method uses the constructive approach and starts with an initial network and incrementally adds hidden nodes to the network until a satisfying solution is found. In cascade-correlation, the new nodes are added as a one-unit hidden layer. Figure 3.2 shows the architecture of the cascade-correlation method.

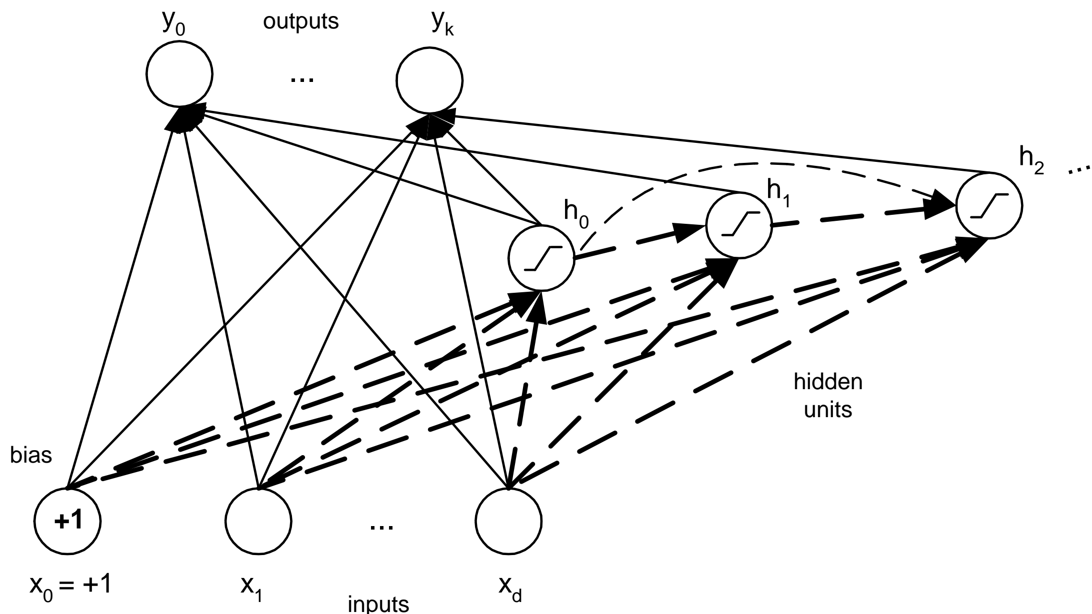


Figure 3.2. Cascade-correlation architecture

The inputs are directly connected to the outputs and to the hidden units. A hidden unit is connected to the inputs, to the outputs and to all the preceeding hidden

units. When a hidden unit is to be added to the network, first its input weights are calculated until its contribution to the network error is minimum and then is added to the network. The calculated input side weights are also frozen. The dashed connections in Figure 3.2 show the frozen weights whereas the straight connections are subject to change during training.

Freezing the input side weights of the hidden units results in training a single layer network each time and this reduces the training time of the algorithm. The algorithm is given in Figure 3.3.

1. Start with an initial network
   - LP
2. Stop training if
   - $E_t \leq C_e$ or $t \geq MAXEPOCHS$
3. Create a hidden node if
   - $E_t > C_e$ and $t \geq patience$
4. To create a hidden node, begin with a candidate unit(s)
   - Connect the candidate unit to all external inputs, pre-existing hidden units and outputs
   - Adjust the candidate unit's input weights by maximizing S:
     - $S = \sum_o | \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}_o)|$
   - Continue until S stops improving
5. Install the candidate unit as a unit in the active network
6. Go to step 2 and continue training with the new network

Figure 3.3. The Cascade-correlation algorithm

3.2.6. Dynamic Node Creation

Dynamic Node Creation (DNC) is a method introduced in [8]. It starts with an initial network and incrementally adds hidden nodes to the network until a satisfactory solution is found. Hidden nodes are added one at a time and to the same hidden layer. The weights of the newly added hidden node are initialized randomly to a small number.

The whole network (both the old hidden nodes and the new one) is re-trained after each hidden node addition.

In DNC, a new hidden node is added to the network when the average error curve begins to flatten out too quickly (Figure 3.4).

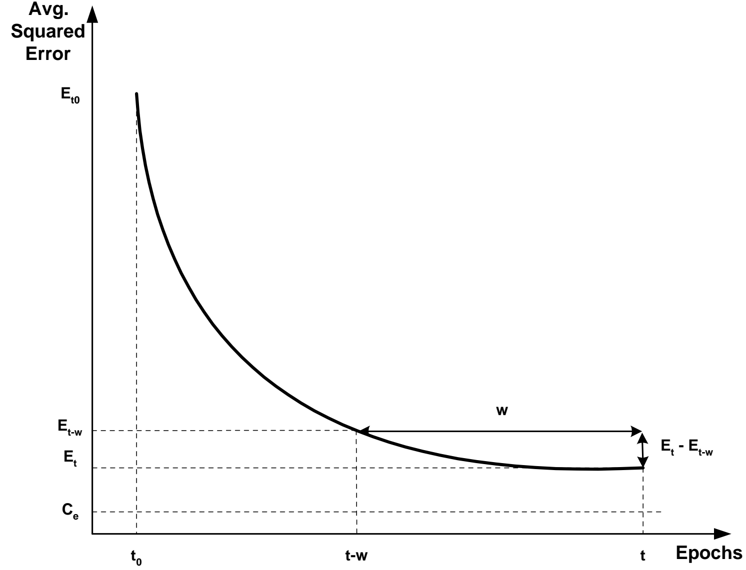


Figure 3.4. Detecting the flattening of the error curve in DNC

A new hidden node is added to the network if the following conditions both hold:

$$t - w \geq t_0 \tag{3.3}$$

$$\frac{E_t - E_{t-w}}{E_{t_0}} < \Delta_r \tag{3.4}$$

Equation 3.3 guarantees that the network is trained at least w epochs after the last hidden node addition. Equation 3.4 checks the flattening of the error curve. If it is flattened enough, then a hidden node is added to the network; if not, then the training continues with the current network and checks for adding a new node in the next epoch. The algorithm is given in Figure 3.5.

1. Start with an initial network
   - MLP with h hidden nodes
2. Stop training if
   - $E_t \leq C_e$ and $m_t \leq C_m$
3. Add a hidden node if
   - $\frac{E_t - E_{t-w}}{E_{t_0}} < \Delta_r$ and $t - w \geq t_0$
4. Initialize the weights of newly created nodes
5. Go to step 2 and continue training with the new network

Figure 3.5. The Dynamic Node Creation algorithm

3.2.7. Constructive Algorithm for Real Valued Examples

Constructive Algorithm for Real Valued Examples (CARVE), proposed in [14], uses convex hull methods for the determination of network weights. The algorithm starts with an empty hidden layer into which thresholds units are added one at a time until the layer is complete. A threshold unit implements a hyperplane in the input domain. A hyperplane that separates a set of points of one class from the rest of the training samples is found and these points are removed from the training set. The next unit added will use the new training set and tries to seperate another set of points. The algorithm continues until all the examples in the training set are correctly classified.

3.2.8. Feedforward Neural Network Construction Using Cross Validation

This algorithm, proposed in [15], uses cross validation for adding units to a single hidden layered network. The network with more hidden units is only accepted if the total accuracy on training and cross validation samples is higher than that of the previous network. The algorithm starts with an initial network and re-trains the whole network when a new unit is added.

4. PROPOSED METHODS

4.1. Modified Dynamic Node Creation

We take DNC algorithm as a base and make some modifications. We start with an initial network, which is a single hidden layer network with one hidden unit in our work, and start to train the network. The network is trained at least w epochs after the last hidden node addition. After w epochs, we add a hidden node if the ratio of the average squared error in last w trials to the average squared error in last hidden node addition (which is again the average of last w trials before the addition) is less than Δ_r. After the hidden node addition, all weights (old and the new ones) in the network are initialized to small numbers. This significantly increased the performance of the algorithm especially in classification problems. This is the main difference with the DNC algorithm. In DNC, the network is trained after each hidden node addition over the existing weights but as can be seen from the results the performance of the algorithm is poor.

A new hidden node is added to the network if the following conditions hold:

$$t - w \geq t_0 \tag{4.1}$$

$$\frac{(\sum_{k=t-w}^{t} E_k)/(w+1)}{(\sum_{k=t_0-w}^{t_0} E_k)/(w+1)} < \Delta_r \tag{4.2}$$

Defining,

$$\bar{E}_t = \frac{\sum_{k=t-w}^{t} E_k}{w+1} \tag{4.3}$$

and

$$\bar{E}_{t_0} = \frac{\sum_{k=t_0-w}^{t_0} E_k}{w+1} \tag{4.4}$$

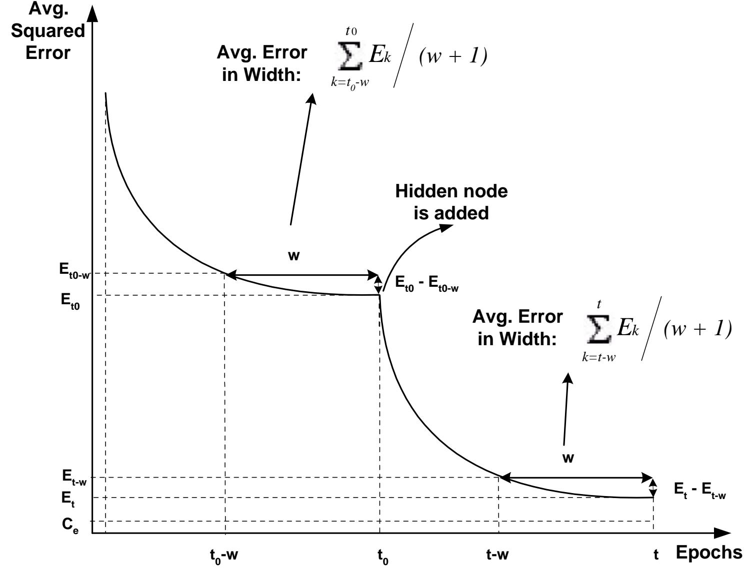


Figure 4.1. Detecting the flattening of the error curve in modified DNC

we can write Equation 4.2 as:

$$\frac{\bar{E}_t}{\bar{E}_{t_0}} < \Delta_r \tag{4.5}$$

Equation 4.1 guarantees that the network is trained at least w epochs after the last hidden node addition. Equation 4.5 checks the flattening of the error curve. If it is flattened enough, then a hidden node is added to the network; if not, the training continues with the current network and checks for adding a new node in the next epoch. Equation 4.5 checks the flattening of the error curve as in Equation 3.4 in DNC. In Equation 3.4 only the difference of the errors is considered but in Equation 4.5 the average error in width is considered. Averaging overcomes the problem of sudden changes in the error curve.

The algorithm stops adding nodes when network error is less than a user-defined value, C_e:

$$E_t \leq C_e \tag{4.6}$$

The algorithm also stops adding nodes when the current error is greater than the old error (the network error when the last node is added to the network). When a new node is added to the network, the network is trained at least w epochs. This parameter, w, must be selected properly so that the network starts to converge within w epochs. w epochs after the last hidden node addition, we start to check the effect of the addition of the last hidden node to the network. Since, an improvement in the network is expected (decrease in the network error), if there is no improvement, the algorithm stops and accepts the previous network (the network before the last hidden node is added) as the final network. Deciding if there is an improvement in the network or not is another issue. There are two cases that we consider as "no improvement" in the network (Equations 4.7 and 4.8) :

- Addition of the last hidden node causes an increase in the network error instead of a decrease (current error is greater than the old error). We define a *threshold* to decide that this increase is significant or not. We restore the old network if the following condition holds.

$$\frac{\bar{E}_{t_0} - \bar{E}_t}{\bar{E}_{t_0}} < threshold \tag{4.7}$$

- Addition of the last hidden node causes a decrease in the network error (current error is less than the old error). We define a *threshold* to decide that this decrease is significant or not. We restore the old network if the following condition holds.

$$\frac{\bar{E}_t - \bar{E}_{t_0}}{\bar{E}_{t_0}} > threshold \tag{4.8}$$

Equation 4.7 prevents over-fitting, and Equation 4.8 prevents adding too many hidden nodes that do not affect the performance.

4.2. Incremental Neural Network Construction Using 5×2 cv F Test

For comparing two different architectures in the state space, using only accuracies can lead to wrong results. Statistical tests can give more reliable answers for the

1. Start with an initial network
   - MLP with h hidden nodes
2. Stop training if
   - $E_t \leq C_e$ or $t \geq MAXEPOCHS$
3. Add a new hidden node if
   - $\frac{E_{t_0}-E_t}{E_{t_0}} < threshold$ and $t - w \geq t_0$
4. Initialize the weights of all nodes
5. Go to step 2 and continue training with the new network

Figure 4.2. The Modified DNC algorithm

goodness of classifiers. Also, it is hard to define parameters such as *threshold* (Section 4.1) or Δ_r (Section 3.2.6) since those parameters are highly problem specific.

The idea in using 5×2 cv F Test, proposed in [16], is that it is a good way of comparing two classifiers, and it needs only one additional parameter, *confidence*, which can easily be set.

We propose two methods that uses 5×2 cv F Test in constructing neural networks. The first method, Constructive Algorithm using Statistical Tests (CAST), considers only neural networks with one hidden layer and decides on the number of hidden units. The second method is an enlarged version. There is no assumption on the number of layers or hidden units. The algorithm starts from a neural network with no hidden layers and constructs the neural network. The resulting network can have more than one hidden layer.

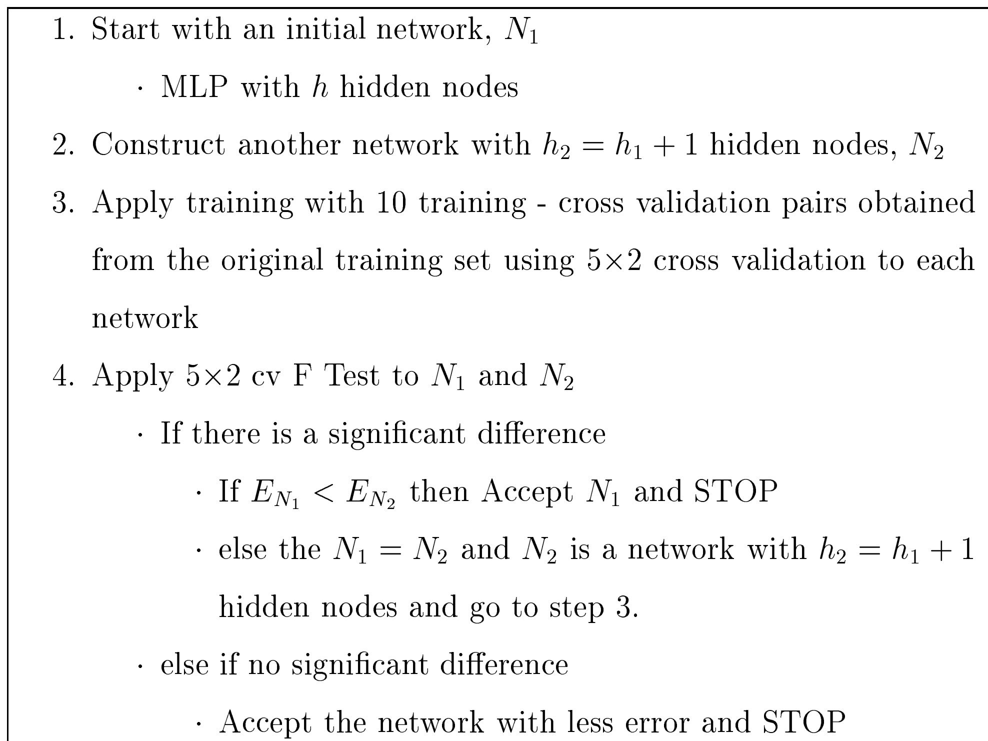
4.2.1. Constructive Algorithm Using Statistical Tests

Constructive Algorithm using Statistical Tests (CAST) is similar to modified DNC except the method in the decision of adding a node. Modified DNC compares the decrease or increase in the error of the current network with the candidate network but this can lead to wrong results. 5×2 cv F Test is used instead, for comparing the two networks.

The algorithm is as follows:

We start with an initial network, call it N_1, with H hidden units , and initialize its weights randomly. Then we form another network, call it N_2, with $H+1$ hidden units, and initialize its weights randomly. Then we apply 5×2 cv F Test to these networks:

- If 5×2 cv F Test rejects the hypothesis that these two classifiers are the same, We calculate the average of errors in ten runs for N_1 and N_2 and if the error of N_1 is less than the error of N_2 then we accept N_1 as the final network. If the error of N_2 is less than the error of N_1, we accept N_2
- If 5×2 cv F Test accepts the hypothesis that these two classifiers are the same, We calculate the average of errors in ten runs for N_1 and N_2, and we accept the one with less error as the final network.



1. Start with an initial network, N_1
   - MLP with h hidden nodes
2. Construct another network with $h_2 = h_1 + 1$ hidden nodes, N_2
3. Apply training with 10 training - cross validation pairs obtained from the original training set using 5×2 cross validation to each network
4. Apply 5×2 cv F Test to N_1 and N_2
   - If there is a significant difference
     - If $E_{N_1} < E_{N_2}$ then Accept N_1 and STOP
     - else the $N_1 = N_2$ and N_2 is a network with $h_2 = h_1 + 1$ hidden nodes and go to step 3.
   - else if no significant difference
     - Accept the network with less error and STOP

Figure 4.3. The CAST algorithm

4.2.2. Applying Multiple Operators Using Statistical Tests

The disadvantage of the CAST algorithm is that it increments the number of hidden nodes in the network one by one and for some problems, one hidden node difference can be statistically insignificant. Another disadvantage is that it assumes that the architecture has a single hidden layer. Architectures with no hidden layers or more than one hidden layer are discarded.

Constructive algorithm with Multiple Operators using Statistical Tests (MOST) makes no assumptions on the number of layers of the network and overcomes the problem of one hidden node addition by applying multiple operators. One hidden node additions or removals are used for finetuning the network.

MOST starts with an initial network, which is a network with no hidden layers. Then it tries to apply the next applicable operator (Figure 4.5). The application of operators, in the order of precedence, is follows:

- Remove a percentage of hidden units from a layer.
- Remove a hidden unit from a layer.
- Add a hidden unit to a layer.
- Add a percentage of hidden units to a layer.
- Add a new layer between the output and the layer below the output. When we add a new layer, the number of hidden units in all the layers are redetermined.

An initial value for the minimum number of hidden units, *MINHIDDEN*, in a layer is specified since the performance of a linear perceptron is better in most of the problems than a multilayer perceptron with small number of hidden units. This initial value can be changed in the algorithm as a result of statistical comparisons. When we apply an operator, if the calculated value for the number of hidden units in a layer is smaller than *MINHIDDEN*, then *MINHIDDEN* is used.

When a new layer is to be added to the network, the hard point is to determine

the number of hidden units in each layer. A popular heuristic is using the average of the number of hidden nodes in the upper and the lower layers. But this can lead to wrong results if the input dimension is very large or small. We applied four heuristics in order to be able to cover different architectures in the search space.

- Number of nodes in the upper layer
- (Number of nodes in the upper layer) $\times$ 2
- Average of the nodes in the upper and lower layers
- (Average of the nodes in the upper and lower layers) / 2

Heuristics one and three are the base heuristics. Heuristics two and four are useful when heuristics one and three are either too small or too large.

When adding hidden layers, we first determine the number of nodes in the newly added layer, which is the layer before the outputs. The simplest of the four heuristics is chosen and the number of hidden nodes in the preceding layer is then calculated. Consider a one hidden layer network with 10 outputs, 18 hidden nodes in the hidden layer and 64 inputs. If we want to add a new layer to this network, the results of above heuristics are 10, 20, 14 and seven respectively. We first select the first minimum, seven as the number of hidden units in that layer and then calculate the heuristics for the next layer. The results are 10, 20, 37, 18. Again the first minimum is chosen and the first candidate network has 10 output, seven hidden nodes in the second layer, seven hidden nodes in the first layer and 64 input. The algorithm continues with the second choice. Table 4.1 shows an example for the determination of candidate network architectures for a current architecture.

The algorithm starts from the first operator and applies that operator if it is applicable. If there is no hidden layer in the current network, then first four operators are not applicable. The algorithm adds a new layer and tries the above mentioned heuristics starting from the simple one to the complex one.

In order to prevent the algorithm to go in a loop, if a simple architecture is selected

Table 4.1. An example for determining the number of hidden units

Network	No. of inputs	No. of nodes in 1st layer	No. of nodes in 2nd layer	No. of outputs	Heuristics calculated	Selection
current	64	18	0	10	*10, 20, 14, 7*	min to max
candidate 1	64	**7**	*7*	10	**7**, 14, 35, 17	1st min
candidate 2	64	**18**	*10*	10	10, 20, 37, **18**	2nd min
candidate 3	64	**28**	*14*	10	14, **28**, 39, 17	3rd min
candidate 4	64	**42**	*20*	10	20, 40, **42**, 21	4th min

and after then a complex one is selected due to significancy, then the algorithm never selects a simpler architecture again.

If the two networks are not significantly different, then selection is done by comparing the complexities. The complexity measure is defined as the number of connections in the network (Figure 4.6).

The pseudocode of the MOST algorithm is given in Figure 4.4.

1. Start with an initial network, N_1
    - $N_1 = LP$
2. For all applicable operators i
    - $N_2 = Operator_i(N_1)$
    - if preferable(N_2, N_1) then $N_1 = N_2$ and start new loop at step 2

Figure 4.4. The MOST algorithm

- Operators (in the order of preference)
  - Remove a hidden unit (or units) from a layer
  - Add a hidden unit (or units) from a layer
  - Add a new hidden layer

Figure 4.5. Operator set used in the algorithm

- Function preferable(new, current)
  - C_t = candidate network
  - C_{t-1} = current network
  - C_{t-2} = the last network before the hidden layer addition
  - If $E(C_t) < E(C_{t-1})$ then

    preferable = TRUE
  - Else if $E(C_t) = E(C_{t-1})$

    and $E(C_t) < E(C_{t-2})$

    and $comp(C_t) < comp(C_{t-1})$ then

    preferable = TRUE
  - Else if $E(C_t) = E(C_{t-1})$

    and $E(C_t) = E(C_{t-2})$

    and $comp(C_t) < comp(C_{t-2})$ then

    preferable = TRUE
  - Else preferable = FALSE

Figure 4.6. Deciding between two architectures

5. EXPERIMENTS

5.1. Datasets

A summary of datasets used in the simulations are given in Table 5.1.

5.1.1. Regression Datasets

Sin dataset (`sin`) is an artificial regression dataset produced from sin function with noise added.

$$y = sin(x) + \epsilon \tag{5.1}$$

The input and the output are one dimensional. There are 500 examples in the training set and 500 examples in the test set.

California housing dataset (`california`) is a regression dataset obtained from the StatLib repository [17]. It contains the housing information in California. The input is eight - dimensional and the output is one - dimensional. The data consists of 20,640 examples. In our work, we divide the data into two equal sets. There are 10,320 examples in the training set and 10,320 examples in the test set.

Boston housing dataset (`boston`) is a regression dataset obtained from the StatLib repository [17]. It contains the house-price data in Boston. The input is 13 - dimensional and the output is one - dimensional. The data consists of 506 examples. In our work, we divided the data into two sets. There are 400 examples in the training set and 106 examples in the test set.

Puma Dynamics dataset is a family of datasets synthetically generated from a realistic simulation of the dynamics of a Unimation Puma 560 robot arm. There are eight variations on the same model. We have used the two of them. `pum8fh` is a fairly

linear dataset with high noise with eight input dimensions. `pum8nh` is a non linear dataset with high noise with eight input dimensions. The dataset can be obtained from Delve Datasets [18].

5.1.2. Classification Datasets

OCR dataset (`ocr`) is a classification dataset obtained from Isabelle Guyon, AT&T Bell Labs. The input is 16×16 bitmap digits, consist of zeros and ones. So, the input is 256 dimensional. There are 10 classes (digits zero...nine). There are 600 examples in the training set and 600 examples in the test set.

Optical digits dataset (`optDigits`) is a classification data set collected in [19] and can be obtained from the UCI Repository [20]. The input is collected as 32×32 bitmap digits, consist of zeros and ones. Each bitmap is divided into 4×4 blocks and the number of ones is calculated for that block. There are 64 4×4 blocks in a 32×32 bitmap, so the input has 64 dimensions, range from zero to 16. There are 3,823 examples in the training set and 1,797 examples in the test set.

Pen digits dataset (`penDigits`) is a classification data set collected in [21] and can be obtained from the UCI Repository [20]. The input has 16 dimensions, which are a sequence of eight equidistant coordinates. There are 7,494 examples from in the training set and 3,498 examples in the test set.

Table 5.1 shows the properties of all datasets.

5.2. Results

MLP, cascade correlation, DNC, modified DNC, CAST, MOST algorithms are applied to all datasets. Parameters used for each dataset are listed in Appendix A. The results for only two datasets (`sin` and `ocr`) are given in the following sections. The figures and tables for other datasets are given in Appendices B- F

Table 5.1. Datasets used in experiments

Dataset	regression / classification	Training Set	Test Set	Input Dimension	Number of outputs
sin	reg	500	500	1	1
california	reg	10320	10320	8	1
boston	reg	400	106	13	1
pum8fh	reg	4096	4096	8	1
pum8nh	reg	4096	4096	8	1
ocr	cls	600	600	256	10
optDigits	cls	3823	1797	64	10
penDigits	cls	7494	3498	16	10

5.2.1. MLP Results

A MLP with one hidden layer and with different number of hidden units is run on all datasets and the results are given in figures. The continous lines show the error on training set and the dashed lines show the error on test set. The errors are averaged over ten runs.

In `sin` dataset, a MLP with three hidden units can actually find successful results but in some runs it can get stuck in local minima. This is the reason of high average errors with three hidden unit architecture (Figure 5.1).

When we look at the error curve of `sin` dataset (Figure 5.2), we see that the decrease in the error is very small after 100 epochs.

As can be seen from Figure 5.3, in `ocr` dataset, MLP with 13 hidden units gives the best result. A training with 50 epochs gives successful results (Figure 5.4).

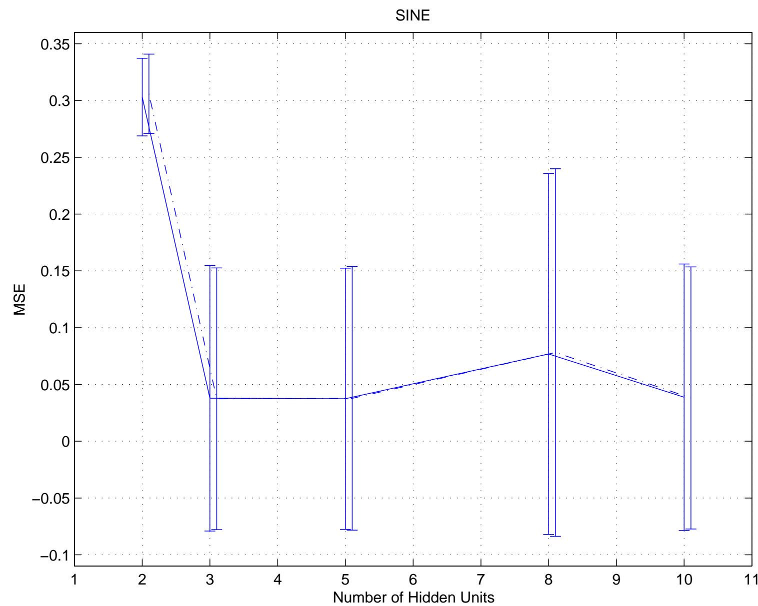


Figure 5.1. The effect of the number of hidden units for `sin` dataset (dashed: test error, continous: training error)

5.2.2. Modified DNC Results

The error graphs of each dataset for DNC and modified DNC algorithms are given in this section. The main difference between the error graphs of DNC and modified DNC is the shifts in error when a new hidden node is introduced to the network. In DNC, the existing weights are used as the initial values when training the new network and only the new weights are randomly determined whereas in modified DNC all weights are initialized to random values close to zero. This difference results in the high shifts in the error curve when adding a new node in modified DNC.

Figure 5.5 and Figure 5.6 are the error graphs of one of the training-validation set pairs of `sin` dataset for DNC and modified DNC respectively.

Sine problem is a relatively simple problem and all algorithms generally perform well with minor differences (Figure 5.7). In our experiments, a MLP with three, four or five hidden units can get stuck in local minima in one of ten runs whereas DNC finds a network with five hidden nodes and that network finds the result successfuly.

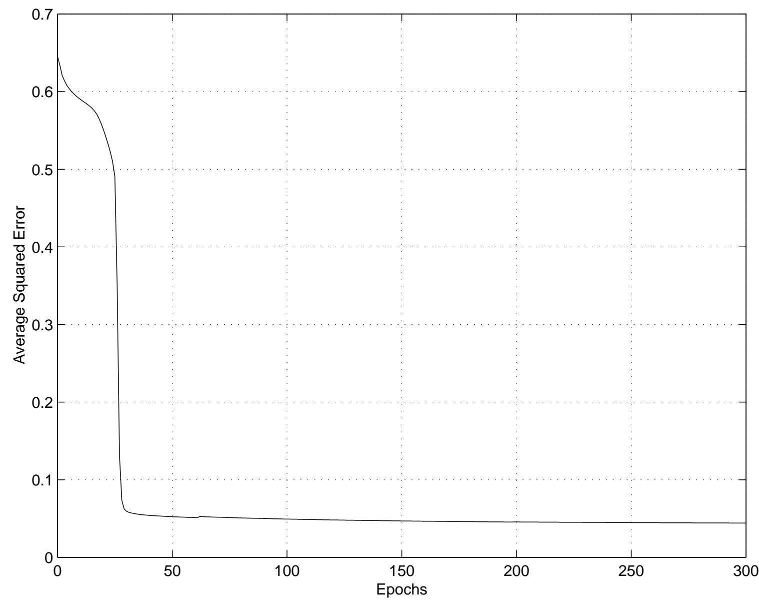


Figure 5.2. The error graph during training of `sin` dataset in MLP

This difference is a result of the training technique of DNC. When a new hidden node is added to the network, the existing weights are used as the initial weights of the new network. This technique causes the resulting network to get stuck in local minima in some problems but it can also cause the network to learn successfuly in some problems like sine problem. Figure 5.8 shows an example plot of a network that stucks in local minima.

In classification datasets that we have used, DNC performed poorly on the test set. Figure 5.9 and Figure 5.10 are the error graphs of one of the training-validation set pairs of `ocr` dataset for DNC and modified DNC, respectively. It can be seen that the performances of DNC and modified DNC in training set are close when the network converges. But the performance of modified DNC on the test set (Section 5.2.5) is better than DNC. The re-training of the whole network continuing from the previous weights in DNC algorithm has a negative effect in the performance of the network in some problems like `ocr`.

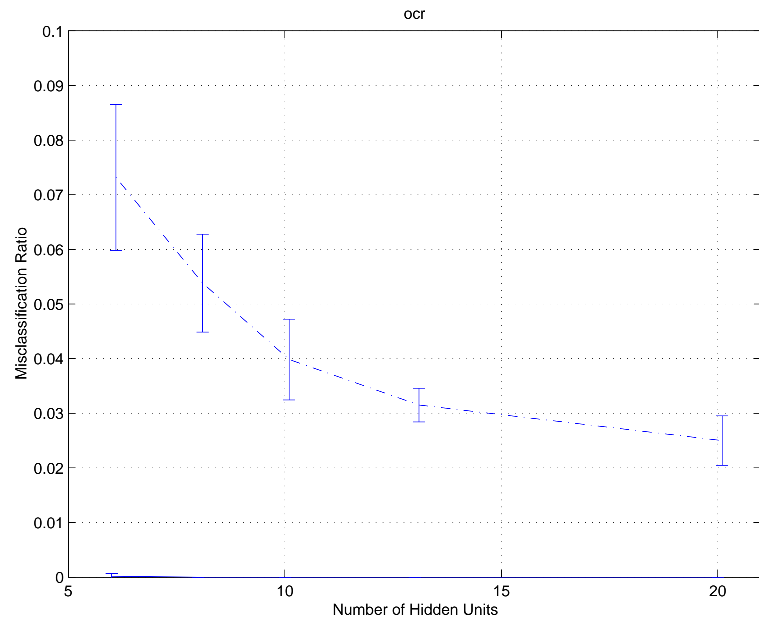


Figure 5.3. The effect of the number of hidden units for `ocr` dataset (dashed: test error, continous: training error)

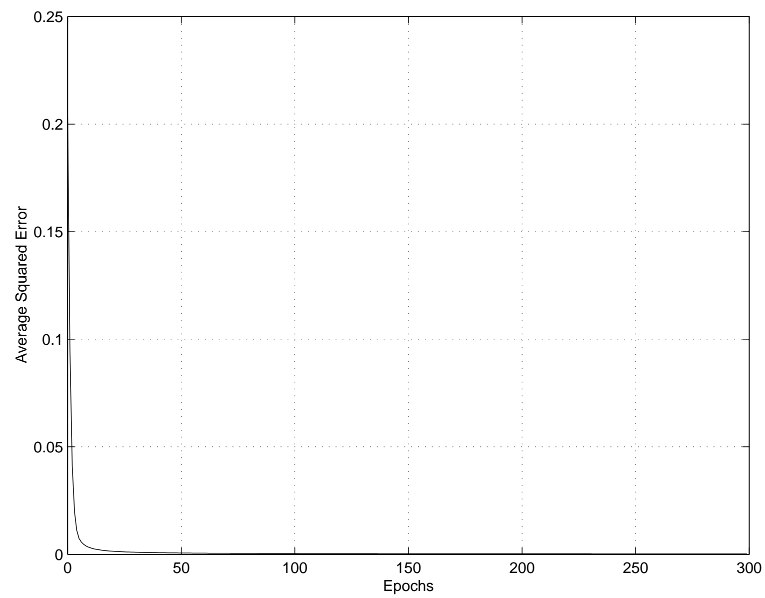


Figure 5.4. The error graph during training of `ocr` dataset in MLP

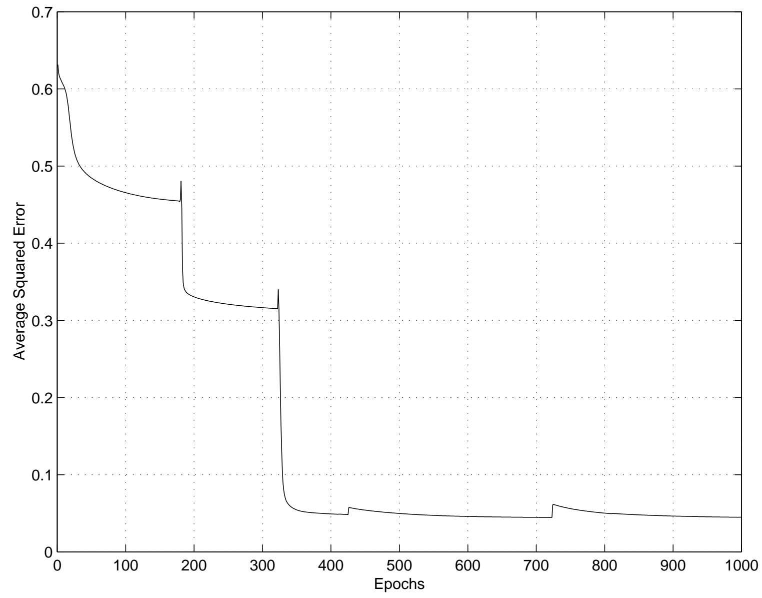


Figure 5.5. The error graph during training of `sin` dataset with the DNC algorithm

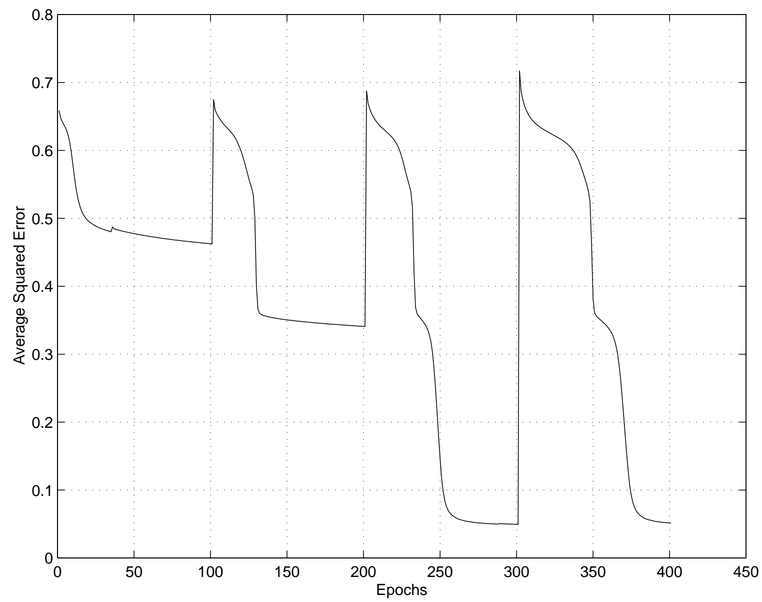


Figure 5.6. The error graph during training of `sin` dataset with the modified DNC algorithm

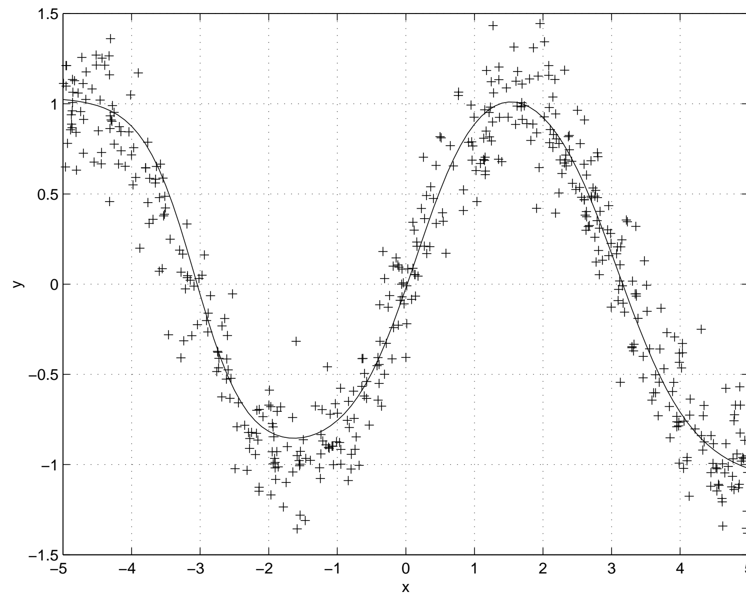


Figure 5.7. A successful learning of `sin` dataset

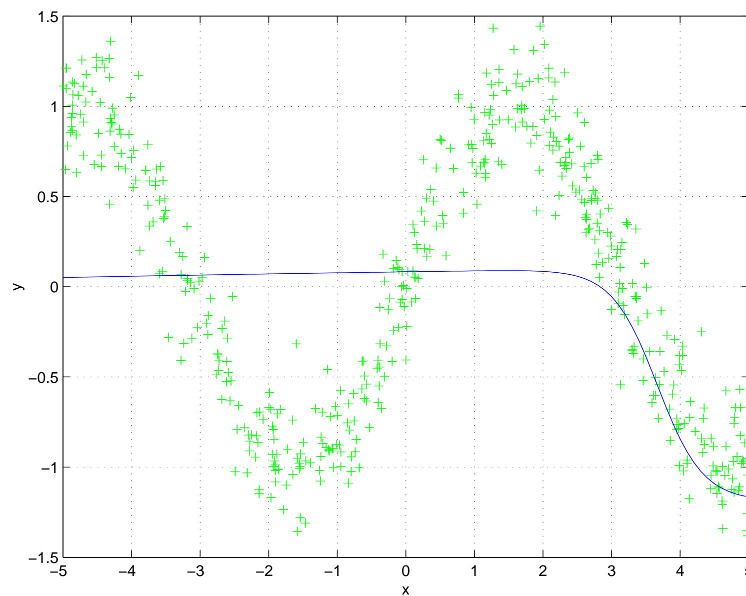


Figure 5.8. A plot of a network that stucks in local minima of `sin` dataset

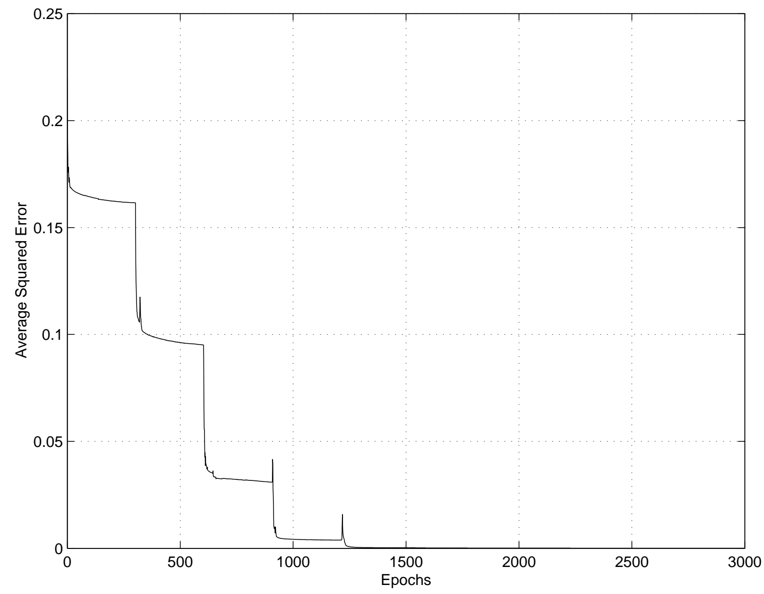


Figure 5.9. The error graph during training of `ocr` dataset with the DNC algorithm

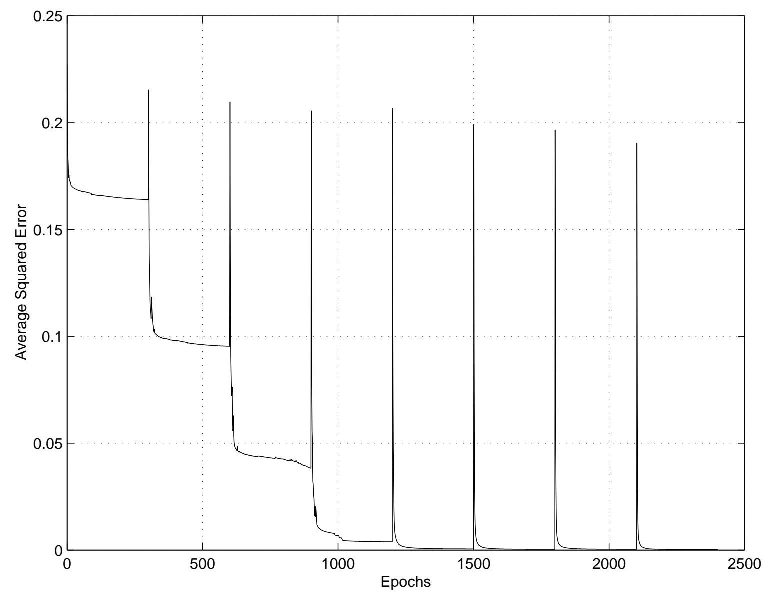


Figure 5.10. The error graph during training of `ocr` dataset with the modified DNC algorithm

5.2.3. CAST Results

The CAST algorithm is applied with five different confidence levels, 0.90, 0.95, 0.99, 0.995, 0.999 and the mean and the range of the errors in five different confidence levels are shown in the figures. The number of hidden units found in each confidence level is also shown. 5×2 cv is applied and the continous line shows the error on validation set whereas the dotted, thin line shows the error on test set.

For `sin` dataset, Figure 5.11 shows the mean and the range of the errors in five different confidence levels. It can be seen from the figure that in each confidence level except 0.999, the CAST algorithm finds a three or four hidden node architecture as the solution. The variance of three hidden node architecture is higher than the four hidden node architecture but these two architectures are statistically insignificant.

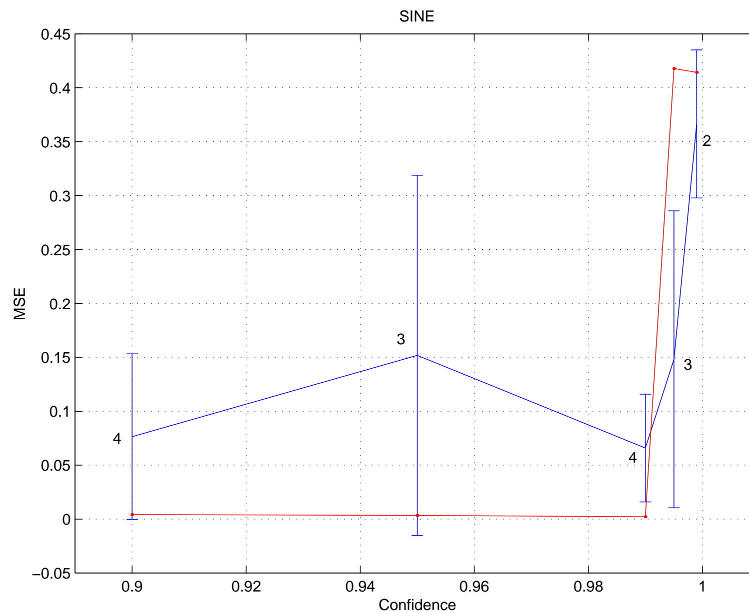


Figure 5.11. The error graph and the number of hidden nodes found for different confidence levels in `sin` dataset with CAST algorithm

Figure 5.12 shows the mean and the range of the errors in `ocr` in five different confidence levels. The CAST algorithm finds a 10, 13, eight, seven and six hidden node architecture as the solution in each confidence level, respectively. The variance and the

average error gets higher as the number of hidden nodes decrease.

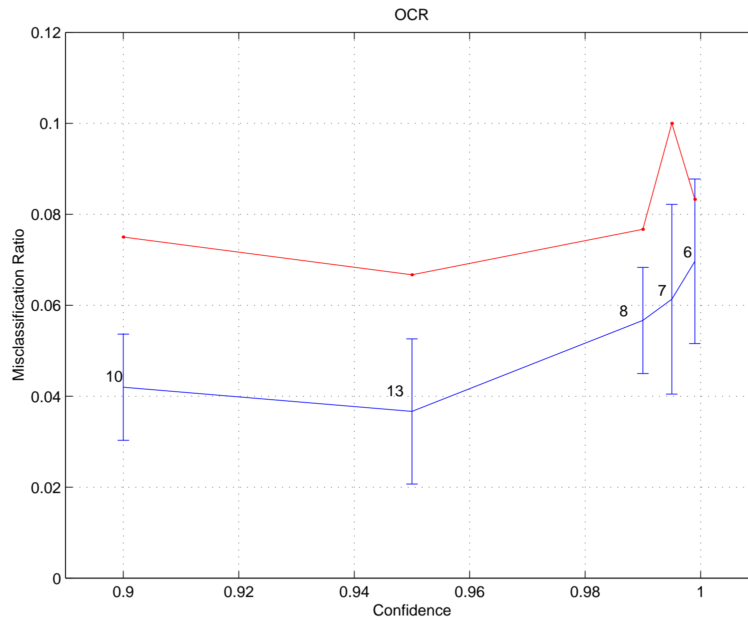


Figure 5.12. The error graph and the number of hidden nodes found for different confidence levels in `ocr` dataset with CAST algorithm

5.2.4. MOST Results

The MOST algorithm starts with LP in `sin` dataset. The dataset has one - dimensional input and one output. When the heuristics in MOST algorithm is applied, the only architecture that can be a candidate is MLP with three hidden units. MOST algorithm selects MLP with three hidden units in comparison to LP after applying 5×2 cv F Test. Then MLP with four hidden units and MLP with two hidden layers, three hidden units in each layer is tried but not preferred to MLP with three hidden units (Figure 5.13).

A classification dataset, `ocr`, has 256 - dimensional input and 10 outputs. When the heuristics in MOST algorithm is applied, the architectures that can be a candidate are MLP with 10 hidden units, MLP with 20 hidden units and MLP with 50 hidden units when starting with LP. MOST algorithm first tries MLP with 10 hidden units but does not select then it tries and selects MLP with 20 hidden units in comparison

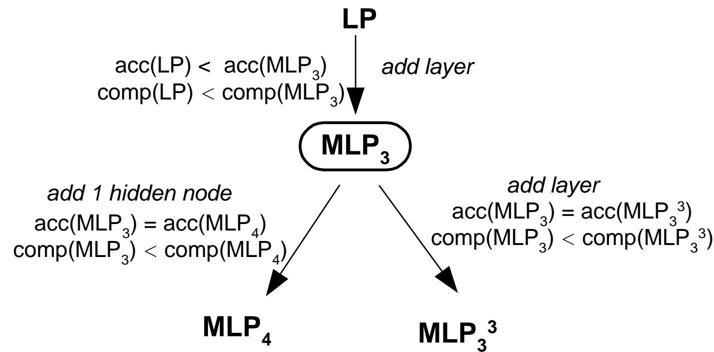


Figure 5.13. Search for `sin` dataset in MOST algorithm

to LP after applying 5×2 cv F Test. Then MLP with 11 hidden units is tried and prefered to MLP with 20 hidden units since there is no significant difference between MLP with 11 hidden units and MLP with 20 hidden units and MLP with 11 hidden units is significantly better than LP and the complexity of MLP with 11 hidden units is less than the complexity of MLP with 20 hidden units. Next candidate networks are tried but not preferred to MLP with 11 hidden units (Figure 5.14).

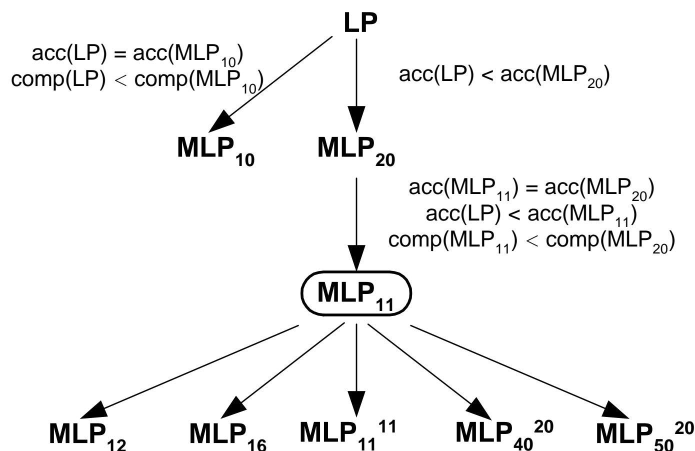


Figure 5.14. Search for `ocr` dataset in MOST algorithm

The search for `penDigits` dataset is given in this section instead of appendix since the search for `penDigits` is more interesting than the other searches. MOST

finds an architecture with two hidden layers. The MOST algorithm starts with LP and the heuristics in MOST algorithm are applied. The architectures that can be a candidate are MLP with six hidden units, MLP with 10 hidden units, MLP with 13 hidden units and MLP with 20 hidden units. MOST algorithm first tries MLP with six hidden units and selects in comparison to LP after applying 5×2 cv F Test. Then MLPs with three, five, seven and nine hidden units are tried and MLP with nine hidden units is prefered to MLP with six hidden units. Next candidate networks are tried and a MLP with two hidden layers (40 hidden units in the first layer and 20 hidden units in the second layer) is selected. In the next step the algorithm first tries to remove a percentage of nodes from the lower layer and MLP with two hidden layers (20 hidden units in the first layer and 20 hidden units in the second layer) is selected. At the end of the algorithm, MLP with two hidden layers (20 hidden units in the first layer and 19 hidden units in the second layer) is selected (Figure 5.15).

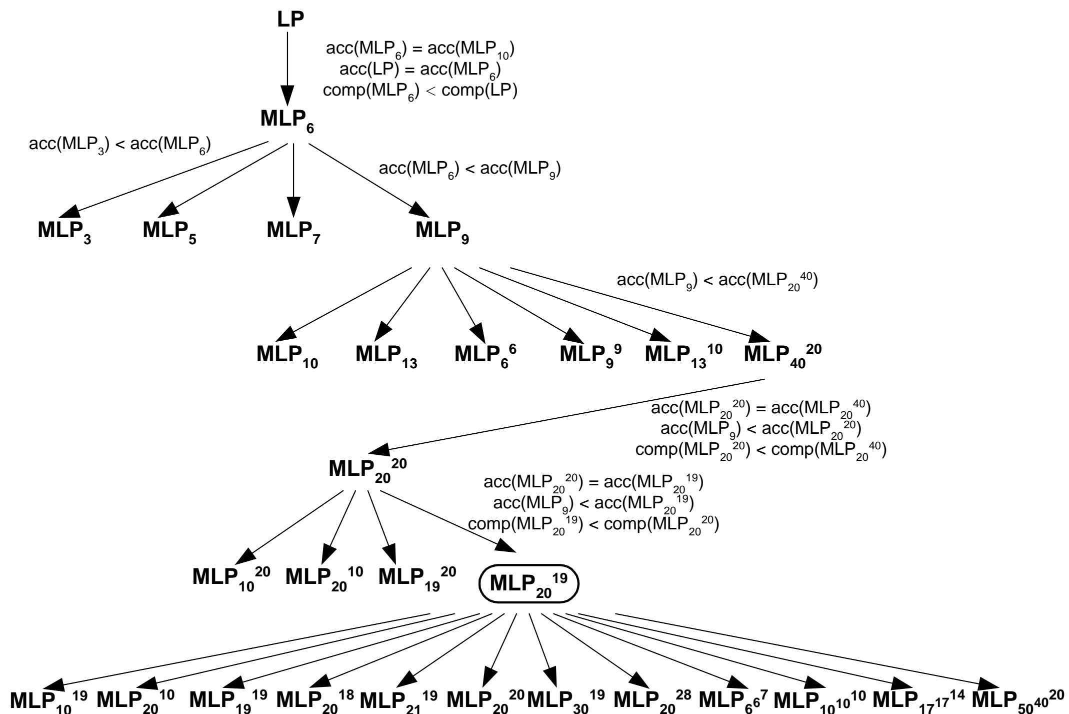


Figure 5.15. Search for `penDigits` dataset in MOST algorithm

5.2.5. Overall Comparison

DNC, modified DNC and Cascade Correlation algorithms are run on 5×2 cv datasets and the found architectures and the number of epochs trained are recorded. For each algorithm, a MLP is constructed for each dataset found and run for appropriate number of epochs. For modified DNC, since the algorithm trains the network from scratch, the number of epochs for MLP is the *width* parameter of the modified DNC algorithm.

Table 5.2 shows the results of 5×2 cross validation for `sin` dataset. In `sin` dataset, DNC finds a network with five hidden units, modified DNC with three hidden units and Cascade correlation with 10 hidden units. The high average error in modified DNC is because of the high variance of two and three hidden unit architectures. Modified DNC selects a two hidden unit architecture in three runs, a three hidden unit architecture in six runs and a four hidden unit architecture in one runs among 10 runs in 5×2 cv. DNC selects a five hidden unit architecture in all 10 runs and performs good with a low average error and a low variance. The average error of Cascade correlation algorithm is also low but when we apply 5×2 cv F Test (Table 5.3) with 90 percent confidence, DNC - Cascade Correlation and MLP - Cascade Correlation results are different. Since the variances of each algorithm is low, the small difference in average errors lead to significancy. The results of other algorithms are statistically insignificant.

The results of 5×2 cross validation for `ocr` dataset is given in Table 5.4. Although DNC finds a more complex network than other algorithms, the performance of DNC is poor in `ocr` dataset. DNC finds network with 10 hidden units, modified DNC with seven hidden units and Cascade correlation with 0 hidden units. The difference of the results of DNC and all other algorithms are statistically significant (Table 5.5). An interesting result is that cascade correlation finds an architecture with no hidden units. The results with this architecture is statistically insignificant when compared to MLP up to 10 hidden units with 90 percent confidence. The most successful algorithm in this dataset is the CAST algorithm. Cast algorithm is significantly better than all the other algorithms with 95 percent confidence.

Table 5.2. 5×2 cv results for `sin` dataset

Methods	Number of hidden	Number of epochs trained	Test Error
DNC	5 ± 0	1000 ± 0	0.0465 ± 0.0022
MLP	5 ± 0	1000 ± 0	0.0443 ± 0.0019
modified DNC	2.6 ± 0.8	361.5 ± 85.3	0.1722 ± 0.1616
MLP	2.6 ± 0.8	100 ± 0	0.2324 ± 0.1908
Cascade Correlation	10 ± 0	392.8 ± 20.3	0.0606 ± 0.0073
MLP	10 ± 0	392.8 ± 20.3	0.0425 ± 0.0015
CAST	3 ± 0	3000 ± 0	0.1518 ± 0.1670
MOST	3 ± 0	3000 ± 0	0.1824 ± 0.1730

The results of all algorithms on all datasets are given in Table 5.6. First the number of hidden units found is given, then the average error and the standard deviation in 10 runs is given in each cell. Newman-Keuls [22] range test is applied to these results and test results are given in Table 5.7. B $>$ A means that the error of B is greater than A so A is a better classifier. B $=$ A means that the two classifiers are the same.

Table 5.3. 5×2 cv F test results for `sin` dataset

Methods	F Value	Confidence
DNC - Modified DNC	1.656445	0.699
DNC - MLP	2.008328	0.771
Modified DNC - MLP	1.119695	0.521
Cascade Correlation - DNC	3.376683	0.904
Cascade Correlation - Modified DNC	1.541563	0.669
Cascade Correlation - MLP	3.602335	0.915
CAST - DNC	1.087174	0.506
CAST - Modified DNC	0.977848	0.454
CAST - Cascade Correlation	0.996733	0.463
MOST - DNC	1.819948	0.736
MOST - Modified DNC	0.971113	0.451
MOST - Cascade Correlation	1.729616	0.717
MOST - CAST	0.954071	0.442

Table 5.4. 5×2 cv results for `ocr` dataset

Methods	Number of hidden	Number of epochs trained	Test Error
DNC	10 ± 0	3000 ± 0	0.1413 ± 0.0312
MLP	10 ± 0	3000 ± 0	0.0473 ± 0.0091
modified DNC	6.5 ± 0.5	2255.5 ± 155.5	0.0689 ± 0.0093
MLP	6.5 ± 0.5	300 ± 0	0.0719 ± 0.0198
Cascade Correlation	0 ± 0	11.4 ± 1.5	0.0711 ± 0.0094
LP	0 ± 0	11.4 ± 1.5	0.0748 ± 0.0476
CAST	13 ± 0	6500 ± 0	0.0318 ± 0.0159
MOST	11 ± 0	4500 ± 0	0.0450 ± 0.0148

Table 5.5. 5×2 cv F test results for `ocr` dataset

Methods	F Value	Confidence
DNC - Modified DNC	4.257248	0.938
DNC - MLP	11.7125	0.993
Modified DNC - MLP	0.609764	0.236
Cascade Correlation - DNC	5.914895	0.968
Cascade Correlation - Modified DNC	0.871378	0.397
Cascade Correlation - LP	1.693323	0.708
Cascade Correlation - MLP	3.602335	0.915
CAST - DNC	10.400801	0.991
CAST - Modified DNC	6.275995	0.972
CAST - Cascade Correlation	6.096275	0.970
MOST - DNC	10.303832	0.991
MOST - Modified DNC	2.348676	0.821
MOST - Cascade Correlation	2.895577	0.874
MOST - CAST	0.790399	0.350

Table 5.6. Overall results

	MLP	cascade	DNC	modDNC	CAST	MOST
sine	3: 0.04 ± 0.12	3: 0.04 ± 0.00	5: 0.00 ± 0.00	4: 0.03 ± 0.09	3: 0.03 ± 0.18	3: 0.03 ± 0.13
california	10: 0.29 ± 0.02	4: 0.31 ± 0.01	15: 0.24 ± 0.01	3: 0.29 ± 0.01	3: 0.29 ± 0.03	3: 0.29 ± 0.01
boston	5: 0.77 ± 0.26	6: 0.32 ± 0.09	8: 0.84 ± 0.45	4: 0.88 ± 0.36	2: 0.60 ± 0.25	3: 0.97 ± 0.48
pum8fh	8: 0.41 ± 0.01	5: 0.39 ± 0.00	5: 0.44 ± 0.06	2: 0.38 ± 0.00	2: 0.44 ± 0.08	3: 0.43 ± 0.05
pum8nh	5: 0.38 ± 0.04	10: 0.34 ± 0.01	10: 0.37 ± 0.02	4: 0.35 ± 0.02	2: 0.44 ± 0.02	3: 0.42 ± 0.04
ocr	20: 0.03 ± 0.01	0: 0.04 ± 0.00	6: 0.12 ± 0.02	7: 0.07 ± 0.02	13: 0.03 ± 0.01	11: 0.04 ± 0.01
optdigits	20: 0.04 ± 0.00	7: 0.06 ± 0.00	12: 0.08 ± 0.00	13: 0.05 ± 0.01	18: 0.04 ± 0.00	21: 0.03 ± 0.00
pendigits	20: 0.03 ± 0.00	20: 0.03 ± 0.00	11: 0.04 ± 0.01	13: 0.04 ± 0.01	5: 0.09 ± 0.02	$\frac{19}{20}$: 0.02 ± 0.01

Table 5.7. Newman-Keuls range test results

`sin`	All classifiers are equal
`california`	cascade=CAST=MOST > MLP=modified DNC > DNC
`boston`	cascade is better than all other classifiers
`pum8fh`	modified DNC is better than all other classifiers
`pum8nh`	CAST=MOST > MLP=DNC > cascade=modified DNC
`ocr`	DNC > modified DNC > MLP=cascade=CAST=MOST
`optDigits`	DNC > cascade=modified DNC > MLP=CAST=MOST
`penDigits`	CAST > DNC=modified DNC > MLP=cascade > MOST

6. CONCLUSIONS

The optimal neural network architecture is the architecture which generalizes the underlying function of a given dataset. Too complex or too simple architectures fail to learn this underlying function. Determining the architecture by trial and error takes too much time and can eliminate some architectures that can be successful. Determining the architecture for a given problem in the learning process is the desired goal.

In this work, we proposed three algorithms for the automatic construction of neural networks. The first one is a modified version of DNC which aims to overcome the problems that are faced in DNC algorithms. CAST algorithm is similar to modified DNC algorithm but it uses 5×2 cv F Test for selecting one of two architectures. CAST algorithm constructs a network with only one hidden layer and decides the number of nodes in that layer by starting with one hidden node and incrementally adds hidden nodes until there is no significant improvement. This approach has two disadvantages: First, it eliminates the other architectures and second, addition of one hidden node can be insignificant in some problems. To overcome these problems, we proposed MOST algorithm. In MOST, the assumption of single hidden layer is discarded and the resulting network can have more than one hidden layer or no hidden layers. In choosing the candidate network, more than one architecture is considered and the one that significantly increases the performance is selected, if any. The results of the algorithms are promising. MOST algorithm finds near optimal results and is useful since the only extra parameter it needs is the confidence value and it can easily be set. The disadvantage of the algorithm is that the time complexity is high since it performs 5×2 cross validation for each architecture.

As a future work, some other statistical tests that do not need 5×2 cross validation can be applied in order to reduce the time complexity of the algorithm. The effect of confidence parameter can be examined in detail. Some other heuristics that are used to determine the candidate architectures can be applied so that the search space gets smaller.

APPENDIX A: PARAMETERS USED IN THE ALGORITHMS

Parameters that are used in the experiments for each dataset is given in Tables A.1– A.8.

Table A.1. Parameter set used in `sin` dataset

w	60
D_r	1.0
C_e	0.05
$threshold$	0.01
η	0.1
α	0.6

Table A.2. Parameter set used in `california` dataset

w	100
D_r	0.1
C_e	0.18
$threshold$	0.01
η	0.01
α	0.6

Table A.3. Parameter set used in `boston` dataset

w	100
D_r	0.13
C_e	0.12
$threshold$	0.01
η	0.1
α	0.6

Table A.4. Parameter set used in `pum8fh` dataset

w	100
D_r	0.13
C_e	0.36
$threshold$	0.01
η	0.1
α	0.6

Table A.5. Parameter set used in `pum8nh` dataset

w	100
D_r	0.13
C_e	0.30
$threshold$	0.01
η	0.1
α	0.6

Table A.6. Parameter set used in `ocr` dataset

w	100
D_r	0.25
C_e	0.0001
$threshold$	0.01
η	0.1
α	0.6

Table A.7. Parameter set used in `optDigits` dataset

w	50
D_r	0.1
C_e	0.0001
$threshold$	0.01
η	0.1
α	0.6

Table A.8. Parameter set used in `penDigits` dataset

w	50
D_r	0.1
C_e	0.0001
$threshold$	0.01
η	0.1
α	0.6

APPENDIX B: MLP RESULTS

The error graphs and the effect of the number of hidden units in MLP are given in Figures B.1– B.12.

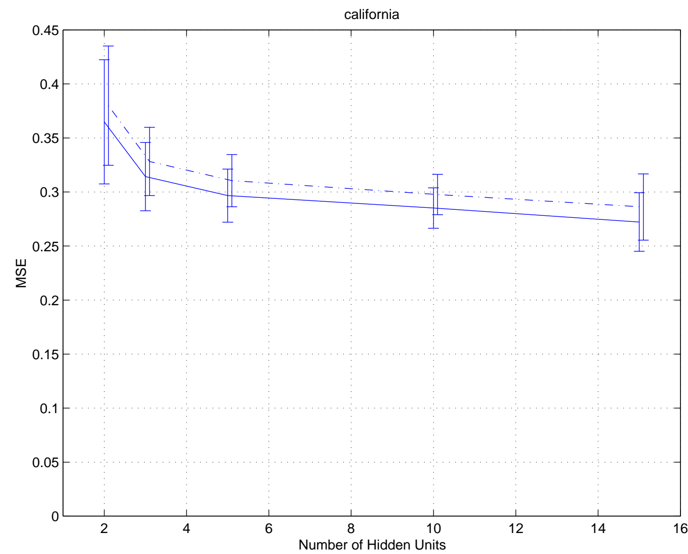


Figure B.1. The effect of the number of hidden units for `california` dataset

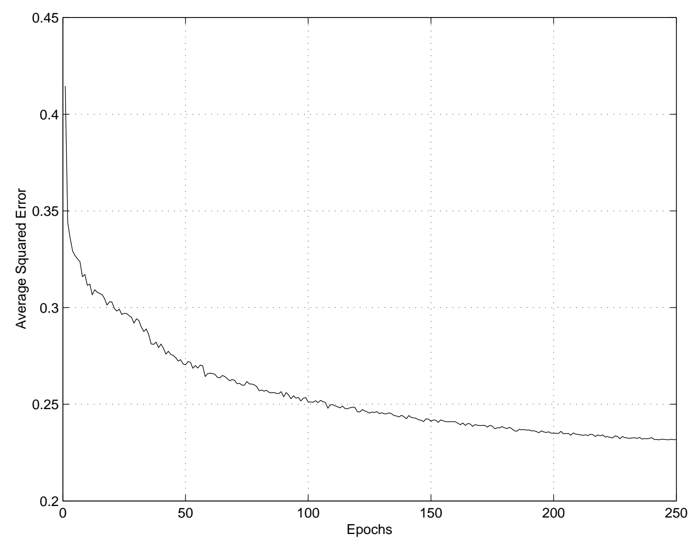


Figure B.2. The error graph during training of `california` dataset in MLP

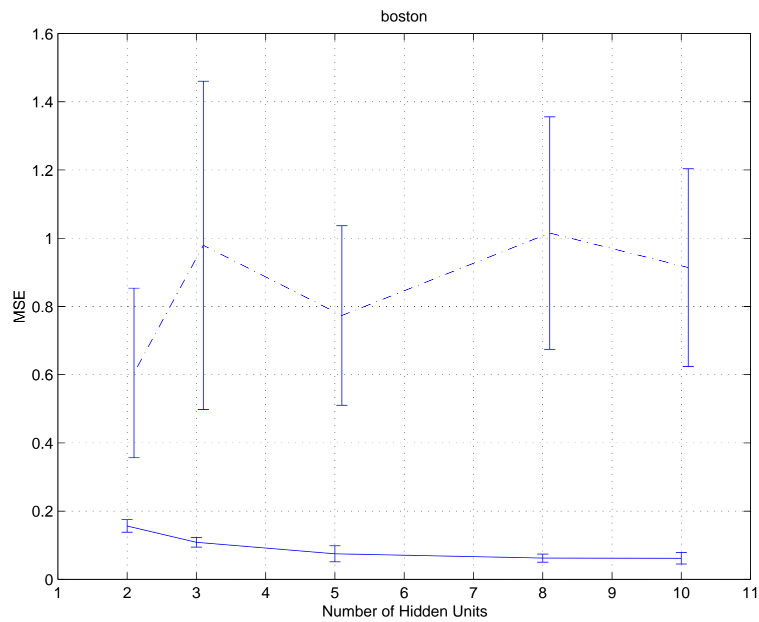


Figure B.3. The effect of the number of hidden units for `boston` dataset

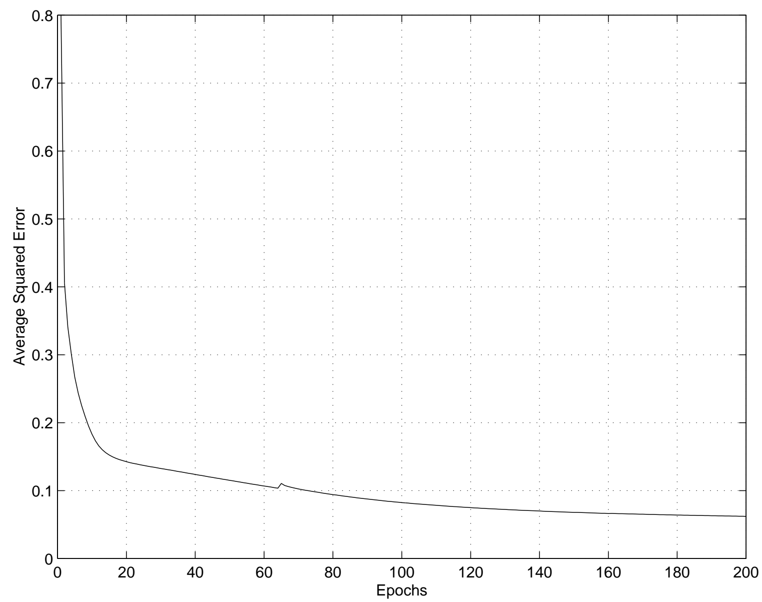


Figure B.4. The error graph during training of `boston` dataset in MLP

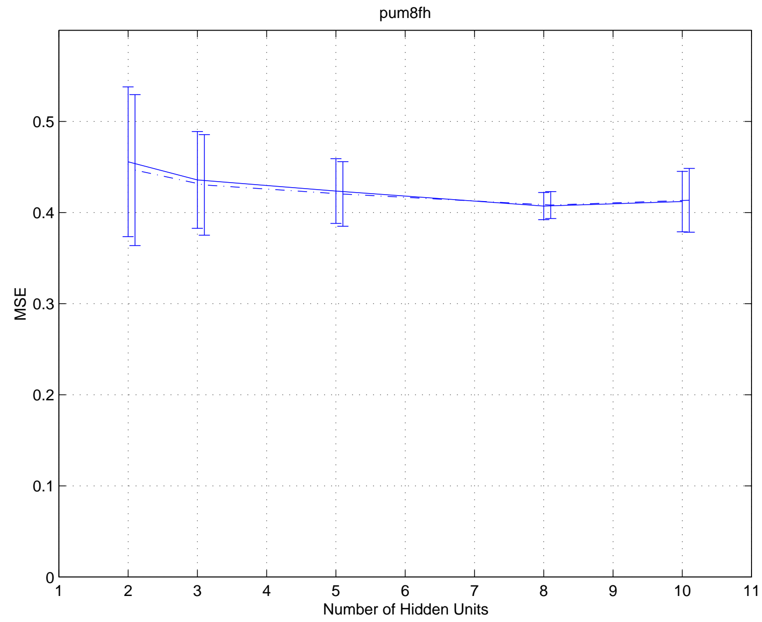


Figure B.5. The effect of the number of hidden units for `pum8fh` dataset

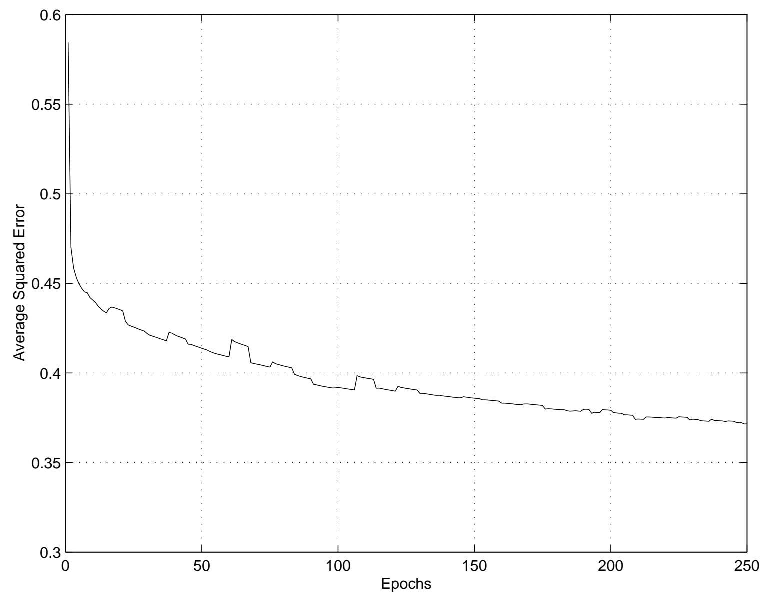


Figure B.6. The error graph during training of `pum8fh` dataset in MLP

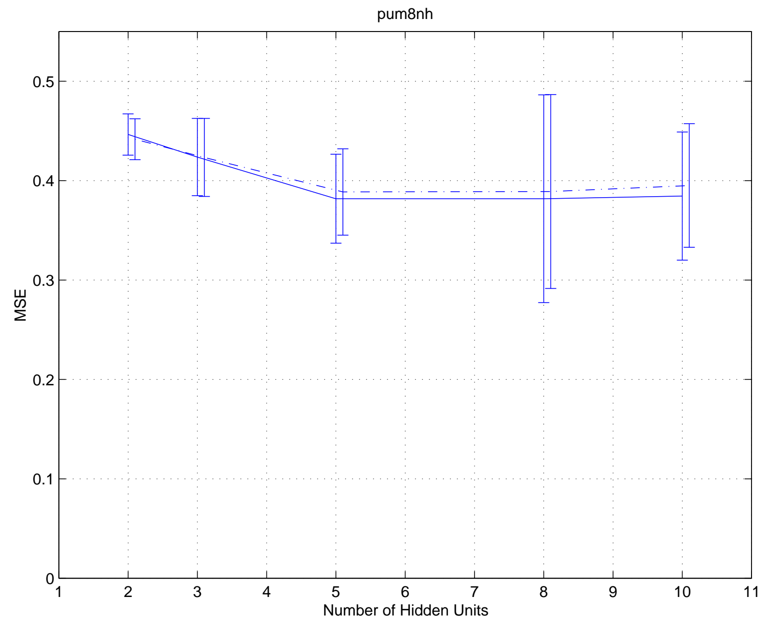


Figure B.7. The effect of the number of hidden units for `pum8nh` dataset

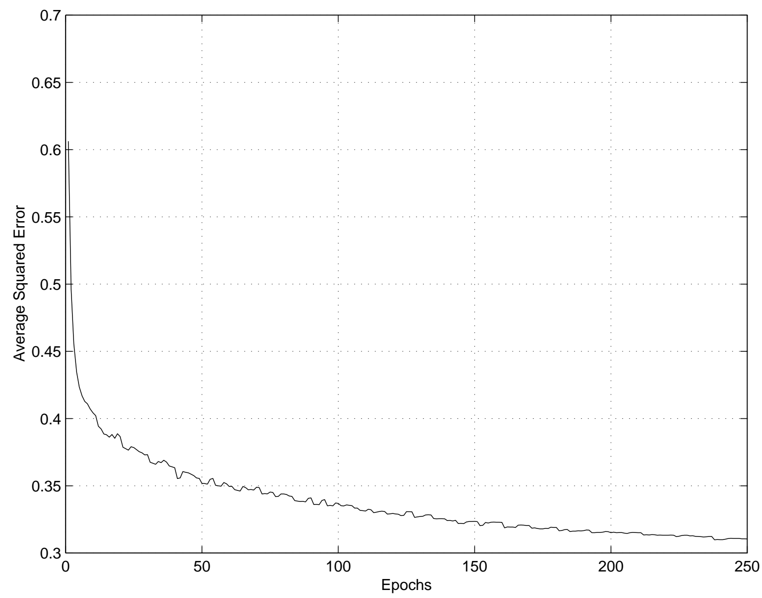


Figure B.8. The error graph during training of `pum8nh` dataset in MLP

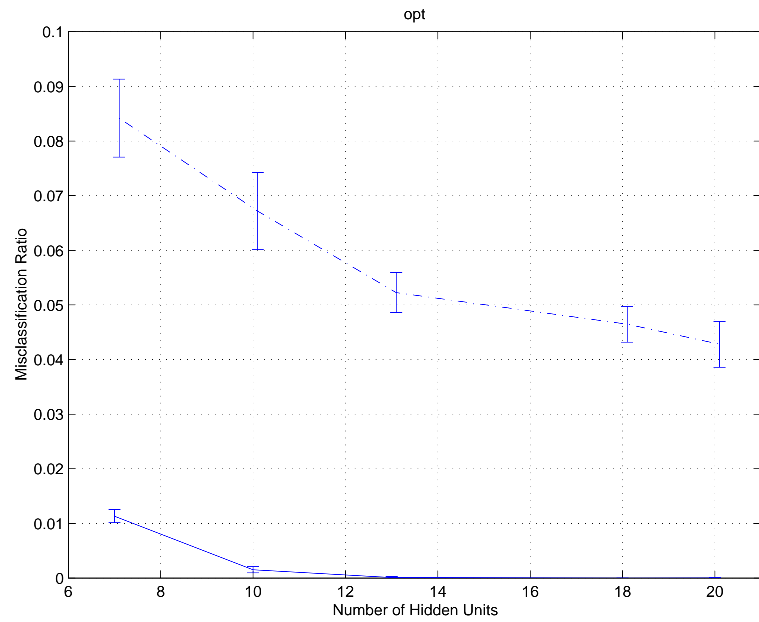


Figure B.9. The effect of the number of hidden units for `optDigits` dataset

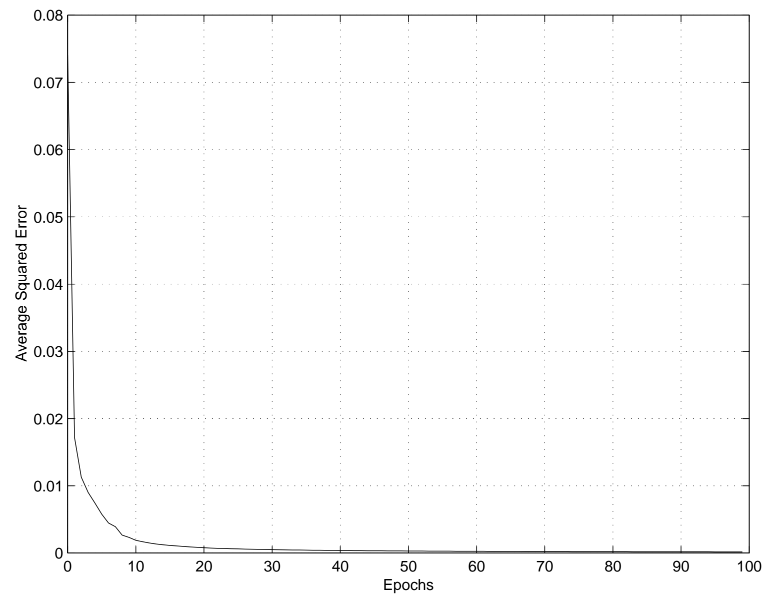


Figure B.10. The error graph during training of `optDigits` dataset in MLP

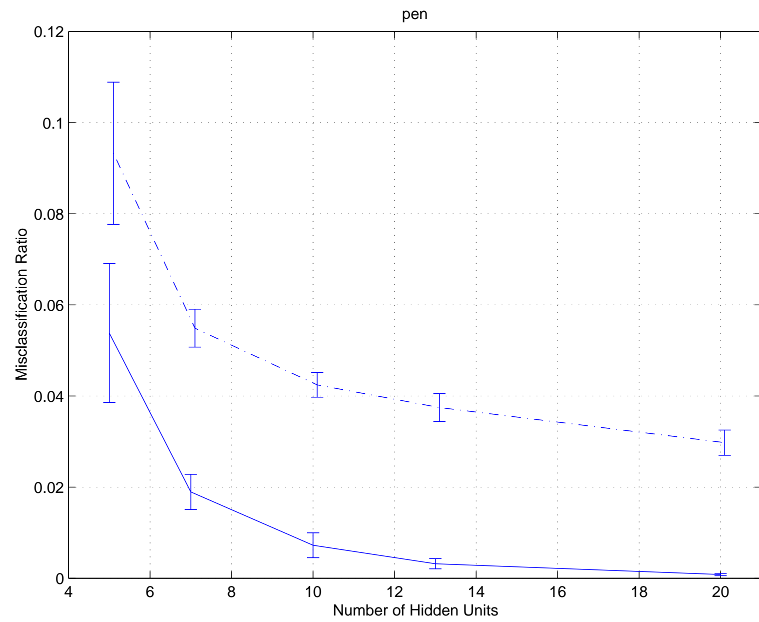


Figure B.11. The effect of the number of hidden units for `penDigits` dataset

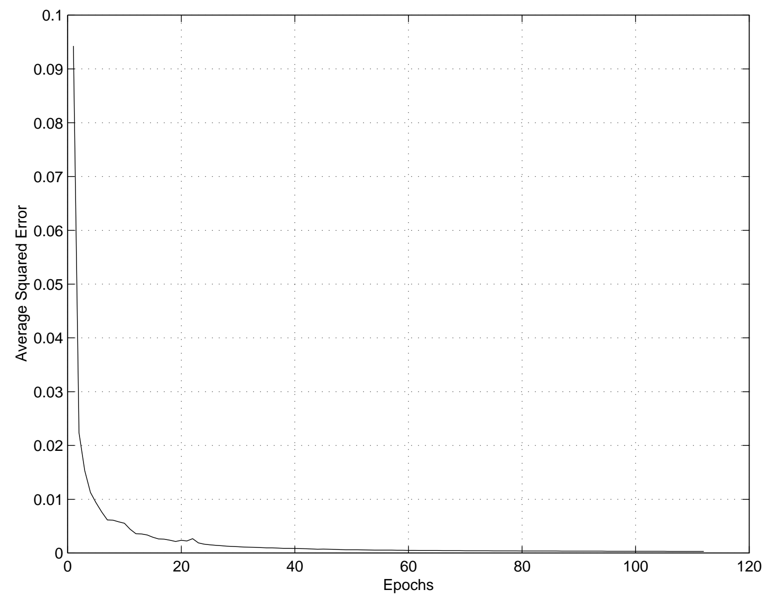


Figure B.12. The error graph during training of `penDigits` dataset in MLP

APPENDIX C: MODIFIED DNC RESULTS

The error graphs in DNC and modified DNC are given in Figures C.1 - C.12.

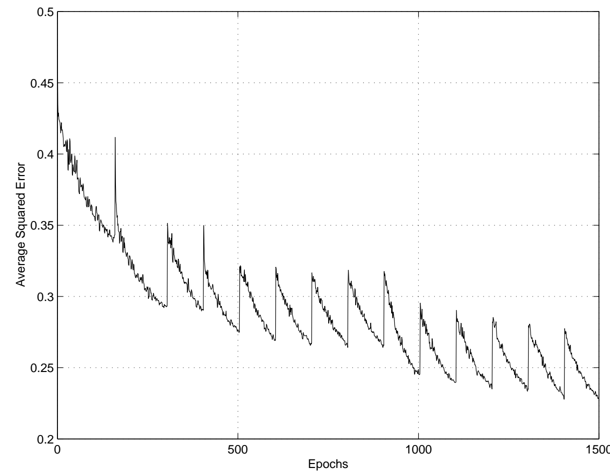


Figure C.1. The error graph during training of `california` dataset with the DNC algorithm

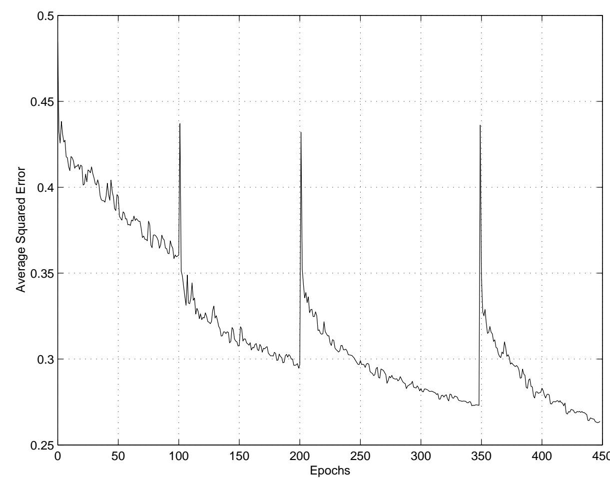


Figure C.2. The error graph during training of `california` dataset with the modified DNC algorithm

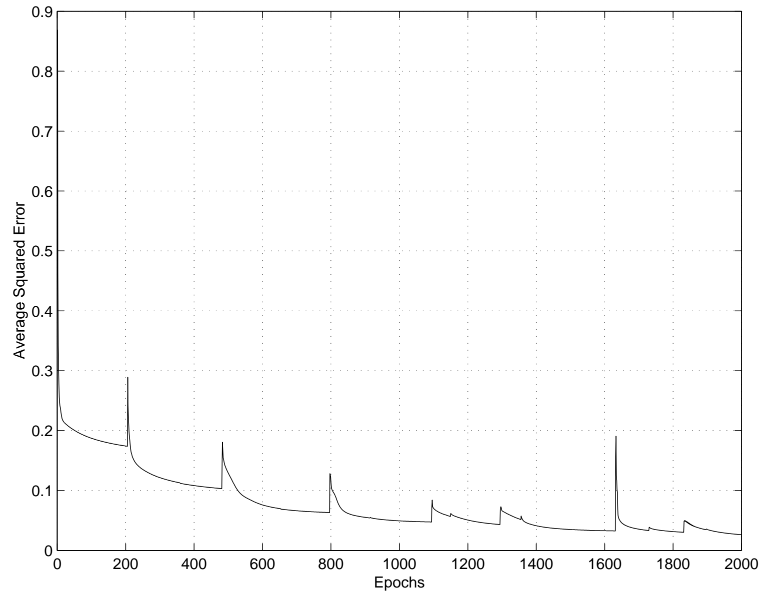


Figure C.3. The error graph during training of `boston` dataset with the DNC algorithm

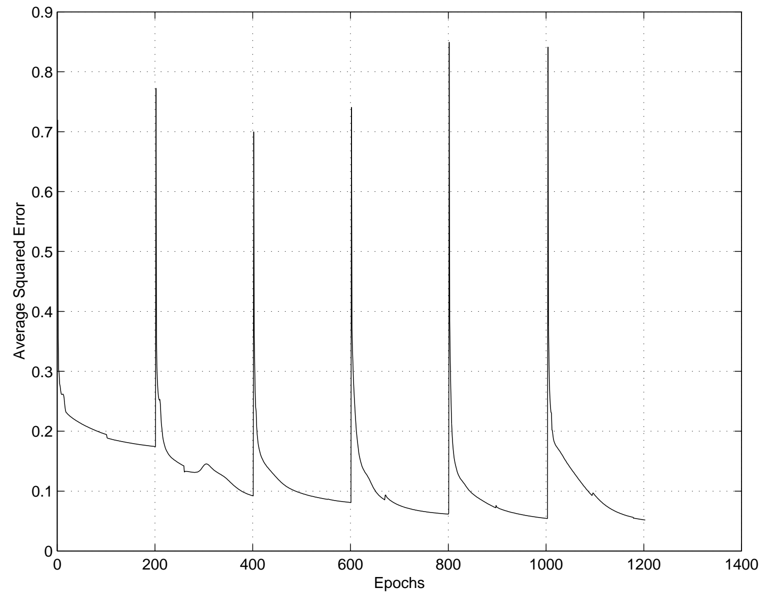


Figure C.4. The error graph during training of `boston` dataset with the modified DNC algorithm

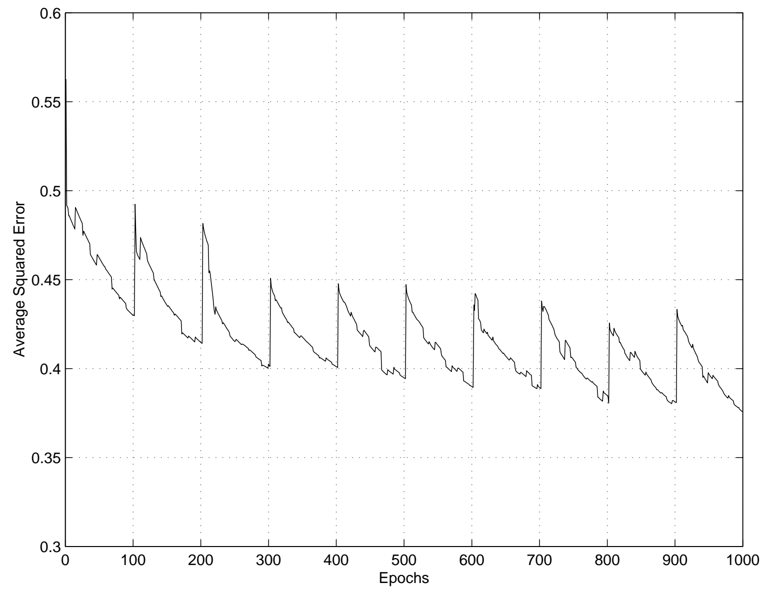


Figure C.5. The error graph during training of `pum8fh` dataset with the DNC algorithm

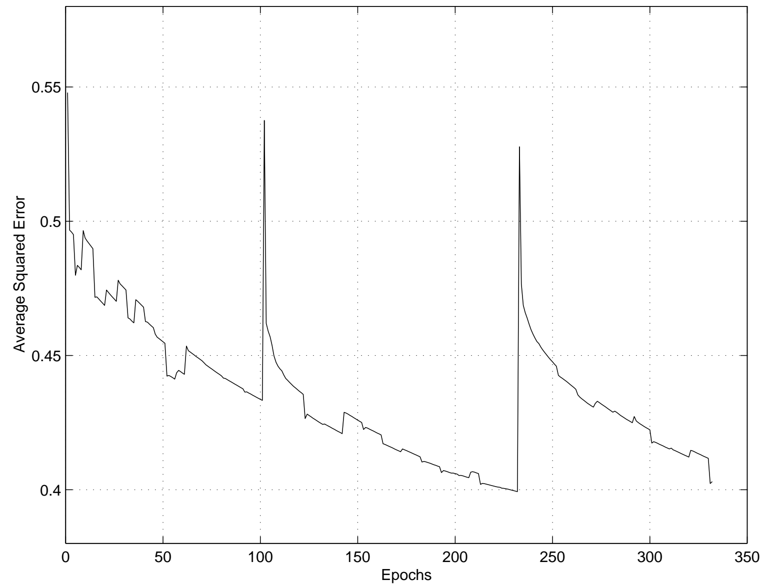


Figure C.6. The error graph during training of `pum8fh` dataset with the modified DNC algorithm

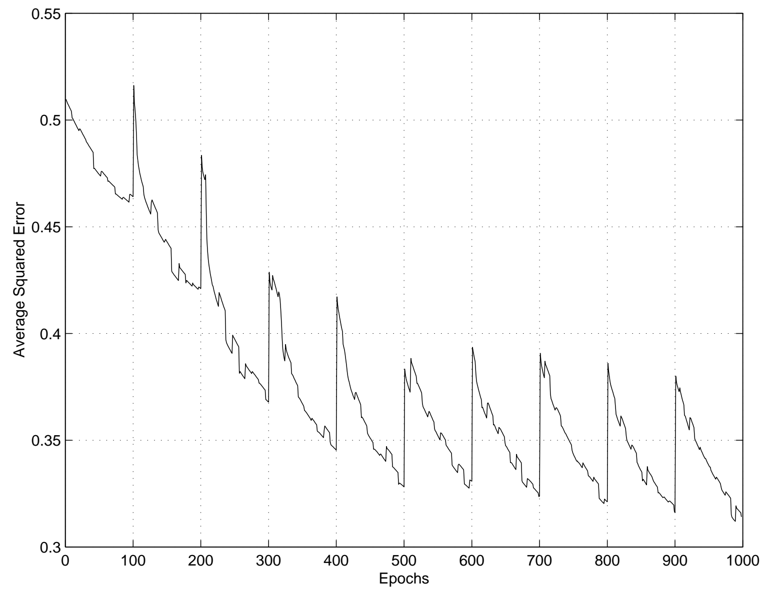


Figure C.7. The error graph during training of `pum8nh` dataset with the DNC algorithm

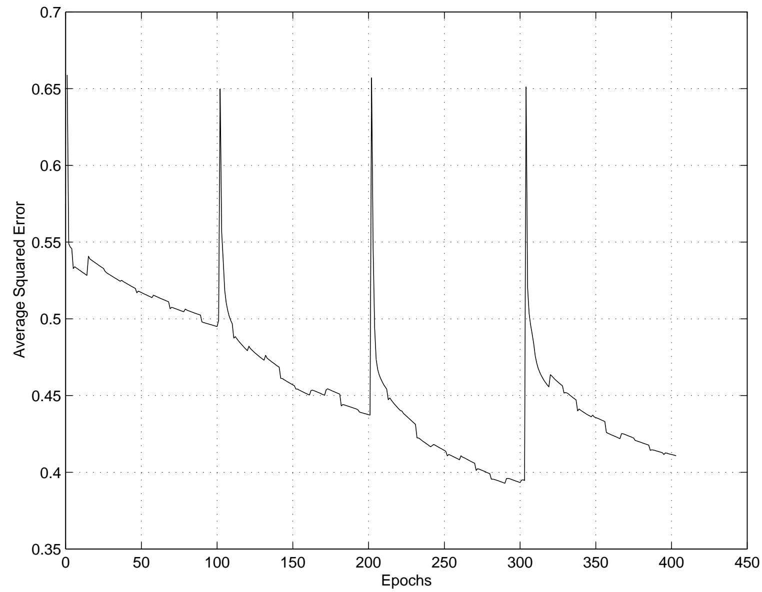


Figure C.8. The error graph during training of `pum8nh` dataset with the modified DNC algorithm

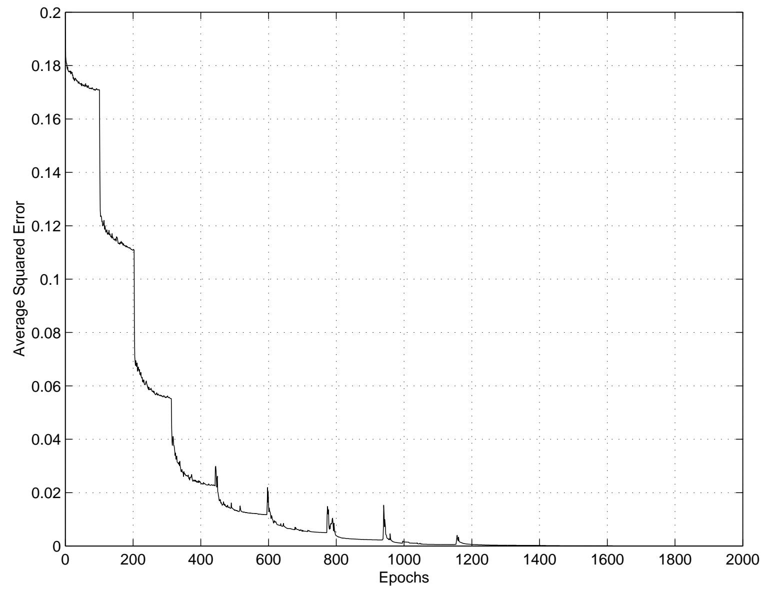


Figure C.9. The error graph during training of `optDigits` dataset with the DNC algorithm

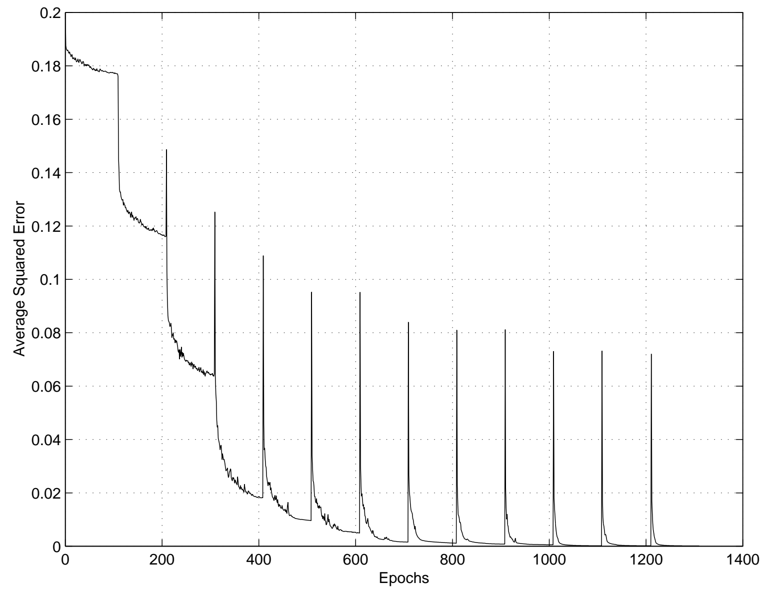


Figure C.10. The error graph during training of `optDigits` dataset with the modified DNC algorithm

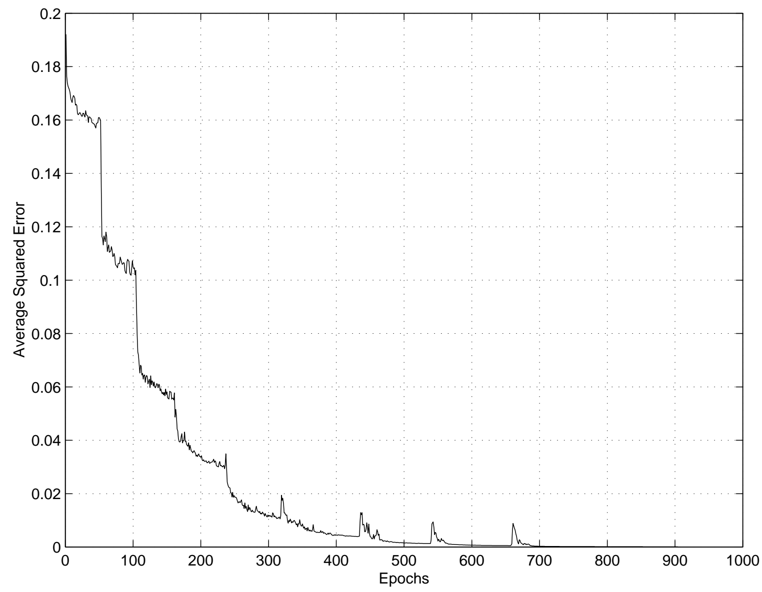


Figure C.11. The error graph during training of `penDigits` dataset with the DNC algorithm

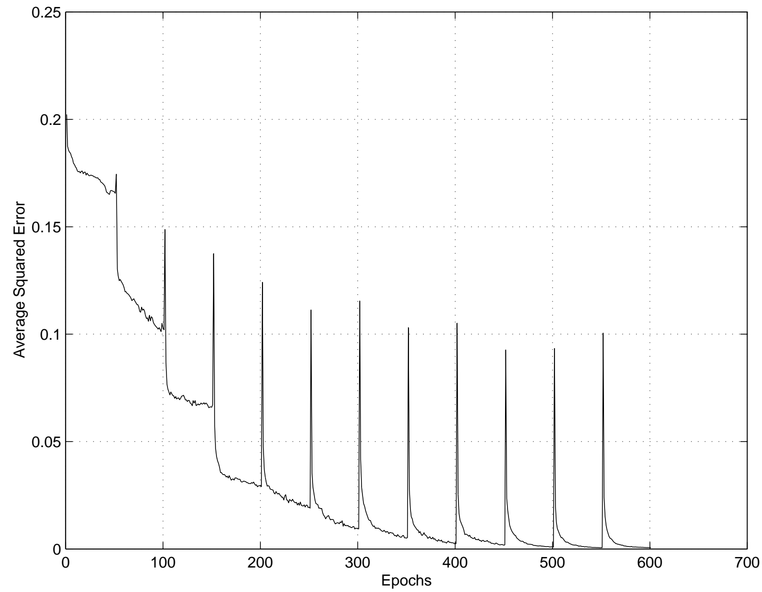


Figure C.12. The error graph during training of `penDigits` dataset with the modified DNC algorithm

APPENDIX D: CAST RESULTS

The effect of confidence parameter in CAST algorithm is given in Figures D.1–D.6.

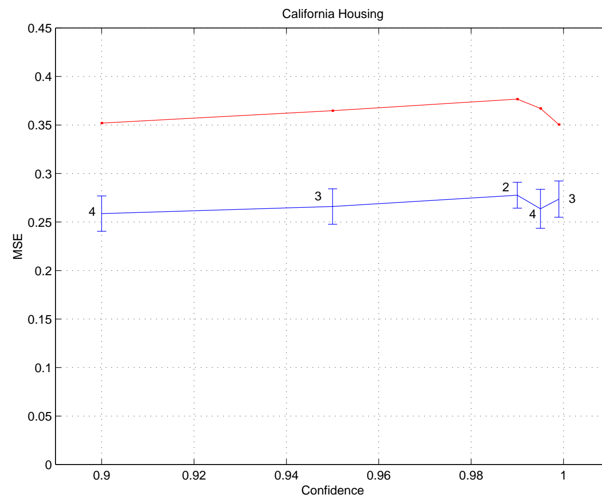


Figure D.1. The error graph and the number of hidden nodes found for different confidence levels in `california` dataset in CAST algorithm

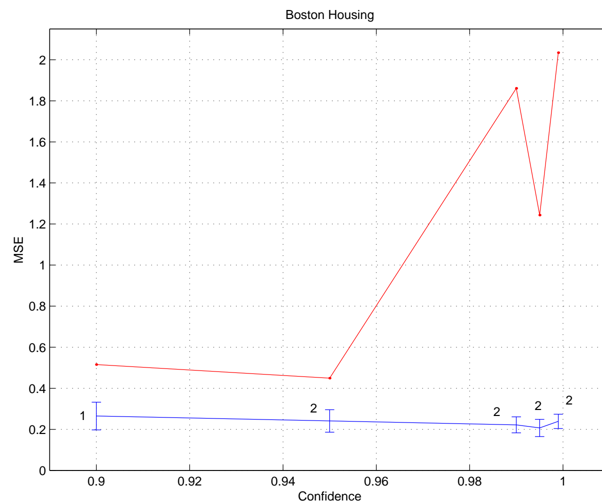


Figure D.2. The error graph and the number of hidden nodes found for different confidence levels in `boston` dataset in CAST algorithm

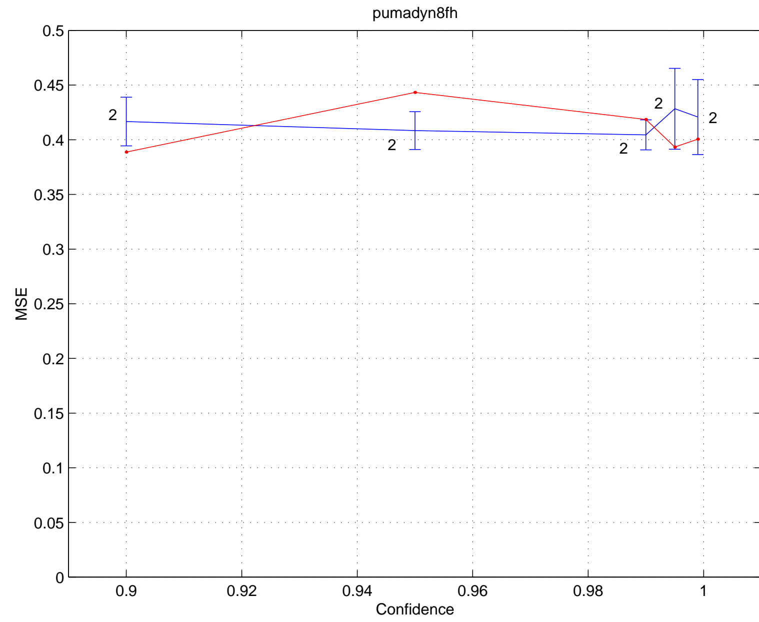


Figure D.3. The error graph and the number of hidden nodes found for different confidence levels in `pum8fh` dataset in CAST algorithm

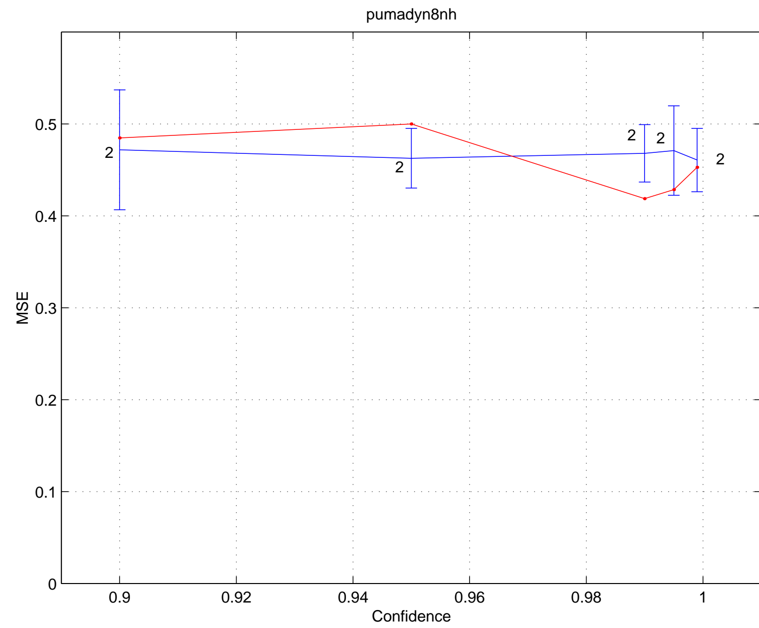


Figure D.4. The error graph and the number of hidden nodes found for different confidence levels in `pum8nh` dataset in CAST algorithm

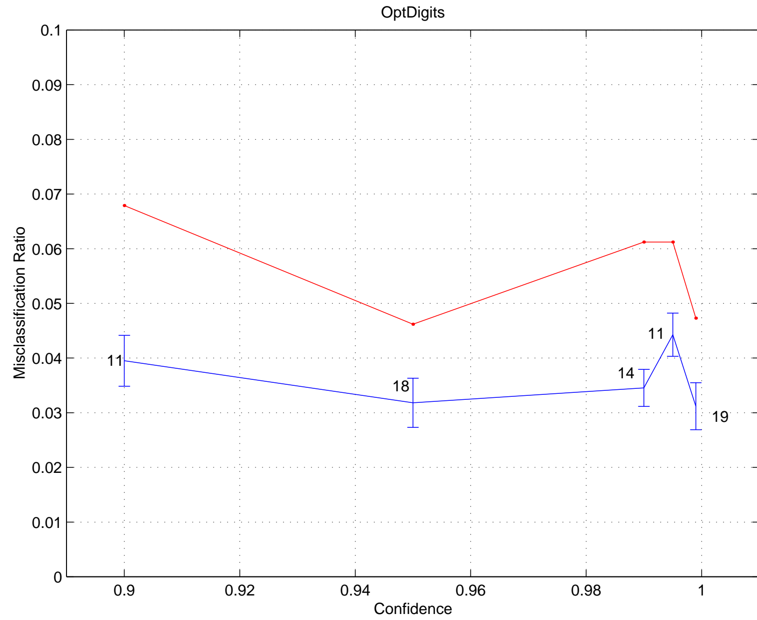


Figure D.5. The error graph and the number of hidden nodes found for different confidence levels in `optDigits` dataset in CAST algorithm

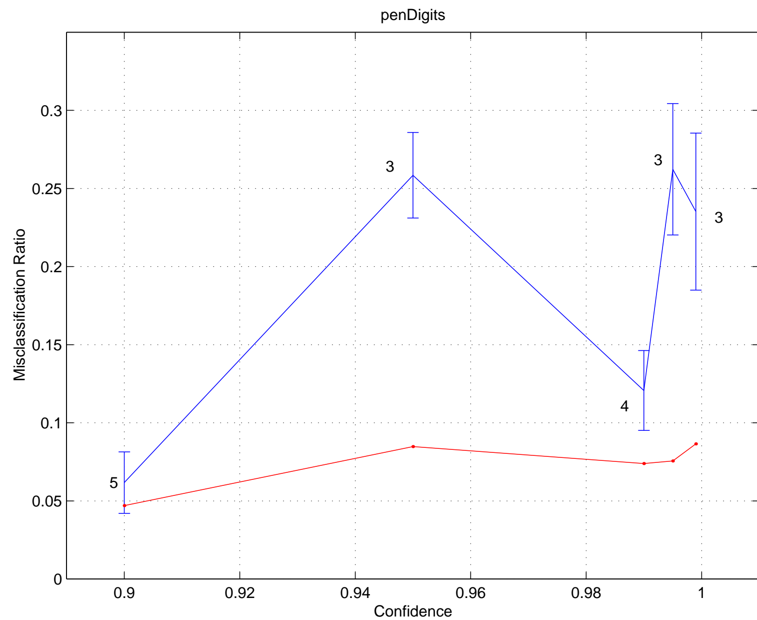


Figure D.6. The error graph and the number of hidden nodes found for different confidence levels in `penDigits` dataset in CAST algorithm

APPENDIX E: MOST RESULTS

The searches in MOST algorithm are given in Figures E.1– E.5.

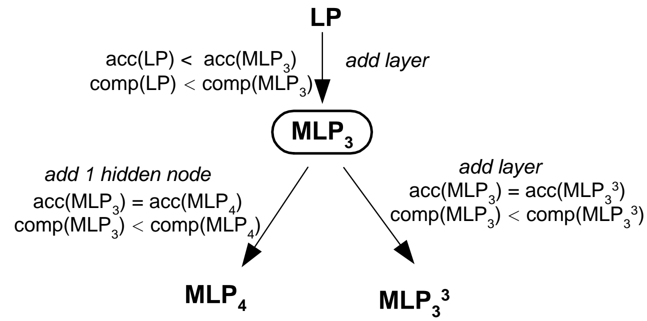


Figure E.1. Search for `california` dataset in MOST algorithm

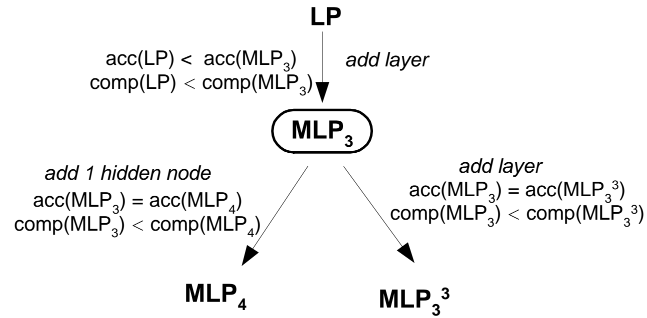


Figure E.2. Search for `boston` dataset in MOST algorithm

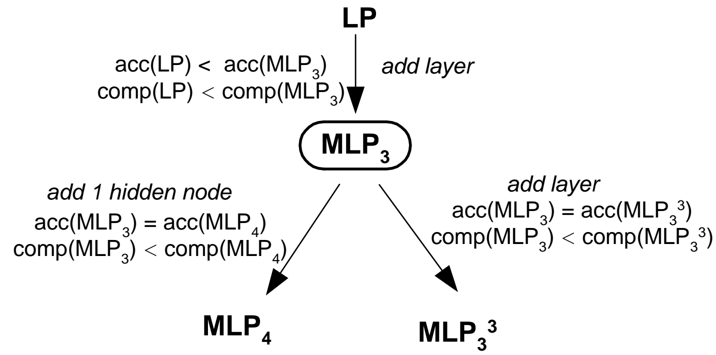


Figure E.3. Search for `pum8fh` dataset in MOST algorithm

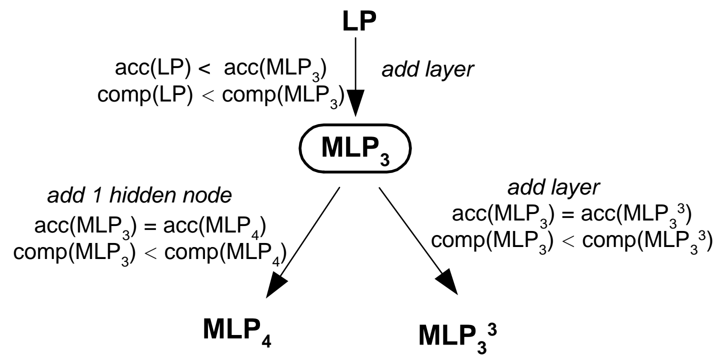


Figure E.4. Search for `pum8nh` dataset in MOST algorithm

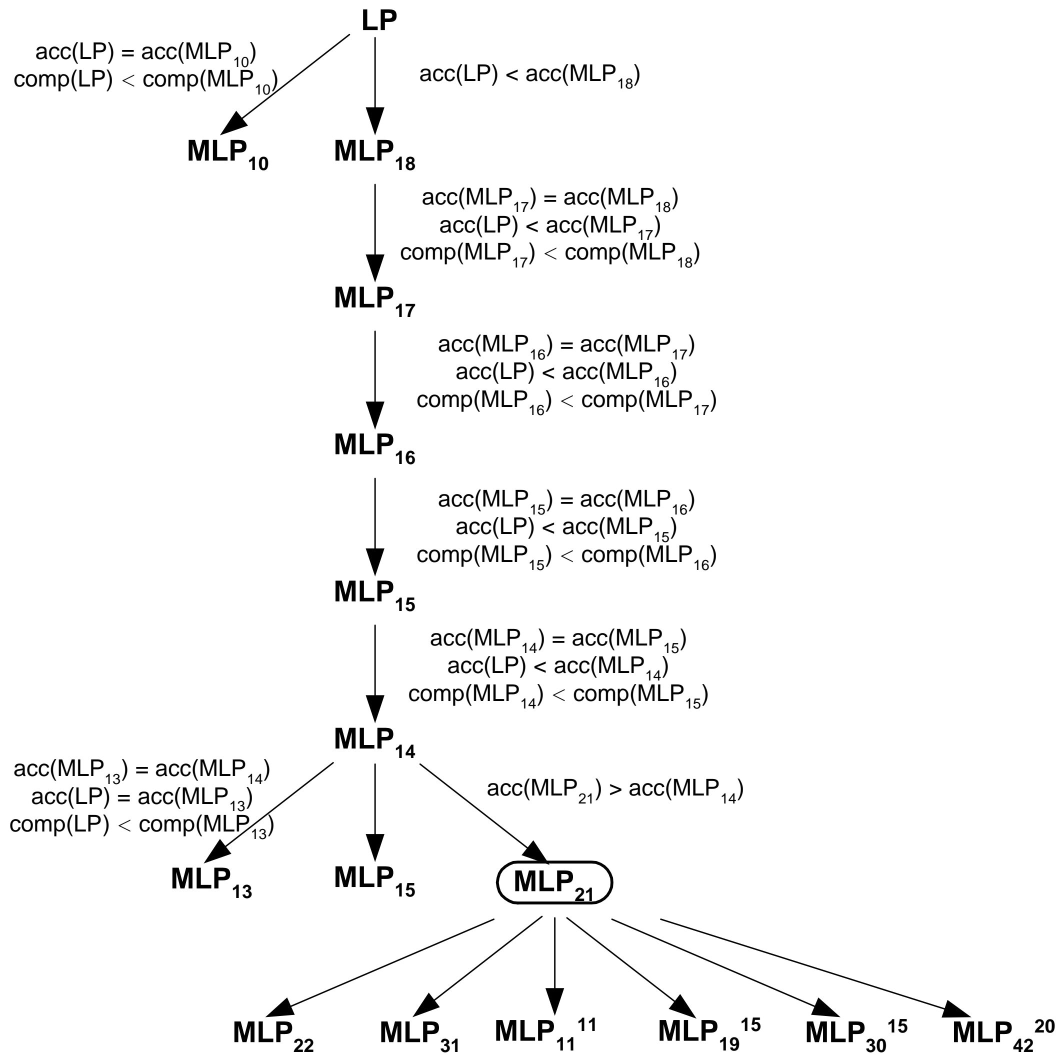


Figure E.5. Search for `optDigits` dataset in MOST algorithm

APPENDIX F: CROSS VALIDATION RESULTS

Table F.1. 5×2 cv results for `california` dataset

Methods	Number of hidden	Number of epochs trained	Test Error
DNC	14.7 ± 0.5	1500 ± 0	0.2600 ± 0.0309
MLP	14.7 ± 0.5	1500 ± 0	0.2154 ± 0.0074
modified DNC	3.1 ± 0.3	444.1 ± 50.5	0.2655 ± 0.0202
MLP	3.1 ± 0.3	100 ± 0	0.2982 ± 0.0961
Cascade Correlation	3.9 ± 1.2	351.6 ± 86.1	0.2844 ± 0.0139
MLP	3.9 ± 1.2	351.6 ± 86.1	0.2976 ± 0.0923
CAST	3 ± 0	3000 ± 0	0.2660 ± 0.0183
MOST	3 ± 0	3000 ± 0	0.2765 ± 0.0211

Table F.2. 5×2 cv F test results for `california` dataset

Methods	F Value	Confidence
DNC - Modified DNC	1.285054	0.587
DNC - MLP	5.588247	0.964
Modified DNC - MLP	1.147056	0.533
Cascade Correlation - DNC	2.509210	0.839
Cascade Correlation - Modified DNC	1.558864	0.674
Cascade Correlation - MLP	0.996444	0.463
CAST - DNC	1.547103	0.671
CAST - Modified DNC	0.617431	0.241
CAST - Cascade Correlation	3.719295	0.920
MOST - DNC	2.123205	0.790
MOST - Modified DNC	0.912444	0.420
MOST - Cascade Correlation	0.851187	0.386
MOST - CAST	1.257656	0.577

Table F.3. 5×2 cv results for `boston` dataset

Methods	Number of hidden	Number of epochs trained	Test Error
DNC	7.8 ± 0.4	2000 ± 0	0.2359 ± 0.0836
MLP	7.8 ± 0.4	2000 ± 0	0.1834 ± 0.0459
modified DNC	4.6 ± 0.7	999.9 ± 156.3	0.1898 ± 0.0507
MLP	4.6 ± 0.7	200 ± 0	0.2241 ± 0.0472
Cascade Correlation	10 ± 0	1042.2 ± 41.9	0.2548 ± 0.0437
MLP	10 ± 0	1042.2 ± 41.9	0.1822 ± 0.0426
CAST	2 ± 0	2000 ± 0	0.2410 ± 0.0549
MOST	3 ± 0	3000 ± 0	0.2196 ± 0.0556

Table F.4. 5×2 cv F test results for `boston` dataset

Methods	F Value	Confidence
DNC - Modified DNC	0.955452	0.442
DNC - MLP	1.110555	0.517
Modified DNC - MLP	0.968601	0.449
Cascade Correlation - DNC	1.03494	0.482
Cascade Correlation - Modified DNC	2.628582	0.851
Cascade Correlation - MLP	1.346714	0.609
CAST - DNC	0.780913	0.345
CAST - Modified DNC	3.929547	0.928
CAST - Cascade Correlation	2.146828	0.794
MOST - DNC	0.762948	0.334
MOST - Modified DNC	1.215565	0.561
MOST - Cascade Correlation	2.737775	0.861
MOST - CAST	1.040126	0.4846

Table F.5. 5×2 cv results for `pum8fh` dataset

Methods	Number of hidden	Number of epochs trained	Test Error
DNC	10 ± 0	1000 ± 0	0.4439 ± 0.0206
MLP	10 ± 0	1000 ± 0	0.4115 ± 0.0069
modified DNC	2 ± 0	351.2 ± 12.4	0.4062 ± 0.0147
MLP	2 ± 0	100 ± 0	0.4213 ± 0.0171
Cascade Correlation	5 ± 0	424.2 ± 20.5	0.4136 ± 0.0098
MLP	5 ± 0	424.2 ± 20.5	0.4152 ± 0.0124
CAST	2 ± 0.0	2000 ± 0	0.4084 ± 0.0172
MOST	3 ± 0.0	3000 ± 0	0.4341 ± 0.0447

Table F.6. 5×2 cv F test results for `pum8fh` dataset

Methods	F Value	Confidence
DNC - Modified DNC	3.690642	0.919
DNC - MLP	2.871599	0.872
Modified DNC - MLP	1.303016	0.594
Cascade Correlation - DNC	2.186016	0.799
Cascade Correlation - Modified DNC	3.085532	0.887
Cascade Correlation - MLP	0.817351	0.366
CAST - DNC	2.728974	0.860
CAST - Modified DNC	0.939561	0.434
CAST - Cascade Correlation	1.145397	0.532
MOST - DNC	1.073542	0.500
MOST - Modified DNC	1.243381	0.572
MOST - Cascade Correlation	0.879107	0.401
MOST - CAST	0.793852	0.352

Table F.7. 5×2 cv results for `pum8nh` dataset

Methods	Number of hidden	Number of epochs trained	Test Error
DNC	10 ± 0	1000 ± 0	0.3751 ± 0.0309
MLP	10 ± 0	1000 ± 0	0.3389 ± 0.0092
modified DNC	3.1 ± 0.9	471 ± 121.2	0.4007 ± 0.0430
MLP	3.1 ± 0.9	100 ± 0	0.4019 ± 0.0479
Cascade Correlation	10 ± 0	424.2 ± 20.5	0.4136 ± 0.0098
MLP	10 ± 0	424.2 ± 20.5	0.3393 ± 0.0089
CAST	2 ± 0.0	2000 ± 0	0.4626 ± 0.0325
MOST	3 ± 0.0	3000 ± 0	0.4158 ± 0.0450

Table F.8. 5×2 cv F test results for `pum8nh` dataset

Methods	F Value	Confidence
DNC - Modified DNC	0.880459	0.402
DNC - MLP	1.48677	0.654
Modified DNC - MLP	1.047314	0.488
Cascade Correlation - DNC	0.676884	0.279
Cascade Correlation - Modified DNC	2.803825	0.866
Cascade Correlation - MLP	4.023580	0.931
CAST - DNC	3.867520	0.926
CAST - Modified DNC	4.196262	0.937
CAST - Cascade Correlation	27.646143	0.999
MOST - DNC	1.634515	0.694
MOST - Modified DNC	1.505652	0.659
MOST - Cascade Correlation	8.253938	0.985
MOST - CAST	3.648490	0.917

Table F.9. 5×2 cv results for `optDigits` dataset

Methods	Number of hidden	Number of epochs trained	Test Error
DNC	13.7 ± 0.5	2000 ± 0	0.0756 ± 0.0059
MLP	13.7 ± 0.5	2000 ± 0	0.0404 ± 0.0041
modified DNC	10.5 ± 1.6	1153.4 ± 165.5	0.0432 ± 0.0080
MLP	10.5 ± 1.6	100 ± 0	0.0466 ± 0.0113
Cascade Correlation	6.6 ± 0.8	1117.3 ± 126.8	0.0418 ± 0.0033
MLP	6.6 ± 0.8	1117.3 ± 126.8	0.0645 ± 0.0236
CAST	18 ± 0	9000 ± 0	0.0318 ± 0.0045
MOST	21 ± 0	8000 ± 0	0.0308 ± 0.0033

Table F.10. 5×2 cv F test results for `optDigits` dataset

Methods	F Value	Confidence
DNC - Modified DNC	9.29015	0.988
DNC - MLP	25.084334	0.998
Modified DNC - MLP	0.755926	0.329
Cascade Correlation - DNC	39.143834	0.999
Cascade Correlation - Modified DNC	0.845336	0.383
Cascade Correlation - MLP	0.73475	0.316
CAST - DNC	104.755263	0.999
CAST - Modified DNC	2.638732	0.852
CAST - Cascade Correlation	9.462282	0.989
MOST - DNC	60.827011	0.999
MOST - Modified DNC	5.853936	0.968
MOST - Cascade Correlation	12.164679	0.994
MOST - CAST	0.628129	0.248

Table F.11. 5×2 cv results for `penDigits` dataset

Methods	Number of hidden	Number of epochs trained	Test Error
DNC	11.5 ± 0.5	974.8 ± 32.6	0.0365 ± 0.0035
MLP	11.5 ± 0.5	974.8 ± 32.6	0.0199 ± 0.0029
modified DNC	9.3 ± 1.6	517.1 ± 78.6	0.0222 ± 0.0059
MLP	9.3 ± 1.6	50 ± 0	0.0230 ± 0.0072
Cascade Correlation	20 ± 0	424.2 ± 20.5	0.0141 ± 0.0015
MLP	20 ± 0	424.2 ± 20.5	0.0156 ± 0.0037
CAST	5 ± 0	2500 ± 0	0.0197 ± 0.0197
MOST	$\frac{19}{20} \pm 0$	14500 ± 0	0.0139 ± 0.0019

Table F.12. 5×2 cv F test results for `penDigits` dataset

Methods	F Value	Confidence
DNC - Modified DNC	10.714724	0.991
DNC - MLP	311.011186	0.999
Modified DNC - MLP	1.011462	0.471
Cascade Correlation - DNC	66.372105	0.999
Cascade Correlation - Modified DNC	5.748113	0.966
Cascade Correlation - MLP	1.040030	0.485
CAST - DNC	3.271175	0.898
CAST - Modified DNC	5.357193	0.961
CAST - Cascade Correlation	7.662046	0.982
MOST - DNC	37.287593	0.999
MOST - Modified DNC	9.354718	0.988
MOST - Cascade Correlation	0.787524	0.348
MOST - CAST	8.421462	0.985

REFERENCES

1. Bishop, C. M., *Neural Networks for Pattern Recognition*, Oxford University Press, 1995.

2. Mitchell, T. M., *Machine Learning*, McGraw-Hill International Editions, 1997.

3. Geman, S., E. Bienenstock and R. Doursat, "Neural networks and the bias/variance dilemma", *Neural Computation*, Vol. 4, pp. 1–58, 1992.

4. Kwok, T. and D. Yeung, "Constructive algorithms for structure learning in feedforward neural networks for regression problems", Technical Report HKUST-CS95-43, Department of Computer Science, Hong Kong University of Science and Technology, 1999.

5. Campbell, C., "Constructive learning techniques for designing neural network systems", `http://citeseer.nj.nec.com/campbell97constructive.html`, 1997.

6. Reed, R., "Pruning algorithms – a survey", *IEEE Transactions on Neural Networks*, Vol. 4, pp. 740–747, 1993.

7. Fahlman, S. E. and C. Leibiere, "The cascade-correlation learning architecture", *In Advances In Neural Information Processing Systems*, Vol. 2, pp. 524–532, 1990.

8. Ash, T., "Dynamic node creation in backpropogation networks", *Connection Science*, Vol. 1, No. 4, pp. 365–375, 1989.

9. Hwang, J., S. You, S. Lay and I. Jou, "What's wrong with the cascade-correlation learning: A projection pursuit learning perspective", *IEEE Transactions on Neural Networks*, Vol. 7, No. 2, pp. 278–289, 1996.

10. Friedman, J. H. and W. Stuetzle, "Projection pursuit regression", *Journal of the American Statistics Association*, Vol. 76, No. 376, pp. 817–823, 1981.

11. Platt, J., “A resource allocating network for function interpolation”, *Neural Computation*, Vol. 3, pp. 213–225, 1990.

12. Tenorio, M. F. and W. T. Lee, “Self-organizing network for optimum supervised learning”, *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, pp. 100–110, 1990.

13. Frean, M. R., “The upstart algorithm: a method for constructing and training feedforward neural networks”, *IEEE Transactions on Neural Networks*, Vol. 2, pp. 198–209, 1990.

14. Young, S. and T. Downs, “CARVE - A constructive algorithm for real valued examples”, *IEEE Transactions on Neural Networks*, Vol. 9, No. 6, pp. 1180–1190, 1998.

15. Setiono, R., “Feedforward neural network construction using cross validation”, *Neural Computation*, Vol. 13, pp. 2865–2877, 2001.

16. Alpaydın, E., “Combined 5 x 2 cv f test for comparing supervised classification learning algorithms”, *Neural Computation*, Vol. 11, pp. 1885–1892, 1999.

17. “Data, software and news from the statistics community”, `http://lib.stat.cmu.edu/`.

18. “Collections of data for developing, evaluating, and comparing learning methods”, `http://www.cs.toronto.edu/ delve/data/datasets.html`.

19. Kaynak, C., *Methods of combining multiple classifiers and their application to handwritten digit recognition*, Master’s Thesis, Boğaziçi University, Turkey, 1995.

20. Blake, C. L. and C. J. Merz, “UCI repository of machine learning databases”, `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

21. Alimoğlu, F., *Combining multiple classifiers for pen-based handwritten digit recog-*

nition, Master's Thesis, Boğaziçi University, Turkey, 1996.

22. Newman, D., "The distribution of the range in samples from a normal population, expressed in terms of an independent estimate of standard deviation", *Biometrika*, Vol. 31, pp. 20–30, 1939.